Machine Learning - CPEN 355

Lecture Notes

Souradeep Dutta
(pronounced as Show-Ro-Deep Duh-tah)

Contents

Chapter 1

What is intelligence ?

Reading :

1. "A logical calculus of the ideas immanent in nervous activity " by McCulloch and Pitts (1943)
2. "Computing machinery and intelligence" by Alan Turing in 1950 (Turing, 2009).

What is intelligence? It is hard to define, I don't know a good definition. We certainly know it when we see it. All humans are intelligent. Dogs are plenty intelligent. Most of us would agree that a house fly or an ant is less intelligent than a dog. What are the common features of these species? They all can gather food, search for mates and reproduce, adapt to changing environments and, in general, the ability to survive. Are plants intelligent? Plants have sensors, they can measure light, temperature, pressure etc. They possess reflexes, e.g., sunflowers follow the sun. This is an indication of "reactive/automatic intelligence". The mere existence of a sensory and actuation mechanism is not an indicator of intelligence. Plants cannot perform planned movements, e.g., they cannot travel to new places.
A Tunicate in Fig. 1.1 is an interesting plant however. Tunicates are invertebrates. When they are young they roam around the ocean floor in search of

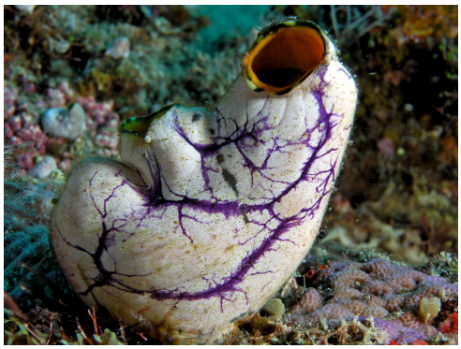


Figure 1.1: A Tunicate on the ocean floor

nutrients, and they also have a nervous system (ganglion cells) at this point of time that helps them do so. Once they find a nutritious rock, they attach themselves to it and then eat and digest their own brain. They do not need it anymore. They are called "tunicates" because they develop a thick covering (shown above) or a "tunic" to protect themselves.

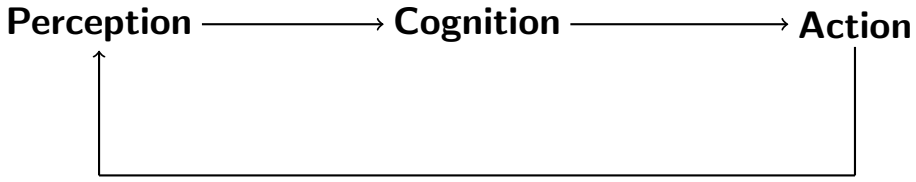
Is a program like AlphaGo intelligent? There is a very nice movie on Netflix on the development of AlphaGo and here's an excerpt from the movie. The commentator in this video is wondering how Lee Se-dol, who was one of the most accomplished Go players in the world then, might defeat this very powerful program; this was I believe after AlphaGo was up 3-0 in the match already. The commentator says so very nonchalantly: if you want to defeat AlphaGo all you have to do is pull the plug. A key indicator of intelligence (and this is just my opinion) is the ability to take actions upon the world. With this comes the ability to affect your environment, preempt antagonistic agents in the environment and take actions that achieve your desired outcomes. You should not think of intelligence (artificial or otherwise) as something that takes a dataset and learns how to make predictions using this dataset. For example, if I dropped my keys at the back of the class, I cannot possibly find them without moving around, using priors of where keys typically hide (which is akin to learning from a dataset) only helps us search more efficiently.

Is an LLM based software tool like ChatGPT/Gemini intelligent ? The short answer is no. ChatGPT does not understand meaning. It produces language by detecting and reproducing statistical patterns in data, not by forming beliefs, intentions, or comprehension about the world. Intelligence requires understanding ChatGPT has none. Philosopher John Searle, (who recently passed away in 2025 !) proposed the Chinese room argument. Consider the following thought experiment : A person who does not understand Chinese sits in a room. He receives Chinese symbols and follows a rulebook to produce correct responses. To outsiders it looks like the person understands Chinese. But

inside, there is no understanding - only *rule-following*. No matter how sophisticated this rule gets, it is still a rule. In my opinion we should not mistake this for intelligence. Just like no matter how good a store mannequin gets at mimicking human gestures, we can always tell the difference between a real human and a robot.

1.1 Key Components of Intelligence

With this definition, we can write down the three key parts that an intelligent, autonomous agent possesses as follows.



Perception refers to the sensory mechanisms to gain information about the environment (eyes, ears, smell, tactile input etc.). Action refers to your hands, legs, or motors/engines in machines that help you move on the basis of this information. Cognition is kind of the glue in between. It is in charge of crunching the information of your sensors, creating a good "representation" of the world around you and then undertaking actions based on this representation. The three facets of intelligence are not sequential and intelligence is not merely a feed-forward process. Your sensory inputs depend on the previous action you took. While searching for something you take actions that are explicitly designed to give you different sensory inputs than what you are getting at the moment.
This class will focus on learning. It is a component, not the entirety, of cognition.

> Learning is in charge of looking at past data and predicting what future data may look like.

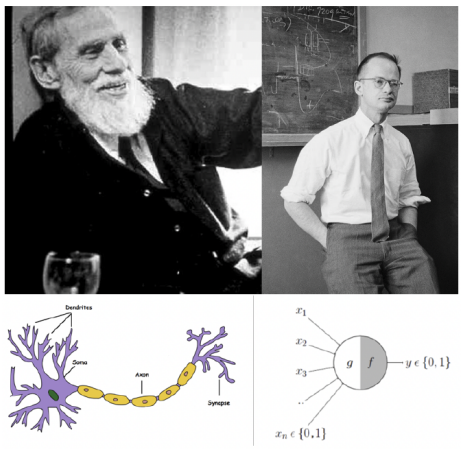
Cognition also involves handling situations when the current data does not match past data, etc. To give you an example, arithmetic problems you solved in elementary school are akin to learning. Whereas figuring out that taking a standard deduction when you file your income tax versus itemized deduction is like cognition. The objective of the learning process is really to crunch past data and learn a prior.
Imagine a supreme agent which is infinitely fast, clever, and can interpret its sensory data and compute the best actions for any task, say driving. Learning from past data is not essential for such an agent; effectively the supreme agent can simulate every physical process around it quickly and 30 decide upon the

best action it should take. Past data helps if you are not as fast as the supreme agent or if you want to save some compute time/energy during decision making.

