LECTURE 1

1. Introduction

Let's loosely define deep learning (DL) to mean the use of (artificial deep) neural networks (DNNs) in the study of machine learning problems. For the time being we will focus on the feedforward networks (as opposed to the recurrent ones). It is well known that DL has made possible significant advances in recent years. Our main topic is the theory of deep learning which is a relatively young but rapidly evolving subject. The most well-developed application of DL is to supervised learning. While this has reached certain amount maturity and is the basis of many industrial applications of DL, in many ways it still remains a research area. In this course, for the most part, we will not discuss the latest and greatest advances in applied DL research; however, we will look at some of the latest theoretical advances. We will focus on some of the main phenomena related to DL in relatively simple settings. This restriction is made in the hope of focusing on the core issues in our theoretical study of these phenomena and not be distracted by many extra features that are added to improve the performance.

For supervised learning the general question is to explain why deep learning works as well as it does on various classification tasks. This question can be decomposed into several related ones: What kind of data are neural nets able to *represent*? Why does neural net training often achieve small training error? Why do neural nets often generalize resaonably well. For what types of data do these phenomena happen?

Of course, even restricted to supervised learning, DNNs are no panacea. On Kaggle, techniques such as gradient boosting are often useful. Some of the issues with present DNNs are: (1) Need for large datasets, (2) susceptibility to adversarial examples, (3) lack of interpretability. We will look at some of these later in the course.

There is a classical theory of machine learning originating with Vapnik and Chervonenkis in the 1970s with later work by Valiant and many others. While this theory is beautiful, it suffers from some of the same issues that the general theoretical computer science worst case modeling suffers from in some situations: the worst case is too pessimistic. Some examples of this are the SATISFIABILITY problem which is NP-complete is nevertheless efficiently solvable for many (most?) formulas that are encountered in practice. We do not seem to have an adequate and reasonably comprehensive theoretical understanding of this phenomenon. On the other hand, compressed sensing and smoothed analysis of the simplex method for linear programming are two well-known examples where again the worst case view is too pessimistic but a non-worst case modeling of the situation leads to a quite satisfactory theoretical understanding.

For deep learning we seem to be facing a similar situation: we know that it's hard in the worst case, but we also know that it works well for many problems of interest. Understanding this situation is one of the main theoretical questions in DL and will be discussed at length in this course. In a sense, this is not a precisely stated mathematical problem but rather a scientific problem about explanation of a phenomenon involving data from the real world (e.g., images, speech and many other types). However, there might be compelling mathematical explanations that do not involve too many details of the data, somewhat along the lines of the examples of compressed sensing and smoothed analysis mentioned above. This is the general direction we will follow for the question of understanding the good performance of deep feedforward networks. We will make the question more precise after the first 2-3 lectures. Remark: We emphasized the need to move away from worst-case modeling for some problems as it seems to be required to make further theoretical progress. For other problems, such as security related applications of DL, worst-case modeling may be more appropriate.

While the classical machine learning theory has not been quite successful in analyzing DL, it provides useful tools and a theoretical context and framework in which we can formulate our problems and study them. So we will begin by quickly outlining those parts of the classical learning theory (variously known as statistical learning theory, PAC (probably approximately correct) learning, computational learning theory) that will be relevant for DL. Our coverage will be brisk; see e.g. [SSBD14] for a more leisurely and detailed introduction. Parts of these notes are directly drawn from [SSBD14].

In this part of the course we will focus on supervised learning. Within supervised learning we will focus on classification with small number of classes and regression. Some standard examples of such problems:

(1) Spam detection.
(2) Digit identification for black and white images with white background.
(3) Image classification: cat or dog?
(4) Papaya: guess if a given papaya is tasty or not based on its colour and hardness.

2. A formal framework for statistical learning

With the above examples of supervised learning problems in mind, we introduce a formal statistical model in which we can study such problems. This model is concerned exclusively with the statistical aspects of the learning problem (e.g. how many samples are needed to solve a problem). Of course the computational aspects (how efficiently can we solve a learning problem) are also important and will be discussed in due course. A *learner* is an algorithm that takes as input labeled instances and produces a rule that can be used to label new instances. We make this more formal below:

Learner's input.

