COMP34312: Mathematical Topics in Machine Learning

Notes pack last updated: January 30, 2024

Gavin Brown *Gavin.Brown@manchester.ac.uk*

Anirbit Mukherjee *Anirbit.Mukherjee@manchester.ac.uk*

In this module, we will not to teach you a list of the latest fashionable ML models. You can learn that from other online resources. Even if we did, given how fast the field moves, they'd be out of date in 1-2 years. Instead, we decided to help you understand a fundamental open question, that challenges the state of the art in our field. The topics were selected to be relatively close to our research interests, meaning we can help you understand some of the very latest issues. So, here we go....

Modern ML is going **BIG**. It seems like Google or Facebook put out a press release every other week, about their latest 100 billion parameter deep learning model. Or is it now 200 billion, or 500 billion parameters? This module will give a formal, mathematical treatment to the following question:

"Are bigger models always better models?"

For example, if we keep making bigger and bigger neural networks, will they just keep getting smarter? Is scale really all we need? There's a lot of hype out there. It's hard to know what's real.

The simple answer is 'no'. The performance of a machine learning model is determined by a combination of factors, including the quality and quantity of the training data, the choice of model architecture, and the skill of the person tuning the model. These are all practical issues, that vary with the skill of the practitioner, and availability of compute/data resources. There are however, several fundamental/theoretical issues that apply to everybody. We aim to give you the mathematical and conceptual tools to discuss all this in an informed manner. **We focus on theoretical issues, i.e. there will be no coding of large models in this module.**

Lecture: Tuesdays, 1pm, Simon Lecture Theatre A.
Examples class: Wednesdays 12pm / Thursdays 4pm. Kilburn Building, g23.
Assessment: 100% closed-book exam in late May.

Chapter...	is for the lecture on...	is about...
1	30 January	Recap of COMP24112
2	6 February	Empirical Risk Minimisation
3	13 February	Generalisation bounds
4	20 February	Bias & Variance
5	27 February	Ensemble Learning
6	5 March	Theory of Ensembles
7	12 March	Some Background mathematics
	EASTER BREAK	
8	9 April	Gradient Descent
9	16 April	GD in over-parameterized models
10	23 April	Rademacher Complexity 1
11	30 April	Rademacher Complexity 2

The module will be delivered in weeks 1-6 by Gavin, and in weeks 7-11 by Anirbit.

Contents

0 Notation used in this module

There will be a *lot* of mathematical notation over the next 11 weeks. We have tried to collect the majority of this notation here, for quick reference. At the beginning of the module, you won't know the meaning of all the terms, but they will get slowly used over the weeks, so have patience, and don't panic.

$\boldsymbol{x}$	Input vector
$\boldsymbol{y}$	Output/target vector
$\mathcal{D}$	A probability distribution over examples $(\boldsymbol{x}, \boldsymbol{y})$
$\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})}[\cdots]$	Expectation (i.e. average) over possible examples drawn from $\mathcal{D}$
$\mathcal{S}_n$	A data set $\mathcal{S}_n \coloneqq \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \mid i = 1, \ldots, n\}$, where each $(\boldsymbol{x}, \boldsymbol{y}) \sim \mathcal{D}$
$\mathbb{E}_{\mathcal{S}_n}[\cdots]$	Expectation (i.e. average) over possible datasets $\mathcal{S}_n$
d	Dimension of $\boldsymbol{x}$ (i.e. number of features)
k	Dimension of $\boldsymbol{y}$ (e.g. number of classes)
$f(\boldsymbol{x})$	A model predicting $\boldsymbol{y}$, evaluated at a point $\boldsymbol{x}$.
$\ell(y, f(\boldsymbol{x}))$	A (non-negative) loss function, evaluated at a point $\boldsymbol{x}, \boldsymbol{y}$.
$R(f) \coloneqq \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\ell(\boldsymbol{y}, f(\boldsymbol{x}))\right]$	Population risk of a model
$\hat{R}(f, \mathcal{S}_n) \coloneqq \frac{1}{n}\sum_{i=1}^{n}\left[\ell(\boldsymbol{y}_i, f(\boldsymbol{x}_i))\right]$	Empirical risk of a model, evaluated on a set $\mathcal{S}_n$
Ω	The space of all possible (measurable) functions.
$\mathcal{F} \subset \Omega$	A function class, as a subset of all possible functions.
$f_{erm} \coloneqq \operatorname*{arginf}_{f\in\mathcal{F}}\left[\hat{R}(f, \mathcal{S}_n)\right]$	Empirical risk minimizer in $\mathcal{F}$
$f^* \coloneqq \operatorname*{arginf}_{f\in\mathcal{F}}\left[R(f)\right]$	Population risk minimizer in $\mathcal{F}$.
$y^* \coloneqq \operatorname*{arginf}_{f\in\Omega}\left[R(f)\right]$	Population risk minimizer in Ω, known as the Bayes-optimal model.
$\hat{\mathcal{R}}_n(\mathcal{H})$	Empirical Rademacher complexity of a function class $\mathcal{H}$
$\mathcal{R}_n(\mathcal{H})$	Averaged Rademacher complexity of a function class $\mathcal{H}$
	Given a $\mathcal{F}$ and ℓ as above our typical instance of $\mathcal{H}$ will be the corresponding "loss class", $\mathcal{H} = \{(\boldsymbol{x}, y) \mapsto \ell(y, f(\boldsymbol{x})) \in [0, \infty) \mid f \in \mathcal{F}\}$. Though in the above setups we would also be interested in the Rademacher complexity of $\mathcal{F}$ and accordingly $\hat{\mathcal{R}}_n(\mathcal{F})$ and $\mathcal{R}_n(\mathcal{F})$ will be defined.
$F : \mathbb{R}^p \to \mathbb{R}^q$	An arbitrary function which we may use with gradient descent.

1 Recap of your 2nd year ML module, COMP24112

Motivation. This week we will cover what this module is about, as well as reviewing the necessary background material in the basics of Machine Learning.

We assume you have a good understanding of everything in COMP24112. However, there are some topics where we will focus more than others. In particular, COMP24112 chapters 5 and 6, on loss functions and training schemes. In this chapter we will briefly review this, though not in depth—refer to your COMP24112 notes for full information. The accompanying lecture will cover some of this, plus review the current issues around large models.

Learning Objectives

By the end of this week you should be able to define and use the following concepts correctly in a conversation.

- the training error, testing error
- overfitting versus underfitting
- the squared loss,
- the cross-entropy loss,
- the gradient of a loss, and global vs local minima,
- gradient descent versus stochastic gradient descent,
- regression trees, and the meaning/importance of tree 'depth'.

1.1 Supervised Learning basics: models and loss functions

In this module we will focus on *supervised learning* tasks. For any input, we have access to a 'label' (sometimes known as a 'target'), which we regard as the correct response for the input. Though it wasn't mentioned last year, the underlying principle for most supervised learning is a mathematical framework called *empirical risk minimization*, which will be introduced over the coming weeks.

Let's look at a concrete (but toy) learning problem, which will illustrate our points. So, here's your *training set*, of $n = 20$ datapoints. Each input is a value on x-axis, the target label you get is the corresponding point on the y-axis. This is a result of some *true* underlying function, plus some noise, i.e. $y = f_{true}(x) + \epsilon$, where $\epsilon = \mathcal{N}(\mu = 0, \sigma = 0.5)$, a random value drawn from a Gaussian with mean 0 and standard deviation 0.5.

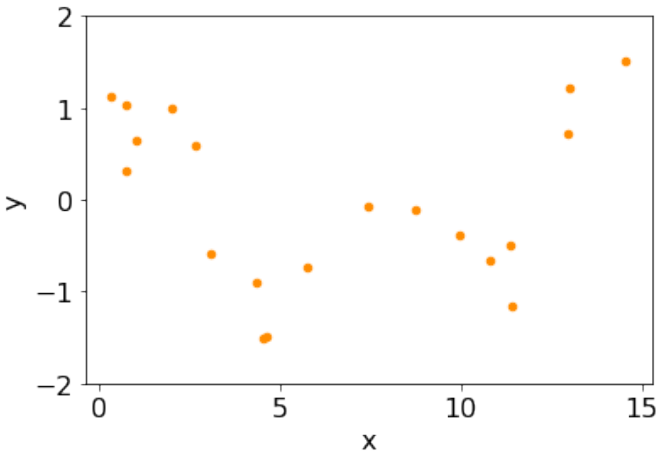
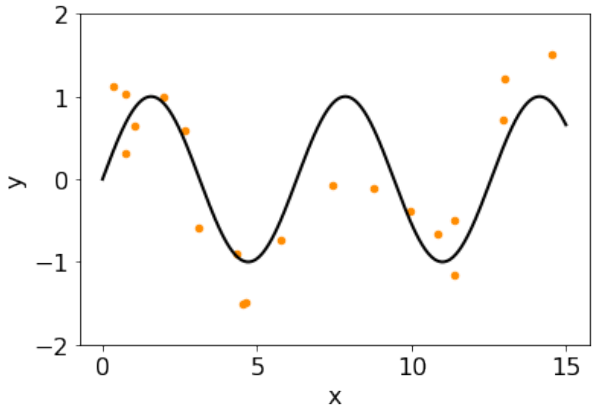


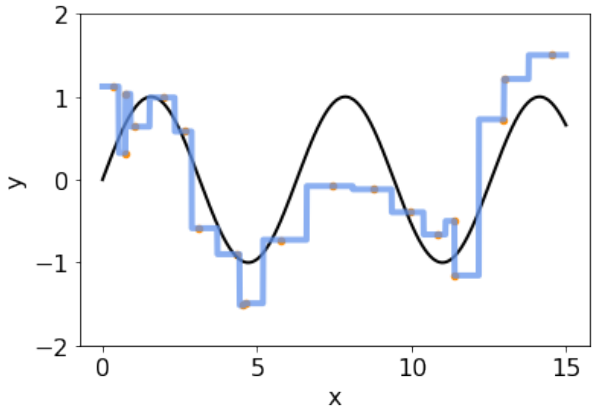
Figure 1: A sample of $n = 20$ points to use as a training set. Can you tell what the true function is?

The task is to learn what the function f_{true} is, from this small (and noisy) sample. Can you see what it is? I'd be surprised if you could...

In fact it's $f_{true} = sin(x)$, shown above. The sample is so small, and so noisy, that it missed the middle



'peak' of the sine wave. Let's see what a Machine Learning model can do with this data. We will use a k-nearest neighbour regression, with a squared distance measure. **If you don't remember what k-nn regression is, go back to your COMP24112 notes (Lecture 2A) for some revision.** The fitted model looks like this:



The blue line shows the fitted predictions of the k-nn regression model, with $k = 1$. We fitted very well (perfectly in fact) to this training set. However, we can see visually that performance on the true underlying function (black line, sine wave) was *terrible*.

We used $k = 1$ here—this determines the *fit* of the model. Maybe if I give you more data, and try different parameters, we can get different performance. Trying $k = 1$, $k = 8$, and $k = 20$, we get the following fits to our new bigger dataset, where $n = 100$.

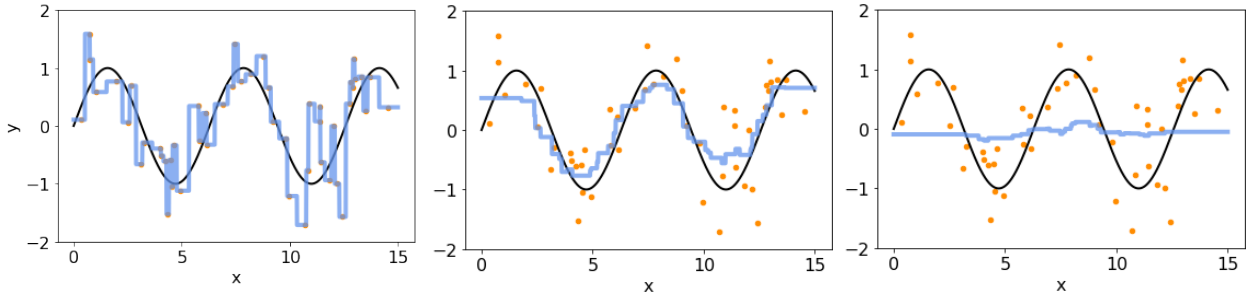


Figure 2: LEFT: $k = 1$, overfitting. MIDDLE: $k = 8$, a good fit. RIGHT: $k = 20$, underfitting.

Hopefully you are remembering some keywords from COMP24112, **over-fitting** and **under-fitting.** The model for $k = 1$ (left) is **over-fitted** because it is "fine-tuned" to the training data, unable to represent the true function—i.e. it has high testing error. The model for $k = 20$ (right) is **under-fitted**, because it has virtually ignored the training data, and is again unable to represent the function, giving again high testing error. The model for $k = 8$ (middle) is a good fit, at least as can be seen visually. As you can see, the idea of over- and under-fitting are intrinsically linked to the notion of *complexity*. The model on the left is very *complex*. The model on the right is very *simple*.

The k-nn is a *memory-based* algorithm, also known as *instance-based learning*. It just memorises the training data, then returns the average label of the k closest neighbours in the feature space, where 'close' is defined by a distance measure—in our case we chose squared distance. This necessitates storing all the training data in memory. As a consequence, it's not great at generalising beyond this training data to new scenarios. So, let's consider other types of models, which are sometimes known as *function approximators*. You met a couple of these in COMP24112, i.e. linear regression, and neural networks, but there are many others:

- Linear/Polynomial Regression
- Support Vector Machines
- Neural networks (e.g. convolutional neural nets, but also logistic regression)
- Naive Bayes (or more generally, probabilistic models)
- Decision Trees (for both regression and classification)
- Ensemble models (i.e. combining some of the models above as a 'committee' model)

The thing that these models have in common is that they work by trying to minimise **loss functions.** Let's see some details of two very common losses.

Loss functions

We can denote a generic model as $f(\boldsymbol{x})$. Here, $\boldsymbol{x}$ is the feature vector, which we will assume is a real-valued vector of length d, or more formally $\boldsymbol{x} \in \mathbb{R}^d$. **In this notation, we have left it implicit that the model has some parameters**, which we denote $\boldsymbol{w} \in \mathbb{R}^p$, i.e. there are p parameters. The model might be for example a neural network, but as you see in the list above, there are many other non-neural models that can perform the same job. For a given model f, we evaluate its performance with a loss function.

For *regression* problems, we predict a value in $\mathbb{R}$, and a very common loss is the *squared* loss function:

$$\ell(y, f(\boldsymbol{x})) = (y - f(\boldsymbol{x}))^2. \tag{1}$$

Thus, we have a measure of performance for our model f, in predicting the correct value y based on input $\boldsymbol{x}$. We have a *training dataset* of n points: $\mathcal{S}_n^{train} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, and we define the *training error* as the loss averaged over this set:

$$\ell_{train}(f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\boldsymbol{x}_i))^2. \tag{2}$$

We then modify the parameters $\mathbf{w}$ so as to minimise this training error:

$$\mathbf{w}^* \;:=\; \underset{\mathbf{w}}{\operatorname{argmin}}\Big[\ell_{train}(f)\Big]. \tag{3}$$

However, our ultimate measure of performance is not the training error, but the performance on a final testing set, that the model has never seen. You can think about this like some lectures (your training set) and a final exam (your testing set).

For a *classification* problem, our model f will output a vector of predicted class probabilities, and we use the *cross-entropy* as our loss. As a reminder, for two classes, the cross-entropy is:

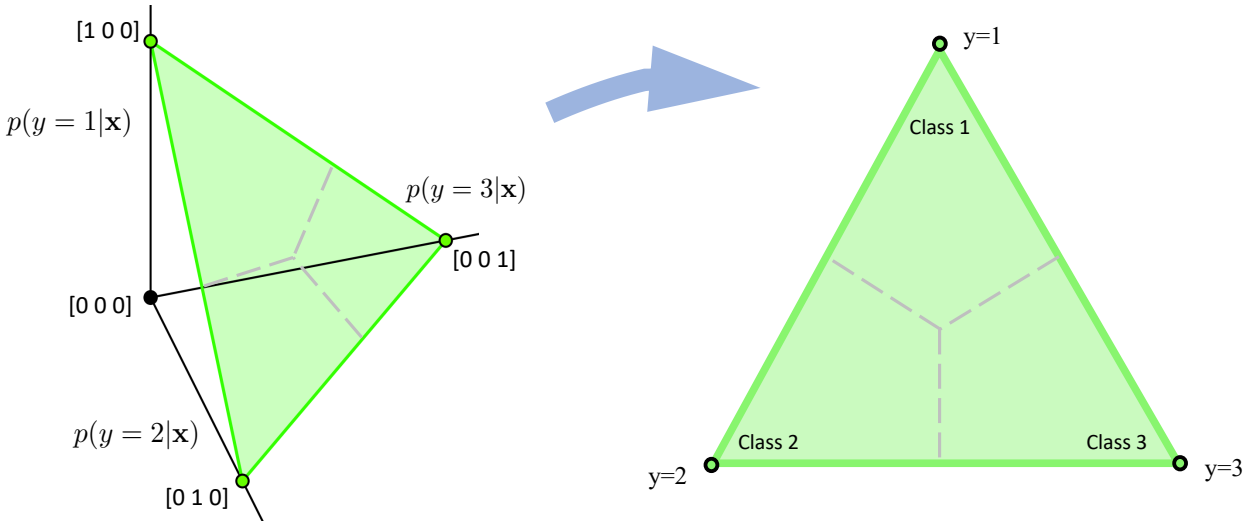
$$\ell(y, f(\boldsymbol{x})) \;:=\; -\Big[y \ln f(\boldsymbol{x}) + (1-y)\ln(1-f(\boldsymbol{x}))\Big] \tag{4}$$

where $y \in \{0,1\}$ and $f(\boldsymbol{x}) \in (0,1)$. For the general multi-class case, it is:

$$\ell(\boldsymbol{y}, f(\boldsymbol{x})) \;:=\; -\Big[\boldsymbol{y} \cdot \ln f(\boldsymbol{x})\Big] \tag{5}$$

where $\boldsymbol{y}$ is a **one-hot** vector. This is a vector of all zeroes, apart from one, which is 1 in the index of the true class. For example, if we have 3 classes, and the correct class for a given example is 3, then $\boldsymbol{y} = [0,0,1]$.

Our model f returns a vector of probability estimates for each class, e.g. $[0.2, 0.3, 0.5]$. We can visualise the 3-class case as a *probability simplex*, where a single prediction $f(\boldsymbol{x})$ is a point on the simplex.



Again, refer back to COMP24112 notes (lectures 5A, 5B, and 5C) for some revision.

Concept Check...
I give you an example where the true label is $y = 1$.
ho My model predicts the probability of $y = 1$ as $f(\boldsymbol{x}) = 0.95$.

Q1. Calculate the cross-entropy loss. Remember to use the **natural** logarithm.

Now, I give you an example $\boldsymbol{x}$, where $\boldsymbol{y} = [1,0,0]$.
My model predicts $f(\boldsymbol{x}) = [0.3, 0.6, 0.1]$.

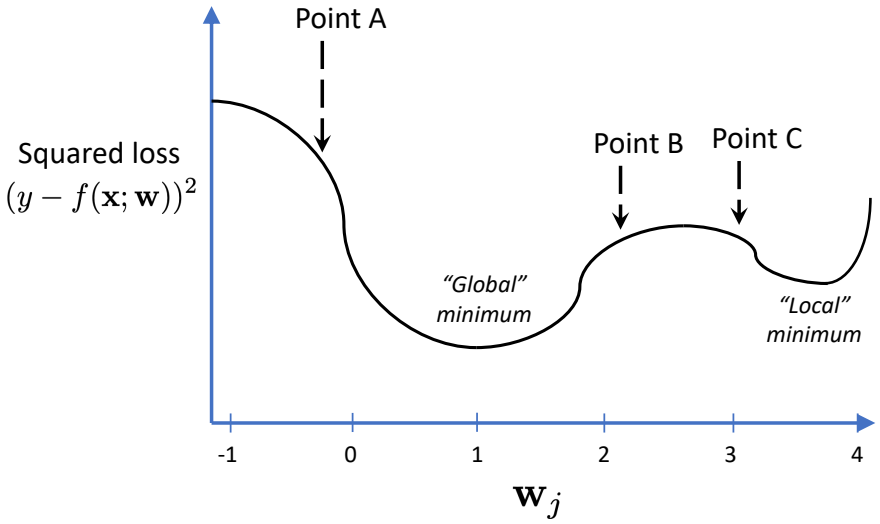
Q2. Calculate the cross-entropy loss again.

Q3. Roughly where in the simplex does this model sit?

1.2 Gradient Descent

One of the most heavily used principles for a learning algorithm is *gradient descent*. In the final weeks of this module, we will be studying advanced aspects of GD, so it's important you understand the basics.

Imagine we have a squared loss function, applied to the output of a big neural network. Somewhere within that neural network there is a weight, $\mathbf{w}_j$, that we decide to change. We vary $\mathbf{w}_j$ *manually* up and down, and record what the squared loss is. The result looks like this:



But we can't manually plot the loss like this, as it's too computationally expensive. If we could, we'd know immediately to set our parameter to the **'global' minimum**, where $\mathbf{w}_j = 1$. And, we'd want to avoid the inferior **'local' minimum** where $\mathbf{w}_j = 3.7$, or indeed any other place on the landscape. But we can't. Instead we can only guess some initial value for $\mathbf{w}_j$, measure the loss, and then guess how we should change it. Consider the situation if we started from point A, B, or C, above.

The factor that differentiates point A from B is the *gradient*, i.e. the slope of the function. Point A is a negative (downhill) gradient, while point B is a positive (uphill) gradient. The principle of gradient descent is to minimize a loss, in the following manner.

> **If the gradient is NEGATIVE, we INCREASE the parameter.**
>
> **If the gradient is POSITIVE, we DECREASE the parameter.**

You can see that by increasing the parameter from point A, we do indeed approach the global minimum. The reverse is true for point B, decreasing the parameter would cause us to minimise the loss. It's important to see that, if we had followed this principle from point C, we would have ended up at a sub-optimal place, the 'local' minimum.

We can calculate the gradient *at any location* by taking derivative with respect to $\mathbf{w}_j$. Remembering that we are using the squared loss,

$$\ell(y, f(\boldsymbol{x})) = (y - f(\boldsymbol{x}))^2, \tag{6}$$

then the gradient is:

$$\frac{\partial \ell(y, f)}{\partial \mathbf{w}_j} = \frac{\partial \ell(y, f)}{\partial f(\boldsymbol{x})} \frac{\partial f(\boldsymbol{x})}{\partial \mathbf{w}_j} = 2\Big(f(\boldsymbol{x}) - y\Big) \frac{\partial f(\boldsymbol{x})}{\partial \mathbf{w}_j}. \tag{7}$$

If you don't know how this gradient is calculated, you should remind yourself of basic calculus—see also Lecture 6C from COMP24112.

Take a look at the landscape on the previous page. If we were at point A (negative gradient) we would increase $\mathbf{w}_j$, exactly as we decided we needed to do in the plotted landscape, and similarly for point B (positive gradient). This can be built into an algorithm.

Algorithm 1 Gradient Descent for parameter vector $\mathbf{w}$ on a differentiable loss $\ell(y, f(\boldsymbol{x}))$

1: **Choose :** A value $T > 0$ for the number of steps to take.
2: **Choose :** A constant η, for the size of each step.
3: **Input:** An initial vector $\mathbf{w}$ of length p.
4: **for** $t = 1, \ldots, T$ **do**
5: $\quad \mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \nabla \ell(y, f(\boldsymbol{x}))$
6: **end for**
7: **Output : w**

Notice that here we used the 'nabla' notation. ∇. which denotes the fact that $\boldsymbol{w}$ is a vector. For a vector $\boldsymbol{w}$ of length d, we have the gradient vector:

$$\nabla \ell\Big(y, f(\boldsymbol{x})\Big) = \left[\frac{\partial \ell(y, f(\boldsymbol{x}))}{\partial \mathbf{w}_1}, \frac{\partial \ell(y, f(\boldsymbol{x}))}{\partial \mathbf{w}_2}, \ldots, \frac{\partial \ell(y, f(\boldsymbol{x}))}{\partial \mathbf{w}_d}\right]^T. \tag{8}$$

Notice also in the above, we are calculating the gradient only for a *single* datapoint $(\boldsymbol{x}, y)$. In reality we will have a full training set to deal with, so what does this mean?

Look at line 5 in the above algorithm. This is known as the **update rule** for gradient descent. If we had a training set of n datapoints, we could replace this with the following, sometimes known as **full batch gradient descent**.

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \left[\nabla \ell(y_i, f(\boldsymbol{x}_i))\right] \tag{9}$$

Alternatively, we could randomly pick a sample $\mathcal{S}$ of m datapoints, and use the following update rule, known as **mini-batch gradient descent**:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \frac{1}{m} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{S}} \left[\nabla \ell(y_i, f(\boldsymbol{x}_i))\right]. \tag{10}$$

If we take the extreme, $m = 1$, this is **stochastic gradient descent** (SGD), though terminologies vary across the ML community, so you may see this referred to as SGD even if $m > 1$. One reason to use a smaller batch size is to have stochastic estimates of the gradient, which tend to help in escaping local minima.

Note: The above algorithm is a *first-order* gradient descent algorithm. There are alternatives, which utilise second-order derivatives of the loss landscape, for example the popular ADAM algorithm.

Concept Check...
Prove Equation 7.

1.3 Decision trees

One important model family is *decision trees*. These are used widely in industry, as they are fast to both train and deploy. They are excellent models for so-called **tabular** data, which means basically anything that fits in a spreadsheet—not images, or speech signals, or videos.

Decision trees come in two types—for classification, and regression. The basic idea is to recursively split the dataset into subsets based on the values of input features, and then fit a simple model (such as a constant label, or a linear regression) to each subset. Once the tree is built, it can be used to predict the target variable for new samples by traversing the tree from the root to a leaf node, and then using the simple model associated with that node to make the prediction. Below you can see a classification tree, partitioning a 2D space and predicting the probability of a blue circle in each sub-area.

