Lecture Notes:
Introduction to Communication Engineering

Changho Suh[1]

June 12, 2020

[1]Changho Suh is an Associate Professor in the School of Electrical Engineering at Korea Advanced Institute of Science and Technology, South Korea (Email: chsuh@kaist.ac.kr).

Changho Suh (chsuh@kaist.ac.kr)

Lecture 1: Logistics and overview

About instructor

Welcome to EE321: Communication Engineering! My name is Changho Suh, an instructor of the course. A brief introduction of myself. A long time ago, I was one of the students in KAIST like you. I spent six years at KAIST to obtain the Bachelor and Master degrees all from Electrical Engineering in 2000 and 2002, respectively. And then I left academia, joining Samsung Electronics. At Samsung, I worked on the design of wireless communication systems like 4G-LTE systems. Spending four and a half years, I then left industry, joining UC-Berkeley where I obtained the PhD degree in 2011. I then joined MIT as a postdoc, spending around one year. And then I came back to KAIST. While in Berkeley and MIT, I worked on a field, so called information theory, which lays the foundation of digital communication that I am going to cover in this course. This is one of the main reasons that I am teaching this course.

Today's lecture

In today's lecture, we will cover two very basic stuffs. The first is logistics of this course: details as to how the course is organized and will proceed. The second thing to cover is a brief overview to this course. Here I am going to tell you what we will study in this course in a very brief manner.

My contact information, office hours and TAs' information

See the syllabus uploaded on the course website (to be detailed soon). One special note: The office hours are *offline*, but if you wish to meet online, please send me an email to request. Upon request, I can open Zoom occasionally. If you cannot make it neither for my office hours nor for TAs' ones, please send me an email to make an appointment in different time slots.

Course format

The class runs online via Zoom. We may have offline lectures on a demand basis. The Zoom meeting ID is 910 581 3558, and this will be fixed throughout the class. You can join via https://kaist.zoom.us/j/9105813558. It will be open 10 minutes prior to the scheduled time.

Prerequisite

The basic prerequisite for this course is familiarity with the concept on two crucial and fundamental topics: (i) *signals & systems*; (ii) *probability*. In terms of courses, this means that you should have taken: (i) EE202: Signals and Systems; (ii) EE210: Probability and Introductory Random Processes, which is an undergraduate-level course on probability. These are the ones that are offered in Electrical Engineering. Some of you from other departments might take different yet equivalent courses. This is also okay.

There is another course which helps understanding this course. That is EE528, which is a graduate-level course on random processes. It deals with the probability in a deeper manner and also covers many important concepts on random processes. So if you have enough energy, passion and time, I strongly recommend you to take this course simultaneously, although it is

not a prerequisite.

There must be a reason as to why the above courses are crucial. The reason is obviously related to the definition of communication that this course is about. What is communication? Communication is the transfer of information from one end (called *transmitter*) to the other (called *receiver*), over a physical medium (like an air) between the two ends. Here the information could be voice signals, or video signals, or text signals. So we see that communication deals with *signals*. Also here we have something between the transmitter and the receiver. One can view that as a black box, which can then be interpreted as a *system*. So communication deals with *systems*. This is why the basic properties on signals and systems play a role to understand the topics on communication. I believe all of you guys already took EE202, or are now taking this. If you don't take this course, I strongly recommend you to take this course. It's okay to take the course at the same time with the communication engineering course. In the beginning part of this course, we are not going to cover the topics concerning the knowledge of signals and systems. So don't worry.

Now let us explain why the probability course matters. Recall that there is a physical medium that lies in between the transmitter and the receiver. The physical medium is so called the *channel*. The channel is the very entity that relates the concept of probability to communication. If you think about how the channel behaves, then you can easily see why. As mentioned earlier, the channel is a *system* (in other words, a function) which takes a transmitted signal as an input and a received signal as an output. Here one big problem arises in the system. The problem is that it is not a *deterministic* function. If the channel is deterministic and one-to-one mapping, then one can easily reconstruct the input from the output. So there is no problem in transferring information from the transmitter to the receiver. However, the channel is not deterministic in reality. It is actually a *random* function. In other words, there is a random entity (also known as *noise* in the field) added into the system. In typical communication systems, the noise is additive: a received signal is the sum of a transmitted signal and the noise. In general, we have no idea of the noise. In mathematics or statistics, there is a terminology which indicates such a random quantity. That is, *random variable* or *random process*, which the probability forms the basis for. This is the very reason that this course requires a deep understanding on probability. Some of you may have no idea of the random processess although you took EE210. Please don't be offended. It is nothing but a sequence of random variables. Whenever the concept on random process comes up, I will provide detailed explanations and/or materials which serve you to understand the contents required.

Course website

We have a course website on the KLMS system. You can simply login with your portal ID. If you want to only sit in this course (or cannot register the course for some reason), please let me or one of TAs know your email address. Upon request, we are willing to distribute course materials to the email address that you sent us.

Text

There is no required textbook. But don't worry. Instead I will provide you with enough materials so that you can readily follow the course. The materials are three folded. First, course notes (CN), which contain every details and thus are self-contained. I believe reading the CNs would suffice for you to understand all the contents covered in this course. Second, lecture slides (LS), which I will use during lectures. The last is lecture videos (LV). The Zoom meeting will be recorded and an edited version of the recording will be uploaded on the course website.

As mentioned earlier, this course requires the familiarity with the concept on probability. So for

those who are not familiar with the probability concept, I strongly recommend you to practice the following material. This is actually a course note drafted by two MIT professors, Bertsekas and Tsitsiklis. The names are very hard to pronounce. So I will simply call them BT. This has already been uploaded on the website. The course note is very easy to read. So I believe you can study by yourselves.

Problem sets

There will be weekly or bi-weekly problem sets. So there would be seven to eight problem sets in total. Your solution (a scanned version of your handwritten writeup or just a type-based soft version) must be submitted via *email* to a responsible TA, who will be designated upon the issue of each problem set. Solutions will be usually available at the end of the due date. This means that in principle, we do not accept any late submission. We encourage you to cooperate with each other in solving the problem sets. However, you should write down your own. You are welcome to flag confusing topics in the problem sets; this will not lower your grade.

Exams

As usual, there will be two exams: midterm and final. Please see the syllabus for the schedule. Notice that exams are *in-class* and *offline* at N1-201. In case the COVID-19 crisis is not improved until the midterm season, then I will think about a way to take the exam in an online manner and will announce details around the time. Otherwise, you will take it offline. Please let us know if someone cannot make it for the schedule. With convincing reasons, we can change the schedule or can give a chance for you to take an exam in a different time slot that we will organize individually.

Three things to notice. The first is about whether to hold regular lectures during the exam weeks assigned by KAIST default. I will hold regular lectures even during the exam weeks. Why? This way, we can finish the course in the middle of June, which was the original semester end season without the outbreak. If you have concerns and/or issues, please let me know. We can discuss further to find a better solution. Second, for both exams, you are allowed to use one cheating sheet, A4-sized and double-sided. So it is a kind of semi-closed-book exam. Lastly, for your convenience, we may provide an instruction note for each exam (if you wish), which contains detailed guidelines as to how to prepare for the exam. Such information includes: (1) how many problems are in the exam; (2) what types of problems are dealt with in what contexts; (3) the best way to prepare for such problems.

Course grade

Here is a rule for the course grade that you are mostly interested in perhaps. The grade will be decided based on four factors: problem sets (22%); midterm (32%); final (40%); and interaction (6%). Here the interaction means any type of interaction. So it includes attendance, in-class participation, questions, and discussion.

Overview

Now let's move onto the second part. Here is information for reading materials: Course Note (CN) 1. In this part, I will tell you the main content that we will study throughout the course. To this end, I will first tell you how communication of this course's interest evolves and then point out what specific component in communication that we will focus on learning about.

History of communication systems

Let's start from the beginning. What is the definition of communication? As mentioned earlier, communication is the transfer of information from one end to the other end. Here the one end is called transmitter, and the other end is called receiver. Something that lies in between is a physical medium, called the channel.

As you can easily image, communication has a long history. Even in the beginning of the world, there was a communication, which is a dialogue between people. The dialogue, also called conversation, is definitely a type of communication although it is a very naive way of communication and has nothing to with electrical engineering. Actually there was a breakthrough in the history of communication, which is now related to the communication systems that have something to do with electrical engineering and so we are interested in. This breakthrough was the invention of telegraph. *Morse* code[1] is the first such example: a very simple transmission scheme that is initially used in telegraph. Actually this invention was based on a simple observation (discovered in physics) that electrical signals, like voltage or current signals can be transmitted over wires such as copper lines. So it is the first communication system that have something to do with electrical signals. Actually this is the main reason as to why communication systems have been studied within the field of electrical engineering where most of you guys are in. Later this technology was further developed. In the 1870s, Alexander Graham Bell invented a more advanced version of such systems, called *telephone*.

Communication systems were further upgraded. The upgrade was based on another finding in physics: not only we can send electrical signals over wires, but we can also send them in a *wireless* environment through electromagnetic waves, simply called radio waves. This finding inspired an Italian engineer at that time, named Guglielmo Marconi. He developed a wireless version of telegraph, called wireless telegraphy. Later this technology was further developed, which led to the invention of radio and TV.

State of the affairs in early 20th century & Claude E. Shannon

These are the communication systems that were developed in the early 20th century: telegraph, telephone, wireless telegraphy, radio and TV. Actually at this time, there was one guy who could make some interesting observations on these communication systems, which in turn made him do some great work in communication. The guy was Claude E. Shannon, known as the Father of Information Theory. He made the following observation: Engineering designs are pretty ad-hoc, being tailored for each specific application. This means that design principles were completely different depending on signals of interest.

What he felt from this observation is that such communication systems are really annoying. He did not like the communication systems because such systems are designed in a pretty ad-hoc manner (no systematic design principle!), in other words, in a completely different manner depending on each specific application of interest. He actually believed that there must be one simple & beautiful framework that can unify all such different communication systems. So he tried to unify the ad-hoc approach. As a specific effort, he raised the following big questions.

Shannon's big questions

The first question is the most fundamental question regarding the *possibility* of unification.

Question 1: Is there a general unified methodology for designing communication systems?

The second question is a natural follow-up question which helps addressing the first question.

[1] Regarding the code, don't be confused with computer programming languages (such as C++ and Python) that have nothing to do with it. *Code* is a sort of terminology used in the context of communication which indicates a transmission scheme.

He thought that if unification is possible, then there may exist a *common currency* (like dollar in the context of economics) w.r.t. (with respect to) information. In the communication systems, we have a variety of different information sources, like text, voice, video, images. The second question that he asked is related to the existence of such common currency.

Question 2: Is there a common currency of information that can represent all such different information sources?

He had spent around eight years to address the above questions. Perhaps he had quite a difficult time as the long period of eight years indicates. But he could make a progress in the end. He could address all the big questions in a single stroke. In the process of doing this, he could develop a unified communication architecture.

Communication architecture

Prior to describing the unified architecture in detail, let us first recall the three terminologies that we introduced: transmitter, receiver, and something that lies in between transmitter and receiver, which is the channel. Now consider information signals that one wishes to transmit, such as text, voice, and image pixels. Here is another terminology which indicates such signals: "information source". What Shannon thought in the first place is that there must be something which processes the information source before transmission. He abstracted this process with a black box that he called "encoder". Of course there must be something at receiver which attempts to recover the information source from the received signals that the channel yields. He abstracted the process at the receiver end with another black box that he called "decoder". This is the very first basic block diagram that Shannon imagined for a communication architecture. See Fig. 1. From a Shannon's viewpoint, a communication system is nothing but a collection of encoder and decoder, so designing a communication system is concerning how to develop such encoder and decoder.

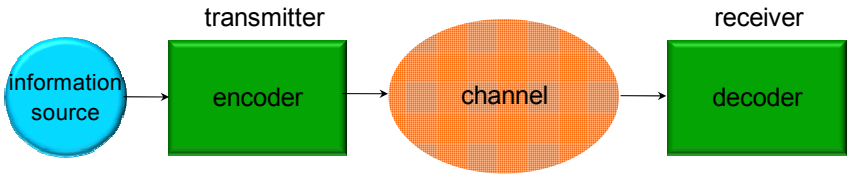


Figure 1: A basic communication architecture.

Representation of information source

With this simple picture in his mind, Shannon wished to unify all the different communication systems that were prevalent in the early 20th century. Most engineers at that time attempted to transmit information sources (which are of course different depending on applications of interest) *directly* without any significant modification. Shannon thought that this is the very reason as to why there were different communication systems.

He then thought that for the purpose of unification, there must be at least something that can represent different information sources. He believed that the common thing would lead to a universal framework. It turns out Shannon's work on master thesis in MIT could play a significant role to find a universal way of representing information source. His master thesis was about Boolean algebra. What he did specifically is that any logical relationship in circuit systems can be represented with 0/1 logic, in other words, binary string.

Inspired by this, Shannon thought that the same thing may happen in the context of communication systems, meaning that any type of information source can be represented with binary string. He showed that it is indeed the case. Specifically what he showed is that the binary string that he called "bits" can represent the meaning of information.

For instance, suppose that information source is an English text that comprises English letters. Now how to represent each English letter with a binary string? Here one key observation is that there are only a *finite* number of candidates that each letter can take on. This number is the total number of English alphabets, which is 26.[2] From this, we see that $\lceil \log_2 26 \rceil = 5$ number of bits suffices to represent each letter.

A two-stage architecture

This observation led Shannon to introduce bits as a common currency of information. Specifically here is what he did. He first thought that we need a block which converts information source into bits of our interest. Motivated by this, Shannon came up with a two-stage architecture in which the encoder is split into two parts. Here the role of the first block is to convert information source into bits. Shannon called the block "source encoder", as it is a part of the encoder as well as is related to how information *source* looks like. The role of the second block is to convert bits into a signal that can actually be transmitted over a channel. Shannon called it "channel encoder" because it is obviously concerning a channel. See the upper part in Fig. 2.

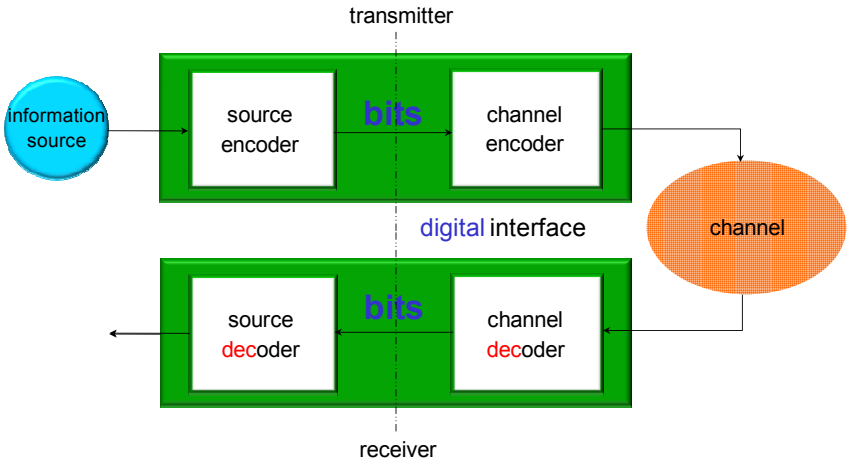


Figure 2: A two-stage communication architecture.

Similarly, receiver consists of two stages. But the way it is structured is opposite. In other words, we first convert the received signals into the bits that we sent at the transmitter (channel decoder). Next, we reconstruct the information source from the bits (source decoder). As you can see, this block is nothing but an inverse function of source encoder. Obviously source encoder should be one-to-one mapping; otherwise, there is no way to reconstruct the information source. See the lower part in Fig. 2.

Actually there is a terminology which indicates the part spanning from channel encoder, channel, to channel decoder. We call it "digital interface". Here one thing to notice is that this digital interface is universal in a sense that it has nothing to do with type of information source because the input to the digital interference is always bits. So in that regard, it is indeed a unified

[2]Here we ignore any special characters such as space.

communication architecture.

Focus of this course

This course is about how to design this *digital interface*. In the next lecture, I will tell you what specific contents within the digital interface that we will study throughout the course. I will then embark on details.

March 20, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 2: A statistical model for additive noise channels

Recap

Last time we introduced the key entity in communication that we are going to focus on learning about throughout the course. Recall the definition of communication. Communication is the transfer of information from one end (called transmitter) to the other end (called receiver) through a physical median (called channel). The entity of this course's focus is: *digital communication* where a transmitted information source is of the form of binary string, bits; see Fig. 2. Here the block that processes the bits to yield a transmitted signal is called the channel encoder, and the one that processes a received signal to reconstruct the bits is called the channel decoder. For simplicity, let us call a collection of them encoder/decoder. The focus of the course is to study how to design the encoder/decoder so as to enable reliable transfer of such binary string.

Figure 1: The basic block diagram of a digital communication system.

Today's lecture

Obviously the design of the encoder/decoder depends on properties of the associated channel. So we need to understand the properties of the channel, which are possibly different depending on a channel type. In today's lecture, we are going to talk about some representative channels that we will focus on throughout this course. I will then provide you with the detailed course outline, being built upon the focused channels. Next, we will consider a simple example of the channels where we can get concrete feelings as to how to design the encoder/decoder.

Modem

Prior to diving into details, let me say a few words about one terminology that you may hear of in the context of communication. The physical world is definitely analogue, so a transmitted signal that is fed into the channel should be a physical quantity, such as an electromagnetic signal, say an electrical voltage signal. So the encoder needs to *modulate* the digital information (bits) into an electrical voltage signal. Also a received signal (the channel output) is a voltage signal, so the decoder needs to *demodulate* the analogue signal to reconstruct the bits. Hence, people often call the encoder/decoder simply the *modem*, highlighting "mo" and "dem" from modulator and demodulator, respectively.

Statistical models of the channel

One key feature of the channel is its uncertainty. The received voltage signal is not a deterministic function of the transmitted signal. While the behavior of the channel over one experiment cannot

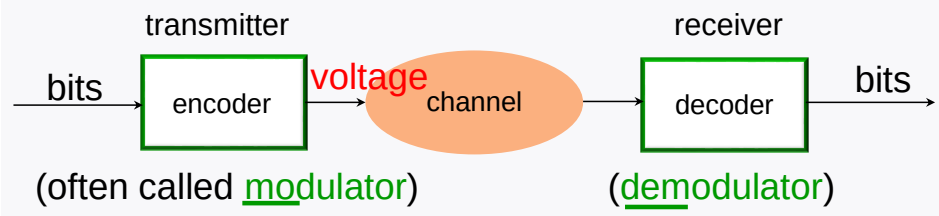


Figure 2: MODEM: MOdulator & DEModulator

be predicted, the average behavior, as seen over many experiments turns out to be well behaved in many physically interesting scenarios.

Also the channel in most practical scenarios is of the form of addition: the received signal is the sum of the transmitted signal and an additive noise, as illustrated in Fig. **??**.

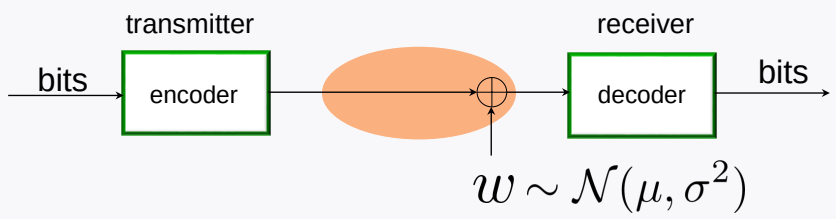


Figure 3: A statistical model for additive noise channels.

Hence, the additive noise, say w, can be described by a random quantity, that is usually referred as a *random variable* in the language of statistics. More precisely, it is a *continuous random variable*, which is fully described by a probability density function (pdf). If you do not understand what I am talking about, please study the concept of a random variable from BT.

Obviously the statistical behaviour of w, i.e., the pdf of w, is crucial in the modem design. Hence, in order to study how to design the modem (of this course's focus), we first need to have reasonable statical models for the channel.

Course outline

In this course, we will first study a couple of reasonable statistical models and then will study how to design the modem accordingly. The course consists of three parts.

In Part I, we will consider the simplest yet practically-relevant setting where the pdf of w can precisely be modeled as a very well-known distribution, called the Gaussian distribution, which you may hear of. The Gaussian distribution can be fully described by two representative quantities: the mean, say μ, and the variance, say σ^2. It is simply denoted by:

$$w \sim \mathcal{N}(\mu, \sigma^2)$$

where the symbol "$\sim$" means "is distributed according to", and the beautiful caligraph letter $\mathcal{N}$ stands for the Normal distribution.[1] The corresponding pdf reads:

$$f_w(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

[1]The rationale behind the naming "normal" is that such distribution appears frequently (normally) in many contexts. This is coined by the famous German mathematician, Carl Friedrich *Gauss*. Later people call it the Gaussian distribution, appreciating Gauss.

To know more about the Gaussian distribution, you may want to refer to Chapter 3 in BT. In Part I, we will first study why the additive noise can be modeled as the Gaussian distribution. We will then study how to design the modem for the additive channel with the Gaussian noise, simply called the additive Gaussian channel.

In Part II, we will move onto another practical scenario where the channel is established via wires (copper lines). Such channel is simply called the wireline channel. You may hear of DSL (digital subscriber line), the name of one popular internet (wireline) service that many homes have subscribed to. The DSL is a perfect example of the wireline channel. It turns out the wireline channel can be modeled as the concatenation of two blocks; see Fig. **??**. The second

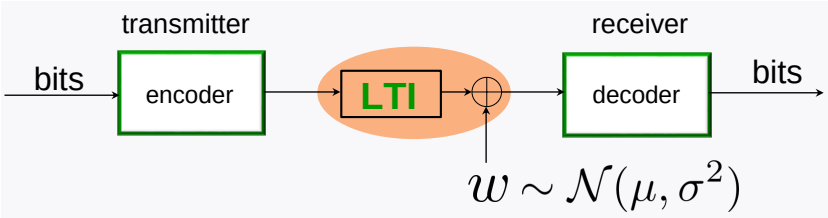


Figure 4: A statistic model for wireline channels.

block is the same as before: the additive block with the Gaussian noise. The first block is a system that you must be very much familiar with, assuming that you took EE202. That is, the LTI (linear time invariant) system. In Part II, we will first study the rationale behind such modeling, and then investigate how to design the modem accordingly.

Now you may guess that the next topic is about *wireless* channel. A prominent example of the wireless channel is the 5G cellular system that is prevalent nowadays in our daily life. It turns out the wireless channel can be modeled as the concatenation of the LTV system block and the additive block. Here the acronym LTV stands for linear time *varying*. In the wireless channel, either transmitter or receiver, or both can move freely because there is no wire between them. Such movement then yields a different channel statistics, making the system time-varying. Unfortunately (or maybe fortunately to those who are not interested in wireless communication), this is beyond the scope of this course. This is mainly due to the interest of time; indeed there are many things to learn for the wireless channel. But if you desire to learn about, you may want to take a course, EE421: Wireless Communication Systems.

Instead in Part III, we are going to switch-gear to explore something different. It turns out that the principles and tools that we will learn about through the prior parts are quite instrumental to addressing some important issues that arise in a trending field, machine learning. So in Part III, we will study such machine learning applications. While there are many applications, we will focus only on the following two:

1. Machine learning classifiers;
2. Speech recognition.

Part I: Additive Gaussian channel

In Part I, we will study why the Gaussian channel is practically relevant, and then learn about the modem design under the Gaussian channel.

A simple example

Let x be the transmitted signal and y be the received signal.

$$y = x + w \tag{1}$$

where w is an additive noise. In order to get concrete feelings about the modem design, prior to the Gaussian case, let us first consider a much simpler setting where w is assumed to be strictly within a certain range, say $\pm\sigma$.

A simple transmission scheme

Suppose we want to send a single bit across the above simple channel. We can do this by transmitting a voltage v_0 to transmit an information content of the bit being "zero", and a voltage v_1 when transmitting an information content of the bit being "one". As long as

$$|v_0 - v_1| > 2\sigma \tag{2}$$

we can be certain that our communication of the one bit of information is reliable over this channel. Physically, the voltage transmitted corresponds to some energy being spent. For simplicity, assume that the energy spent in transmitting a voltage v Volts is v^2 Joules. In this context, a natural question arises: How many bits can we reliably communicate with an energy constraint of E Joules?

Suppose we intend to transmit the number k of bits. A bit of thought lets us come up with the following transmission scheme. We choose to transmit one of a collection of discrete voltage levels:

$$\left\{-\sqrt{E}, -\sqrt{E} + 2\sigma, \cdots, -\sqrt{E} + 2i\sigma, \cdots, +\sqrt{E}\right\}, \tag{3}$$

where i denotes some integer. Here how many intervals do we have in order to ensure the successful transmission of k bits? That is, $2^k - 1$. Why? This is because we should have at least 2^k possible voltage-level candidates to guarantee the number k of bits. So we should have:

$$(2^k - 1) \cdot (2\sigma) \leq 2\sqrt{E}$$

For simplicity, let us assume that $\sqrt{E}$ is divisible by σ. This then leads to:

$$k = \log_2\left(1 + \frac{\sqrt{E}}{\sigma}\right). \tag{4}$$

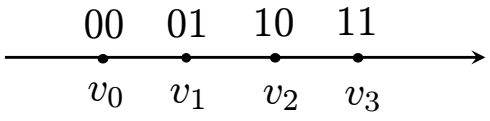


Figure 5: Mapping from bits to voltage levels

The illustration in Fig. **??** demonstrates one possible mapping between the 4 sequences of 2 bits to the 4 discrete voltage levels being transmitted. Here $\sqrt{E} = 3\sigma$ and

$$v_i = -\sqrt{E} + 2i\sigma, \quad i = 0, 1, 2, 3. \tag{5}$$

Relation between energy and reliable information transmitted

For a given energy constraint E, the number of bits we can communicate reliably is, from (5),

$$\log_2\left(1+\frac{\sqrt{E}}{\sigma}\right). \tag{6}$$

A natural sort of question that one can ask is then: If we want to send an additional bit reliably how much more energy do we need to expend? We can use the above expression to answer this question: the new energy $\tilde{E}$ required to send an extra bit of information reliably has to satisfy:

$$\log_2\left(1+\frac{\sqrt{\tilde{E}}}{\sigma}\right) = 1+\log_2\left(1+\frac{\sqrt{E}}{\sigma}\right) \tag{7}$$

$$1+\frac{\sqrt{\tilde{E}}}{\sigma} = 2\left(1+\frac{\sqrt{E}}{\sigma}\right) \tag{8}$$

$$\tilde{E} = \sigma + 2\sqrt{E}. \tag{9}$$

In other words, we need more than quadruple the energy constraint to send just one extra bit of information reliably.

Another interesting thing to note is that the amount of reliable communication transmitted depends on the *ratio* between the transmit energy budget E and σ^2 related to the energy of the noise. This ratio, $\frac{E}{\sigma^2}$, is called the *signal to noise ratio* and will feature prominently in the other additive noise models we will see.

Look ahead

This simple example of a channel model gave us a feel for simple transmission and reception strategies. It also gave us an idea of how a physical resource such as energy is related to the amount of information we can communicate reliably. The deterministic channel model we have used here is rather simplistic; in particular, the choice of σ might have to be overly conservative if we have ensure that the additive noise has to lie in the range $\pm\sigma$ with full certainty. It turns out this issue brings up the motivation of knowing a more precise *statistical* behavior of w. Next time, we will discuss it in detail.

March 25, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 3: Additive Gaussian noise model

Recap

Last time, I outlined the course contents that we will study throughout the course, and mentioned that the contents are built upon two representative & practically-relevant channels: (i) the additive Gaussian channel; (ii) the wireline channel (which can be modelled as concatenation of an LTI system and the additive Gaussian noise block).

In an effort to get a concrete feel as to how to design the modem of this course's main interest, prior to the Gaussian channel, we considered a simple channel example where the additive noise w has the upper and lower bounds: $-\sigma \leq w \leq \sigma$. Under this channel, we then investigated a simple transmission scheme in which a voltage-level signal is decided based on the patten of multiple bits, say k bits, such that each voltage level is set apart at least 2σ to ensure the perfect recovery of the bits at the receiver. We call this scheme *pulse amplitude modulation*, PAM for short. The reception strategy was the Nearest Neighbour (NN) rule: Declaring a voltage level that is nearest to a received signal, as the transmitted voltage signal. We then found the tradeoff relationship between the energy budget of E and the maximum number k of bits that can be transmitted reliably:

$$k = \log_2 \left(1 + \frac{\sqrt{E}}{\sigma}\right).$$

At the end of the last lecture, I mentioned that the assumption made on w in the simple channel example is not realistic. I then claimed that in reality, w respects a very well-known distribution: the Gaussian (or Normal) distribution.

Today's lecture

Today we will prove this claim. Specifically what we are going to do are three folded. First we will invoke physics: investigating how the additive noise occurs in practical communication systems. We will then rely on such physics to come up with a set of mathematical assumptions that the additive noise well respects. Finally, we will exploit the mathematical assumptions to prove that w is indeed distributed according to the *Gaussian* distribution.

Physics

As we outlined in the previous section, let us first invoke physics, by investigating how such noise occurs in interested communication systems. Here what I mean by interested communication systems is the one with electronic circuits, which I simply call electronic communication systems. In such electronic communication systems, a signal level is determined by its voltage. And we know from the basic introductory course on circuits which I believe you already took: the voltage level is dictated by the movement of electrons. The less electrons, the higher a voltage level.

In the early 1900s, it was discovered by a physicist, named John B. Johnson, that the behavior of electrons contributes to inducing a noise that we cannot control over. The discovery was based on an interesting observation that electrons are *randomly agitated* by heat. This random agitation then makes the voltage level be out of control, as it may depend on a non-controllable factor

which is the temperature. So the precise movement control of electrons is almost impossible in reality. Jonhson interpreted such undesirable random fluctuation as a major source of *noise*.

At that time, Johnson wanted to figure out the statistical behavior of the noise, but he could not do so because he was an experimental physicist, not being good at math. Instead he had a colleague in his workplace, Bell Labs, who is very smart and particularly strong at math. The colleague was Harry Nyquist[1]. So he shared his experimental observation with Nyquist. As Johnson expected, Nyquist could establish a mathematical theory for such noise to demystify its statistical behavior. The theory formed the basis of the Gaussian noise modelling that we will study in the sequel. At the time, the noise was named as the thermal noise, as it is dependent of the temperature.

In reality, a noise may also depend on device imperfections and/or measurement inaccuracies. But the major source of the randomness is due to the thermal noise. So we are going to focus on the mathematical theory of the thermal noise done by Nyquist.

Assumptions made on the thermal noise

The mathematical theory starts with making concrete assumptions on physical properties that the thermal noise respects. These are four folded.

1. As mentioned earlier, the noise is due to the random movement of electrons. And an electrical signal contains tons of electrons. So one natural assumption that one can make is: the additive noise is the overall consequence of many additive "sub-noises'. A natural follow-up assumption is that the number of sub-noises is infinity, which reflects the fact that there are tons of electrons in an electrical signal.

2. These electrons are known to be typically uncorrelated with each other. So the second assumption is: the sub-noises are statistically independent.

3. Also it is known that there is no particularly dominating electron that affects the additive noise most significantly. So the third assumption that one can make as a simplified version of this finding is: each sub-noise contributes exactly the same amount of energy to the total energy in the additive noise.

4. Finally, the noise energy is typically not so big relative to the energy of an interested voltage signal. Obviously it does not blow up. So the last reasonable assumption is: the noise energy is finite.

Mathematical representation of the four assumptions

Now let us express the above four assumptions in a mathematical language. To this end, we first introduce some mathematical notations to write the total additive noise w as the the sum of m sub-noises $n_1, \ldots, n_m$:

$$w = n_1 + n_2 + \cdots + n_m. \tag{1}$$

The mathematical expressions of the first and second assumptions are: $m \to \infty$ and $(n_1, \ldots, n_m)$ are statistically independent with each other. Here what it means by "independence" is that the joint pdf of $(n_1, \ldots, n_m)$ can be expressed as the product of individual pdfs:

$$f_{n_1,n_2,\cdots,n_m}(a_1, a_2, \ldots, a_m) = f_{n_1}(a_1) f_{n_2}(a_2) \cdots f_{n_m}(a_m) \tag{2}$$

[1]Yes, he is the famous Nyquist, being very prominent for the Nyquist sampling theorem that you may hear of from the course on Signals and Systems.

where $a_i \in \mathbb{R}$ denotes a realization of the sub-noise n_i. If you are not familiar with the definition of the statistical independence, please see Chapter 3 in BT. A mathematical assumption that we will take while leading to the third assumption is: $(n_1, \ldots, n_m)$ are identically distributed. Here the property of being independent and identically distributed is simply called i.i.d.

Now without loss of generality[2], one can assume that w has zero-mean. Why? Suppose we have a bias on w: $\mathbb{E}[w] = \mu \neq 0$. We can then always subtract the bias from the received signal so that the mean of the *effective noise* is zero. More precisely, we subtract the bias μ from the received signal $y = x + w$ to obtain:

$$y - \mu = x + (w - \mu). \tag{3}$$

Then, $w - \mu$ can be interpreted as the *effective noise*, and this has indeed zero-mean. Due to the i.i.d. assumption, each sub-noise has the same amount of energy:

$$\mathbb{E}[n_i^2] = \mathbb{E}[n_j^2], \quad \forall i, j = 1, \ldots, m. \tag{4}$$

The energy in the total additive noise w is

$$\begin{aligned} \mathbb{E}[w^2] &= \mathbb{E}\left[\left(\sum_{i=1}^{m} n_i\right)^2\right] \\ &\overset{(a)}{=} \sum_{i=1}^{m} \mathbb{E}[n_i^2] + \sum_{j \neq i} \mathbb{E}[n_j n_i] \\ &\overset{(b)}{=} m\mathbb{E}[n_1^2] + \sum_{j \neq i} \mathbb{E}[n_j]\mathbb{E}[n_i] \\ &\overset{(c)}{=} m\mathbb{E}[n_1^2] \end{aligned} \tag{5}$$

where (a) is due to the linearity of the expectation operator; (b) comes from the i.i.d. assumption; (c) is because of the zero-mean assumption of $\mathbb{E}[w] = m\mathbb{E}[n_1] = 0$ (leading to $\mathbb{E}[n_i] = 0$ as well). Notice that the i.i.d. assumption implies the (pairwise) independence between n_i and n_j (for $j \neq i$), thereby leading to $\mathbb{E}[n_j n_i] = \mathbb{E}[n_j]\mathbb{E}[n_i]$. If you are not familiar with this, again see Chapter 3 in BT.

Lastly consider the finite energy assumption. Denoting this finite energy by σ^2, we see from (5) that the energy of the sub-noises must shrink as more and more sub-noises are added:

$$\mathbb{E}[n_i^2] = \frac{\sigma^2}{m}, \quad i = 1, \ldots, m. \tag{6}$$

The statistical behavior of the thermal noise in the limit of m

We are interested in the statistical property of the total additive noise w. In the language of probability, we are asking for the probability distribution or pdf of the random variable w. It turns out that under the assumptions made on w as above, the probability distribution for w is Gaussian, formally stated below.

Theorem 0.1 *Let $w = n_1 + n_2 + \cdots + n_m$. Assume that $(n_1, \ldots, n_m)$ are i.i.d. such that $\mathbb{E}[n_i] = 0$, $\mathbb{E}[n_i^2] = \frac{\sigma^2}{m}$ and $\sigma^2 < \infty$ (finite). Then, as $m \to \infty$,*

$$w \sim \mathcal{N}(0, \sigma^2). \tag{7}$$

[2]It means that the general case can be readily covered with some proper modification to the ground assumption that follows after the phrase. That's why here we say that there is no loss of generality.

Nyquist proved this theorem by relying on a very well-known theorem in probability and statistics, called the *central limit theorem* that you may hear of. If you want to know a precise mathematical statement of the central limit theorem, see Chapter 7 in BT.

Proof of Theorem (0.1)

The pdf of w depends obviously on the pdfs of the sub-noises. Here the relationship is very well-known when the sub-noises are *independent*, as we assumed as above. Actually you learned about the relationship from the course on probability (say EE210). That is,

> the pdf of the sum of independent random variables can be represented as the *convolution* of the pdfs of the random variables involved in the sum.

If you forget or don't know about this, please refer to Chapter 4 in BT. Also you will have a chance to prove this in PS1. So then how can we represent $f_w(a)$ in terms of $(f_{n_1}(a), \ldots, f_{n_m}(a))$? It is the convolution of all the pdfs of the sub-noises:

$$f_w(a) = f_{n_1}(a) * f_{n_2}(a) * \cdots * f_{n_m}(a) \tag{8}$$

where * indicates the convolution operator, defined as: $f_{n_1}(a) * f_{n_2}(a) := \int_{-\infty}^{+\infty} f_{n_1}(t) f_{n_2}(a-t) dt$.

But we face a challenge here. The challenge is that the convolution formula is very complicated, as the meaning of the word "convoluted" suggests. Even worse, we have an infinite number m of such convolutions (more precisely $m-1$). Another thing you learned now from the course on Signals and Systems (say EE202) comes to rescue. The convolutions can be very much simplified in the Fourier or Laplace transform domain. Since the pdf is a *continuous* signal, the interested transform must be Laplace transform. The simplification is based on the following key property: the Laplace transform of the convolution of multiple pdfs is the product of the Laplace transforms of all the pdfs involved in the convolution. To state what it means in detail, recall the definition of the Laplace transform $F_w(\cdot)$ w.r.t. the pdf $f_w(\cdot)$ of w:

$$F_w(s) := \int_{-\infty}^{+\infty} f_w(a) e^{-sa} da. \tag{9}$$

Now using the fact that the convolution is translated to the multiplication in the Laplace transform domain, we can re-write (8) as:

$$F_w(s) = [F_n(s)]^m \tag{10}$$

where $F_n(\cdot)$ indicates the Laplace transform of the pdf $f_n(\cdot)$ of the generic random variable n that represents a sub-noise n_i.

Now how to proceed with (10)? More precisely, how to compute $F_w(s)$ with what we know: $\mathbb{E}[n] = 0$ and $\mathbb{E}[n^2] = \frac{\sigma^2}{m}$? It turns out these moment information (0th and 1st moments) play a crucial role to compute. Here a key observation is that these moments appear as coefficients in the Taylor series expansion of $F_n(s)$. And it turns out this expansion leads to an explicit expression for $F_w(s)$, thereby hinting the pdf of w. To see this, let us try to obtain the Taylor series expansion of $F_n(s)$. To this end, we first need to compute the kth derivative of $F_n(s)$:

$$\begin{aligned} F_n^{(k)}(s) &:= \frac{d^k F_n(s)}{ds^k} \\ &= \int_{-\infty}^{\infty} (-1)^k a^k f_n(a) e^{-sa} da \end{aligned} \tag{11}$$

where the second equality is due to the definition of the Laplace transform: $F_n(s) := \int_{-\infty}^{+\infty} f_n(a)e^{-sa}da$. Applying the Taylor expansion at $s = 0$, we get:

$$F_n(s) = \sum_{k=0}^{\infty} \frac{F_n^{(k)}(0)}{k!} s^k \tag{12}$$

Plugging $s = 0$ into (11), we get:

$$F_n^{(k)}(0) = \int_{-\infty}^{\infty} (-1)^k a^k f_n(a)da = (-1)^k \mathbb{E}[n^k].$$

This together with (12) yields:

$$F_n(s) = \sum_{k=0}^{\infty} \frac{(-1)^k \mathbb{E}[n^k]}{k!} s^k. \tag{13}$$

Applying $\mathbb{E}[n] = 0$ and $\mathbb{E}[n^2] = \frac{\sigma^2}{m}$ to the above, we arrive at:

$$F_n(s) = 1 + \frac{\sigma^2}{2m} s^2 + \sum_{k=3}^{\infty} \frac{(-1)^k \mathbb{E}[n^k]}{k!} s^k. \tag{14}$$

Now what can we say about $\mathbb{E}[n^k]$ for $k \geq 3$ in the above? Notice that $\mathbb{E}[n^k] = \mathbb{E}[(n^2)^{\frac{k}{2}}]$ and $\mathbb{E}[n^2] = \frac{\sigma^2}{m}$ decays like $\frac{1}{m}$. It turns out this leads to the fact that $\mathbb{E}[(n^2)^{\frac{k}{2}}]$ decays like $\frac{1}{m^{k/2}}$, which exhibits a faster decaying rate for $k \geq 3$, as compared to $\frac{1}{m}$. It turns out this scaling yields:

$$\begin{aligned} F_w(s) &= \lim_{m\to\infty} \left(1 + \frac{\sigma^2}{2m} s^2 + \sum_{k=3}^{\infty} \frac{(-1)^k \mathbb{E}[n^k]}{k!} s^k\right)^m \\ &= \lim_{m\to\infty} \left(1 + \frac{\sigma^2}{2m} s^2\right)^m \\ &= e^{\frac{\sigma^2 s^2}{2}} \end{aligned} \tag{15}$$

where the last equality is due to the fact that

$$e^x = \lim_{m\to\infty} \left(1 + \frac{x}{m}\right)^m. \tag{16}$$

Now from (15), what can we say about $f_w(a)$? To figure this out, we need to do the Laplace *inverse* transform: $f_w(a) = \text{InverseLaplace}(F_w(s))(a) := \frac{1}{2\pi i} \lim_{T\to\infty} \int_{\mathsf{Re}(s)-iT}^{\mathsf{Re}(s)+iT} F_w(s)e^{sa}ds$. But you may feel headache because it looks very much complicated. So instead we will do the reverse engineering: guess-&-check. A good thing about the Laplace transform is that it is an one-to-one mapping. Also one can easily check (in PS1):

$$\text{LaplaceTransform}\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{a^2}{2\sigma^2}}\right) = e^{\frac{\sigma^2 s^2}{2}}.$$

This together with (15) gives:

$$f_w(a) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{a^2}{2\sigma^2}}, \quad a \in \mathbb{R}. \tag{17}$$

This is indeed the Gaussian distribution with mean 0 and variance σ^2. So this completes the proof.