> A deep network or a machine learning model is not a mechanism that directly undertakes the actions. It is rather a prior on the possible actions to take. Other algorithms that rely on real-time sensory data will be in charge of picking one action out of these predictions. This is very easy to appreciate in robotics: how a car should move depends more upon the real-time data than any amount of past data. This aspect is less often appreciated in non-robotics applications but it holds there as well. Even for something like a recommendation engine that recommends movies in Netflix, the output of a prediction model will typically be modified by a number of algorithms before it is actually recommended to the user, e.g., filters for sensitive information, or toxicity in a chatbot.

1.2 Intelligence : The Beginning (1942-50)

Let us give a short account of how our ideas about intelligence have evolved.

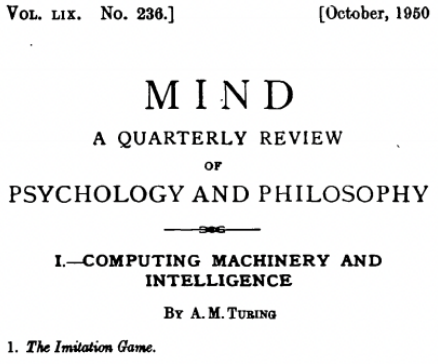


A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

WARREN S. MCCULLOCH AND WALTER PITTS
University of Illinois, College of Medicine,
Department of Psychiatry at the Illinois Neuropsychiatric Institute,
University of Chicago, Chicago, U.S.A.

The story begins roughly in 1942 in Chicago. These are Warren McCulloch who was a neuroscientist and Walter Pitts who studied mathematical logic. They built the first model of a mechanical neuron and propounded the idea that simple elemental computational blocks in your brain work together to per-

form complex functions. Their paper(McCulloch and Pitts, 1943) is an assigned reading for this lecture.



VOL. LIX. No. 236.] [October, 1950

MIND

A QUARTERLY REVIEW

OF

PSYCHOLOGY AND PHILOSOPHY

I.—COMPUTING MACHINERY AND INTELLIGENCE

BY A. M. TURING

1. *The Imitation Game.*

Around the same time in England, Alan Turing was forming his initial ideas on computation and neurons. He had already published his paper on computability by then. This paper(Turing, 2009) is the second assigned reading for this lecture.
McCulloch was inspired by Turing's idea of building a machine that could compute any function in finitely-many steps. In his mind, the neuron in a human brain, which either fires or does not fire depending upon the stimuli of the other neurons connected to it, was a binary object; rules of logic 10 where a natural way to link such neurons, just like the Pitt's hero Bertrand Russell rebuilt modern mathematics using logic. Together, McCulloch & Pitts' and Turing's work already had all the terms of neural networks as we know them today: nonlinearities, networks of a large number of neurons, training the weights in situ etc. Let's now move to Cambridge, Massachusetts. Norbert Wiener, who was a famous professor at MIT, had created a little club of enthusiasts around 1942. They would coin the term "Cybernetics" to study exactly the perception-cognition-action loop we talked about. You can read more in the original book titled "Cybernetics: or control and communication in the animal and the machine" (Wiener, 1965). You can also look at the book "The Cybernetic Brain" (Pickering, 2010) to read more.

1.2.1 Representation Learning

Perceptual agents, from plants to humans, perform measurements of physical processes ("signals") at a level of granularity that is essentially continuous. They also perform actions in the physical space, which is again continuous. Cognitive science on the other hand thinks in terms of discrete entities like concepts, ideas, objects, or categories. These can be manipulated with tools of logic and inference. It is useful to ask what information is transferred from the perception system to the cognition system to create such symbols from signals,

or from cognition to control which creates back signals from the symbols? We will often call these symbols the "internal representation" of an agent.

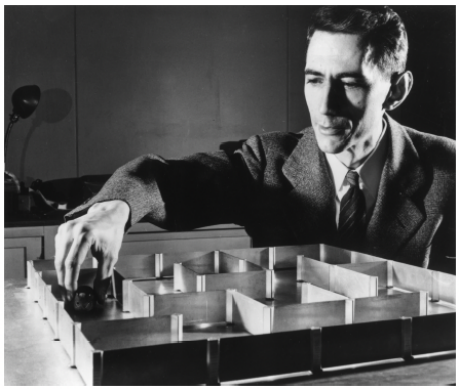


Figure 1.2: Claude Shannon studied information theory. This is a picture of a maze solving mouse that he made around 1950, among the world's first examples of machine learning; read more here

Claude Shannon formulated information theory which is one way to study these kind of ideas. Shannon devised a representation learning scheme for compressing (e.g., taking the intensities at each pixel of the camera and encoding them into something less redundant like JPEG), coding (adding redundancy into the representation to gain resilience to noise before transmitting it across some physical medium such as a wireless channel), decoding (using the redundancy to guess the parts of the data 5 packet that were corrupted during transmission) and finally decompressing the data (getting the original signal back, e.g., pixel intensities from JPEG). Information theory as described above is a tool to transmit data correctly between a sender and a receiver. We will use this theory for a different purpose. Compression, decompression etc. care about never 10 losing information from the data; machine learning necessarily requires you forget some of the data. If the model focuses too much on the grass next to the dogs in the dataset, it will "over-fit" to the data and next time when you see grass, it will end up predicting a dog. It not easy to determine which parts of the data one should forget and which parts one should remember.

The study of artificial intelligence has always had this diverse flavor. Computer scientists trying to understand perception, electrical engineers trying to understand representations and mechanical and control engineers building actuation mechanisms.