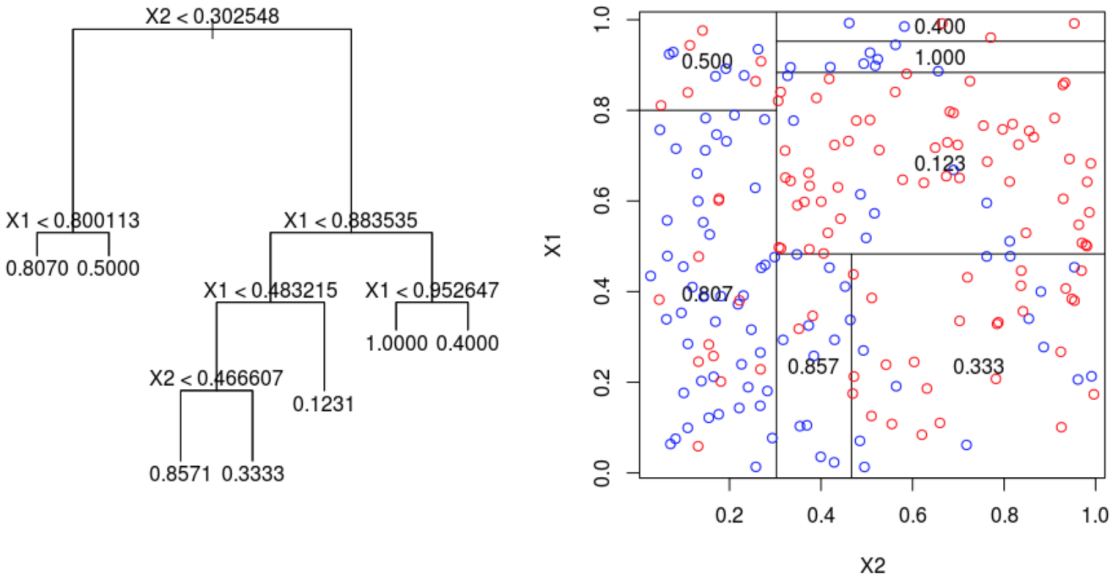


Figure 3: A classification tree.

Regression trees look similar to this, but in each subset of the data they predict a **continuous value**, such as the price of a house. The recursive splits are chosen in such a way as to minimize the variability of the target variable within each subset. We won't go into details of the training procedure here.

With every ML model, there are parameters for you to tune. With trees, the main one is how *deep* it is. This translates to a more fine grained partitioning of the space shown above, equating to a more complex decision boundary. In Figure 5 you can see the decisions from a regression tree as I increase the depth. This is using the sine wave toy problem shown earlier.

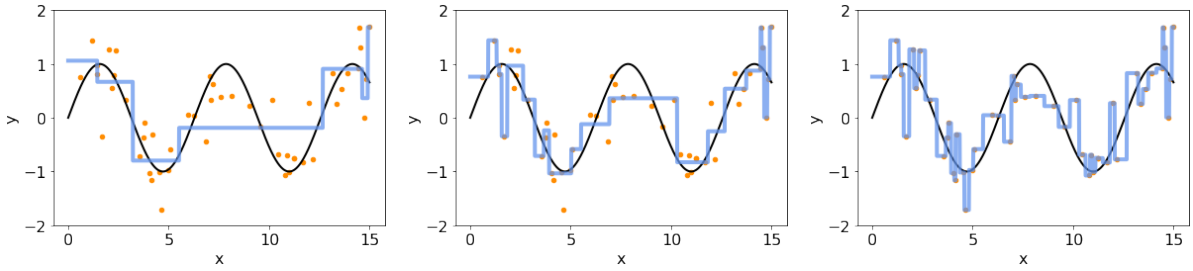


Figure 4: Decision boundaries for trees of increasing depth. From left to right, depth 3, 5, and 16.

Plotting the full range of depth from 1 to 16 reveals that over-fitting occurs around depth 5 or 6. Below we see this, where the darker red lines are averages over 50 repeats of the experiment, and the faint red lines are the 50 individual runs.

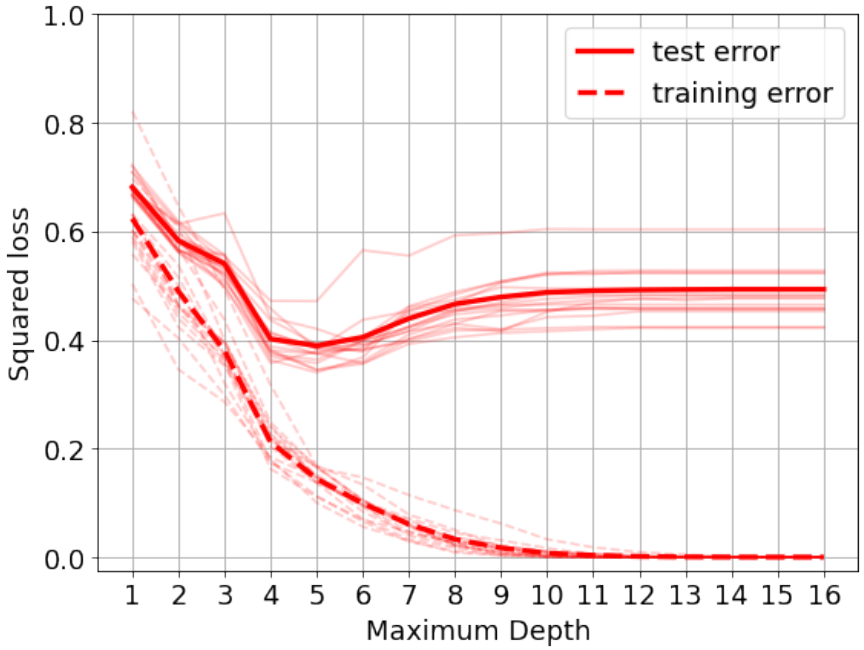


Figure 5: Training and testing loss, as we increase the depth of a regression tree.

1.4 Summary and Outlook for this Module

This brings us to the main question for the module—are bigger models always better? The deep trees on the right are very BIG. They are very deep, and make a complex decision boundary. But you can see they have over-fitted. So here, certainly very big models are not doing well. However, you'd have to have been living under a rock in the desert to not notice the popular press talking about 'deep learning' in the past few years. The idea here is you make very deep/complex neural networks with lots of parameters to tune. And they are doing AMAZING things. You may have heard of MidJourney, Stable Diffusion, and most recently chatGPT. These are models with BILLIONS of parameters to learn.

In this module we will address the theoretical issues around such large models, starting next week with the principle of *empirical risk minimisation*, leading to the wider framework of *statistical learning theory*. SLT provides a formal language to understand the performance of machine learning algorithms and to make informed decisions. It can be used to analyze the robustness and stability of machine learning algorithms, which can be important in safety-critical applications. It also gives an insight on the "sample complexity" of the algorithms which helps in understanding how much data is required to train a model with a certain level of accuracy, this can be useful in cases when data is scarce or expensive to collect.

To prepare for next week, make sure everything in this week's session is **easy** for you. We'll encounter much more challenging mathematics in the weeks to come.

2 Empirical Risk Minimization

Motivation. Today we start the journey addressing our question "are bigger models always better models?". To do this, we will study the mathematical framework of 'statistical learning theory', which we can use to reason about Machine Learning models, data, and algorithms. The first step on the journey is 'empirical risk minimisation', which is the basis of almost all supervised learning algorithms. In fact, you've met this before—you just didn't know the formal details....

Learning Objectives

By the end of this week you should be able to define and use the following concepts correctly in a conversation.

- **loss**, **empirical risk**, and **population risk** of a model
- empirical risk minimizer
- the best model within a model family/class
- Bayes model
- approximation error
- estimation error
- optimisation error

2.1 Let's start with some terminology

Models. Let's say we want to solve a *classification problem*, e.g., our ML model should take an image and predict whether it's a cat or a dog. We can denote this in abstract mathematical form (i.e. not referring to an particular *type* of ML model) as a function $f : \mathcal{X} \to \mathcal{Y}$. The notation here means a function that maps from some *input* space $\mathcal{X}$ to some *output* space $\mathcal{Y}$. The form of the function itself is determined by the choice of model—e.g., what architecture of neural network, or whether you choose a decision tree, or a logistic regression, or any other model that you've heard of. We denote the **family of models** that we choose by the set $\mathcal{F}$. This is of course a subset of the space of all possible models, which we denote by Ω. It's an important thing to realise that $\mathcal{F}$ is a subset of Ω, i.e. we *always* have a restricted choice of models—the matter of precisely *how* you restrict the model is the key to solving the supervised learning problem.

Input and Output spaces. For a classification problem, the input space is $\mathcal{X} = [0, 255]^{128\times128}$, i.e. an image of resolution 128 by 128 pixels, with each pixel in the range 0 to 255. The output space is $\mathcal{Y} = \{cat, dog\}$, or perhaps for convenience of manipulating the data in a computer we might say it's $\mathcal{Y} = \{0, 1\}$. We might instead define this output space to be a *probabilty* specifying the probability of the image being a dog—then $\mathcal{Y} = [0, 1]$. In general, we will say that our input space is of dimension d and output space is dimension k. In the above example, $d = 16384 = 128\times128$, and $k = 1$ if we predict the probability. For simplicity below, we will assume $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}^k$, but many different learning scenarios and challenges are raised by considering the form these two spaces take.

Data sets. You've been taught the concept of a *training* set, and a *testing* set. In these, each possible data point $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^d \times \mathbb{R}^k$ is assumed to be an independent *sample* from a distribution $\mathcal{D} = P(\boldsymbol{x}, \boldsymbol{y})$. The overall data set $\mathcal{S}_n$ is then a sample from a joint random variable $P(\boldsymbol{x}, \boldsymbol{y})^n$, i.e. sampling n times, independently. Formally, this is known as the *independent and identically distributed* (iid) assumption.

Loss Functions. The framework of *Empirical Risk Minimization* rests on the definition of a *loss function*, which specifies the quality of a prediction, relative to the true answer. This loss function can be written in similarly abstract form, $\ell : \mathbb{R}^k \times \mathbb{R}^k \to [0, \infty)$, meaning it takes two values (the true label and the model prediction), and returns a loss that ranges from 0 upwards (not including infinity). Note, by historical convention, the *first argument* is always the true label, so the loss of model is $\ell(y, f(\boldsymbol{x}))$.

2.2 Empirical versus Population Risk

Until now, you've been taught that the 'error' of a model is the *average loss over a data set*. In the framework of Statistical Learning Theory, this terminology of 'error' is usually referred to as the *risk*. Given a set of data $\mathcal{S}_n = \{(\boldsymbol{x}_i, \boldsymbol{y}_i), i = 1, \ldots, n\}$, we define the **empirical** risk as follows.

Definition 1 (Empirical risk of a model)**.**

$$\hat{R}(f, \mathcal{S}_n) := \frac{1}{n} \sum_{i=1}^{n} \ell(\boldsymbol{y}_i, f(\boldsymbol{x}_i)). \tag{11}$$

The empirical risk can contrasted with the **population risk**, which can be thought of as when $n = \infty$:

Definition 2 (Population risk of a model)**.**

$$R(f) := \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\Big[\ell(\boldsymbol{y}, f(\boldsymbol{x}))\Big] = \int \int \ell(\boldsymbol{y}, f(\boldsymbol{x}))p(\boldsymbol{x}, \boldsymbol{y})d\boldsymbol{x}d\boldsymbol{y} \tag{12}$$

Here the notation $\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}$ should be read as the 'expectation with respect to the true data distribution', or informally, the average over all possible data points. The population risk expresses the error that a model f would make, on an average, over all possible inputs. This is also known as the generalisation error. Note that this *does not yet depend on any choice of training data*. We have not yet talked about training a model, we just assume we have f in our hand, somehow.

Learning a model by ERM. Now, again, you've been taught that to obtain a model, we use a training data sample. First, let's assume we have our finite training data $\mathcal{S}_n$, and some very clever learning algorithm that can somehow find a *global minimum* of the empirical risk. This is is (perhaps unsurprisingly) referred to as an *empirical risk minimizer*.

Definition 3 (Empirical risk minimizer in $\mathcal{F}$)**.**

$$f_{erm} := \operatorname*{arginf}_{f\in\mathcal{F}} \hat{R}(f, \mathcal{S}_n). \tag{13}$$

This is the model we would get if we could find the global minimum[1] on the data sample $\mathcal{S}_n$, when we have restricted ourselves to a model family $\mathcal{F}$. Equally, we can define a *population* risk minimizer :

Definition 4 (Population risk minimizer in $\mathcal{F}$, also known as the "best in family" model)**.**

$$f^* := \operatorname*{arginf}_{f\in\mathcal{F}} R(f). \tag{14}$$

This has removed one of our assumptions—we assumed we have infinite data. The final assumption we have is the restriction of a model family, i.e., using $\mathcal{F}$ instead of Ω. If we remove that final restriction, we can define the population risk minimizer as:

Definition 5 (Population risk minimizer in Ω, also known as the Bayes model)**.**

$$y^* := \operatorname*{arginf}_{f\in\Omega} R(f) \tag{15}$$

This is a **hypothetical** model, the optimal model with no restriction on the model family, and no restrictions on the data. This is also referred to as the *Bayes prediction* or *Bayes model*. For squared loss $y^* = \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[\boldsymbol{x}]$, and for 0/1 loss, $y^* = \operatorname{argmax}_y p(y|\boldsymbol{x})$. **The proof is optional reading at the end of this chapter.** These are however not computable in real scenarios, since it requires the full data distribution (as we can see from the definition of R), to which we don't have access.

[1] Note, the arginf instead of argmin—this acknowledges the fact that the minimizer may not be unique.

Before we proceed any further, it's important you understand these definitions. The empirical risk is the error on a *sample* of data, of a specific size n. The population risk is the *true* risk, where we assume we could somehow get an infinite supply of data. We visualise this below.

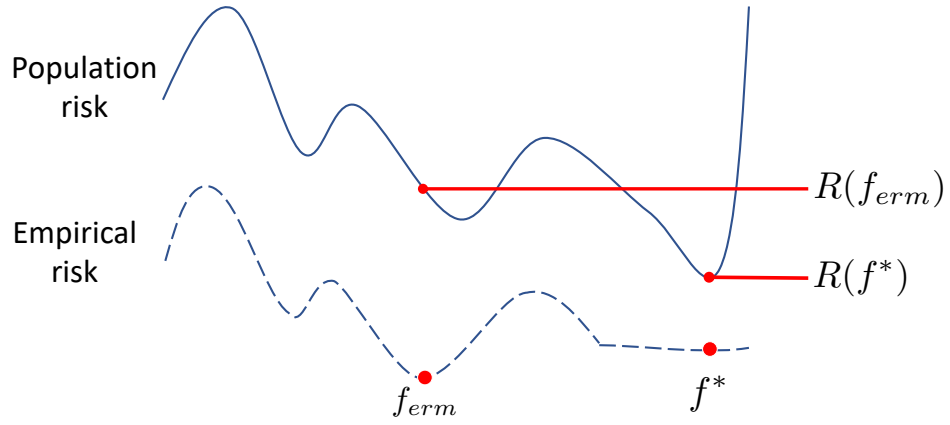


Figure 6: A hypothetical scenario: empirical (i.e. estimated) risk, versus population (i.e. true) risk. Note that here I have offset the landscapes from each other by a constant, so you can see them more clearly. This is fine to do, as the absolute value of the risk does matter, only the gradients and location of the optima.

The horizontal axis represents a chosen space of possible models—you might imagine it corresponds to one parameter somewhere within a neural network. By varying this parameter we obtain different predictive models, with different risks. The model f_{erm} is the global minimum of the empirical risk, and f^* is the global minimum of the population risk. Note that these do not necessarily coincide. This is one of the primary challenges to solve in ML—a small data sample can mis-lead you.

Concept Check...
Note that we have not illustrated the Bayes model y^* in this diagram. Why?

2.3 The Approximation/Estimation decomposition

So far, we have defined three very important models.

- f_{erm} the empirical risk minimizer
- f^* the best model in in our family
- y^* the Bayes model

We can talk about the *risk* for each of these models. The expression $R(f_{erm})$ denotes the population risk of an ERM, and $R(f^*)$ is the population risk of f^*. The term $R(y^*)$ denotes the population risk of the Bayes model, which we conventionally refer to as the *Bayes risk*. Using these, we introduce another important piece of terminology, the **excess risk**.

Definition 6 (Excess risk)**.** *The excess risk of a model f is the difference between its population risk and the Bayes risk, i.e., $R(f) - R(y^*)$.*

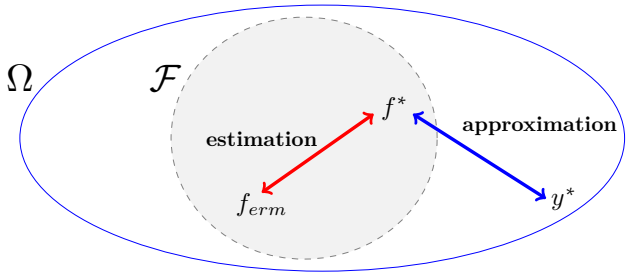
This quantity tells us how much further we could hypothetically reduce the population risk, since we know it is impossible to do any better than the Bayes model y^*. Note that this is the excess risk for an arbitrary model f, but we can also define the excess risk of a different model, e.g., the excess risk of an ERM is the quantity $R(f_{erm}) - R(y^*)$, which will be important to understand the next important concept....

We can now define the so-called **approximation/estimation decomposition**.

Definition 7 (Approximation-Estimation decomposition)**.** *Given an empirical risk minimizer f_{erm}, the excess risk decomposes as follows.*

$$\underbrace{R(f_{erm}) - R(y^*)}_{\text{excess risk of } f_{erm}} = \underbrace{R(f_{erm}) - R(f^*)}_{\text{estimation error}} + \underbrace{R(f^*) - R(y^*)}_{\text{approximation error}} . \tag{16}$$

Whilst this seems a trivial decomposition, the resulting components actually have meaningful interpretations as parts of the excess risk. **Estimation error** is there because we had a limited sample $\mathcal{S}_n$ with which to find f_{erm}. If we had a bigger data set, our ERM would (probably) have been closer to f^*. This quantity depends on the random sample $\mathcal{S}_n$, and thus is a random variable. **Approximation error** is there because we had to restrict our model family. This depends only on the choice of model family, and thus is a constant. We can illustrate[2] the components as follows.

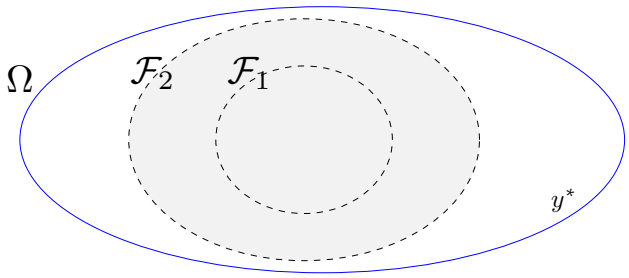


The treats the learning problem in two stages: *approximation*, where we choose a family $\mathcal{F}$; and, *estimation*, where we estimate which model within $\mathcal{F}$ is the best, using our finite data sample.

Changing the size of our model family

Our family, $\mathcal{F}$, might be for example be the weight space for a neural network with a fixed architecture. Or, or all decision trees of a fixed depth. Or, the weight space of a simple logistic regression. The key thing is that we are constrained to only be able to learn things that can be represented by that family of functions $\mathcal{F}$, as a subset of Ω, the space of *all* functions.

Take the neural network example, and call it the set $\mathcal{F}_1$. Now give the network more layers (i.e. make it bigger) and we have the set $\mathcal{F}_2$. These two sets are represented below.

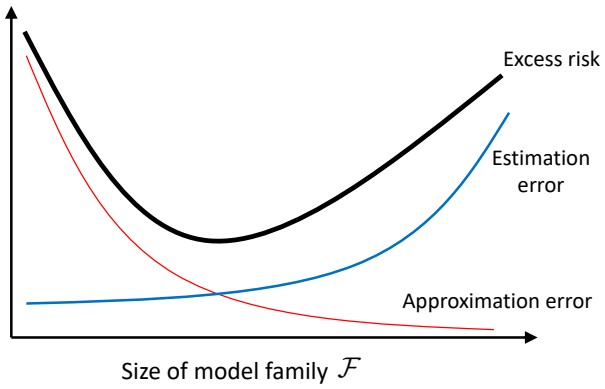


Increasing the complexity of the model has expanded the size of the function class. In other words, a more complex architecture has given us more degrees of freedom, and therefore $\mathcal{F}_2$ can represent more functions than $\mathcal{F}_1$. We also see that in this case, expanding the size of the function class has brought us 'closer' to containing the Bayes model. Be careful, this is not a literal distance, but purely a conceptual illustration.

There are some subtleties here, regarding the relation between the size of the function class, and the complexity of the functions within it. This will become clearer as the course goes on.

[2] Be careful with interpretations of this—it is just an illustration, and the notion of distance is not to be taken literally.

There is a natural trade-off (see below) as we change the size of $\mathcal{F}$, keeping data size fixed. We denote the size of the function class by $|\mathcal{F}|$. As we increase this, approximation error will likely decrease (potentially to zero, if $y^* \in \mathcal{F}$), but estimation error will increase, as it becomes harder to find f^* in the larger space. The reason it is harder, is not just the vast number of functions to evaluate, but also the problem of distinguishing between them with a fixed amount of data. This is in effect the multiple hypothesis testing problem[3]. Since the excess risk is the sum of the two, this tends to give a U-shaped curve. **Do you recognise this phenomenon?**



Recall the idea of model 'over-fitting' from previous modules. If your model is too much fine-tuned to its training data, then it will tend to perform badly on testing data. This is over-fitting.

Overfitting tends to occur when our function class is too **large**, relative to the amount of data we have.

Underfitting tends to occur when our function class is too **small**, relative to the amount of data we have.

Concept Check...
If the Bayes model y^* is in $\mathcal{F}$, then the approximation error is zero. Can you state clearly why?

2.4 The Approximation/Estimation/Optimisation decomposition

I have one final point to make... in the decomposition above we assumed that we can find the empirical risk minimizer. In other words, we assumed we can find the global minimum of the empirical risk on the training data set. This is not realistic, in many scenarios. In most scenarios we can only have a sub-optimal model f. An additional risk component then emerges: $R(f) - R(f_{erm})$, referred to as the *optimisation error*. In special cases (e.g., linear models or very deep decision trees) this will be zero, but in general it is not. **Optimisation error** is the component of the excess risk caused by a poor learning algorithm. In this situation, the excess risk of f decomposes into a sum of *optimisation* error, estimation error, and approximation error:

$$\underbrace{R(f) - R(y^*)}_{\textbf{excess risk of } f} = \underbrace{R(f) - R(f_{erm})}_{\textbf{optimisation error}} + \underbrace{R(f_{erm}) - R(f^*)}_{\textbf{estimation error}} + \underbrace{R(f^*) - R(y^*)}_{\textbf{approximation error}}. \tag{17}$$

This decomposition highlights the 3 design elements that we have to consider to solve any supervised learning problem. These terms describe the learning process in abstract form: accounting respectively for the choice of **learning algorithm**, the quality/amount of **data**, and the choice of **model**. This triple, of *data/model/algorithm* gives you a template which all supervised learning procedures must address.

Poor source of data... your estimation error (and/or the risk of the Bayes model) will be high.
Poor choice of model family... your approximation error will be high.
Poor choice of learning algorithm... your optimisation error will be high.

However, these three components also **interact with each other**. It's not an easy problem.

[3]Google it, or ask chatGPT for a plain english summary.

2.5 OPTIONAL READING: Proving the form of the Bayes model

We defined the Bayes model as the population risk minimizer over Ω. It turns out that we can obtain a closed-form expression for this, for particular loss functions.

Squared loss: For a model, f, the population risk with squared loss is,

$$R(f) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\Big[(f(\boldsymbol{x}) - y)^2\Big]. \tag{18}$$

Theorem 2.1. *The population risk minimizer (also known as the Bayes prediction) for the squared loss is the expected label, given an input, i.e.* $y^*(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]$.

Proof. The population risk is:

$$R(f) \coloneqq \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\Big[(f(\boldsymbol{x}) - y)^2\Big] \tag{19}$$

At each point $\boldsymbol{x}$, we will minimise the expected loss, i.e. $y^* \coloneqq \operatorname{arginf} \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[(f(\boldsymbol{x}) - y)^2]$.

Now expand the square:

$$\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\Big[(f(\boldsymbol{x}) - y)^2\Big] = \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\Big[\,(\,f(\boldsymbol{x})^2 + y^2 - 2yf(\boldsymbol{x})\,)\Big] \tag{20}$$

To find the minimum, we differentiate[4] with respect to f, and set to zero:

$$\frac{\partial}{\partial f(\boldsymbol{x})}\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\Big[\,(\,f(\boldsymbol{x})^2 + y^2 - 2yf(\boldsymbol{x})\,)\Big] = \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\Big[\,(\,2f(\boldsymbol{x}) - 2y\,)\Big] = 0 \tag{21}$$

Now...

$$\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\Big[\,(\,2f(\boldsymbol{x}) - 2y\,)\Big] = (\,2f(\boldsymbol{x}) - 2\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\,) = 0. \tag{22}$$

Solving for f, we see for any given input $\boldsymbol{x}$, the minimizer is $f(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]$, which proves the theorem. □

The 0/1 classification loss: The squared loss is appropriate for regression problems, i.e. those where we are estimating a real-valued quantity. If we are instead performing a classification task, e.g. predicting the class label of an image, then a more appropriate option here is the 0/1 loss.

$$\ell_{0/1}(y, f(\boldsymbol{x})) = \begin{cases} 1 & \text{if } y \neq f(\boldsymbol{x})); \\ 0 & \textit{otherwise.} \end{cases} \tag{23}$$

i.e. there is a loss of 1 if we get it wrong, otherwise no loss at all. We define the population risk,

$$R(f) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\Big[\ell_{0/1}(y, f(\boldsymbol{x}))\Big] = p(f(\boldsymbol{x}) \neq y) \tag{24}$$

which is just the probability of the prediction not equalling the true label. Similarly, the empirical risk is the is mis-classification rate on a sample of data,

$$\hat{R}(f, \mathcal{S}_n) = \frac{1}{n}\sum_{i=1}^{n}\Big[\ell_{0/1}(y_i, f(\boldsymbol{x}_i))\Big]. \tag{25}$$

Though a proof of this is beyond the scope of this module, the Bayes model for the 0-1 loss is,

$$y^* \coloneqq \operatorname*{argmin}_{f\in\Omega} R\,(f) = \operatorname*{argmax}_{y} p(y \mid \boldsymbol{x}) \tag{26}$$

i.e. the class label y that has the highest (true) probability according to the distribution $p(y|\boldsymbol{x})$.

See a slightly more detailed / different presentation of all this at :
https://stephens999.github.io/fiveMinuteStats/decision_theory_bayes_rule.html

[4] For those worried about differentiating under the integral, it's fine. The Leibniz integral rule applies in almost every problem we might consider, as our spaces $\mathcal{X}, \mathcal{Y}$ can be bounded by constants less than infinity. If you don't know what I'm on about, don't worry, it's beyond the scope of this module.