Look ahead

Making some assumptions on the additive thermal noise inspired by physics, we mathematically demonstrated that the additive noise can precisely be modelled as the Gaussian distribution. Next time, we will go back to our main interest for the course: exploring the design of the modem but now under the Gaussian channel. We will then investigate the tradeoff relationship between the number of transmission bits and the energy budget, as we did in the prior simple channel example.

Changho Suh (chsuh@kaist.ac.kr)

Lecture 4: Optimal receiver

Recap

In Lecture 2, we considered the simple channel example where the additive noise is within $\pm\sigma$ voltages, and then came up with a simple transmission scheme and the corresponding reception strategy: PAM and the NN rule. Under such tx/rx schemes, we next established the tradeoff relationship between the energy budget and the number of bits transmitted reliably. Since the considered channel is not realistic, last time we started investigating practical communication systems and then demonstrated that the additive noise in real systems can be modelled as a random variable with the Gaussian distribution.

Today's lecture

Today we will go further now with the realistic additive channel, simply called the Gaussian channel. Specifically we are going to cover the following three stuffs. First we will study a simple transmission scheme as we did in Lecture 2. Essentially we will adopt exactly the same transmission scheme: PAM. Next we will explore a reception strategy. Recall the simple channel example in which the NN decision rule enables the *perfect transmission* as long as the minimum distance between any two voltage levels in transmission exceeds 2σ. Unlike this channel, in the Gaussian channel, we now need to worry about transmission *failure*, as the Gaussian noise level can exceed $+\sigma$ (or go below $-\sigma$) and this can incur a decision error under the NN rule (actually under any other reception scheme that one can imagine). One natural effort given the situation where transmission failure is unavoidable is to simply accept that we can have errors in decision and then to come up with the best reception strategy as much as we can. In a mathematical language, the best (or optimal) decision rule is defined the one that maximizes the success rate for transmission (the probability that we make a correct decision), equivalently minimizes the probability of error. So in the second part, we will derive such optimal decision rule. Lastly we will show that under a reasonable assumption (that I will detail later), the optimal decision rule is exactly the same as the intuitive NN decision rule that we studied earlier.

Since there is another concept (the success rate or simply called reliability) introduced in the Gaussian channel, we are now interested in characterizing the tradeoff relationship between three quantities: (i) the energy budget; (ii) the number of bits that can be reliably transmitted; (iii) reliability. This would be the topic of the next lecture. Today, we will focus only on the following two: energy and reliability. So we will consider the case of sending only one bit.

A simple transmission scheme

As mentioned earlier, consider the simple case of sending one bit: $b = 1$ or 0. Given the energy budget E, the transmission scheme that we will take is the same as before. That is PAM: transmitting $+\sqrt{E}$-voltage signal for $b = 1$ while sending $-\sqrt{E}$-voltage signal for $b = 0$, as illustrated in Fig. 1.

Reception scheme

As mentioned earlier, the optimal decision rule is the one that maximizes reliability. Prior to deriving the rule, let us first describe two main factors that affect the decision rule.

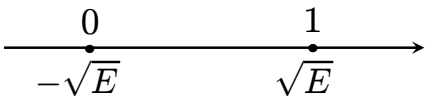


Figure 1: Mapping from sending one bit across the Gaussian channel.

- *A priori knowledge*: If there is some prior information on the statistical behaviour of the information bit b, then that could factor in the decision rule. Why? You can easily see why if you think about one extreme case where we somehow knew (prior to the communication process) that the information bit is 1 for sure. In this case, we do not need to take a look at the received voltage y. We can simply declare that the information bit is 1, no matter what. In reality, however, we often times have no access to such prior information. In this case, what we can do is to simply assume that the information bit is *equally likely* to be 1 or 0. So here we will take this assumption.
- *Noise statistics*: Knowing the statistical behaviour of the noise will help the receiver make a decision. For instance, if the noise is more likely to be near zero, a good receiver would be the one that picks the nearer of the two possible transmit voltages as compared to the received voltage (that is exactly the *nearest-neighbor rule* that we studied earlier).

It turns out that under the assumption that the information bit is equally likely, the NN decision rule is *optimal*, i.e., maximizes the reliability. So for the rest of this lecture, we will prove that the NN rule is indeed the optimal decision rule that maximizes the reliability.

The optimal decision rule

To begin with, let us list all the information that the receiver has; also see Fig. 2 that illustrates the action taken at the receiver.

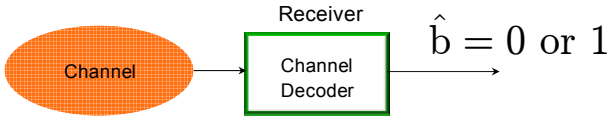


Figure 2: The basic block diagram of a receiver

1. The a priori probabilities of the two values the information bit can take. As mentioned earlier, we will assume that these are equal (0.5 each).
2. The received voltage y.
3. The *encoding rule*: We assume to know how the information bit is mapped into voltages at the transmitter. This means that the mapping in illustrated in Fig. 1 should be known to the receiver. This is a reasonable assumption because in real communication systems, we can fix the transmission rule as a communication *protocol* that both transmitter and receiver should agree upon.

The reliability of communication conditioned on a specific received voltage level (say, a) is simply the probability of a correct decision event, denoted by $\mathcal{C}$ (the information bit b being equal to the estimate $\hat{b}(a)$):

$$\Pr(\mathcal{C}|y = a) = \Pr(b = \hat{b}(a)|y = a). \tag{1}$$

We want to maximize $\Pr(\mathcal{C}|y=a)$, so we should just choose $\hat{b}(a)$ as follows:

$$\hat{b}(a) = \arg\max_{i\in\{0,1\}} \Pr(b=i|y=a) \tag{2}$$

where "arg max" means "argument that maximizes". The quantity $\Pr(b=i|y=a)$ in the above is known as the a *posteriori* probability. Why "posteriori"? This is because the unconditional probability $\Pr(b=i)$ is called the a *priori* probability. So $\Pr(b=i|y=a)$ can be interpreted as the altered probability *after* we make an observation. The word "posteriori" is a Latin word that means "after". Hence, this optimum rule is called the *Maximum A Posteriori probability (MAP)* rule.

Computation of the MAP rule

Now let us try to prove what I claimed earlier: the MAP rule is the same as the NN rule assuming that $\Pr(b=i)=\frac{1}{2}$ for all $i=0,1$. To this end, we need to compute the interested quantity $\Pr(b=i|y=a)$, possibly using the three quantities that the receiver has access to (enumerated at the beginning of the previous section). We face a challenge in the computation though. The challenge comes from the fact that the Gaussian statistics of our interest here are described for analog voltage-level signals. Actually this incurs the following technical problem in calculating the a posteriori probability:

$$\Pr(y=a|b=i) \text{ and } \Pr(y=a) \tag{3}$$

are both zero: the chance that an analog noise level is exactly a value we want is simply zero. So we cannot use the Bayes rule which is crucial in computing the a posteriori probability. Recall that the Bayes rule is:

$$\Pr(A|B) = \frac{\Pr(A)\Pr(B|A)}{\Pr(B)} \tag{4}$$

for any two events A and B where $\Pr(B)\neq 0$. To resolve this issue, the a posteriori probability is defined as

$$\Pr(b=i|y=a) := \lim_{\epsilon\to 0} \Pr(b=i|y\in(a,a+\epsilon)) \tag{5}$$

For $\epsilon>0$, we can now use the Bayes rule to get:

$$\begin{aligned}
\Pr(b=i|y=a) &\overset{(a)}{=} \lim_{\epsilon\to 0} \frac{\Pr(b=i)\Pr(y\in(a,a+\epsilon)|b=i)}{\Pr(y\in(a,a+\epsilon))} \\
&\overset{(b)}{=} \lim_{\epsilon\to 0} \frac{\Pr(b=i)\epsilon f_y(a|b=i)}{\epsilon f_y(a)} \\
&= \frac{\Pr(b=i) f_y(a|b=i)}{f_y(a)}
\end{aligned} \tag{6}$$

where (a) comes from the definition (5) and the Bayes rule; and (b) follows from the definition of the pdfs $f_y(\cdot)$ and $f_y(\cdot|b=i)$ w.r.t. the *analog* voltage signals. Here the conditional pdfs, named as the *likelihoods* in the literature,

$$f_y(a|b=1) \text{ and } f_y(a|b=0) \tag{7}$$

are to be calculated based on the statistical knowledge of the channel noise. So, the MAP rule when the received voltage is equal to a is:

decide $\hat{b} = 1$ if

$$\Pr(b = 1) f_y(a|b = 1) \geq \Pr(b = 0) f_y(a|b = 0) \tag{8}$$

and 0 otherwise.

ML decision rule

Under the assumption that the information bit is equally likely, the MAP rule simplifies even further. It now suffices to just compare the two likelihoods (the two quantities in (7)). The decision rule is then to decide that $\hat{b}$ is 1 if

$$f_y(a|b = 1) \geq f_y(a|b = 0), \tag{9}$$

and 0 otherwise. This rule is called the *maximum likelihood (ML)* rule.

For the Gaussian channel, it is straightforward to calculate the conditional pdf of the received voltage:

$$\begin{aligned} f_y(a|b = 1) &\overset{(a)}{=} f_y(a|x = +\sqrt{E}) \\ &\overset{(b)}{=} f_w(a - \sqrt{E}|x = +\sqrt{E}) \\ &\overset{(c)}{=} f_w(a - \sqrt{E}) \end{aligned} \tag{10}$$

where (a) comes from the fact that the event $b = 1$ is equivalent to the event $x = +\sqrt{E}$ due to our encoding rule; (b) is due to the fact that the event $y = a$ is equivalent to the event $w = a - \sqrt{E}$; (c) is because of the independence between w and x. Here $f_w(\cdot) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\cdot)^2}{2\sigma^2}}$. So, the ML rule for the Gaussian channel is:

decide $\hat{b} = 1$ if

$$f_w(a - \sqrt{E}) \geq f_w(a + \sqrt{E}) \tag{11}$$

and 0 otherwise.

Using the Gaussian pdf, we can further simplify the ML rule: decide $\hat{b} = 1$ if

$$f_w(a - \sqrt{E}) \geq f_w(a + \sqrt{E}) \tag{12}$$

$$(a - \sqrt{E})^2 \leq (a + \sqrt{E})^2 \tag{13}$$

$$4\sqrt{E}a \geq 0 \tag{14}$$

$$a \geq 0. \tag{15}$$

In other words, the ML decision rule take the received voltage $y = a$ and estimates:

$a \geq 0 \Longrightarrow 1$ was sent;

Else, 0 was sent.

Fig. 3 illustrates the ML decision rule when superposed on the "bits to voltage" mapping (cf. Fig. 1). The decision rule picks that transmitted voltage level that is closer to the received voltage (closer in the usual sense of Euclidean distance). Hence, the maximum likelihood (ML) rule is exactly the nearest neighbor (NN) rule.

Look Ahead

In the next lecture, we will calculate the reliability of reception using the ML decision rule. Our focus will be on arriving at an understanding of the relation between reliability level, the transmitted energy constraint and the variance of the noise.

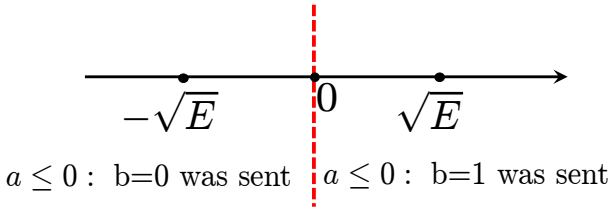


Figure 3: The ML rule superposed on Fig. 1

April 1, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 5: Error probability

Recap

Last time, we started working on transmission/reception schemes for the Gaussian channel. Unlike the simple channel example that we investigated in Lecture 2, the Gaussian channel is faced with unavoidable communication errors. The error concept then motivated us to introduce the definition of the optimal decision rule: the one that maximizes the success rate of communication, simply being referred to as *reliability* (or equivalently minimizes the probability of error). Using this definition, we then showed that the optimal decision rule is always the maximum a posteriori probability (MAP) rule:

$$\hat{b}_{\mathsf{MAP}}(a) = \arg \max_{i \in \{0,1\}} \Pr(b = i | y = a). \tag{1}$$

Next we showed that for the Gaussian channel together with the reasonable assumption of the equal a priori probabilities, the MAP rule reduces to the maximum likelihood (ML) rule:

$$\hat{b}_{\mathsf{ML}}(a) = \arg \max_{i \in \{0,1\}} f_y(a | b = i). \tag{2}$$

Lastly we found that the ML rule matches with the intuitive NN decision rule:

$$\hat{b}_{\mathsf{NN}}(a) = \arg \min_{i \in \{0,1\}} \|a - (2 * i - 1)\sqrt{E}\|. \tag{3}$$

At the end of the last lecture, I then emphasized that one of our main goals in the course is to characterize the tradeoff relationship between the following three: (i) energy budget E; (ii) the number of transmission bits; (iii) reliability of communication. Obviously this requires us to know how to compute reliability when using the NN rule.

Today's lecture

Today we will make an attempt towards achieving the goal. Specifically we are going to do the following three. First we will review the simple tx/rx schemes under the simple case of sending one bit. Next we will compute reliability. Lastly we will characterize the tradeoff relationship between reliability and energy budget.

Recall the transmission/reception schemes

Let us start by reviewing the transmission/reception schemes under the simple setting where we transmit only one bit. We employed a very simple transmission scheme: PAM. We used the optimal decision rule, which was shown to be the NN rule. See Fig. 1.

Reliability of communication

Recall that conditioned on $y = a$, the reliability of communication is simply the probability of the success event that the information bit b is equal to its estimate $\hat{b}(a)$:

$$\Pr(\mathcal{C} | y = a) = \Pr(b = \hat{b}(a) | y = a). \tag{4}$$

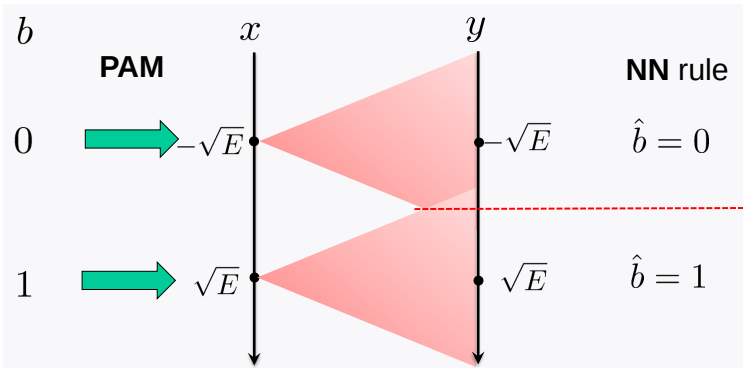


Figure 1: The transmission/reception schemes: PAM and the NN rule.

But there is an issue in this quantity of interest. The issue that the quantity is a function of a and hence it varies depending on a particular realization of the received signal y. So it cannot serve as a representative quantity that is obviously preferred to be fixed. In an effort to find a representative quantity that does not rely on a specific realization, we consider its *expectation*: a weighted sum of the reliabilities over all of the realizations a's with the weight determined by the pdf $f_y(a)$:

$$\begin{aligned} &\int_{-\infty}^{\infty} f_y(a)\Pr(\mathcal{C}|y=a)da \\ &\overset{(a)}{=} \int_{-\infty}^{\infty} \Pr(\text{correct decision}, y=a)da \\ &\overset{(b)}{=} \Pr(\mathcal{C}) \end{aligned} \tag{5}$$

where (a) is due to the definition of conditional probability; and (b) is because of the total probability law. If you don't know what the total probability law is, please see BT. This is one of very important laws in probability.

Probability of error

Alternatively we may consider the average *error* probability:

$$\Pr(\mathcal{E}) = 1 - \Pr(\mathcal{C}) \tag{6}$$

where $\mathcal{E}$ denotes an error event. In this course, we will focus on evaluating $\Pr(\mathcal{E})$ instead. The receiver makes an error if it decides that a 1 was sent when a 0 was sent, or vice versa. The average error is a weighted sum of the probabilities of these two types of error events, with the weights being equal to the a priori probabilities of the information bit:

$$\begin{aligned} \Pr(\mathcal{E}) &= \Pr(\mathcal{E}, b=0) + \Pr(\mathcal{E}, b=1) \\ &= \Pr(b=0)\Pr(\mathcal{E}|b=0) + \Pr(b=1)\Pr(\mathcal{E}|b=1). \end{aligned} \tag{7}$$

Why? Again this is due to the total probability law! More precisely, the law dictates the first equality in the above and the second equality comes from the definition of conditional probability. As mentioned earlier, we assume that the a priori probabilities are equal (to 0.5 each). Let us focus on one of the error events by assuming that the information bit was actually 0. Then with

the NN rule, we get:

$$\begin{aligned}
\Pr(\mathcal{E}|b=0) &= \Pr(\hat{b}(y)=1|b=0) \\
&\overset{(a)}{=} \Pr(y>0|b=0) \\
&= \Pr(-\sqrt{E}+w>0|b=0) \\
&\overset{(b)}{=} \Pr\left(\frac{w}{\sigma} > \frac{\sqrt{E}}{\sigma}\right) \\
&\overset{(c)}{=} \int_{\frac{\sqrt{E}}{\sigma}}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} dz \\
&\overset{(d)}{=} Q\left(\frac{\sqrt{E}}{\sigma}\right)
\end{aligned} \tag{8}$$

where (a) is due to the NN rule[1]; (b) is because of the independence between w and b; (c) follows from the fact that $\frac{w}{\sigma} \sim \mathcal{N}(0,1)$ (why? please think about it); and (d) comes from the definition of $Q(a) := \int_a^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} dz$. The Gaussian random variable with mean 0 and variance 1 is called the *standard* Gaussian. It is a very important and useful random variable and the Q-function is always tabulated in all probability textbooks and wikipedia. This integration can also be computed numerically by using a command "`erfc`" in `python`:

$$\texttt{erfc}(x) := \int_x^\infty \frac{2}{\sqrt{\pi}} e^{-t^2} dt.$$

The relation between $Q(a)$ and $\texttt{erfc}(x)$ is:

$$\begin{aligned}
Q(a) &:= \int_a^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} dz \\
&\overset{(a)}{=} \int_{\frac{a}{\sqrt{2}}}^\infty \frac{1}{\sqrt{\pi}} e^{-t^2} dt \\
&= \frac{1}{2} \cdot \texttt{erfc}\left(\frac{a}{\sqrt{2}}\right).
\end{aligned}$$

where (a) comes from the change of variable $t := \frac{z}{\sqrt{2}}$ ($dz = \sqrt{2}dt$).

Let us now consider $\Pr(\mathcal{E}|b=1)$. Actually the error does not depend on which information bit is transmitted. The complete *symmetry* of the mapping from the bit values to the voltages levels and the NN decision rule (see Fig. 1) would suggest that the two error probabilities are identical. For completeness, we go through the calculation for $\Pr(\mathcal{E}|b=1)$ and verify that it is indeed the same:

$$\begin{aligned}
\Pr(\mathcal{E}|b=1) &= \Pr(\hat{b}(y)=0|b=1) \\
&= \Pr(y \le 0|b=1) \\
&= \Pr(\sqrt{E}+w \le 0|b=1) \\
&= \Pr\left(\frac{w}{\sigma} \le -\frac{\sqrt{E}}{\sigma}\right) \\
&= Q\left(\frac{\sqrt{E}}{\sigma}\right).
\end{aligned} \tag{9}$$

[1]For simplicity of analysis, we assume that for the event $y=0$, $\hat{b}$ is decided to be 0 although we should flip a coin in the case as per the NN rule. Since the event has measure-zero (the probability of the event being occurred is 0), the error analysis remains the same.

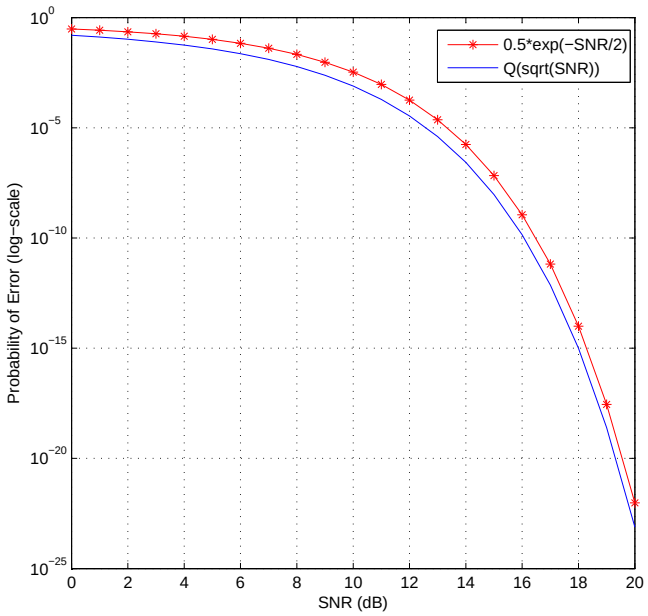


Figure 2: Comparison of $Q(\sqrt{\mathsf{SNR}})$ and its upper bound $\frac{1}{2}e^{-\frac{\mathsf{SNR}}{2}}$.

This together with (8) gives:

$$\Pr(\mathcal{E}) = Q\left(\frac{\sqrt{E}}{\sigma}\right). \tag{10}$$

SNR and error probability

The first observation that one can make from the expression (10) on error probability is that it depends only on the ratio of E and σ^2: the error probability is

$$\Pr(\mathcal{E}) = Q\left(\sqrt{\mathsf{SNR}}\right). \tag{11}$$

We have already seen this phenomenon earlier. This ratio is called the signal to noise ratio, or simply SNR. Basically, the communication engineer can design for a certain reliability level by choosing an appropriate SNR setting. While the $Q(\cdot)$ function can be found in standard statistical tables, it is useful for the communication engineer to have a rule of thumb for how sensitive this SNR "knob" is in terms of the reliability each setting offers. For instance, it would be useful to know by how much the reliability increases if we double the SNR setting. To do this, it helps to use the following approximation:

$$Q(a) \approx \frac{1}{2}e^{-\frac{a^2}{2}}. \tag{12}$$

Here we use a very rough approximation that we denote by "$\approx$". This approximation means that the exponent of $Q(a)$ is very close to that of $\frac{1}{2}e^{-\frac{a^2}{2}}$, i.e., $\ln Q(a) \approx \ln(\frac{1}{2}e^{-\frac{a^2}{2}})$. The reason that we consider this very rough approximation is that the exponent plays an enough role as a proper measure when we deal with the probability of error. For example, the following probabilities of error (10^{-7} and $2 \cdot 10^{-7}$) are considered to provide almost the same performance although those

differ by two times, since they have roughly the same exponent. Actually $\frac{1}{2}e^{-\frac{a^2}{2}}$ is an upper bound of the Q-function. See Fig. 2. You will have a chance to check this rigorously from one of the problems in PS2. In this course, we will frequently approximate $Q(a)$ by the upper bound. This approximation implies that the unreliability level

$$Q\left(\sqrt{\mathsf{SNR}}\right) \approx \frac{1}{2}e^{-\frac{\mathsf{SNR}}{2}}. \tag{13}$$

Equation (13) is saying something very interesting. It says that the SNR has an exponential effect on the probability of error. What happens if we double the energy, i.e., $\tilde{E} = 2E$?

$$Q\left(\sqrt{2\mathsf{SNR}}\right) \approx \frac{1}{2}e^{-\mathsf{SNR}} \approx \left[Q\left(\sqrt{\mathsf{SNR}}\right)\right]^2. \tag{14}$$

Doubling the energy has thus squared the probability of error and squaring a number less than 1 means that we have significantly reduced the probability of error. Actually a typical range of SNR is $100 \sim 10000$, and for this range, the probability of error is quite a small value.

Look ahead

So far we have considered the case of sending only one bit. Recall that we are interested in characterizing the tradeoff relationship between the three quantities: (i) energy budget; (ii) the number of transmission bits; (iii) error probability. So next time we will figure out by how much the reliability is reduced when we transmit multiple bits under an energy budget.

April 3, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 6: Transmitting multiple bits

Recap

Last time we studied how to compute the error probability when using the optimal decision rule in the case of sending only one bit. Based on this, we could find a tradeoff relationship between error probability and energy budget. From this, we saw that the SNR has a significant effect upon error probability: doubling the SNR, the doubly-exponential decaying in error probability.

At the end of the last time, I mentioned that we are interested in investigating the tradeoff relationship between three quantities: (i) energy budget; (ii) the number of transmission bits; (iii) error probability.

Today's lecture

Today we are going to derive such triplet relationship. Specifically what we are going to do are four-folded. First, we will consider a simple setting of sending two bits and employ a simple transmission scheme: PAM. We will then derive the optimal decision rule, thereby showing that it is the same as the NN decision rule. Next we will analyse error probability. Finally we will extend to an arbitrary number of k bits to figure out the tradeoff between the three quantities.

A transmission scheme for sending two bits

For illustrative purpose, let us first consider a simple case: sending two bits. Assume that we have the same energy budget E as in the single-bit transmission case. As in Lecture 2, we adopt the same PAM: mapping the bits to voltage levels so that they are as far apart from each other. Specifically, the encoding rule is the following:

$$\begin{aligned} 00 &\to -\sqrt{E} \\ 01 &\to -\frac{\sqrt{E}}{3} \\ 10 &\to \frac{\sqrt{E}}{3} \\ 11 &\to \sqrt{E}. \end{aligned}$$

Also see Fig. 1 for illustration. Here there is one important factor that I would like to highlight: the minimum distance, say d. It is defined as the minimum distance among all distances between any two voltage levels. So in this case, $d = \frac{2\sqrt{E}}{3}$. It turns out this factor plays a crucial role to compute error probability. This will be clearer soon.

The optimal decision rule

Now what is the optimal decision rule w.r.t. such transmission scheme? Recall that the optimal decision rule is the one that maximizes the reliability of communication. As we proved during the past few lectures, it is always the MAP rule: choosing a transmission candidate that maximizes the a posteriori probability:

$$\hat{b}_{\mathsf{MAP}} = \arg \max_{i,j \in \{0,1\}} \Pr(b = ij | y = a). \tag{1}$$

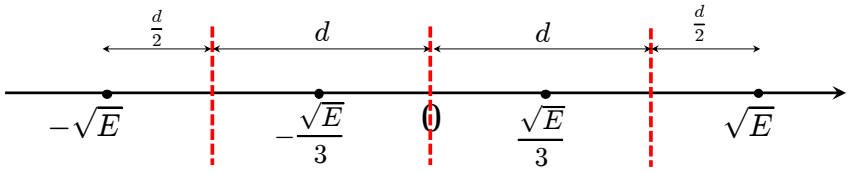


Figure 1: Sending two bits over the Gaussian channel using PAM.

Similar to the prior settings, let us consider a realistic scenario where we have no idea as to the statistics of two information bits. So we assume that transmission bits are simply equally likely: $\Pr(b = ij) = \frac{1}{4}$ for all possible pairs of (i, j). Massaging the above MAP rule together with this assumption, we can then show that the MAP is the same as the ML rule:

$$\begin{aligned}
\hat{b}_{\mathsf{MAP}} &= \arg \max_{i,j\in\{0,1\}} \Pr(b = ij|y = a) \\
&\overset{(a)}{=} \arg \max_{i,j\in\{0,1\}} \frac{f_{b,y}(ij, a)}{f_y(a)} \\
&\overset{(b)}{=} \arg \max_{i,j\in\{0,1\}} \frac{\Pr(b = ij) f_y(a|b = ij)}{f_y(a)} \\
&\overset{(c)}{=} \arg \max_{i,j\in\{0,1\}} \Pr(b = ij) f_y(a|b = ij) \\
&\overset{(d)}{=} \arg \max_{i,j\in\{0,1\}} f_y(a|b = ij) =: \hat{b}_{\mathsf{ML}}
\end{aligned} \tag{2}$$

where (a) follows from the definition of conditional probability (here $f_{b,y}(ij, a)$ denotes the joint pdf of b and y); (b) is due to the definition of conditional probability (this together with (a) is essentially the Bayes rule); (c) is because $f_y(a)$ is irrelevant of the change of information bits b; and (d) comes from the assumption that the bits are equally likely.

Using the properties of the Gaussian likelihood function $f_y(a|b = ij)$ that it is *symmetric* around the mean and more likely near the mean, we can readily verify that the ML rule is the same as the NN rule: picking a transmit voltage level that is closest to the received voltage level. See Fig. 2 for justification.

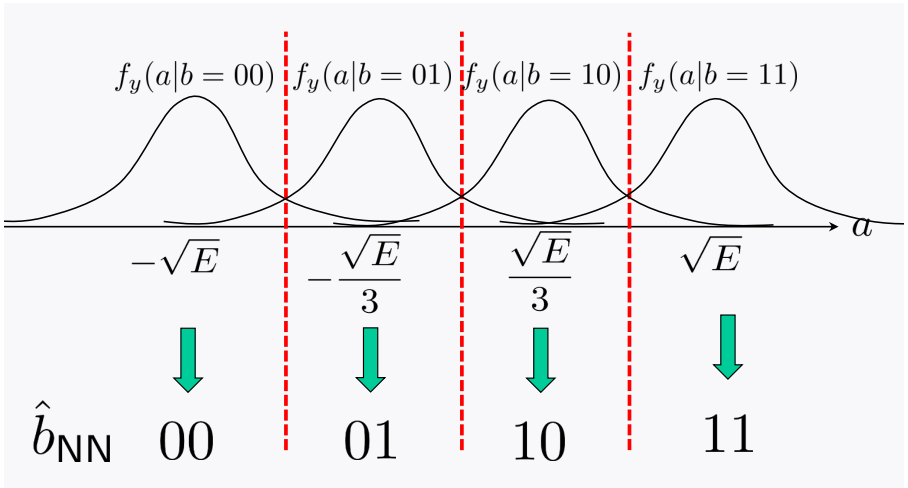


Figure 2: The ML rule is equivalent to the NN rule.

Error probability

Now let us analyse the probability of error when we use the optimal NN decision rule under the PAM illustrated in Fig. 1. Using the total probability law, we get:

$$\begin{aligned}\Pr(\mathcal{E}) = &\Pr(b=00)\Pr(\mathcal{E}|b=00) + \Pr(b=11)\Pr(\mathcal{E}|b=11) \\ &+ \Pr(b=01)\Pr(\mathcal{E}|b=01) + \Pr(b=10)\Pr(\mathcal{E}|b=10).\end{aligned} \tag{3}$$

From the encoding rule as illustrated in Fig. 1, we see that there are two types of voltage levels: (i) *inner* voltage levels ($-\frac{\sqrt{E}}{3}$ and $\frac{\sqrt{E}}{3}$); (ii) *outer* voltage levels ($-\sqrt{E}$ and $\sqrt{E}$). With the *symmetry* argument that we made in the last lecture, we can see that the probability of error for the inner voltage levels are the same with each other. Similarly for the outer voltage levels. So let us consider only one of the voltage levels within the same kind.

Let us first calculate the error probability w.r.t. an outer level, say $-\sqrt{E}$. Using similar procedures that we did in Lecture 5, we get:

$$\begin{aligned}\Pr\left(\mathcal{E}|b=00\right) &= \Pr\left(\hat{b}(y) \neq 00 | b=00\right) \\ &\overset{(a)}{=} \Pr\left(y > -\sqrt{E} + \frac{d}{2} | b=00\right) \\ &\overset{(b)}{=} \Pr\left(w > \frac{d}{2}\right) \\ &\overset{(c)}{=} Q\left(\frac{d}{2\sigma}\right) \\ &\overset{(d)}{=} Q\left(\frac{\sqrt{E}}{3\sigma}\right)\end{aligned} \tag{4}$$

where (a) is because the event $\hat{b}(y) \neq 00$ is equivalent to the event $y > -\sqrt{E} + \frac{d}{2}$ under the NN rule; (b) is due to the independence between w and b and the fact that $w = y - x = y + \sqrt{E}$ under $b = 00$; (c) comes from the fact that $\frac{w}{\sigma} \sim \mathcal{N}(0,1)$; and (d) is due to $d = \frac{2\sqrt{E}}{3}$.

On the other hand, the error probability w.r.t. an inner voltage level, say $-\frac{\sqrt{E}}{3}$ is:

$$\begin{aligned}\Pr\left(\mathcal{E}|b=01\right) &= \Pr\left(\left\{y > -\frac{\sqrt{E}}{3} + \frac{d}{2}\right\} \cup \left\{y < -\frac{\sqrt{E}}{3} - \frac{d}{2}\right\} | b=01\right) \\ &= \Pr\left(\left\{w > \frac{d}{2}\right\} \cup \left\{w < -\frac{d}{2}\right\}\right) \\ &\overset{(a)}{=} \Pr\left(w > \frac{d}{2}\right) + \Pr\left(w < -\frac{d}{2}\right) \\ &\overset{(b)}{=} 2Q\left(\frac{\sqrt{E}}{3\sigma}\right)\end{aligned} \tag{5}$$

where (a) is because the two associated events are disjoint; and (b) is due to the fact that $-w$ follows the same distribution as $w \sim \mathcal{N}(0,1)$.

Applying (4) and (5) to (3), we compute the average probability of making an error (averaged

over all the four different voltage levels):

$$\begin{aligned}\Pr(\mathcal{E}) &= 2 \times \frac{1}{4} \times Q\left(\frac{\sqrt{E}}{3\sigma}\right) + 2 \times \frac{1}{4} \times 2Q\left(\frac{\sqrt{E}}{3\sigma}\right) \\ &= \frac{3}{2} Q\left(\frac{\sqrt{E}}{3\sigma}\right).\end{aligned} \tag{6}$$

Extension to the case of sending k bits

So far we have focused on the case of transmitting $k = 2$ information bits. The same procedures carry over to the general number k of information bits.

Extending the encoding rule in Fig. 1 to general k, we can now see that the minimum distance d in this setting reads:

$$d = \frac{2\sqrt{E}}{2^k - 1} \tag{7}$$

where the denominator $2^k - 1$ denotes the number of intervals in the bits-to-voltage mapping. We can also readily verify that the optimal decision rule (MAP) is the same as the ML rule and the ML rule is identical to the NN rule. See Fig. 3 for a short justification.

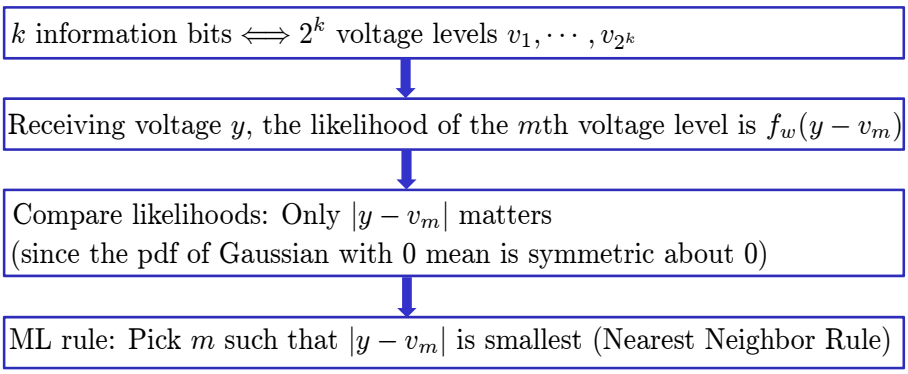


Figure 3: ML rule for k information bits is the nearest neighbor rule.

Now using the *symmetry* argument, we can calculate the error probability as:

$$\begin{aligned}\Pr(\mathcal{E}) &= \frac{\#\text{ of outer levels}}{2^k} \cdot \Pr(\mathcal{E}|b = 00\cdots0) + \frac{\#\text{ of inner levels}}{2^k} \cdot \Pr(\mathcal{E}|b = 00\cdots1) \\ &= \frac{2}{2^k} \cdot \Pr(\mathcal{E}|b = 00\cdots0) + \frac{2^k - 2}{2^k} \cdot \Pr(\mathcal{E}|b = 00\cdots1)\end{aligned} \tag{8}$$

The probability of making an error given the transmission of an outer voltage level is:

$$\begin{aligned}\Pr(\mathcal{E}|b = 00\cdots0) &\overset{(a)}{=} \Pr\left(w > \frac{d}{2}\right) \\ &\overset{(b)}{=} Q\left(\frac{d}{2\sigma}\right) \\ &\overset{(c)}{=} Q\left(\frac{\sqrt{E}}{(2^k - 1)\sigma}\right),\end{aligned} \tag{9}$$

where (a) is because the only thing that matters in an error event is whether the noise w is beyond the minimum distance divided by 2; (b) is due to $\frac{w}{\sigma} \sim \mathcal{N}(0,1)$; and (c) comes from the fact that the minimum distance is $d = \frac{2\sqrt{E}}{2^k-1}$.

As in the $k = 2$ case, the error probability w.r.t. an inner voltage level is simply twice the one w.r.t. an outer voltage level:

$$\Pr(\mathcal{E}|b = 00\cdots1) = 2Q\left(\frac{\sqrt{E}}{(2^k-1)\sigma}\right), \tag{10}$$

Why? An error event w.r.t. an inner voltage level involves another tail probability of error.

Hence, the error probability is readily calculated to be:

$$\begin{aligned}\Pr(\mathcal{E}) &= \left(2 - \frac{1}{2^{k-1}}\right) Q\left(\frac{\sqrt{E}}{(2^k-1)\sigma}\right) \\ &= \left(2 - \frac{1}{2^{k-1}}\right) Q\left(\frac{\sqrt{\mathsf{SNR}}}{2^k-1}\right).\end{aligned} \tag{11}$$

An engineering conclusion

At the end of Lecture 2, in the simple channel example where $-\sigma \le w \le \sigma$, we noted the relationship between k and SNR:

$$k = \log_2\left(1 + \sqrt{\mathsf{SNR}}\right). \tag{12}$$

Here we saw that k has a logarithmic relationship with $\sqrt{\mathsf{SNR}}$: k increases very slowly (logarithmically) with an increase in $\sqrt{\mathsf{SNR}}$.

Interestingly even in the Gaussian channel where having errors is unavoidable and hence the concept of reliability comes naturally, k has exactly the same (logarithmic) relationship with $\sqrt{\mathsf{SNR}}$. To see this clearly, recall the key equation (11) that we derived in the prior section. Consider a certain reliability level. Now in order to guarantee the same reliability, the input argument in the Q-function, say t, should be similar among different pairs of $(k, \sqrt{\mathsf{SNR}})$:

$$t \approx \frac{\sqrt{\mathsf{SNR}}}{2^k - 1}. \tag{13}$$

This implies that

$$k \approx \log_2\left(1 + \frac{\sqrt{\mathsf{SNR}}}{t}\right). \tag{14}$$

Hence, we can see that even in the Gaussian channel, the essential relationship between transmit energy and the number of information bits you can reliably transmit (at any reliability level) is unchanged.

Look ahead

One key observation that we can make from (11) is that error probability shoots up to 1, as the number k of information bits increases. This is sort of disappointing. Also if you think about current communication systems where we still enable reliable communication even with a very large number k of bits, this result looks weird. You may then ask: what's wrong with

the tradeoff relationship (11)? Is our math derivation simply wrong? Or something else? It turns out it is due to something else: this is just a consequence of the specific communication scheme (PAM) that we have chosen in our setting. Actually there are multiple ways to improve reliability significantly. Next time, we will investigate one of them.