1.3 Intelligence : Reloaded(1960-2000)

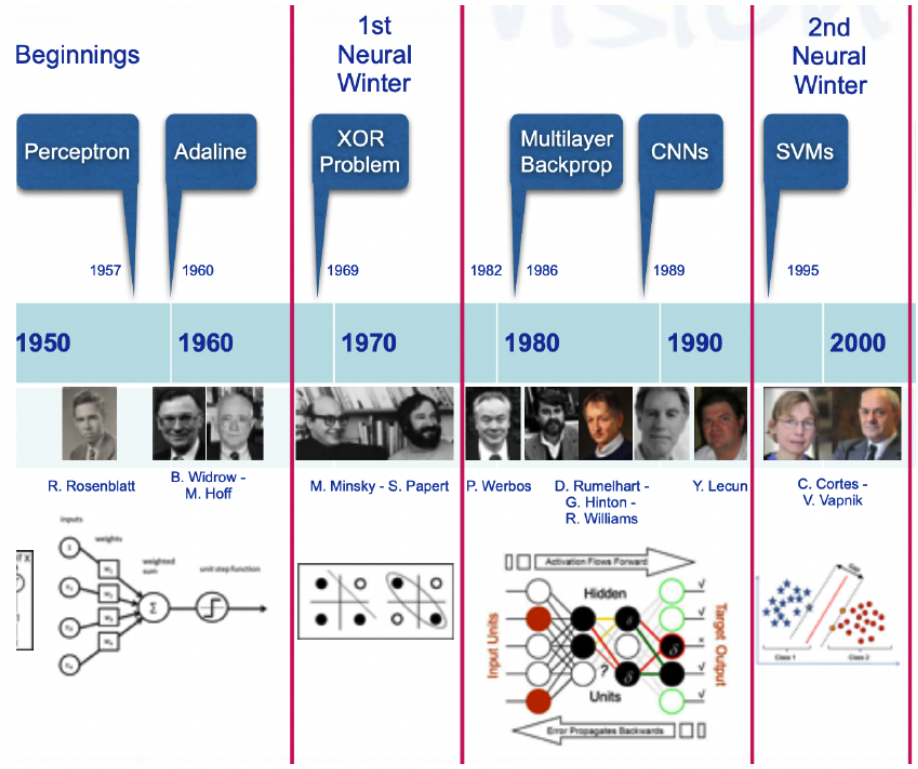
The early period created interest in intelligence and developed some basic ideas. The first major progress of what one would call the second era was made by Frank Rosenblatt in 1957 at Cornell University. Rosenblatt's model called the perceptron is a model with a single binary neuron. It was a machine designed to distinguish punch cards marked on the left from cards marked on the right, and it weighed 5 tons (Link). The input integration is implemented through the addition of the weighted inputs that have fixed weights obtained during the training stage. If the result of this operation is larger than a given threshold, the neuron fires. When the neuron fires its output is set to 1, otherwise it is set to 0. It looks like the function

$$f(x; w) = \text{sign}(w^\top x) = \text{sign}(w_1 x_1 + \cdots + w_d x_d).$$

Rosenblatt's perceptron ((Rosenblatt, 1958)) had a single neuron so it could not handle complicated data. Marvin Minsky and Seymour Papert discussed this in a famous book titled Perceptrons ((Minsky and Papert, 2017)). But unfortunately this book was widely perceived as two very well established researchers being skeptical of artificial intelligence itself. Interest in building neuron-based artificial intelligence (also called the connectionist approach) waned as a result. The rise of symbolic reasoning and the rise of computer science as a field coincided with these events in the early 1970s and caused what one would call the "first AI winter".

There was resurgence of ideas around neural networks, mostly fueled by the (re)-discovery of back-propagation by Rumelhart et al. (1985); Shunichi Amari developed methods to train multi-layer neural networks using gradient descent all the way back in 1967 and this was also written up in a book but it was in Japanese (Amari, 1967). Multi-layer networks came back in vogue because they could now be trained reasonably well. This era also brought along the rise of convolutional neural networks built upon a large body of work starting from two neuroscientists Hubel and Wiesel who did very interesting experiments in the 60s to discover visual cell types (Hubel and Wiesel, 1968) and Fukushima who implemented convolutional and downsampling layers in his famous Neocognitron (Fukushima, 1988). Yann LeCun demonstrated classification of handwritten digits using CNNs in the early 1990s and used it to sort zipcodes (LeCun et al., 1989, 1998). Neural networks in the late 80s and early 90s was arguably, as popular a field as it is today.

Support Vector Machines (SVMs) were invented in Cortes and Vapnik (1995). These were (are) brilliant machine learning models with extremely good performance. They were much easier to train than neural networks. They also had a nice theoretical foundation and, in general were a delight to use as compared to neural networks. It was famously said in the 90s that only the neural network researchers were able to get good performance with neural networks and no one else could train them well. This was largely true even until 2015 or so before the rise of libraries like PyTorch and TensorFlow. So we should



give credit to these libraries for popularizing deep learning in addition to all the researchers in deep learning. Kernel methods, although known much before in the context of the perceptron (Aizerman, 1964; Schölkopf and Smola, 2018), made SVMs very powerful. The rise of Internet commerce in the late 90s meant that a number of these algorithms found widespread and impactful applications. Others such as random forests (Breiman, 2001) further led the progress in machine learning. Neural networks, which worked well when they did but required a lot of tuning and expertise to get to work, lost out to this competition. However, there were other neural network-based models in the natural language processing (NLP) community such as LSTMs (Hochreiter and Schmidhuber, 1997) which were discovered in this period and have remained very popular and performant all through.

1.4 Intelligence: Revolutions(2006-)

The growing quantity of data and computation came together in late 2000s to create ideas like deep Belief Networks (Hinton et al., 2006), deep Boltz-

mann machines (Salakhutdinov and Larochelle, 2010), large-scale training using GPUs (Raina et al., 2009) etc. The watershed moment that got everyone's attention was when Krizhevsky et al. (2012) trained a convolutional neural network to show dramatic improvement in the classification performance on a large dataset called ImageNet. This is a dataset with 1.4 million images collected across 1000 different categories. Performing well on this dataset was considered very difficult, the best approaches in 2011 (ImageNet challenge used to be an annual competition 30 until 2016) achieved about 25% error. Krizhevsky et al. (2012) managed to obtain an error of 15.3%. Many significant results in the world of neural networks have been achieved since 2012. Today, deep networks in their various forms run a large number of applications in computer vision, natural language processing, speech processing, robotics, physical sciences such as physics, chemistry and biology, medical sciences, and many many others (LeCun et al., 2015).

1.5 A summary of our goals in this course

This course will take off from around late 1990s (kernel methods) and develop ideas in deep learning that bring us to today. Our goals are to

1. become good at using modern machine learning tools, i.e., implementing them, training them, modeling specific problems using ideas in ML;
2. understanding why the many quixotic-looking ideas in machine learning works.

After taking this course, we expect to be able to not only develop methods that use machine learning, but more importantly improve existing ideas using foundational understanding of the mathematics behind these ideas and develop new ways of improving machine learning theory and practice.