2.6 Summary

The framework of statistical learning theory gives us a mathematical language with which we can reason about learning problems. The approximation/estimation/optimisation decomposition allows us to discuss the capacity of a model, separately from the data we provide to it, and from the learning algorithm.

A problem with the decomposition is that we cannot *estimate* these quantities on real data. If we could, the quantities would give us some insight into *why* any given algorithm succeeds or fails—in effect they would act as *diagnostics* for what we should be doing to solve the problem.

Concept Check...
Can you state clearly why its impossible to estimate the terms on real data?

There is however, another decomposition, where we can estimate terms. The *bias-variance* decomposition will be the topic of section 4.

3 Generalisation Bounds

Motivation. If I deploy ML models in safety-critical or sensitive domains (e.g., for public/governmental services) then I want to be able to give *guarantees* that the model will perform as I say it will. For example, I want to state *with confidence* that it will maintain a certain level of accuracy. A generalisation bound is a mathematical inequality that enables this. It tells you what the future performance is likely to be, in terms of the training error, and the *complexity/size* of the model.

Learning Objectives

By the end of this week you should be able to define and use the following concepts correctly in a conversation.

- generalisation bounds
- Hoeffding's inequality
- finite vs infinite function classes

3.1 Lots of maths today. But what will we end up with?

There is a lot of maths today. More than usual. In the end, we will end up with a statement of the form:

$$\textbf{Population risk} \leq \textbf{Empirical risk} + \epsilon \tag{27}$$

where $\epsilon \geq 0$ is a term that depends on (1) the size of the function class $\mathcal{F}$ that you use, (2) the number of data points, n, that you have, (3) a confidence parameter δ. This is called a *generalisation bound*. For example, imagine we have a function class of size $|\mathcal{F}| = 10,000$. We want to make a statement about models learned from this family, with 99% confidence, so $\delta = 0.01$. We will evaluate the models on a testing set of $n = 2000$ datapoints.

Given these three parameters ($|\mathcal{F}| = 10,000$, $n = 2000$, $\delta = 0.01$), our calculation today will tell us that $\epsilon \approx 6\%$. So, if we train a model, and see an empirical risk of 10%, the *true* (population) risk will be at most 16%.

Using this sort of technique, we could provide *guarantees* on the future performance of a model. This might be useful in *safety-critical* or *sensitive* areas: e.g., using ML in governmental/public services, we might quantify the potential dangers of deploying a model.

The particular type of bound we will see today is very simple, and just one of many—the research area is very active. You will see another type of bound in the final weeks of this module.

3.2 How good is our estimate of the generalisation error?

We want a model that minimizes population risk, as far as possible. We know (from last week) that there we can't **measure** the population risk, but instead only **estimate** it, in the form of a data sample of size n, known as the *empirical* risk. The *law of large numbers* is something you've probably heard of. Informally, this states that an estimate of some number will approach the true value, as the sample size increases. But *how fast* does it approach? How many samples do we need to get within a distance ϵ of the true value?

The answer is given by *Hoeffding's inequality*. The full version of the inequality applies for any bounded range $[a, b]$. Today we will illustrate it for the case of $[0, 1]$. Any other finite range can be scaled to this, and the same principle applies. With this, we have the following inequality.

Definition 8 (Hoeffding's inequality, simplified version). *Assume $z_1, z_2, ..., z_n$ are independent instantiations of a random variable Z, such that $z_i \in [0, 1]$. Hoeffding's inequality states:*

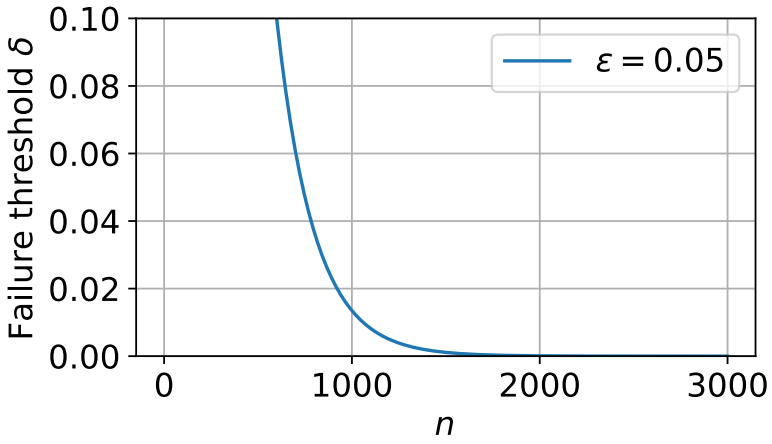
$$p\left(\left| \mathbb{E}[Z] - \frac{1}{n} \sum_{i=1}^{n} z_i \right| \geq \epsilon \right) \leq \delta = 2\exp(-2n\epsilon^2) \tag{28}$$

Proof of this is outside the scope of the module. In plain English, this is saying:

> *Given a true value $\mathbb{E}[Z]$, and an empirical estimate $\frac{1}{n}\sum_i z_i$, the probability of the estimate deviating by more than ϵ from the true value is less than or equal to δ.*

That's quite a sentence, so I'll say it another way. We are estimating some quantity $\mathbb{E}[Z]$, and we want our estimate to be within some tolerance ϵ of this. The chances of being *outside* this tolerance (i.e., having a really bad estimate) is the probability δ. We will refer to δ our *failure threshold*, or *failure probability,* in that we tried to estimate $\mathbb{E}[Z]$ and it 'failed', i.e., was outside our tolerance zone.

Let's plot the function, $\delta = 2\exp(-2n\epsilon^2)$, to see what it looks like and get some more intuition. Here we fix our tolerance at $\epsilon = 0.05$, and vary $n \in [1, 3000]$ samples.



With $n \approx 1000$, we have $\delta \approx 0.0135$. This is the probability of our estimate being more than ϵ away from the true value. If we had a larger sample, $n = 2000$, this reduces drastically, to ≈ 0.00009.

Concept Check...
Calculate δ for $n = 2000$, $\epsilon = 0.05$, exactly. You should end up with $\delta \approx 0.00009$.
If I increase ϵ, i.e. increase my tolerance, what will happen to δ?

So how do we use this in our ML problem? Well, if we use the 0/1 loss, i.e. a classification problem, our empirical/population risks will be values in the range $[0, 1]$, so we can apply the inequality above.

In the inequality above, we will substitute $Z = \ell(y, f(\boldsymbol{x}))$, i.e. the Z is the loss at a random point $(\boldsymbol{x}, y)$.

The **population** risk is then the 'true' value ... $\mathbb{E}_{(\boldsymbol{x},y)}[Z] = \mathbb{E}_{(\boldsymbol{x},y)}[\ell(y, f(\boldsymbol{x}))] = R(f)$

The **empirical** risk is then our 'estimate' ..... $\frac{1}{n}\sum_{i=1}^{n} z_i = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f(\boldsymbol{x}_i)) = \hat{R}(f, \mathcal{S}_n)$

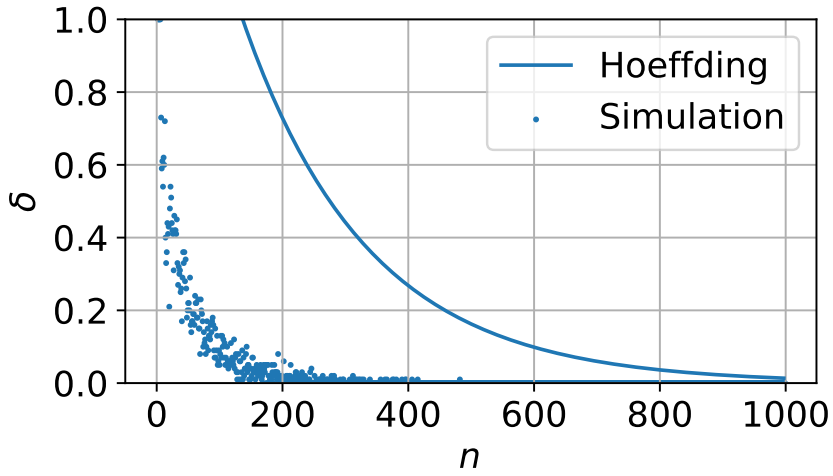
Writing this out ...

$$p\left(\left|R(f) - \hat{R}(f, \mathcal{S}_n)\right| \geq \epsilon\right) \leq \delta = 2\exp(-2n\epsilon^2) \tag{29}$$

This says that, if we have an empirical risk for a **particular** (pre-trained) model, f, on a data sample of size n, the chances of your estimate being more than ϵ away from the true value is bounded by δ.

An important thing to note here, is that **the bound is often extremely *loose***. In the scenario above, this has the consequence that the $n = 2000$ is way more samples than we actually need. We can simulate a hypothetical situation to see this.

Imagine we have a population risk $R(f) = 0.1$. I generate a sequence of n binary values, with $p(z = 1) = 0.1$. To get my estimate, I average these values. I then check whether the estimate is within the tolerance ϵ of the true value, 0.1. I repeat this process over 100 trials, and see how many of the trials were outside the tolerance zone—this is my failure rate, which I can compare to the Hoeffding bound, plotted below.

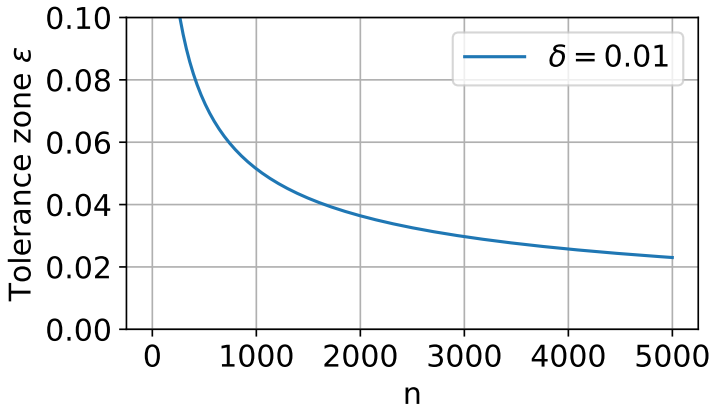


In practice, the probability of being outside $\epsilon = 0.05$ decays to practically zero by about $n \approx 400$, so in this case it's about a 5× overestimate.

We can also rearrange Hoeffding's inequality to give different forms. Solving the expression for ϵ:

$$\epsilon = \sqrt{\frac{\ln(2/\delta)}{2n}}. \tag{30}$$

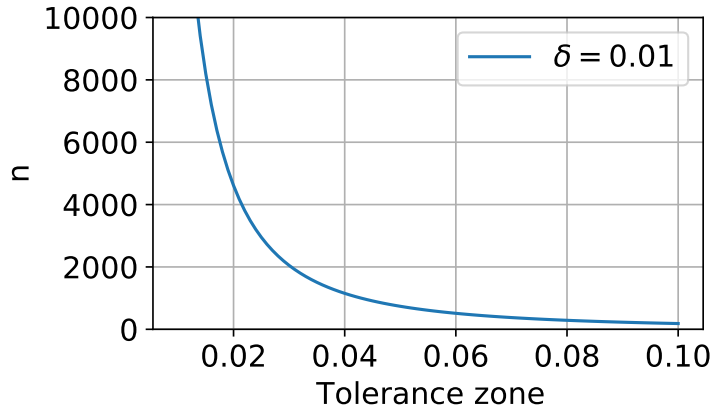
We can now fix our $\delta = 0.01$—which says we want to be 99% confident—then vary n and ask what tolerance ϵ we will have, i.e. how close we will be to the true value.



With 3000 samples, we can be 99% confident we'll be within 0.03 of the true value. With 5000, this gets even closer. Take note... as you saw with the simulation, this bound is loose too.

Concept Check...
Adapt my simulation code to show how close we are in reality, with different numbers of samples.

Or, we can flip this round to see another interpretation. Let's say we're happy with being 95% confident ($\delta = 0.05$), and solving for n (I'll leave that up to you) we get the following figure.



Again, there are things we can read into this—notably we see how many samples we need to be within certain distances of the true value. If we want to be within $\epsilon = 0.02$, we'll need at least $n = 4000$.

3.3 A Generalisation Bound for Finite Function Classes

We're about to see how all this relates to our main question—are bigger models always better?

Hoeffding's inequality as we applied it in Equation (29) is talking about a single (hypothetical) pre-trained model f, i.e. we have already chosen the parameters for the model, and we are just evaluating its risk on our data set. But, when training a model we start with a space[5] of *possible* models, and then use our training data to pick one. As last week, the space of possible models we will refer to as a **function class** and denote it by $\mathcal{F}$.

Since we could in theory end up picking any of the models from $\mathcal{F}$, we'd like to have some guarantee that they will all yield good estimates of the risk, i.e. not violate the tolerance zone we discussed above.

We can do this by considering the probability that **at least one** of the models has a risk estimate outside the tolerance zone. To understand how we are going to compute this, imagine we flip two coins, independently. Define two boolean events, V_1 and V_2. The event V_1 is **true** if coin 1 comes up as heads, and similarly V_2 is **true** if coin 2 comes up as heads. The probability rule for the **at least one** scenario is:

$$P(V_1 \text{ or } V_2) \;=\; P(V_1) + P(V_2) - P(V_1 \text{ and } V_2). \tag{31}$$

Here, the probability of V_1 or V_2 being true, is equal to the sum of the two independent probabilities, minus the probability of both occurring at the same time.

Since $P(V_1 \text{ and } V_2) \geq 0$, we know that $P(V_1 \text{ or } V_2) \leq P(V_1) + P(V_2)$. This inequality generalises to M events, where the logical OR is denoted by the 'union' operator, and is known as Boole's inequality.

Definition 9 (Union bound, also known as Boole's Inequality)**.**

$$P\left(\bigcup_{i=1}^{M} V_i\right) \leq \sum_{i=1}^{M} P\left(V_i\right) \tag{32}$$

Applying this inequality, we find that...

$$P\left(\exists f \in \mathcal{F},\ \left|R(f) - \hat{R}(f, \mathcal{S}_n)\right| \geq \epsilon\right) \;\leq\; \sum_{i=1}^{M} P\left(\left|R(f_i) - \hat{R}(f_i, \mathcal{S}_n)\right| \geq \epsilon\right) \tag{33}$$

where f_i is the ith model in the function class.

Notice that the sum of probabilities on the right is in the form we had from Hoeffding's inequality for one model, $\delta = 2\exp(-2n\epsilon^2)$, and there are simply M of these terms, repeated. We can now state our inequality over the whole space, known as the **uniform deviation bound.**

Definition 10 (Uniform Deviation Bound)**.** *Assume we are picking a model f from an overall model family $\mathcal{F}$, of size $|\mathcal{F}|$. The following bound holds.*

$$p\Big(\exists f \in \mathcal{F},\ \left|R(f) - \hat{R}(f, \mathcal{S}_n)\right| \;\geq\; \epsilon\Big) \leq \delta = 2|\mathcal{F}|\exp(-2n\epsilon^2) \tag{34}$$

Notice the key change, that this bound is now a function of $|\mathcal{F}|$, the size of the function class.

This says: the probability that there exists a model which violates our tolerance zone is less than or equal to δ. Equivalently, we can say that with probability $1 - \delta$, all of our estimates are within the zone, i.e. $\forall f$, we have $|R(f) - \hat{R}(f, \mathcal{S}_n)| < \epsilon$.

[5] It's important to note that we are assuming this space, $\mathcal{F}$, is *finite*. An example of a finite space is the set of all decision trees of fixed depth. However, we are often picking from an **infinite** space—e.g. the space of models defined by d real-valued weights in a neural network. We'll only deal with the finite case for now.

Finally (I'm sure you're glad we are finally there) we can state our **generalisation bound.** First, we take the upper tail of the distributiopn. This is the expression $\delta = |\mathcal{F}|\exp(-2n\epsilon^2)$. Notice we removed the 2 from the right hand side of the bound as we used before. This is because the bound is two a two-tailed scenario. We are only interested in checking our risk is bounded from one direcgtion. We now solve for ϵ.

$$\epsilon = \sqrt{\frac{\ln(|\mathcal{F}|) + \ln(1/\delta)}{2n}} \tag{35}$$

We plug this into $R(f) - \hat{R}(f, \mathcal{S}_n) < \epsilon$ and rearrange:

Definition 11 (Generalisation bound for finite function classes)**.**

$$R(f) \leq \hat{R}(f, \mathcal{S}_n) + \sqrt{\frac{\ln(|\mathcal{F}|) + \ln(1/\delta)}{2n}} \tag{36}$$

This reads as *"the true error is less than the training error plus a term, dependent on the number of training samples and the function capacity"*. If $\delta = 0.05$, it further translates that we can be 95% confident in this statement.

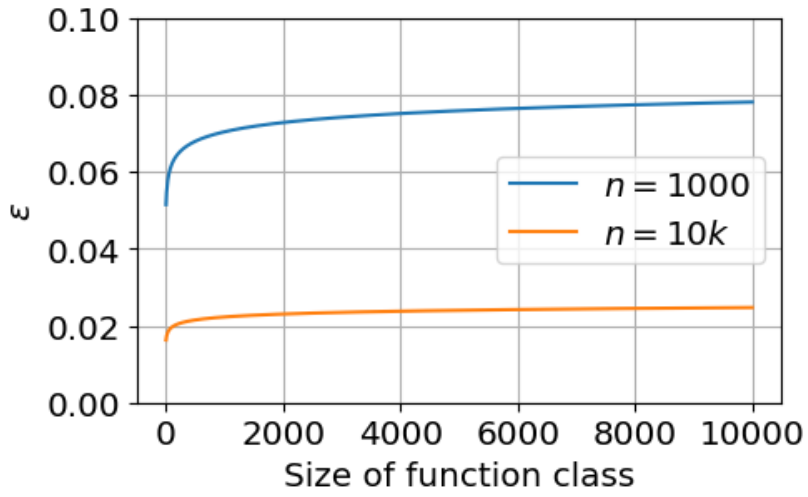


Figure 7: $\delta = 0.05$, i.e. 95% confidence.

3.4 Summary

Today we derived a **generalisation bound**. This is a milestone in this module. We explicitly related the size of the dataset (n), and the capacity of the model (in the form of the function class $\mathcal{F}$) to the generalisation error. It showed us that, if we increase the size of the model, our confidence in the error estimate drops, unless we also increase the size of the data alongside it.

Whilst this is all very interesting, it is somewhat esoteric. Next week we'll see a little more of a practical element, in the *bias-variance* decomposition. This is kind of like the approximation-estimation decomposition, but where we can additionally estimate the two terms, and plot them for real models.

4 The Bias-Variance decomposition

Motivation. In week 2 we met the approximation-estimation decomposition. This showed there were two *components* to the excess risk, with different interpretations. It turned out to be impossible to estimate the terms from data. But, there is another decomposition, with similar interpretations, and we can estimate the terms. Today we meet the *bias-variance* decomposition, and the bias-variance trade-off. If you fully understand this, it will fundamentally shift how you think about ML. Let's go.

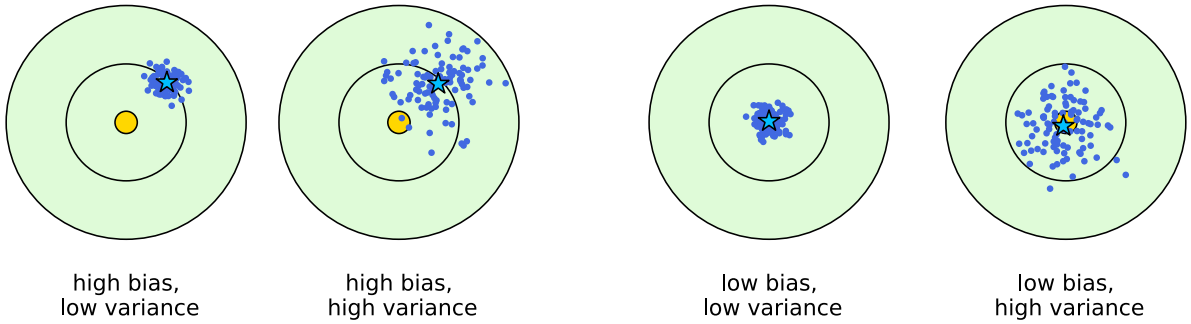
Learning Objectives

By the end of this week you should be able to define and use the following concepts correctly in a conversation.

- the expected risk of a model
- the bias-variance decomposition for squared and cross-entropy losses
- the bias-variance trade-off
- the trade-off in 'over-parameterized' models

4.1 The intuitive explanation

Imagine I want to train a model to throw darts at a dartboard. In the visualisation below, the bullseye (yellow circle) is the target for a *single test point*. I take a training set $\mathcal{S}_n$, train a model, and make a prediction. Each blue dot is a prediction from a model using a different randomly sampled *training* set.



(a) Predictions from two models with high bias.

(b) Predictions from two models with low bias.

First, look at the pair of dartboards on the left. Both models here have missed the target by quite a long way—so we call them *high bias* models. However, one is a tight cluster, and one is more spread out. If the cluster is tight, it means the predictions did not change much as a result of getting a different training set. This is called a *low variance* model, and is clearly quite *insensitive* to the training data, producing mostly the same prediction each time. The other one is very spread out, sensitive to which training data it got, so we call it a *high variance* model.

Now, look at the pair of dartboards on the right. Both these clusters are centred on the bullseye, so they are called *low bias* models. The tightly clustered one is not so sensitive to its training data, thus we call it a *low variance* model. The one that is more spread out is clearly quite sensitive to which training data set we provide, so is called *high variance*.

This is the very rough idea. The **variance of a model** is a measure of sensitivity to the training data. The **bias of a model** is how far the cluster of predictions are from the target, which roughly translates to a measure of strength in the predictor. These are only rough descriptions—for example, there is another component called the 'noise' which we cannot easily illustrate with the dartboard analogy. To fully understand the details, it requires us to go into the mathematics.

4.2 The mathematical explanation

Note: *Our model f is dependent on the training data $\mathcal{S}_n$ that we give to it. So, in the text below, strictly, we should ideally be using notation $f(\boldsymbol{x}; \mathcal{S}_n)$, instead of just $f(\boldsymbol{x})$. But for compactness we chose not to do so, so you'll have to remember this dependency yourself.*

Our training set $\mathcal{S}_n$ is just one from the space of *possible* training sets that we could have seen. Each data point $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^d \times \mathbb{R}^k$ is assumed to be an *independent* sample from an unknown distribution $\mathcal{D} = P(\boldsymbol{x}, \boldsymbol{y})$. The training data set $\mathcal{S}_n = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$ is therefore a sample from a joint random variable $P(\boldsymbol{x}, \boldsymbol{y})^n$, i.e. sampling n times independently. We can therefore consider the **average risk**, over *all possible training sets* we might have encountered. We call this *expected* risk—we'll first look at *squared loss*, and progress to others.

Definition 12 (Expected squared risk)**.** *The expected risk is the population risk averaged over all possible training data sets of a fixed size n:*

$$\mathbb{E}_{\mathcal{S}_n}\Big[R(f)\Big] = \mathbb{E}_{\mathcal{S}_n}\Big[\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[(f(\boldsymbol{x}) - y)^2\right]\Big] \tag{37}$$

There are two expectations in this expression. The outer one, $\mathbb{E}_{\mathcal{S}_n}$, is averaging over possible *training sets*. The inner one, $\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}$, is averaging over possible *testing points*. **Take a minute to reflect on this**.

The bias-variance decomposition (Geman et al., 1992) breaks this down into three components.

Definition 13 (Bias-Variance decomposition for squared risk)**.** *Given a model f, and a random variable over training data sets, the expected squared risk decomposes into three components.*

$$\underbrace{\mathbb{E}_{\mathcal{S}_n}\Big[R(f)\Big]}_{\text{expected squared risk}} = \mathbb{E}_{\boldsymbol{x}}\Bigg[\underbrace{\mathbb{E}_{y|\boldsymbol{x}}\Big[\big(y - \mathbb{E}_{y|\boldsymbol{x}}[y]\big)^2\Big]}_{\text{noise}} + \underbrace{\Big(\mathbb{E}_{\mathcal{S}_n}\left[f(\boldsymbol{x})\right] - \mathbb{E}_{y|\boldsymbol{x}}[y]\Big)^2}_{\text{bias}} + \underbrace{\mathbb{E}_{\mathcal{S}_n}\left[\Big(f(\boldsymbol{x}) - \mathbb{E}_{\mathcal{S}_n}\left[f(\boldsymbol{x})\right]\Big)^2\right]}_{\text{variance}}\Bigg]. \tag{38}$$

where $\mathbb{E}_{\mathcal{S}_n}\left[f(\boldsymbol{x})\right] = \int \Big[f(\boldsymbol{x})\Big] p(\mathcal{S}_n)\, \mathrm{d}\mathcal{S}_n$ is the "expected model".

Note: The formal proof of this is optional reading, in subsection 4.5.

The first term is called the *noise*. This is an irreducible constant, independent of any model parameters. It is caused by choice of data/features, and not by the model. The only way to reduce it is to get better quality labelled data, with more informative features. Adding more examples will not reduce this. You might notice that it is equal[6] to the *Bayes Risk*, i.e. $R(y^*)$. We saw this back in Definition 6 of section 2. This reflects that there is a relationship between this bias-variance decomposition and the approximation-estimation decomposition. They are *not* equivalent, but are strongly related.

The second term is the *bias*. This is the loss of the *expected model* against $\mathbb{E}_{y|\boldsymbol{x}}[y]$. The *expected model* is the average response we would get if we could average over all possible training data sets. The main way to reduce this is to increase the flexibility of our model. In the terminology from section 2, this means to increase the size of our model family. We could potentially reduce this by adding more features.

The third term is the *variance*. This captures variation in f due to different training sets, varying around the expected model. If the model is *too* flexible, this will grow large. We could potentially decrease this by increasing the number of training examples, or adding some regularization to the model. Another way to reduce it is the *Bagging* algorithm—this will be covered in section 5.

A bias-variance decomposition holds for several losses, not just the squared loss. For example, it holds for the very commonly used *cross-entropy*, which we reviewed all the way back in subsection 1.1.

[6] Well, as it's labelled in the equation above, not exactly. Can you see the difference?

Bias/Variance for Cross-entropy

Cross-entropy is (by far) the most commonly used loss to train neural networks. The bias-variance decomposition for this case is written in terms of the *Kullback-Leibler* divergence, or KL-divergence, which is related to the cross-entropy in the following manner. Let's assume[7] we have a true class distribution, $\boldsymbol{y} = [0.1, 0.1, 0.8]$. This means for a given input $\boldsymbol{x}$, the most likely outcome is that the true class is 3, but there is a 10% chance that it could be class 1 or 2. The KL divergence is :

$$K(\boldsymbol{y} \,||\, f(\boldsymbol{x})) := \sum_{c=1}^{3} \boldsymbol{y}_c \ln \frac{\boldsymbol{y}_c}{f_c(\boldsymbol{x})} \tag{39}$$

where $f_c(\boldsymbol{x})$ denotes the probability assigned to the cth class by our model.