April 8, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 7: Sequential communication

Recap

Last time, we investigated the tradeoff relationship between three quantities: (i) SNR; (ii) reliability; (iii) the number of transmission bits (say B bits). This study was based on a particular transmission scheme: PAM; see Fig. 1 for illustration. Using PAM together with the assump-

Figure 1: Pulse amplitude modulation.

tion that the bits are equally likely, we showed that the optimal decision rule (the MAP rule) is equivalent to the intuitive NN rule. We then employed basic probability tools to analyse the error probability:

$$\Pr(\mathcal{E}) = \frac{2}{2^B} \cdot Q\left(\frac{d}{2\sigma}\right) + \frac{2^B - 2}{2^B} \cdot 2Q\left(\frac{d}{2\sigma}\right)$$

where the number 2 colored in blue indicates the number of outer voltage levels associated with the one-sided tail error event and the number $2^B - 2$ colored in denotes the number of inner voltage levels associated with the two-sided tail error event. Here the minimum distance d plays a crucial role to compute the error probability. Plugging $d = \frac{2\sqrt{E}}{2^B - 1}$, we get:

$$\begin{aligned} \Pr(\mathcal{E}) &= \left(2 - \frac{1}{2^{B-1}}\right) Q\left(\frac{\sqrt{\mathsf{SNR}}}{2^B - 1}\right) \\ &\approx \left(1 - \frac{1}{2^B}\right) e^{-\frac{\mathsf{SNR}}{2(2^B-1)^2}}. \end{aligned} \tag{1}$$

At the end of the last lecture, one critical observation that we made was: the error probability surges with an increase in B. This is in fact disappointing because in practical communication scenarios, there are tons of bits to communicate (think about typical file transmission in our daily life which amounts easily to a few mega (or giga) bytes (1 byte = 8 bits), and this huge-sized file transmission yields *almost-sure error* under a reasonable SNR level (ranging typically from 100 to 10,000). On the other hand, our current communication systems are working properly to enable reliable communication even for huge-sized file transmission that amounts to a few giga bytes. A natural question that one can ask is then: What is wrong with above tradeoff relationship (1)?

Today's lecture

Today we will try to address the question. Specifically what we are going to do are three folded. First of all, I will point out that the tradeoff relationship (1) is an artifact of the specific

communication scheme (PAM) that we have chosen. By introducing another communication resource that is not employed in the prior scheme, we will then investigate another natural alternative that exploits the introduced communication resource. Finally we will derive error probability under the scheme to figure out whether we can obtain a reasonably-small error probability even with a very large B.

A natural alternative

Recall the previous transmission scheme in Fig. 1. Here one key observation is that we employ only one time slot, although there can be much more time slots available for communication. In other words, we never exploited another very natural communication resource: *time*!

So one natural alternative is to employ multiple time slots. For instance, we can think of the following transmission scheme, as illustrated in Fig. 2. We read the information bits serially,

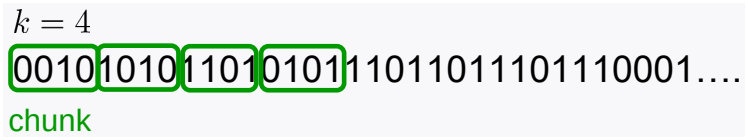


Figure 2: Sequential communication.

say, k bits at a time. Let us consider these k bits as one entity that we are going to send in one time slot. Let's name the entity *chunk*. Then, one natural idea is to send these chunks in a sequential manner. For example, in time 1, we send one chunk spanning the first k bits, using PAM. In time 2, we transmit another chuck again (spanning the next k bits) using the same PAM, and all the way up to the last chunk. We call the scheme "sequential communication". In the example of Fig. 2, the chunk size k is 4. In the case of sending B bits, the number (say n) of chunks is B/k.

The optimal decision rule

Now the question is: Can this sequential communication boost up reliability significantly with a moderate level of SNR? To answer this question, we have to evaluate the reliability (equivalently the probability of error) when we apply this sequential communication scheme. To this end, we first need to figure out what the optimum decision rule is.

By definition, the optimal decision rule is always the MAP rule. Assuming a typical setting where the bits are equally likely:

$$\Pr((b_1, \ldots, b_B) = (i_1, \ldots, i_B)) = \frac{1}{2^B},$$

the MAP rule is the same as the ML rule. If you still don't know why, please review Lecture 4. Now in order to figure out what the ML rule is, we need to know about the pdf of the received signal:

$$y[m] = x[m] + w[m], \qquad m = 1, \ldots, n, \tag{2}$$

where $(x[m], w[m])$ indicate the transmitted signal and the additive noise at the mth chunk. Notice that the received signal contains a sequence of additive noises $w[m]$'s. So we need to figure out the statistics of the noise sequence.

A statistical model for $w[m]$'s

Our previous discussion in Lecture 3 lets us argue that the statistics of the additive noise at any time instant is *Gaussian*. In practice, the statistics does not change over time significantly. So one reasonable assumption is that the mean and the variance are unchanged over time. Also the noises have little correlation across different time slots. So another reasonable assumption is that $w[m]$'s are statistically independent. In other words, the noises are assumed to be i.i.d. Such a noise model is said to be *white*. So we will refer to this noise as the additive white Gaussian noise or simply AWGN. You may wonder why we name it "white". It turns out the AWGN contains the frequency components that span the entire spectrum. So it has the same property that the *white light* has: containing every frequency component. That's why people name it white.

The ML rule

Now under the AWGN channel, what is the ML decision rule? First consider the first chunk of k bits, say $b[1]$. The ML estimate $\widehat{b[1]}$ is obviously a function of the sequence of the received signals. So let us denote it by

$$\widehat{b[1]} = \mathsf{ML}(b[1]|y[1], \ldots, y[n]).$$

Recall the typical assumption that we made earlier: the bits are equally likely.

$$\Pr((b_1, \cdots, b_B) = (i_1, \cdots, i_B)) = \frac{1}{2^B}.$$

From this, one can also show that the bits b_i's are i.i.d. Why? Check in PS3. This then implies that the transmitted signals $x[m]$'s at different time slots are also statistically independent with each other. Since $w[m]$'s are i.i.d. under the AWGN, the received signals $y[m]$'s are also statistically independent. This then suggests that the first chuck of bits $b[1]$ is independent of the received signals w.r.t. the remaining chunks $(y[2], \ldots, y[n])$. So one can readily see that $(y[2], \ldots, y[n])$ does not help decoding $\widehat{b[1]}$. This yields:

$$\begin{aligned}\widehat{b[1]} &= \mathsf{ML}(b[1]|y[1], y[2], \ldots, y[n]) \\ &= \mathsf{ML}(b[1]|y[1]).\end{aligned}$$

Applying the same argument at different time slots, we arrive at:

$$\widehat{b[m]} = \mathsf{ML}(b[m]|y[m]), \qquad m = 1, 2, \ldots, n.$$

In other words, sequential transmission over the AWGN channel naturally leads to sequential reception as well: at each time m, make an ML decision on the local k bits transmitted over that time instant. As we verified in the previous lectures, this local ML rule is the same as the NN rule.

Reliability of sequential communication

In the last lecture, we have analysed the reliability of communication w.r.t. a particular time slot. Denoting by p the error probability for one time slot, we get:

$$p = \left(2 - \frac{1}{2^{k-1}}\right) Q\left(\frac{\sqrt{\mathsf{SNR}}}{2^k - 1}\right). \tag{3}$$

Due to the statistical independence of error events, the reliability w.r.t. the entire time slots n would be:

$$\Pr\left(\mathcal{C}_1 \cap \mathcal{C}_2 \cap \cdots \cap \mathcal{C}_n\right) = \prod_{i=1}^{n} \Pr(\mathcal{C}_i) = (1-p)^n, \tag{4}$$

where $\mathcal{C}_i$ denotes the correction decision event at time i. Now we get a hint of what might go wrong with sequential communication: even though we have ensured that the reliability is pretty high at any given time instant, the overall reliability of communicating correctly at *each and every* time instant is quite slim. To get a concrete feel, let us consider the following example: $k = 4$ and $n = 20000$. Suppose we had set the error probability p at any time instant to be 10^{-4} (based on (3), how much should the SNR be?). Then the reliability of the entire sequential communication process is, from (4),

$$(1 - 10^{-4})^{20000} \approx 2 \times 10^{-9}. \tag{5}$$

This is almost zero. Obviously this is a very low reliability level. The only way to compensate for this is to set the SNR so high that the overall reliability is large enough. Typical reliability levels are of the order of 10^{-10} or so. For large enough files this would mean astronomical SNR values. Even small files need very large SNR values. Details will be explored in PS3.

Look ahead

You may wonder if the above phenomenon is fundamental, i.e., the unavoidable price for reliable communication. Successful communication technologies around us provide a solid clue to the answer. It turns out the trouble is with the sequential nature of transmission and reception. Next time, we will study another communication scheme that can improve reliability significantly.

April 10, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 8: Repetition coding

Recap

Last time, in an effort to improve reliability, we investigated a chunk-based sequential communication scheme where we read information bits as a chunk unit (consisting of k bits), apply PAM w.r.t. each chunk, and then transmit the PAM signal using a single time slot. See Fig. 1.

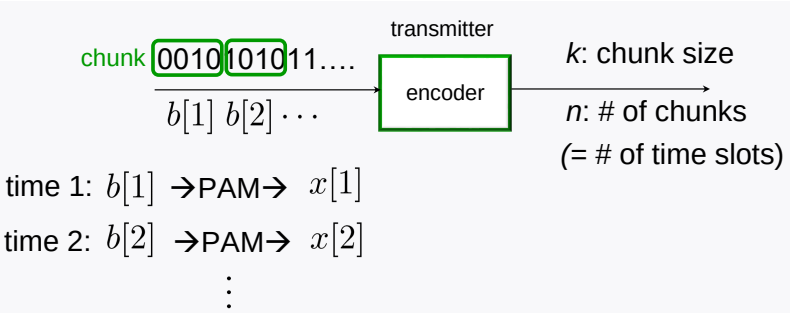


Figure 1: A chunk-based sequential communication scheme.

We showed that under the uniform distribution assumption on information bits, the optimal decision rule is simply the local NN rule where we apply the NN rule locally w.r.t. each received signal $y[m]$. We found that this sequential communication scheme could indeed improve reliability relative to the naive PAM using only one time slot. But the improvement was not that impressive, as error probability cannot be made small enough.

Recall in current communication systems that we enable reliable transmission even with many bits. This implies that the sequential communication is not the one that is employed in the current systems. There must be something else.

Today's lecture

Today we will study another communication scheme that can improve reliability significantly. Specifically we are going to do the following three. First I will introduce a new idea for transmission, which was never discussed before, but is the one that people can naturally come up with. Next we will study a simple scheme (inspired by the idea) in the simplest non-trivial setting where we wish to transmit a single bit ($B = 1$) yet using *two* time slots ($n = 2$). Finally we will extend to arbitrary (B, n), thereby demonstrating that the extended scheme can make the probability of error arbitrarily small.

The idea of repetition

Recall the sequential communication scheme illustrated in Fig. 1. Here we see that each chunk signal contains only the corresponding bit string, being irrelevant of those w.r.t. other chunks. For instance, $x[2]$ is concerned only about $b[2]$, having nothing to do with $b[1]$. Actually this is not what we are doing for communication in our daily life. In our daily conversation, what are we doing especially when communication fails? We say *again* until a listened person can

understand. But this idea, so called the *repetition* idea, was never implemented in the sequential communication scheme, although it is a very natural idea that anybody can easily think of. So one natural alternative is to employ such repetition idea. It turns out the idea of repetition helps a lot. So from now on, we will show that a simple repetition coding scheme can indeed improve reliability significantly.

Repetition coding for the simplest case $(B, n) = (1, 2)$

Let us first consider the simplest yet non-trivial setting. That must be the case where we send only one bit ($B = 1$) but now through two time slots ($n = 2$). Assume that the energy budget per time slot is E. A naive trial based on the repetition idea could be: sending a voltage level $\pm\sqrt{E}$ at time 1 and send the *same* voltage again at time 2. See Fig. 2. A fancy term for this

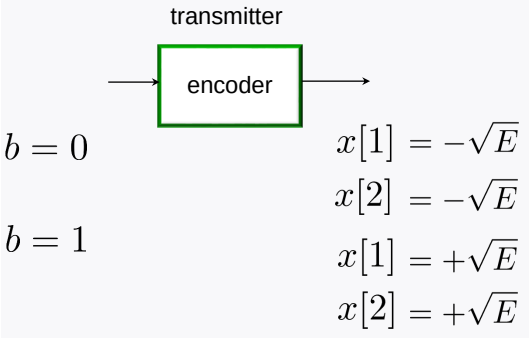


Figure 2: Repetition coding for $(B, n) = (1, 2)$.

kind of a scheme is *repetition coding*. For this coding scheme, the received signals read:

$$\begin{aligned} y[1] &= x[1] + w[1] = (2b-1)\sqrt{E} + w[1], \\ y[2] &= x[2] + w[2] = (2b-1)\sqrt{E} + w[2]. \end{aligned} \tag{1}$$

Guess for the optimal decision rule

Now what is the optimal decision rule? In other words, how does the MAP rule look like? Remember the case of $n = 1$. Under the uniform distribution on b, the optimal decision rule (MAP) is the same as the ML rule, and it further reduces the NN decision rule. Now any guess for the optimal decision rule when $n = 2$? The following observation gives an insight.

Notice in (1) that $x[1], x[2]$ are identical. So one naive thinking is to add the received signals and then make a decision w.r.t. the sum:

$$\underbrace{y[1] + y[2]}_{=:y} = \underbrace{x[1] + x[2]}_{=:x} + \underbrace{w[1] + w[2]}_{=:w}. \tag{2}$$

This way, one can *align* $x[1]$ and $x[2]$ to boost up the power of an interested signal. On the other hand, $w[1]$ and $w[2]$ point to different random directions, so such booting effect is not expected w.r.t. the summed noise w. It turns out that as you may expect, the NN rule w.r.t. the summed received signal y is indeed the optimal decision rule. Let us prove this in the next section.

The optimal decision rule

Assuming the uniform distribution on b, the MAP is equivalent to the ML rule:

$$\hat{b}_{\mathsf{ML}} = \arg \max_{i \in \{0,1\}} f_{\mathbf{y}}(\mathbf{a}|b=i) \tag{3}$$

where $\mathbf{a} = (a[1], a[2])^T$ denotes a realization of the received signal vector $\mathbf{y} = (y[1], y[2])^T$. Here $(\cdot)^T$ indicates a transpose. Now consider the likelihood function of interest:

$$\begin{aligned}
f_{\mathbf{y}}(\mathbf{a}|b=i) &= f_{\mathbf{y}}\left(a[1], a[2]|b=i\right) \\
&= f_{\mathbf{w}}\left(a[1]-x[1], a[2]-x[2]|b=i\right) \\
&\overset{(a)}{=} f_{\mathbf{w}}\left(a[1]-(2i-1)\sqrt{E}, a[2]-(2i-1)\sqrt{E}|b=i\right) \\
&\overset{(b)}{=} f_{\mathbf{w}}\left(a[1]-(2i-1)\sqrt{E}, a[2]-(2i-1)\sqrt{E}\right) \\
&\overset{(c)}{=} f_{w[1]}\left(a[1]-(2i-1)\sqrt{E}\right) f_{w[2]}\left(a[2]-(2i-1)\sqrt{E}\right) \\
&\overset{(d)}{=} \frac{1}{2\pi\sigma^2}\exp\left(-\frac{1}{2\sigma^2}\left\{\left(a[1]-(2i-1)\sqrt{E}\right)^2+\left(a[2]-(2i-1)\sqrt{E}\right)^2\right\}\right) \\
&= \frac{1}{2\pi\sigma^2}\exp\left(-\frac{1}{2\sigma^2}(a[1]^2+E+a[2]^2+E)\right) \times \exp\left(\frac{1}{\sigma^2}(a[1]+a[2])(2i-1)\sqrt{E}\right)
\end{aligned}$$

where (a) comes from our encoding rule ($x[m] = (2b-1)\sqrt{E}$); (b) is due to the independence between $\mathbf{w}$ and b; (c) is due to the independence of the additive noises at different time slots; and (d) comes from the the explicit pdf of the Gaussian noise.

Here one key observation that we can make is that the first part in the last equality (marked in red) is *irrelevant* of i. Hence, we get:

$$\begin{aligned}
\hat{b}_{\mathsf{ML}} &= \arg \max_{i \in \{0,1\}} f_{\mathbf{y}}(\mathbf{a}|b=i) \\
&= \arg \max_{i \in \{0,1\}} \exp\left(\frac{1}{\sigma^2}(a[1]+a[2])(2i-1)\sqrt{E}\right) \\
&= \arg \max_{i \in \{0,1\}} (a[1]+a[2])(2i-1).
\end{aligned} \tag{4}$$

From this, we see that the sum $a[1]+a[2]$ plays a significant role in the decision:

$$a[1]+a[2] \geq 0 \Longrightarrow \hat{b}_{\mathsf{ML}} = 1;$$

$$a[1]+a[2] < 0 \Longrightarrow \hat{b}_{\mathsf{ML}} = 0.$$

What sense can we make of this rule? It seems that we are collecting the two received signals and taking some sort of a "joint opinion". If the sum is positive, we decide that the bit must correspond to a positive signal (and vice-versa). This indeed coincides with our initial guess: the NN decision rule w.r.t. $y := y[1]+y[2]$.

A sufficient statistic

As above, we observed that the sum of the received signals

$$a[1]+a[2] \tag{5}$$

is sufficient to evaluate the ML rule for repetition coding. In other words, even though we received two different voltage levels, only their sum is relevant to making a decision based on

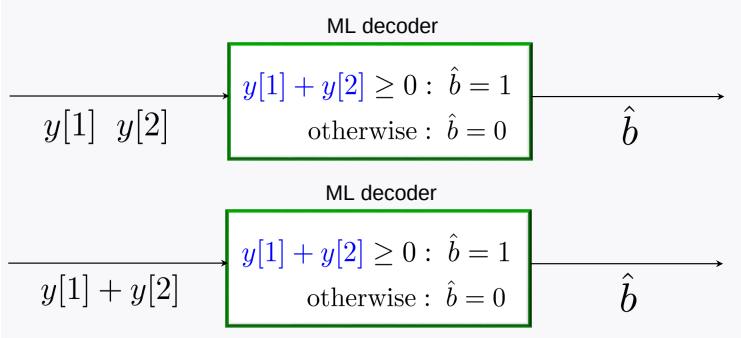


Figure 3: The sum is a sufficient statistic for deriving the ML rule.

the ML rule. Such a quantity is called a *sufficient statistic*; we say that the sum of the received signals is a sufficient statistic to derive the ML rule for repetition coding.

Error probability

Now what about the probability of error $\Pr(\mathcal{E})$? If you focus on the sufficient statistic, then the analysis of the error probability is straightforward

$$\underbrace{y[1] + y[2]}_{=:y} = \underbrace{x[1] + x[2]}_{=:x} + \underbrace{w[1] + w[2]}_{=:w}. \tag{6}$$

Here one key claim is that the summed noise w is also Gaussian; the mean is 0 and the variance is $2\sigma^2$:

$$\begin{aligned}
\mathbb{E}[w^2] &= \mathbb{E}[(w[1] + w[2])^2] \\
&= \mathbb{E}[w[1]^2] + \mathbb{E}[w[2]^2] + \mathbb{E}[w[1]w[2]] \\
&\overset{(a)}{=} \mathbb{E}[w[1]^2] + \mathbb{E}[w[2]^2] \\
&\overset{(b)}{=} 2\sigma^2
\end{aligned}$$

where (a) follows due to the independence and zero-mean properties of the AWGN; (b) is because $\mathbb{E}[w[1]^2] = \mathbb{E}[w[2]^2] = \sigma^2$. Yo may wonder why w is also Gaussian. You will have a chance to prove this in PS3.

Now what is $\Pr(\mathcal{E})$? If we can apply what we learned during the past lectures, then this calculation is trivial. In the case of sending one bit, $\Pr(\mathcal{E})$ always takes the following formula:

$$\Pr(\mathcal{E}) \ = Q\left(\frac{d}{2\sqrt{\mathsf{var[w]}}}\right) \tag{7}$$

where d denotes the minimum distance w.r.t. $x := x[1] + x[2]$:

$$d = 2\sqrt{E} - (-2\sqrt{E}) = 4\sqrt{E}. \tag{8}$$

So we get:

$$\Pr(\mathcal{E}) = Q\left(\frac{4\sqrt{E}}{2\sqrt{2\sigma^2}}\right) = Q\left(\sqrt{2 \cdot \mathsf{SNR}}\right). \tag{9}$$

Extension to $(B, n) = (1, n)$

We transmit the *same* voltage level at *each* time instant, for n consecutive time slots. So the received signals read:

$$y[m] = x[m] + w[m] = (2b-1)\sqrt{E} + w[m], \quad m = 1, \cdots, n. \tag{10}$$

Applying exactly the same arguments that we made in the previous sections, one can easily verify that the optimal decision rule is again the NN rule w.r.t. the sum:

$$\underbrace{y[1] + \cdots + y[n]}_{=:y} = \underbrace{x[1] + \cdots + x[n]}_{=:x} + \underbrace{w[1] + \cdots + w[n]}_{=:w}.$$

If you are not convinced, please check. Also the error probability is simply given by:

$$\begin{aligned} \Pr(\mathcal{E}) &= Q\left(\frac{d}{2\sqrt{\mathsf{var}[w]}}\right) \\ &\overset{(a)}{=} Q\left(\frac{2n\sqrt{E}}{2\sqrt{n\sigma^2}}\right) \\ &= Q\left(\sqrt{n \cdot \mathsf{SNR}}\right) \end{aligned}$$

where (a) is because the minimum distance is $d = n\sqrt{E} - (-n\sqrt{E}) = 2n\sqrt{E}$ and $\mathsf{var}[w] = n\,\mathrm{var}[w[1]] = n\sigma^2$.

Extension to arbitrary (B, n)

The idea for transmission is to apply PAM to the entire bit strong and then do repetition $x[1] = x[2] = \cdots = x[m]$. Again one can also prove that the sum $y[1] + y[2] + \cdots + y[n]$ is a sufficient statistic to derive the ML rule.

$$\underbrace{y[1] + \cdots + y[n]}_{=:y} = \underbrace{x[1] + \cdots + x[n]}_{=:x} + \underbrace{w[1] + \cdots + w[n]}_{=:w}.$$

Now x is a 2^B-PAM signal; see Fig. 4. So the minimum distance d reads:

$$d = \frac{2n\sqrt{E}}{2^B - 1}. \tag{11}$$

The summed noise $w \sim \mathcal{N}(0, n\sigma^2)$.

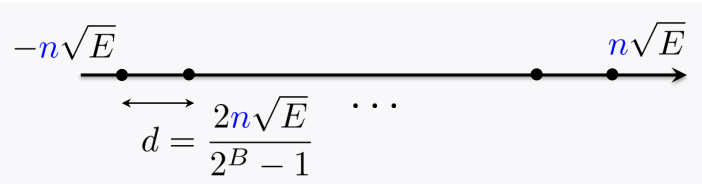


Figure 4: A 2^B-PAM signal w.r.t. $x = x[1] + x[2] + \cdots + x[n]$.

Again the optimal decision rule is the NN rule w.r.t. the sum, and the corresponding error probability is:

$$\begin{aligned}\Pr(\mathcal{E}) &= \left(2 - \frac{1}{2^{B-1}}\right) Q\left(\frac{d}{2\sqrt{\text{ noise var}}}\right) \\ &= \left(2 - \frac{1}{2^{B-1}}\right) Q\left(\frac{2n\sqrt{E}}{2(2^B-1)\sqrt{n\sigma^2}}\right) \\ &= \left(2 - \frac{1}{2^{B-1}}\right) Q\left(\frac{\sqrt{n\mathsf{SNR}}}{2^B-1}\right).\end{aligned} \tag{12}$$

Can we make $\Pr(\mathcal{E}) \to 0$?

Now let us check if we can make the error probability small enough with the repetition coding scheme. From (12), we can see that it is possible. For example, let us set:

$$n = B \cdot 2^{2B}. \tag{13}$$

Plugging this into (12), we get:

$$\Pr(\mathcal{E}) \approx 2Q\left(\sqrt{B \cdot \mathsf{SNR}}\right).$$

So by increasing B, we can make $\Pr(\mathcal{E})$ arbitrarily close to zero.

Any other issue?

But there is an issue in the repetition coding scheme. The issue is w.r.t. transmission efficiency, often quantified as data rate:

$$R := \frac{\text{total \# of transmission bits}}{\text{total \# of time slots}} = \frac{B}{n} \quad \text{(data rate)}.$$

In the above example $n = B \cdot 2^{2B}$, we see that the data rate tends to zero as B increases:

$$R = \frac{B}{n} = \frac{1}{2^{2B}} \to 0 \quad \text{as } B \to \infty.$$

Obviously this is not what we want.

Look ahead

A natural question that arises is then: Is there any smarter communication scheme that yields $R > 0$ while ensuring $\Pr(\mathcal{E}) \to 0$? In the next lecture, we will answer this question.

Changho Suh (chsuh@kaist.ac.kr)

Lecture 9: Capacity of the AWGN Channel

Recap

Last time, we investigated repetition coding. In an effort to check whether the scheme enables reliable communication (meaning $\Pr(\mathcal{E}) \approx 0$) even for a typical range of SNR, we analysed the probability of error:

$$\Pr(\mathcal{E}) = \left(2 - \frac{1}{2^{B-1}}\right) Q\left(\frac{\sqrt{n\mathsf{SNR}}}{2^B - 1}\right) \tag{1}$$

where B indicates the total number of transmission bits, and n denotes the total number of time slots used. From this, we found that repetition coding indeed enables reliable communication. Recall an example in which we choose the number of time slots as $n = B \cdot 2^{2B}$. In this example, we see that

$$\Pr(\mathcal{E}) \approx 2Q\left(\sqrt{B \cdot \mathsf{SNR}}\right) \to 0 \qquad \text{as } B \to \infty.$$

While repetition coding yields a good performance in reliability, we found it comes at a cost in transmission efficiency, which can be quantified as data rate $R := \frac{B}{n}$ (bits/time). In the above example, the increase in B that yields an arbitrarily small $\Pr(\mathcal{E})$ makes also R approach 0. This is obviously disappointing. So one natural question that one can ask in this context is:

> Is there a scheme that yields both $R > 0$ and reliable communication $\Pr(\mathcal{E}) \to 0$?

At the end of the last lecture, I claimed that the answer is yes.

Today's lecture

Today I am going to provide details that support my claim. Specifically what I will do are three folded. First I will tell you a story regarding the question. Actually there was an initial reaction and a follow-up conjecture on the question that many communication engineers had in the early 20th century. The conjecture was: whenever we want $\Pr(\mathcal{E}) \to 0$, we must have $R \to 0$. Here I will tell why people believed so. Next I will tell you a story about a guy who answered the question positively and how it was addressed. Actually the guy is the one that I mentioned in Lecture 1, the Father of Information Theory: Claude Shannon. Here I will tell you details on how he addressed the question. Interestingly, Shannon could answer the question without developing explicit coding schemes. But later people came up explicit coding schemes that yield both $R > 0$ and reliable communication. Lastly I will tell you two major such efforts.

A belief in the early 20th century

In the early 20th century, most people thought that the only way to achieve reliable communication over the *noisy* AWGN channel (i.e., to make the error probability as small as desired) was to add redundancies *infinitely many* (like repetition coding). Adding an infinite amount of redundancies leads to a negligible data rate. So the common belief at the time was that: it is impossible to achieve $\Pr(\mathcal{E}) \to 0$ as long as the date rate R is *strictly positive*.

Claude Shannon's finding

In contrast to such intuition, however, Shannon proved that this belief is incorrect. Actually he could do so in the process of developing a mathematical theory of communication that I mentioned earlier: *information theory*. Specializing the information theory into the AWGN channel of our current interest, he could show that intelligent coding schemes beyond repetition coding can yield both $R > 0$ and reliable communication:

$$\Pr(\mathcal{E}) \to 0 \quad \text{for } R > 0. \tag{2}$$

Shannon's efforts

Let us now explain how he could come up with the interesting result reflected in (2). First of all, he developed a non-trivial transmission scheme which is obviously different from repetition coding. Here for simplicity, I will not elaborate on the scheme. He then came up with a receiver strategy, which turns out to be not the optimal decision rule yet pretty close to the optimal rule. You may wonder why he considered such a non-optimal yet close-to-optimal decision rule. There was a highly non-trivial reason behind his choice. Explaining the reason is not that simple, so I will omit details - I do not want to distract you much; nonetheless, if you are interested in, you may want to take an information theory course: EE326 (an undergraduate level) or EE623 (a graduate level). Next he tried to derive $\Pr(\mathcal{E})$ in an effort to check if the probability of error can be made arbitrarily close to zero even when $R > 0$. Unfortunately, he failed to do so. Then, how could he prove (2) without deriving $\Pr(\mathcal{E})$? This is where one can see Shannon's genius.

A smart upper-bound technique

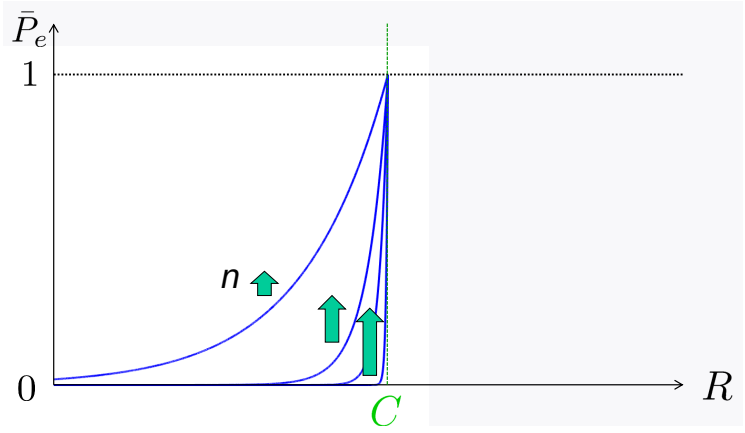


Figure 1: Upper bounds $\bar{P}_e$ on error probability for various n.

Instead he was able to derive an *upper bound* on the error probability, say $\bar{P}_e$. This inspired him to come up with the following very smart trick: if an upper bound tends to 0, then the *exact* error probability also goes to 0. Why? Because the probability cannot go below 0. The trick is based on a sort of sandwich argument: $0 \le \Pr(\mathcal{E}) \le \bar{P}_e \to 0$ implies that $\Pr(\mathcal{E}) \to 0$. Here the key was that Shannon was able to derive a tight-enough upper bound so that he can apply such smart trick. Here is the upperbound formula that Shannon derived:

$$\Pr(\mathcal{E}) \le \exp\left(n(R - C)\right) =: \bar{P}_e \tag{3}$$

where n denotes the total number of time slots used and C indicates some strictly positive value that I will detail soon. From this, we see that as long as $R < C$, we can make $\bar{P}_e$ arbitrarily

close to 0 as $n \to \infty$, therefore making the exact error probability $\Pr(\mathcal{E})$ arbitrarily small as well. Fig. 1 shows how the upper bound varies depending on a choice of n. Notice that increasing n, the more sharply the upper bound drops to 0 around at $R = C - \epsilon$. This is how he proved the main claim (2) of our interest.

Law governed by the Nature

In fact, this was not the end of the story. Shannon wanted to go beyond this. He was a sort of *scientist* who pursues to understand the *law governed by the Nature*. He actually believed that there must be a *fundamental limit* on the data rate above which reliable communication is not guaranteed fundamentally (i.e., $\Pr(\mathcal{E})$ cannot be made arbitrarily close to 0 no matter what and whatsoever). It turned out this is indeed the case. Specifically what he proved was that: if $R > C$, then there is no way to make $\Pr(\mathcal{E}) \to 0$, i.e., it is fundamentally impossible to enable reliable communication whenever the data rate exceeds the quantity C.

To prove this, he came up with another very smart trick: if a lower bound of the error probability does not go to 0, then the exact error probability does not go to 0 either. This is because: $\Pr(\mathcal{E}) \geq \underline{P_e} \not\to 0$ implies that $\Pr(\mathcal{E}) \not\to 0$. Shannon was able to come up with a tight-enough lower bound so that he can apply the smart trick. Here is the lower bound formula that Shannon developed:

$$\Pr(\mathcal{E}) \geq 1 - \frac{C}{R} - \frac{1}{nR} \qquad \forall n. \tag{4}$$

Here the bound holds for all n. So we must have:

$$\begin{aligned} \Pr(\mathcal{E}) &\geq \max_{n \in \mathbb{N}} \left(1 - \frac{C}{R} - \frac{1}{nR}\right) \\ &= 1 - \frac{C}{R} =: \underline{P_e} \end{aligned} \tag{5}$$

where the equality is achieved when $n \to \infty$. From this, we see that if $R > C$, $\underline{P_e} > 0$ and therefore $\Pr(\mathcal{E}) > 0$, meaning it is impossible to enable reliable communication. Fig. 2 illustrates such a lower bound: the lower bound is strictly above 0 for any $R > C$.

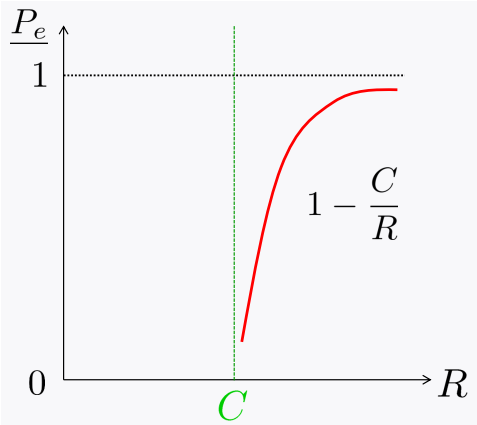


Figure 2: A lower bound $\underline{P_e}$ on error probability.

Capacity of the AWGN Channel

The upper (3) and lower (5) bounds imply that the quantity C is a sort of *fundamental* quantity which delineates the sharp threshold on the data rate below which reliable communication is possible and above which reliable communication is impossible no matter what. See Fig. 3. There

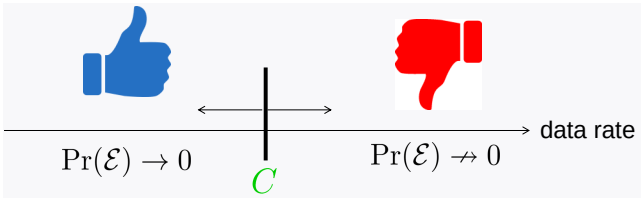


Figure 3: Error probability as a function of R.

is a terminology which indicates such maximum data rate that enables reliable communication. It is called the *channel capacity* or *Shannon capacity*. You may wonder where the letter C comes from. Here C stands for *capacity*.

What is the explicit formula for the channel capacity C? Let me now specify what the capacity C is. Given a power constraint P, the capacity of the AWGN channel is shown to be

$$C = \frac{1}{2}\log_2\left(1 + \frac{P}{\sigma^2}\right) \quad \text{bits/time} \tag{6}$$

where σ^2 is the noise variance. The rigorous proof is out of the scope of this course. So we will not prove this here. Instead we will provide an intuition as to why this must be the capacity in Appendix. If you are interested in a detailed derivation, again you may want to take an information theory course (EE326 or EE623).

The engineering conclusion

Note that the capacity (6) depends only on $\frac{P}{\sigma^2}$, which is the average SNR. What does this remind you of? Yes, you saw such a similar relationship in Lecture 2 in the context of the simple channel example. Hence, we can arrive at the conclusion: the SNR is the sole factor that determines the number of transmission bits.

Moreover, we can see a fundamental relationship between the data rate R and SNR. Earlier we noted a relationship between the two: the required energy essentially quadrupled when we looked to transmit one extra bit. Here you can see that the essential relationship is unchanged. To see this clearly, let us do the following calculation. Suppose that $\tilde{\mathsf{SNR}}$ is a new SNR required to transmit an extra one bit on top of the C bits/time. We then have

$$\frac{1}{2}\log_2\left(1 + \tilde{\mathsf{SNR}}\right) = 1 + \frac{1}{2}\log_2\left(1 + \mathsf{SNR}\right) \tag{7}$$

$$1 + \tilde{\mathsf{SNR}} = 4(1 + \mathsf{SNR}) \tag{8}$$

$$\tilde{\mathsf{SNR}} \approx 4\mathsf{SNR}. \tag{9}$$

Note that the $\tilde{\mathsf{SNR}}$ is a roughly quadruple of the SNR.

Capacity-achieving codes?

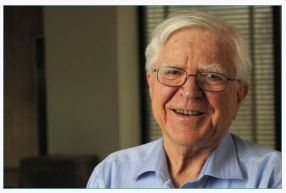
One may wonder how to achieve the capacity. Unfortunately, Shannon never constructed an *explicit* coding scheme that achieves the capacity. Instead he could show only the *existence* of such a capacity-achieving coding scheme. You may not understand what I am talking about

here. To fully understand this, you should actually grasp the proof of the upper bound (3) which I omitted. Let's just not worry about this, and simply trust me even if you are annoyed.

The fact that Shannon could only show the existence of an optimal code is one of the reasons that Shannon's work could not be applied to the design of real communication systems at that time. You may wonder what about now. Perhaps surprisingly to you, capacity-achieving explicit codes for the AWGN channel are still open.

From an engineering standpoint, however, we may be okay as long as we can achieve a performance close to the capacity. Hence, lots of works have thus far been done along the following research direction: developing *capacity-approaching* explicit coding schemes. Let us here introduce two such efforts.

Two major efforts towards capacity-approaching codes



Robert G. Gallager

Erdal Arikan

Figure 4: (Left): Robert G. Gallager; (Right): Erdal Arikan.

One effort was made by Robert G. Gallager. See the left picture in Fig. 4. He is actually my academic grandfather - an advisor of my PhD advisor, David Tse. He developed a code called *"Low Density Parity Check code"* (LDPC code for short) in 1960. Obviously it is an explicit code, meaning that it provides a detailed guideline as to how to design such a code. The performance of the code is shown to be remarkable. It has been shown that it approaches the capacity as the number n of time slots used tends to infinity, although it does not match the capacity exactly. You may wonder how close it approaches to the capacity. Here is an example that demonstrates the degrees of closeness.

Suppose we want the data rate $R = 0.5$ bits/time. Then, according to Shannon's capacity result (6), we must admit:

$$0.5 \leq \frac{1}{2}\log_2\left(1+\mathsf{SNR}\right). \tag{10}$$

This is then equivalent to saying that:

$$\mathsf{SNR} \geq 1 \ (0 \text{ dB}). \tag{11}$$

Here what it means by an LDPC code approaches the capacity is that: the LDPC code yields very small error probability $\Pr(\mathcal{E}) \approx 0$ with $\mathsf{SNR} = 1.0233$ (0.1 dB), which has only a 0.1 dB gap to the optimal SNR limit 0 dB. Here you may wonder how very small the error probability it. It is very small in the following sense: with $n = 8000$, $\Pr(\mathcal{E}) \approx 10^{-4}$ and we can achieve a smaller $\Pr(\mathcal{E})$ with a larger n.

Encouragingly (perhaps especially to you guys), he developed such a code during his "PhD study", meaning that you can also do that kind of great work in the near future if you purse

PhD study! Unfortunately, his work did not receive enough credits at that time. The main reason was that the LDPC code was of high implementation complexity considering the digital signal processing (DSP) technology of the day.[1] 30 years later, however, the code was revived, receiving significant attention. The reason was that the code *became* an *efficient* code – the DSP technology had been evolved, finally enabling the code to be implemented. Later the code was more appreciated with a more advanced DSP technology – it is now widely being employed in a variety of systems such as LTE, WiFi, DMB[2] and storage systems.

At the time when Gallager developed the LDPC code, he was not fully satisfied with his result though. The reason was that his code is not guaranteed to *exactly achieve* the capacity even in the limit of n. This motivated one of his PhD students, Erdal Arikan (see the right picture in Fig. 4), to work on developing the *capacity-achieving* code. It turned out Arikan could develop such a code, called *Polar code*. It is a first capacity-achieving code. Interestingly, he could develop the code in 2007, 30+ years later than his PhD day when the motivation began.[3] Due to its great performance and low-complexity nature, it is now being seriously considered for implementation in a variety of systems.

In fact, learning about LDPC and Polar codes requires lots of efforts and strong backgrounds on *random processes* which you are not familiar with. So in this course, we will not deal with the codes. If you are interested in, you may want to take a graduate level course on coding theory: EE621.

Summary of Part I

We have so far focused on the AWGN channel. We learned how to design transmission and reception strategies, and investigated the relationship between the data rate R, SNR and $\Pr(\mathcal{E})$. We also learned about the concept of channel capacity and some efforts towards achieving the channel capacity. This constitutes Part I of this course.

Outline of Part II

Actually the AWGN channel is far from realistic channels that arise in real communication systems. It turns out it only serves as a building block of many interested practical channels. So from next lectures, we will consider one of the practical channels: the wireline channel. We will first show that the channel can be modeled as a concatenation of LTI system and the AWGN block. We will then learn how to design transmission and reception strategies for the channel. These topics constitute Part II.

[1] Nonetheless, he became an MIT faculty right after graduation mainly due to that piece of work. It is fortunate that there are some serious & patient scholars who appreciate the *potential* of the great work which does not demonstrate any immediate impact upon the world.

[2] It is the name of the technology for a digital broadcast system, standing for Digital Multmedia Broadcasting.

[3] Remarkably he devoted most of his time to the development of the code, and finally succeeded. How dramatic the story is!