- Domain set X (also called instance space): This is the set of objects we want the learner to label. More precisely, this is the set of *encodings* of objects we want the learner to label. Normally this is represented as a vector x in $\mathbb{R}^d$ for some d (though other forms may also be used; in particular, some of the coordinates might take on only discrete values). Elements of X are called instances. The vector is called feature vector and the coordinates of this vector are called features or attributes. Examples: (1) In the spam detection example, a simple feature vector associated to an email could be the frequencies of dictionary words in the email: there is a coordinate for each dictionary word and its value is the number of times the word occurs. (2) For digit classification feature vector could be raw pixels. Or it could be something else: ratio of black pixels to white pixels, number of loops in the black pixels, height and width of the black pixels...and similarly for cats and dogs. (4) For papaya classification it's a two dimensional vector indicating the color and hardness by two real numbers.

- Label set Y: For binary classification task this is $\{0, 1\}$ or $\{-1, 1\}$. For classification into k classes this could be $\{0, 1, \ldots, k-1\}$. For regression it is $\mathbb{R}$. (There are supervised learning problems with more complex label sets which we are not considering.) The labels of instances are also called dependent variable, response variable or target. In the spam detection example, the label 0 can correspond to not spam and 1 corresponds to spam and similarly for other examples above.
- Training data S: This is a sequence of labeled instances $S = ((x_1, y_1), \ldots, (x_m, y_m))$ where $x_i \in X$ and $y_i \in Y$. The tuples (x_i, y_i) are called training examples. S is also called the training set.

Given training data, we would like to design a learner that correctly outputs the labels of new instances particularly the ones that the learner has not seen before in the training data. This is the central task of machine learning.

But before we get to this problem we need to decide on what feature vectors to use. To answer this we can do no better than to quote Domingos [Dom12] from 2012 (just around the time DL started to take off in really big way):

> At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used. If you have many independent features that each correlate well with the class, learning is easy. On the other hand, if the class is a very complex function of the features, you may not be able to learn it. Often, the raw data is not in a form that is amenable to learning, but you can construct features from it that are. This is typically where most of the effort in a machine learning project goes. It is often also one of the most interesting parts, where intuition, creativity and "black art" are as important as the technical stuff.

The process of coming up with features is called feature engineering and involves domain expertise. For some problems, DL obivates the need for feature engineering and works with "raw" features. This is one of the main advantages of deep learning. But for problems with small amount of data feature engineering and other classification methods may still be better. For our statistical framework, we will just assume that feature vectors have been fixed somehow.

The learning algorithm "knows" the target class C in the sense that the algorithm has been designed with the class in mind. [page 8 of Kearns–Vazirani]

Learner's output. Learner outputs a hypothesis $h : X \to Y$ that will be used to classify new instances. A hypothesis is also called a predictor, a prediction rule, and a classifier. If the learner is denoted A then the output of the learner on input S will be denoted by $A(S)$.

Examples

The input to the learner: Of course the learner gets the training data (and we implicitly assumed that it knows the domain and label sets X, Y.) It does not know the distribution $\mathcal{D}$.

Measuring the quality of the output of the learner and the data generation process. The learner has done a good job if it correctly classifies all or most new instances called test examples. To make this notion precise and quantitative we need to say something about how training and test examples are chosen. In general, it's not easy to give a precise description and in any case it seems harder than the classification task at hand: how are the images of cats and dogs constructed, how are the spam emails composed? One solution that has worked quite well and has become the standard is to bring in an important new element into the model, namely probability, by introducing a distribution on the data. We posit a **data distribution** $\mathcal{D}$ on $X \times Y$. Each example (x, y) is generated i.i.d. (identically and independently at random) by sampling $(x, y) \sim \mathcal{D}$. Thus, the examples in the training and test data are generated according to this procedure.