Concept Check...
Assume $\boldsymbol{y} = [0.1, 0.1, 0.8]$, and $f(\boldsymbol{x}) = [0.2, 0.2, 0.6]$. Compute the KL divergence $K(\boldsymbol{y} \,||\, f(\boldsymbol{x}))$.

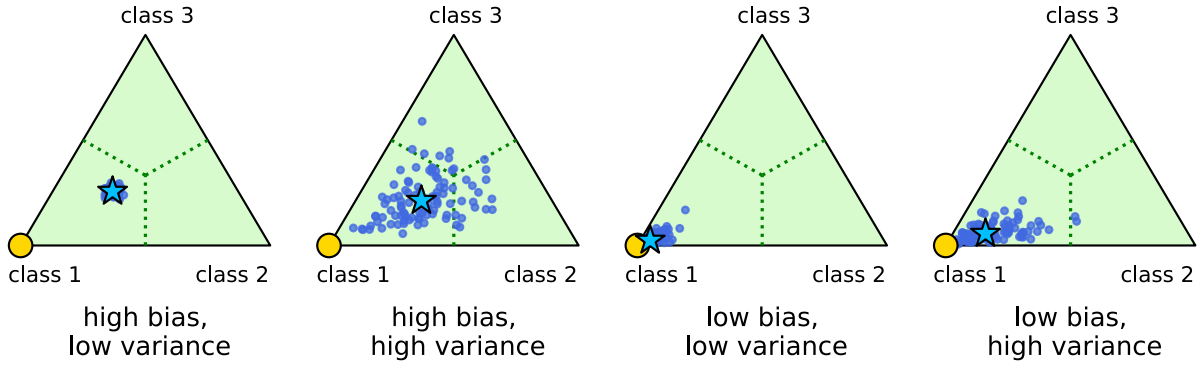
Now, assume $\boldsymbol{y}$ is a one-hot vector, i.e. the label for a *single* $\boldsymbol{x}$. The KL is now equal to:

$$K(\boldsymbol{y} \,||\, f(\boldsymbol{x})) \;=\; \sum_{c=1}^{3} \boldsymbol{y}_c \ln \frac{\boldsymbol{y}_c}{f_c(\boldsymbol{x})} \;=\; \underbrace{-\sum_{c=1}^{3} \boldsymbol{y}_c \ln f_c(\boldsymbol{x})}_{cross\text{-}entropy}, \tag{40}$$

where use the convention that $0 \ln 0 = 0$. If we use notation $\ell(\boldsymbol{y}, f(\boldsymbol{x}))$ for the cross-entropy, then the following decomposition holds.

$$\underbrace{\mathbb{E}_{\mathcal{S}_n}\Big[\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\Big[\ell(\boldsymbol{y}, f(\boldsymbol{x}))\Big]\Big]}_{\text{expected cross-entropy risk}} \;=\; \mathbb{E}_{\boldsymbol{x}}\Big[\underbrace{\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[K(\boldsymbol{y} \,||\, \boldsymbol{y}^*)\right]}_{noise} + \underbrace{K(\boldsymbol{y}^* \,||\, \mathring{f}(\boldsymbol{x}))}_{bias} + \underbrace{\mathbb{E}_{\mathcal{S}_n}\left[K(\mathring{f}(\boldsymbol{x}) \,||\, f(\boldsymbol{x}))\right]}_{variance}\Big], \tag{41}$$

where, $\boldsymbol{y}^* = \mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[\boldsymbol{y}]$, and $\mathring{f} := Z^{-1} \exp(\mathbb{E}_{\mathcal{S}_n}[\ln f])$ is the *normalized geometric mean* of the model distribution. So again, we have **noise**, **bias**, and **variance** terms. These are expressed in KL-divergences, but they have the same interpretation as in the squared loss case. Note that the *geometric* mean has taken the place of the *arithmetic* mean $\mathbb{E}_{\mathcal{S}_n}[f]$ from the squared loss decomposition, so we don't have the 'expected model' any more, and instead refer to it as the 'centroid' model. **Proof of this is well outside the scope of this module, but if you're interested I can point you at relevant literature.** Incidentally, the 'dartboard' analogy can also be made, below.



[7] This does occur in practice, e.g. the problem could be to diagnose a medical condition—and different doctors will have different opinions, so the true outcome is actually subjective, or 'noisy'.

4.3 The bias-variance trade-off

The bias/variance terms can be estimated from data—the Python code for this is provided on Blackboard. We will build regression trees of increasing depth, and predict a noisy sine wave, as we did in one of the previous sessions. On the x-axis we increase the depth of a decision tree. The faint red lines are 50 repeats of the model being trained from scratch with a newly sampled training data set. The bold red line is the average of these, i.e. an estimate of the expected squared risk of the model f.

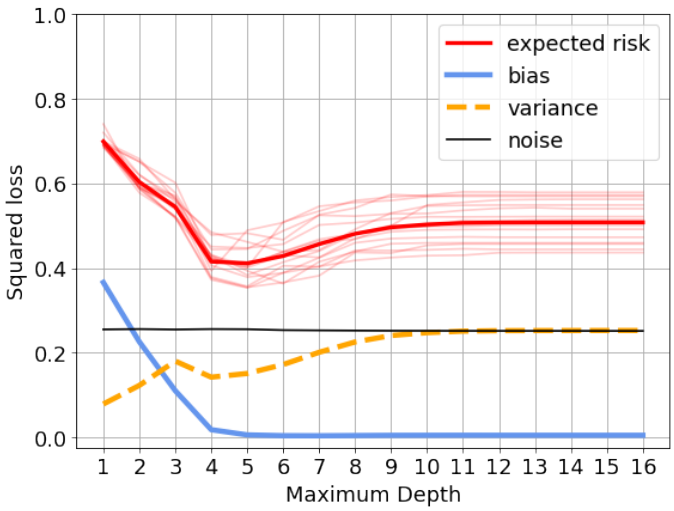


Figure 9: Building regression trees of increasing depth. **Code for this is on Blackboard.**

We see that, as the tree depth increases, the bias tends to *decrease*, and the variance tends to *increase*: this is commonly referred to as the *bias-variance trade-off*. The depth here is our proxy for how *powerful* the model is, or how much *capacity* it has to learn. If it is too 'powerful' it tends to pick up on the noise present in the training data, and hence over-fit.

Whilst we showed this for decision trees, the same principle applies for *all models*—though for some, the trade-off is not so straightforward as shown above.

Linear regression. In linear regression, high bias is associated with the assumption that the relationship between input features and output is linear. If the true relationship is more complex, linear regression may exhibit high bias, leading to underfitting. Variance is generally low in linear regression. You can control the trade-off with L2 weight regularization.

Decision trees. Tree can have low bias, as they can represent complex relationships in the data. However, decision trees are prone to high variance, especially when they grow deep and capture noise in the training data. Techniques like pruning can be applied to control variance and avoid over-fitting.

k-Nearest Neighbors. A k-NN can have low bias, especially in complex, non-linear datasets. However, it may suffer from high variance, particularly in the presence of noisy data or irrelevant features. Choosing an appropriate value of k manages the bias/variance trade-off here.

Neural Networks. Neural networks, especially deep ones, can model highly complex relationships and have low bias. However, they are prone to high variance, especially if the network is too large or trained for too long. Regularization techniques, such as dropout, and early stopping are used to manage variance. Variance can also be kept low through *implicit regularization*—details will be covered in later weeks of this module.

By thinking about the bias-variance decomposition with various models, we gain a nuanced understanding of how different algorithms balance between capturing underlying patterns (low bias) and avoiding noise (low variance). This knowledge helps us make informed decisions when selecting and fine-tuning models.

4.4 Double Descent, and the trade-off in "over-parameterized" models

Now we get to the core question of our module. What happens to bias and variance when you have a HUGE model? Let's quantify this. Imagine we had p parameters, and n training data points.

Definition 14 (Over-parameterization ratio)**.** *With p parameters to learn, and n training points, the over-parameterization ratio is $\rho = p/n$. A model is said to be over-parameterized if $\rho > 1$, i.e. $p > n$.*

With really huge neural nets, we almost always have $\rho \gg 1$. So in these scenarios, when we increase complexity... will the bias go down, as it did in the simple decision tree scenario? Will the variance always go up? The answers are a bit more complex. It turns out that deep learning models display a non-monotonic behaviour in the variance: it first goes up, then down again. See Figure 10.

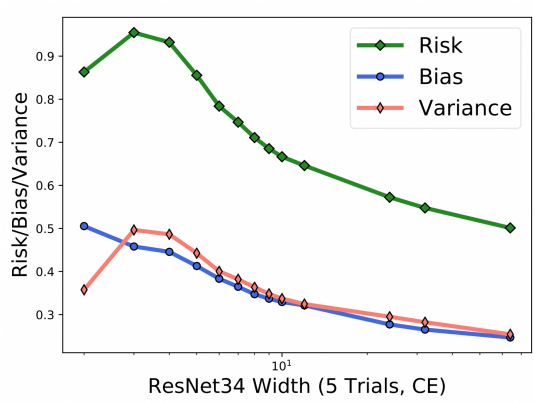
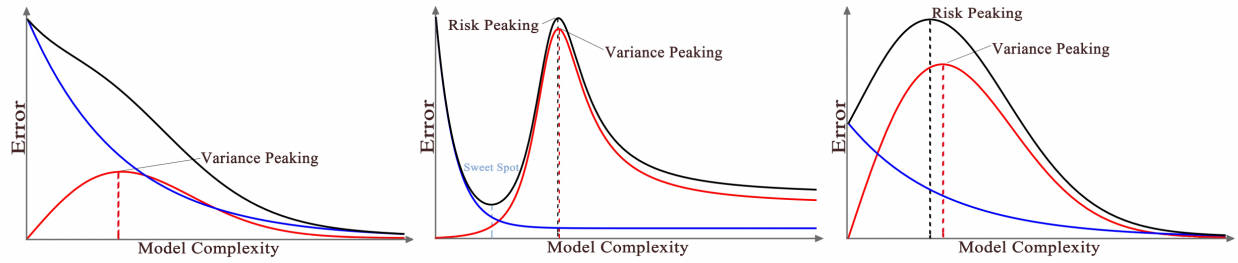


Figure 10: Typical bias-variance tradeoff in a deep neural network as we increase the width of the network, i.e. the number of neurons in a single layer. Figure borrowed from Yang et al. (2020).

There are 3 scenarios which commonly occur with neural nets. These are visualised below.



(a) Case 1 (b) Case 2 (c) Case 3

Figure 11: Typical scenario for expected risk (black) in a deep network: bias (blue) and variance (red). Figure borrowed from Yang et al 2020.

The left scenario is very common. The risk (black line) seems to decrease slowly, but then accelerates somehow. The bias was going down, but the variance was going up a little bit. This is even more pronounced in the far-right scenario, where the risk actually rises a lot, then drops.

The middle scenario is an important and recent issue called 'double descent': the risk initially decreases, then increases, then decreases again. **Exactly why this happens is an open research question.** Current thinking is that the networks are *implicitly regularized* by the stochastic gradient descent algorithm, but the overall issue is far from resolved. The details of this will be covered in section 9.

You can also read more in Yang et al. (2020), if you want to.

4.5 OPTIONAL READING: Proof of Bias/Variance decomposition for Squared Loss

Extra Reading (Non-Assessed)
The proof below is not part of the assessed material, so you will not be asked to reproduce it in an exam. But you may find it interesting, so here we go....

Our model $f(\boldsymbol{x})$, is dependent on a training sample $\mathcal{S}_n$. The decomposition is:

$$\underbrace{\mathbb{E}_{\mathcal{S}_n}\left[\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[(f(\boldsymbol{x})-y)^2\right]\right]}_{expected\ risk} =$$

$$\mathbb{E}_{\boldsymbol{x}}\left[\underbrace{\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[(y-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y])^2\right]}_{noise} + \underbrace{\left(\mathbb{E}_{\mathcal{S}_n}[f(\boldsymbol{x})]-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)^2}_{bias} + \underbrace{\mathbb{E}_{\mathcal{S}_n}\left[(f(\boldsymbol{x})-\mathbb{E}_{\mathcal{S}_n}[f(\boldsymbol{x})])^2\right]}_{variance}\right] \tag{42}$$

We will now prove this. We note that $\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}[\cdots]$ can be factorized into $\mathbb{E}_{\boldsymbol{x}}\left[\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[\cdots]\right]$, which is true by definition of the expectation. We take the left hand side, for the moment ignoring the expectation over $\mathcal{S}_n$.

$$\begin{aligned}
&\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[(f(\boldsymbol{x})-y)^2\right]\\
&\quad=\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[(f(\boldsymbol{x})-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]+\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]-y)^2\right]\\
&\quad=\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\left(f(\boldsymbol{x})-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)^2+\left(\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]-y\right)^2+2\left(f(\boldsymbol{x})-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)\left(\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]-y\right)\right]\\
&\quad=\mathbb{E}_{\boldsymbol{x}}\left[\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\left(f(\boldsymbol{x})-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)^2+\left(\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]-y\right)^2+2\left(f(\boldsymbol{x})-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)\left(\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]-y\right)\right]\right]\\
&\quad=\mathbb{E}_{\boldsymbol{x}}\left[\left(f(\boldsymbol{x})-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)^2\right]+\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\left(\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]-y\right)^2\right]+\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[2\left(f(\boldsymbol{x})-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)\left(\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]-y\right)\right]
\end{aligned} \tag{43}$$

We claim that the final term on the right is equal to zero.

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[2\left(f(\boldsymbol{x})-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)\left(\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]-y\right)\right] &= 2\left(f(\boldsymbol{x})-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\left(\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]-y\right)\right]\\
&= 2\left(f(\boldsymbol{x})-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)\left(\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)\\
&= 0.
\end{aligned} \tag{44}$$

The first step here holds because the term $2\left(f(\boldsymbol{x})-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)$ is independent of the expectation $\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[\cdots]$. Now, we reapply the expectation over $\mathcal{S}_n$, and have:

$$\mathbb{E}_{\mathcal{S}_n}\left[\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[(f(\boldsymbol{x})-y)^2\right]\right]=\mathbb{E}_{\mathcal{S}_n}\left[\mathbb{E}_{\boldsymbol{x}}\left[\left(f(\boldsymbol{x})-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)^2\right]\right]+\mathbb{E}_{\boldsymbol{x}}\left[\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}\left[\left(\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]-y\right)^2\right]\right] \tag{45}$$

Notice that the expectation over $\mathcal{S}_n$ does not apply to the final term here, as it is independent of the model—this is the *noise* term. Now take the first term on the right, and develop it as follows.

$$\begin{aligned}
&\mathbb{E}_{\mathcal{S}_n}\left[\mathbb{E}_{\boldsymbol{x}}\left[\left(f(\boldsymbol{x})-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)^2\right]\right]\\
&=\mathbb{E}_{\boldsymbol{x}}\left[\mathbb{E}_{\mathcal{S}_n}\left[\left(f(\boldsymbol{x})-\mathbb{E}_{\mathcal{S}_n}[f(\boldsymbol{x})]+\mathbb{E}_{\mathcal{S}_n}[f(\boldsymbol{x})]-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)^2\right]\right]\\
&=\mathbb{E}_{\boldsymbol{x}}\left[\mathbb{E}_{\mathcal{S}_n}\left[(f(\boldsymbol{x})-\mathbb{E}_{\mathcal{S}_n}[f(\boldsymbol{x})])^2+\left(\mathbb{E}_{\mathcal{S}_n}[f(\boldsymbol{x})]-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)^2+2\left(f(\boldsymbol{x})-\mathbb{E}_{\mathcal{S}_n}[f(\boldsymbol{x})]\right)\left(\mathbb{E}_{\mathcal{S}_n}[f(\boldsymbol{x})]-\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[y]\right)\right]\right]
\end{aligned}$$

Again we claim that the final term here is equal to zero. The proof is identical to the earlier claim, but using $\mathbb{E}_{\mathcal{S}_n}[\cdots]$ in place of $\mathbb{E}_{\boldsymbol{y}|\boldsymbol{x}}[\cdots]$. Combining all results, we have the decomposition as presented in (42).

4.6 Summary

We've examined the bias-variance decomposition for squared and cross-entropy loss. It shows that a model's expected behaviour (with respect to the random training data you provide it) can decompose into two meaningful components: the bias, and the variance. These allow us to diagnose what's going on inside a model. It also raises some non-intuitive behaviours with *over-parameterized models,* leading to the *double-descent phenomenon.*

It is VERY important to note that such decompositions do not hold for ALL losses. For example, the decomposition as described above does not hold for the 0/1 loss, also known as the classification error.

Next week we'll take a break from all the theory, and look at a practical element—*ensemble methods.* These are **committees of models** that can cooperate to solve problems.

Extra Reading (Non-Assessed)

In section 2, we met the approximation-estimation decomposition. This is **not the same as** the bias-variance decomposition, but there are obvious similarities in the interpretation and behaviour of the components. If you want to read more and find out the exact relation, there is a recent research paper on this issue (Brown & Ali, 2024).

5 Ensemble Methods: bigger, and better?

Motivation. This week we meet something slightly more practical, but still around our main question. If I take a given model, and *duplicate it M* times, with some random perturbations to each, I then have a 'committee' or 'ensemble' of models, each slightly different from one another. The ensemble has more parameters, and thus seems to be *bigger* than a single model. But how does this type of *big* model behave in practice? Just duplicating models seems a bad idea—so how can we build good ensembles?

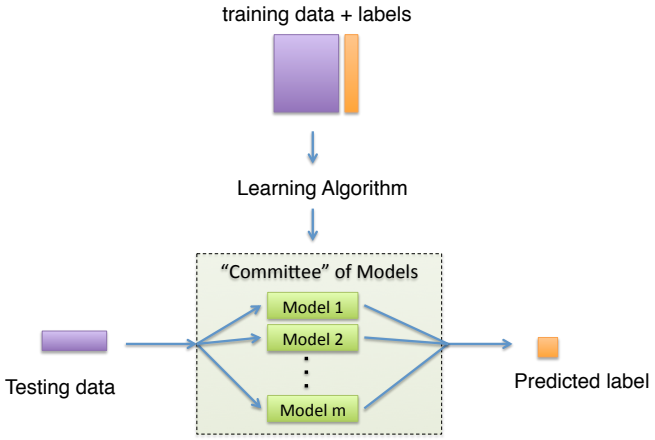
Learning Objectives

By the end of this week you should be able to define and use the following concepts correctly in a conversation about ML.

- the Bagging algorithm
- the Random Forests algorithm
- the Adaboost algorithm
- the ambiguity decomposition

5.1 Learning with Ensembles of Models

The word "*ensemble*" refers to the idea of having *multiple* predictive models, and *combining*[8] their predictions, treating them as a committee.



With ensembles, we learn multiple models, not just one. When we get a new test datapoint, we pass that point to the different models, and they all predict what the answer should be. Their predictions are merged in some way, so the whole group is used to make the final prediction. or predicting class labels, perhaps directly (e.g. predict $y = 1$), or perhaps via probabilities (e.g. predict $p(y = 1|\boldsymbol{x}) = 0.7$), or a ranking of the set of possible labels. The decisions can be combined by many methods, including voting, averaging, or various probabilistic methods.

The underlying principle of the field is a recognition that in real-world situations, every model has limitations and will make some errors. The "trick" that ensembles can exploit is that when a single model cannot properly fit the data, we can make multiple versions of that same model — each of which make errors in different ways — then, we can vote or average their predictions, cancelling out the errors of the individuals by the

[8] Interesting fact 1: it's actually French, meaning "together". Interesting fact 2: The Beatles wrote a song containing the line: "*C'est sont mots qui vont trés bien ensemble*" – which translates as 'these are words that go together well'.

committee decision. In most cases, the same ensemble algorithm ideas can be applied regardless of what type of model you are using — whether a simple decision tree, or a deep[9] convolutional neural network.

Time has shown, again and again, that while the field comes up with new ideas for building better predictive models, the constant that never goes away is to make *ensembles* out of of them. If we can answer why, when, and how particular ensemble methods can be applied successfully, we will have made progress toward a powerful new tool for Machine Learning: *the ability to automatically exploit the strengths and weaknesses of different learning systems.*

We will start with the idea of combining multiple predictions of a real-valued quantity, so a *regression* problem, and continue with the idea of combining *votes*, for a *classification* problem.

Combining Real-Valued Predictions

There is a reasonably well known incident from 1907 — Francis Galton, a notable scientist of the time, attended a county fair in Cornwall; it was common at the time to play a game where people would pay a penny to guess the weight of an ox, and the person with the closest guess won a prize. Galton recorded all the guesses, and he did publish a paper in Nature about his experiences. The purpose of the study was simply to highlight to farming communities the potential utility of recording data and analysing it.

Galton's paper reported the true weight at 1198lb, and various properties of the 787 guesses from local people on the day. The median guess of the crowd was 1207lb, and the *mean* was 1197lb. So, if a group people on the day had colluded together, and averaged their guesses, there is a good chance they would have been closer to the truth than any individual person — and hence won the prize.

Just one year before, in 1906, a result had been proven that would have added to Galton's paper quite nicely. We know it as *Jensen's inequality*. Imagine we had a set of M real valued numbers, $x_1, ..., x_M$. Jensen's inequality says that

$$\phi\Big(\sum_{i=1}^{M} \lambda_i x_i\Big) \leq \sum_{i=1}^{M} \lambda_i \phi\Big(x_i\Big) \tag{46}$$

where $\sum_i \lambda_i = 1$, and ϕ is any convex function. This seems quite abstract, until we find the analogy to the situation Galton was dealing with. Imagine the values x_i are the M guesses about the weight of the cow, from people on the day, and we have the true weight of the cow, y. Now, we choose $\phi(x) = (x-y)^2$. Further, we assume that $\lambda_i = \frac{1}{M}$ for all i, and use a shorthand $\bar{x} = \frac{1}{M}\sum_i x_i$. With this, Jensen's inequality gives us:

$$\Big(\bar{x} - y\Big)^2 \leq \frac{1}{M}\sum_i \Big(x_i - y\Big)^2 \tag{47}$$

This says that the distance of the mean guess $\bar{x}$ from the truth is less than (or equal to) the average of the distances for each individual guess x_i. The error of the combined guess of the group is *guaranteed* to be less than the average error of each individual. So how much better exactly? It's not too difficult to extend this and show:

$$\Big(\bar{x} - y\Big)^2 = \frac{1}{M}\sum_i \Big(x_i - y\Big)^2 - \frac{1}{M}\sum_i \Big(x_i - \bar{x}\Big)^2 \tag{48}$$

Try proving this for yourself — it's not difficult[10]. Summarising this in machine learning terms, for an ensemble of regression estimators, *the squared error of the ensemble is guaranteed to be less than or equal to*

[9] Yes, even Deep Learning uses ensemble methods. It's not magic.

[10] A hint: start with the right hand side of the inequality, and add/subtract $\bar{x}$ inside the square.

the average squared error of the individual estimators. In the ensemble academic research community, this is known as the *Ambiguity decomposition* (Krogh et al., 1995).

But, the results above are phrased for regression, and we are quite often interested in classification. Perhaps we should think about classifiers like decision trees, that output class labels – if we had an ensemble combined by a majority voting scheme, what guarantees do we have?

Combining votes

In 1785, a French aristocrat and noted mathematician and political scientist, Nicolas de Caritat contributed a work now known as the Condorcet Jury Theorem. The theorem assumes we have a group of voters, who make decisions independently of one another. It also assumes there is a single "correct" decision — obviously not always the case in politics, but let's assume there is one, for now. It also assumes each member makes the *wrong* decision (i.e. commits an error) with known probability ϵ, which is less than 0.5 – so in other words, each voter has a better than 50/50 chance of guessing correctly. In this case, the chances that group (combined by majority vote) makes the correct decision is guaranteed to *increase* as the number of members of the group increases. The first work to apply this to ensemble learning was Dietterich (2000), which is a really nice paper, I recommend you read it. As we said, imagine a single voter has a probability ϵ of making an error, then among a group of M voters, the probability of *exactly* k making errors is given by:

$$p(\text{exactly } k \text{ errors}) = \binom{M}{k} \epsilon^k (1-\epsilon)^{(M-k)} \tag{49}$$

We can then sum this probability for all k greater than half plus one of the voters, in order to get the probability of error in a majority vote.

$$p(majority\ vote\ error) = \sum_{k \geq \lceil \frac{M+1}{2} \rceil} \binom{M}{k} \epsilon^k (1-\epsilon)^{(M-k)} \tag{50}$$

This is illustrated in the figures below.

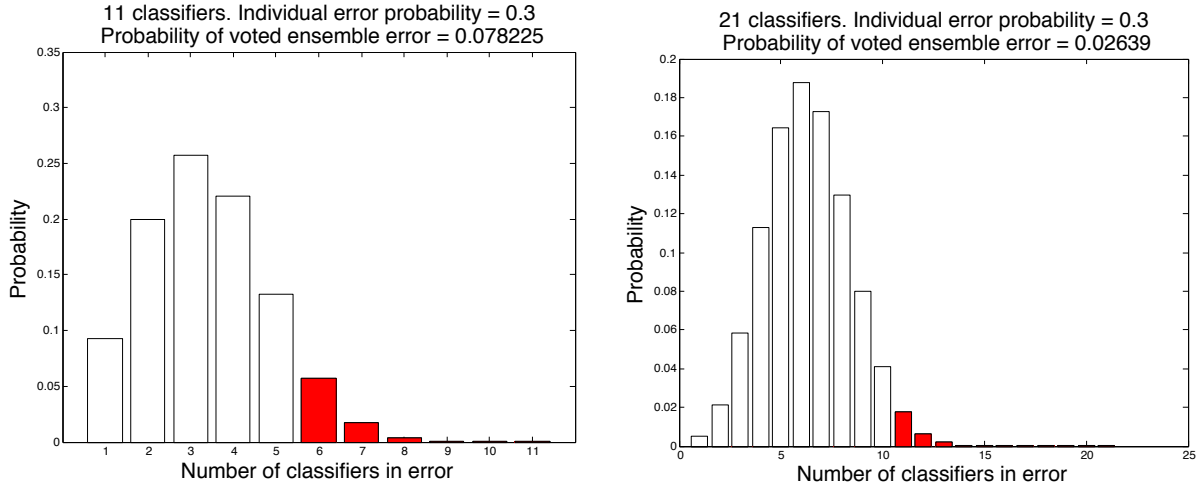


Figure 12: Probability of error for voting ensembles of different sizes (left: 11 classifiers, or right: 21.)