Chapter 2

Feasibility of Learning

This chapter gives a preview of generalization performance of machine learning models. We will take a more abstract view of learning algorithms here and focus only on binary classification. We will arrive at a "learning model" in Section 2.5.1, i.e., a formal description of what learning means. The topics we will discuss stem from the work of two people: Leslie Valiant who developed the most popular learning model called Probably Approximately Correct Learning (PAC-learning) and Vladimir Vapnik who is a Russian statistician who developed a theory (called the VC-theory) that provided a definitive answer on the class of hypotheses that were learnable under the PAC model.

2.1 Setup : Supervised Learning

- Data pairs $(x, y) \sim \mathcal{P}$ sampled IID from a joint distribution over (features, labels) space $\mathcal{X} \times \mathcal{Y}$.
  - Training dataset $\mathcal{D} := \{(x_i, y_i)\}_{i=1}^{n}$ consisting of n training data (sampled IID from P).
  - Test data (x, y) sampled from $\mathcal{P}$. At test time, we only observe x. Label y is unknown to us.
- Learning algorithm
  - Hypothesis set $\mathcal{H}$ consists of (many) hypotheses (functions) $h : \mathcal{X} \to \mathcal{Y}$.
  - The learning algorithm $\mathcal{A}$ takes as input the training set D and selects a specific hypothesis from $\mathcal{H}$, which we denote $\hat{h}$.

Notation : h **vs** $\hat{h}$. We reserve h to denote an arbitrary hypothesis in $\mathcal{H}$, while $\hat{h}$ denotes the hypothesis selected by the learning algorithm. That is, $\hat{h}$ depends on (i) the learning algorithm and (ii) the training dataset $\mathcal{D}$. One could write $\hat{h}_{A,D}$, but we lighten notation while urging you to keep this dependence in mind.

2.1.1 Running Example: Linear Classification

Let's ground these abstract concepts with a simple, visual example that we can keep in mind throughout the chapter: binary linear classification in 2D.

- **Data Space:** Our features live in a 2D plane, so $\mathcal{X} = \mathbb{R}^2$. The labels are binary, $\mathcal{Y} = \{-1, +1\}$. Data points $(\mathbf{x}, y)$ are pairs where $\mathbf{x}$ is a point in the plane and y is its class label.
- **Hypothesis Set $\mathcal{H}$:** The hypotheses are lines through the origin. Each line is defined (parameterized) by a weight vector $\mathbf{w} \in \mathbb{R}^2$. The classification rule for a given $\mathbf{w}$ is $h_{\mathbf{w}}(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x})$. The set $\mathcal{H}$ is the infinite collection of all such lines.
- **Training Data $\mathcal{D}$:** The learning algorithm is given n data points $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ sampled IID from (some distribution) $\mathcal{P}$. This is our concrete set of blue '+' and red 'o' points on the plane.
- **Learning Algorithm $\mathcal{A}$ and Final Hypothesis $\hat{h}$:** The algorithm's job is to look at all the training points and pick one specific line, $\hat{h}$, that it thinks is best.
- **The Goal:** As shown in Figure 1, our ultimate goal is to use the chosen line $\hat{h}$ to perform well on **new, unseen test points $\mathbf{x}$**. The next section quantifies "perform well."

2.2 How good is a hypothesis? Memorization vs Generalization

In order to formalize the goal of learning, we first need to formalize how we quantify whether a given hypothesis $h \in \mathcal{H}$ is "good" and "how good" it is. For concreteness, assume for now a **classification** setting such that $\mathcal{Y} = \{1, \ldots, k\} := [k]$, where k is the number of classes (e.g., cats, dogs, planes, etc.).

Definition 2.2.1 (Test Error)**.** *The test error (or risk, or population error, or out-of-sample error) of a hypothesis $h \in \mathcal{H}$ is defined as*

$$R(h) := \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{P}}\big[\mathbf{1}[h(\mathbf{x}) \neq y]\big] = \Pr_{(\mathbf{x},y)\sim\mathcal{P}}\big(h(\mathbf{x}) \neq y\big),$$

*where $\mathbf{1}[h(\mathbf{x}) \neq y]$ is the **zero-one (0/1) loss**.*

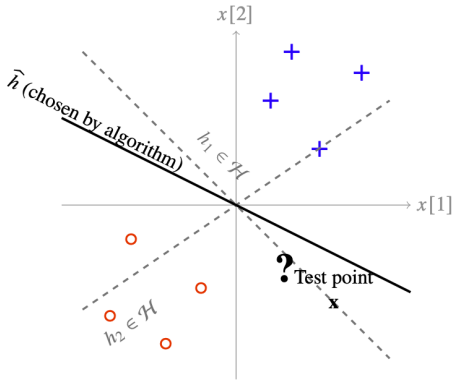


Figure 2.1: Visualizing the learning setup. The algorithm sees the blue '+' and red 'o' training points and selects the solid black line/hypothesis ($\hat{h}$) from the set of all possible lines/hypotheses (a few are shown as dashed). The goal is to perform well (see Sec. 2.2) on a new test point (the '?').

Generalization is the ultimate goal of learning. A hypothesis h that achieves small test error $R(h)$ is said to generalize well.
The probabilistic setup is crucial here. We define error as an average over all possible data points, assuming they are drawn randomly from $\mathcal{P}$. Without this assumption, generalization would be impossible. Recall the problem from our last discussion: if a test point x were chosen adversarially, our training data would provide no guarantee about its corresponding label y. This is analogous to sampling n balls from a bin; the sample tells us nothing about the color of a specific, hand-picked ball that was not part of our random draw. The IID assumption is what allows us to connect performance on seen data to performance on unseen data.
Now that we have quantified a measure of performance, we can formalize the goal of learning as that of selecting a hypothesis h that minimizes the risk $R(h)$ over all hypotheses in our set $\mathcal{H}$. The obvious challenge here is that we cannot actually measure $R(h)$ because it involves an expectation over the entire (and unknown) distribution $\mathcal{P}$. Instead, we only have access to the finite training set $\mathcal{D}$.
It is certainly possible to measure how well a given h performs on the training set. We do this by averaging the $0/1$ loss over the given examples.

Definition 2.2.2 (Training Error)**.** The training error (or empirical risk, or in-sample error) of a hypothesis $h \in \mathcal{H}$ is defined as

$$\hat{R}_n(h) := \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[h(\mathbf{x}_i) \neq y_i].$$

The $\hat{}$ denotes the quantity depends on the training data and the subscript n makes explicit the size of the train set.