A special case of this model, called **the realizable case** is often considered: here we have a distribution $\mathcal{D}$ on X and a labeling function $f : X \to Y$. Now examples (x, y) are generated i.i.d. by sampling $x \sim \mathcal{D}$ and outputting $(x, f(x))$. Our more general model of distribution on $X \times Y$ corresponds to the situation where the feature vector x does not categorically determine the label y. This is a more realistic model as in general one cannot expect that feature vector captures everything about the underlying instance.

We can now define measures of the quality of the learner's output, the predictor h. The simplest measure for classification is:

$$L_{\mathcal{D}}(h) := \Pr_{(x,y)\sim\mathcal{D}}[h(x) \neq y].$$

This is also called test error, risk, or loss. It is also called generalization error although confusingly this word is also used for the difference between training and test errors; we will refer to this difference as *generalization gap*.

For regression the simplest measure is the square loss:

$$L_{\mathcal{D}}(h) := \mathbb{E}_{(x,y)\sim\mathcal{D}}(h(x) - y)^2.$$

Later, we will consider other measures.

We can now state a preliminary informal version of the supervised learning problem:

Problem 2.1 (Main problem of supervised learning, informal version)**.** *Design a learner A with the following properties. A gets as input the domain X, label set Y, training set $S = ((x_1, y_1), \ldots, (x_m, y_m))$ with each example i.i.d. from $\mathcal{D}$. The learner does not get $\mathcal{D}$ itself. In addition, the learner may have some more information about the problem structure, in particular what kind of hypotheses might be appropriate. The goal of A is to output a predictor $A(S) = h$ so as to minimize $L_{\mathcal{D}}(h)$.*

If the learner knew $\mathcal{D}$ completely then the best predictor, the one that minimizes $L_{\mathcal{D}}(\cdot)$, would have been the **Bayes-optimal predictor**. For binary classification this is given by

$$f_{\mathcal{D}}(x) := \begin{cases} 1 & \Pr[y = 1|x] \geq 1/2, \\ 0 & \text{otherwise.} \end{cases}$$

No other predictor (deterministic or randomized) can have lower 0-1 error than the Bayes-optimal predictor. The error of the Bayes-optimal predictor measures the inherent randomness of the problem and the goal of the learner is to come close to this. But constructing Bayes-optimal predictor is not practical as the learner does not know $\mathcal{D}$ completely and all the learner's information about $\mathcal{D}$ comes via the training set S. Which predictor h should the learner output? If we could compute $L_{\mathcal{D}}(h)$, then the problem would become an optimization problem over $\mathcal{H}$. But since we don't know $\mathcal{D}$ we cannot compute $L_{\mathcal{D}}(h)$. This is the essential new statistical element in learning problems over optimization.

An alternative to $L_{\mathcal{D}}(h)$ is the **empirical loss** $L_S(h)$ given by

$$L_S(h) := \frac{1}{m}|\{i \in [m] : h(x_i) \neq y_i\}|.$$

Note that $\mathbb{E}_S(L_S(h)) = \mathbb{E}_{\mathcal{D}}(h)$. Assuming that $L_S(h) \approx L_{\mathcal{D}}(h)$ for all h, the learner could output h for which $L_S(h)$ is small. There are two possible issues with this: (1) We need to check if our assumption $L_S(h) \approx L_{\mathcal{D}}(h)$ is correct; (2) there may be many h for which $L_S(h)$ is small and some of these may be good ($L_{\mathcal{D}}(h)$ is small) and others bad ($L_{\mathcal{D}}(h)$ is large). We will come to (1) shortly, but let's take up (2) right away. One example of bad h is

$$h_S(x) := \begin{cases} y_i & \text{if } \exists i \in [m] \text{ s.t. } x_i = x, \\ 0 & \text{otherwise.} \end{cases}$$

For the spam filtering example, h_S says classify the *training* examples of spam as spam and all other examples are classified as non-spam. For the Papaya example, this has simple geometric picture. The above hypothesis does very well on empirical loss: $L_S(h_S) = 0$. But h_S is useless on new examples as it has memorized the training examples S and instances outside S are all labeled 0 which could be incorrect most of the time leading to $L_{\mathcal{D}}(h_S)$ being large. We say that h_S suffers from **overfitting**. More precisely, the learner A suffers from overfitting if the difference between the true risk of its output, $L_{\mathcal{D}}(A(S))$, and the empirical risk of its output, $L_S(A(S))$, is large.