Appendix: Packing spheres

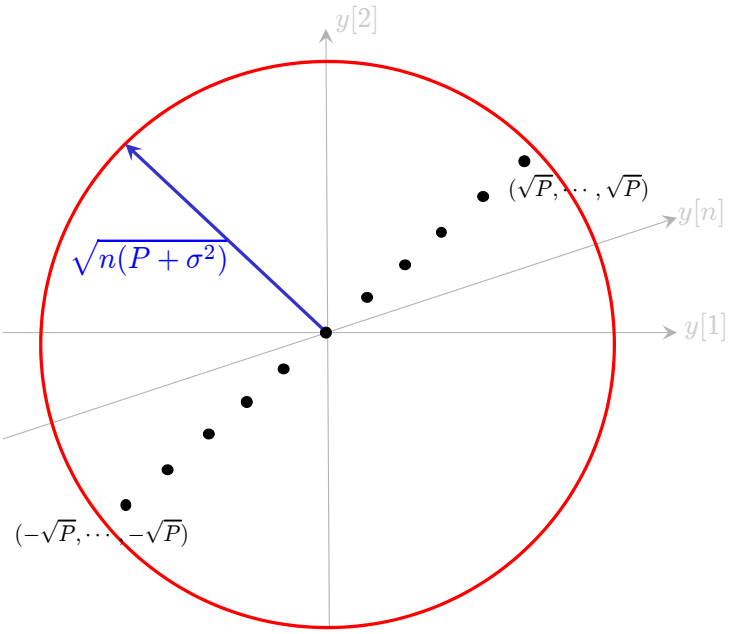


Figure 5: Repetition coding packs constellation points inefficiently in the n-dimensional signal space.

Geometrically, repetition coding puts all the constellation points in just one dimension. Fig. 5 provides an illustration. Here all the constellation points are on the same line. On the other hand, the signal space has a large number of dimensions n. This is a very inefficient way of packing the constellation points. To communicate more efficiently, the points should be spread over all the n dimensions.

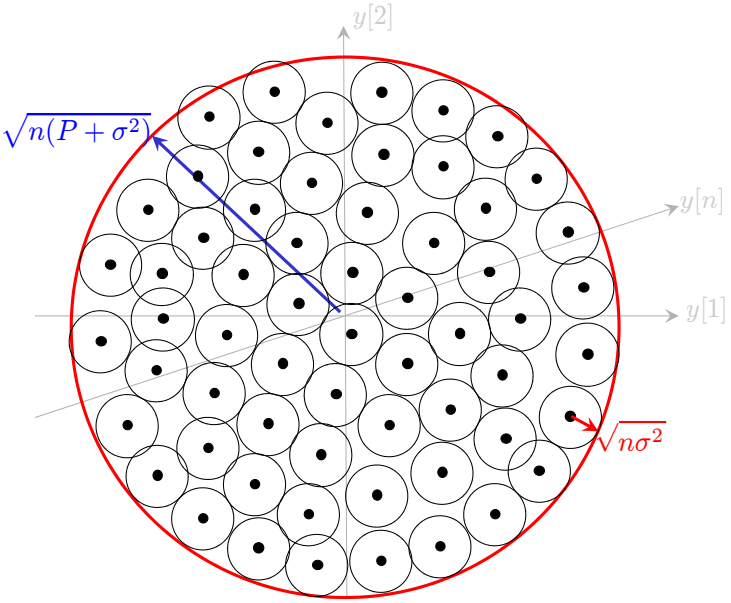


Figure 6: The number of noise spheres that can be packed into the received-signal-sphere yields the maximum number of constellation points that can be reliably distinguished.

We can get an estimate on the maximum number of constellation points that can be packed in for the given power constraint P, by appealing to the classic sphere-packing picture. See Fig. 6. By the law of large numbers[4], the n-dimensional received vector $\mathbf{y} = \mathbf{x} + \mathbf{w}$ will, with high probability, lie within a received-signal-sphere of radius $\sqrt{n(P+\sigma^2)}$. So without loss of generality we need only focus on what happens inside this sphere. On the other hand,

$$\frac{1}{n}\sum_{m=1}^{n} w[m]^2 \to \sigma^2 \tag{12}$$

as $n \to \infty$, by law of large numbers again. So, for large n, the received vector $\mathbf{y}$ lies, with high probability, near the surface of a *noise sphere* of radius $\sqrt{n\sigma^2}$ around the transmitted signal point. Reliable communication occurs as long as the noise spheres around the constellation points do not overlap. The maximum number of constellation points that can be packed with non-overlapping noise spheres is the ratio of the volume of the received-signal-sphere to the volume of a noise sphere:[5]

$$\frac{\left(\sqrt{n(P+\sigma^2)}\right)^n}{\left(\sqrt{n\sigma^2}\right)^n}. \tag{13}$$

This implies that the maximum number of bits per time slot that can be reliably communicated is

$$\frac{1}{n}\log_2\left[\frac{\left(\sqrt{n(P+\sigma^2)}\right)^n}{\left(\sqrt{n\sigma^2}\right)^n}\right] = \frac{1}{2}\log_2\left(1+\frac{P}{\sigma^2}\right). \tag{14}$$

This is indeed the capacity of the AWGN channel. The argument might sound very heuristic. If you are interested in a rigorous proof, you may want to take EE326 or EE623.

[4]To learn about the law of large numbers, see BT.

[5]The volume of an n-dimensional sphere of radius r is proportional to r^n.

April 22, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 10: Waveform shaping (1/2)

Recap

So far we have focused on the AWGN channel. Under the AGWN channel, we have studied several transmission/reception schemes and investigated corresponding tradeoff relationships between SNR, data rate and error probability. However, as I mentioned at the end of the last lecture, the AWGN channel is far from reality. It can serve only as a building block of practical channels.

In fact, the impracticality is related to the concept of time slots that we introduced earlier yet without explicitly specifying what the time slots mean in reality. Here not specifying the physical meaning of the time slots mean that we have considered only *discrete-time* signals which look like lollypops as illustrated in Fig. 1 (denoted by $x[m]$'s).

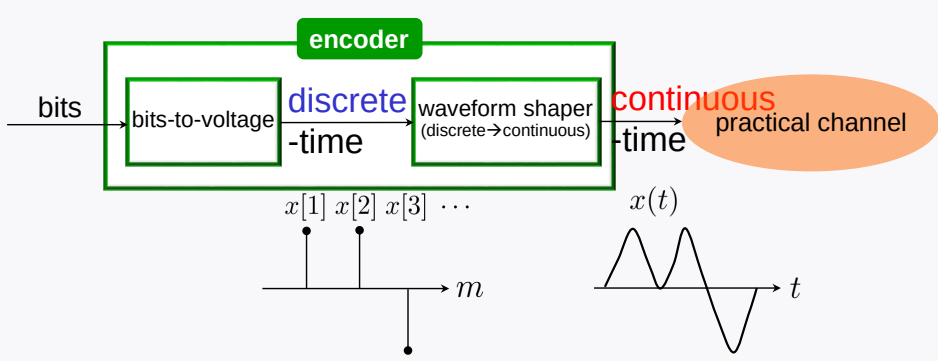


Figure 1: The two-stage mapping in the encoder: (i) bits-to-voltages mapping; (ii) voltages-to-waveforms mapping.

Here the issue is that the discrete-time signals are not what we sent in reality. What we send out are actually electromagnetic waveforms which convey information about the discrete-time signals. See Fig. 1 for an example of such waveform, denoted by $x(t)$. Obviously these electromagnetic waveforms need to be specified in a *continuous-time* domain. So we have to somehow convert the discrete-time signals to continuous-time waveforms.

Today's lecture

Today we will study how to do such a conversion. Specifically what we are going to do are three folded. First we will try to identify design criteria that the converter should satisfy. It turns out that one of the design criteria that I will mention soon comes from a key property of practical channels that we will focus on. So in the second part, we will investigate the key property and then figure out the corresponding design criterion accordingly. Lastly we will phrase the design criterion explicitly using a mathematical language. Due to the interest of time, next time, we will embark on the design of a converter that respects the to-be-identified design criteria.

The waveform shaping problem

Fig. 1 illustrates an encoder structure that practical communication systems have to take. As mentioned earlier, the encoder should consist of two key mappings. The first is mapping bits to voltages – this is the mapping that we have studied in Part I. Here our focus is on the second mapping: mapping voltages to waveforms that we can transmit across a practical channel. There is a commonly used terminology that indicates the second mapper. That is, *waveform shaper*. This naming comes from the fact that the output signal should be *waveforms*.

Design criteria for waveform shaper

There are two design criteria that the waveform shaper should satisfy. The first is an obvious one: the input $x[m]$ to the waveform shaper must have *one-to-one relationship* with the output $x(t)$: $x(t)$ should contain exactly the same information as the discrete-time signals $x[m]$ (we need no loss of information). As mentioned in the beginning, the second criterion comes from a property of practical channels. So let us first investigate what the property is.

Properties of practical channels

One of the practical channels that we will deal with mostly is: the *wireline channel*. It is also called the telephone channel, as it has been a representative wireline channel in the communication history. It turns out the wireline channel of our main interest induces restriction on the waveform $x(t)$. Actually the restriction comes even when there is no additive noise, say $w(t)$, in the system. So for simplicity, let us assume for the time being that there is no additive noise.

The wireline channel is very simple. It is nothing but a copper wire. Why copper? Why not gold? Because the copper is sort of the cheapest conductor that enables the flow of electrical signals. With terminologies in the field of *electrical circuits*, the copper wire can be interpreted as a collection of three key components: (i) a resistance R; (ii) inductance L; (iii) capacitance C. See Fig. 2. This interpretation together with what we learned form EE201 (Circuit Theory)

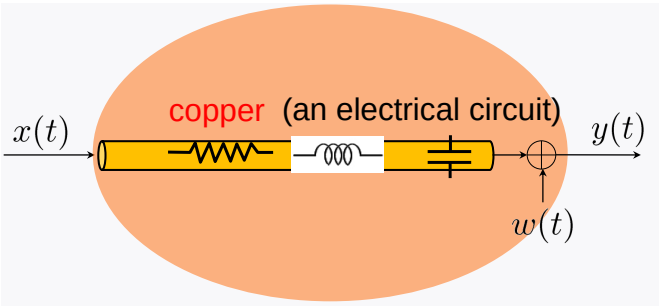


Figure 2: An electrical circuit based interpretation of the wireline channel is: the channel consists of three components: (i) resistance R; (ii) inductance L; (iii) capacitance C.

and/or EE202 (Signals & Systems) enables us to figure out one key property that the wireless channel has. That is, *linearity*. Remember one important property that we learned from the courses: any system that contains (R, L, C) elements only is *linear*. If you don't remember, that is fine too. Actually you don't need to understand why such RLC-based system is linear. It suffices to simply adopt that is the case.

Now what is the mathematical definition of a linear system? We say that a system is linear if any linear combination of two inputs yields exactly the same linear combination of the corresponding outputs. Precisely, given input-output pairs $(x_1(t), y_1(t))$ and $(x_2(t), y_2(t))$, a linear system

respects:

$$c_1x_1(t) + c_2x_2(t) \longrightarrow c_1y_1(t) + c_2y_2(t) \tag{1}$$

where c_1 and c_2 are arbitrary real values.

There is another property that the wireline channel has. The property is related to the fact that (R,L,C) parameters depend on environmental conditions, e.g., distance between transmitter and receiver, and temperature of the wire. Here the conditions can change over time of course, so the parameters may vary. But a key observation here is that the time scale of such change is pretty large relative to the communication time scale. Hence, it is fairly reasonable to assume that the system does not vary over time, in other words, *time invariant*. This forms the second property of the channel: *time invariance*. The formal definition of a time-invariant system is the following. A system is said to be time invariant if a time-shifted input $x(t - t_0)$ by t_0 yields the output $y(t - t_0)$ with exactly the same shift.

The above two properties imply that the wireless channel is a linear time invariant (LTI) system.

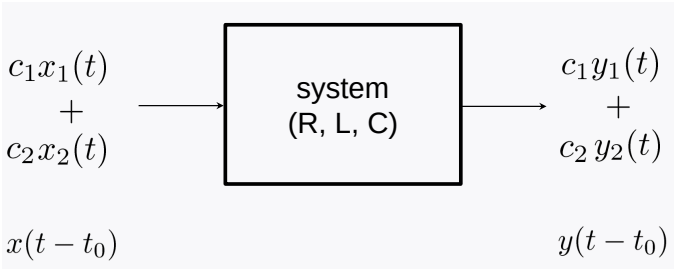


Figure 3: Key properties of an LTI system: (i) linearity; (ii) time invariance.

Impulse response $h(t)$ of an LTI system

We know that an LTI system can be described by an entity that you learned from EE202: *impulse response*. So the channel can be fully expressed by an impulse response, say $h(t)$. Do you remember what the mathematical definition of the impulse response is? The impulse response is defined as the output fed by the Dirac delta input $\delta(t)$, defined as a function satisfying the following two:

$$\delta(t) = \begin{cases} \infty, & t = 0; \\ 0, & t \neq 0, \end{cases} \qquad \int_{-\infty}^{+\infty} f(t)\delta(t)dt = f(0) \tag{2}$$

for a function $f(t)$. The definition allows us to express the relationship between input $x(t)$ and output $y(t)$ via $h(t)$. Actually you already know that the output is a *convolution* between the input and the impulse response:

$$y(t) = x(t) * h(t) := \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau. \tag{3}$$

For those who forgot about this and/or do not understand why that is the case, let me prove it. The proof is very simple. It is based on the LTI property and the definition of impulse response. First notice that an input $\delta(t-\tau)$ yields an output $h(t-\tau)$ due to the time invariance property. Now by the multiplicative property of linearity, an input $x(\tau)\delta(t-\tau)$ gives $x(\tau)h(t-\tau)$. Finally

the superposition property of linearity yields:

$$\int_{-\infty}^{\infty} x(\tau)\delta(t-\tau)d\tau \longrightarrow \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau. \tag{4}$$

Notice that

$$\int_{-\infty}^{\infty} x(\tau)\delta(t-\tau)d\tau = x(t) \tag{5}$$

due to the definition of the Dirac delta function. Hence, this proves the convolution relation (3) between $x(t)$ and $y(t)$. Also see Fig. 4.

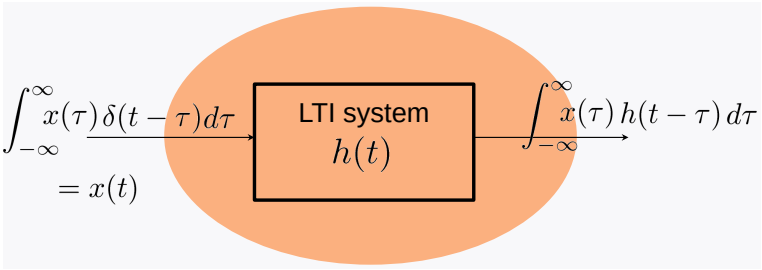


Figure 4: Impulse response of an LTI system and the relationship between $x(t)$ and $y(t)$.

Shape of $h(t)$ and its spectrum

You may now wonder how $h(t)$ looks like in practice. One typical example is shown in Fig. 5. Here we define a notion which indicates how much the impulse response is spread over time.

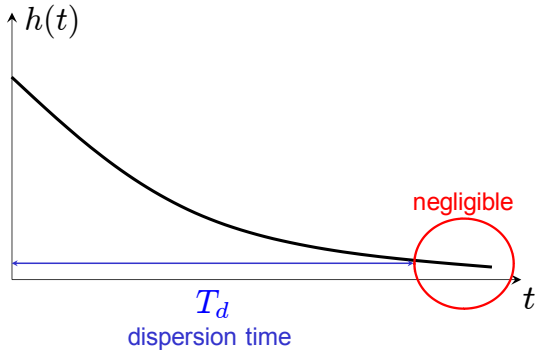


Figure 5: A typical shape of the impulse response of a wireline channel.

We adopt a very rough definition for this. Let T_d be the *dispersion* time of the channel which denotes the time duration during which the impulse response is not negligible. We may have a different value for T_d depending on the definition of the negligibility. Here for simplicity we will just take as a given parameter of the channel without going into more discussion.

The corresponding typical spectrum shape is shown in Fig. 6. One distinctive feature of the curve is that it is *symmetric*. This is because the magnitude of the spectrum of any *real-valued* signal is symmetric. The second feature is that the more spread in the time domain, the sharper the spectrum looks like, and vice versa. Similar to the dispersion time T_d, there is another notion in the frequency domain. That is, W_H, which indicates a range of frequencies in which

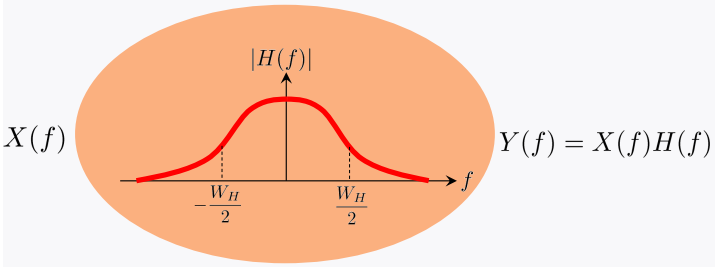


Figure 6: A typical spectrum shape of the impulse response of the wireline channel. HereW_H indicates a range of frequencies in which $|H(f)|$ is not negligible.

the spectrum magnitude $|H(f)|$ is not negligible. Then, what the second feature means in terms of T_d and W_H is that the smaller T_d, the larger W_H and vice versa.

Restriction on $x(t)$

It turns out the shape of the spectrum of the wireline channel as illustrated in Fig. 6 induces some restriction on $x(t)$. To see this, consider:

$$Y(f) = H(f)X(f) \tag{6}$$

where $X(f)$ and $Y(f)$ denote the Fourier transforms of $x(t)$ and $y(t)$, respectively. From this, we see that the high frequency components in $x(t)$ are significantly attenuated, e.g., where $|f|$ is beyond $\frac{W_H}{2}$. To avoid this distortion, we need restriction on $x(t)$. The restriction is that $x(t)$ should contain no component at high frequencies:

$$|X(f)| = 0 \qquad |f| \geq \frac{W_H}{2}. \tag{7}$$

We can rephrase this restriction with an important concept that often arises in communication systems. That is, *bandwidth*. The bandwidth, often denoted by W, is defined as a range of non-zero frequencies. A formal definition of the bandwidth relies on some mathematical notations. Let the upper frequency f_{U} be the largest frequency such that $|X(f)| = 0$ for $f \geq f_{\mathsf{U}}$. Let the lower frequency f_{L} be the smallest frequency such that $|X(f)| = 0$ for $f \leq f_{\mathsf{L}}$. Then, the bandwidth W is defined as the difference between the upper and lower frequencies: $W := f_{\mathsf{U}} - f_{\mathsf{L}}$. For example, in Fig. 7, $f_{\mathsf{U}} = \frac{W}{2}$, $f_{\mathsf{L}} = -\frac{W}{2}$ and thus the bandwidth is W.

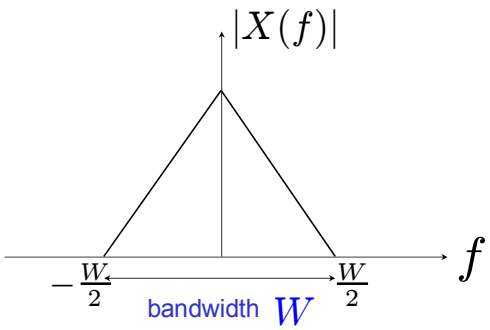


Figure 7: Spectrum of a continuous-time signal $x(t)$.

Now in terms of W and W_H, the restriction (7) can then be rephrased as:

$$W \leq W_H. \tag{8}$$

Look ahead

In this lecture, we have identified two design criteria that the waveform shaper should satisfy: (i) the shaper must be one-to-one mapping; (ii) $x(t)$ should be bandwidth-limited: $W \leq W_H$. Next time, we will develop "waveform shaper" that respects these criteria.

April 24, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 11: Waveform shaping (2/2)

Recap

Last time, we intended to design a block (followed by the bits-to-voltage mapper at the transmitter side), of which the role is to convert discrete signals $x[m]$ (lollypops) into waveforms $x(t)$ that can actually be transmitted over practical channels. See Fig. 1. We called the block

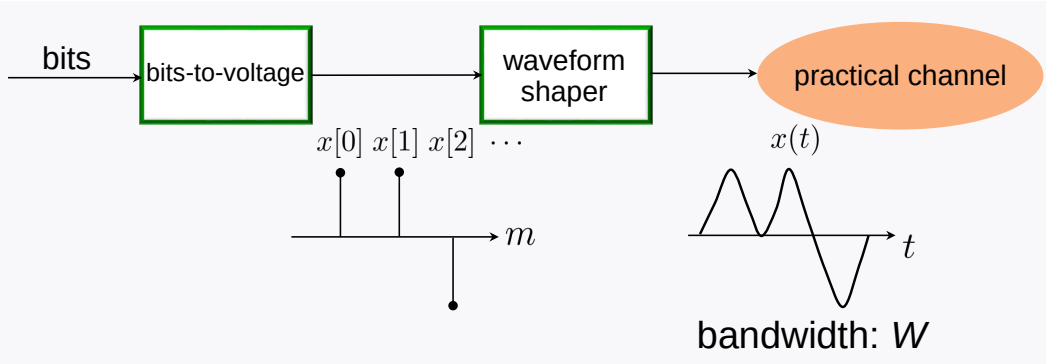


Figure 1: The two-stage mapping in the encoder: (i) bits-to-voltages mapping; (ii) voltages-to-waveforms mapping.

"waveform shaper", inspired by the naming of $x(t)$: waveforms.

As an initial effort, we figured out two design criteria that the waveform shaper should satisfy. The first is that it must be an one-to-one mapping function in order to ensure no loss of information. The second criterion comes from the key property of practical wireline channels of our main interest: there is a range of frequencies in which the spectrum $|H(f)|$ of channel impulse response $h(t)$ is not negligible: $|H(f)| \approx 0$ (or considered to be negligible) for $|f| \geq \frac{W_H}{2}$. So high frequency components in the interested signal $x(t)$ can be significantly distorted. To avoid this, we imposed the second constraint: the bandwidth W of $x(t)$ should be limited, i.e., $W \leq W_H$.

Today's lecture

Today we will design "waveform shaper" that meets the above two criteria. Specifically what we are going to do are three folded. First we will introduce a proper structure of the waveform shaper that turns out to ease construction. Next we will incorporate the two design criteria into the structure and then translate the criteria into mathematical expressions. Finally, by relying on Signals & Systems backgrounds (Fourier transform and series techniques), we will derive a precise mathematical relationship between $x[m]$ and $x(t)$, thus completing the design.

One-to-one mapping

Let us start by investigating the one-to-one mapping criterion. Here what the one-to-one mapping means is that we have to ensure reconstruction of $x(t)$ from $x[m]$ and vice versa. See Fig. 2. In fact, the other way around (obtaining $x[m]$ from $x(t)$) is called "sampling", and the sampling process is straightforward. As long as we specify the sampling period, say T (indicating the

Figure 2: Meaning of the one-to-one mapping criterion.

physical time duration between two consecutive samples), this process is obvious as the naming suggests. Here the one-to-one mapping criterion means that we have to ensure:

$$x[m] = x(mT) \qquad \forall m \in \{0, 1, \ldots, n-1\} \tag{1}$$

where n denotes the total number of time slots used. Here we start from time 0. This is sort of convention that many people adopt in the field, although some people prefer to use time 1 as a starting point.

As for the forward direction of our interest, there are actually infinitely many ways to connect discrete signals (lollypops), reflected in many green curves in Fig. 2. So you may wonder which way to take. It turns out the way to connect (often called the *interpolation* in the literature) is intimately related to a careful choice of the sampling period T and the bandwidth constraint $W \leq W_H$. Let us investigate the relationship from the next section on.

Structure of waveform shaper

In an effort to ease construction, we introduce a proper structure for the waveform shaper. Notice that the waveform shaper is sort of an annoying system that we are not familiar with. Why annoying? This is because the *signal types* are distinct across input $x[m]$ and output $x(t)$: $x[m]$ is discrete while $x(t)$ being continuous. In fact, from EE202 or other relevant courses, we have mainly studied sort of homogeneous systems where the signal types are the same. In order to close this gap, we introduce a two-stage architecture as illustrated in Fig. 3.

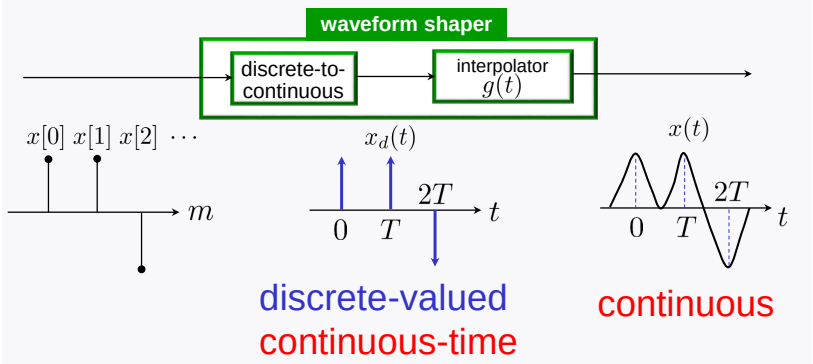


Figure 3: A two-stage architecture for waveform shaper: (i) time-domain converter (from discrete to continuous); (ii) interpolator $g(t)$.

First we have a so called *time-domain converter* from discrete to continuous. Here the output to the converter is defined in the *continuous time*, yet still preserving *discrete-valued* signals. What does it mean by having discrete-valued signals? It means that signals contain spiked

values (reflected in the Dirac delta function, $x[m]\delta(t-mT)$; also see the middle plot in Fig. 3) only at sampled time instants, while having nothing at other continuous times. So it is sort of an intermediate imaginary signal introduced only for conceptual simplicity.

The role of the second block is to generate *continuous-valued* signals from such intermediate signals. See the right plot in Fig. 3. The second block is obviously a standard homogeneous LTI system where the signal types are the same: input and output are all *continuous-time* signals. Hence, we can describe it simply with an impulse response, say $g(t)$. The conventional name of the block is *interpolator*, as its role suggests: interpolating discrete values to generate smoothly changing analog signals.

Now the question of interest is then: how to design $g(t)$? As I hinted earlier, the way to design $g(t)$ is related to a choice of the sampling period T and the bandwidth constraint $W \leq W_H$. Let us now dive into details on this.

Representation of continuous-time yet still discrete-valued signal $x_d(t)$

In fact, there is a useful and common way to represent such immediate signal $x_d(t)$. The way is to use the Dirac delta function $\delta(t)$, as I briefly mentioned while referring to the second plot in Fig. 3. I told you that $x_d(t)$ contains discrete values only at the sampled time instants, say $0, T, 2T$. The signal at time 0 corresponds to $x[0]$. But in the *continuous-time* domain, we represent the signal in a slightly different manner. We employ the Dirac delta function to indicate the signal as $x[0]\delta(t)$. For time 1, we express it as $x[1]\delta(t-T)$. So the compact formula of $x_d(t)$ reads:

$$x_d(t) = \sum_{m=0}^{n-1} x[m]\delta(t-mT). \tag{2}$$

Using the fact that $x[m] = 0$ for $m < 0$ and $m \geq n$, we can alternatively represent $x_d(t)$ as:

$$x_d(t) = \sum_{m=-\infty}^{\infty} x[m]\delta(t-mT). \tag{3}$$

For notational simplicity, we will use $\sum_m$ to indicate $\sum_{m=-\infty}^{\infty}$. You may wonder why we multiply by the Dirac delta function. This will be clearer when I will explain the design of $g(t)$. Please be patient until we get to the point.

Applying one of the criteria $x[m] = x(mT)$ to (3), $x_d(t)$ must read:

$$\begin{aligned} x_d(t) &= \sum_m x(mT)\delta(t-mT) \\ &\overset{(a)}{=} \sum_m x(t)\delta(t-mT) \\ &= x(t)\left(\sum_m \delta(t-mT)\right) \end{aligned} \tag{4}$$

where (a) comes from the fact that $\delta(t-mT) = 0$ when $t \neq mT$.

How to design the interpolator $g(t)$?

Now what is $g(t)$ such that $x(t) = x_d(t) * g(t)$? The formula (4) shows the relationship between $x_d(t)$ and $x(t)$. However, it is still very difficult to figure it out from the formula (4). It turns

out taking *Fourier transform*, we can gain insights into identifying $g(t)$. To see this, let us take Fourier transform on both sides in (4) to obtain:

$$X_d(f) = X(f) * \mathcal{F}\left(\sum_m \delta(t - mT)\right) \tag{5}$$

where we used the *duality* property of Fourier transform: $x(t)h(t) \leftrightarrow X(f) * H(f)$. Here $\mathcal{F}(\cdot)$ indicates the Fourier transform of $(\cdot)$.

There is a key observation which allows us to represent (5) in a more insightful formula. The key observation is that the insider of $\mathcal{F}(\cdot)$ in (5), $\sum_m \delta(t-mT)$, is a *periodic* signal with period T. So we can now use *Fourier series* to represent the periodic signal as:

$$\sum_m \delta(t - mT) = \sum_k a_k e^{j\frac{2\pi kt}{T}} \tag{6}$$

where a_k indicates the kth Fourier coefficient, expressed as:

$$\begin{aligned} a_k &= \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} \left(\sum_m \delta(t - mT)\right) e^{-j\frac{2\pi kt}{T}} dt \\ &= \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} \delta(t) e^{-j\frac{2\pi kt}{T}} dt \\ &= \frac{1}{T}. \end{aligned} \tag{7}$$

Here the second equality comes from the fact that $\sum_m \delta(t - mT) = \delta(t)$ for $-\frac{T}{2} \le t \le \frac{T}{2}$; and the last equality is because of the definition of the Dirac delta function. Plugging (6) (together with (7)) into (5), we then get:

$$\begin{aligned} X_d(f) &= X(f) * \mathcal{F}\left(\frac{1}{T}\sum_k e^{j\frac{2\pi kt}{T}}\right) \\ &\overset{(a)}{=} X(f) * \left(\frac{1}{T}\sum_k \delta\left(f - \frac{k}{T}\right)\right) \\ &\overset{(b)}{=} \frac{1}{T}\sum_k X\left(f - \frac{k}{T}\right) \end{aligned} \tag{8}$$

where (a) is due to $\mathcal{F}(e^{j2\pi f_0 t}) = \delta(f - f_0)$; and (b) is because $X(f) * \delta(f - \frac{k}{T}) = X(f - \frac{k}{T})$.

What is $G(f)$ that respects (8)?

Notice in (8) that $X_d(f)$ is a *periodic* signal with the period $\frac{1}{T}$. Here the shape of the periodic signal $X_d(f)$ depends on the relationship between the bandwidth W of $x(t)$ and the period $\frac{1}{T}$. Suppose we choose T such that the period $\frac{1}{T}$ exceeds the bandwidth W. Then, we can get a spectrum which looks like the one in Fig. 4. Observe in the low frequency regime $|f| \le \frac{W}{2}$ that we have $\frac{1}{T}X(f)$. Here the key point is that there is no overlap across many shifted copies because the period $\frac{1}{T}$ is greater than the bandwidth W. Hence, the interested signal $X(f)$ is sort of intact, although it is scaled by $\frac{1}{T}$.

On the other hand, if T is chosen such that $\frac{1}{T} < W$, then the interested signal $X(f)$ is now compromised by the neighboring shifted copies. See Fig. 5.

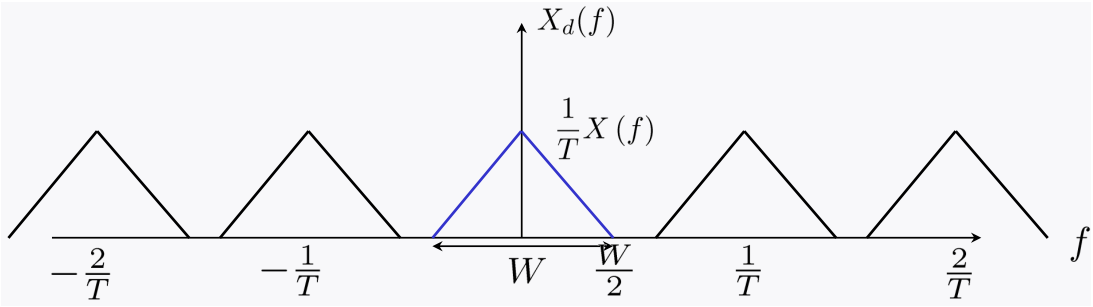


Figure 4: The spectrum of $x_d(t)$ when $\frac{1}{T} \geq W$: $X_d(f) = \frac{1}{T}\sum_k X\left(f - \frac{k}{T}\right)$.

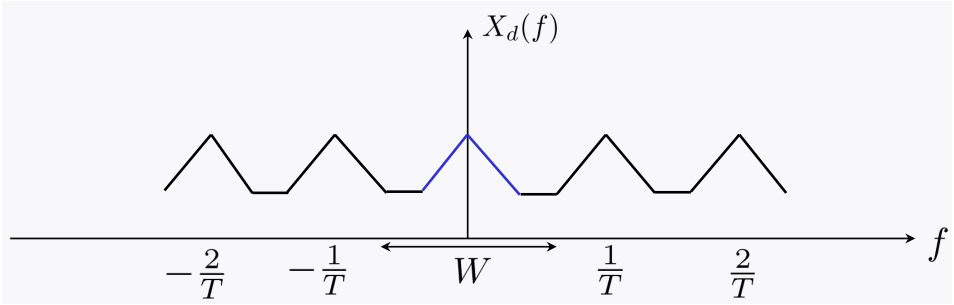


Figure 5: The spectrum of $x_d(t)$ when $\frac{1}{T} < W$: $X_d(f) = \frac{1}{T}\sum_k X\left(f - \frac{k}{T}\right)$.

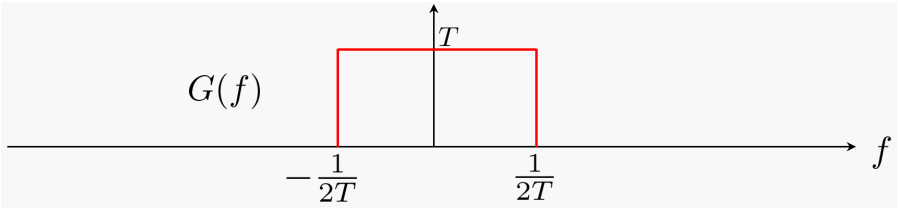


Figure 6: An example of the spectrum $G(f)$ of an interpolator that satisfies the relationship (4) between $x_d(t)$ and $x(t)$.

So this advises us to choose T such that $\frac{1}{T} \geq W$. With the choice, there is hope to extract the interested signal $X(f)$ from $X_d(f)$. Indeed, by applying $G(f)$ with the shape as illustrated in Fig. 6, we can get the desired $X(f)$.

The impulse response $g(t)$ that corresponds to $G(f)$ in Fig. 6

Now the question is: what is $g(t)$ that corresponds to $G(f)$ in Fig. 6? To figure this out, let us use some notation used in the Signals-&-Systems literature:

$$\mathsf{rect}(x) = \begin{cases} 1, & -\frac{1}{2} \leq x \leq \frac{1}{2}; \\ 0, & \text{otherwise.} \end{cases}$$

Using this rect function, we can represent $G(f)$ in Fig. 6 as:

$$G(f) = T\mathsf{rect}(fT). \tag{9}$$

We learned from EE202 that the rect function has the Fourier transform relationship with the sinc function:

$$T\mathsf{rect}(fT) \leftrightarrow \mathsf{sinc}\left(\frac{t}{T}\right) \tag{10}$$

where $\mathsf{sinc}(x) := \frac{\sin(\pi x)}{\pi x}$. Hence, we get:

$$g(t) = \mathsf{sinc}\left(\frac{t}{T}\right). \tag{11}$$

Relationship between $x[m]$ and $x(t)$

Recall in the introduced structure of waveform shaper that:

$$x(t) = x_d(t) * g(t). \tag{12}$$

Plugging (11) into the above with the choice of T such that $\frac{1}{T} \geq W$, we then get:

$$\begin{aligned} x(t) &= x_d(t) * \mathsf{sinc}\left(\frac{t}{T}\right) \\ &\overset{(a)}{=} \left(\sum_m x[m]\delta(t - mT)\right) * \mathsf{sinc}\left(\frac{t}{T}\right) \\ &\overset{(b)}{=} \sum_m x[m]\mathsf{sinc}\left(\frac{t - mT}{T}\right) \\ &= \sum_m x[m]\mathsf{sinc}\left(\frac{t}{T} - m\right) \end{aligned} \tag{13}$$

where (a) is due to (2); and (b) comes from the fact that $y(t) * \delta(t - t_0) = y(t - t_0)$ for any $y(t)$ and t_0. Finally, recall the bandwidth constraint: $W \leq W_H$. This constraint can be easily satisfied with the following choice of T:

$$W \leq \frac{1}{T} \leq W_H. \tag{14}$$

Look ahead

So far we have focused only on the transmitter side. So a natural follow-up question is: what about for the receiver side? Next time, we will develop a corresponding receiver architecture.

May 1, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 12: Optimal receiver structure

Recap

During the past two lectures, we studied how to design the waveform shaper, wherein the role is to convert discrete-time signals (lollypops) into continuous-time signals (waveforms). The reason that we had to introduce such converter was that the actual signals (that would be transmitted over practical channels) should be continuous-time analog signals. See Fig. 1.

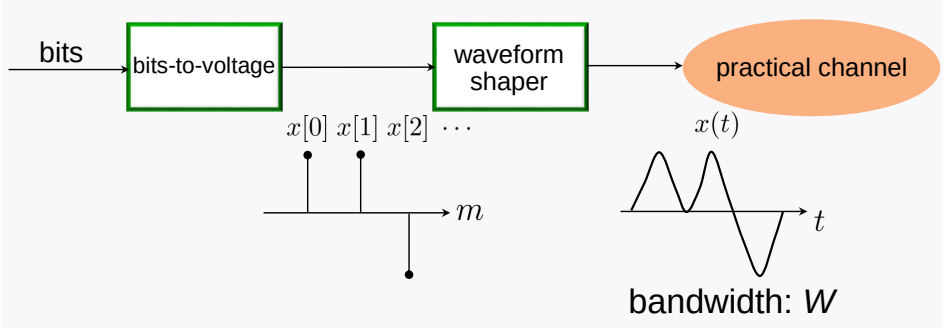


Figure 1: Encoder: (i) bits-to-voltages mapper; (ii) waveform shaper.

For the design, we considered two criteria. The first is that it should be an one-to-one mapping function, i.e., the output $x(t)$ of the waveform shaper should contain the same information as the discrete-time signal $x[m]$ (having no information loss in the process). The second criterion is that $x(t)$ should be bandwidth-limited. This is because in reality, practical channels may induce signal distortion. We could see this clearly in the frequency domain: the frequency spectrum of the channel's impulse response is not flat, so without the bandwidth constraint, the transmitted signal may be significantly distorted. To avoid such distortion, we impose the bandwidth constraint: the bandwidth of the transmitted signal, say W, does not exceed the range W_H of frequency bands in which channel's spectrum is significant (at least not negligible).

Keeping in mind the two design criteria, we could design the shaper. Inspired by the Nyquist rate above which one can avoid signal overlap that occurs in the frequency domain (called *aliasing*), we set the sampling period T such that the sampling rate $\frac{1}{T}$ is above the signal bandwidth W. Relying to the picture in the frequency domain, we could then come up with an explicit way to interpolate the discrete signals to yield continuous signals:

$$x(t) = \sum_m x[m]\mathsf{sinc}\left(\frac{t}{T} - m\right) \tag{1}$$

where $\mathsf{sinc}(x) := \frac{\sin \pi x}{\pi x}$.

This is what we have done so far. Notice that this design only concerns the transmitter side. So a natural follow-up question is: What about for the receiver side? What is a corresponding receiver architecture?

Today's lecture

Today we will explore details on this question. Specifically what we are going to do are four folded. First we will come up with a corresponding receiver architecture in a very simple context: the perfect channel setting in which the channel's impulse response $h(t) = \delta(t)$ and there is no additive noise $w(t) = 0$. It turns out the receiver architecture in the simple setting includes exactly the reverse operation (relative to the waveform shaper). That is, *sampling* which yields $y[m]$ from $y(t)$. We will then argue the optimality of the *sampler-based* receiver architecture, still under the AWGN channel and even under practical channels including the wireline channel of interest. Next we will rely on the structure to derive an equivalent discrete-time channel model which directly relates $x[m]$ to $y[m]$. Lastly we will discuss how to design the optimal receiver in the context of discrete-time channel model.