Memorization [1] We say a hypothesis h memorizes the training data, or interpolates the data, if $\hat{R}_n(h) = 0$. That is, the training data gets memorized when the hypothesis makes no mistake when evaluated on all the training examples.

2.3 Generalization Error: From Train to Test

Question: What does the value of the training error tell us about the holy grail of learning, which is the **test error**?
Intuition: Ideally, we would be happy if small training error translates to small test error. This would give a ready recipe for learning: select the hypothesis that minimizes training error! So, is $\hat{R}_n(h)$ close to $R(h)$? This is where the IID assumption is handy. Once h is fixed, the individual-sample errors $R_i := \mathbf{1}[h(\mathbf{x}_i) \neq y_i]$ are IID Bernoulli random variables with mean $R(h)$. By the Law of Large Numbers (LLN), their average converges to the mean:

$$\hat{R}_n(h) \to R(h).$$

With infinite samples, the training error of a fixed hypothesis approaches the true risk! Are we done?
Subtleties: There are two important subtleties to discuss.

- The asymptotic statement from the LLN is encouraging, but it does not quantify the **rate** of convergence. We need to know how good our approximation is for a finite number of samples n.
- The approximation is only valid for a single, fixed hypothesis h. But what we really care about is the performance of $\hat{h}$, the hypothesis we choose after seeing the data.

Rate: How fast the train error of a fixed hypothesis approaches the test error? We can answer this by recalling a statement stronger than the LLN, that is the central limit theorem (CLT)!

[1] Memorization can often be an overloaded term in ML. The technical term for achieving zero training error is *interpolation*.

First, review the central limit theorem quickly. Let $X_1, X_2, X_3, \ldots, X_n$ be independently and identically distributed random variables with :

- finite mean $\mu = \mathbb{E}[X_i]$
- finite non-zero variance $\sigma^2 = Var(X_i)$

Let

$$\hat{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$$

, be the sample mean. Then,

$$\sqrt{n}(\hat{X}_n - \mu) \xrightarrow{d} \mathcal{N}(0, \sigma^2).$$

Equivalently for large n, $\hat{X}_n \approx \mathcal{N}\left(\mu, \frac{\sigma^2}{n}\right)$

According to the CLT, the normalized and centered empirical mean $\sqrt{n}\left(\hat{R}_n(h) - R(h)\right)$ converges in distribution to a Gaussian random variable $\mathcal{N}(0, \sigma^2)$. Equivalently, the gap $\hat{R}_n(h) - R(h)$ is distributed as $\mathcal{N}(0, \sigma^2/n)$. Importantly, note that the variance of the gaussian decreases with n and in the limit of $n \to \infty$ the Gaussian has its entire mass at zero, recovering the LLN. Recall also that the Gaussian distribution has an *exponential tail*. Thus, intuitively the gap $\hat{R}_n(h) - R(h)$ goes to zero exponentially fast in n. This intuition can be formalized by an inequality known as **Hoeffding inequality**:

$$\Pr\left(\,|\hat{R}_n(h) - R(h)| > t\,\right) \leq 2e^{-2t^2 n} \qquad \text{for all } t > 0. \tag{2.1}$$

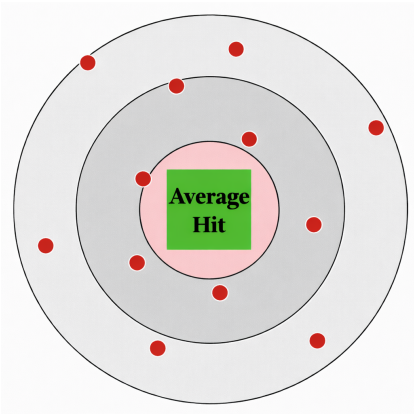
What is the probability over in the above expression? Remember, we took h to be fixed and also $R(h)$ already involves an expectation, so is itself a deterministic quantity (convince yourself!). The only random variable is $\hat{R}_n(h)$ which depends on the training set $\mathcal{D}$. The training set $\mathcal{D}$ consisting of n IID samples $(\mathbf{x}_i, y_i)$ is itself random. So, the probability above is over the randomness of drawing n IID samples $(\mathbf{x}_i, y_i)$ from the distribution $\mathcal{P}$. What the inequality says is that for any threshold t, no matter how small, if we draw a random training set from $\mathcal{P}$ and compute the training error $\hat{R}_n(h)$ then it will be at most t away from the holy grail test-error $R(h)$ with probability at least $1 - 2e^{-2t^2 n}$. To better see how the threshold determines the probability and vice versa, an equivalent way to state Hoeffding's inequality is to first pick some desired probability of success $1 - \delta \in (0, 1)$ (say $\delta = 0.05 \Rightarrow 1 - \delta = 0.95$) and select

$$t = \sqrt{\frac{\log(2/\delta)}{2n}}.$$

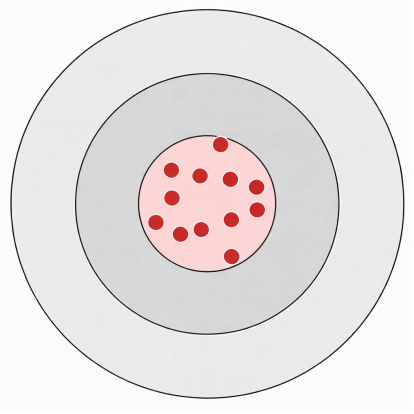
Then,

$$\text{With probability at least } 1 - \delta, \qquad |\hat{R}_n(h) - R(h)| \leq \sqrt{\frac{\ln(2/\delta)}{2n}}.$$

2.4 Remarks on Concentration



(a) **Expectation**: This estimator's "shots" are widely scattered. While any single shot is unreliable, the process is unbiased: the average position of all shots is exactly on the bullseye. The estimator is correct in expectation.



(b) **Concentration**: This archer/estimator is not only unbiased but also precise. Every single shot is tightly clustered around the bullseye, making any single shot a reliable indicator of the true center. In machine learning, concentration guarantees that the single estimate we calculate from our one dataset is very likely to be close to the true value.