How can we deal with overfitting? Intuitively, one reason we ran into overfitting was that the set of predictors was too large, in particular it contained "unreasonable" ones. To restrict to the "reasonable" ones apart from the information about the training set (and associated information X, Y), the learner should also use some background information about the problem otherwise it's impossible to solve the learning problem (this is the No Free Lunch Theorem discussed below). One of the main ways in which this prior information is used by the learner is by choosing the predictor from a set $\mathcal{H}$ of predictors. $\mathcal{H}$ is determined by the learner before seeing S and thus only depends on the learner's background knowledge of the problem. The set $\mathcal{H}$ is called *hypothesis class*, its elements are function of type $X \to Y$ and are also called *hypotheses.*

We can now restate the procedure of choosing the predictor with small empirical error restricted to $\mathcal{H}$ called **ERM** for empirical risk minimization:

$$ERM_{\mathcal{H}}(S) = \arg\min_{h \in \mathcal{H}} L_S(h).$$

As we mentioned earlier, for the moment we are ignoring the important issue of how the hypothesis minimizing the empirical loss is computed and will be discussed later for specific $\mathcal{H}$ such as neural networks. Restriction of h to $\mathcal{H}$ is an example of **inductive bias**. Generally speaking, inductive bias is any choice in the design of the learner that influences the final output hypothesis. Some examples: Some of the popular supervised learning methods include Naive Bayes, k-nearest neighbors, logistic regression, SVMs, kernel machines, decision trees, random forests, gradient boosting, and feedforward neural networks. Each of these can be thought of as providing a certain inductive bias by restricting the class of hypotheses that can be output. For example, SVM's are biased towards linear classifiers with maximum margin (The *algorithm* to find the hypothesis also can lead to strong inductive bias and will be discussed later.) For different problems different methods may be appropriate. A comprehensive study by Caruana et al. [CN06, CKY08] applied many of the above methods (with neural networks being shallow) to several datasets and compared them on several different metrics. Random forests was the best peformer overall with neural nets second or third. Later studies have found similar results with deeper networks. Deep learning tends to be the best in tasks involving "perceptual" inputs (images, speech etc.), but for other tasks or tasks where the data is limited other approches may be better. In Kaggle competitions gradient boosting is often the method of choice along with DL (and ensemble methods). A very general type of inductive bias called Occam's razor suggests choosing the "simplest" hypothesis that's consistent with the data (i.e., $L_S(h) = 0$). The notion of simplest could be formalized in various ways: for example the shortest program to compute the hypothesis.

ERM when applied to very rich $\mathcal{H}$ doesn't do well on test data. We need to change at least one of $\mathcal{H}$ or the ERM rule.

Examples: The situation is well-illustrated by the regression task for polynomials. Another good example is the class of rectangles vs the class of polytopes.

Because of the randomness and finite size of the training set S we only get partial information about $\mathcal{D}$ and thus the predictor cannot be expected to achieve the loss equal to that of the Bayes-optimal predictor. To this end we allow the output predictor to be worse by an error ϵ. Also because of the randomness in S it is possible with non-zero (though very small) probability that S is such that it provides almost no information about $\mathcal{D}$, for example if all elements of S are the same (for discrete $\mathcal{D}$) or very close to each other (for continuous $\mathcal{D}$). The proper quantification of the problem is provided by Valiant's PAC (probably approximately correct) model later extended to the agnostic setting discussed below. We will work with a more general type of loss functions than the ones discussed earlier. Let $Z = X \times Y$ and let $\ell : \mathcal{H} \times Z \to \mathbb{R}_+$ be a function, which will be called a (per-example) loss function. $\ell(h, z)$ is the "error" or "loss" of predictor h on example z. The error or risk or loss function $L_\mathcal{D}(f)$ is the expected (per-example) loss of the predictor for distribution $\mathcal{D}$ on Z:

$$L_\mathcal{D}(h) := \mathbb{E}_{z\sim\mathcal{D}}\ell(h, z).$$

For our earlier examples, of the generalization error $\Pr[h(x) \neq y]$, the corresponding loss function is the 0-1 loss:

$$\ell_{0-1}(h, (x, y)) := \begin{cases} 1 & h(x) \neq y, \\ 0 & \text{otherwise.} \end{cases}$$

For the square loss the (per-example) loss function is $\ell_{sq}(h, (x, y)) := (h(x) - y)^2$.