A corresponding receiver architecture in the perfect channel

Recall that the wireline channel of our main interest can be modeled as a concatenation of an LTI system and the AWGN channel. Hence, one can write down the channel output $y(t)$ as:

$$y(t) = h(t) * x(t) + w(t) \tag{2}$$

where $h(t)$ denotes an impulse response of the LTI system and $w(t)$ indicates the AWGN noise. The received waveform $y(t)$ is then fed into the optimal receiver which intends to decode information bits b. See Fig. 2. The question of interest is: What is the optimal receiver structure

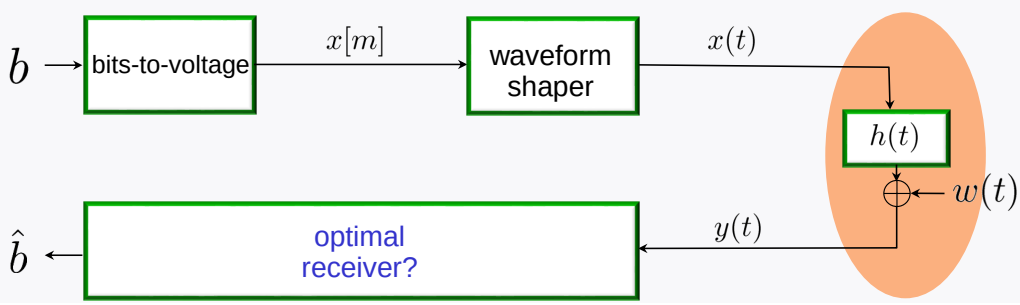


Figure 2: A block diagram of a two-stage communication system.

corresponding to the transmitter structure developed last time?

Observe two complications here in the received waveform (2). One is that these are continuous-time signals. We never learned about the optimal decision rule concerning continuous-time signals. The second complication comes from the *convolution* operation. To gain insights, let us first consider a simple perfect channel setting in which $h(t) = \delta(t)$ and $w(t) = 0$:

$$y(t) = x(t). \tag{3}$$

Remember in the transmitter side that we took discrete-time signals $x[m]$ and interpolated them to obtain waveforms $x(t)$ that contain all the information in $x[m]$. The ultimate goal at the receiver is to decode the transmitted bits, so we can readily see that the receiver strategy would be to do exactly the *reverse* of what the transmitter is doing – *sample* the received waveforms $y(t)$ at $t = mT$, and obtain discrete-time signals $y[m]$. Under this perfect channel, we then get:

$$y[m] = y(mT) = x(mT) = x[m]. \tag{4}$$

Optimality of the sampler-based receiver

Now what if we have an AWGN noise? Assuming to still employ the sampler, for the AWGN channel, $y[m]$ would be the sum of $x[m]$ and an additive noise:

$$y[m] = x[m] + w[m] \tag{5}$$

where $w[m]$ denotes a sample version of $w(t)$. Since $w(t)$ is an AWGN, $w[m] := w(mT)$ is also i.i.d. $\sim \mathcal{N}(0, \sigma^2)$. In the presence of a noise, all we need to do for decoding the information bits from $y[m]$ is to apply the ML decision rule. This is something that we have been doing in Part I. Given $y[m]$'s, make the best possible estimate $\hat{b}$. This leads to the initial guess of the optimal receiver architecture as illustrated in Fig. 3.

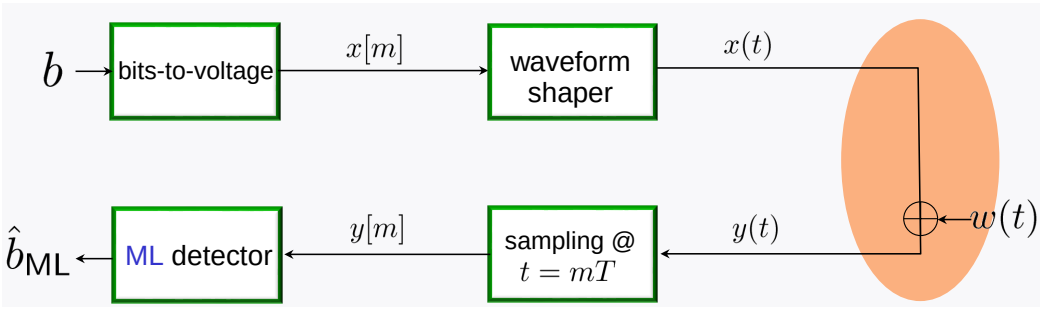


Figure 3: The sampler-based receiver architecture.

At this moment, this is nothing but a guess for the optimal receiver. Why still a guess? Remember in the perfect channel that we had only the sampler, and we saw that the sampler-based structure leads naturally to the optimal receiver. On the other hand, in the presence of a noise, it is not quite clear if we need the sampler in the first place. In fact, the optimal receiver should have the following structure in which we have the continuous-time waveform-level signal input. See Fig. 4.

Figure 4: The waveform-level optimal ML receiver.

Here $\hat{b}_{\mathsf{ML,wave}}$ denotes the output of the waveform-level optimal ML detector:

$$\hat{b}_{\mathsf{ML,wave}} := \arg\max_i f_{y(t)}(a(t)|b = i) \tag{6}$$

where $f_{y(t)}(a(t)|b = i)$ indicates the likelihood function w.r.t. the received waveform $y(t)$ given $b = i$. Now a natural question arises:

> Is the sampler-based ML solution, say $\hat{b}_{\mathsf{ML}}$, the same as the waveform-level (raw signal level) ML counterpart $\hat{b}_{\mathsf{ML,wave}}$?

It turns out that for typical channels including the AWGN channel as well as the wireline channel, $\hat{b}_{\mathsf{ML}}$ is indeed the same as $\hat{b}_{\mathsf{ML,wave}}$. This implies that we do not lose any information while passing through the sampling process. In fact, this is a *deep* result known as the *irrelevance*

theorem. A formal proof of this theorem is not that easy. So we will omit the proof here. If you are interested in the proof, you may want to take a graduate-level course on Random Processes. In this course, we will simply adopt the optimal receiver structure that features the sampler followed by the ML detector.

An equivalent discrete-time channel model

Now the only thing that is left is how to design such sampler-based ML detector in the context of the wireline channel. To this end, we first need to figure out relationship between input/output in the *discrete-time domain*: $x[m]$ and $y[m]$. Starting with the property of LTI systems, we get:

$$\begin{aligned} y(t) &= h(t) * x(t) + w(t) \\ &= \int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau + w(t) \\ &\overset{(a)}{=} \int_{0}^{\infty} h(\tau)x(t-\tau)d\tau + w(t) \\ &\overset{(b)}{=} \int_{0}^{\infty} h(\tau)\left(\sum_k x[k]\mathsf{sinc}\left(\frac{t-\tau}{T}-k\right)\right)d\tau + w(t) \\ &= \sum_k x[k]\left(\int_0^{\infty} h(\tau)\mathsf{sinc}\left(\frac{t-\tau}{T}-k\right)d\tau\right) + w(t) \end{aligned} \tag{7}$$

where (a) is because $h(\tau) = 0$ when $\tau < 0$ for a realistic wireline channel (see Fig. 5; you may wonder why $h(\tau) = 0$ for $\tau < 0$; this is because otherwise the output is a function of *future* inputs, which never occurs in reality); (b) follows from (1).

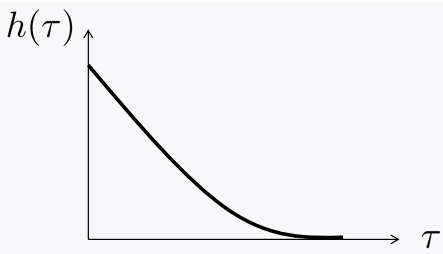


Figure 5: A typical shape of the wireline channel's impulse response.

Now by sampling $y(t)$ at $t = mT$, we get:

$$\begin{aligned} y[m] &:= y(mT) \\ &= \sum_k x[k] \underbrace{\int_0^{\infty} h(\tau)\mathsf{sinc}\left(m-k-\frac{\tau}{T}\right)d\tau}_{=:h[m-k]} + w[m] \\ &= \sum_k x[k]h[m-k] + w[m]. \end{aligned} \tag{8}$$

We recognize (8) as a *discrete-time convolution* and thus we have for the final received signal:

$$y[m] = h[m] * x[m] + w[m] \tag{9}$$

where $h[m]$ is a discrete-time impulse response that represents a *sampled* version of $h(t)$:

$$h[m] = \int_0^{\infty} h(\tau)\mathsf{sinc}\left(m-\frac{\tau}{T}\right)d\tau. \tag{10}$$

This gives us an equivalent discrete-time channel model as summarized in Fig. 6.

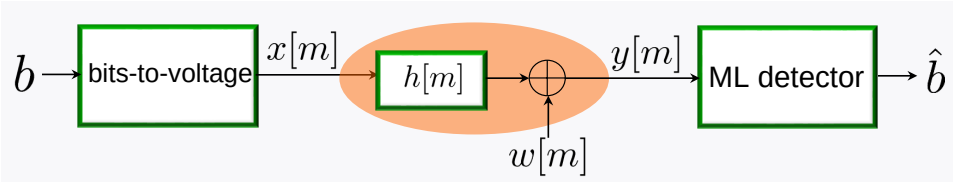


Figure 6: The equivalent discrete-time model for the wireline channel.

Remarks on $h[m]$

Remember our main goal: designing the optimal ML detector based on $y[m]$ in (9). Now how can we achieve this? It turns out the discrete-time convolution in (9) incurs a serious issue. Prior to discussing what that issue is, let me say a few about about $h[m]$. First of all, there is a terminology for this. To figure this out, notice that

$$\begin{aligned} y[m] &= x[m] * h[m] + w[m] \\ &\overset{(a)}{=} h[m] * x[m] + w[m] \\ &= \sum_k h[k]x[m-k] + w[m] \end{aligned} \tag{11}$$

where (a) is due to the commutative property of convolution. Here the kth channel impulse response $h[k]$ is called a *channel tap* or simply a *tap*. Why do we call tap? It is a very short name. Also this is because the channel impulse response looks like a sequence of faucets (taps). With the notation $h[k]$, the tap index k may be confused with the time index m. Hence, in order to differentiate the tap index from the time index, we denote $h[k]$ by h_k.

The second remark is about a range of k for non-zero h_k's. As mentioned earlier, the output cannot depend on future inputs, which led to $h(\tau) = 0$ for $\tau < 0$ (gain see Fig. 5). In the discrete-time domain, this is translated to:

$$h_k = 0, \quad k < 0. \tag{12}$$

To see this clearly, observe:

$$\begin{aligned} y[m] &= \sum_k h_k x[m-k] + w[m] \\ &= h_0 x[m] && \text{(current)} \\ &\quad + h_1 x[m-1] + h_2 x[m-2] + \cdots && \text{(past)} \\ &\quad + h_{-1} x[m+1] + h_{-2} x[m+2] + \cdots && \text{(future)} \\ &\quad + w[m]. \end{aligned} \tag{13}$$

In light of the current symbol, say $x[m]$, $(h_{-1}, h_{-2}, \dots,)$ are w.r.t. the *future* inputs $(x[m+1], x[m+2], \dots)$. Hence, $(h_{-1}, h_{-2}, \dots,)$ must be all zero.

Intersymbol interference (ISI)

Now let us get back to the issue. What is that issue? To see this, let us again observe (13):

$$\begin{aligned} y[m] = \; & h_0 x[m] && \text{(current)} \\ & + h_1 x[m-1] + h_2 x[m-2] + \cdots && \text{(past)} \\ & + w[m]. \end{aligned} \tag{14}$$

To see the issue concretely, let us consider a sequential communication scheme in which the mth information bit, say b_m, is mapped to $x[m]$. Here in light of the current symbol, say $x[m]$, the past symbols ($x[m-1]$, $x[m-2]$, $\dots$) play as *interference*. This phenomenon is called the *intersymbol interference*, ISI for short. This is the new phenomenon that we did not observe in Part I. It turns out the ISI incurs some complication in the ML decision rule.

Look ahead

Next we will study how to deal with ISI and then to develop the optimal ML decision rule.

May 13, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 13: Optimal receiver in ISI channels

Recap

Last time, we came up with an optimal receiver architecture that corresponds to the sampler-based transmitter which we developed during Lectures 11 & 12. It turns out the optimal receiver architecture includes the *sampler* in front that yields $y[m] := y(mT)$ from the received waveform $y(t)$. With the proper and smart definition of $h[m]$ as below:

$$h[m] := \int_0^\infty h(\tau)\mathsf{sinc}\left(m - \frac{\tau}{T}\right) d\tau, \tag{1}$$

we could then establish an equivalent discrete-time channel model that relates $x[m]$ directly to $y[m]$:

$$y[m] = h[m] * x[m] + w[m] \tag{2}$$

where $w[m]$'s are i.i.d. $\sim \mathcal{N}(0, \sigma^2)$ and $h(t)$ indicates an impulse response of the wireline channel. We then introduced a terminology that indicates the discrete-time channel's impulse response. That was, *channel tap*, and we denoted it by h_k in an effort to differentiate the tap index k from the time index m. Hence, the equation (2) is re-written as:

$$y[m] = \sum_{k=0}^{\infty} h_k x[m-k] + w[m].$$

Notice that $h_k = 0$ for $k < 0$. This is because the output cannot depend on future inputs.

We then investigated an issue that arises in the above channel. To see the issue clearly, we considered a sequential communication scheme in which an information bit b_m is mapped to each transmitted signal $x[m]$, and focused on the equation in the following form:

$$\begin{aligned} y[m] = {} & h_0 x[m] \\ & + h_1 x[m-1] + h_2 x[m-2] + \cdots \\ & + w[m]. \end{aligned} \tag{3}$$

The issue was that in view of the current symbol, say $x[m]$, the past symbols (marked in red above) cause *interference*. The phenomenon is called *inter-symbol interference*, ISI for short.

Today's lecture

Today we will explore the design of the optimal receiver that takes into account the ISI issue. Specifically what we are going to do are three folded. First we will derive the optimal ML receiver. I will then argue that a naive way of implementing the optimal receiver comes with a challenge in *complexity*. Lastly I will introduce an efficient algorithm that addresses the issue. That is, the *Viterbi algorithm*.

Signal representation

For illustrative simplicity, let us consider a simple two-tap ISI channel:

$$y[m] = h_0 x[m] + h_1 x[m-1] + w[m]. \tag{4}$$

We consider the sequential communication scheme in which we send one bit at a time:

$$x[m] = \begin{cases} +\sqrt{E}, & b_m = 1; \\ -\sqrt{E}, & b_m = 0 \end{cases} \tag{5}$$

where $m = 0, \ldots, n-1$. Here n denotes the total number of time slots used. Suppose we want to decode $x[m]$. Note that at time m, the interference $h_1 x[m-1]$ explicitly contains information about the bit that was sent in time $m-1$. Since $x[m-1]$ is related to $x[m-2]$ due to the ISI and so on, it implicitly contains information about all the bits that were sent before time m. Hence we can readily expect that all of the received signals $y[m]$'s affect decoding $x[m]$. Similarly decoding the other transmitted signals is affected by all of the received signals. This motivates us to consider a *block* decoding scheme in which we decode all the bits looking at all the received signals. Specifically, given $(y[0], \ldots, y[n-1])$, we wish to decode $(x[0], \ldots, x[n-1])$ altogether.

In light of this block decoding, the signals $x[m]$ and $x[m-1]$ in (4) are *all desired*. This then naturally motivates us to represent (4) as:

$$y[m] = \underbrace{\begin{bmatrix} h_1 & h_0 \end{bmatrix}}_{=:\mathbf{h}^T} \underbrace{\begin{bmatrix} x[m-1] \\ x[m] \end{bmatrix}}_{=:\mathbf{s}[m]} + w[m]. \tag{6}$$

This way, the equation (6) looks like the *AWGN channel* for which we are familiar with the ML derivation. Here we call $\mathbf{s}[m]$ a *state*. Later you will readily figure out the rationale behind the naming. Since decoding $(x[0], x[1], \ldots, x[n-1])$ is equivalent to decoding$(\mathbf{s}[0], \mathbf{s}[1], \ldots, \mathbf{s}[n-1])$, it suffices to decode the states. Here we will focus decoding the states instead.

Optimal ML decision rule

The optimal ML decision rule is to find the sequence of states that maximizes the likelihood function w.r.t. the received signals $\mathbf{y} := (y[0], y[1], \ldots, y[n-1])$. So let us massage the likelihood function:

$$\begin{aligned} f_{\mathbf{y}}\left(y[0], \ldots, y[n-1] | \mathbf{s}\right) &\overset{(a)}{=} f_{\mathbf{w}}\left(y[0] - \mathbf{h}^T \mathbf{s}[0], \ldots, y[n-1] - \mathbf{h}^T \mathbf{s}[n-1] | \mathbf{s}\right) \\ &\overset{(b)}{=} \prod_{m=0}^{n-1} f_{w[m]}\left(y[m] - \mathbf{h}^T \mathbf{s}[m]\right) \\ &\overset{(c)}{=} \prod_{m=0}^{n-1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\left(y[m] - \mathbf{h}^T \mathbf{s}[m]\right)^2\right) \end{aligned} \tag{7}$$

where (a) is because $w[m] = y[m] - \mathbf{h}^T \mathbf{s}[m]$; and (b) is due to the independence of $w[m]$'s; and (c) comes from $w[m] \sim \mathcal{N}(0, \sigma^2)$. Hence, the optimal $\mathbf{s}^*$ would be:

$$\begin{aligned} \mathbf{s}^* &= \arg\max_{\mathbf{s}} \prod_{m=0}^{n-1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\left(y[m] - \mathbf{h}^T \mathbf{s}[m]\right)^2\right) \\ &= \arg\max_{\mathbf{s}} \exp\left(-\frac{1}{2\sigma^2} \sum_{m=0}^{n-1} \left(y[m] - \mathbf{h}^T \mathbf{s}[m]\right)^2\right) \\ &= \arg\min_{\mathbf{s}} \sum_{m=0}^{n-1} \left(y[m] - \mathbf{h}^T \mathbf{s}[m]\right)^2. \end{aligned} \tag{8}$$

A challenge in computation

Now how to compute $\mathbf{s}^*$ in (8)? One naive way is to do an *exhaustive* search. For all possible sequences of states, we compute the following interested quantity:

$$\sum_{m=0}^{n-1} \left(y[m] - \mathbf{h}^T \mathbf{s}[m]\right)^2. \tag{9}$$

We then find the sequence pattern that yields the minimum quantity. However, under this naive way, we are faced with a challenge in *complexity*. Why? Think about the total number of possible sequence patterns. How many possible sequences? That is, 2^n. Why? This is because decoding $(\mathbf{s}[0], \mathbf{s}[1], \ldots, \mathbf{s}[n-1])$ is equivalent to decoding $(x[0], x[1], \ldots, x[n-1])$ and the sequence of $(x[0], x[1], \ldots, x[n-1])$ takes one of the 2^n possibilities. So the challenge is that the complexity grows *exponentially* with n, which is definitely very prohibitive for a large value of n. To figure out exactly what the complexity is, let us compute the numbers of multiplication, addition and comparison required to figure out $\mathbf{s}^*$ from (8).

For each time slot, say m, the computation of $\left(y[m] - \mathbf{h}^T \mathbf{s}[m]\right)^2$ requires 3 multiplications and 2 additions. Since we have n of such computations, we get the following complexity:

$$\begin{aligned} &\#\text{ multiplication:} && 3n \cdot 2^n; \\ &\#\text{ addition:} && (2n + (n-1)) \cdot 2^n; \\ &\#\text{ comparison:} && 2^n - 1, \end{aligned}$$

where the number $n-1$ marked in blue comes from the complexity w.r.t. $\sum_{m=0}^{n-1}$.

A low complexity solution

Now a natural question that arises is then: Is there any smarter approach that implements the ML receiver with a much lower complexity? A giant in the communication and information theory fields addressed the question to develop a very efficient algorithm in which the complexity grows *linearly* with n. The name of the giant is *Andrew Viterbi*; see Fig. 1. Naming after his last name, the algorithm is called the *Viterbi algorithm*. For the rest of this lecture, we will study some concepts that form the basis of the Viterbi algorithm.

Figure 1: Andrew Viterbi is a giant in the communication and information theory fields.

Cost & finite state machine

The first is concerned about an interested quantity in the minimization problem:

$$\mathbf{s}^* = \arg\min_{\mathbf{s}} \sum_{m=0}^{n-1} \underbrace{\left(y[m] - \mathbf{h}^T \mathbf{s}[m]\right)^2}_{=:\, c_m(\mathbf{s}[m])} \tag{10}$$

Here the quantity $\left(y[m] - \mathbf{h}^T \mathbf{s}[m]\right)^2$ colored in green can be viewed as something *negative*, because the smaller the quantity, the better the situation is. Hence, we can call it "cost" w.r.t. the state $\mathbf{s}[m]$.

The second is related to the state (or state vector) $\mathbf{s}[m]$ that can take one of the following six candidates:

$$\begin{bmatrix} 0 \\ -\sqrt{E} \end{bmatrix}, \begin{bmatrix} 0 \\ +\sqrt{E} \end{bmatrix}, \underbrace{\begin{bmatrix} +\sqrt{E} \\ +\sqrt{E} \end{bmatrix}}_{s0}, \underbrace{\begin{bmatrix} -\sqrt{E} \\ +\sqrt{E} \end{bmatrix}}_{s1}, \underbrace{\begin{bmatrix} -\sqrt{E} \\ -\sqrt{E} \end{bmatrix}}_{s2}, \underbrace{\begin{bmatrix} +\sqrt{E} \\ -\sqrt{E} \end{bmatrix}}_{s3}.$$

Observe that the first two occur only at the beginning ($m = 0$), assuming that $x[-1] = 0$. So the two states are negligible relative to the others for a large value of n. For simplification, let us not worry about the two states. To this end, we intentionally set $x[-1] = +\sqrt{E}$.

Notice that each state $\mathbf{s}[m]$ can move from one to another, depending on the value of $x[m]$. So one can now think of a *Finite State Machine* (FSM) which exhibits state transitions as illustrated in Fig. 2. Here the $+$ or $-$ labeled above a transition arrow indicates the sign of $x[m+1]$, given

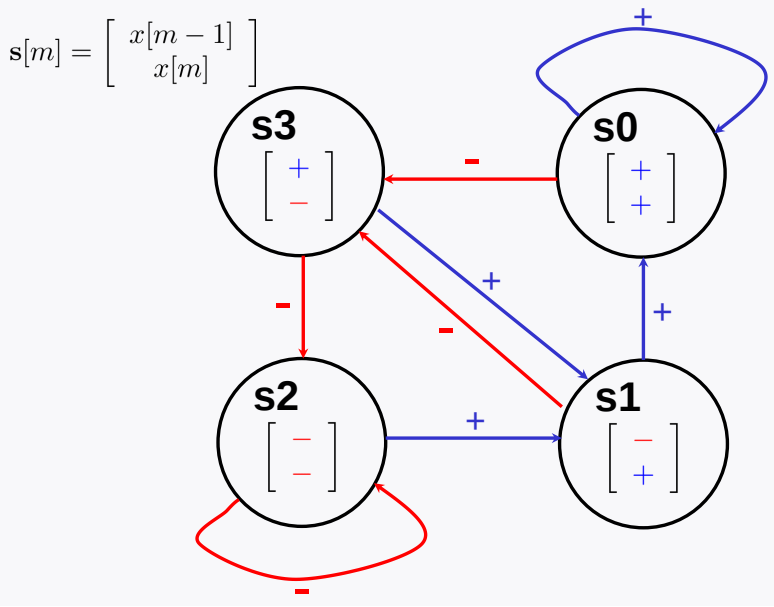


Figure 2: A finite state machine with four states. A transition occurs depending on the value of $x[m]$.

the current state $\mathbf{s}[m]$. For instance, suppose that $x[m+1] = -\sqrt{E}$ given the current state s0. Then, we move along the red transition arrow to arrive at the state s3 of $[+\sqrt{E}; -\sqrt{E}]$. While the state transition diagram well represents how each state moves around to another, it does

not capture the *time evolution*. This is exactly where another concept w.r.t. the state machine arises. That is, the *trellis diagram*.

Trellis diagram

The trellis diagram exhibits both state transitions and time evolution. To clearly understand how it works, let me walk you through details with Figs. 3 and 4.

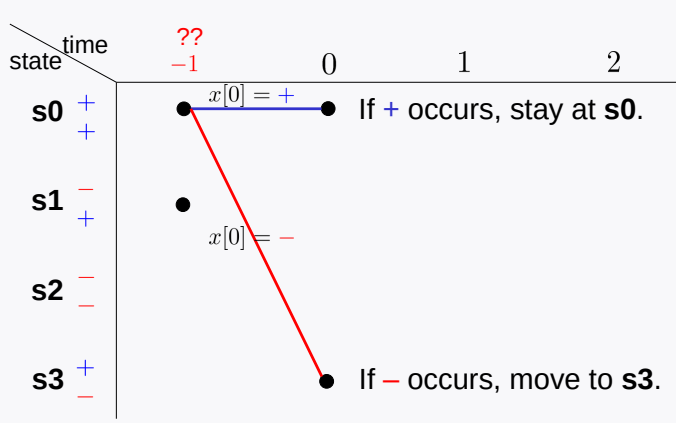


Figure 3: A trellis diagram at $m = 0$.

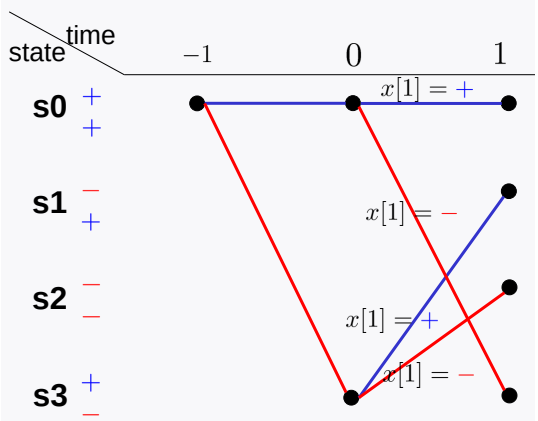


Figure 4: A trellis diagram at $m = 1$.

Consider the state at time 0:

$$\mathbf{s}[0] = \begin{bmatrix} x[-1] \\ x[0] \end{bmatrix} = \begin{bmatrix} +\sqrt{E} \\ x[0] \end{bmatrix} = \begin{cases} \begin{bmatrix} +\sqrt{E} \\ +\sqrt{E} \end{bmatrix} = \text{s0}, & \text{if } x[0] = +\sqrt{E}; \\ \begin{bmatrix} +\sqrt{E} \\ -\sqrt{E} \end{bmatrix} = \text{s3}, & \text{if } x[0] = -\sqrt{E}, \end{cases}$$

where the second equality is due to our assumption $x[-1] = +\sqrt{E}$. Notice that $\mathbf{s}[0]$ takes one of the two possible states s0 and s3, reflected in the two dots at time 0 in Fig. 3. In time -1, we have two options for $\mathbf{s}[-1]$ depending on the value of $x[-2]$: s0 and s1. To remove the ambiguity in time -1, let us further assume $x[-2] = +\sqrt{E}$ to fix the initial state as s0. This way, we can ensure that we start always from s0. As mentioned earlier, if $x[0] = +\sqrt{E}$, $\mathbf{s}[0] = \text{s0}$; otherwise

$\mathbf{s}[0] = \text{s3}$. To be more familiar with how it works, let us consider one more time slot as illustrated in Fig. 4. Suppose $\mathbf{s}[0] = \text{s0}$. If $x[0] = +\sqrt{E}$, then it stays at the same state s0 (reflected in the blue transition arrow; otherwise, it moves to s3 (reflected in the red arrow). On the other hand, given $\mathbf{s}[0] = \text{s3}$, if $x[0] = +\sqrt{E}$, then it moves to s1; otherwise, it goes to s2.

Cost calculation

Now how to calculate the cost in (10) of our interest from the trellis diagram? To see how it works, consider a concrete example for $n = 4$, as illustrated in Fig. 5.

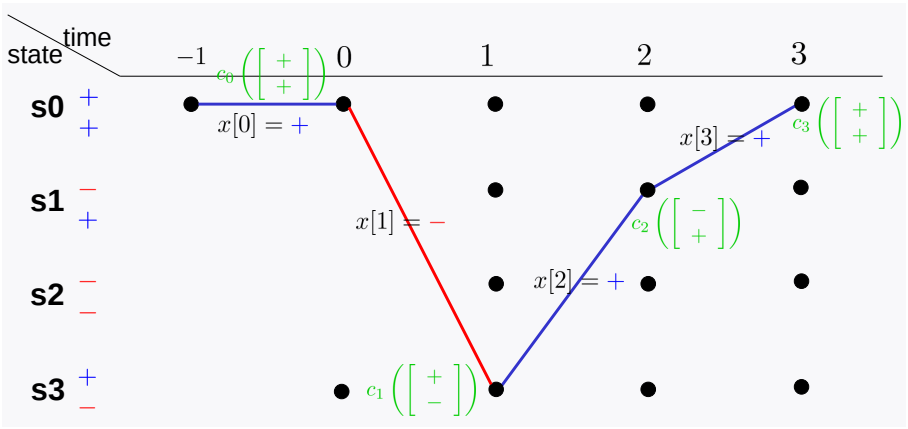


Figure 5: A trellis diagram for the sequence $(x[0], x[1], x[2], x[3]) = (+\sqrt{E}, -\sqrt{E}, +\sqrt{E}, +\sqrt{E})$.

This is the example in which $(x[0], x[1], x[2], x[3]) = (+\sqrt{E}, -\sqrt{E}, +\sqrt{E}, +\sqrt{E})$, so the trellis path takes the blue-red-blue-blue transition arrows, yielding the state change as: s0-s0-s3-s1-s0. To ease cost calculation, here we leave an associated cost at the corresponding state node. For instance, we put $c_0([+\sqrt{E}; +\sqrt{E}])$ (marked in green in Fig. 5) nearby the black dot s0 in time 0. Similarly we leave $c_1([+\sqrt{E}; -\sqrt{E}])$, $c_2([-\sqrt{E}; +\sqrt{E}])$, $c_3([+\sqrt{E}; +\sqrt{E}])$ for the corresponding black dots. By aggregating all the costs associated with the black dots, we can readily compute the cost for one such sequence. Considering all possible sequence patterns, we can compute the optimal path as:

$$\mathbf{s}^* = \arg\min_{\mathbf{s}} \{c_0(\mathbf{s}[0]) + c_1(\mathbf{s}[1]) + c_2(\mathbf{s}[2]) + c_3(\mathbf{s}[3])\} .$$

As mentioned earlier, a naive *exhaustive* search requires 2^4 possibilities - so the complexity is very expensive especially for a large n.

Look ahead

Next time, we will study the Viterbi algorithm that well exploits the structure of the trellis diagram to find $\mathbf{s}^*$ efficiently.

May 15, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 14: The Viterbi algorithm

Recap

Last time, we investigated the optimal ML receiver for a simple two-tap ISI channel:

$$y[m] = h_0 x[m] + h_1 x[m-1] + w[m] \tag{1}$$

where $w[m]$'s are i.i.d. $\sim \mathcal{N}(0, \sigma^2)$. For simplicity, we considered a simple sequential communication scheme in which an information bit b_m is mapped to each transmitted signal $x[m]$. Since each symbol propagates upto the last received signal through the ISI term $h_1 x[m-1]$, we considered a block decoding in which we wish to decode the entire symbols $(x[0], x[1], \ldots, x[n-1])$ based on all the received signals $(y[0], y[1], \ldots, y[n-1])$. Here n denotes the total number of time slots used. Specifically we represented (1) as:

$$y[m] = \underbrace{[h_1 \quad h_0]}_{=:\mathbf{h}^T} \underbrace{\begin{bmatrix} x[m-1] \\ x[m] \end{bmatrix}}_{=:\mathbf{s}[m]} + w[m]. \tag{2}$$

We then intended to encode the entire state vectors $(\mathbf{s}[0], \mathbf{s}[1], \ldots, \mathbf{s}[n-1])$ which is equivalent to decoding $(x[0], x[1], \ldots, x[n-1])$. Manipulating the likelihood function w.r.t. the received signals, we could obtain the ML solution:

$$\mathbf{s}^* = \arg\min_{\mathbf{s}} \sum_{m=0}^{n-1} \big(y[m] - \mathbf{h}^T \mathbf{s}[m]\big)^2. \tag{3}$$

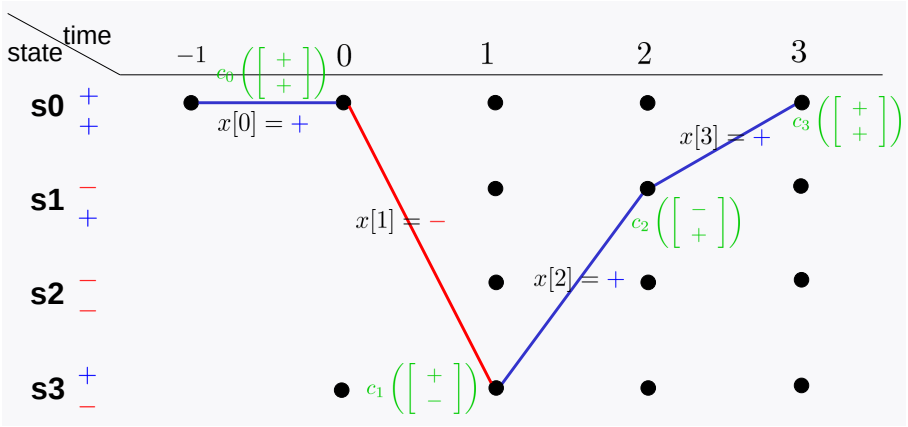


Figure 1: A trellis diagram for the sequence $(x[0], x[1], x[2], x[3]) = (+\sqrt{E}, -\sqrt{E}, +\sqrt{E}, +\sqrt{E})$.

Since a native exhaustive approach requires a heavy computation for 2^n possible sequence patterns, we introduced an efficient algorithm with a much lower complexity: the Viterbi algorithm. We then studied a couple of concepts that form the basis of the algorithm. One is the *cost* that represents an interested quantity in the optimization (3), marked in blue. The second is the

finite state machine (FSM) that concerns the state vector $\mathbf{s}[m]$. The third is the trellis diagram that visualizes how each state changes in time. Fig. 1 illustrates an example of the trellis diagram for the sequence $(x[0], x[1], x[2], x[3]) = (+\sqrt{E}, -\sqrt{E}, +\sqrt{E}, +\sqrt{E})$, assuming that $x[-1] = x[-2] = +\sqrt{E}$.

Today's lecture

It turns out the structure of the trellis diagram gave a significant insight into the invention of the Viterbi algorithm. Today we will study the trellis-diagram-inspired Viterbi algorithm which yields a much lower complexity growing linearly with n.

Key observation

The Viterbi algorithm is inspired by the following key observation. To see this clearly, let us consider two possible sequence patterns presented in Fig. 2:

(i) $(x[0], x[1], x[2], x[3]) = (-\sqrt{E}, +\sqrt{E}, -\sqrt{E}, +\sqrt{E})$ (marked in purple);
(ii) $(x[0], x[1], x[2], x[3]) = (-\sqrt{E}, +\sqrt{E}, -\sqrt{E}, -\sqrt{E})$ (makred in blue).

Here the key observation is that the two trellis paths are *significantly overlapped*; hence, the

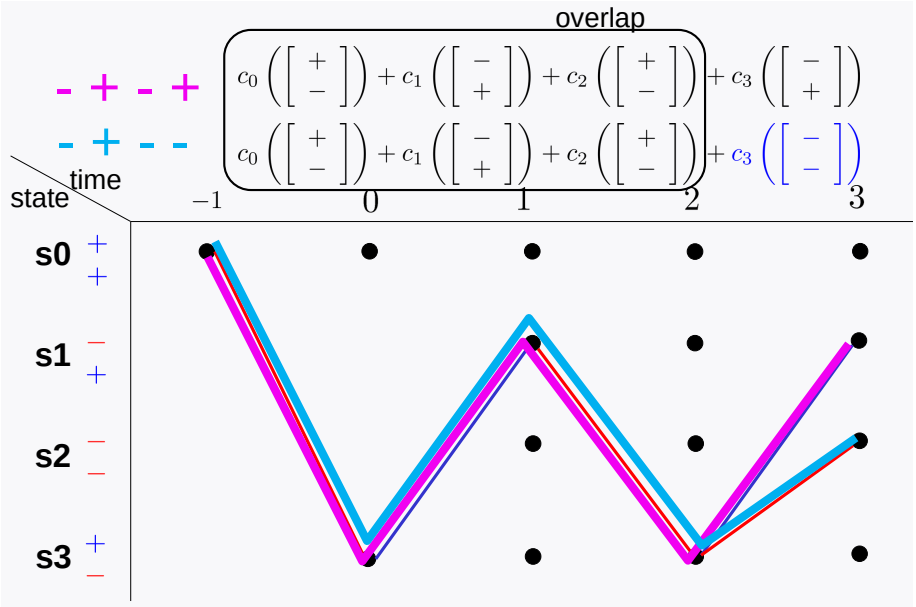


Figure 2: Key observation.

two corresponding costs are identical except w.r.t. the last state vector. This motivated Viterbi to come up with the following natural idea.

Idea of the Viterbi algorithm

The idea is to successively store only an *aggregated cost* up to time t and then use this to compute a follow-up aggregated cost w.r.t. the next time slot. In order to understand what this means in detail, let us consider a simple example in which $(h_0, h_1) = (1, 1)$, $E = 1$ and we receive:

$$(y[0], y[1], y[2], y[3]) = (2.1, 1.8, 0.5, -2.2).$$

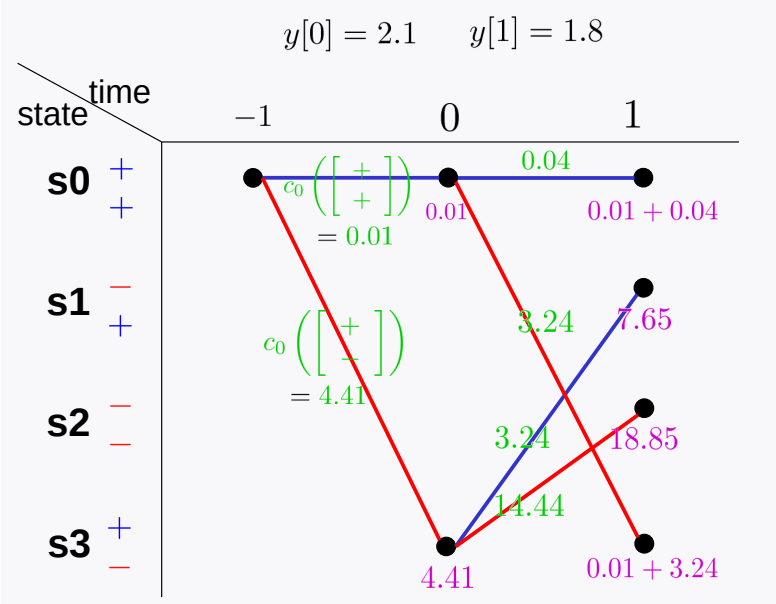


Figure 3: Cost computation and store strategy.

See Figs. 3 and 4. First compute the cost at time 0 when $x[0] = +1$:

$$c_0\left(\begin{bmatrix} + \\ + \end{bmatrix}\right) = \left(y[0] - \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} +1 \\ +1 \end{bmatrix}\right)^2 = 0.01. \tag{4}$$

The cost value of 0.01 is then placed along the blue transition arrow, as illustrated in Fig. 3. We then store the cost at the state s0 node at time 0 (a black dot). Here we indicate the store by marking a purple-colored number nearby the black dot. We do the same thing w.r.t. the cost yet now when $x[0] = -1$:

$$c_0\left(\begin{bmatrix} + \\ - \end{bmatrix}\right) = \left(y[0] - \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} +1 \\ -1 \end{bmatrix}\right)^2 = 4.41. \tag{5}$$

We place the 4.41 along the associated red transition arrow and then store 4.41 nearby the black dot w.r.t. the state s3.

Next we compute the cost w.r.t. time 1. The cost for $x[1] = +\sqrt{E}$ is computed as 0.04 (check!). So the aggregated cost up to time 1 w.r.t. the state s0 would be 0.01+0.04, as illustrated in Fig. 3. Similarly the aggregated costs for the other states (s1, s2, s3) would be 4.41+3.24, 4.41+ 14.44, 0.01+3.24, respectively. Also check!