The shaded red area is the probability that the voting ensemble makes an error. Notice in the left figure (with 11 voters) the area is quite big, with a 7.8% chance of ensemble error. In the right figure, (with 21 voters) it is smaller, at about 2.6% chance of error. Yet in both, the chances of an error for an *individual* voter is the same, $\epsilon = 0.3$. The voters are the same, but the ensemble increases in performance because there are more independent voters.

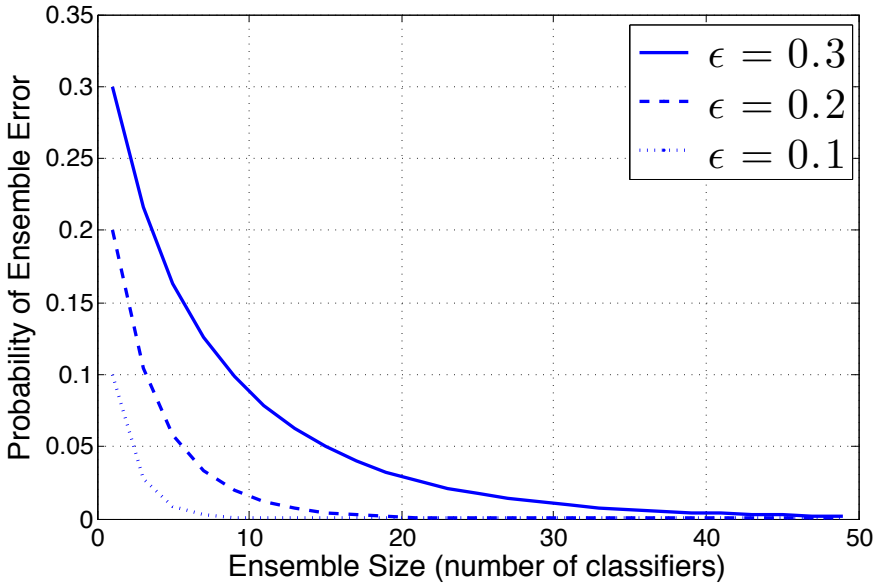


Figure 13: Theoretical majority voting error as we increase the ensemble size (number of voters), *assuming errors are statistically independent.* If each model has only 20% error rate ($\epsilon = 0.2$), the ensemble will have $< 1\%$ error if we combine 15 of them, or $< 0.1\%$ if we combine 20 of them.

We can illustrate this idea further, and plot the voting error as the number of voters increases. Figure 13 shows this, for different values of ϵ. Note that the error approaches zero as the number of voters increases.

Seems like magic, right? No — the critical assumption here is that the voters make *statistically independent* errors. An interesting question is, how realistic is this voting independence assumption — if we generate classifiers, will they be statistically independent? Probably not. Can you see why?

5.2 Training a good ensemble... is not so easy.

If we were to train two classifiers from two *identical* training sets, the only difference between the final models will emerge from their learning algorithms. If the learning algorithm was the same, then we get identical models. We don't want that in an ensemble — otherwise there's no point in having a set of predictors — we want some sort of "diversity" in their predictions. So, what if we took our training set, and *divided* it amongst the classifiers? If we had for 2000 examples, we could make two classifiers by giving them 1000 examples each. We know the usual assumption that our training data is independent and identically distributed — so surely this will result in perfectly independent classifiers? We can extend this idea – dividing the 2000 examples amongst 3, 4, or more classifiers.

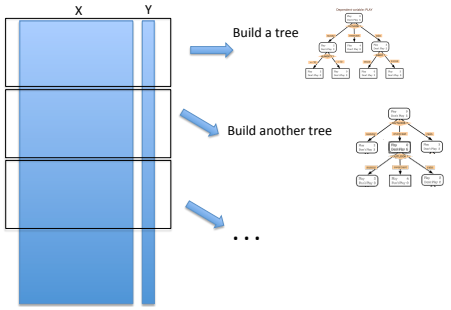


Figure 14: Should we divide the training set between our classifiers?

The results of this process are plotted below, using two different types of classifier – on the left is decision trees, and on the right, Naive Bayes. So we've supposedly supplied independent training sets – and (quite disappointingly) the error of the decision tree ensemble is remains flat as we add more trees! This should have matched the behaviour predicted in Figure 13. What went wrong?

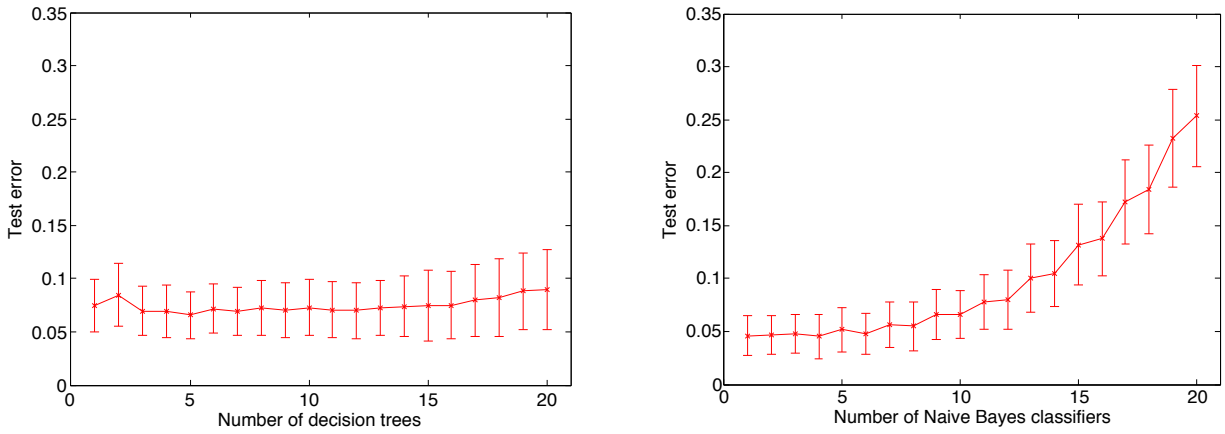


Figure 15: Splice data (3 classes, 60 features, 3175 examples). Using disjoint training sets with ensembles of decision trees (left) and Naive Bayes (right). The loss in performance due to reduced training set size outweighs the gain in performance from having an ensemble.

Remember here, that as we increase the size of the ensemble, *the size of the individual training sets necessarily decreases*, since there are more ensemble members to share the n datapoints between. Consequently, whilst the individuals are very different from each other, they become less and less accurate on testing data, as their training set size decreases. This seems to be a dilemma — decrease the overlap between training sets to make them different, but that causes them to lose accuracy. We need ways of training models such that they are both individually quite accurate, but also sufficiently different from one another that we can achieve something close to the behaviour predicted in Figure 13.

5.3 Parallel Algorithms: Bagging and Random Forests

We're now going to see two interesting algorithms that try to create this "diversity" between classifiers, whilst maintaining a reasonable degree of accuracy. They are both based around the idea of randomly perturbing the classifiers by feeding them slightly different training sets.

5.3.1 The Bagging algorithm

A data *bootstrap* is a random sample of examples from our original dataset. Given a dataset of size n, we randomly select n examples, *with replacement*. The *with replacement* part is very important. It means that we randomly pick one example to be in our training set, then put it back, ensuring that it could be picked *again*. An example dataset and two bootstrap samples are shown below.

Original data

id	x1	x2	x3	y
1	0.18	0.45	0.8	0
2	0.11	0.82	0.07	0
3	0.87	0.3	0.21	1
4	0.34	0.49	0.18	1
5	0.95	0.64	0.63	0
6	0.03	0.59	0.15	1

Bootstrap 1

id	x1	x2	x3	y
1	0.18	0.45	0.8	0
1	0.18	0.45	0.8	0
3	0.87	0.3	0.21	1
4	0.34	0.49	0.18	1
5	0.95	0.64	0.63	0
5	0.95	0.64	0.63	0

Bootstrap 2

id	x1	x2	x3	y
1	0.18	0.45	0.8	0
2	0.11	0.82	0.07	0
3	0.87	0.3	0.21	1
6	0.03	0.59	0.15	1
6	0.03	0.59	0.15	1
6	0.03	0.59	0.15	1

This shows a dataset (left) and two bootstrap samples taken from it (right). Notice that the first bootstrap (top right) contains **2** copies of the first example, but none of the second or sixth examples. The second bootstrap is generated by following the same randomised procedure, but results in a different training set—with **3** copies of the sixth example, but no copies of the fourth example.

The bootstrapping procedure generates slightly different training sets. These differences between training sets are exploited to build different models, using an algorithm — **B**ootstrap **Agg**regat**ing**, or **Bagging** (Breiman, 1996). The combination method is majority vote if we have a classification problem, or a simple average if doing a regression problem. Optionally we could use non-uniform weights; but, this would require an extra hold out dataset to optimise these weights and very often this tends to overfit.

Bagging (requires training data+labels $\mathcal{S}_n$, and choice of number of models M)

---

for $j = 1$ to M **do**
  Take a *bootstrap* sample $\mathcal{S}'_n$ from $\mathcal{S}_n$
  Build a model using $\mathcal{S}'_n$.
  Add the model to the set.
end for
return set of models
For a test point $\boldsymbol{x}$, get a response from each model, and combine predictions.

The result of this procedure varies due to the inherent randomness, and, a typical outcome is trees that look like the following:

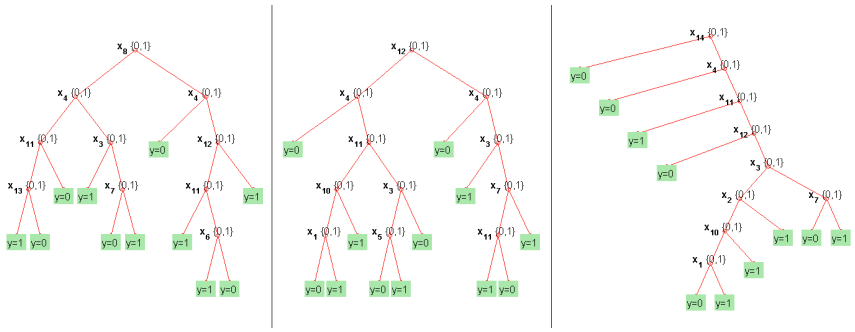


Figure 16: Typical result of Bagging: individual trees are different but still reasonably accurate.

Here, the trees are clearly not identical, and they have different splits as well as different depths. Importantly, they are not as accurate as they could have been, if they had access to all the data, but the differences between them can offset this problem, at least in theory. Let's now see how well it performs in practice. The red curve is for the ensemble using bootstraps (i.e. Bagging), while the blue line shows the performance from a single predictor using all the data.

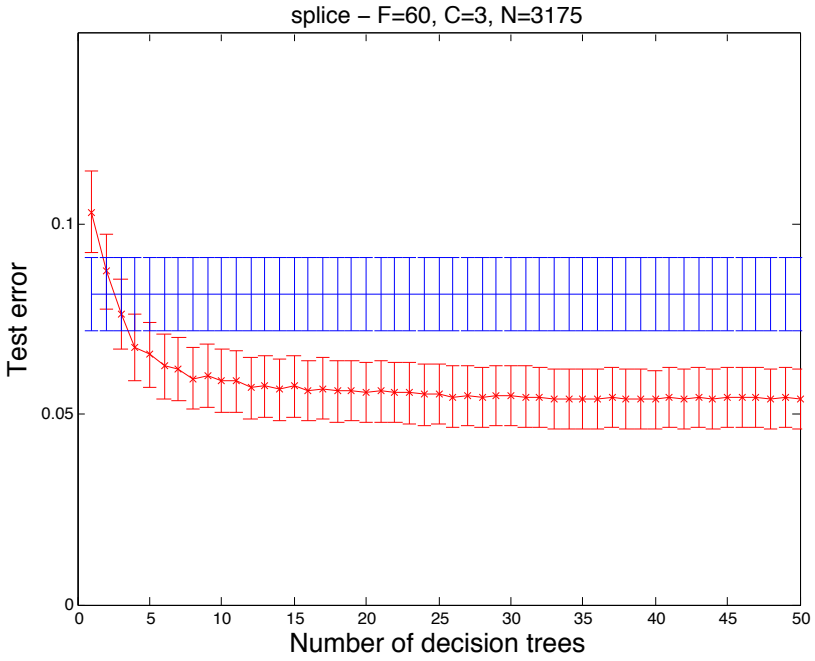


Figure 17: Bagging trees on the Splice dataset.

These results can be contrasted with the theoretical performance predicted in Figure 13. Notice that a single bagged tree (red line) here has an error of approximately 10.1%, so according to the theory prediction, by the time we get to an ensemble of size $M = 10$, we should be well under 1% error — but in fact we have about 6%. Still, this is better than a single tree working on *all* the data, shown as the blue (flat) line above. Let's examine why we might not be achieving this theoretical error.

Analyzing Bagging

Maybe the individual trees are not accurate enough? We are indeed throwing away training data in our bootstrapping procedure. Let's look closer, how much data are we discarding. Well, we know there is a uniform $\frac{1}{n}$ probability of selecting each example. Now imagine a *specific* example. I can tell you that the chances of including that particular example in a data bootstrap is given by:

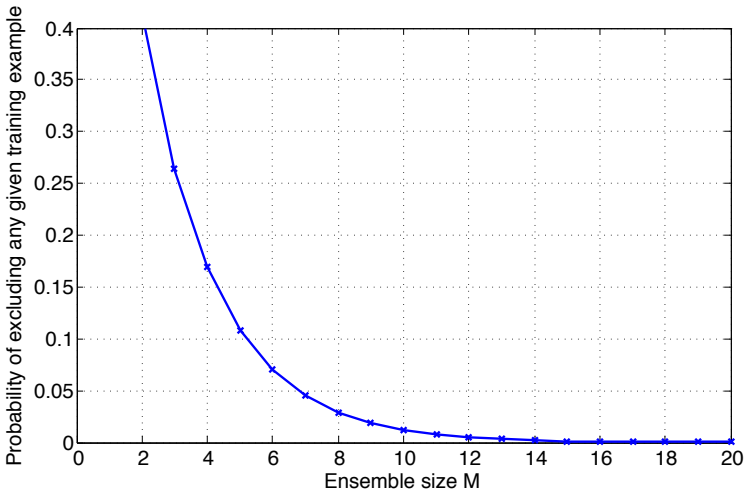
$$p = 1 - \left(1 - \frac{1}{n}\right)^n \tag{51}$$

Let's understand this. Breaking it down, if we imagine we focus on a single specific example, there is a $\frac{1}{n}$ probability of including it in a single example selection, therefore a $1 - \frac{1}{n}$ probability of *not* including it; if we repeat this, there is a $(1 - \frac{1}{n})^n$ chance of *not including* it in the sample of size n. Finally, 1 minus this quantity tells is the chance of including a single example given n repeated random selections. As the data set size increases, this converges to a probability of about 0.6321.

$$\lim_{n\to\infty} \left\{1 - \left(1 - \frac{1}{n}\right)^n\right\} = 1 - e^{-1} \approx 0.6321 \tag{52}$$

The consequence is that for any given bootstrap, it will contain only 63.2% of the original data, and the other 36.8% of examples will be excluded.

You may think this worrying that we are effectively throwing away data, but similar probabilistic arguments can be made to illustrate the chance of ignoring any given example as the ensemble size grows — i.e. what is the chance that a particular example gets completed ignored by an ensemble of size M? With This probability, of excluding a given example from the ensemble, decreases rapidly – shown below. By the time we have an ensemble of size $M = 10$, there is less than a 1% chance that any given example would have been excluded from the ensemble completely.



It appears that as soon as we have a reasonably sized ensemble, we will almost certainly be using all the training data available to us. So this does not explain the sub-optimal performance. What else can we look at? We can also look at the statistical dependence between classifiers generated. We now learn an ensemble of size $M = 50$, and plot these dependencies . This is illustrated in the figure below, where a white square indicates a statistically significant dependence between classifier i and j (χ^2 test, $\alpha = 0.05$).

In the left figure (Decision trees), every single tree is significantly correlated with at least one other! We have *completely violated* our assumption that the classifiers make independent errors, therefore we cannot possibly get to the theoretical performance predicted by Condorcet. In the second figure, we show the same for Naive Bayes classifiers — even more correlations! Why is this? The answer is that the function class explored by Naive Bayes is much more restricted than the trees — it is a 'high bias' model. Thus,

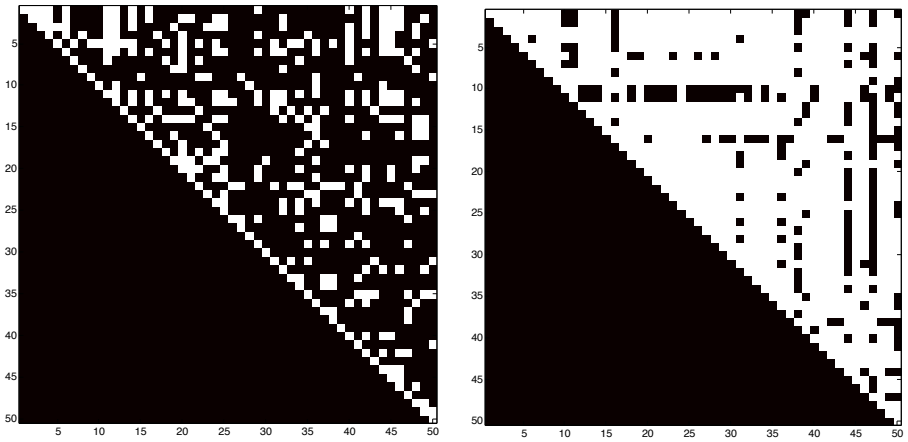


Figure 18: Dependencies between 50 bagged classifiers – trees (left), Gaussian Naive Bayes (right).

the number of possible models that can be learnt from the data is more restricted, hence we end up with near duplicate models, highly correlated. The trees have a larger function class to explore, so we end up with less correlations. Even more interestingly, we can look at the *training errors* of the trees. These are below, indicating that the trees being built from a bootstrap are in fact *overfitting*, and this is helping us!

	Single tree (all data)	Single tree from a bootstrap
Training error	5.8% (±2.9%)	3.6% (±2.1%)
Testing error	16.9% (±4.7%)	20.7% (±4.5%)

Overfitting our trees is helping us? What a paradox! But, not if you consider that it's not the individual tree that makes a prediction.... it is the ensemble. As long as the overfitting patterns of one tree is different from that of another, they will effectively cancel out each others mistakes.

So, we need a way of further reducing dependencies between our classifiers. We will stick with trees for now, and examine a very popular method which produces further differences by randomisation procedures.

5.3.2 Random Forests

Random Forests is an ensemble algorithm only for *decision trees*, hence the name—a forest of randomized trees. The randomisation is generated via *two* mechanisms: a **bootstrap**, (as in Bagging), and a **random selection of features** at each split point. The algorithm is:

Random Forests (input training data+labels $\mathcal{S}_n$, number of trees M)

---

for j = 1 to M **do**
  Take a bootstrap $\mathcal{S}'_n$ from $\mathcal{S}_n$
  Build a tree using $\mathcal{S}'_n$, but, at every split point:
    - Choose a random fraction K of the remaining features.
    - Pick the best feature from that subset.
  Add the tree to the set.
end for
return set of trees
For a test point $\boldsymbol{x}$, get a response from each tree, and take a majority vote.

Commonly, $K = \lceil \sqrt{d} \rceil$ where d is the total number of features. The trees are effectively forced to not choose the best feature at every split, but in a random way. The result is that the trees will be even more different, but still quite accurate. The majority vote ensures that the little drop in accuracy for each tree doesn't matter too much, and the differences between them ensures that they don't make simultaneous mistakes.

The performance of RF on the splice dataset from the previous subsection is shown below, up to 50 trees. Notice that whilst Bagging levelled off in performance around 5.5%, RF continues to decrease test error beyond this. The full run up to 500 trees is shown in Figure 20, zoomed in to show RF levelling off at around 150 trees, with 3.8% average test error (20x 2 fold cross validation).

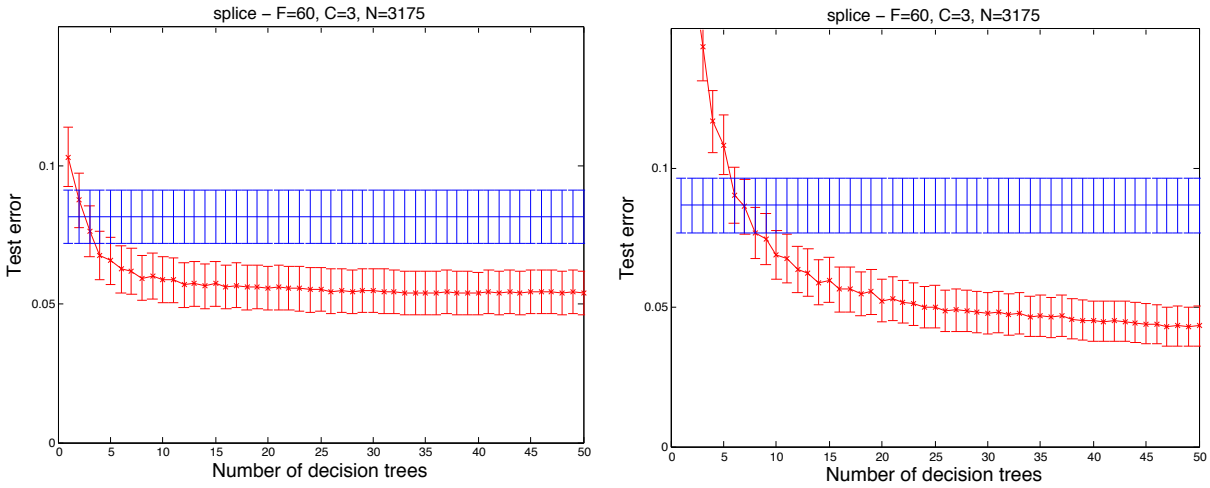


Figure 19: Bagging (LEFT) vs Random Forests (RIGHT) on the Splice dataset.

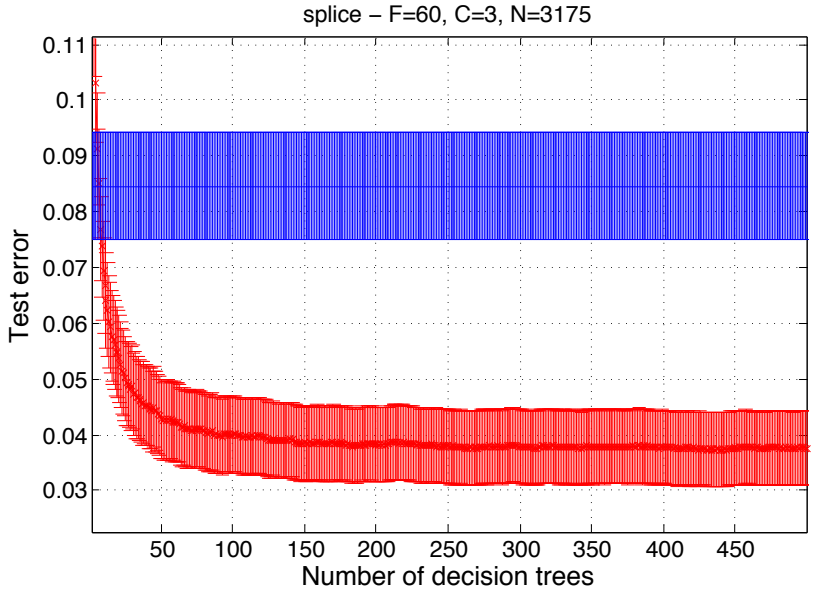


Figure 20: 500 trees on the Splice dataset.

In general RF should be thought of as a *methodology* rather than a precise algorithm. There are various ways to randomise the tree structures, and no single way is the "right" way. However a general point is that RF works best when there are a large number of features — allowing for lots of diversity in the forest.

5.4 A Sequential Algorithm: Boosting

We will now see an ensemble method which works *sequentially*—where each classifier aims to correct the errors of its predecessors. This is called a 'boosting' procedure.

5.4.1 The general idea

Each stage of the procedure involves training one new model, then identifying where it makes mistakes in the training data. Its mis-classified training examples have their emphasis increased, while the correctly classified examples have their emphasis decreased. This generates a new dataset, which is used to train the next model in the boosting chain.

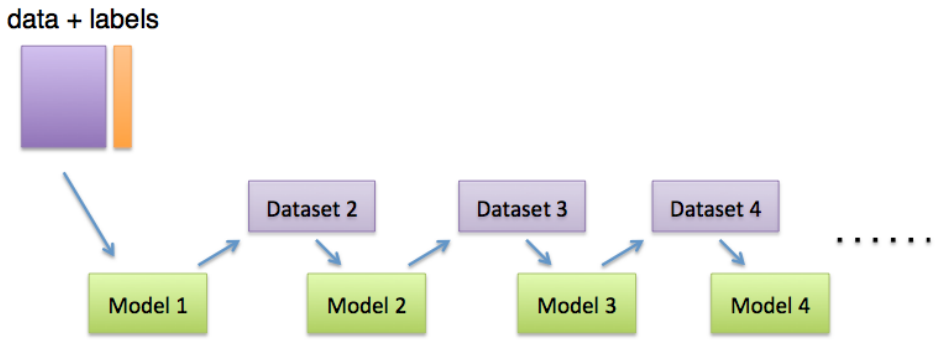


Figure 21: General principle of boosting – each model receives a dataset generated based on the errors made by its predecessor.

The emphasis placed on across the training examples is represented as a probability distribution P. An informal writing of the algorithm is below.

Boosting: input training data+labels $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, and required number of models M

Define a distribution over the training set, $P_1(i) = \frac{1}{n}$, $\forall i$.
for $t = 1$ to M **do**
  Build a model h_t from the training set, using distribution P_t.
  Update distribution to P_{t+1} :
    Increase the weight on examples that h_t incorrectly classifies.
    Decrease the weight on examples that h_t correctly classifies.
end for
For a new testing point $\boldsymbol{x}$, we take a weighted majority vote from $\{h_1, \ldots, h_M\}$.

It is important to know that Boosting occupies somewhat of a special place in the history of ensemble methods. Though the procedure seems heuristic, the algorithm is in fact grounded in a rich body of literature from computational learning theory. In 1989, Yoav Freund and Rob Schapire addressed a question on the nature of two complexity classes of learning problems. The two classes are *strongly learnable* and *weakly learnable* problems. They showed that these classes were *equivalent*, and had the corollary that if we have a weak prediction model, performing only slightly better than random guessing, *should* be able to be "boosted" into an arbitrarily accurate *strong* model. The original Boosting algorithm (by Freund) was a proof by construction of this equivalence, though had a number of impractical assumptions built-in. Schapire and Freund later improved upon this, producing 'Adaboost', the most well known boosting algorithm. So, let's get into some details of how Adaboost works.