Definition 2.1 (Agnostic PAC-learnability)**.** A hypothesis class $\mathcal{H}$ is agnostically PAC-learnable if there exist a function $m_\mathcal{H} : (0, 1)^2 \to \mathbb{N}$ and a learner A with the following property: for every error parameter $\epsilon \in (0, 1)$ and confidence parameter $\delta \in (0, 1)$, and any distribution $\mathcal{D}$ on $X \times Y$, with training set S of size at least $m_\mathcal{H}(\epsilon, \delta)$ generated i.i.d. from $\mathcal{D}$, the learner $A(S)$ outputs a hypothesis h such that with probability at least $1 - \delta$ (over the choice of S as well as any randomness of A) we have

$$L_\mathcal{D}(h) \leq \min_{h'\in\mathcal{H}} L_\mathcal{D}(h') + \epsilon.$$

The word "agnostic" refers to the fact that we do not require $\min_{h'\in\mathcal{H}} L_\mathcal{D}(h') = 0$, that is to say no hypothesis perfectly fits $\mathcal{D}$. We only want the output of the learner to be close to the best possible hypothesis. The special case where realizability assumption holds is $\min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') = 0$.

The No Free Lunch Theorem. The No Free Lunch Theorem is a mathematical statement of the simple intuitive fact that for any learner to be able to classify beyond the training examples, the learner must have some prior knowledge (or make an assumption) of the problem otherwise on the unseen examples anything is possible and thus the learner will not achieve good generalization. For example, for spam filtering task, we can try to see words that occur frequently in spam training examples and declare a test example spam if it has such words. This assumption reflected our prior knowledge about the problem. Without some such assumption, we cannot have a learning algorithm. A bit more precisely, the No Free Lunch theorem states that on average over all possible $\mathcal{D}$, every predictor has the same error on unseen instances. This error is the same as doing a random prediction. This means that no predictor is better than another for some $\mathcal{D}$. However, if we have prior knowledge or reasonable assumptions about the kind of $\mathcal{D}$'s that are likely to arise in the problem at hand, the learner can choose a predictor that's well-suited for that class of $\mathcal{D}$ and do well. For a more detailed treatment of the No Free Lunch Theorem see [SSBD14] or the original paper of Wolpert [?]. The main phiosophical implication of this theorem is that there is no single universal learning algorithm for all learning problems. This seems to raise further philosophical questions about learning but we will not go there.

Bias-variance trade-off. We have seen that it is important to incorporate prior knowledge into the learner to avoid overfitting and no free lunch. One of the main ways of doing this is by choosing the hypothesis class $\mathcal{H}$ carefully for the problem at hand. Here we face a trade-off called bias-complexity trade-off. To discuss this we decompose the error of the classifier according to the expressiveness of $\mathcal{H}$ and

For h_S an $ERM_{\mathcal{H}}(S)$ hypothesis we can decompose

$$L_{\mathcal{D}}(h_S) = \epsilon_{app} + \epsilon_{est},$$

where $\epsilon_{app} = \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$ and $\epsilon_{est} = L_{\mathcal{D}}(h_S) - \epsilon_{app}$. Here ϵ_{app} is the approximation error: how well are the predictors in $\mathcal{H}$ able to fit $\mathcal{D}$. This is determined by how expressive $\mathcal{H}$ is. And ϵ_{est} is the estimation error that controls overfitting: how much error is incurred because we work with the sample S instead of with $\mathcal{D}$. If we make $\mathcal{H}$ more *complex* or *expressive* or increase its *capacity* (we are using all these terms in an informal sense here; we will give some formalizations soon), we can reduce ϵ_{app} but it's likely that this will lead to overfitting increasing ϵ_{est}. If we make $\mathcal{H}$ simpler we can reduce ϵ_{est} but it might increase ϵ_{app} leading to underfitting. In other words, if we reduce the inductive *bias* by allowing $\mathcal{H}$ to be more complex, we allow more *variance* in the hypothesis that is output as it will start to depend on the randomness in the data. Methods such as k-nearest neighbors have low bias but high variance. Linear classifiers are the opposite. Informally, bias $= \epsilon_{app}$ and variance $= \epsilon_{est}$.