Now one can see the core idea of the Viterbi algorithm from time 2. See Fig. 4. Consider the state s0 at time 2. This state occurs when $x[2] = +1$ and comes from two possible prior states: s0 and s1. The cost for $x[2] = +1$ is first computed as 2.25. So the aggregated cost assuming that it comes from the prior s0 state (storing 0.05 for the aggregated cost up to time 1) would be: $0.05 + 2.25 = 2.3$. On the other hand, the aggregated cost w.r.t. the prior s1 state would be: $7.65 + 2.25 = 9.9$. Now how to deal with the two cost values? Remember at the end of the day that we are interested in finding the path that yields the *minimum* aggregated cost. So the path w.r.t. the larger cost 9.9 would be eliminated in the competition. So we don't need to worry about any upcoming paths w.r.t. the larger cost. This naturally motivates us to store only the *minimum* between the two cost values at the state s0 in time 2, while ignoring the other loser path. So we store 2.3 at the node, as illustrated in Fig. 5. We do the same thing

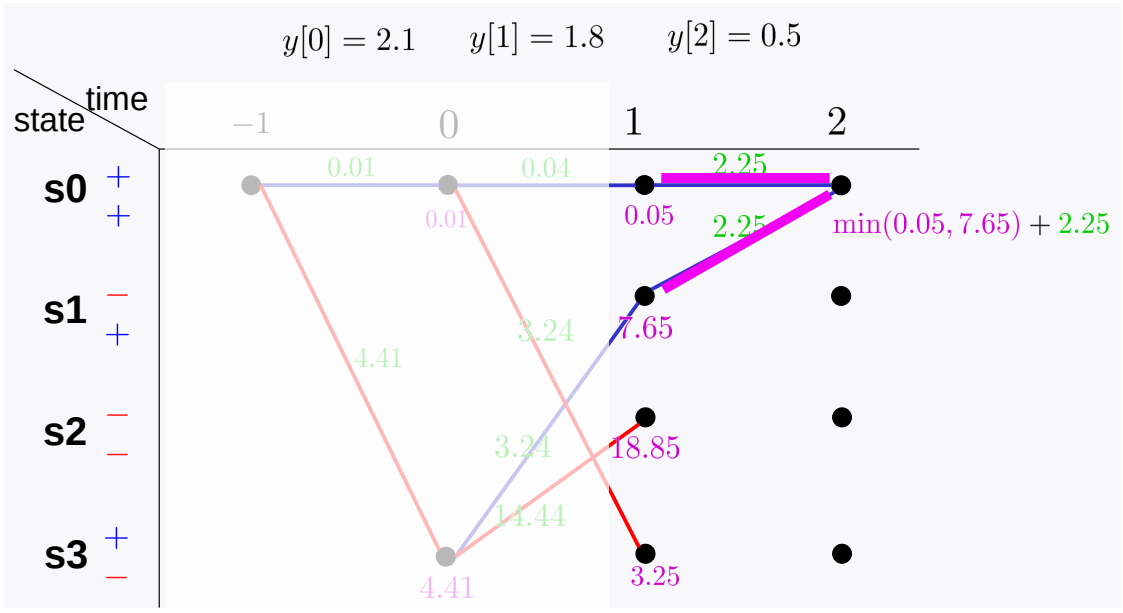


Figure 4: Ided of the Viterbi algorithm

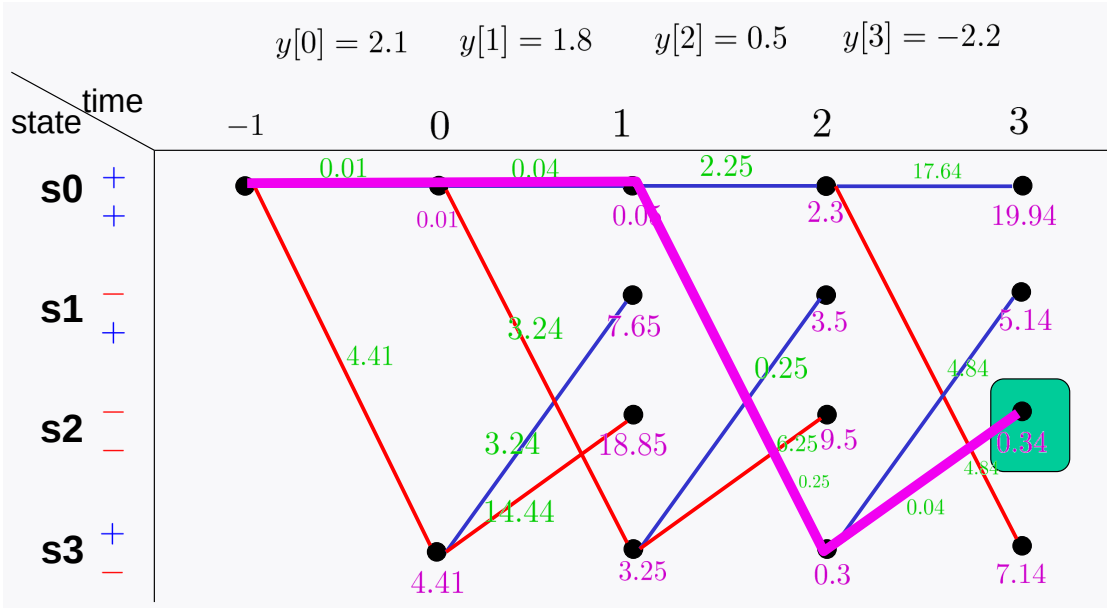


Figure 5: Choose $\mathbf{s}^*$ that minimizes the aggregated cost.

for the other states. For the state s1, the *lower* path turns out to be the winner, so we store the corresponding aggregated cost 3.5 at the state s1 node, while deleting the upper loser path. Similarly for the states s2 and s3. We repeat this procedure until the last time slot. See all the associated computations in Fig. 5.

Now how to find the path that yields the minimum aggregated cost from the picture? It is very simple. Take a look at the four aggregated costs at the last time slot: 19.94, 5.14, 0.34, 7.14. We then pick up the minimum cost 0.34. Now how to find the corresponding sequence pattern? Since we leave only the survivor paths in the picture while deleting loser paths, we can readily find the path via *backtracking*. The survivor paths form a purple trajectory in Fig. 5, which corresponds to the sequence $(x[0], x[1], x[2], x[3]) = (+\sqrt{E}, +\sqrt{E}, -\sqrt{E}, -\sqrt{E})$.

Complexity

Remember I argued that the complexity of the Viterbi algorithm grows linearly with n. To see this clearly, let us consider the essential operation that occurs in each node. Focus on the operation w.r.t. the state s0 node at time 2, as illustrated in Fig. 6. Here the current cost computation

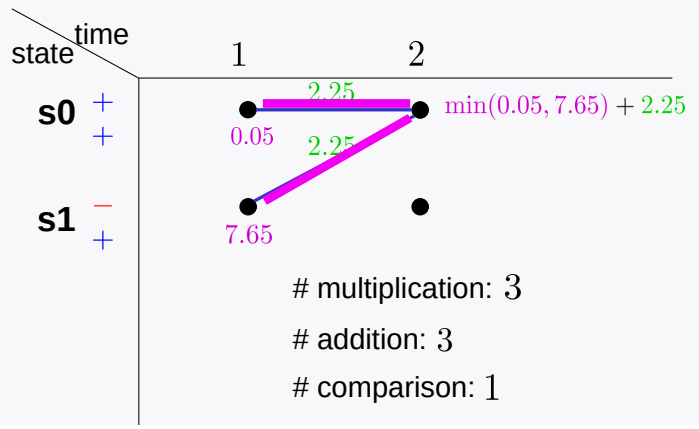


Figure 6: Complexity per state.

requires three multiplications and two additions. Next we need one comparison for taking the minimum between the two, and one additional addition for aggregating the costs. Hence, the complexity per state would be: three multiplications; three additions; and one comparison.

This operation is repeated for the other states spanning the entire time slots, reflected in all the dots in the picture. Since the total number of black dots is $4n$, the complexity of the Viterbi algorithm grows linearly with n; see the precise numbers in Fig. 7.

Issue

While the Viterbi algorithm has a much lower complexity relative to the native exhaustive search, it still comes with a challenge in complexity. The complexity is concerned about the *number of states*. The problem is that this number can be large. For instance, for an L-tap ISI channel: the number of states would be 2^L. Hence, the complexity grows *exponentially* with L.

Look ahead

It turns out there is another scheme that can further reduce the complexity even when L is large. Next time, we will study such scheme.

	Exhaustive	Viterbi
# multiplication:	$3n \cdot 2^n$	$3 \cdot 4n$
# addition:	$(3n-1) \cdot 2^n$	$3 \cdot 4n$
# comparison:	$2^n - 1$	$1 \cdot 4n$

Figure 7: Complexity comparison.

May 20, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 15: OFDM (1/3)

Recap

Last time, we investigated how to optimally decode transmitted symbols in the context of a two-tap ISI channel:

$$\begin{aligned} y[m] &= h_0 x[m] + h_1 x[m-1] + w[m] \\ &= \underbrace{[1 \quad h_0]}_{=:\mathbf{h}^T} \underbrace{\begin{bmatrix} x[m-1] \\ x[m] \end{bmatrix}}_{=:\mathbf{s}[m]} + w[m] \end{aligned} \tag{1}$$

where $w[m]$'s are i.i.d. $\sim \mathcal{N}(0, \sigma^2)$. Inspired by the fact that $x[m]$'s are closely coupled in the received signals via the ISI term, we focused on the *block decoding* which intends to decode $(x[0], x[1], \ldots, x[n-1])$ altogether, based on the entire received signals: $(y[0], y[1], \ldots, y[n-1])$. Here n indicates the number of time slots used. We then derived the optimal ML receiver w.r.t. the sequence of state vectors $\mathbf{s}[m]$'s (one-to-one mapping with $(x[0], x[1], \ldots, x[n-1])$):

$$\mathbf{s}^* = \arg \min_{\mathbf{s}} \sum_{m=0}^{n-1} \left(y[m] - \mathbf{h}^T \mathbf{s}[m]\right)^2. \tag{2}$$

We emphasized that a naive way of implementing the ML solution based on an exhaustive search requires a very expensive complexity: 2^n growing exponentially with n.

I then mentioned about another yet very efficient way of implementing the ML solution: the *Viterbi algorithm* whose complexity grows *linearly* with n. However, in light of the complexity w.r.t. the number of states in an associated FSM, it is still problematic. The complexity grows linearly in the number of states 2^L, therefore it is *exponential* with L. Here L could be very large in practice, so it could be very much problematic.

Today's lecture

Today we will study another scheme that can significantly reduce the complexity even when L is large. The scheme that we will study is known as *Orthogonal Frequency Division Multiplexing*, OFDM for short. Today we will study the key idea of OFDM as well as how it works in the context of the simplest ISI channel setting: $L = 2$. Next time, we will extend to arbitrary L-tap ISI channels.

A fixed transmission scheme

So far we have considered a *receiver-centric* approach in which a transmission scheme is *fixed*, e.g., as the sequential communication scheme where we send each bit b_m by transmitting $x[m] = \pm\sqrt{E}$. It turns out using a sophisticated *transmitter-side* technique, one can make a channel effectively ISI-free so that the complexity becomes irrelevant to L. Encouragingly, the technique yields a receiver architecture no more complicated than the one used in the past lectures. While this seems too good to be true, we will see this is the case remarkably.

Cyclic signaling

Suppose we send N transmitted symbols:

$$\mathbf{x} := [x[0], x[1], \ldots, x[N-1]]^T.$$

Note that time slots begin at zero. We have not yet specified how to map information bits into $x[m]$'s. Later we will provide details for the encoding rule. Consider received signals:

$$\begin{aligned} y[0] &= h_0 x[0] + w[0] \\ y[1] &= h_1 x[0] + h_0 x[1] + w[1] \\ y[2] &= h_1 x[1] + h_0 x[2] + w[2] \\ &\vdots \\ y[N-1] &= h_1 x[N-2] + h_0 x[N-1] + w[N-1]. \end{aligned} \tag{3}$$

Employing a matrix notation, we can then write a sequence of the received signals as:

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{w}, \tag{4}$$

where $\mathbf{y} := [y[0], \ldots, y[N-1]]^T$, $\mathbf{w} := [w[0], \ldots, w[N-1]]^T$, and

$$\mathbf{H} = \begin{bmatrix} h_0 & & & & & \\ h_1 & h_0 & & & & \\ & h_1 & h_0 & & & \\ & & & \ddots & & \\ & & & & h_1 & h_0 \end{bmatrix}. \tag{5}$$

Notice that $\mathbf{H}$ has a very *systematic* structure: Each row contains a shifted version of (h_1, h_0), except the first row. Why we have an exception in the first row? This is because $x[-1] = 0$. This motivates us to consider a beautiful yet imaginary structure as the one in Fig. 1. Here we

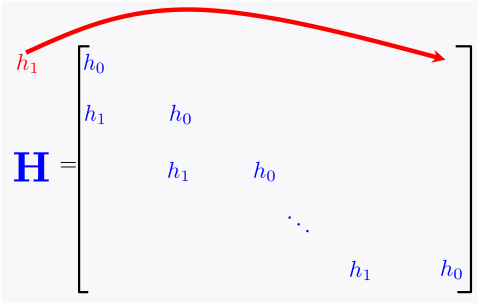


Figure 1: A nicely looking structure for $\mathbf{H}$.

have h_1 in front of h_0 in the first row. But it is placed in an *invalid* position in the matrix. Another beautiful yet *valid* matrix might be the following:

$$\mathbf{H}_c = \begin{bmatrix} h_0 & & & & & h_1 \\ h_1 & h_0 & & & & \\ & h_1 & h_0 & & & \\ & & & \ddots & & \\ & & & & h_1 & h_0 \end{bmatrix}. \tag{6}$$

where the position for h_1 in the first row (marked in red) is now valid while yielding a shifted copy of (h_1, h_0). Here the shift means any type of shift that allows for a *cyclic* shift. Observe

that every row is rotated one element to the right relative to the preceding row. This is a well-known property in the linear algebra literature, called the *circulant* property. So $\mathbf{H}_c$ is a circulant matrix. It turns out that the circulant matrix has a very nice property that helps converting the original ISI channel into multiple ISI-free subchannels.

A way to enable cyclic signaling

Actually we can also implement this cyclic signaling. So prior to digging into the nice property, let us first investigate the way that enables such signaling. Notice in Fig. 1 that h_1 in the first row (marked in red) is placed in a position corresponding to time slot -1 in view of the real channel $\mathbf{H}$. In light of $\mathbf{H}_c$, however, the corresponding time slot w.r.t. h_1 is $N-1$. Hence, the idea is to send a dummy symbol $x[-1]$ at time -1 so that it acts as the symbol at time $N-1$. In other words, the idea is to set $x[-1] = x[N-1]$ and then send $\mathbf{x}'$ that augments $x[-1] = x[N-1]$ in front of $\mathbf{x}$:

$$\mathbf{x}' = [x[N-1], x[0], \ldots, x[N-1]]^T. \tag{7}$$

See Fig. 2.

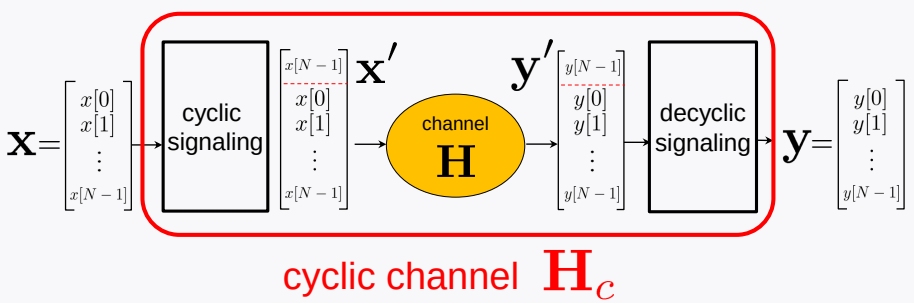


Figure 2: How to generate cyclic signaling.

This way we get:

$$\mathbf{y}' = \mathbf{H}\mathbf{x}' + \mathbf{w}' \tag{8}$$

where $\mathbf{y}' := [y[-1], y[0], \ldots, y[N-1]]^T$ and $\mathbf{w}' := [w[-1], w[0], \ldots, w[N-1]]^T$. By removing the first component in $\mathbf{y}'$ (that we call "decyclic operation" in Fig. 2), we obtain:

$$\mathbf{y} = \mathbf{H}_c\mathbf{x} + \mathbf{w}. \tag{9}$$

A nice property of $\mathbf{H}_c$

Now let us investigate the nice property that enables us to remove ISI very efficiently with the cyclic structure reflected in (9). The nice property is that an eigenvalue decomposition of the circulant matrix $\mathbf{H}_c$ has the following structure:

$$\mathbf{H}_c = \mathbf{F}^{-1}\mathbf{\Lambda}\mathbf{F} \tag{10}$$

where $\mathbf{F}$ denotes the Discrete Fourier Transform (DFT) matrix:

$$\mathbf{F} := \frac{1}{\sqrt{N}} \begin{bmatrix} \omega^{0\cdot 0} & \omega^{1\cdot 0} & \cdots & \omega^{(N-1)\cdot 0} \\ \omega^{0\cdot 1} & \omega^{1\cdot 1} & \cdots & \omega^{(N-1)\cdot 1} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{0\cdot (N-1)} & \omega^{1\cdot (N-1)} & \cdots & \omega^{(N-1)\cdot (N-1)} \end{bmatrix}. \tag{11}$$

Here $w := e^{-j\frac{2\pi}{N}}$.

Frequency-domain signals

Now why does the nice property (10) help removing ISI? To see this, let us apply (10) to (9) and then multiply the DFT matrix $\mathbf{F}$ to both sides. This way we obtain:

$$\begin{aligned}\tilde{\mathbf{y}} &:= \mathbf{F}\mathbf{y} \\ &= \Lambda \underbrace{(\mathbf{F}\mathbf{x})}_{=:\tilde{\mathbf{x}}} + \underbrace{(\mathbf{F}\mathbf{w})}_{=:\tilde{\mathbf{w}}} \\ &= \Lambda\tilde{\mathbf{x}} + \tilde{\mathbf{w}}.\end{aligned} \tag{12}$$

Here the key observation is that Λ is a *diagonal* matrix:

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{N-1} \end{bmatrix}. \tag{13}$$

This means that $\tilde{\mathbf{y}}$ does not contain any ISI terms w.r.t. frequency-domain signal components in $\tilde{\mathbf{x}}$. So if we map information bits into frequency-domain signal vector $\tilde{\mathbf{x}}$, then we get multiple ISI-free parallel subchannels. Denoting $\tilde{\mathbf{x}} = [X[0], \ldots, X[N-1]]^T$ and $\tilde{\mathbf{y}} = [Y[0], \ldots, Y[N-1]]^T$, the parallel subchannels are described as:

$$\begin{aligned} Y[0] &= \lambda_0 X[0] + W[0]; \\ Y[1] &= \lambda_1 X[1] + W[1]; \\ &\vdots \\ Y[N-1] &= \lambda_{N-1} X[N-1] + W[N-1]. \end{aligned} \tag{14}$$

Note that frequency-domain signals $X[k]$'s are not interfering with each other. Hence, the above transmitter-receiver technique that yields (14) is called *Orthogonal Frequency Division Multiplexing*, OFDM for short. Here is the overall OFDM structure. See Fig. 3.

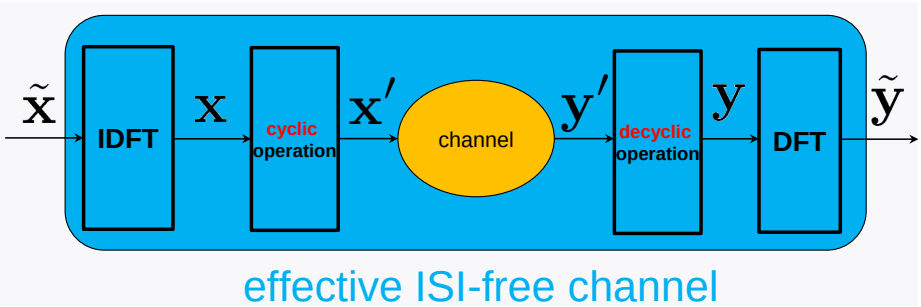


Figure 3: The transmitter-receiver structure of OFDM.

At the transmitter, we first generate frequency-domain signal vector $\tilde{\mathbf{x}}$ with information bits - we will study how to map information bits into $\tilde{\mathbf{x}}$ shortly. We then apply the IDFT matrix $\mathbf{F}^{-1}$, in order to convert information-bearing frequency-domain signals $\tilde{\mathbf{x}}$ into time-domain signals $\mathbf{x}$. Finally we send over the channel $\mathbf{x}'$ that augments $x[N-1]$ in front of $\mathbf{x}$, as described in (7).

At the receiver, we do exactly the other way around. We first remove the first component from the received signal vector $\mathbf{y}'$ to obtain $\mathbf{y}$. We then multiply the DFT matrix $\mathbf{F}$ to $\mathbf{y}$, thus obtaining $\tilde{\mathbf{y}}$ that will be used for decoding the information bits.

How to map information bits

Now the last thing to worry about is how to map information bits into $X[k]$'s. The idea is straightforward: mapping bits into $\tilde{\mathbf{x}}$ via some encoding rule as illustrated in Fig. 4. As for

Figure 4: How to map information bits into frequency-domain signals in OFDM.

encoding, we may want to any coding schemes that we learned in Part I, such as repetition code, LDPC code, Polar code.

What about decoding? Note that the frequency-domain received signals in (12) is very much similar to those in the AWGN channel. Here one thing that is not clear is whether $W[k]$'s are i.i.d. $\sim \mathcal{N}(0, \sigma^2)$. It turns out this is also the case. So we can apply any decoding schemes used in the AWGN channel.

Look ahead

We developed a transmitter-receiver technique (OFDM) that yields multiple ISI-free subchannels in the context of $L = 2$. While doing this, we relied on the nice property (10) as well as the fact that $W[k]$'s are i.i.d. $\sim \mathcal{N}(0, \sigma^2)$, without proving them. Next time, we will extend to general L-tap ISI channels. We will also prove the properties in the context of general channels. Further we will figure out the explicit expression for Λ in terms of h_k's. Finally we will demonstrate that the complexity of the scheme is reasonably good even for a large value of L.

May 22, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 16: OFDM (2/3)

Recap

Last time, we investigated a transmitter-receiver technique, called OFDM, in the context of the simplest ISI channel setting $L = 2$:

$$y[m] = h_0 x[m] + h_1 x[m-1] + w[m], \qquad m = 0, 1, \ldots, N-1 \tag{1}$$

where $w[m]$'s are i.i.d. $\sim \mathcal{N}(0, \sigma^2)$ and N denotes the number of transmitted symbols. Introducing matrix notations $\mathbf{x} := [x[0], x[1], \ldots, x[N-1]]^T$, $\mathbf{y} := [y[0], y[1], \ldots, y[N-1]]^T$ and $\mathbf{w} := [w[0], w[1], \ldots, w[N-1]]^T$, we represented (1) simply as:

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{w} \tag{2}$$

where

$$\mathbf{H} = \begin{bmatrix} h_0 & & & & & \\ h_1 & h_0 & & & & \\ & h_1 & h_0 & & & \\ & & & \ddots & & \\ & & & & h_1 & h_0 \end{bmatrix}. \tag{3}$$

Motivated by the beautiful (yet non-perfectly beautiful) structure of $\mathbf{H}$, we invoked the idea of sending an augmented signal vector $\mathbf{x}' := [x[-1], x[0], x[1], \ldots, x[N-1]]^T$ while setting $x[-1] = x[N-1]$. This then led to:

$$\mathbf{y} = \mathbf{H}_c\mathbf{x} + \mathbf{w} \tag{4}$$

where $\mathbf{H}_c$ indicates a circulant matrix:

$$\mathbf{H}_c = \begin{bmatrix} h_0 & & & & & h_1 \\ h_1 & h_0 & & & & \\ & h_1 & h_0 & & & \\ & & & \ddots & & \\ & & & & h_1 & h_0 \end{bmatrix}. \tag{5}$$

Next by relying on the nice property of the circulant matrix (that we deferred proving):

$$\mathbf{H}_c = \mathbf{F}^{-1}\mathbf{\Lambda}\mathbf{F}, \tag{6}$$

we could make the ISI channel effectively ISI-free in the frequency domain:

$$\tilde{\mathbf{y}} = \Lambda\tilde{\mathbf{x}} + \tilde{\mathbf{w}}. \tag{7}$$

Here $\tilde{\mathbf{y}} = \mathbf{F}\mathbf{y}$, $\tilde{\mathbf{x}} = \mathbf{F}\mathbf{x}$, $\tilde{\mathbf{w}} = \mathbf{F}\mathbf{w}$ and $\mathbf{F}$ denotes the Discrete Fourier Transform (DFT) matrix:

$$\mathbf{F} := \frac{1}{\sqrt{N}} \begin{bmatrix} \omega^{0\cdot 0} & \omega^{1\cdot 0} & \ldots & \omega^{(N-1)\cdot 0} \\ \omega^{0\cdot 1} & \omega^{1\cdot 1} & \ldots & \omega^{(N-1)\cdot 1} \\ \vdots & \vdots & \ddots & \vdots \\ \omega^{0\cdot(N-1)} & \omega^{1\cdot(N-1)} & \ldots & \omega^{(N-1)\cdot(N-1)} \end{bmatrix} \tag{8}$$

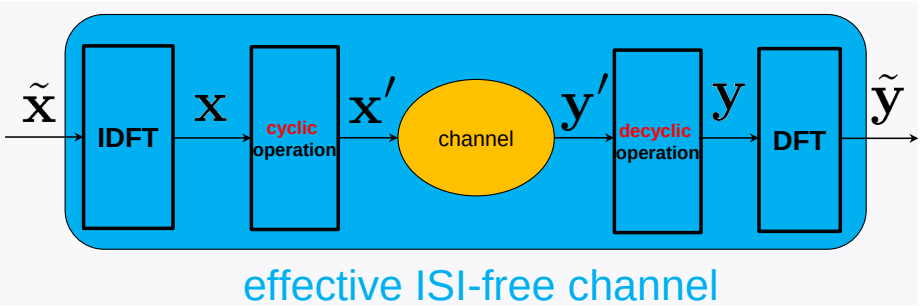


Figure 1: The transmitter-receiver structure of OFDM.

where $w := e^{-j\frac{2\pi}{N}}$. The overall procedure is illustrated in Fig. 1.

Today's lecture

Today we will extend this idea to general L-tap ISI channels. We will then demonstrate that the computational complexity of OFDM is reasonably good even for a large value of L. Finally we will prove the nice property (6) and investigate an explicit expression for the effective channel matrix Λ.

General L-tap ISI channel

In the general L-tap ISI channel, the received signals read:

$$y[m] = h_0 x[m] + h_1 x[m-1] + \cdots + h_{L-1} x[m-L+1] + w[m], \qquad m = 0, 1, \ldots, N-1. \quad (9)$$

In light of the matrix notation (2), the channel matrix reads:

$$\mathbf{H} = \begin{bmatrix} h_0 & & & & & & \\ h_1 & h_0 & & & & & \\ h_2 & h_1 & h_0 & & & & \\ & & & \ddots & & & \\ & & & & h_{L-1} & \cdots & h_1 & h_0 \end{bmatrix}. \quad (10)$$

The corresponding circulant matrix would be then:

$$\mathbf{H}_c = \begin{bmatrix} h_0 & & & h_{L-1} & & & \cdots & h_1 \\ h_1 & h_0 & & & h_{L-1} & & \cdots & h_2 \\ h_2 & h_1 & h_0 & & & h_{L-1} & \cdots & h_3 \\ & & & \ddots & & & & \\ & & & & h_{L-1} & \cdots & h_1 & h_0 \end{bmatrix}. \quad (11)$$

Now imagine the desired received signal w.r.t. the first row in (11):

$$y[0] = h_0 x[0] + h_1 x[N-1] + h_2 x[N-2] + \cdots + h_{L-1} x[N-L+1] + w[0]. \quad (12)$$

This motivates us to send the following dummy symbols in front of $\mathbf{x}$ as:

$$(x[-L+1], x[-L+2], \ldots, x[-1]) = (x[N-L+1], x[N-L+2], \ldots, x[N-1]). \quad (13)$$

In other words, we send an augmented signal vector $\mathbf{x}'$:

$$\mathbf{x}' = [x[N-L+1], \ldots, x[N-1], x[0], x[1], \ldots, x[N-L], x[N-L+1], \ldots, x[N-1]]^T. \quad (14)$$

Here the first $(L-1)$ elements are identical to the last $(L-1)$ elements. $\mathbf{x}'$ passes through the channel matrix $\mathbf{H}$, yielding:

$$\mathbf{y}' = \mathbf{H}\mathbf{x}' + \mathbf{w} \tag{15}$$

where $\mathbf{y}' := [y[-L+1], \ldots, y[-1], y[0], \ldots, y[N-1]]^T$. By removing the first L components from $\mathbf{y}'$ (that we call "remove CP" in Fig. 2), we can then obtain:

$$\mathbf{y} = \mathbf{H}_c\mathbf{x} + \mathbf{w}. \tag{16}$$

The whole procedure is illustrated in Fig. 2.

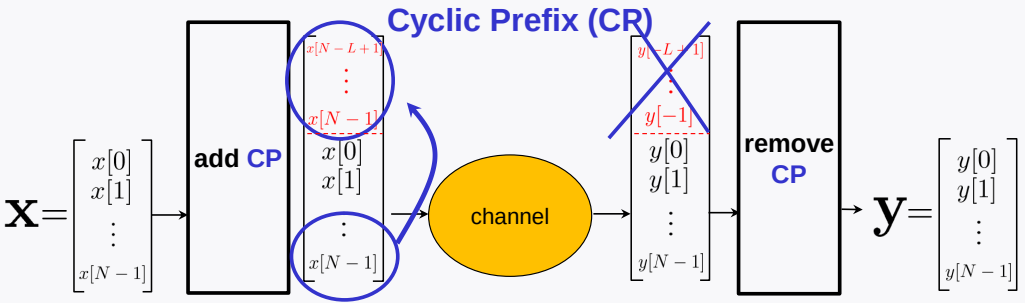


Figure 2: How to generate cyclic signaling.

Frequency-domain signals

Recall the nice property of the circulant matrix:

$$\mathbf{H}_c = \mathbf{F}^{-1}\mathbf{\Lambda}\mathbf{F}. \tag{17}$$

Applying this to (16), we obtain an effective channel:

$$\tilde{\mathbf{y}} = \Lambda\tilde{\mathbf{x}} + \tilde{\mathbf{w}} \tag{18}$$

where $\tilde{\mathbf{y}} = \mathbf{F}\mathbf{y}$, $\tilde{\mathbf{x}} = \mathbf{F}\mathbf{x}$ and $\tilde{\mathbf{w}} = \mathbf{F}\mathbf{w}$ are frequency-domain signals. Note that the effective channel is ISI-free, as Λ is a diagonal matrix.

Complexity

Now let us investigate the complexity of OFDM to demonstrate that it is indeed irrelevant to L. If we employ the normal IDFT/DFT matrix operations that require N^2 multiplications, we may still be concerned about a computational complexity. However, there is a very well-known transform that implements the DFT matrix operation much more efficiently. That is, Fast Fourier Transform, FFT for short. The FFT operation requires roughly $N \log_2 N$ multiplications, which is much smaller than N^2 especially for a large value of N. Compared to the complexity of the Viterbi algorithm ($\approx 2^L N$ multiplications), this is also a lower complexity particularly when L is large.

Penalty

Unfortunately there is a price to pay for reducing complexity in OFDM. Notice that we need lots of dummy signals when L is large. So the penalty (adding $(L-1)$ redundant signals) for creating the cyclic signaling increases with L.

Proof of (6)

We are now ready to prove the nice property (6) that we have relied on. By multiplying $\mathbf{F}^{-1}$ to the right in both sides in (6), we get:

$$\mathbf{H}_c\mathbf{F}^{-1} = \mathbf{F}^{-1}\mathbf{\Lambda}. \tag{19}$$

The IDFT matrix $\mathbf{F}^{-1}$ reads:

$$\mathbf{F}^{-1} = \frac{1}{\sqrt{N}} \begin{bmatrix} e^{j\frac{2\pi}{N}0\cdot 0} & e^{j\frac{2\pi}{N}0\cdot 1} & \cdots & e^{j\frac{2\pi}{N}0\cdot(N-1)} \\ e^{j\frac{2\pi}{N}1\cdot 0} & e^{j\frac{2\pi}{N}1\cdot 1} & \cdots & e^{j\frac{2\pi}{N}1\cdot(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ e^{j\frac{2\pi}{N}(N-1)\cdot 0} & e^{j\frac{2\pi}{N}(N-1)\cdot 1} & \cdots & e^{j\frac{2\pi}{N}(N-1)\cdot(N-1)} \end{bmatrix}.$$

Here what we are going to prove specifically is that the following vector $\mathbf{v}_k$ (the kth column vector in $\mathbf{F}^{-1}$) is an *eigenvector* of $\mathbf{H}_c$.

$$\mathbf{v}_k := \begin{bmatrix} e^{j\frac{2\pi}{N}0\cdot k} \\ e^{j\frac{2\pi}{N}1\cdot k} \\ \vdots \\ e^{j\frac{2\pi}{N}(N-1)\cdot k} \end{bmatrix}.$$

To this end, we compute:

$$\mathbf{H}_c\mathbf{v}_k = \begin{bmatrix} h_0 & & & h_{L-1} & & & \cdots & h_1 \\ h_1 & h_0 & & & h_{L-1} & & \cdots & h_2 \\ h_2 & h_1 & h_0 & & & h_{L-1} & \cdots & h_3 \\ & & & \ddots & & & & \\ & & & & h_{L-1} & \cdots & h_1 & h_0 \end{bmatrix} \begin{bmatrix} e^{j\frac{2\pi}{N}0\cdot k} \\ e^{j\frac{2\pi}{N}1\cdot k} \\ \vdots \\ e^{j\frac{2\pi}{N}(N-1)\cdot k} \end{bmatrix}.$$

Consider the 0th component:

$$\begin{aligned} & h_0 e^{j\frac{2\pi}{N}0\cdot k} + h_1 e^{j\frac{2\pi}{N}(N-1)\cdot k} + h_2 e^{j\frac{2\pi}{N}(N-2)\cdot k} + \cdots + h_{L-1} e^{j\frac{2\pi}{N}(N-(L-1))\cdot k} \\ &= h_0 e^{-j\frac{2\pi\cdot 0k}{N}} + h_1 e^{-j\frac{2\pi\cdot k}{N}} + h_2 e^{-j\frac{2\pi\cdot 2k}{N}} + \cdots + h_{L-1} e^{-j\frac{2\pi\cdot(L-1)k}{N}} \\ &= \sum_{\ell=0}^{L-1} h_\ell e^{-j\frac{2\pi\ell k}{N}} =: \lambda_k \end{aligned}$$

where the first equality is due to $e^{j2\pi} = 1$. Similarly we can compute the 1st component as:

$$\begin{aligned} & h_0 e^{j\frac{2\pi}{N}1\cdot k} + h_1 e^{j\frac{2\pi}{N}(1-1)\cdot k} + h_2 e^{j\frac{2\pi}{N}(1+N-2)\cdot k} + \cdots + h_{L-1} e^{j\frac{2\pi}{N}(1+N-(L-1))\cdot k} \\ &= e^{j\frac{2\pi\cdot 1k}{N}} \left[h_0 e^{-j\frac{2\pi\cdot 0k}{N}} + h_1 e^{-j\frac{2\pi\cdot k}{N}} + h_2 e^{-j\frac{2\pi\cdot 2k}{N}} + \cdots + h_{L-1} e^{-j\frac{2\pi\cdot(L-1)k}{N}} \right] \\ &= e^{j\frac{2\pi\cdot 1k}{N}} \sum_{\ell=0}^{L-1} h_\ell e^{-j\frac{2\pi\ell k}{N}} \\ &= e^{j\frac{2\pi\cdot 1k}{N}} \lambda_k. \end{aligned}$$

Here the point is that it is a simple multiplication version of the 0th component λ_k (by a factor of $e^{j\frac{2\pi\cdot 1k}{N}}$). The similar computation leads us to the computation of the 2nd component: $e^{j\frac{2\pi\cdot 2k}{N}}\lambda_k$. Hence,

$$\mathbf{H}_c\mathbf{v}_k = \lambda_k \begin{bmatrix} e^{j\frac{2\pi}{N}0\cdot k} \\ e^{j\frac{2\pi}{N}1\cdot k} \\ \vdots \\ e^{j\frac{2\pi}{N}(N-1)\cdot k} \end{bmatrix} = \lambda_k \mathbf{v}_k.$$

This proves that $\mathbf{v}_k$ is indeed an eigenvector of $\mathbf{H}_c$ with an eigenvalue of λ_k.

Frequency-domain channel response Λ

Next let us figure out the explicit expression for λ_k that relates the kth frequency-domain transmitted signal $X[k]$ to the corresponding received signal counterpart $Y[k]$:

$$Y[k] = \lambda_k X[k] + W[k]. \tag{20}$$

Actually we already figured this out while proving (6). λ_k was the eigenvalue w.r.t. $\mathbf{v}_k$:

$$\lambda_k = \sum_{\ell=0}^{L-1} h_\ell e^{-j\frac{2\pi}{N}\ell k}. \tag{21}$$

Manipulating this in terms of the DFT matrix $\mathbf{F}$, we get:

$$\begin{aligned} \lambda_k &= \sqrt{N} \cdot \left\{ \frac{1}{\sqrt{N}} \sum_{\ell=0}^{N-1} h_\ell e^{-j\frac{2\pi}{N}\ell k} \right\} \\ &= \sqrt{N} \cdot [\mathbf{F}\mathbf{h}]_k \end{aligned} \tag{22}$$

where $\mathbf{h} := [h_0, \ldots, h_{L-1}, 0, \ldots, 0]^T$. We can then also view this as a *frequency-domain* channel response.

Look ahead

In Lecture 15, we argued that mapping information bits into frequency-domain transmitted signals $\tilde{\mathbf{x}}$ is simple. But it turns out this is not that simple. This is because the time-domain transmitted signal vector $\mathbf{x} = \mathbf{F}^{-1}\tilde{\mathbf{x}}$ may contain some *complex signals* which we have no idea about how to send over a practical channel. Next time, we will study details on this. We will also study how to design the optimal receiver accordingly.

May 27, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 17: OFDM (3/3)

Recap

During the past lectures, we investigated the OFDM technique that enables making the original ISI channel effectively ISI-free. The key idea of the technique was to generate "cyclic signaling" so as to yield the following beautiful relationship between $\mathbf{x} := [x[0], \ldots, x[N-1]]^T$ and $\mathbf{y} := [y[0], \ldots, y[N-1]]^T$:

$$\mathbf{y} = \mathbf{H}_c\mathbf{x} + \mathbf{w} \tag{1}$$

where $\mathbf{w} := [w[0], \ldots, w[N-1]]^T$ and $\mathbf{H}_c$ indicates a circulant matrix:

$$\mathbf{H}_c = \begin{bmatrix} h_0 & & & h_{L-1} & & & \cdots & h_1 \\ h_1 & h_0 & & & h_{L-1} & & \cdots & h_2 \\ h_2 & h_1 & h_0 & & & h_{L-1} & \cdots & h_3 \\ & & & \ddots & & & & \\ & & & & h_{L-1} & \cdots & h_1 & h_0 \end{bmatrix}. \tag{2}$$

Here L is the number of channel taps and N denotes the number of transmitted signals. The way to generate cyclic signaling was to send an augmented signal vector $\mathbf{x}' \in \mathbf{R}^{N+L-1}$ over a channel, as illustrated in Fig. 1.

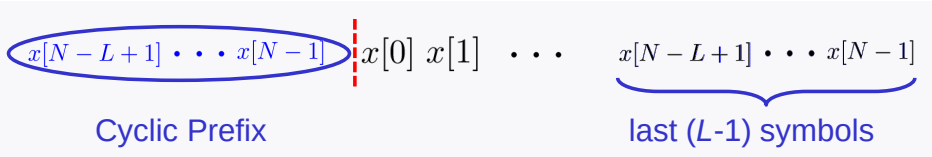


Figure 1: An augmented signal vector $\mathbf{x}'$ with cyclic prefix.

We then applied the nice property of the circulant matrix $\mathbf{H}_c = \mathbf{F}^{-1}\mathbf{\Lambda}\mathbf{F}$ into (1), thus obtaining the effective ISI-free channel that relates the frequency-domain transmitted signal vector $\tilde{\mathbf{x}} = \mathbf{F}\mathbf{x}$ to the frequency-domain received signal vector $\tilde{\mathbf{y}} = \mathbf{F}\mathbf{y}$ as follows:

$$\tilde{\mathbf{y}} = \mathbf{\Lambda}\tilde{\mathbf{x}} + \tilde{\mathbf{w}} \tag{3}$$

where $\tilde{\mathbf{w}} = \mathbf{F}\mathbf{w}$ and $\mathbf{\Lambda} := \mathsf{diag}(\lambda_0, \lambda_1, \ldots, \lambda_{N-1})$. Here the frequency-domain channel response λ_k reads:

$$\lambda_k = \sum_{\ell=0}^{L-1} h_\ell e^{-j\frac{2\pi}{N}\ell k}.$$

At the end of the last lecture, I then mentioned that an issue arises in mapping information bits into frequency-domain signal vector $\tilde{\mathbf{x}}$. The issue was that the time-domain signal vector $\mathbf{x} = \mathbf{F}^{-1}\tilde{\mathbf{x}}$ may contain some *complex-valued* signals which we have no idea about how to

send over a practical channel. So the question of interest is: How to ensure *real-valued* signal components for $\mathbf{x}$?

Today's lecture

Today we will study a mapping rule that ensures *real-valued* signal components for $\mathbf{x}$. We will then investigate encoding/decoding schemes in the context of the OFDM system. Lastly we will study the optimal receiver in-depth yet under a simple scenario.