Adaboost (Adaptive Boosting) is the most well known of the boosting family of procedures. Adaboost is naturally a two-class classifier, and uses a particular formalism: assume each classifier is $h_t(\boldsymbol{x}) \in \{-1, +1\}$, then the decision of a weighted majority voting ensemble can be written as $H(\boldsymbol{x}) = \text{sign}(\sum_t \alpha_t h_t(\boldsymbol{x}))$. Here, α_t is the weight assigned to voter t in the weighted vote. The detailed algorithm is shown below.

Adaboost: input training data+labels $\{(\boldsymbol{x}_1, y_1)...(\boldsymbol{x}_n, y_n)\}$, and required number of models M

Define a distribution over the training set, $P_1(i) = \frac{1}{n}$, $\forall i$.
for $t = 1$ to M **do**
  Build a classifier h_t, using distribution P_t.
  Set $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  Update distribution to P_{t+1} :
    Set $P_{t+1}(i) = P_t(i)\exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))$
    Renormalize: $P_{t+1} \leftarrow P_{t+1}/Z_t$.
end for

For a new testing point $(\boldsymbol{x}', y')$, we take a weighted majority vote, $H(\boldsymbol{x}') = sign\Big(\sum_{t=1}^{M} \alpha_t h_t(\boldsymbol{x}')\Big)$

The distribution P_t is used by model h_t. For models that can naturally take into account different emphases on getting certain points correct, like Naive Bayes, the model can directly use the distribution—this is called the *reweighting* method. For others, like maybe simple decision stumps, we *re-sample* n items (with replacement) from the training set according to the distribution P_t. Notice that at the first iteration when P_1 is uniform, this is *exactly* equivalent to Bagging. However once the first model is built, the distribution is updated to become non-uniform, and Bagging/Boosting diverge.

5.4.2 Deriving the Algorithm

The algorithm seems like magic: provide a base classifier just *slightly* better than random guessing, and Adaboost will boost it up into an arbitrarily strong ensemble. But where does it come from? What's so interesting about Adaboost is that it can be derived from many different perspectives. The exact same updates can be derived using game theory, or probabilistic models, or dynamical systems, and others. We will now see one of the most commonly cited derivations, the *greedy minimisation of exponential loss*.

Ideally, we would *like* to create an ensemble that minimises the *average classification error*, that is $\frac{1}{n}\sum_i \delta(H(\boldsymbol{x}_i) \neq y_i)$. However, due to the discontinuity of the δ function, this is hard. So, we will instead find an *upper bound* for the classification error, and minimise that instead. The bound we choose is exponential:

$$\frac{1}{n}\sum_{i=1}^{n} \delta(H(\boldsymbol{x}_i) \neq y_i) \leq \frac{1}{n}\sum_{i=1}^{n} \exp(-y_i \sum_{t=1}^{M} \alpha_t h_t(\boldsymbol{x}_i)) \tag{53}$$

This bound is shown in Figure 22.

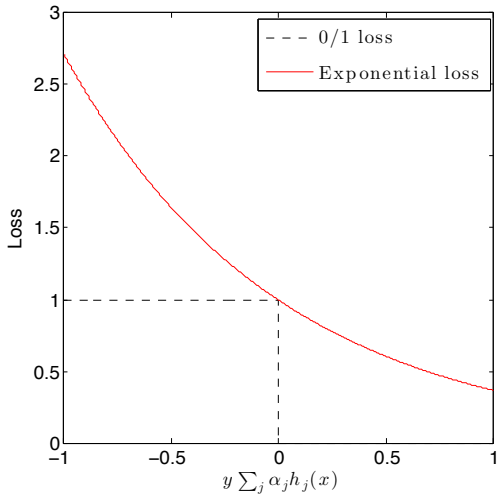


Figure 22: An exponential bound on the classification loss.

The loss function assigns a higher penalty when the ensemble is incorrect, i.e. $y \sum_t \alpha_t h_t(\boldsymbol{x}) < 0$. Since both y and h_t are in $\{-1, +1\}$, then the sign of this product indicates whether the ensemble is correct or not. Take a moment to consider this idea if you do not see it immediately. Notice that the 0/1 loss assigns a value 1 when the ensemble is incorrect, though the exponential loss applies a much harsher penalty. Notice also that when the ensemble is correct ($y \sum_t \alpha_t h_t(\boldsymbol{x}) > 0$), the exponential loss is still not *zero* — this is an important fact that brings Boosting one of its most useful properties, that we will discuss later.

We will see now how we can train an ensemble *sequentially*; i.e. we will greedily minimise the loss of our ensemble, measured by this exponential bound, at each step adding a new classifier to the ensemble. When training the first model h_1, our loss is:

$$E_1 = \frac{1}{n} \sum_{i=1}^{n} e^{-y_i h_1(\boldsymbol{x}_i)} \tag{54}$$

This loss is minimised by the single model h_t getting as many of the n data points correct as it can. Now let's add another model to our ensemble. This means our loss at time step $t = 2$ is:

$$E_2 = \frac{1}{n} \sum_{i=1}^{n} e^{-y_i \sum_{t=1}^{2} \alpha_t h_t(\boldsymbol{x}_i)} \tag{55}$$

where you should notice that we have had to introduce a parameter α_t that says how "important" model t is in the combination of the two classifiers. Intuitively, this should be set proportional to how likely each is to get a datapoint correct, but we will see exactly how to set it shortly — for now just assume it is fixed. The most pertinent point to see here is that due to the properties of the exponential, this loss function decomposes:

$$E_2 = \sum_{i=1}^{n} \underbrace{\frac{1}{n} e^{-y_i \alpha_1 h_1(\boldsymbol{x}_i)}}_{\text{constant}} e^{-y_i \alpha_2 h_2(\boldsymbol{x}_i)} \tag{56}$$

where we have marked the terms that are constant with respect to the parameters h_2, α_2, that we are now optimising. Remembering that we have *already* learnt the parameters of the model h_1, and fixed the α_1 value, we are minimising the following at stage 2:

$$E_2 = \sum_{i=1}^{n} w_2(i) e^{-y_i \alpha_2 h_2(\boldsymbol{x}_i)} \tag{57}$$

where $w_2(i) = \frac{1}{n} e^{-y_i \alpha_1 h_1(\boldsymbol{x}_i)}$. This w_2 term is the relative importance of each datapoint i, and *is constant* when we are learning the parameters of h_2. It quantifies the errors made by the previous classifier on each datapoint; if $w_2(i)$ is large (more precisely if $w_2(i) > 1$) then h_1 got datapoint i incorrect. So in order to

reduce its own loss, h_2 has to focus on data points that h_1 incorrectly classified. If we define $w_1(i) = \frac{1}{n}$ for all i — then $w_2(i)$ is a function of $w_1(i)$, that is $w_2(i) = w_1(i)e^{-y_i\alpha_2 h_2(\boldsymbol{x}_i)}$. This pattern continues when we add a third model:

$$E_3 = \sum_{i=1}^{n} \underbrace{\frac{1}{n} e^{-y_i\alpha_1 h_1(\boldsymbol{x}_i)} e^{-y_i\alpha_2 h_2(\boldsymbol{x}_i)}}_{w_3(i)} e^{-y_i\alpha_3 h_3(\boldsymbol{x}_i)} \tag{58}$$

So, we can see that this is in fact a recursive update:

$$E_3 = \sum_{i=1}^{n} \underbrace{\underbrace{\underbrace{\frac{1}{n}}_{w_1(i)} e^{-y_i\alpha_1 h_1(\boldsymbol{x}_i)}}_{w_2(i)} e^{-y_i\alpha_2 h_2(\boldsymbol{x}_i)}}_{w_3(i)} e^{-y_i\alpha_3 h_3(\boldsymbol{x}_i)} \tag{59}$$

In general form, the relative importance of each datapoint at when training the next model is:

$$w_{t+1}(i) \leftarrow w_t(i) e^{-y_i\alpha_t h_t(\boldsymbol{x}_i)} \tag{60}$$

Notice that we started with a valid distribution for w_1, that is $\sum_i w_1(i) = 1$, but this is not true as we continue adding models, so instead we will normalise the weights at each timestep — but this in no way affects the shape of the loss function. To emphasise that we are normalising the weights, we will call it a distribution, D:

$$P_{t+1}(i) \leftarrow \frac{P_t(i) e^{-y_i\alpha_t h_t(\boldsymbol{x}_i)}}{Z_t} \qquad \textit{where } Z_t = \sum_{i=1}^{n} P_t(i) e^{-y_i\alpha_t h_t(\boldsymbol{x}_i)} \tag{61}$$

which, if you refer back to the algorithm, you will find is the Adaboost update, nice huh? :-) As mentioned, this update means each classifier trains so it corrects the mistakes of its predecessors, and there are *two* things to find: the distribution on which to train each classifier, and the weights α in the majority vote. The α values can be found by considering the loss function that each model is optimising:

$$E_t = \sum_{i=1}^{n} P_t(i) e^{-\alpha_t y h_t(\boldsymbol{x}_i)} \tag{62}$$

where P_t is the distribution we set from the previous time step. By differentiating E_t with respect to α_t, setting it to zero, and solving for α_t, we find that:

$$\alpha_t^* = \arg\min_{\alpha_t} \left\{ E_t \right\} = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{63}$$

where $\epsilon_t = \sum_i P_t(i) \delta(H(\boldsymbol{x}_i) \neq y_i)$, i.e. the weighted error rate of model t. So, with this, we have derived the Adaboost algorithm from scratch.

5.4.3 Summary

Boosting is an incredibly rich family of algorithms to study. There is even a paper published showing how to learn deep networks using boosting algorithms (Huang et al., 2017). One of the reasons Adaboost has been adopted so widely is its flexibility with different classifiers, and the fact that it can be *incredibly* fast when implemented. A seminal paper in computer vision by Viola and Jones (Viola & Jones, 2001) showed how to use it for real-time face detection, and as a result during the early 2000s it quickly became the standard method used in digital cameras.

In summary, ensemble methods are a great tool to have in your arsenal, and a fascinating field to study. Next week we will look deeper into the question of *why* a method works when it does — this is the issue of 'diversity' in ensembles.

6 Ensemble Theory: why do diverse opinions help?

Motivation. We saw last week a family of 'ensemble' algorithms, where we can increase the number of models, seemingly without too much overfitting. One might think that increasing the number of models (i.e. making a bigger overall model) necessarily increases complexity, but this is not the case. The mathematical framework of bias/variance (that we met in week 4) can still be used to understand ensemble algorithms, and why they succeed when they do. This is the topic of this week—the theory of ensemble learning.

Learning Objectives

By the end of this week you should be able to define and use the following concepts correctly in a conversation.

- the bias-variance-diversity trade-off,
- the centroid combiner rule,
- the nature of diversity in squared, cross-entropy, and 0/1 losses

6.1 The Bias-Variance-Diversity decomposition

Last week we met the idea of model 'ensembles', i.e. taking a *set* of models, and combining their predictions. The intuition behind this is similar to a human committee, that the committee will 'average out' the errors of the individuals. For this to happen, the ensemble must be 'diverse' in its predictions. This is an appealing **anthropomorphism**, invoking ideas like the "wisdom of the crowds". However, we can understand this formally, building on the bias/variance concepts we met a few weeks ago. The material for this week is a recent research paper:

> **A Unified Theory of Diversity in Ensemble Learning**
> Journal of Machine Learning Research (vol 24, December 2023)
> https://jmlr.org/papers/volume24/23-0041/23-0041.pdf

The mandatory readings are Sections 1-4, plus the conclusion in Section 8.

Other sections are optional, if you wish to read further.

On the following page are a few tips on reading papers, that may help with this one, or indeed any future reading you may do. These tips themselves are not examinable/assessed, but just presented to help you.

6.2 How to read a research paper

Reading a scientific paper is a very different process than reading a book, a newspaper article about science, or a popular science magazine. They have a very different structure to those other sources you may have read in the past. A typical scientific paper is structured as so:

Abstract: 150-200 words or so, explains the high level achievements and context of the research.
Introduction: Explains the context of the work in a little more detail, but not big technical things.
Background: Provides the necessary explanations and background to the problem tackled. Often very technical, and with lots of pointers to various other papers.
Methods: Explains the core technical achievement—usually very technical.
Experiments: Empirical support to the methods proposed, sometimes blended in with the methods section.
Discussion/Conclusion: Summarises limitations as well as what new challenges are raised by the work.

First, we generally don't read a scientific paper sequentially, in the order it is presented. This may seem odd, but bear with me. The first step is to read the **abstract**. This should summarise the main elements of the paper, and help you decide whether it is really relevant to your work. Then, read the **conclusions**, and to try to figure out what the authors claim they have achieved, and hopefully what evidence they have for their claim. Finally, if the abstract and conclusion have still got your interest, read the main body of the paper, starting with **background** if you're not already familiar. When you encounter bits you don't understand, maybe scribble some notes on the paper, but feel free to **skip over them**. Otherwise, you'll hit a wall and not get past it. Truly understanding a paper takes multiple passes, and can take many hours (or weeks, or months!) of dedicated time, so don't expect to read a paper in one sitting. You'll come back to those difficult bits later, and understand more.

You shouldn't just *read* a paper. You should be reading and, simultaneously, evaluating. Critical thinking is the essence of research, and involves asking yourself a series of questions as you read. I can suggest three criteria you might consider, with a set of questions for each. If you can answer these questions, you will have a deeper understanding of the paper.

Originality: What is really 'novel' in this paper? What do the authors claim they have achieved? Are they addressing a new or complex problem? Are they proposing a new approach to an old problem, or demonstrating a new capability or property that has not been seen before? Maybe they present new perspectives, arguments, or insights about an existing problem or solution? How does it relate to other papers you know? An extension or special case? Does it support other work, or contradict it?

Significance: Is this really an important problem to solve, or an important proposed solution? What is the evidence that other people really care about it? How many subsequent papers have cited[11] this one? Are the results really that surprising? Even if they are addressing an important problem—are they addressing it properly, without making too many simplifying assumptions?

Rigour: How thorough have the authors been in their investigation? Have they considered all the relevant literature—are there papers that they should have cited? Is their chain of reasoning solid—their assumptions justifiable? Are there unwritten assumptions that they may be glossing over, hoping you won't notice? Do they present sufficient evidence to really convince you of each of their claims? In simple language, how easy is it to pull a hole in their conclusions?

This has been a short set of tips that hopefully will help you. There are many other guides out there on the web, that may help in reading papers. Another good one is the following:

- How to read a paper (https://web.stanford.edu/class/ee384m/Handouts/HowtoReadPaper.pdf)

[11] You can use Google Scholar to check this. However, be aware that a large citation track takes a while to build up—papers don't generally get any citations until at least a few months to a year or more after publication.

7 An Introduction To Some Required Concepts In Mathematics

Motivation. Related to the theme of the course about "When are Bigger Models Better?", in Lectures 8 – 11 we will learn proofs of two kinds (a) how certain ML algorithms work in non-intuitive ways when applied to situations where the number of trainable parameters is larger than the number of training data and (b) how to measure the dependency of the gap between test and train error on the number of trainable parameters.

These kinds of analysis will require being able to quantify various properties of the loss functions, matrices and sequences of vectors. This chapter will introduce all these basic mathematics that will be required in the last 4 lectures of this module.

Starting from this section we shall assume that the reader is already familiar with the contents in Appendix C - and we shall further build upon it. In particular Appendix C informally introduces the idea of a function. In here we state 2 different ways of specifying a function that we shall use interchangeably - as may be suitable in a given context.

Definition 15 (**Some Notation About Functions**). *Let $\mathbb{R}^p$ and $\mathbb{R}^q$ be $p-$ and $q-$dimensional Euclidean spaces. Then a function F between them would be specified by either of the following 2 definitions,*

$$\begin{aligned} F : \mathbb{R}^p &\to \mathbb{R}^q \\ \boldsymbol{x} &\mapsto f(\boldsymbol{x}) \end{aligned} \tag{64}$$

or in an "in-line" form,

$$\mathbb{R}^p \ni \boldsymbol{x} \mapsto F(\boldsymbol{x}) \in \mathbb{R}^q$$

If one wants to specify that the function F depends on certain parameters (say a vector $\boldsymbol{z}$) then one might replace F with $F_{\boldsymbol{z}}$.

As a demonstration of the above notation, let's consider the squared loss function (say ℓ) on linear predictors at a labelled data $(\boldsymbol{x}, y)$ where y is 1−dimensional and $\boldsymbol{x}$ is $p-$dimensional. Then we have,

$$\begin{aligned} \ell_{(\boldsymbol{x},y)} : \mathbb{R}^p &\to \mathbb{R} \\ \boldsymbol{w} &\mapsto \ell_{(\boldsymbol{x},y)}(\boldsymbol{w}) = \frac{1}{2} \cdot \left(y - \langle \boldsymbol{w}, \boldsymbol{x} \rangle\right)^2 \end{aligned} \tag{65}$$

or in an "in-line" form,

$$\mathbb{R}^p \ni \boldsymbol{w} \mapsto \ell_{(\boldsymbol{x},y)}(\boldsymbol{w}) = \frac{1}{2} \cdot \left(y - \langle \boldsymbol{w}, \boldsymbol{x} \rangle\right)^2 \in \mathbb{R}$$

When the data dependency of the loss is not needed to be emphasized one may also simplify $\ell_{(\boldsymbol{x},y)}(\boldsymbol{w})$ to just $\ell(\boldsymbol{w})$. Note that in above, the inner product $\boldsymbol{w}^\top \boldsymbol{x}$ between vectors $\boldsymbol{w}$ and $\boldsymbol{x}$ is denoted as $\langle \boldsymbol{w}, \boldsymbol{x} \rangle$ - a notation that we urge the reader to get familiar with.

Three key properties of differentiable functions that we will focus on are whether they are "Lipschitz" or not, "convex" or not and whether they are "smooth" or not. Which of these properties are true for a given loss function makes a dramatic difference to how machine learning algorithms behave for them.

7.1 Introduction to Lipschitzness, Convexity and Smoothness

Definition 16 (**Lipschitzness**). *A function $F : \mathbb{R}^p \to \mathbb{R}$ is said to be $L-$Lipschitz if $\forall\ \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^p$, we have,*

$$|F(\boldsymbol{x}) - F(\boldsymbol{y})| \leq L \|\boldsymbol{x} - \boldsymbol{y}\|$$

Consider, $F(x) = x$. Its clear that this is 1−Lispchitz.

But on the contrary, consider the function $F(x) = x^2$. Now we can compare its values between say a point x and 0 and ask if it is true that there exists a $L > 0$ s.t for all x we will have, $|F(x) - F(0)| = x^2 \leq L|x|$. But clearly for no $L > 0$ can the inequality $x^2 \leq L|x|$ hold if $|x| > L$. Thus $F(x) = x^2$ is not a Lipschitz function.

Concept Check...
At this point the reader is strongly urged to read up on the Huber loss and convince themselves that the squared loss is not a Lipschitz function but the Huber loss is.

M.L. for almost all known scenarios is an optimization problem i.e a question of wanting to minimize or maximize some "objective" function. Finding the global minimum or maximum of a function is extremely challenging for a vast number of classes of functions and yet this is easily approximately doable for very many useful M.L. cases. This dichotomy is at the heart of research in this field.

In this regard, the idea of "convexity" has proven to be remarkably fundamental, to the point that the tractability of an optimization problem is nowadays assessed, more often than not, by whether or not the problem benefits from some sort of underlying convexity. And decades ago this was already pointed out in the famous monograph Rockafellar (1993) where it was said,

"... the great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity."

A convex function F can be informally understood as one whose graph between any two points x_1 and x_2 always lie below the straight line joining the points $(x_1, F(x_1))$ and $(x_2, F(x_2))$. *Note, that this notion of convexity does not need the function to be differentiable anywhere.* This general idea of convexity is informally defined via the diagram below,

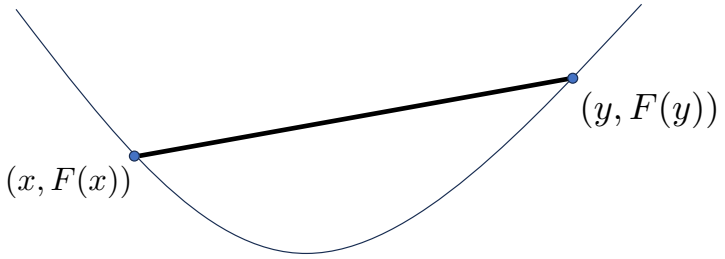


Figure 23: Illustration of Definition 17 for a convex function F

Definition 17 (Convexity (Version 1)). *A function $F : \mathbb{R}^n \to \mathbb{R}$ will be said to be a convex function if $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and $\forall \theta \in [0, 1]$ we have,*

$$F(\theta \boldsymbol{x} + (1 - \theta)\boldsymbol{y}) \leq \theta F(\boldsymbol{x}) + (1 - \theta)F(\boldsymbol{y})$$

This above view of convexity will be extremely important in the last lecture of this course.

Additionally, we would call F to be "strictly convex" if the inequality above holds in the strict sense – i.e it never becomes equality for any choice of $\boldsymbol{x}, \boldsymbol{y}$ and $\theta \in (0, 1)$. Note that at $\theta = 0, 1$ i.e at the end points of the cord, the inequality becomes equality by definition. For intuition consider the function, $F(x) = x^4$ - note that this is an example of a strictly convex function while any $\mathbb{R} \to \mathbb{R}$ convex function is not strictly convex if it becomes a constant on any interval in the domain, like $f(x) = \max\{0, x\}$, the "Rectified Linear Unit (ReLU)" - which is the most common activation function in neural nets.

But if the function can be assumed to be differentiable (as is the case for all optimization examples that we will consider) then the following notion of convexity also becomes useful.

Definition 18 (Convexity (Version 2)). *An at least once differentiable function $F : \mathbb{R}^p \to \mathbb{R}$ is convex if $\forall\ \boldsymbol{x}, \boldsymbol{y}$ we have,*

$$F(\boldsymbol{x}) + \nabla F(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}) \leq F(\boldsymbol{y})$$

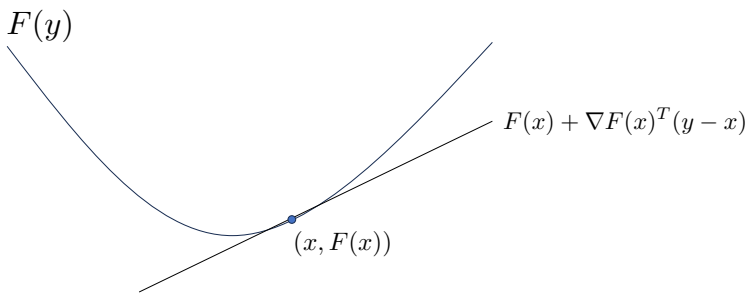


Figure 24: We can visualize Definition 18 as saying that a differentiable convex function can be thought of as having the property that for any point x in the domain, the tangent to the function at that point is below the graph of the function.

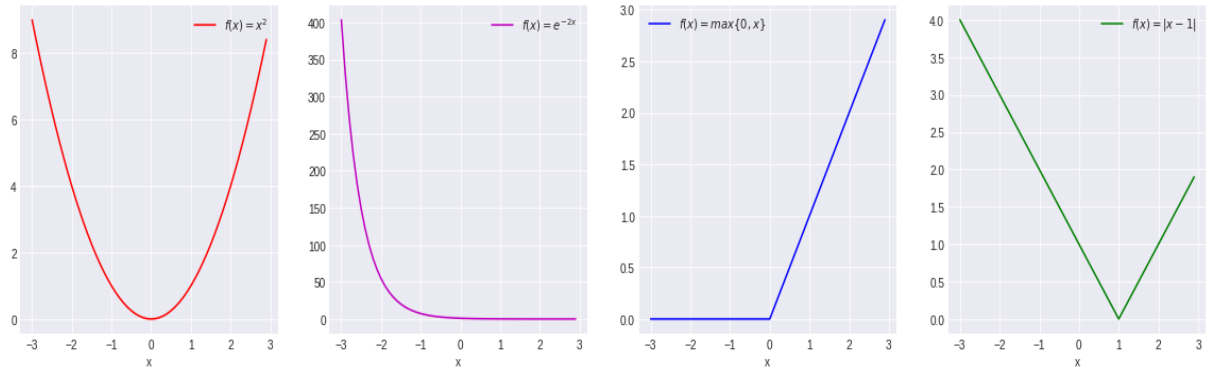


Figure 25: In here we can see represenative samples of what convex functions can look like. From left-to-right notice that we have examples of convex functions which (a) have a unique global minima (b) have no minima at all (c) are not differentiable everywhere and have an uncountable number of global minima and (d) are not differentiable everywhere but have a unique global minima.

Strongly convex functions are a particular subset of convex functions which are of great significance.

Definition 19 (α–Strong Convexity (Version 1)). *For some $\alpha > 0$, a $F : \mathbb{R}^p \to \mathbb{R}$ is said to be α–strongly convex if we have that $F(\boldsymbol{x}) - \frac{\alpha}{2}\|\boldsymbol{x}\|^2$ is a convex function.*

Essentially, strong-convexity captures the intuition of having a convex function be lowerbounded by a paraboloid. From the definition above it is clear that, for any $\alpha > 0$, adding $\frac{\alpha}{2}x^2$ to any convex function would yield a α–strongly convex function. Thus we get a simple mechanism to generate a large number of examples of strongly convex functions.