Figure 2.2: Expectation vs Concentration

It is important to understand and appreciate how concentration is more powerful that something holding in expectation. The training error $\hat{R}_n(h)$ is an **estimator** of the true, but unknown, test error $R(h)$. For a fixed hypothesis set, this estimator is **unbiased**, that is, it is correct in expectation[2]:

$$\mathbb{E}_{\mathcal{D}}\left[\hat{R}_n(h)\right] = R(h).$$

What this means is that if we could draw many many different datasets and calculate our estimator on each one, the average of all those estimates would be the true value we're trying to find!
Unbiased-ness, however, does *not* promise anything about the single estimate we get from our *one* dataset :(
Concentration inequalities (like Hoeffding's, which we discussed) give us a guarantee about this. They tell us that if our dataset is large enough, the result of our *single* experiment becomes very likely ("concentrated") to be extremely close to the true expected value. This is exactly what we want in ML practice.

[2]Convince yourself this is true. Expectation is over realizations of the training set. *Hint:* Use THE property of expectation, i.e. linearity

We only have one dataset and we compute our estimator (the training error) just once. **Concentration gives us confidence that the single value we calculated is a reliable reflection of the true, underlying value.**
Hoeffding's inequality is just an example of a concentration inequalities. Concentration inequalities more generally can be thought of as finite-sample refinements of the LLN and the CLT. The term 'concentration' captures the essence of these inequalities (as well as the CLT and the LLN): a function (e.g., sum) of many many (large n) random variables that does not depend too much on any small change of any individual variable (e.g., the sum of large n terms doesn't change much by changing only one of the terms a little) concentrates to its expectation. More details are far beyond the scope of the course, but keep in mind that the concentration phenomenon is key in machine learning and more generally in high-dimensional data analysis. In an important sense, concentration is a blessing of dimensionality (contrast to computation which is often regarded as a curse of dimensionality).

What's next?

Does the above inequality hold for the final chosen hypothesis $\hat{h}$? The answer is no. The reason is that $\hat{h} = \hat{h}_{\mathcal{D},\mathcal{A}}$ depends on the training data. This causes the independence assumption to break. Concretely, the set of binary random variables $R_i = \mathbf{1}[h(x_i) \neq y_i]$ are no longer independent if h depends on the set $\{(x_i, y_i)\}_{i\in[n]}$. The final hypothesis $\hat{h}$ certainly does and so we will have to work around this technical challenge. Once we do that, we can revisit the question of how to achieve the goal of learning (minimizing the test error) by only having access to training data.

2.5 Generalization Bound

Goal of Learning. Find a hypothesis

$$h : \mathcal{X} \to \mathcal{Y}$$

that minimizes the *test error*

$$R(h) = \mathbb{E}_{(x,y)\sim P}\left[\mathbf{1}\left[h(x) \neq y\right]\right].$$

That is, solve the following optimization problem:

$$h^* = \arg\min_h R(h).$$

Optimization lingo : An optimization problem consists of an *objective function* (here, $R(\cdot)$) and an *optimization variable* (here, h). Solving an optimization problem amounts to finding the h that minimizes the objective. We call this specific h the *solution* to the optimization and denote it by

$$\arg\min_h R(h),$$

or simply by $h^\star$. We call the value of the objective at that solution, namely $R(h^\star)$, the *optimal cost* of the problem. In our learning setup, $h^\star$ can be thought of as the *golden hypothesis*, and $R(h^\star)$ as the *minimum risk*.

The challenge, of course, is that we cannot evaluate the objective function of this minimization problem, since it involves computing an expectation over the unknown data distribution P. Instead, we are given access to a training set

$$D := \{(x_i, y_i)\}_{i\in[n]}$$

consisting of n examples sampled IID from the underlying (unknown) distribution P.

Holy Grail of Machine Learning Practice: Find a "good" empirical estimate (i.e., one that can be evaluated on training data) of the test error, and minimize that instead.
A natural guess for such an estimate is the *training error* $R_n(h)$, and one may attempt to minimize this quantity instead. That is, solve the following optimization problem:

$$\hat{h} = \arg\min_{h\in\mathcal{H}} R_n(h).$$

Recall that we use the $\widehat{\cdot}$ notation for quantities that depend on the training set, and the solution h of the above minimization certainly depends on the training set, since $R_n(\cdot)$ depends on the training data. Also note that the minimization is constrained to be over a hypothesis set $\mathcal{H}$, which we get to choose. Finally, note that empirical risk minimization (ERM) is an example (in fact, a very popular one) of a learning algorithm.

So suppose (for now) that we pick a hypothesis set $\mathcal{H}$ and solve the above **empirical risk minimization problem (ERM)**. Certainly, $\widehat{R}_n(\hat{h})$ is the smallest among $\widehat{R}_n(h)$ for all $h \in \mathcal{H}$. But is the test error $R(\hat{h})$ evaluated at $\hat{h}$ also small? How small or large is it?
We could answer that question if we had a way to upper bound the so-called **generalization gap**

$$\hat{\Delta}_n := \left|\widehat{R}_n(\hat{h}) - R(\hat{h})\right|.$$

If $\hat{\Delta}_n$ is small, say smaller than some threshold t, then we are guaranteed that the unknown test error $R(\hat{h})$ is at most $\hat{R}_n(\hat{h})$, which we can measure and have

ensured is small, plus t. Concretely, if we can select $\hat{h}$ such that $\widehat{R}_n(\hat{h}) \approx 0$, then

$$R(\hat{h}) \lesssim t,$$

that is, the test error is at most t.
Let us pause for a moment and inspect the nature of the quantity $\hat{\Delta}_n$ that we want to bound. Since $\widehat{R}_n(\hat{h})$ is the empirical risk evaluated on the training set, and the training set consists of n random examples (x_i, y_i), the quantity $\widehat{R}_n(\hat{h})$ is itself random. Thus, $\hat{\Delta}_n$ is also random ! What does it mean, then, to bound a random variable by some quantity t?
What we aim for is a bound that holds with high probability. This probability is over the source of randomness, which—as explained above—comes from randomly drawing the n examples of the training set. Thus, probabilities are taken over the randomness of the training set D.
Concretely, we seek a bound that holds with probability at least $1 - \delta$, for some very small $\delta \in (0, 1)$ (e.g., $\delta = 0.05$, so the bound holds with probability at least 0.95). Formally, our goal becomes to find a threshold t (possibly depending on the number of samples n, the hypothesis set $\mathcal{H}$, and the failure probability δ) such that

$$\Pr\Big(\big|\widehat{R}_n(\hat{h}) - R(\hat{h})\big| \leq t\Big) \geq 1 - \delta,$$

or equivalently (check your understanding of this equivalence!),

$$\Pr\Big(\big|\widehat{R}_n(\hat{h}) - R(\hat{h})\big| > t\Big) < \delta \tag{2.2}$$

This should remind you of Section 2.3 where we bound the gap between empirical and test error for an arbitrary fixed hypothesis $h \in \mathcal{H}$. Using Hoeffding's inequality, we showed that for any fixed hypothesis $h \in \mathcal{H}$, with probability at least $1 - \delta$,

$$\big|\hat{R}_n(h) - R(h)\big| \leq \sqrt{\frac{\ln(2/\delta)}{2n}},$$

or equivalently, with probability at most δ,

$$\big|\hat{R}_n(h) - R(h)\big| > \sqrt{\frac{\ln(2/\delta)}{2n}}. \tag{2.3}$$

How do we arrive at a statement like Equation 2.2 that holds for $\hat{h}$(the data-dependent hypothesis) from the statement in Equation 2.3 that holds for a fixed ("not dependent on the data") hypothesis ?