How to map information bits

With the notations of $\tilde{\mathbf{x}} := [X[0], \ldots, X[N-1]]^T$, $\tilde{\mathbf{y}} := [Y[0], \ldots, Y[N-1]]^T$ and $\tilde{\mathbf{W}} := [W[0], \ldots, W[N-1]]^T$, one can write down all the components in (3) as:

$$\begin{aligned} Y[0] &= \lambda_0 X[0] + W[0]; \\ Y[1] &= \lambda_1 X[1] + W[1]; \\ &\vdots \\ Y[N-1] &= \lambda_{N-1} X[N-1] + W[N-1]. \end{aligned} \tag{4}$$

Since $Y[k]$'s are *complex* values in general, we can view these as $2N$ real subchannels. Denoting $W[k] := W_R[k] + W_I[k]$, one can readily verify that $W_R[k]$ and $W_I[k]$ are Gaussian. So an individual subchannel is an additive Gaussian channel that we are very much familiar with. This naturally leads us to map information bits into frequency-domain transmitted signals $X[k]$'s as an encoding rule. This way, we can apply every transmission/reception technique that we learned in Part I. See Fig. 2.

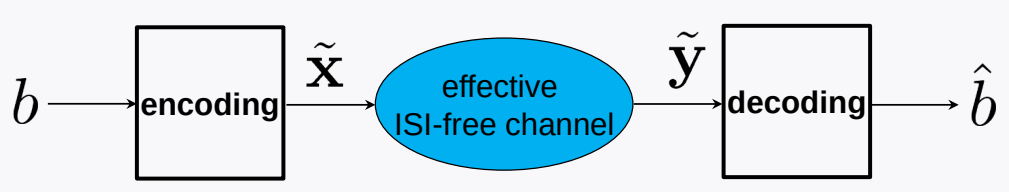


Figure 2: How to map information bits into frequency-domain signals in OFDM.

However, there is an issue in mapping information bits into $X[k]$'s. The problem is that we may have *complex-valued* signals for actually-transmitted signals $x[m]$'s although $x[m]$'s must be *real* values. For example, in a sequential communication scheme where we send $X[k] = \pm\sqrt{E}$ depending on the value of b_k, $x[m]$'s can often be complex numbers. So we need to worry about the condition on $X[k]$'s that ensures real-valued signals for $x[m]$'s. To find this, let us start by considering a necessary and sufficient condition for real-valued $x[m]$'s:

A condition for ensuring real-valued $x[m]$'s

The condition is obviously the following:

$$x[m] = x^*[m], \quad \forall m. \tag{5}$$

where * denotes a *complex conjugate*. Now what is then an equivalent condition w.r.t. $X[k]$'s? To figure this out, one can apply the IDFT on both sides in the above to obtain:

$$x[m] = \sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}mk} = \sum_{k=0}^{N-1} X^*[k] e^{-j\frac{2\pi}{N}mk} = x^*[m]. \tag{6}$$

Substituting $k = N - k$ in the right-hand-side of the above, we then get:

$$\sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}mk} = \sum_{k'=1}^{N} X^*[k] e^{j\frac{2\pi}{N}mk'}. \tag{7}$$

Here we used the fact that

$$\begin{aligned} e^{-j\frac{2\pi}{N}m(N-k')} &= e^{-j2\pi m} \cdot e^{j\frac{2\pi}{N}mk'} \\ &= 1 \cdot e^{j\frac{2\pi}{N}mk'}, \end{aligned}$$

which then yields:

$$\begin{aligned} x^*[m] &= \sum_{k=0}^{N-1} X^*[k] e^{-j\frac{2\pi}{N}mk} \\ &= \sum_{k'=1}^{N} X^*[N-k'] e^{j\frac{2\pi}{N}mk'}. \end{aligned}$$

An equivalent condition w.r.t. $X[k]$'s

Now we can readily obtain an equivalent condition w.r.t. $X[k]$'s by rewriting (7) as:

$$X[0] + \sum_{k=1}^{N-1} X[k] e^{j\frac{2\pi}{N}mk} = X^*[0] + \sum_{k'=1}^{N-1} X^*[N-k'] e^{j\frac{2\pi}{N}mk'}. \tag{8}$$

Hence, the condition can be summarized as:

$$\begin{aligned} &X[0] = X^*[0]; \\ &X[k] = X^*[N-k], \quad 1 \le k \le N-1. \end{aligned} \tag{9}$$

This implies that there are actually N real symbols that can be freely chosen out of $2N$ real symbols. Why? To see this clearly, let us consider a simple setting in which N is an even number. First express $X[k]$ as:

$$X[k] = X_R[k] + X_I[k], \qquad k = 0, 1, \ldots, N-1. \tag{10}$$

Then, as per the condition (9), we must respect:

$$\begin{aligned} X[0] &= X_R[0] \\ X[k] &= X_R[0] + jX_I[k], \qquad k = 1, \ldots, \frac{N}{2} - 1, \\ X\left[\frac{N}{2}\right] &= X_R\left[\frac{N}{2}\right]; \end{aligned} \tag{11}$$

and the other signals should be *fixed* as:

$$X[N-k] = X_R[k] + jX_I[k], \qquad k = 1, \ldots, \frac{N}{2} - 1. \tag{12}$$

We see that there are only N real-valued symbols that can be freely chosen. For instance, given the sequential communication scheme, the mapping rule between information bits and $X[k]$'s can be chosen as the one in Fig. 3. Note that only N real symbols are employed in mapping N

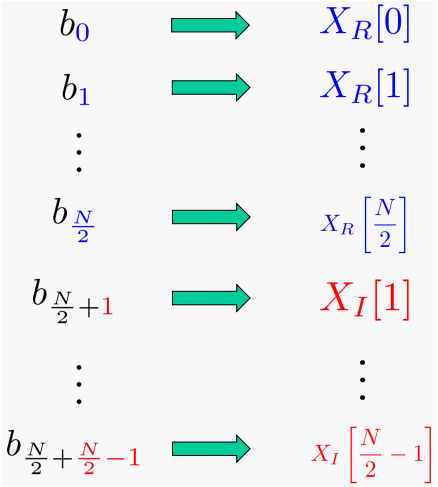


Figure 3: How to map information bits into frequency-domain signals in OFDM when employing a sequential communication scheme.

information bits.

Of course, we can apply any other coding schemes that we learned in Part I (such as repetition codes, LDPC codes, and Polar codes), as long as we satisfy the *real-valued signal constraint*, reflected in (9).

Optimal decision rule

Now what about decoding? For the rest of this lecture, we will explicitly develop the optimal receiver for a very simple scenario in which $N = 2$ and $L = 2$ and we use the sequential communication scheme. In this case, the number of so-called *effective* symbols (that can be freely chosen) is $N = 2$. So we map (b_0, b_1) to $(X_R[0], X_R[1])$, which in turn yields: $X[0] = X_R[0]$ and $X[1] = X_R[1]$. Now consider the received signals in the frequency domain:

$$\begin{aligned} Y[0] &= \lambda_0 X_R[0] + W[0]; \\ Y[1] &= \lambda_1 X_R[1] + W[1] \end{aligned} \tag{13}$$

where

$$\lambda_0 = \sum_{\ell=0}^{1} h_\ell e^{-j\frac{2\pi}{2}\ell\cdot 0} = h_0 + h_1; \qquad \lambda_1 = \sum_{\ell=0}^{1} h_\ell e^{-j\frac{2\pi}{2}\ell\cdot 1} = h_0 - h_1, \tag{14}$$

and the frequency-domain noise signals are:

$$\begin{aligned} W[0] &= \frac{1}{\sqrt{2}} \sum_{\ell=0}^{1} w[m] e^{-j\frac{2\pi}{2}m\cdot 0} = \frac{1}{\sqrt{2}}(w[0] + w[1]); \\ W[1] &= \frac{1}{\sqrt{2}} \sum_{\ell=0}^{1} w[m] e^{-j\frac{2\pi}{2}m\cdot 1} = \frac{1}{\sqrt{2}}(w[0] - w[1]). \end{aligned} \tag{15}$$

There is a good news w.r.t. the relationship between $W[0]$ and $W[1]$. The good news is that

$$(W[0], W[1]) \text{ are statistically independent.} \tag{16}$$

Why is good? The reason is that the independence also yields the independence between $Y[0]$ and $Y[0]$ and therefore the optimal decoder would be the local ML rule:

$$\hat{b}_k = \mathsf{ML}(b_k|Y[k]), \quad k = 0, 1.$$

Proof of (16)

Let us finally prove the good news of (16). We can readily prove this by using the definition of independence: the joint distribution $f_{W[0],W[1]}(W[0], W[1])$ is the product of individual distributions:

$$\begin{aligned} f_{W[0],W[1]}(W[0], W[1]) &\overset{(a)}{=} f_{w[0],w[1]}\left(\frac{W[0]+W[1]}{\sqrt{2}}, \frac{W[0]-W[1]}{\sqrt{2}}\right) \\ &\overset{(b)}{=} \frac{1}{2\pi\sigma^2}\exp\left(-\frac{1}{2\sigma^2}\left(\frac{(W[0]+W[1])^2}{2}+\frac{(W[0]-W[1])^2}{2}\right)\right) \\ &= \frac{1}{2\pi\sigma^2}\exp\left(-\frac{1}{2\sigma^2}\left(W^2[0]+W^2[1]\right)\right) \\ &= f_{W[0]}(W[0]) f_{W[1]}(W[1]) \end{aligned} \tag{17}$$

where (a) comes from (15); and (b) is due to the independence of $w[0]$ and $w[1]$.

Look ahead

In Parts I and II, we have studied channel modeling as well as transmission/reception strategies for the two channels: (i) AWGN channel (Part I); (ii) Wireline channel (Part II). In Part III, we will study applications of the key principles that we have learned into other interested fields beyond communications.

Lecture 18: Overview of Part III and supervised learning

Key principles covered in Parts I and II

In Parts I and II, we have learned a lot while investigating channel modeling as well as transmission/reception strategies for two representative channels in communications: (i) the AWGN channel (Part I); (ii) the Wireline channel (Part II). Among many tools and concepts that we have learned thus far, I would like to put a particular emphasis on the following key principles covered in the two parts:

1. Maximum A Posterior (MAP) decision rule;
2. Maximum Likelihood (ML) decision rule;
3. Viterbi algorithm.

The reason that I am emphasizing the principles is that these are shown to play crucial roles in many other fields beyond communications.

Goal of Part III

The goal of Part III is to demonstrate such roles in one recent spotlight field:

Machine learning

that you may be excited about. Specifically we will investigate: (i) Role of the MAP/ML principles in the design of one very popular methodology that arises in machine learning, called *supervised learning*; (ii) Role of the Viterbi algorithm in the design of one very important machine learning system, *speech recognition*.

For this and next lectures

For this and next lectures, we will dig into details regarding the role of the MAP/ML principles in the context of supervised learning. Specifically what we are going to do are three folded. First we will study what supervised learning is. We will then formulate an optimization problem that allows us to achieve the goal of supervised learning. Next we will demonstrate that the MAP/ML principles play an important role in the design of an optimization problem that ensures optimality in a certain sense (to be detailed later).

Machine learning

Let us start by figuring out what machine learning is. Machine learning is about an algorithm which is defined to be a set of instructions that a computer system can execute. Formally speaking, machine learning is the study of *algorithms* with which one can train a computer system so that the trained system can perform a specific task of interest. Pictorially, it means the following; see Fig. 1.

Here the entity that we are interested in building up is a computer system, which is definitely a *machine*. Since it is a system (i.e., a function), it has an input and an output. The input, usually denoted by x, indicates information which is employed to perform a task of interest.

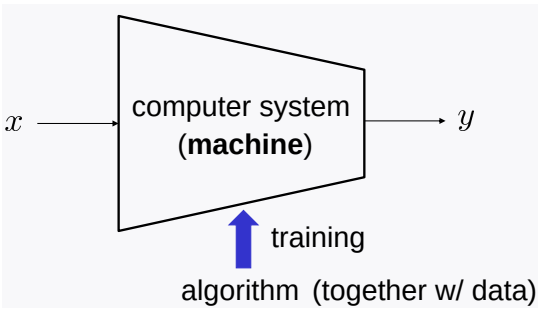


Figure 1: Machine learning is the study of algorithms which provide a set of explicit instructions to a computer system (machine) so that it can perform a specific task of interest. Let x be the input which indicates information employed to perform a task. Let y be the output that denotes a task result.

The output, usually denoted by y, indicates a task result. For instance, if a task of interest is legitimate-emails filtering against spam emails, then x could be multi-dimensional quantities[1]: (i) frequency of a keyword like *dollar signs* \$\$\$; (ii) frequency of another keyword, say *winner*. And y could be an email entity, e.g., $y = +1$ indicates a legitimate email while $y = -1$ denotes a spam email. In machine learning, such y is called a *label*. Or if an interested task is cat-vs-dog classification, then x could be *image-pixel values* and y is a binary value indicating whether the fed image is a cat (say $y = 1$) or a dog ($y = 0$).

Machine learning is about designing *algorithms*, wherein the main role is to train (teach) the computer system (machine) so that it can perform such task well. In the process of designing algorithms, we use something that is *data*.

Why called "Machine Learning"?

One can easily see the rationale of the naming via changing a viewpoint. From a *machine's perspective*, one can say that a *machine learns* the task from data. Hence, it is called *machine learning*, ML for short. This naming was coined in 1959 by Arthur Lee Samuel. See Fig. 2.

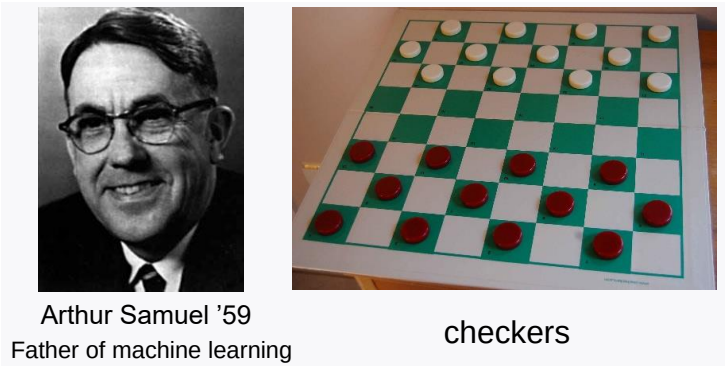


Figure 2: Arthur Lee Samuel is an American pioneer in the field of artificial intelligence, known mostly as the Father of machine learning. He found the field in the process of developing computer checkers which later formed the basis of AlphaGo.

[1]In machine learning, such quantity is called the *feature*. Usually this refers to a key component that well describes characteristics of data.

Arthur Samuel is actually one of the pioneers in the broader field of *Artificial Intelligence* (AI) which includes machine learning as a sub-field. The AI field is the study of creating *intelligence* by *machines*, unlike the *natural intelligence* displayed by intelligent beings like humans and animals. Since the ML field is about building up a human-like machine, it is definitely a sub-field of AI.

In fact, Arthur Samuel found the ML field in the process of developing a human-like computer player for a board game, called *checkers*; see the right figure in Fig. 1. He proposed many algorithms and ideas while developing computer checkers. It turns out those algorithms could form the basis of *AlphaGo*, a computer program for the board game Go which defeated one of the 9-dan professional players, Lee Sedol with 4 wins out of 5 games in 2016.

Mission of machine learning

Since ML is a sub-field of AI, its end-mission is *achieving artificial intelligence*. So in view of the block diagram in Fig. 1, the goal of ML is to design an algorithm so that the trained machine *behaves like intelligence beings*.

Supervised learning

There are some methodologies which help us to achieve the goal of ML. One specific yet very popular method is the one called:

Supervised Learning.

What supervised learning means is *learning* a function $f(\cdot)$ (indicating a functional of the machine) with the help of a *supervisor*. See Fig. 3.

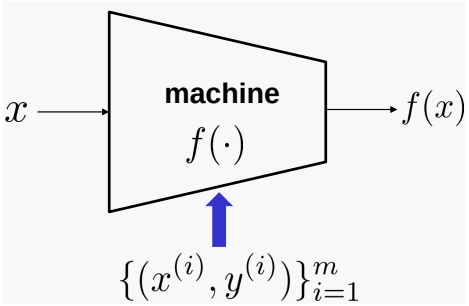


Figure 3: Supervised Learning: Design a computer system (machine) $f(\cdot)$ with the help of a supervisor which offers input-output pair samples, called a training dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$.

What the supervisor means in this context is the one who provides input-output samples. Obviously the input-output samples form the *data* employed for training the machine, usually denoted by:

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}, \tag{1}$$

where $(x^{(i)}, y^{(i)})$ indicates the ith input-output sample (or called a *training sample* or an *example*) and m denotes the number of samples. Using this notation (1), one can say that supervised learning is to:

$$\text{Estimate } f(\cdot) \text{ using the training samples } \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}. \tag{2}$$

Optimization

A common way to estimate $f(\cdot)$ is through *optimization*. To understand what this means, let us first explain what optimization is.

Optimization is to choose an *optimization variable* that minimizes (or maximizes) a certain quantity of interest possibly given some constraints. There are two important quantities in the definition. One is the optimization variable which affects the interested quantity and is subject to our design. This is usually a multi-dimensional quantity. The second is the certain quantity of interest that we wish to minimize (or maximize). This is called the *objective function* in the field, and an one-dimensional scalar.

Objective function

Now what is the *objective function* in the supervised learning framework? To figure this out, we need to know about the *objective* that supervised learning wishes to achieve. In view of the goal (2), what we want is obviously:

$$y^{(i)} \approx f(x^{(i)}), \quad \forall i \in \{1, \ldots, m\}.$$

A natural question that arises is then: How to quantify *closeness* (reflected in the "$\approx$" notation) between the two quantities: $y^{(i)}$ and $f(x^{(i)})$? One very common way that has been used in the field is to employ a function, called a *loss* function, usually denoted by:

$$\ell(y^{(i)}, f(x^{(i)})). \tag{3}$$

One obvious property that the loss function $\ell(\cdot, \cdot)$ should have is that it should be small when the two arguments are close, while being zero when the two are identical. Using such loss function (3), one can then formulate an optimization problem as:

$$\min_{f(\cdot)} \sum_{i=1}^{m} \ell(y^{(i)}, f(x^{(i)})). \tag{4}$$

How to introduce optimization variable?

Now what is *optimization variable*? Unfortunately, there is no variable. Instead we have a different quantity that we can optimize over: *the function* $f(\cdot)$, marked in red in (4). The question is then: How to introduce optimization variable? A common way employed in the field is to represent the function $f(\cdot)$ with *parameters (or called weights)*, denoted by w, and then consider such weights as an optimization variable. Taking this approach, one can then translate the problem (4) into:

$$\min_{w} \sum_{i=1}^{m} \ell(y^{(i)}, f_w(x^{(i)})) \tag{5}$$

where $f_w(x^{(i)})$ denotes the function $f(x^{(i)})$ parameterized by w.

The above optimization problem depends on how we define the two functions: (i) $f_w(x^{(i)})$ w.r.t. w; (ii) the loss function $\ell(\cdot, \cdot)$. In machine learning, lots of works have been done for the choice of such functions.

A choice for $f_w(\cdot)$

Around at the same time when the machine learning (ML) field was founded, one architecture was suggested for the function $f_w(\cdot)$ in the context of simple binary classifiers in which y takes one among the two options only, e.g., $y \in \{-1, +1\}$ or $y \in \{0, 1\}$. The architecture is called:

$$\textit{Perceptron},$$

and was invented in 1957 by one of the pioneers in the AI field, named Frank Rosenblatt. Interestingly, Frank Rosenblatt was a *psychologist*. He was interested in how brains of intelligent beings work and his study on brains led him to come up with the perceptron, and therefore gave significant insights into *neural networks* that many of you guys heard of.

How brains work

Here are details on how the brain structure inspired the perceptron architecture. Inside a brain, there are many *electrically excitable cells*, called *neurons*; see Fig. 4. Here a red-circled one

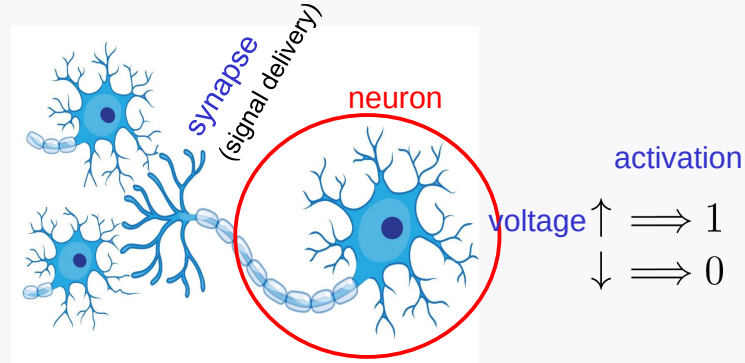


Figure 4: Neurons are electrically excitable cells and are connected through synapses.

indicates a neuron. So the figure shows three neurons in total. There are three major properties about neurons that led to the architecture.

The first is that a neuron is an *electrical* quantity, so it has a *voltage*. The second property is that neurons are connected with each other through mediums, called *synapses*. So the main role of synapses is to deliver electrical voltage signals across neurons. Depending on the connectivity strength level of a synapse, a voltage signal from one neuron to another can increase or decrease. The last is that a neuron takes a particular action, called *activation*. Depending on its voltage level, it generates an all-or-nothing pulse signal. For instance, if its voltage level is above a certain threshold, then it generates an impulse signal with a certain magnitude, say 1; otherwise, it produces nothing.

Perceptron

The above three properties led Frank Rosenblatt to propose the perceptron architecture, as illustrated in Fig. 5.

Let x be an n-dimensional real-valued signal: $x := [x_1, x_2, \ldots, x_n]^T$. Suppose each component x_i is distributed to each neuron, and let x_i indicates a voltage level of the ith neuron. The voltage signal x_i is then delivered through a synapse to another neuron (placed on the right in the figure, indicated by a big circle). Remember that the voltage level can increase or decrease depending on the connectivity strength of a synapse. To capture this, a weight, say w_i, is multiplied to x_i so $w_i x_i$ is a delivered voltage signal at the terminal neuron. Based on an empirical observation that the voltage level at the terminal neuron increases with more connected neurons, Rosenblatt

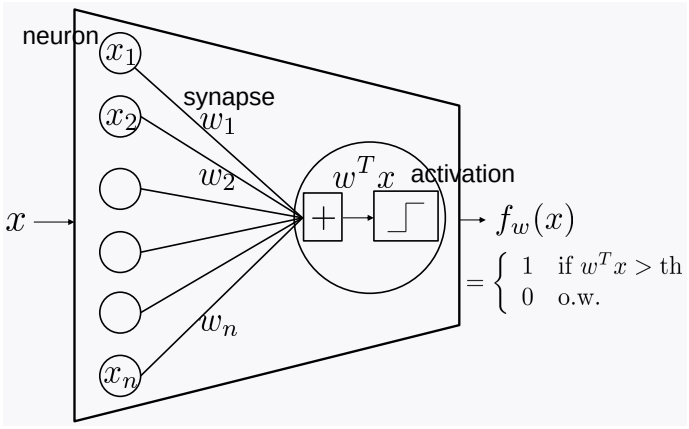


Figure 5: The architecture of perceptron.

introduced an adder which simply aggregates all the voltage signals coming from many neurons, so he modeled the voltage signal at the terminal neuron as:

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = w^T x. \tag{6}$$

Lastly in an effort to mimic the *activation*, he modeled the output signal as

$$f_w(x) = \begin{cases} 1 & \text{if } w^T x > \text{th}, \\ 0 & \text{o.w.} \end{cases} \tag{7}$$

where "th" indicates a certain threshold level. It can also be simply denoted as

$$f_w(x) = \mathbf{1}\{w^T x > \text{th}\}. \tag{8}$$

Activation functions

Taking the percentron architecture in Fig. 5, one can then formulate the optimization problem (5) as:

$$\min_w \sum_{i=1}^m \ell(y^{(i)}, \mathbf{1}\{w^T x^{(i)} > \text{th}\}). \tag{9}$$

This is an initial optimization problem that people came up with. However, people figured out there is an issue in solving this optimization. The issue comes from the fact that the objective function contains an indicator function, so it is *not differentiable*. It turns out non-differentiability makes the problem difficult to solve. This will be clearer soon in a later lecture. Please be patient until we get to the point. What can we do then? One typical way that people have taken in the field is to *approximate* the activation function. There are many ways for approximation. From below, we will investigate one of them.

Approximate the activation!

One popular way is to use the following function that makes a *smooth* transition from 0 to 1:

$$f_w(x) = \frac{1}{1 + e^{-w^T x}}. \tag{10}$$

Notice that $f_w(x) \approx 0$ when $w^T x$ is very small; it then grows exponentially with an increase in $w^T x$; later grows logarithmically; and finally saturates as 1 when $w^T x$ is very large. Actually the function (10) is a very popular one used in statistics, called the *logistic*[2] function. There is another name for the function, which is the *sigmoid*[3] function.

There are two good things about the logistic function. First it is differentiable. The second is that it can serve as the *probability* for the output in the binary classifier, e.g., $\Pr(y = 1)$ where y denotes the ground-truth label in the binary classifier. So it is interpretable.

Look ahead

Under the choice of the logistic activation, what is then a good choice for a *loss* function? It turns out the MAP/ML principles play an important role in the design of an *optimal* loss function in some sense. Next time we will investigate in what sense it is optimal. We will then figure out how the principles come up in the design of the optimal loss function.

[2]The word *logistic* comes from a Greek word which means a slow growth, like a logarithmic growth.
[3]Sigmoid means resembling the lower-case Greek letter sigma, S-shaped.

June 3, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 19: Logistic regression

Recap

Last time we formulated an optimization problem for supervised learning based on the percetron architecture:

$$\min_{w} \sum_{i=1}^{m} \ell(y^{(i)}, \hat{y}^{(i)}) \tag{1}$$

where $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$ indicate data-label paired examples; $\ell(\cdot, \cdot)$ denotes a loss function; and $\hat{y}^{(i)} := f_w(x^{(i)})$ is the prediction parameterized by the weights w. As an activation function, we considered a logistic function that is widely used in the field:

$$f_w(x) = \frac{1}{1 + e^{-w^T x}}. \tag{2}$$

I then claimed that the ML principle plays a crucial role in the design of the *optimal* loss function.

Today's lecture

Today we will prove this claim. Specifically what we are going to do are three-folded. We will first investigate what it means by the *optimal* loss function. We will next figure out how the ML principle comes up in the design of the optimal loss function. Lastly we will learn how to solve the formulated optimization.

Optimality in a sense of maximizing likelihood

A binary classifier with the logistic function (2) is called *logistic regression*. See Fig. 1 for illustration.

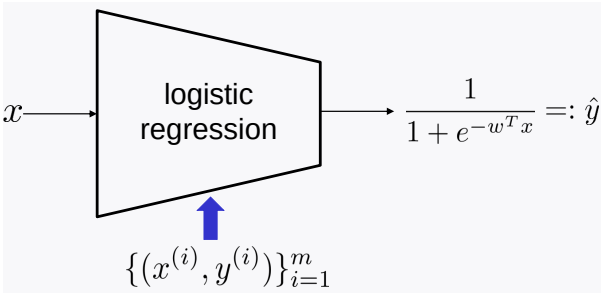


Figure 1: Logistic regression

Notice that the output $\hat{y}$ lies in between 0 and 1:

$$0 \leq \hat{y} \leq 1.$$

Hence, one can interpret the output as a *probability* quantity. Now the optimality of a classifier can be defined under the following assumption inspired by such interpretation:

$$\text{Assumption}: \hat{y} = \Pr(y = 1|x). \tag{3}$$

To understand what it means in detail, consider the *likelihood* of the ground-truth classifier:

$$\Pr\left(\{y^{(i)}\}_{i=1}^m \mid \{x^{(i)}\}_{i=1}^m\right). \tag{4}$$

Notice that the classifier output $\hat{y}$ is a function of weights w. Hence, we see that assuming (3), the likelihood (4) is also a function of w.

We are now ready to define the optimality of w. The optimal weight, say w^*, is defined as the one that yields the *maximum likelihood* (4):

$$w^* := \arg\max_w \Pr\left(\{y^{(i)}\}_{i=1}^m | \{x^{(i)}\}_{i=1}^m\right). \tag{5}$$

Similarly the *optimal loss function*, say $\ell^*(\cdot,\cdot)$ is defined as the one that satisfies:

$$\arg\min_w \sum_{i=1}^m \ell^*(y^{(i)}, \hat{y}^{(i)}) = \arg\max_w \Pr\left(\{y^{(i)}\}_{i=1}^m | \{x^{(i)}\}_{i=1}^m\right). \tag{6}$$

It turns out the condition (6) would give us the optimal loss function $\ell^*(\cdot,\cdot)$ that yields a very well-known machine learning classifier: *logistic regression*, in which the loss function is:

$$\ell^*(y, \hat{y}) = \ell_{\mathsf{logistic}}(y, \hat{y}) = -y \log y - (1-y)\log(1-y). \tag{7}$$

Let us now prove this.

Derivation of the optimal loss function $\ell^*(\cdot,\cdot)$

Usually samples are obtained from different data $x^{(i)}$'s. Hence, it is reasonable to assume that such samples are independent with each other:

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^m \text{ are independent over } i. \tag{8}$$

Under this assumption, we can then rewrite the likelihood (4) as:

$$\begin{aligned}
\Pr\left(\{y^{(i)}\}_{i=1}^m | \{x^{(i)}\}_{i=1}^m\right) &\overset{(a)}{=} \frac{\Pr\left(\{(x^{(i)}, y^{(i)})\}_{i=1}^m\right)}{\Pr\left(\{x^{(i)}\}_{i=1}^m\right)} \\
&\overset{(b)}{=} \frac{\prod_{i=1}^m \mathbb{P}\left(x^{(i)}, y^{(i)}\right)}{\prod_{i=1}^m \mathbb{P}(x^{(i)})} \\
&\overset{(c)}{=} \prod_{i=1}^m \mathbb{P}\left(y^{(i)}|x^{(i)}\right)
\end{aligned} \tag{9}$$

where (a) and (c) are due to the definition of conditional probability; and (b) comes from the independence assumption (8). Here $\mathbb{P}(x^{(i)}, y^{(i)})$ denotes the probability distribution of the input-output pair of the system:

$$\mathbb{P}(x^{(i)}, y^{(i)}) := \Pr(X = x^{(i)}, Y = y^{(i)}) \tag{10}$$

where X and Y indicate random variables of the input and the output, respectively.

Recall the probability-interpretation-related assumption (3) made with regard to $\hat{y}$:

$$\hat{y} = \Pr(y = 1|x).$$

This implies that:

$$\begin{aligned} y = 1: &\quad \mathbb{P}(y|x) = \hat{y}; \\ y = 0: &\quad \mathbb{P}(y|x) = 1 - \hat{y}. \end{aligned}$$

Hence, one can represent $\mathbb{P}(y|x)$ as:

$$\mathbb{P}(y|x) = \hat{y}^y (1 - \hat{y})^{1-y}.$$

Now using the notations of $(x^{(i)}, y^{(i)})$ and $\hat{y}^{(i)}$, we then get:

$$\mathbb{P}\left(y^{(i)}|x^{(i)}\right) = (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{1-y^{(i)}}.$$

Plugging this into (9), we get:

$$\Pr\left(\{y^{(i)}\}_{i=1}^m | \{x^{(i)}\}_{i=1}^m\right) = \prod_{i=1}^m (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{1-y^{(i)}}. \tag{11}$$

This together with (5) yields:

$$\begin{aligned} w^* &= \arg\max_w \prod_{i=1}^m (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{1-y^{(i)}} \\ &\overset{(a)}{=} \arg\max_w \sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \\ &\overset{(b)}{=} \arg\min_w \sum_{i=1}^m -y^{(i)} \log \hat{y}^{(i)} - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \end{aligned} \tag{12}$$

where (a) comes from the fact that $\log(\cdot)$ is a non-decreasing function and $\prod_{i=1}^m (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{1-y^{(i)}}$ is positive; and (b) is due to changing the sign of the objective function.

In fact, the term inside the summation in the last equality in (12) respects the formula of an important notion that arises in the field of information theory: *cross entropy*. In particular, in the context of a loss function, it is named the cross entropy loss:

$$\ell_{\mathsf{CE}}(y, \hat{y}) := -y \log \hat{y} - (1 - y) \log(1 - \hat{y}). \tag{13}$$

Hence, the optimal loss function that yields the maximum likelihood is the cross entropy loss:

$$\ell^*(\cdot, \cdot) = \ell_{\mathsf{CE}}(\cdot, \cdot).$$

Remarks on cross-entropy loss (13)

Let me say a few words about why the loss function (13) is called the *cross-entropy* loss. Actually this comes from the definition of *cross entropy*. The cross entropy is defined w.r.t. two random variables. For simplicity, let us consider two binary random variables, say $X \sim \mathsf{Bern}(p)$ and $Y \sim \mathsf{Bern}(q)$ where $X \sim \mathsf{Bern}(p)$ indicates a binary random variable with $p = \Pr(X = 1)$. For such two random variables, the cross entropy is defined as:

$$H(p, q) := -p \log q - (1 - p) \log(1 - q). \tag{14}$$

Notice that the formula of (13) is exactly the same as the term inside summation in (12), except for having different notations. Hence, it is called the *cross entropy loss*. In PS7, you will have a chance to know about the rationale behind the naming "cross entropy". If you want to know more about cross entropy, you may want to take the course on information theory: EE326.

How to solve (12)?

From (12), we can write the optimization problem as:

$$\min_w \sum_{i=1}^{m} -y^{(i)} \log \frac{1}{1+e^{-w^T x^{(i)}}} - (1-y^{(i)}) \log \frac{e^{-w^T x^{(i)}}}{1+e^{-w^T x^{(i)}}}. \tag{15}$$

Let $J(w)$ be the normalized version of the objective function:

$$J(w) := \frac{1}{m} \sum_{i=1}^{m} -y^{(i)} \log \hat{y}^{(i)} - (1-y^{(i)}) \log(1-\hat{y}^{(i)}). \tag{16}$$

It turns out the above optimization belongs to *convex optimization* which is one of very important optimization classes. Simply put, convex optimization means a class of optimization problems which can be efficiently solved on a computer. More formally, it is a class of problems wherein the objective is a *convex function*. Roughly speaking, here the convex function is defined as a bowl-shaped function as illustrated in Fig. 2. See Appendix for the formal definition.

In light of the definition of convex optimization, $J(w)$ is convex in optimization variable w. So for the rest of this lecture, we will prove the convexity of the objective function, and then discuss how to solve such convex optimization.

Proof of convexity

One can readily show that convexity preserves under addition (why? think about the definition of convex functions). So it suffices to prove the following two:

$$\text{(i)} - \log \frac{1}{1+e^{-w^T x}} \text{ is convex in } w;$$
$$\text{(ii)} - \log \frac{e^{-w^T x}}{1+e^{-w^T x}} \text{ is convex in } w.$$

Since the second function in the above can be represented as the sum of a linear function and the first function:

$$- \log \frac{e^{-w^T x}}{1+e^{-w^T x}} = w^T x - \log \frac{1}{1+e^{-w^T x}},$$

it suffices to prove the convexity of the first function.

Notice that the first function can be rewritten as:

$$- \log \frac{1}{1+e^{-w^T x}} = \log(1+e^{-w^T x}). \tag{17}$$

In fact, proving the convexity of (17) is a bit involved if one relies directly on the definition of convex functions. It turns out there is another way to prove. That is based on the computation of the second derivative of a function, called the Hessian. How to compute the Hessian? What is the dimension of the Hessian? For a function $f : \mathbf{R}^d \to \mathbf{R}$, the gradient $\nabla f(x) \in \mathbf{R}^d$ and the Hessian $\nabla^2 f(x) \in \mathbf{R}^{d \times d}$. If you are not familiar, check from the vector calculus course or from Wikipedia.

A well-known fact says that if the Hessian of a function is positive semi-definite (PSD)[1], then the function is convex. We will not prove this here. Don't worry about the proof, but do remember this fact. The statement itself is very instrumental. Here we will use this fact to prove the convexity of the function (17).

Taking a derivative of the RHS formula in (17) w.r.t. w, we get:

$$\nabla_w \log(1 + e^{-w^T x}) = \frac{-xe^{-w^T x}}{1 + e^{-w^T x}}.$$

This is due to a chain rule of derivatives and the fact that $\frac{d}{dz} \log z = \frac{1}{z}$, $\frac{d}{dz} e^z = e^z$ and $\frac{d}{dw} w^T x = x$. Taking another derivative of the above, we obtain a Hessian as follows:

$$\begin{aligned}
\nabla_w^2 \log(1 + e^{-w^T x}) &= \nabla_w \left(\frac{-xe^{-w^T x}}{1 + e^{-w^T x}} \right) \\
&\overset{(a)}{=} \frac{xx^T e^{-w^T x}(1 + e^{-w^T x}) - xx^T e^{-w^T x} e^{-w^T x}}{(1 + e^{-w^T x})^2} \\
&= \frac{xx^T e^{-w^T x}}{(1 + e^{-w^T x})^2} \\
&\succeq 0
\end{aligned} \tag{18}$$

where (a) is due to the derivative rule of a quotient of two functions: $\frac{d}{dz} \frac{f(z)}{g(z)} = \frac{f'(z)g(z) - f(z)g'(z)}{g^2(z)}$. Here you may wonder why $\frac{d}{dw}(-xe^{-w^T x}) = xx^T e^{-w^T x}$. Why not xx, $x^T x^T$ or $x^T x$ in front of $e^{-w^T x}$? One rule-of-thumb that I strongly recommend is to simply try all the candidates and choose the one which does not have a syntax error (matrix dimension mismatch). For instance, xx (or $x^T x^T$) is just an invalid operation. $x^T x$ is not a right one because the Hessian must be an d-by-d matrix. The only candidate left without any syntax error is xx^T! We see that xx^T has the single eigenvalue of $\|x\|^2$ (Why?). Since the eigenvalue $\|x\|^2$ is non-negative, the Hessian is PSD, and therefore we prove the convexity.

Gradient descent algorithm

Now how to solve the convex optimization problem (12)? Since there is no constraint in the optimization, w^* must be the *stationary* point, i.e., the one such that

$$\nabla J(w^*) = 0. \tag{19}$$

But there is an issue in deriving such optimal point w^* from the above. The issue is that analytically finding such point is not doable because it turns out there is no closed-form solution (check!). However, there is a good news. The good news is that there developed several algorithms which allow us to find such point efficiently without the knowledge of the closed-form solution. One prominent algorithm that has been widely used in the field is: the *gradient descent algorithm*.

Here is how the algorithm works. It is an *iterative* algorithm. Suppose that at the t-th iteration, we have an estimate of w^*, say $w^{(t)}$. We then compute the gradient of the function evaluated at the estimate: $\nabla J(w^{(t)})$. Next we update the estimate along a direction being *opposite* to the direction of the gradient:

$$w^{(t+1)} \longleftarrow w^{(t)} - \alpha \nabla J(w^{(t)}) \tag{20}$$

[1]We say that a *symmetric* matrix, say $Q = Q^T \in \mathbf{R}^{d \times d}$, is positive semi-definite if $v^T Q v \geq 0$, $\forall v \in \mathbf{R}^d$, i.e., all the eigenvalues of Q are non-negative. It is simply denoted by $Q \succeq 0$.

where $\alpha > 0$ indicates the stepsize (or called the learning rate). If you think about it, this update rule makes sense. Suppose $w^{(t)}$ is placed right relative to the optimal point w^*, as illustrated in Fig. 2.

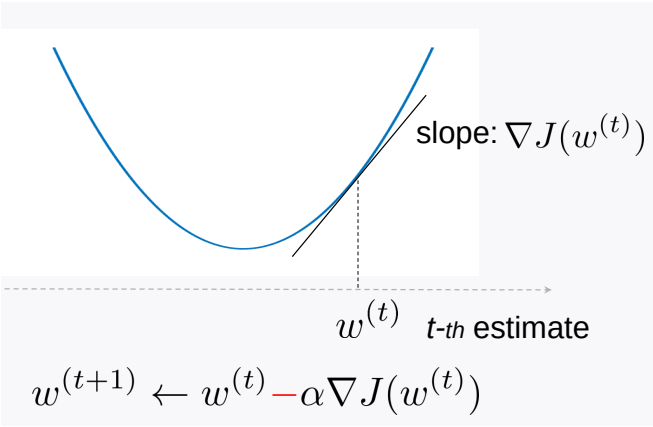


Figure 2: Gradient descent algorithm.

Then, we should move $w^{(t)}$ to the *left* so that it is closer to w^*. The update rule actually does this, as we *subtract* by $\alpha \nabla J(w^{(t)})$. Notice that $\nabla J(w^{(t)})$ points to the *right* direction given that $w^{(t)}$ is placed right relative to w^*. We repeat this procedure until it converges. It turns out: as $t \to \infty$, it actually converges:

$$w^{(t)} \longrightarrow w^*, \tag{21}$$

as long as the learning rate is chosen properly, like the one delaying exponentially w.r.t. t. We will not touch upon the proof of this convergence. Actually the proof is not that simple - even there is a big field in statistics which intends to prove the convergence of a variety of algorithms (if it is the case).