If we assume differentiability then we can also define strong convexity as follows,

Definition 20 (α–**Strong Convexity (Version 2)**). *For some $\alpha > 0$, an atleast once differentiable function $F : \mathbb{R}^p \to \mathbb{R}$ is said to be $\alpha-$strongly convex if $\forall\ \boldsymbol{x}, \boldsymbol{y}$ we have,*

$$F(\boldsymbol{y}) \geq F(\boldsymbol{x}) + \nabla F(\boldsymbol{x})^{\top}(\boldsymbol{y} - \boldsymbol{x}) + \frac{\alpha}{2} \cdot \|\boldsymbol{y} - \boldsymbol{x}\|^2$$

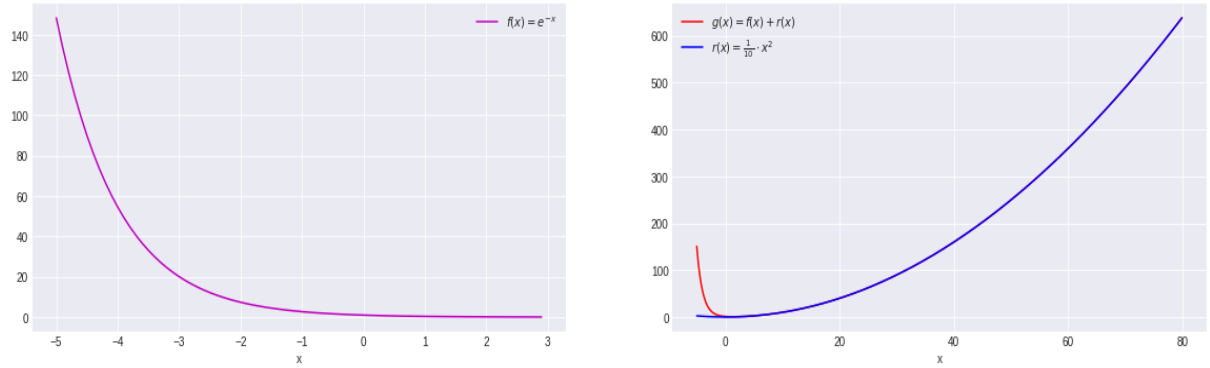


Figure 26: In above we can see how adding an arbitrarily small amount of a strongly convex function r to a convex function f with no minima leads to a strongly convex function g with a unique global minima

Concept Check...
For training data of the form $(\boldsymbol{x}, y)$ and for a real valued predictor h consider the loss being given as $\ell((\boldsymbol{x}, y), f) = e^{-yh(\boldsymbol{x})}$. If $y = \pm 1$ i.e in a binary classification setting, this exponential loss is a very standard loss to try to minimize (over choices of h) for trying to find a good classifier. In particular, it provides a neat sandbox for proving various algorithmic properties which can be messy to establish for other losses. Consider the special case when $h(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle$. Then for any $\alpha \geq 0$, a "regularized" version of the exponential loss for the aformentioned data would be,

$$\ell(\boldsymbol{w}) = e^{-y\langle \boldsymbol{w}, \boldsymbol{x} \rangle} + \frac{\alpha}{2}\|\boldsymbol{w}\|^2$$

Show that the above is strongly convex (in $\boldsymbol{w}$) for $\alpha > 0$ but only convex if $\alpha = 0$.

Concept Check...
Prove the following :

Claim 1. *If F is at least twice differentiable then F is $\alpha-$strongly convex iff $\lambda_{\min}([\partial_i \partial_j F]_{i,j=1,\ldots,p}) \geq \alpha$*

The $p \times p$ matrix $[\partial_i \partial_j F]_{i,j=1,\ldots,p}$ is called the "Hessian" of such an at least twice differentiable F.

We end this short discussion on convexity by noting that algorithmic checking for convexity of a function is extremely difficult and this was formally established in this famous paper, Ahmadi et al. (2013).

Definition 21 (**Lipschitz-Smoothness**). *For some $\beta > 0$, an at least once differentiable function $F : \mathbb{R}^p \to \mathbb{R}$ is said to be $\beta-$smooth or $\beta-$Lipschitz smooth or $\beta-$Gradient Lipschitz if its gradients are $\beta-$Lipschitz i.e $\forall\ \boldsymbol{x}, \boldsymbol{y}$, we have,*

$$\|\nabla F(\boldsymbol{x}) - \nabla F(\boldsymbol{y})\| \leq \beta \|\boldsymbol{x} - \boldsymbol{y}\|$$

Eg. $F(x) = x^2$ is smooth with $\beta = 2$.

Note that this notion of "smoothness" as given above albeit standard across machine learning and optimization books, differs from what is called a smooth function in calculus or analysis textbooks! In there,

smoothness refers to a function being infinitely differentiable and we can see that there are easy examples of infinitely differentiable functions which are not smooth by the above definition, like $F(x) = x^3$. *One needs to be careful about this conflict of terminologies when reading across different sources!*

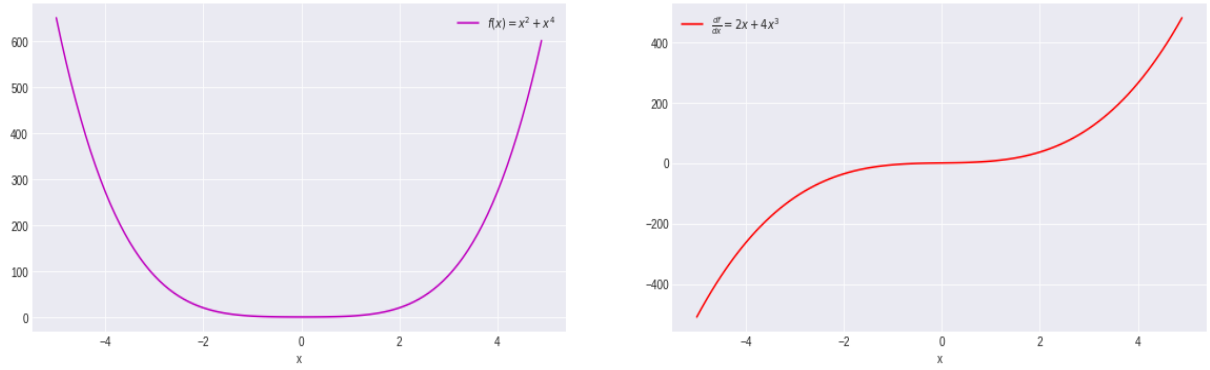


Figure 27: In above we see an example of a quartic function on the left and a plot of its derivative on the right. Its easy to convince onself from these diagrams as to why this quartic function is not Lipschitz smooth.

Concept Check...
Prove the following statements and do the computational exercises embedded inside some of them. Note : consider the following to be functions of the vector $\boldsymbol{x}$ or the scalar x, as the case maybe, and all other letters represent constants/parameters the function depends on.

Some Instructive Examples of Lipschitz Functions

- $\log\left(1+e^{-\beta x}\right), e^{-\beta x}$
  are smooth, Lipschitz & convex functions which are not strongly convex
- $\|\boldsymbol{x}\|_1$
  is non-smooth, Lipschitz & convex but not strongly convex
  – with a unique global minima at $\mathbf{0}$.
- $\frac{1}{1+e^{-\beta x}}$:
  is a smooth, Lipschitz & non-convex function
  – with no critical points
- (a) $-e^{-\lambda\|\boldsymbol{x}\|^2} - p \cdot e^{-\|\boldsymbol{x}-\boldsymbol{a}\|^2}$
  is a smooth, bounded gradient/Lipschitz & non-convex function
  – with a unique global minima and a non-trivial local minima. (find their locations!)
  (b) $\frac{1}{1+e^{-|x|}}$:
  is a non-smooth, Lipschitz & non-convex function
  – with a unique global minima. (find its location!)

Concept Check...
Prove the following statements and do the computational exercises embedded inside some of them. Note: consider the following to be functions of the vector $\boldsymbol{x}$ or the scalar x, as the case maybe, and all other letters represent constants/parameters the function depends on.

Some Instructive Examples of non-Lipschitz Functions

- (a) $x^4 + x^3$
  is a non-smooth, non-Lispchitz & non-convex function
  –with a unique global minima at $x = -\frac{3}{4}$ and an inflection point at $x = 0$.
  (b) $(1-|x|)^2$
  is a non-smooth, non-Lipschitz & non-convex function
  – with 2 global minima at $x = \pm 1$.
- $x^2 + |x|$
  is a non-smooth, non-Lipschitz & strongly convex function.
  – with a unique global minima. (find its location!)
- $\|\boldsymbol{x}\|^2$
  is a smooth, non-Lipschitz & strongly convex function.

From the above two sets of exercises, one might have already started guessing that there is an interesting relationship between convexity and the presence or multiplicity of global minima. One fundamental result about this is in the exercise stated below.

Concept Check...
Prove that, for a convex function every local minima of it is global and if it is also strongly convex then it has a unique global minimum.

Recall the function x^4 that we saw earlier as an example of a strictly convex function. Note that it has a unique global minima (at $x = 0$) while it is not strongly convex. But, consider the function e^{-x}, this is another strictly convex function but it does not have any minima at all. And this dichotomy between these two examples is representative – the reader is encouraged to convince themselves that a strictly convex function can either have no global minima or have only one global minima but not multiple.

Next, consider the function that is at the heart of modern deep-learning, $F(x) = \max\{0, x\}$ – which is a convex function that is not differentiable everywhere (and hence it falls within the ambit of Definition 17 but not 18) - and it has an uncountably infinite number of global minima (i.e all $x \leq 0$). This is neither strictly convex nor strongly convex.

Moving ahead from the fact established in the above exercises, we realize that way too many real-world loss functions have local minima which are not global. In the following we shall see an example of such a loss function - and in this example we shall use a strongly convex regularizer to emphasize that its presence alone cannot ameliorate this complexity of having non-trivial local minima.

Example 7.1 (**Example Of A Neural Loss**). *Consider the following training dataset consisting of* 4 *data,* $(x_1, y_1) = (0.5, -100), (x_2, y_2) = (-1, 300), (x_3, y_3) = (1, 1), (x_4, y_4) = (-0.5, -400)$. *We recall that one of the simplest neural nets is one consisting of a single sigmoid gate of weight* w *which would be mapping,*

$$\mathbb{R} \ni x \mapsto \frac{1}{1 + e^{-w \cdot x}} \in [0, \infty)$$

Then a very natural instance of a regularized squared loss function, which we call ℓ *below, on the above gate for the above training data would be,*

$$\mathbb{R} \ni w \mapsto F(w) \coloneqq \frac{1}{4} \sum_{i=1}^{4} \ell_{x_i, y_i}(w) + \lambda w^2 \coloneqq \frac{1}{4} \sum_{i=1}^{4} \frac{1}{2} \cdot \left(y_i - \frac{1}{1 + e^{-w x_i}} \right)^2 + \lambda w^2 \in [0, \infty) \tag{66}$$

Notice how we think of losses as being univariate functions of the weights, while the gates are thought of as univariate functions of the input data. [12]

Now we can plot this F *above, for say* $\lambda = 0.13$, *as a function of* w *to obtain the following plot - which is clearly a non-Lipschitz function (owing to the regularizer) and neither is it convex (since it has a local maxima) and notice how it has two local minima, only one of which is a global minima.*

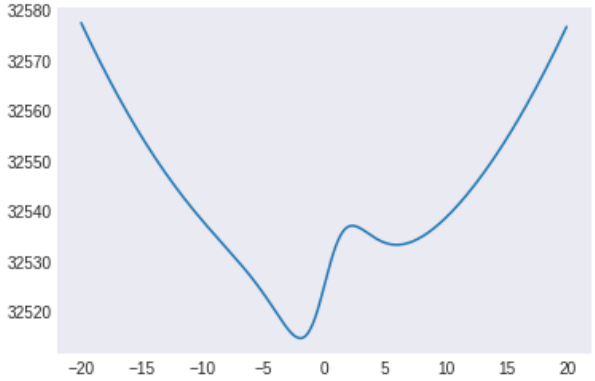


Figure 28: A plot of the empirical loss function, $f(w)$ of a very simple neural net vs its weight w

[12] Up to the λ–term note that the function F in equation 66 is an instance of the quantity "$\hat{R}(f, \mathcal{S}_n)$" defined in Section 0.

7.2 Introduction to Convergence

At this point, the students who have done the course COMP24112 might want to review what they have already seen in Chapter 6A therein, about how gradient descent works on ℓ_2−regularized linear regression. A key feature of that example - which made things very easy - is that the loss function used there was convex. But in a very large number of powerful ML applications in the real-world this property does not hold. Towards enlarging the horizon of possibilities that we want to capture, in what follows we shall again see an example of gradient descent on a regularized squared loss function but now on neural nets - a situation where the loss is not a convex function of the training parameters.

To motivate the idea of convergence let us return to the simple example of a neural loss function that we had seen in equation 66. To match usual conventions in mathematical literature on convergence, we shall rename the weight variable therein as x. Thus we have,

$$\mathbb{R} \ni x \mapsto F(x) \coloneqq \frac{1}{4}\sum_{i=1}^{4} \ell_{x_i,y_i}(x) + 0.13 \cdot x^2 \coloneqq \frac{1}{4}\sum_{i=1}^{4} \frac{1}{2} \cdot \left(y_i - \frac{1}{1+e^{-x \cdot x_i}}\right)^2 + 0.13 \cdot x^2 \in [0, \infty) \tag{67}$$

Recall, that on the above loss function , an implementation of the "Gradient Descent" algorithm at a constant "step-length" of $\eta > 0$ and starting from x_0 would be an iterative execution of the following method,

$$x_{t+1} = x_t - \eta \cdot \frac{\mathrm{d}F}{\mathrm{d}x}\Big|_{x_t}$$

Said in words, at "time t" or at the t^{th}−step we change the current value of the weight additively by an amount which is $-\eta$ times the current value of the gradient - that is we take a step η times the value of the current gradient but in the opposite direction of the current gradient. Let's see visually as to what the progress of such an algorithm looks like when starting from $x_0 = 7$ and the above being run for 10^3 steps (Figure 29), 10^4 steps (Figure 30) and 10^5 steps (Figure 31).

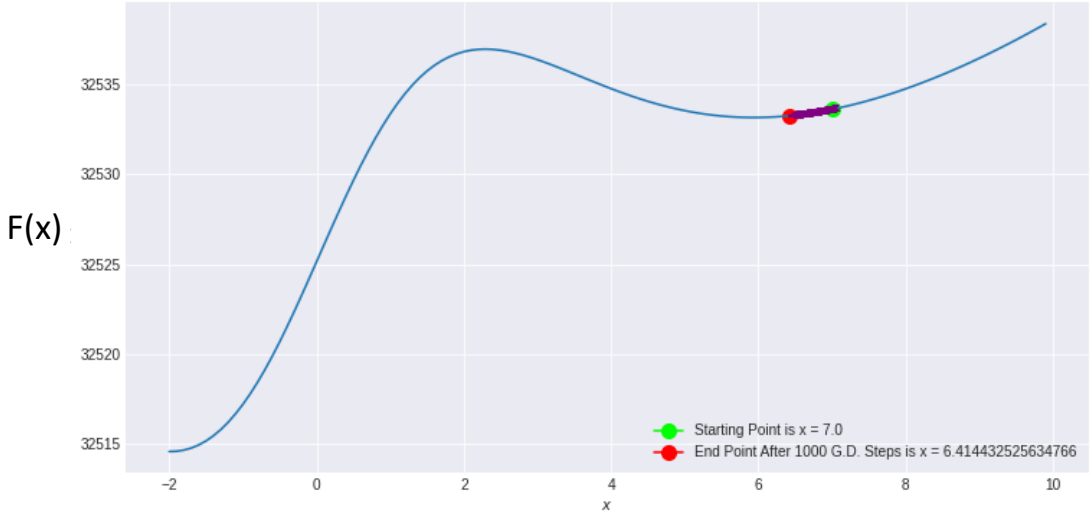


Figure 29: G.D. on F starting from $x_0 = 7$ and using a constant step-length of $\eta = 10^{-3}$ and running for 10^3 steps reaches $x \sim 6.414$

It is clear from the Figures 29 - 31 that the algorithm makes significant progress between where it was after 10^3 steps and the next 9000 steps. But for the next 90000 steps after $t = 10^4$, almost nothing seems to

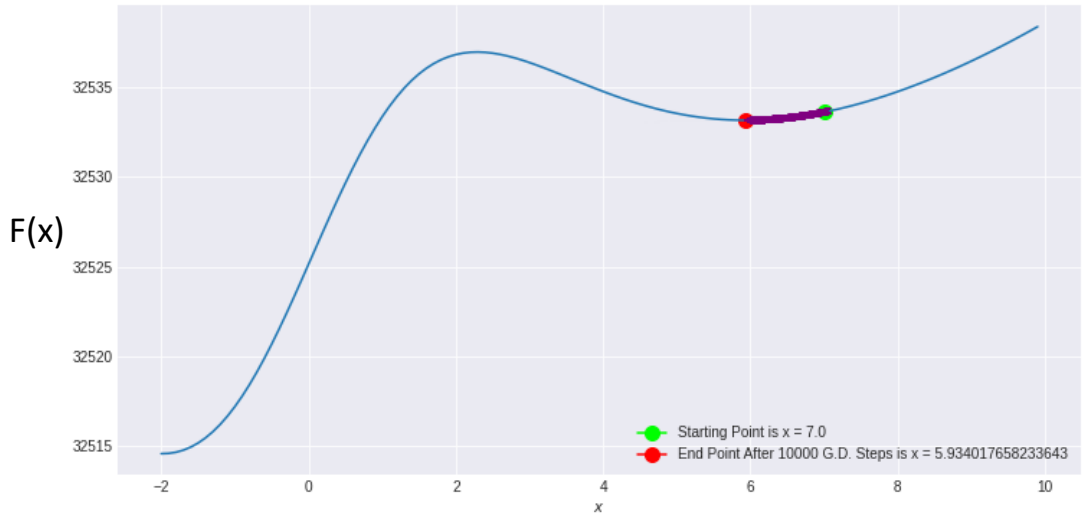


Figure 30: G.D. on F starting from $x_0 = 7$ and using a constant step-length of $\eta = 10^{-3}$ and running for 10^4 steps reaches $x \sim 5.934$

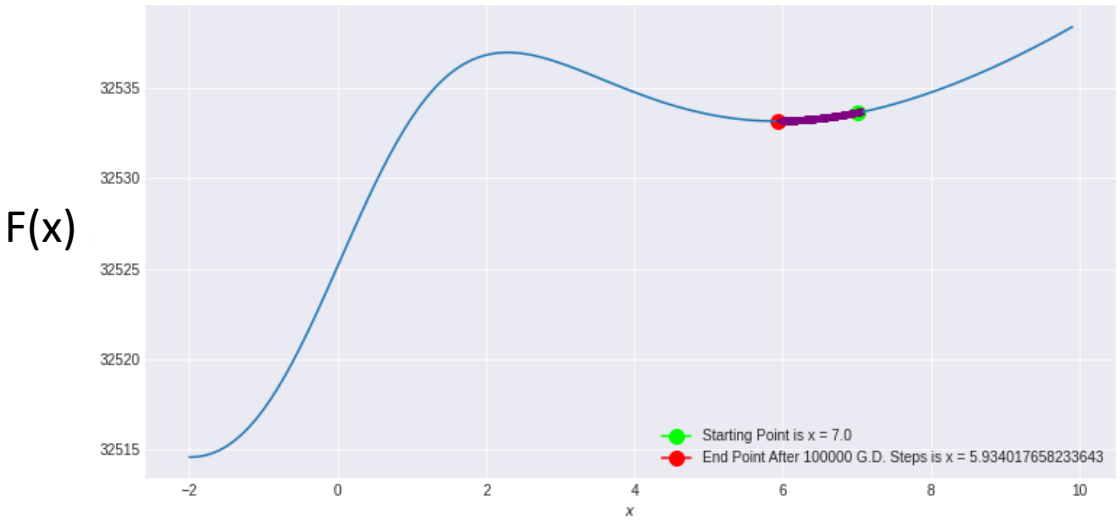


Figure 31: G.D. on F starting from $x_0 = 7$ and using a constant step-length of $\eta = 10^{-3}$ and running for 10^5 steps reaches $x \sim 5.934$

happen - the algorithm seems to have stalled. This is a tell-tale sign of "convergence" - that this sequence of numbers x_t seems to be getting arbitrarily close to a certain number as $t \to \infty$.

Also, since the function f here is an actual example of a neural loss function, along the way these experiments also happen to point out that without a careful choice of starting point and step-lengths, G.D. can get "stuck" at a non-trivial local minima - and not reach the global minima of the function. From Figure 28 we know that this f indeed has a unique global minima, which the G.D. settings in the above diagrams seem to fail to reach.

Concept Check...
Play around with the above function and G.D. on it to figure out values of x_0 and η for which G.D. would converge to its global minima.

Thus we realize that to understand how the gradient descent algorithm works, it's absolutely essential to formally define and understand the notion of convergence. And that's what we shall do below, in general for sequences in arbitrary dimensions - as is the ubiquitous case for G.D. running on multi-dimensional/parameter models that are useful in practical M.L.

Definition 22 (**Defining Convergence (Version 1)**). *A sequence of points $\boldsymbol{x}_i, \forall i = 1, 2, \ldots$ in $\mathbb{R}^n$, will be said to converge to a point $\boldsymbol{x}_* \in \mathbb{R}^n$ if $\lim_{n\to\infty} \|\boldsymbol{x}_n - \boldsymbol{x}_*\|_2 = 0$. If this condition is satisfied then $\boldsymbol{x}_*$ will be called the "limit point" or the "accumulation point" of the sequence.*

Definition 23 (**Defining Convergence (Version 2)**). *A sequence of points $\boldsymbol{x}_i, \forall i = 1, 2, \ldots$ in $\mathbb{R}^n$, will be said to converge to a point $\boldsymbol{x}_* \in \mathbb{R}^n$, if $\forall \epsilon > 0$, all but finitely many members of the sequence satisfy the condition that $\|\boldsymbol{x}_n - \boldsymbol{x}_*\|_2 \leq \epsilon$. If this condition is satisfied then $\boldsymbol{x}_*$ will be called the "limit point" or the "accumulation point" of the sequence.*

The later of the above views is more fundamental because it can be generalized to more situations than just the Euclidean space, $\mathbb{R}^n$. For the purposes of this course, the first definition shall be sufficient.

Concept Check...
Why are the above two definitions of convergence equivalent?

We can build further intuition by seeing two simple examples,

- The sequence $\frac{1}{2}, \frac{1}{2^2}, \frac{1}{2^3}, \ldots$ is a decreasing sequence that converges to 0
- The sequence $\left(-1 - \frac{1}{2}\right), \left(-1 - \frac{1}{3}\right), \left(-1 - \frac{1}{4}\right), \ldots$ is an increasing sequence that converges to the point -1

7.3 Introduction to Supremums and Infimums

Consider the set of numbers $\{0, -1, 1.2\}$. Its quite easy to say that -1 is the lowest number in the set and 1.2 is the largest. But note that this simple idea gets a lot more subtle when we have to deal with infinite sets. Consider the set of numbers which are greater than equal to 0 but less than 1 i.e the interval $[0, 1)$. (Such an interval is said to be "closed" on the left and "open" on the right. In this course we shall not get into the details of this terminology.) Now notice two salient points, that (a) every number in this interval is less than 1 while 1 is not in the interval and (b) that if one picks any number $x \in [0, 1)$ and wants to declare x as the maximum of this interval then one can always come up a very small number $\epsilon > 0$ s.t $x + \epsilon \in [0, 1)$ and thus $x + \epsilon$ becomes a more credible candidate for being called the maximum - and starting from $x + \epsilon$ we can again repeat this argument and this can be continued ad infinitum!

Thus we are led to realize that the language of "maximums" and "minimums" is simply not sufficient to quantify how large an infinite set is. Thus arises the need to define the ideas of "supremums" and "infimums". Towards that we need to first define the idea of an "upper bound" and a "lower bound".

Definition 24 (**Upper Bound & Lower Bound**). *Let I be a subset of $\mathbb{R}$. Suppose there exists a number M s.t $x \leq M$, $\forall x \in I$. Then M is an upper bound on I. Suppose there exists a number m s.t $x \geq m$, $\forall x \in I$. Then m is a lower bound on I.*

To understand the above definition consider two situations (a) For the case $I = [0, 1)$ considered earlier realize that it has uncountably many upper and lower bounds. For a start realize that the integers $1, 2, 3, \ldots$ are all upper bounds on I and further, every real number greater than 1 is an upper bound on I - and there is an uncountably infinite number of them. Similarly, $-1, -2, -3, \ldots$ are all lower bounds on I and that further, every real number less than 0 is a lower bound on I - and there is an uncountably infinite number of them.

(b) Consider $I = [0, 1) \cup [1.5, 2]$. Now a number like 1.3 is an upper bound on a part of the I and is a lower bound on another part of I. Thus 1.3 is neither an upper bound nor a lower bound on this I. But realize

that every real number greater than 2 is an upper bound on this I and every real number less than 0 is a lower bound on this I.

The realization of the multiplicity of the idea of upper and lower bounds leads us to the definition of supremums and infimums.

Definition 25 (**Supremum**). *Let I be a subset of $\mathbb{R}$. Suppose there exists an upper bound M on I s.t $M \leq M'$, $\forall$ upper bounds M' on I. Then M is the "least upper bound" on I and it's called the "supremum" of I and is denoted as $\sup I$. And when $\sup I \in I$ we call it the "maximum" of I, $\max I$.*

Definition 26 (**Infimum**). *Let I be a subset of $\mathbb{R}$. Suppose there exists a lower bound m on I s.t $m \geq m'$, $\forall$ lower bounds m' on I. Then m is the "greatest lower bound" on I and it's called the "infimum" of I and is denoted as $\inf I$. And when $\inf I \in I$ we call it the "minimum" of I, $\min I$.*

For intuition, we can look back at the previous examples to realize that (a) when $I = [0, 1)$, we have $\sup I = 1$ and $\inf I = 0 = \min I$. And when $I = [0, 1) \cup [1.5, 2]$ we have, $\inf I = 0 = \min I$ and $\sup I = 2 = \max I$.

It is easy to see that one can as well use the above definitions when the interval under consideration arises as the evaluation of some real valued function on a space of functions. Like, consider the "functional" F which maps a polynomial (say f) to its derivative at 1 i.e $F(f) = \frac{\mathrm{d}f}{\mathrm{d}x}|_{x=1}$. Now consider a space of quadratic functions given as, $\mathcal{F} = \{x^2 + m \cdot x \mid m \in [-1, 1]\}$. Then we have $F(x^2 + m \cdot x) = 2 + m$ – and its largest value over all functions in the set $\mathcal{F}$ is 3. This we shall write as, $\sup_{f \in \mathcal{F}} \left(\frac{\mathrm{d}f}{\mathrm{d}x}|_{x=1}\right) = 3$

Concept Check...
Recompute the value of $\sup_{f \in \mathcal{F}} \left(\frac{\mathrm{d}f}{\mathrm{d}x}|_{x=1}\right)$ if $m \in [-1, 1)$

7.4 Introduction to Spectral Norms of Matrices

Definition 27. *For any matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ we define its $(2,2)$-norm as,*

$$\|\boldsymbol{A}\|_{2,2} \coloneqq \sup_{\|\boldsymbol{x}\|_2 = 1} \|\boldsymbol{A}\boldsymbol{x}\|_2 = \sup_{\boldsymbol{x} \neq 0} \frac{\|\boldsymbol{A}\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2}$$

When its clear from context, the above will also be called just the $2-$norm of $\boldsymbol{A}$ and denoted as $\|\boldsymbol{A}\|_2$ or spectral norm of $\boldsymbol{A}$.