Examples: polynomials

Sections 5.2 to 5.5 of [?] also cover some of the topics discussed above.

3. Finite hypothesis classes are agnostically PAC-learnable

Hypothesis classes with finite size are a good starting point to study PAC-learnability. The basic idea for showing learnability of this class is to show that for sufficiently large m, a random sample S is such that $|L_S(h) - L_{\mathcal{D}}(h)| \leq \epsilon$ for all $h \in \mathcal{H}$. Such a training set S is called **ϵ-representative** (with respect to $Z, \mathcal{H}, \mathcal{D}$).

If we have an $\epsilon/2$-representative sample S, then the $ERM_{\mathcal{H}}(S)$ yields a good hypothesis: for any $h_S \in \arg\min L_S(h)$, we have $L_{\mathcal{D}}(h_S) \leq \min_{h\in\mathcal{H}} L_{\mathcal{D}}(h) + \epsilon$. This is because for every $h \in \mathcal{H}$ we have:

$$L_{\mathcal{D}}(h_S) \leq L_S(h_S) + \epsilon/2 \leq L_S(h) + \epsilon/2 \leq L_{\mathcal{D}}(h) + \epsilon.$$

Theorem 3.1. *Let $Z = X \times Y$ be a domain and $\mathcal{H}$ a hypothesis class of hypotheses of the form $h : X \to Y$. Let $\ell : \mathcal{H} \times Z \to [0,1]$ be a loss function. Then with probability at least $1 - \delta$ a sample of size $\lceil \frac{2\log(2|\mathcal{H}|/\delta)}{\epsilon^2} \rceil$ is an ϵ-representative sample.*

Proof. The idea of the proof is to use the fact that $L_S(h)$ is an average of i.i.d. random variables which we know is concentrated around the mean ($L_{\mathcal{D}}(h)$ in this case) by inequalities such as Hoeffding's. For $S = (z_1, \ldots, z_m)$, we have $L_S(h) = \frac{1}{m}\sum_{i\in[m]} \ell(h, z_i)$. Notice that $\mathbb{E}_{z\sim\mathcal{D}}\ell(h,z) = L_{\mathcal{D}}(h)$. Using Hoeffding's inequality (see [SSBD]) we have

$$\Pr_S[|L_S(h) - L_{\mathcal{D}}(h)| \geq \epsilon] \leq 2\exp(-2m\epsilon^2).$$

Using the union bound over $h \in \mathcal{H}$,

$$\Pr_S[\text{there exists } h \in \mathcal{H} \text{ such that } |L_S(h) - L_{\mathcal{D}}(h)| \geq \epsilon] \leq 2|\mathcal{H}|\exp(-2m\epsilon^2).$$

Choosing $m \geq \frac{\log(2|\mathcal{H}|/\delta)}{2\epsilon^2}$ makes the right hand side in the above inequality at most δ, completing the proof. □

For the realizable case a similar result can be proved more simply.

4. Generalization bounds

We discussed above the decomposition of the output $h_S \in ERM_{\mathcal{H}}(S)$ as $L_D(h_S) = \epsilon_{app} + \epsilon_{est}$. The task of the learner to choose $\mathcal{H}$ so that both of these errors are small. It will often be the case that for $\mathcal{H}$ we use ϵ_{app} can be small (though finding the corresponding predictor may not be easy). This amounts to how expressive a the hypothesis class $\mathcal{H}$ is. We will consider this question later. Bounding ϵ_{est} is the problem we will now consider. Many techniques have been developed for this. We will consider VC-dimension, Rademacher complexity, compression bounds, PAC-Bayes bounds. Here we introduce these methods in the statistical learning framework. Later, we will apply them to neural networks.