Look ahead

So far we have formulated an optimization problem for supervised learning, and found that the ML principle serves as a key component in the design of the optimal loss function. We also learned how to solve the problem via a famous algorithm, called gradient descent. This is the end of the first application. Next time, we will move onto the second application: *speech recognition*.

Appendix: Convex functions

An informal yet intuitive definition of a convex function is the following. We say that a function is convex if it is bowl-shaped, as illustrated in Fig. 3.

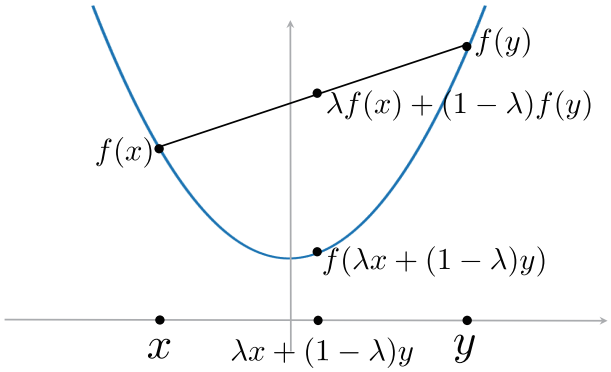


Figure 3: A geometric intuition behind a convex function.

Now what is the formal definition? The following observation can help us to easily come up with the definition. Take two points, say x and y, as in Fig. 3. Consider a point that lies in between the two points, say $\lambda x + (1-\lambda)y$ for $\lambda \in [0,1]$. Then, the bowl-shaped function suggests that the function evaluated at an λ-weighted linear combination of x and y is less than or equal to the same λ-weighted linear combination of the two functions evaluated at x and y:

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y). \tag{22}$$

This motivates the following definition. We say that a function f is convex if (22) holds for all $\lambda \in [0,1]$.

June 5, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 20: Speech recognition: Modeling

Recap

In Parts I & II, we have considered a communication problem wherein the goal is to deliver a sequence of information bits from a transmitter to a receiver over a channel. Two prominent channels are taken into consideration: (i) the additive white Gaussian channel (in Part I); the wireline ISI channel (in Part II). Specifically, what we did repeatedly throughout the past lectures is: Given an encoding strategy (e.g., the PAM-based sequential communication scheme, repetition coding, etc), we strive to achieve two things:

1. Derive the optimal receiver for decoding the bit string;
2. Analyze the error probability when using the optimal receiver.

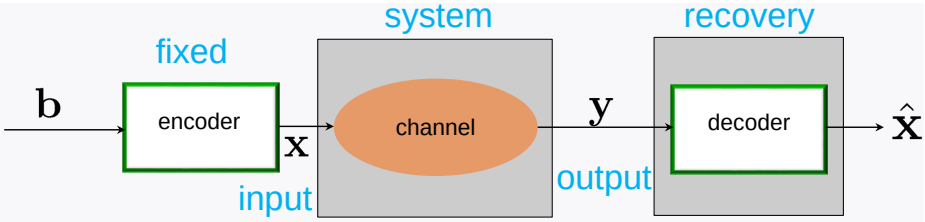


Figure 1: Viewing the communication problem as an inference problem.

With the one-to-one mapping relationship between the information bits and the transmitted signals $\mathbf{x} := [x[0], x[1], \ldots, x[n-1]]^T$, we see that decoding the bit string is equivalent to decoding $\mathbf{x}$. We can then view this communication problem as an *inference problem*, as the goal of the problem is now to infer input $\mathbf{x}$ from output $\mathbf{y}$. Note that any problem that aims to infer an input from an output is categorized into an inference problem, as long as the input and the output are related in a probabilistic manner. In the communication problem setting, the randomness coming from the channel dictates the probabilistic relation between the input and the output.

We have so far learned useful decoding principles such as MAP, ML and the Viterbi algorithm. It turns out that the MAP principle and the Viterbi algorithm play a crucial role in addressing many of the significant inference problems in other various applications. For the remaining lectures, we will demonstrate the tools from communication are indeed instrumental for one such interference problem: *speech recognition*.

Today's lecture

Today we will focus on proving the probabilistic relationship between the input/output of speech recognition systems. In other words, we will show that speech recognition is indeed an inference problem.

Speech recognition systems

Speech recognition is one of popular applications prevalent in our daily life. Siri in the iphone is one such example, aiming at speech recognition. We speak into the iphone microphone, which

samples the analog sound waveform. The iphone then tries to figure out what we spoke. The goal of speech recognition is to transform the analog waveform (comprising spoken words) into a written command, which can then be represented in the form of a *text* without losing the meaning of the spoken words. So what we wish to decode in speech recognition is a text. We will first show that speech recognition can be cast as an inference problem: decoding input $\mathbf{x}$ (text) from output $\mathbf{y}$ that has bearing on spoken words. See Fig. 2 for explicit details.

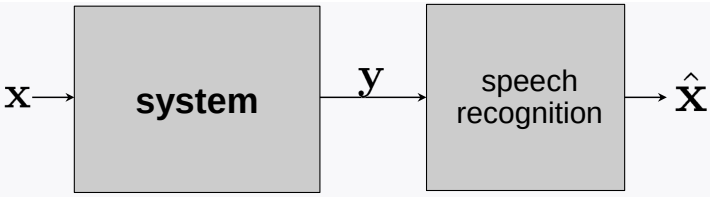


Figure 2: Viewing speech recognition as an inference problem.

In Fig. 2, the input $\mathbf{x}$ refers to a text that we intend to recognize in the speech recognition block. Now what are components that constitute the text $\mathbf{x}$? How does the text look like? Of course it depends on a spoken language. Here we will consider the English language.

Structure of an English text

In computer science and electrical engineering, **speech recognition** (SR) is the translation of spoken words into text. It is also known as "automatic speech recognition" (ASR), "computer speech recognition", or just "speech to text" (STT).

Some SR systems use "speaker-independent speech recognition"[1] while others use "training" where an individual speaker reads sections of text into the SR system. These systems analyze the person's specific voice and use it to fine-tune the recognition of that person's speech, resulting in more accurate transcription. Systems that do not use training are called "speaker-independent" systems. Systems that use training are called "speaker-dependent" systems.

Speech recognition applications include voice user interfaces such as voice dialling (e.g. "Call home"), call routing (e.g. "I would like to make a collect call"), domotic appliance control, search (e.g. find a podcast where particular words were spoken), simple data entry (e.g., entering a credit card number), preparation of structured documents (e.g. a radiology report), speech-to-text processing (e.g., word processors or emails), and aircraft (usually termed Direct Voice Input).

The term *voice recognition*[2][3][4] or *speaker identification*[5][6] refers to identifying the speaker, rather than what they are saying. Recognizing the speaker can simplify the task of translating speech in systems that have been trained on a specific person's voice or it can be used to authenticate or verify the identity of a speaker as part of a security process.

From the technology perspective, speech recognition has a long history with several waves of major innovations. Most recently, the field has benefited from advances in deep learning and big data. The advances are evidenced not only by the surge of academic papers published in the field, but more importantly by the world-wide industry adoption of a variety of deep learning methods in designing and deploying speech recognition systems. These speech industry players include Microsoft, Google, IBM, Baidu (China), Apple, Amazon, Nuance, IflyTek (China), many of which have publicized the core technology in their speech recognition systems being based on deep learning.

Figure 3: Structure of an English text.

A text is composed of a sequence of words, so one can view each word as a natural unit into which we can decompose the text. Actually we can further decompose the words into smaller units. For instance, consider a word "*speech*". One natural smaller unit that one can think of is an English alphabet (such as "s")? But there is an issue in adopting such a unit. The problem comes from the fact that an actual input to the speech recognition block is something related to *actual sound*. Here an issue is that the mapping between sound and alphabet is not one-to-one. For example, /i/ corresponds to "e" when we say "nike", but it may refer to "i" when we say "bit". Here the slash indicates a conventional notation used for representing phonemes ("eumso"

in Korean).

On the other hand, from a phonetic point of view, a word is decomposed into phonemes. For example, the word "*speech*" consists of four phonemes: /s/, /p/, /i/ and /ch/. The phoneme is indeed the smallest phonetic unit in a language. There are two types of phonemes: (1) consonants; (2) vowels. There are 44 phonemes (24 consonants and 20 vowels) in English. See Fig. 4.

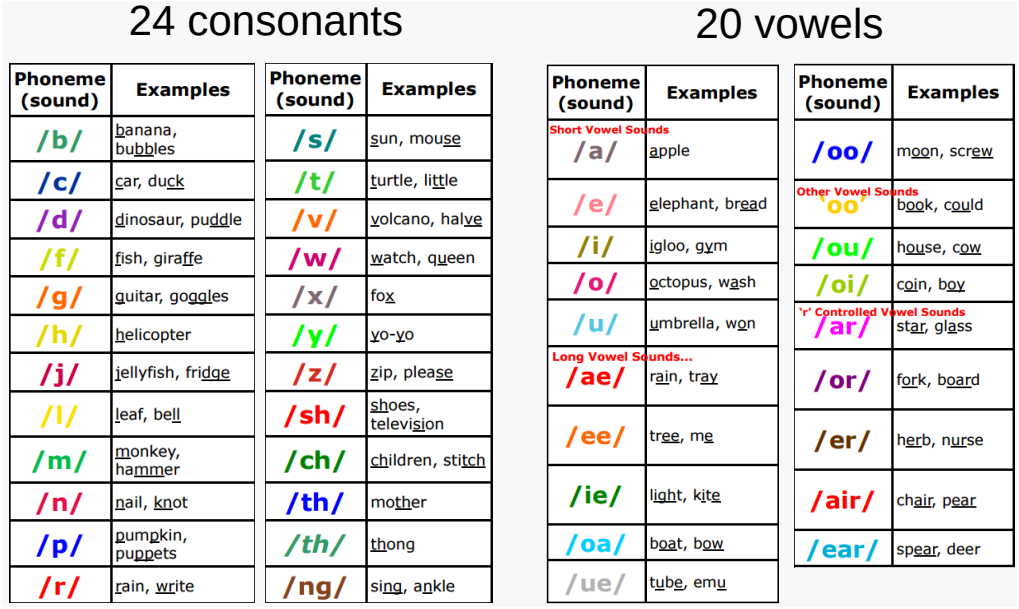


Figure 4: 44 phonemes in English: (1) 24 consonants; (2) 20 vowels.

In light of this, the speech recognition problem can be viewed as the problem of figuring out the sequence of phonemes that forms a text. The sequence $\mathbf{x}$ of phonemes is the one that we wish to decode, so the phonemes can be considered as random variables. Here we let $\mathbf{x}[m] \in \mathcal{X}$ be the mth phoneme of $\mathbf{x}$ where $\mathcal{X}$ is the set of phonemes whose alphabet size is $|\mathcal{X}| = 44$.

Output in speech recognition systems

Now let us figure out what the output is in the speech recognition system? The output $\mathbf{y}$ is the one that will be put into the speech recognition block, so it should reflect something related to actual sound. To figure out what it is, we first need to relate $\mathbf{x}$ to the actual sound signal that will be picked at the microphone in the system. A user who desires to say what the text $\mathbf{x}$ means speaks corresponding spoken words into the microphone, generating the analog waveform, say $y(t)$. This analog waveform is then translated into a sequence $\mathbf{y}$ of *discrete*-time signals through the following procedure.

Each phoneme roughly spans 10 ms. We chop the analog waveform into 10 ms intervals. We then take the signal in each 10 ms interval and extract some key features from it. It turns out that the relevant information contained in speech is most apparent in the frequency domain. So usually, a short window Fourier analysis is done on the sampled signal from each 10 ms time interval, and the corresponding Fourier coefficients are extracted. These coefficients serve as spectral information to describe the phoneme in that 10 ms interval. Typically there are multiple Fourier coefficients for the signal in each 10 ms interval. But let us simplify the story by assuming there is only one spectral component. Here we call that component a feature. We let $y[m]$ be the mth feature with respect to the mth phoneme. See Fig. 5 for the entire system

diagram that describes this procedure.

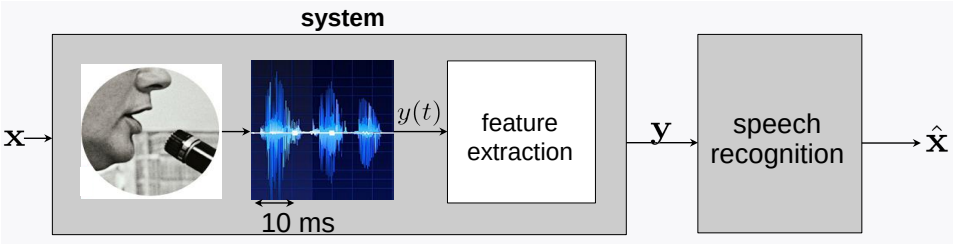


Figure 5: A block diagram of speech recognition.

Relation between x and y

Now how do $\mathbf{x}$ and $\mathbf{y}$ relate with each other? There is a lot of randomness involved in the system. Two major sources of the randomness are: (1) different voice characteristics (e.g., accent) and (2) noise (e.g., thermal noise due to random movements of electrons in the electrical circuit). This randomness induces uncertainty in $\mathbf{y}$, making the input and the output related probabilistically. Hence, we can view the speech recognition problem as an inference problem.

Optimal algorithm

Now a natural question that arises is: What is the speech recognition algorithm that optimally decodes $\mathbf{x}$ from $\mathbf{y}$? Suppose we say that an algorithm is optimal if it minimizes the probability of error $P_e := \Pr(\hat{\mathbf{x}} \neq \mathbf{x})$. Then, as we learned in Part I, the optimal algorithm is the MAP rule which finds the one that maximizes the a posterior probability:

$$\begin{aligned}\hat{\mathbf{x}}_{\mathsf{MAP}} &= \arg\max_{\mathbf{x}} \Pr(\mathbf{x}|\mathbf{y}) \\ &= \arg\max_{\mathbf{x}} \frac{\Pr(\mathbf{x},\mathbf{y})}{f(\mathbf{y})} \\ &= \arg\max_{\mathbf{x}} \Pr(\mathbf{x}) f(\mathbf{y}|\mathbf{x})\end{aligned}$$

when the second equality follows from the definition of conditional probability. One can easily see that to compute the MAP solution, we need to figure out two quantities: (1) A priori probability on the input: $p(\mathbf{x})$; (2) conditional pdf (likelihood function): $f(\mathbf{y}|\mathbf{x})$, which captures the relation between the input and the output. Here the speech recognition system can be considered as a channel by analogy with the communication problem.

Look ahead

Next time, we will study how to obtain these two and then will compute the MAP solution given the information. It turns out that there is a surprising parallel between the speech recognition and communication problems, allowing us to invoke the Viterbi algorithm which provides a computationally efficient way in computing the MAP solution.

Changho Suh (chsuh@kaist.ac.kr)

Lecture 21: Statistical modeling for speech recognition

Recap

Last time we showed that the speech recognition problem can be cast into an inference problem in which the goal of the problem is to decode a sequence $\mathbf{x}$ of phonemes (that allows us to recognize what a speaker says) from a sequence $\mathbf{y}$ of features (that have bearing on actual spoken words). See Fig. 1 for details.

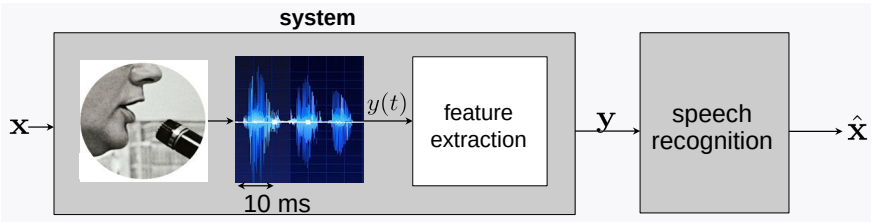


Figure 1: A block diagram of the speech recognition system and recovery block

We also found that there is a lot of randomness in the system due to different voice characteristics of a certain speaker and the system noise (e.g, thermal noise). The randomness is the one that makes the input and output of the system related probabilistically, demonstrating that the speech recognition problem is indeed an inference problem. We then derived the optimal MAP algorithm:

$$\hat{\mathbf{x}}_{\mathsf{MAP}} = \arg\max_{\mathbf{x}} p(\mathbf{x}) f(\mathbf{y}|\mathbf{x}). \tag{1}$$

Here the optimality is w.r.t. maximizing the reliability, quantified as $\Pr(\hat{\mathbf{x}} = \mathbf{x})$.

Today's lecture

In todays lecture, we will study how to obtain the quantities which are required to compute the MAP solution: (1) a priori knowledge $p(\mathbf{x})$; and (2) the likelihood function $f(\mathbf{y}|\mathbf{x})$. Next time we will use the two quantities to compute the MAP solution.

A sequence of $x[m]$'s (random process)

A naive approach to obtain $p(\mathbf{x})$ is investigating the quantity for each sequence of $\mathbf{x}$. However, this simple way comes with a challenge in computation. The reason is that the number of possible patterns of the sequence $\mathbf{x}$ grows exponentially with n:

$$|\mathcal{X}|^n = 44^n, \tag{2}$$

where $\mathcal{X}$ denotes a set of all the phonemes that each $x[m]$ can take on, usually called the alphabet. Here the equality comes from the fact that there are 44 phonemes in English: 24 consonants and 20 vowels. Note that the number of probability values required to fully specify $p(\mathbf{x})$ is huge for a large value of n, rendering it challenging to obtain the a priori knowledge. But it turns out there is a very nice statistical structure of the random process concerning the

sequence of phonemes. Exploiting the statistical structure, we are able to model the sequence of phonemes as a simple random process for which the joint distribution $p(\mathbf{x})$ can be specified with a much smaller number of parameters.

One naive way of modeling the sequence of phonemes is assuming that these random variables are independent. We saw an independent process before many times. Actually the additive white Gaussian noise that often appeared is an i.i.d. random process. However, this is not a good idea in this application. Some phonemes are more likely to follow other phonemes. For example, the phoneme /th/ is more likely to be followed by /e/ rather than /s/. So assuming the random variables to be independent seems like a very bad idea.

A simple dependency model: A Markov process

Now how are we going to capture the dependency of phonemes? It turns out we can capture the dependency, to some extent, with a simple dependency model, in which the dependency of $x[m]$ on the past is entirely through the random variable $x[m-1]$ that immediately precedes it. What it means is that

$$p(x[m]|x[m-1], x[m-2], \ldots, x[0]) = p(x[m]|x[m-1]). \tag{3}$$

Here the interpretation is that given the current state, say $x[m-1]$, the future $x[m]$ and the past $(x[m-1], \ldots, x[0])$ are conditionally independent. Actually the random process with this property was intensively explored by a very famous Russian mathematician, Andrey Markov. See Fig. 2. So named after him, this is called the Markov property, and a process with the

Andrey Markov 1906

Figure 2: Andrey Markov is a Russian mathematician who made important achievements w.r.t. the random process (later named a Markov process) that satisfies a simple dependency property reflected in (3).

property is called a Markov process. In fact, the dependency of phonemes in reality is much more complicated than that imposed by the simple Markov property. But it turns out that we can properly capture the dependency of a realistic sequence of phonemes by invoking a so-called generalized Markov process that is characterized by

$$p(x[m]|x[m-1], \ldots, x[m-\ell], x[m-\ell-1], \ldots, x[0]) = p(x[m]|x[m-1], \ldots, x[m-\ell]).$$

Observe that the dependency of $x[m]$ on the past is now through possibly more past states, say $\ell : x[m-1], x[m-2], \ldots, x[m-\ell]$. One can readily see that this is indeed a generalized Markov process, as it subsumes the Markov process as a special case of $\ell = 1$. For simplicity, we will assume that the sequence of phonemes is a single-memory Markov process.

A graphical model

In statistics and machine learning, this Markov property is usually represented by an insightful picture which illustrates the relation across random variables: called a *graphical model*. In fact, the graphical model is concerning a generic random process, say $(X_1, X_2, \ldots, X_n)$, not limited to the Markov process. It captures the statistical structure of a random process with two entities: (1) *nodes* (corresponding to random variables); (2) *edges* (reflecting the dependency of a pair of two random variables involved). The interpretation of a graph is as follows. If one disconnects the interested graph into two subgraphs $\mathcal{G}_1$ and $\mathcal{G}_2$ by removing a node X_i, then the random variables in $\mathcal{G}_1$ and $\mathcal{G}_2$ are independent conditional on X_i. For instance, consider (X_1, X_2, X_3) with $p(X_1, X_2, X_3) = p(X_1)p(X_2|X_1)p(X_3|X_2)$. One can easily see from the relation that $p(X_3|X_2, X_1) = p(X_3|X_2)$, implying that X_1 are X_3 are independent conditional on X_2. So the graph is illustrated as:

$$X_1 - X_2 - X_3. \tag{4}$$

Note that the removal of X_2 disconnects X_1 and X_3.

A priori probability $p(\mathbf{x})$

Going back to the sequence of phonemes, recall we assume that $x[m]$ is independent of all the past given $x[m-1]$. In this case, one can simply obtain the graphical model as:

$$x[0] - x[1] - x[2] - \cdots - x[n-2] - x[n-1]. \tag{5}$$

This model is called a Markov model. Or it is called a Markov chain as it looks like a chain. Using this statistical structure, one can now write down the joint distribution as:

$$\begin{aligned} p(\mathbf{x}) &= p(x[0], x[1], x[2], \ldots, x[n-1]) \\ &\overset{(a)}{=} p(x[0])p(x[1]|x[0])p(x[2]|x[1], x[0]) \cdots p(x[n-1]|x[n-2], \ldots, x[0]) \\ &\overset{(b)}{=} p(x[0])p(x[1]|x[0])p(x[2]|x[1]) \cdots p(x[n-1]|x[n-2]) \\ &= p(x[0]) \prod_{m=1}^{n-1} p(x[m]|x[m-1]) \end{aligned} \tag{6}$$

where (a) is due to the definition of condition probability; (b) comes from the Markov property.

Note that it suffices to know only $p(x[0])$ and $p(x[m]|x[m-1])$ to compute $p(\mathbf{x})$. So we need to specify only 44 values for $p(x[0])$ and 44^2 values for $p(x[m]|x[m-1])$. The sum $44 + 44^2$ is much smaller than the huge number 44^n required to specify the joint distribution when the statistical structure is not exploited. In reality, one can estimate individual pmf $p(x[0])$ and the transition probability $p(x[m]|x[m-1])$ from any large text by computing the following averages:

$$p(s) = \Pr(x[0] = s) \approx \frac{\text{\# of occurences of ``}s\text{''}}{\text{\# of phonemes in the interested text}}; \tag{7}$$

$$p(t|s) = \Pr(x[m] = t|x[m-1] = s) \approx \frac{\text{\# of ``}t\text{'' that follows ``}s\text{''}}{\text{\# of occurrences of ``}s\text{''}}. \tag{8}$$

By the law of large numbers[1], these estimates become concentrated around the ground-truth distributions as the number of phonemes in the text increases.

Likelihood function $f(\mathbf{y}|\mathbf{x})$

[1]This is one of very important and famous laws in mathematics and statistics. I assume that you may be familiar with this law dealt with in EE210. Otherwise, don't worry. You can simply consider it as a simple and intuitive law, saying that the *sample average* converges to the *true average* as the number of samples tends to infinity. A rigorous proof of this is not that difficult, once you rely on some important inequalities such as Markov inequality and Chebyshev inequality. If you feel offended, just forget about it for now.

Now let us figure out how to obtain the knowledge of the likelihood function. We find that a key observation on the system enables us to identify the statistical structure of the random process $\mathbf{y}$, thus providing a concrete way of computing $f(\mathbf{y}|\mathbf{x})$. Here the key observation is that $y[m]$ can be viewed as a noisy version of $x[m]$ and the noise has nothing to do with any other random variables involved in the system. The mathematical representation of the observation is that given $x[m]$, $y[m]$ is statistically independent of all the other random variables: e.g.,

$$y[1] \perp (x[0], x[1], \ldots, x[n-1], y[0]) | x[1]$$

This property leads to the graphical model for $\mathbf{y}$ as follows:

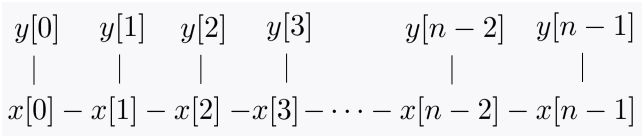


Figure 3: A Hidden Markov Model (HMM) for the speech recognition system output $\mathbf{y}$.

Notice that if we remove the node $x[m]$, the node $y[m]$ will be disconnected from the rest of the graph. This reflects the fact that $y[m]$ depends on other random variables only through $x[m]$. One interesting question about the model: Is the observation sequence $y[0], \ldots, y[n-1]$ a Markov model? No! Why? But the *underlying sequence* that we want to figure out is a Markov model. That is the reason as to why this is called the *Hidden Markov Model*, HMM for short.

Using this statistical property, one can now write down the conditional distribution as:

$$\begin{aligned}
f(\mathbf{y}|\mathbf{x}) &= f(y[0], y[1], \ldots, y[n-1] | x[0], x[1], \ldots, x[n-1]) \\
&= f(y[0]|x[0], x[1], \ldots, x[n-1]) f(y[1], \ldots, y[n-1] | x[0], x[1], \ldots, x[n-1], y[0]) \\
&\overset{(a)}{=} f(y[0]|x[0]) f(y[1], \ldots, y[n-1] | x[0], x[1], \ldots, x[n-1], y[0]) \\
&\quad \vdots \\
&\overset{(b)}{=} f(y[0]|x[0]) f(y[1]|x[1]) \cdots f(y[n-1]|x[n-1]) \\
&= \prod_{m=0}^{n-1} f(y[m]|x[m])
\end{aligned} \tag{9}$$

where (a) and (b) are because given $x[m]$, $y[m]$ is independent of everything else for any m.

Note that it suffices to know only $f(y[m]|x[m])$ to compute $f(\mathbf{y}|\mathbf{x})$. Now a question is: How can we obtain the knowledge of the individual likelihood function $f(y[m]|x[m])$. If the mth feature $y[m]$ is a *discrete* value, then we need to specify only the number $|\mathcal{X}| \times |\mathcal{Y}|$ of possible values for $f(y[m]|x[m])$. But the value of the feature is in general a *continuous* value although it can be quantized so that it can be represented by a discrete random variable. So we need to figure out the *functional relation* across $y[m]$ and $x[m]$ to specify the likelihood function.

Look ahead

It turns out there is a way to estimate $f(y[m]|x[m])$. Next time, we will learn how to estimate $f(y[m]|x[m])$. We will then employ the estimate to compute the MAP solution.

June 12, 2020
Changho Suh (chsuh@kaist.ac.kr)

Lecture 22: Viterbi algorithm & supervised learning

Recap

During the past few lectures, we modeled the speech recognition problem, as illustrated in Fig. 1. We let $x[m]$ be the mth phoneme and $y[m]$ be the corresponding feature extracted from the sound

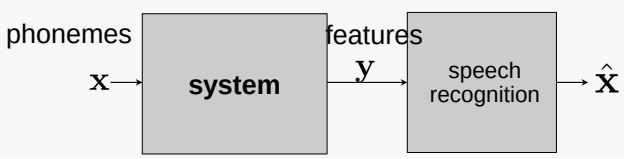


Figure 1: A block diagram of the speech recognition system and recovery block.

picked up at the microphone. In general, each $y[m]$ is complicated and multi-dimensional. For example, $y[m]$ can be a vector of Fourier coefficients of the signal. But to simplify matter, we assumed that $y[m]$ is a single continuous random variable. We then derived the optimal MAP algorithm associated with the inference problem:

$$\hat{\mathbf{x}}_{\mathsf{MAP}} = \arg\max_{\mathbf{x}} p(\mathbf{x}) f(\mathbf{y}|\mathbf{x}). \tag{1}$$

Last time, we have shown that the relation between the input and output of the system can be represented by the graphical model as illustrated in Fig. 2.

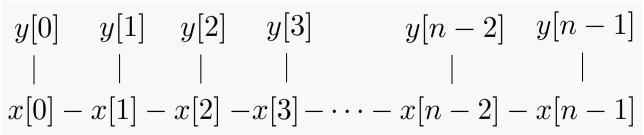


Figure 2: A graphical model for the speech recognition system.

Using this property, we could then identify the statistical structure of $\mathbf{x}$ and $\mathbf{y}$:

$$\begin{aligned} p(\mathbf{x}) &= p(x[0]) \prod_{m=1}^{n-1} p(x[m]|x[m-1]); \\ f(\mathbf{y}|\mathbf{x}) &= \prod_{m=0}^{n-1} f(y[m]|x[m]). \end{aligned} \tag{2}$$

We also learned how to estimate $p(x[0])$ and $p(x[m]|x[m-1])$ from a sample text containing sufficiently many words. But for $f(y[m]|x[m])$, I just claimed that there is a way to estimate.

Today's lecture

Today we will learn about the way to estimate the likelihood $f(y[m]|x[m])$. But prior to that, we will first finish computing the MAP solution (1), assuming the knowledge of $f(y[m]|x[m])$

together with $p(x[0])$ and $p(x[m]|x[m-1])$. To this end, we will first show the equivalence to the communication problem for the ISI channel. We will next show that the Viterbi algorithm can serve as an efficient way to find the MAP solution.

Viterbi algorithm

Using (2) and defining $p(x[0]|x[-1]=\emptyset) := p(x[0])$, we can write down the objective function in (1) as:

$$\begin{aligned} p(\mathbf{x})f(\mathbf{y}|\mathbf{x}) &= p(x[0]) \prod_{m=1}^{n-1} p(x[m]|x[m-1]) \prod_{m=0}^{n-1} f(y[m]|x[m]) \\ &= \prod_{m=0}^{n-1} p(x[m]|x[m-1]) \prod_{m=0}^{n-1} f(y[m]|x[m]) \\ &= \prod_{m=0}^{n-1} p(x[m]|x[m-1]) f(y[m]|x[m]). \end{aligned} \tag{3}$$

Taking a logarithm function at both sides, we then get:

$$\log\left(p(\mathbf{x})f(\mathbf{y}|\mathbf{x})\right) = \sum_{m=0}^{n-1} \log\left\{p(x[m]|x[m-1])f(y[m]|x[m])\right\}. \tag{4}$$

Now using the fact that $\log(\cdot)$ is a non-decreasing function, we can obtain another optimization problem equivalent to (1):

$$\hat{\mathbf{x}}_{\mathsf{MAP}} = \arg\min_{\mathbf{x}} \sum_{m=0}^{n-1} \log\left\{\frac{1}{p(x[m]|x[m-1])f(y[m]|x[m])}\right\}. \tag{5}$$

Letting $\mathbf{s}[m] = [x[m-1], x[m]]^T$, one can view the term inside the log function in (5) as a function depending solely on m and $\mathbf{s}[m]$. Note that the feature $y[m]$ is given, and the transition probability $p(\cdot|\cdot)$ and the likelihood function $f(y[m]|x[m])$ are assumed to be known. In fact, we already learned how to estimate the transition probability, and we are planning to study how to obtain the knowledge of the likelihood function later. And this interpretation motivates us to define the cost function as:

$$c_m(\mathbf{s}[m]) := \log\left\{\frac{1}{p(x[m]|x[m-1])f(y[m]|x[m])}\right\}. \tag{6}$$

Now noting one-to-one mapping relation between $\mathbf{x}$ and $\mathbf{s} := (s[0], s[1], \ldots, s[n-1])$, we can translate the problem (5) into the one that finds the sequence $\mathbf{s}$ of states:

$$\hat{\mathbf{s}}_{\mathsf{MAP}} = \arg\min_{\mathbf{s}} \sum_{m=0}^{n-1} c_m(\mathbf{s}[m]). \tag{7}$$

Here the key observation is that this problem is exactly the same as that we encountered in the two-tap ISI communication problem. The only difference is that in the communication problem, the cost function is defined in a different manner:

$$c_m(\mathbf{s}[m]) := \left(y[m] - \mathbf{h}^T\mathbf{s}[m]\right)^2 \quad \text{(communication problem)}.$$

Remember that there was a fascinating low-complexity algorithm that solves the optimization problem of the form (7) in a very efficient way: the *Viterbi algorithm*. Hence, we can simply run

the Viterbi algorithm to find $\hat{\mathbf{s}}_{\mathsf{MAP}}$. Recall that the complexity of Viterbi algorithm is roughly the order of the number of states times n. In the speech recognition problem, each $x[m]$ can take on 44 possible phonemes, so the number of possible states that $\mathbf{s}[m]$ can be in is 44^2. Therefore, the complexity of Viterbi algorithm, associated with the speech recognition problem setting, is:

$$\sim (\#\text{ of states}) \times n = 44^2 n.$$

Statistical modeling for $y[m]$'s

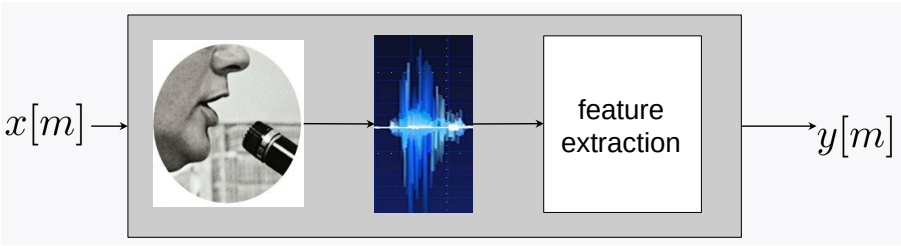


Figure 3: Speech recognition system.

So far we have assumed the knowledge of the likelihood function $f(y[m]|x[m])$. Now we will explain how to estimate this. Remember that the likelihood function depends on randomness that occurs in the system due to different voice characteristics and the noise.

If the system has no noise and a speaker is given, then we can think of $y[m]$ as simply a deterministic function of $x[m]$. For example, given $x[m] = /a/$, $y[m]$ is a deterministic function of $/a/$, say μ_a. However, due to the noise in the system, the feature $y[m]$ would be randomly distributed around the true feature μ_a corresponding to the phoneme $/a/$. Notice that the major source of this noise is the random movement of electrons due to heat, known as the thermal noise. We learned in Part I that the thermal noise can be modeled as an additive white Gaussian noise (AWGN). Hence, given $x[m] = /a/$, we can model $y[m]$ as:

$$y[m] = \mu_a + w[m] \tag{8}$$

where $w[m]$'s are i.i.d. $\sim \mathcal{N}(0, \sigma^2)$. Similarly given $x[m] = /b/$, $y[m]$ can be modeled as the true feature concerning $/b/$, say μ_b, plus an additive Gaussian noise. Here for simplicity, we will assume that we know the variance of the additive noise σ^2. Actually there are a variety of ways to estimate σ^2. One simple way is to infer the variance from multiple measurements $y[m]$'s fed by null signals. More concretely, suppose $x[m] = \emptyset$ for $m = 0, 1, \ldots, n-1$. Then, under the statistical modeling on $y[m]$ as above, we get:

$$y[m] = w[m], \quad m = 0, 1, \ldots, n-1. \tag{9}$$

Now then the empirical mean of $y^2[m]$'s can be used as a good surrogate of σ^2:

$$\frac{y^2[0] + y^2[1] + \cdots + y^2[n-1]}{n} \approx \sigma^2. \tag{10}$$

For a large value of n, the empirical mean is expected to concentrate around the variance, and especially as n tends to infinity, it does converge to the variance, by the law of large numbers.

Even if we know the noise variance, the likelihood function $f(y[m]|x[m])$ is not specified yet. This is because the true features such as μ_a and μ_b are *user-specific* parameters, so these can be viewed as *random variables*. Hence, these need to be estimated to fully specify the likelihood

function. There is one very popular approach that enables the estimation. For the rest of this lecture, we will study that approach.

Supervised learning

Assume for a moment that we knew the sequence of phonemes. This sounds like an absurd assumption given that the sequence of phonemes is exactly what we would like to infer! However, we could ask the speaker to say some predetermined phonemes for us at the beginning. In fact, some speech recognition software does this. Then we could use this sequence to estimate μ_a and μ_b based on the known phonemes. Note that in this method, we are learning the parameters by instructing (or *supervising*) the speaker to do something at the beginning. In other words, the parameters are learned in a *supervising* way. So this learning approach is *supervised learning*.

Here is how supervising learning works. Suppose we ask the user to say "*aaaaaaaa*" for eight time slots. This then gives us:

$$y[m] = \mu_a + w[m], \quad m = 0, 1, \ldots, 7. \tag{11}$$

Here notice that μ_a is the *mean* of the samples $y[m]$'s since $w[m]$ has zero mean. So one reasonable approach to estimate the parameter μ_a is taking the empirical mean:

$$\hat{\mu}_a = \frac{y[0] + y[1] + \cdots + y[7]}{8}. \tag{12}$$

Now a question is: Is this estimate optimal? It turns out the answer is yes! To see this, let us consider the optimal estimate. As in Part I and Part II, we say that an estimate is optimal if it maximizes the *reliability*. But here is the conventional definition of the reliability $\Pr(\hat{\mu}_a = \mu_a)$ proper? No! The reason is that the definition leads the reliability to take always zero no matter what an estimate is. Note that the entity μ_a that we wish to infer takes a *continuous* value. To resolve this, we employ a so-called ϵ-reliability:

$$\epsilon\text{-reliability} := \Pr(\mu_a - \epsilon \leq \hat{\mu}_a \leq \mu_a + \epsilon). \tag{13}$$

Note that this quantity can be strictly positive for $\epsilon > 0$. One can easily verify that the estimate that maximizes the ϵ-reliability in the limit of $\epsilon \to 0$ is an MAP estimate:

$$\begin{aligned} \hat{\mu}_a^{\mathsf{MAP}} &= \arg\max_{\mu_a} f_a(\mu_a|\mathbf{y}) \\ &= \arg\max_{\mu_a} f_a(\mu_a) f(\mathbf{y}|\mu_a). \end{aligned} \tag{14}$$

Check in PS8. Here we encounter a challenge. The challenge is that we know nothing about $f_a(\mu_a)$. One reasonable way to proceed in this case is assuming a uniformly distributed density for $f_a(\mu_a)$. Making this assumption, we can then simplify the MAP estimate as the ML estimate:

$$\hat{\mu}_a^{\mathsf{MAP}} = \arg\max_{\mu_a} f(\mathbf{y}|\mu_a) =: \hat{\mu}_a^{\mathsf{ML}}. \tag{15}$$

Now using the fact that $y[m] \sim \mathcal{N}(\mu_a, \sigma^2)$, we can explicitly compute the ML estimate as follows:

$$\begin{aligned} \hat{\mu}_a^{\mathsf{ML}} &= \arg\max_{\mu_a} f(\mathbf{y}|\mu_a) \\ &= \arg\max_{\mu_a} \exp\left(-\frac{1}{2\sigma^2}\sum_{m=0}^{7}(y[m] - \mu_a)^2\right) \\ &= \arg\min_{\mu_a} \sum_{m=0}^{7}(y[m] - \mu_a)^2. \end{aligned} \tag{16}$$

Taking the derivative of the objective function w.r.t. μ_a and setting it to 0, we finally get:

$$\hat{\mu}_a^{\mathsf{ML}} = \frac{y[0] + y[1] + \cdots + y[7]}{8}$$

which coincides with the good estimate that we expected earlier.

Closing

Finally I would like to leave a remark which may be helpful for your future careers. One key message that we could arrive at from Part III is that fundamental concepts (the MAP/ML principles and the Viterbi algorithm) play important roles. So in view of this, I strongly recommend you to make every efforts for *being strong at fundamentals!* The fundamentals that I would like to put a special emphasis on are w.r.t. modern AI technologies that you may be very interested in. As you may image, being an expert in the AI field requires many backgrounds. One major background that I believe crucial is *mathematics*, in particular four fundamental branches in mathematics.

The first is obviously *optimization*. Remember that the goal of machine learning can be achieved through optimization. The second is a field which provides instrumental tools with which one can translate the objective function and/or constraints into simple and tractable formulas. The field is: *linear algebra*. As you may be familiar with, many seemingly-complicated mathematical formulas can be expressed as simple terms that involve matrix multiplications and additions. The third is a field that plays a role in dealing with uncertainty that appears in random quantities. The field is: *probability*. The last is a field that serves to shed optimal architectural insights into machine learning models of interest. That is: *information theory*. Remember the role of cross entropy (an information-theoretic notion) in the design of the optimal loss function.

These are the very fundamentals which I believe are crucial for advancing the 4th industrial revolution empowered by AI technologies. So my advice is: Be strong at these fundamentals. Here are some relevant courses offered at KAIST:

1. EE424: Introduction to Optimization;
2. MA109: Introduction to Linear Algebra;
3. EE210: Introduction to Probability & Random Processes;
4. EE326: Introduction to Information Theory & Coding.

One caveat here is that such fundamentals are highly likely to be built only when you are at school. Of course it is a bit exaggerated, but it seems indeed the case according to the experiences of my own and many others. You may be able to understand what this means *after* you graduate; not enough time would be given for you to deeply understand some principles and also your stamina would not be as good as that of now.