2.5.1 The PAC-Learning Model

One way to go about it is to observe that the event

$$\big\{|\hat{R}_n(\hat{h}) - R(\hat{h})| > t\big\}$$

is a subset of the event " there exists some hypothesis in $\mathcal{H}$ for which the gap is greater than t." After all, if the specific data-dependent hypothesis $\hat{h}$ has a large gap,it immediately implies that at least one hypothesis in the set has a large gap. Therefore,the probability of the first event must be less than or equal to the probability of the second, more general event

$$\Pr\left(|\hat{R}_n(\hat{h}) - R(\hat{h})| > t\right) \leq \Pr\big(\exists h \in \mathcal{H} : \ |\hat{R}_n(h) - R(h)| > t\big)\,.$$

Now, the probability that there exists $h \in \mathcal{H}$ for which $|\widehat{R}_n(h) - R(h)| > t$ is exactly the probability of the union of events $|\widehat{R}_n(h) - R(h)| > t$ over $h \in \mathcal{H}$. That is,

$$\Pr\left(|\hat{R}_n(\hat{h}) - R(\hat{h})| > t\right) \leq \Pr\big(\bigcup_{h\in\mathcal{H}} |\widehat{R}_n(h) - R(h)| > t\big)$$

To bound this, we use the union bound. For this, assume (for now) that the hypothesis set $\mathcal{H}$ is finite and contains m hypotheses $h_1, h_2, \ldots, h_m$.The probability that at least one hypothesis has a large error gap is bounded by the sum of individual probabilities:

$$\begin{aligned}\Pr\left(\bigcup_{h\in\mathcal{H}} |\widehat{R}_n(\hat{h}) - R(\hat{h})| > t\right) &\leq \sum_{j=1}^{m} \Pr\left(|\widehat{R}_n(h_j) - R(h_j)| > t\right) \\ &\leq \sum_{j=1}^{m} 2e^{-2t^2 n} = 2me^{-2t^2 n}\end{aligned} \tag{2.4}$$

Here, after the union bound on the first inequality, for each individual probability $h_1, h_2, \ldots, h_m \in \mathcal{H}$ we then used Equation 2.1. By setting this probability of failure to a small value δ, we derive our main result.

Key Result. With probability at least $1 - \delta$, for all $h \in \mathcal{H}$ (and therefore for our chosen $\hat{h}$),

$$|\widehat{R}_n(h) - R(h)| \leq \sqrt{\frac{\ln(2m/\delta)}{2n}}.$$

This gives us the famous **generalization bound**,

$$R(\hat{h}) \leq \underbrace{\widehat{R}_n(\hat{h})}_{\text{Training Error}} + \underbrace{\sqrt{\frac{\ln(2m/\delta)}{2n}}}_{\text{Complexity Penalty}} \tag{2.5}$$

That, the estimated hypothesis $\hat{h}$ is approximately (t) correct with some probability (at least $1 - \delta$). Giving it the name the **PAC learning** framework.
We have shown that under the probabilistic setup, the training set D tells us something likely about fresh data: We can indeed trade the task of minimizing the test error $R(\hat{h})$ (which we cannot evaluate) to that of minimizing the training error $\hat{R}_n(\hat{h})$ (which we can evaluate on the training set). Moreover, this

trading results in paying a complexity penalty term that increases logarithmically with the number of hypothesis (m) and decreases proportional to $1/\sqrt{n}$ with the number of examples n.
Note that the way we arrived at bounding the generalization error of $\hat{h}$ (the empirical risk minimizer in (ERM)) is via having obtained a bound in Eq. 2.3 that holds simultaneously for all hypotheses $h \in \mathcal{H}$. Such bounds are called uniform bounds. The remark to be made is that from such a uniform bound, we can immediately get the same generalization bound as in Eq 2.5 for any data-dependent hypothesis $\hat{h}$ irrespective of how this was chosen after seeing the data. Choosing $\hat{h}$ by solving(ERM)is one option (often a good one), but the bound holds more generally for any learning algorithm $\mathcal{A}$. To give an example, $\hat{h}$ could have been the solution of a so-called regularized ERM:

$$\hat{h} = \arg\min_{h \in \mathcal{H}} \widehat{R}_n(h) + \Omega(h) \tag{2.6}$$

where now the objective function is augmented by a regularization term Ω. We will revisit regularization later in the course. For now, think of $\Omega(h)$ as a heuristic proxy that aims to capture the complexity of the hypothesis: 2.6 minimizes the sum of the training error and this complexity penalty term motivated by Eq 2.5. The important thing to note is that the bound in 2.5 holds exactly as is for both the (ERM) or the (rERM) solution. This is both a blessing (the bound is very general!) and a curse (the bound is not properly capturing the impact on generalization of the learning algorithm!)

2.6 The Tradeoff of Model Complexity

Looking closer at (2.5), the generalization bound reveals a fundamental tradeoff. To minimize the right-hand side, we must balance two competing goals:
Competing Goal #1: Minimize training error. To achieve a low $\widehat{R}_n(\hat{h})$, we prefer a large, complex hypothesis set (large m) so we have a better chance of finding a function that fits the data well.
Competing Goal #2: Minimize generalization gap. The complexity penalty grows with m. A more complex model requires more data (n) to ensure the gap between training and test error is small. The number of samples required to achieve a certain error is called the sample complexity [3].
This trade off is captured nicely by what is known as the error vs model-complexity curves. On the one hand,training error $\hat{R}_n$ is (likely to be) monotonically decreasing with increasing model complexity. On the other hand, the generalization gap $\sqrt{\frac{\log(2m/\delta)}{2n}}$ is increasing. This results in a U-shaped test-error curve for $R(h)$. This tells us that there is an optimal, not too small but also not too large,complexity level for the hypothesis set.