5. VC-dimension

Classically, the generalization bounds were proven using the notion of VC-dimension of a hypothesis class. This fundamental notion turns out to have various applications beyond learning theory. We will cover it only very briefly as we will not use it much in our discussion of DL.

In our proof of Theorem 3.1 we used a union bound over all $h \in \mathcal{H}$. For infinite-size classes this argument doesn't apply. While the number of hypotheses may be infinite, one could perhaps prove a bound by exploiting the property that hypotheses are similar to each other. VC-dimension provides a notion of complexity of infinite hypothesis classes and in a precise sense characterizes their PAC-learnability. This applies only to the case when the labels are binary and we restrict to the case $Y = \{0, 1\}$ in this section. The hypotheses of this type can also be thought of as sets.

To define the VC-dimension of a hypothesis class $\mathcal{H}$ we first need to define the concept of *shattering*. A hypothesis class $\mathcal{H}$ shatters a set $C = x_1, \ldots, x_m \subseteq X$ if the set $\{(h(x_1, \dots, h(x_m))) : h \in \mathcal{H}\}$ contains all possible 2^m binary vectors of length m. The VC-dimension of $\mathcal{H}$ is the largest m for which there is a set C of size m shattered by $\mathcal{H}$.

Examples.

(1) The class of axis-aligned rectangles in the plane has VC-dimension 4.
(2) The class of half-spaces in $\mathbb{R}^d$ has VC-dimension $d+1$.
(3) More generally, fix a degree $D \geq 1$. Then every degree D polynomial in d variables $x_1, \dots, x_d$ defines a subset of $\mathbb{R}^d$ given by $\{(x_1, \dots, x_d) : p(x_1, \dots, x_d) \geq 0\}$. The class of these subsets has VC-dimension at most $\binom{d+D}{d}$. Note that the case of half-spaces in the previous item is a special case with $D = 1$.
(4) The class of L_2-balls in $\mathbb{R}^d$ has VC-dimension at most $\binom{d+2}{d}$. This follow from the previous item.
(5) The class of polygons in the plane with at most n sides has VC-dimension $2n+1$.
(6) VC-dimension of neural networks is quite well-understood : If W is the number of weights and L is the number of fully-connected layers in a ReLU network, then the VC-dimension is $O(WL \log W)$ and $\Omega(WL \log(W/L))$. See [BHLM17] for full description of the result.

The importance of VC-dimension in learning is underscored by the following theorem:

Theorem 5.1 (The fundamental theorem of statistical learning)**.** *For a domain X let $\mathcal{H}$ be a hypothesis class of functions of type $X \to \{0,1\}$ and let the loss function be 0-1 loss. Then $\mathcal{H}$ is agnostically PAC-learnable with at most $O(\frac{VC(\mathcal{H})+\log 1/\delta}{\epsilon^2})$ samples. The same number of samples also guarantees that the sample is ϵ-representative with probability at least $1-\delta$: $L_{\mathcal{D}}(h) \leq L_S(h) + \epsilon$ for all $h \in \mathcal{H}$.*

This implies that the examples of hypothesis classes mentioned previously are agnostically PAC learnable with a finite number of samples.

To appreciate the power of the theorem

6. Rademacher complexity

VC-dimension is a worst-case notion in the sense that the sample complexity bounds it provides hold for a given hypothesis class and *all* distributions D. In other words, the bounds provided by VC-dimension are distribution-independent. When the VC-dimension is small this leads to strong results, but if the VC-dimension is large it's still possible that for specific distributions sample complexity is not large. Rademacher complexity is a more recent and more versatile notion which is also a notion of the complexity of the hypothesis class but it also depends on the distribution D. Moreover, Rademacher complexity is defined for real-valued function classes unlike VC-dimension which is defined only for Boolean-valued functions (though there are generalizations of VC-dimension).

Why is Rademacher complexity more than renaming?

7. Sample Compression bounds

The ERM rule suggests that we output a hypothesis h that is maximally consistent with the data, i.e. minimizes $L_S(h) = 0$. Often it's possible to show that h can be fully specified by a small subset of the data S. Let's see some examples. (1) Learning axis-aligned rectangles in the plane with $X = \mathbb{R}^2$ and $Y = \{0,1\}$. Each $\mathcal{D}$ corresponds to an axis aligned rectangle R, and if $x \in R$ then the correponding $y = 1$ and if $x \notin R$ then $y = 0$. We take $\mathcal{H}$ to be the class of axis-aligned rectangles. Given samples S generated from such a distribution, there are many h (i.e. rectangles) that may be consistent with S (i.e. $L_S(h) = 0$). But some of them can be specified with just four samples from S: take the left-most, right-most, top-most and bottom-most points from S with label 1. From these four examples we can reconstruct the labels of all other examples in S. Thus (one of the) ERM hypotheses can be specified with just four sample points. Exercise: Generalize this to the d-dimensional space.

(2) Union of k intervals on the real line. Exercise: Show that $2k$ samples suffice.

(3)

Sample compression schemes can be used as learning algorithms. We now formalize the notion of sample compression scheme motivated by the examples above and then prove a generalization bound.

While the proof below is short and simple they are also subtle. For simplicity we will work in the realizable setting. The agnostic setting is similar but more complex and is discussed in [SSBD14].

Definition 7.1. Let $\mathcal{H}$ be a hypothesis class of functions of type $X \to Y$. A compression scheme for $\mathcal{H}$ consists of A sample compression scheme of size k for $\mathcal{H}$ consists of two maps

- a compression map $C_m : Z^m \to \binom{[m]}{k}$ for every $m \geq k$,
- a decompression map $D : Z^k \to \mathcal{H}$,

with the following property: for any $h \in \mathcal{H}$, for arbitrary $x_1, \dots, x_m \in X$, we can compress the sample sequence $S = ((x_1, h(x_1)), \dots, (x_m, h(x_m)))$ using the compression map $C(S)$ to get a subset $I = \{i_1, \dots, i_k\} \subset [m]$. Then applying the decompression map D to the restriction S_I, produces a hypothesis $D(S_I) = h'$ such that $L_S(h') = 0$.

If a hypothesis class admits a sample compression scheme then this scheme can also serve as a good learning scheme as proved below.

Theorem 7.1. *In the setting of Definition 7.1, let $\mathcal{D}_X$ be any distribution on X and let $x_1, \dots, x_m \sim X$ be chosen i.i.d., and let $S = ((x_1, h(x_1)), \dots, (x_m, h(x_m)))$. Then the probability that $L_\mathcal{D}(h') > \epsilon$ is at most $\binom{m}{k}(1-\epsilon)^{m-k}$. (Here $\mathcal{D}$ is the distribution of $(x, h(x))$.)*

Proof. For given sample $S \in Z^m$ there are $\binom{m}{k}$ potential values for the set $I = C(S)$. Each I gives a hypothesis $h_I = D(S_I)$. Now fix I arbitrarily. The hypothesis h_I depends only on S_I and is independent of $S_{[m]\setminus I}$. Thus if $L_\mathcal{D}(h_I) \geq \epsilon$, then the probability that $L_{S \setminus I}(h_I) = 0$ is at most $(1-\epsilon)^{m-k}$. By the union bound, the probability that there is an I, such that $L_\mathcal{D}(h_I) \geq \epsilon$ and yet $L_{S\setminus I}(h_I) = 0$ is at most $\binom{m}{k}(1-\epsilon)^{m-k}$. This means that with probability at most $\binom{m}{k}(1-\epsilon)^{m-k}$, the output hypothesis h' (which satisfies $L_S(h') = L_{S\setminus I}(h') = 0$) we have $L_\mathcal{D}(h') \geq \epsilon$. □

Does this work for non-binary classification or for regression?

References

[BHLM17] Peter L. Bartlett, Nick Harvey, Chris Liaw, and Abbas Mehrabian. Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks, 2017.

[CKY08] Rich Caruana, Nikolaos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, pages 96–103, 2008.

[CN06] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 161–168, 2006.

[Dom12] Pedro M. Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87, 2012.

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, New York, NY, USA, 2014.