[3] According to (2.5),to have error at most ϵ hold with probability at least $1-\delta$, we need $n \geq \frac{2}{\epsilon^2}\log(m/\delta)$

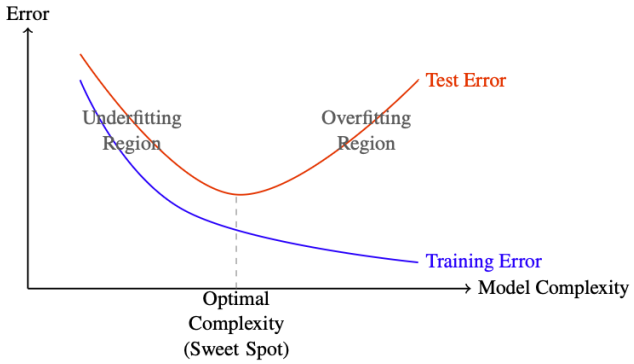


Figure 2.3: A typical U-shaped curve showing the relationship between model complexity and error. As complexity increases, training error decreases, but test error initially falls and then rises.

2.7 Infinite Hypothesis Class and VC Dimension

Let's take a second look at our friend Equation 2.4, it is straightforward to see that if our desiderata includes an error probability of at most δ, then it can be achieved if $2me^{-2t^2n} \leq \delta$. In other words, the least number of training samples required is given by

$$n \geq \frac{1}{2t^2} \log\left(\frac{2m}{d}\right)$$

Thus, if our hypothesis class $\mathcal{H}$ has infinite number of elements in it ($m \to \infty$) , the number of samples n goes to infinity ($n \to \infty$) as well. Note, that Equation 2.5 is still valid since its an upper bound. However, it's a vacuous bound.
Vladimir Vapnik and Alexey Chernovenkis (Vapnik, 2013) developed the so-called VC-theory to answer the above question. Technically, VC-theory transcends PAC-Learning but we will discuss only one aspect of it within the confinements of the PAC framework. VC-theory assigns a "complexity" to each hypothesis $h \in \mathcal{H}$. Before, we can explore VC-dimension we need to understand a basic concept along the way.

Shattering of a set of inputs We say that the set of inputs $D = \{x_1, x_2, \ldots, x_n\}$ is shattered by the hypothesis class $\mathcal{H}$, if we can achieve every possible labeling out of the 2^n labellings using some hypothesis $h \in \mathcal{H}$. The size of the largest set D that can be shattered by $\mathcal{H}$ is called the VC-dimension of the hypothesis class $\mathcal{H}$. It is a measure of the complexity/expressiveness of the class; it counts how many different classifiers the class can express.
If we find a configuration of n inputs such that when we assign any labels to these data, we can still find a hypothesis in $\mathcal{H}$ that can realize this labeling, then $VC(\mathcal{H}) \geq n$. On the other hand, if for every possible configuration of $n+1$ inputs, we can always find a labeling such that no hypothesis in $\mathcal{H}$ can

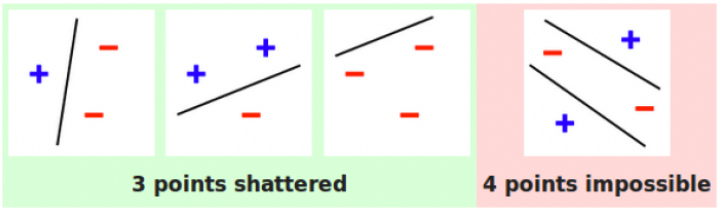


Figure 2.4: d=2: See that for the lower bound, we found some configuration of the 3 points, such that a linear threshold function always separates the points consistently with the labels; for any possible labeling. 3 such labellings are shown, convince yourselves that it can be done for all 8 cases. Observe that we cannot do the same for 4 points. In the figure above one such unrealizable configuration is given (With the "XOR" labeling). To prove the upper bound we need to talk about ANY configuration though. See that the only other case for 4 points, is that one point is inside the convex hull generated from the other 3. Find the labeling that cannot be obtained with linear classifiers in this case.

realize this labeling, then $n \geq VC(\mathcal{H})$. If we find some n for which both of the above statements are true, then $VC(\mathcal{H}) = n$.
Some examples.

- d-dim Linear Threshold Functions: VC-dim $= d + 1$.
- 2 dimensional axis aligned rectangles: VC-dim = 4 (exercise)
- If the hypothesis class is finite, then

$$VC(\mathcal{H}) \leq \log |\mathcal{H}|$$

- For a neural network with p weights and sign activation function $VC = O(plogp)$.

It is a deep result that if the VC-dimension of hypothesis space is finite $V = VC(\mathcal{H}) < \infty$, then this class has the uniform convergence property (for any $h \in \mathcal{H}$, the empirical and population error are close). The number of samples required is lower bounded in the following fashion :

$$n \geq O\left(\frac{V + \log(1/\delta)}{t^2}\right)$$

If a hypothesis class has infinite VC-dimension, then it is not PAC-learnable and it also does not have the uniform convergence property. For a more in depth study and proof of the above theorem result, I would encourage you to read Chapter 6, 7 in Shalev-Shwartz and Ben-David (2014).

Bibliography

Aizerman, M. A. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837.

Amari, S. (1967). A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, EC-16(3):299–307.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119–130.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hubel, D. H. and Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

Minsky, M. and Papert, S. A. (2017). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA.

Pickering, A. (2010). *The Cybernetic Brain: Sketches of Another Future*. University of Chicago Press, Chicago.

Raina, R., Madhavan, A., and Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 873–880.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, Institute for Cognitive Science, University of California, San Diego, La Jolla, CA.

Salakhutdinov, R. and Larochelle, H. (2010). Efficient learning of deep boltzmann machines. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research, pages 693–700.

Schölkopf, B. and Smola, A. J. (2018). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

Turing, A. M. (2009). Computing machinery and intelligence. In Epstein, R., Roberts, G., and Beber, G., editors, *Parsing the Turing Test*, pages 23–65. Springer, Dordrecht.

Vapnik, V. N. (2013). *The Nature of Statistical Learning Theory*. Statistics for Engineering and Information Science. Springer, 2 edition.

Wiener, N. (1965). *Cybernetics: Or Control and Communication in the Animal and the Machine*, volume 25. MIT Press, Cambridge, MA.