Its clear from the above definition that the spectral norm of a matrix captures the largest the matrix can stretch in length any unit vector in its domain. As a special case of the above consider the situation when $n = 1$ i.e when $\boldsymbol{A}$ is a column vector of m dimensions.

Then we have,

$$\|\boldsymbol{A}\|_2 = \sup_{|x|=1} \|\boldsymbol{A}x\|_2 = \sup_{|x|=1} |x| \|\boldsymbol{A}\|_2 = \|\boldsymbol{A}\|_2 = \sqrt{\sum_{i=1}^{m} A_i^2}$$

Thus we see how the idea of $(2,2)-$norm contains as a special case the more elementary idea of an Euclidean norm of a vector.

Lets see a somewhat tricky example to get a feel for why the idea of a spectral norm of a matrix is more subtle than it might seem at first glance. Consider the following family of matrices parameterized by a real number θ,

$$\boldsymbol{A}(\theta) \coloneqq \begin{bmatrix} 0 & \theta \\ 0 & 0 \end{bmatrix} \tag{68}$$

One can solve the equation $\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}$ to obtain the eigenvalues (λ) and eigenvectors ($\boldsymbol{v}$) for this matrix and realize that its only eigenvalue is 0. But, for an arbitrary vector on the unit circle in $\mathbb{R}^2$ - parameterized as say $\begin{bmatrix}\sin(\alpha)\\ \cos(\alpha)\end{bmatrix}$ we have,

$$\left\|\boldsymbol{A}(\theta)\begin{bmatrix}\sin(\alpha)\\ \cos(\alpha)\end{bmatrix}\right\| = \left\|\begin{bmatrix}\theta\cos(\alpha)\\ 0\end{bmatrix}\right\| = |\theta|\cdot|\cos(\alpha)|$$

Thus we have,

$$\|\boldsymbol{A}(\theta)\| = \sup_{\alpha}|\theta|\cdot|\cos(\alpha)| = |\theta|$$

Thus by choosing θ to be arbitrarily large in magnitude we can make the spectral norm of the matrix $\boldsymbol{A}(\theta)$ as large as we want, while the largest eigenvalue magnitude (also called the "spectral radius" of a matrix) remains at 0 at all θ. In general one can show that the spectral radius of a square matrix is upperbounded by its spectral norm but as we see in this example, the gap between them can be arbitrarily large. One can generalize this argument above to any upper triangular matrix in any dimensions with 0s along its diagonal and construct similar examples in larger dimensions.

Extra Reading (Non-Assessed)
The reader is strongly encouraged to read up about the ideas of a "singular value" of a matrix and understand how it is equal to the idea of the 2−norm given above.

7.5 Introduction to Rank of Matrices

Inside $\mathbb{R}^2$ denote two special vectors, $\boldsymbol{e}_1 = \begin{bmatrix}1\\0\end{bmatrix}$ and $\boldsymbol{e}_2 = \begin{bmatrix}0\\1\end{bmatrix}$, Note that any vector $\boldsymbol{w}$ in $\mathbb{R}^2$ can be written as $\boldsymbol{w} = w_1\boldsymbol{e}_1 + w_2\boldsymbol{e}_2$ for some choice of numbers w_1 and w_2. Hence the two vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are said to "span" the "vector space" $\mathbb{R}^2$. We can formally define the idea of "span" as follows,

Definition 28 (**Span**). *Given a finite set of vectors $\boldsymbol{v}_1,\ldots,\boldsymbol{v}_k$, each in $\mathbb{R}^n$, we denote,*

$$\text{Span}(\{\boldsymbol{v}_1,\ldots,\boldsymbol{v}_k\}) \coloneqq \left\{\sum_{i=1}^{k} w_i\boldsymbol{v}_i \mid w_i \in \mathbb{R}, \forall i = 1,\ldots,k\right\}$$

And we will call the set of vectors $\{\boldsymbol{v}_1,\ldots,\boldsymbol{v}_k\}$ as being the "spanning set" for the (vector) space, $\text{Span}(\{\boldsymbol{v}_1,\ldots,\boldsymbol{v}_k\})$.

Infact, a little more thinking gets us to realize that every vector in $\mathbb{R}^2$ can be written in a unique way as a weighted linear combination of $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$. This makes the set $\{\boldsymbol{e}_1,\boldsymbol{e}_2\}$ not only a spanning set for $\mathbb{R}^2$ but also a "basis" for the space $\mathbb{R}^2$. That $\mathbb{R}^2$ can be spanned by a basis of size two is what defines the fact that $\mathbb{R}^2$ is a "2−dimensional" vector space.

Concept Check...
Using the fact that $\boldsymbol{e}_1^\top\boldsymbol{e}_2 = 0$, prove the claim that every vector in $\mathbb{R}^2$ can be written in a unique way as a weighted linear combination of the vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$.

It is easy to see how to extend the above discussion to $\mathbb{R}^n$.

And, we give the following definitions in that generality,

Definition 29 (**Basis & Dimension**). *Given a finite set of mutually orthogonal vectors $\boldsymbol{z}_1,\ldots,\boldsymbol{z}_p$, each in $\mathbb{R}^n$, we denote,*

$$p = \text{dimension}\left(\text{Span}(\{\boldsymbol{z}_1,\ldots,\boldsymbol{z}_p\})\right)$$

$$\{\boldsymbol{z}_1,\ldots,\boldsymbol{z}_p\} \text{ is an instance of a basis of } \text{Span}(\{\boldsymbol{z}_1,\ldots,\boldsymbol{z}_p\})$$

It is important to note that (a) in above $\text{Span}(\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_p\})$ *has other basis too than the one specified above and (b) the ideas of a "basis" and "dimension" do not need the notion of orthogonality to be defined - as was assumed in above. Towards understanding those more general ideas, that the reader is strongly encouraged to read up on the formal definition of a vector space (link) - an idea which will be good to know to understand some of the details in Chapter 9.*

Concept Check...
Let $\boldsymbol{z} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$. Now convince yourself that the set of vectors $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{z}\}$ is also a spanning set for $\mathbb{R}^2$. Now choose an arbitrary vector, say $\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ and show that it can be realized as multiple distinct linear combinations of the vectors $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{z}\}$. Thus realize that this set of three vectors although a spanning set for $\mathbb{R}^2$ is not a basis.

Now, consider the following 3 matrices,

$$\boldsymbol{A} \coloneqq \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}, \boldsymbol{B} \coloneqq \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix}, \boldsymbol{C} \coloneqq \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix},$$

All the 3 matrices map from $\mathbb{R}^2 \to \mathbb{R}^2$ i.e given any arbitrary vector $\boldsymbol{v} \coloneqq \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \in \mathbb{R}^2$ we gave $\boldsymbol{A}[\boldsymbol{v}], \boldsymbol{B}[\boldsymbol{v}]$ & $\boldsymbol{C}[\boldsymbol{v}]$ all in $\mathbb{R}^2$. We realize that, $\forall \boldsymbol{v} \in \mathbb{R}^2$, we have,

$$\boldsymbol{A}[\boldsymbol{v}] = 2v_1\boldsymbol{e}_1 + v_2\boldsymbol{e}_2, \ \boldsymbol{B}[\boldsymbol{v}] = -v_1\boldsymbol{e}_1$$

$$\boldsymbol{C}[\boldsymbol{v}] = 0$$

Relating back to the earlier discussion we can see that the above is tantamount to saying that,

$$\text{dimension}\left(\{\boldsymbol{A}[\boldsymbol{v}] \mid \boldsymbol{v} \in \mathbb{R}^2\}\right) = 2, \ \text{dimension}\left(\{\boldsymbol{B}[\boldsymbol{v}] \mid \boldsymbol{v} \in \mathbb{R}^2\}\right) = 1$$

$$\text{dimension}\left(\{\boldsymbol{C}[\boldsymbol{v}] \mid \boldsymbol{v} \in \mathbb{R}^2\}\right) = 0$$

This is what is encapsulated by the idea of "rank" of a matrix and the calculation done above can be said to have proven that the rank of the matrices $\boldsymbol{A}, \boldsymbol{B}$ &$\boldsymbol{C}$ are $2, 1$ & 0 respectively. *It is easy to see how to extend the above discussion to arbitrary rectangular matrices.* Thus we have motivated the following definitions,

Definition 30 (**Image and Rank of a Matrix**). *Given any* $m \times n$ *matrix* $\boldsymbol{T}$ *i.e* $\boldsymbol{T} : \mathbb{R}^n \to \mathbb{R}^m$*, we define its "image" as,*

$$\text{Image}(\boldsymbol{T}) \coloneqq \{\boldsymbol{T}[\boldsymbol{v}] \mid \boldsymbol{v} \in \mathbb{R}^n\}$$

Then we can define,

$$\text{rank}(\boldsymbol{T}) \coloneqq \text{dimension}(\text{Image}(\boldsymbol{T}))$$

Lastly, if $\text{rank}(\boldsymbol{T}) = \min\{n, m\}$*, then we call* $\boldsymbol{T}$ *to be of "full rank".*

Definition 31 (**Null Space of a Matrix**). *Given any* $m \times n$ *matrix* $\boldsymbol{T}$ *i.e* $\boldsymbol{T} : \mathbb{R}^n \to \mathbb{R}^m$*, we define its "null space" as,*

$$\text{Null Space}(\boldsymbol{T}) \coloneqq \{\boldsymbol{v} \in \mathbb{R}^n \mid \boldsymbol{T}[\boldsymbol{v}] = 0\}$$

Note that sometimes "null space" is also called a "kernel" and we would write the above set as $\ker(\boldsymbol{T})$.

The above two ideas shall be of critical importance in Chapter 9 - which gives one of the most basic demonstrations of a key theme in this course, that of understanding the role of overparameterization.

Concept Check...

Consider the matrix $\boldsymbol{Z} = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 0 & 2 \end{bmatrix}$. What is the rank of $\boldsymbol{Z}$ and $\boldsymbol{Z}^\top$?

8 A Formal Introduction to the Gradient Descent Algorithm

Motivation. This module addresses the question *'Are bigger models always better models?'*. This is motivated by the fact that in recent years, very large models with billions of parameters have been shown to outperform almost every other type of model. Its strangely curious that the algorithm used to train such models is almost always some variant of the Gradient Descent (G.D) algorithm. And over extensive experiments it is now well known that G.D. behaves very non-intuitively on large models. This week we will see the formal mathematical statement of the Gradient Descent algorithm. Next week we will use this framework to understand a specific case of G.D. interacting with large models. A basic kind of surprise with G.D is that there exists numerous loss functions s.t on them the G.D. iterates keep getting close to their global minima. Understanding this, is a very deep mystery that is largely unsolved. As an immediate use case of the mathematical definition of G.D., in this lecture we will see a simple example where this phenomenon can be proven.

In the last few years there has been a surge in literature on provable training of various kinds of neural nets in certain regimes of their widths or depths or for very specifically structured data. Motivated by detailed experimental studies it has often been surmised that even basic neutral training algorithms, like the so-called Stochastic Gradient Descent (S.G.D.) running on neural losses – with proper initialization and learning rate – converges to a low-complexity solution that generalizes - when it exists. But quantifying these nebulous ideas has proved to be a daunting task.

This situation is further complicated by the fact, that in the real world (for actually training the large neural net models in use) the successful algorithms take their steps by a complicated transformation of the gradients they compute.

Extra Reading (Non-Assessed)

- For further motivation into the topic the reader may want to read the discussions given in https://cbmm.mit.edu/sites/default/files/publications/CBMM-Memo-067-v4.pdf.
- The reader is urged to look up Sections 12.7 to 12.11 of Zhang et al. (2021) to familiarize themselves with the various kinds of gradient based training methods that are most successful on modern neural nets.

To really be able to appreciate the training technologies in use at the bleeding edge of modern data-science, we need to start from the vanilla form of this genre of algorithms i.e **"Gradient Descent" (G.D.)** whose pseudocode we give below,

Algorithm 2 G.D. on a Differentiable Function $F : \mathbb{R}^p \to \mathbb{R}$

1: **Input:** An initial point $\boldsymbol{w}_1$ and a specification $T > 0$ of the number of steps for the G.D.
2: **Choose :** A step-size sequence $\eta_t > 0, \ \forall t = 1, 2, \ldots$.
3: **for** $t = 1, \ldots, T$ **do**
4: $\quad \boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \cdot \nabla F(\boldsymbol{w}_t)$
5: **end for**
6: **Output :** $\boldsymbol{w}_T$

In this part of the course we shall take some key steps towards answering the following key question,

"If F has global minima then when does Algorithm 2 find one of them, – and which one and how fast?"

A fully general answer to the above question is quite a core topic at the forefront of research. *To the best of our knowledge such questions are still not answered for most neural net classes when being trained via squared loss!*

8.1 Introduction to Proof of Convergence of G.D. & More Demonstrations

At this point, the students who have done the course COMP24112 might want to review what they haD already seen in Chapter 6C therein, about how gradient descent works on quadratic functions. In what follows we shall start with taking a more precise look at the same.

We recall the idea of convergence of gradient descent that was introduced in the previous lecture. In the following theorem we shall see an example of making that precise for a simple objective function - that is strongly convex, differentiable and Lipschitz smooth.

Theorem 8.1. *Consider doing G.D. on the function $F(x) = x^2$, starting from any $x_0 \in \mathbb{R}$, $x_0 \neq 0$ and using any choice of constant step-length of $\frac{1-k}{2}$ for any $k \in (0,1)$. Then, (a) the iterates will asymptotically in time converge to the global minima at $x = 0$ and (b) taking $\frac{\log\left(\frac{|x_0|}{\varepsilon}\right)}{\log \frac{1}{k}}$ number of steps is sufficient for the iterates to be within a distance of ε of the global minima, for all $\varepsilon > 0$.*

One way to read the above conclusion is to realize that for G.D to reach ε close to the global minima, it is sufficient for G.D needs to take $\frac{1}{\log \frac{1}{k}} \cdot \left(\log|x_0| + \log\left(\frac{1}{\varepsilon}\right)\right)$ number of steps. One often ignores the constants involved here and focusses on the ε dependence of the required runtime and states this required number of steps to be scaling with ε as, $\mathcal{O}\left(\log\left(\frac{1}{\varepsilon}\right)\right)$. This is called "linear time convergence" - the reasons for this name if we were to explain would take us far off course.

Proof. Since $x = 0$ is the (unique) global minima of the objective function F (and $F(0) = 0$), we shall firstly try to find sufficient conditions so that $\lim_{t\to\infty} x_t = 0$ for x_t being given via Algorithm 2.

From the specification of the iterates of the algorithm we have,

$$\begin{aligned} x_{t+1} =& x_t - \eta_t F'(x_t) \\ =& x_t - \eta_t \cdot 2x_t \\ =& (1 - 2\eta_t)x_t \end{aligned} \tag{69}$$

From here we can "unroll" the recursion relation back in time to get,

$$\begin{aligned} x_{t+1} &= (1-2\eta_t)x_t = (1-2\eta_t)(1-2\eta_{t-1})x_{t-1} \\ &= (1-2\eta_t)(1-2\eta_{t-1})...(1-2\eta_{t-t})x_{t-t} \\ &= x_0 \prod_{i=0}^{t}(1-2\eta_i) \end{aligned} \tag{70}$$

Looking at the above equation we can make a number of observations,

- If it were to be true that $x_0 = 0$ then we can read off from the above that for all times t, we would have $x_t = 0$. Thus the algorithm would not move at all. This is why it was important to assume in the theorem that $x_0 \neq 0$.

  More generally if the algorithm were to start at a point x_0 s.t $F'(x_0) = 0$ i.e x_0 is a "critical point" of F, then the algorithm would not move at all. Thus critical points are not to be used as starting points of G.D. algorithms.

- We want $x_t \to 0$ (the global minima of f) as $t \to \infty$, and one can see from the above formula that a simple sufficient condition for that to happen is if for some constant $k \in (0,1)$, $(1-2\eta_i) = k,\ \forall i$ – and this can be ensured by choosing a constant step-length as,

$$\eta_t = \frac{1}{2} \cdot (1-k) \tag{71}$$

Making the above choice we can rewrite our previous expression for x_{t+1} as,

$$x_{t+1} = x_0 \prod_{i=0}^{t} (k) = x_0 \cdot k^t \tag{72}$$

Thus we have,

$$\lim_{t\to\infty} x_t = \lim_{t\to\infty} x_0 \cdot k^t = 0 \tag{73}$$

In the last equality above we have crucially invoked the fact that $k \in (0,1)$. Thus, we have shown that for a range of choices of constant step-length, gradient descent converges on x^2 to its global minimum, regardless of x_0.

To obtain convergence time ("non-asymptotic") bounds, we check how long it takes to get within a $\varepsilon > 0$ interval of the global minima. That is,

$$|x_{t+1}| \leq \varepsilon \implies |x_0| \cdot k^t \leq \varepsilon \implies t \geq \frac{\log\left(\frac{|x_0|}{\varepsilon}\right)}{\log \frac{1}{k}} \tag{74}$$

Thus we have arrived at the fact that we set out to prove. □

8.1.1 Important Demonstrations of G.D. Convergence

Now we start to shift gears towards non-trivial examples. Recall the definition of convexity and Lipschitz smoothness seen earlier. One of the key insights in the theory of gradient descent is that one of the basic ways in which a function gets non-trivial for G.D. to work with is if one or both of these properties stop being true – *as is almost always the case in the real world!*. (Note that both the properties were true in the example above!). Thus we need to start to go beyond the confines of smooth convex functions.

In Figures 32 and 33 we see explicit experimental data of how the iterates of Algorithm 2 progress with time on two very carefully chosen edge cases. In the first one the objective function is a convex function which is not strongly convex and is neither Lipschitz nor Lipschitz smooth. In the second one the objective is a non-convex function which is neither Lipschitz nor Lipschitz smooth. Thus we see two quite varied examples where this deceptively simple looking gradient descent algorithm seems to be finding the global minima and also while using constant step-sizes i.e η_t in Algorithm 2 is chosen to be a constant (called η in the figure captions) and hence $t-$independent.

Demonstration of G.D. convergence on a differentiable convex function which is neither Lipschitz nor Lipschitz smooth. The reader is urged to recognize that in Figure 32. we see an instance of a GD algorithm running with fixed step size, starting from $x = 2.5$, and 'converging' to $x = 0$.

Demonstration of G.D. convergence on a differentiable function which is neither convex, nor Lipschitz nor Lipschitz smooth. The example in Figure 33 can be said to be more complex than the previous one since the target function here is not even convex. But we yet again see that there is an instance of a constant step-size G.D algorithm which for all practical purposes seems to converge to the global minima of the target.

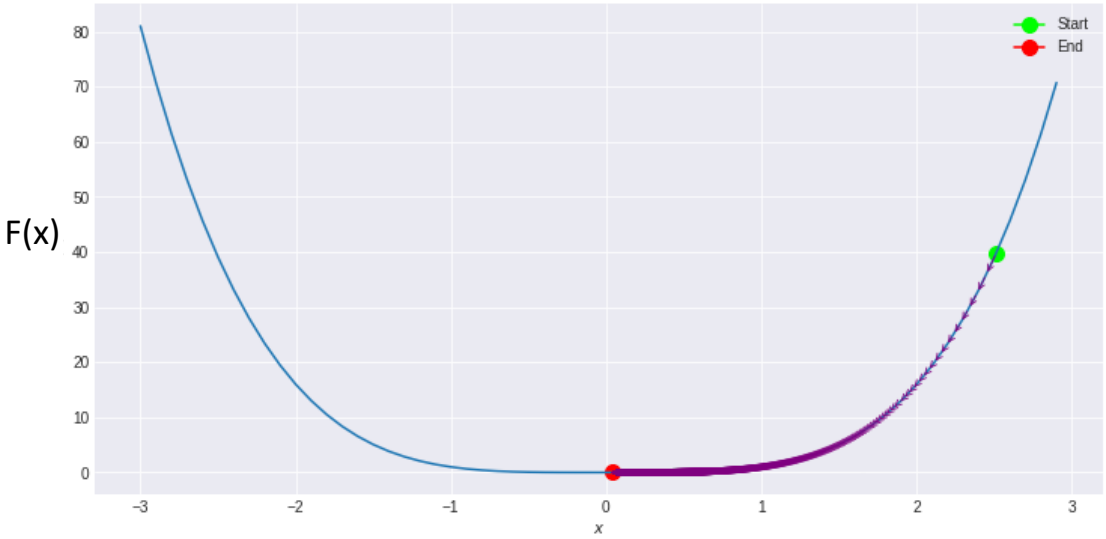


Figure 32: Progress to the unique global minima of Algorithm 2 on the function $F(x) = x^4$ for step-size $\eta = 10^{-3}$ and $T = 10^5$ and the algorithm starting near 2.5

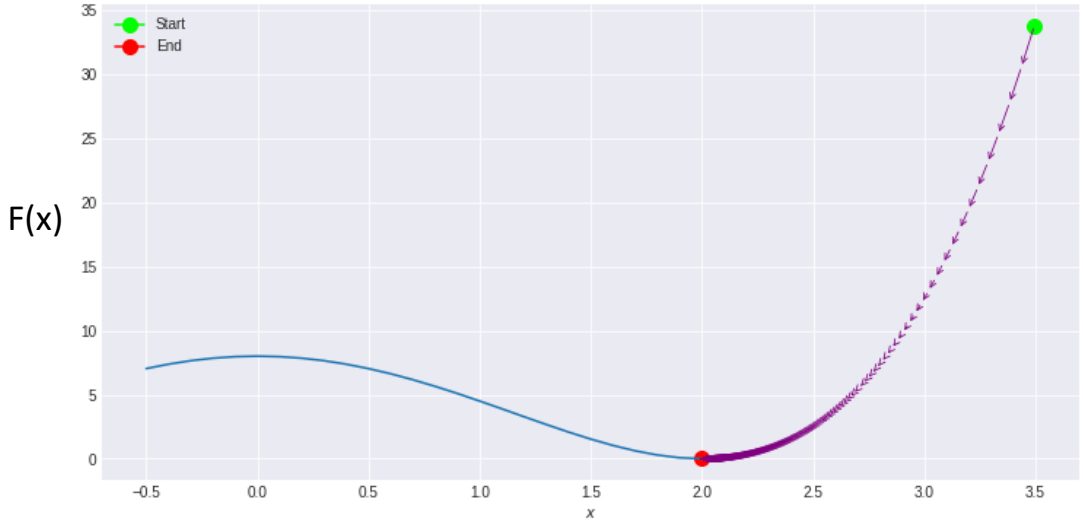


Figure 33: Progress to a global minima of Algorithm 2 on the function $F(x) = \frac{1}{2} \cdot (x^2 - 2^2)^2$ for $\eta = 10^{-3}$ and $T = 10^4$ and the algorithm starting near 3.5

In the examples session we shall see some of the above examples in more details. *The reader is strongly urged to recognize that examples like the one in Figure 33 are not at all rare and in certain ways this can be seen to reflect some of the key features of how actual deep-learning happens.*

9 Understanding Gradient Descent on Overparameterized Linear Regression

Motivation. In this section we will see how to understand the progress of gradient descent in this deceptively simple setup of doing linear regression via this algorithm. We will realize that understanding this case already needs some non-trivial mathematical thinking. In this setup we will see an explicit example of a scenario of the loss function having multiple global minima and hence the question arises as to which one of them will the algorithm find. Eventually we will realize that in this case, because of the number of trainable parameters being larger than the number of training data, makes the algorithm converge to that global minima of the linear regression loss that is nearest to the origin. This is surprising because neither in the loss function nor in the algorithm was there anything that was explicitly promoting this choice. Thus we will see our first example of how overparameterization interacts with gradient based algorithms to induce an effect that is often called "implicit bias" or "algorithmic regularization".

Consider being given n data points in d dimensions, stacked as a $n \times d$ matrix called $\boldsymbol{X}$ - whose $(i,j)-$entry would be denoted as X_{ij}. Let the corresponding "labels" be given as a column vector $\boldsymbol{y} \in \mathbb{R}^{n\times 1}$ - whose $i^{\text{th}}-$entry would be denoted as y_i. Now we aim to understand the "linear regression" optimization question which can be stated as,

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^d} \frac{1}{2}\left\| \underbrace{\boldsymbol{y}}_{n\times 1} - \underbrace{\boldsymbol{X}}_{n\times d}\underbrace{\boldsymbol{\beta}}_{d\times 1}\right\|_2^2$$

This motivates us to define the empirical loss function of linear regression, $\hat{R}$ as,

$$\hat{R}(\boldsymbol{\beta}) = \frac{1}{2}\cdot\|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\|_2^2 \tag{75}$$

Notice that, foreshadowing the upcoming analysis, in above we write $\hat{R}$ as being a function of the weight parameters of the linear predictor $\boldsymbol{\beta}$ rather than emphasizing its dependence on the training data. We assume two things about this setup - and what would make this benign looking question a lot more interesting than it might seem a priori,

- We assume that the model is "overparameterized" i.e we are in the regime when n (no. of training data) $< d$ (the number of training parameters).
- The data matrix $\boldsymbol{X}$ is full rank.

Now we need to get through a crucial mathematical fact which lies at the core of why our eventual analysis works out,

Claim 2. *If $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ is of rank n and $n < d$ then $\boldsymbol{X}\boldsymbol{X}^\top \in \mathbb{R}^{n\times n}$ is invertible.*

Proof. For an arbitrarily chosen $\boldsymbol{v} \in \mathbb{R}^n$, consider the following expression,

$$\boldsymbol{v}^\top(\boldsymbol{X}\boldsymbol{X}^\top)\boldsymbol{v} = \left\|\boldsymbol{X}^\top\boldsymbol{v}\right\|_2^2$$

Now we realize that $\boldsymbol{X}^\top$ is a full-rank (rank n) matrix mapping from (small) n dimensions to (large) d dimensions. Hence by the rank-nullity theorem (link) it follows that it does not map non-zero vectors to 0 and hence it has a trivial null-space. Thus $\boldsymbol{v}^\top(\boldsymbol{X}\boldsymbol{X}^\top)\boldsymbol{v} > 0$ for all non-zero v. This in turn means that $\boldsymbol{X}\boldsymbol{X}^\top$ has a trivial null-space (i.e it has only one element, the zero vector) and that proves that it is an invertible matrix. □

Given the above realization, we can conclude that at $\boldsymbol{\beta} = \boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{y}$ our loss function $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2$ is at its global minima i.e 0. We denote,

$$\boldsymbol{X}^\dagger \coloneqq \boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{X}^\top)^{-1} \in \mathbb{R}^{d\times n}$$

and note that $\boldsymbol{X}\boldsymbol{X}^\dagger = \boldsymbol{I}_{n\times n}$. This leads to the terminology of calling $\boldsymbol{X}^\dagger$ as the "pseudo-inverse" of the data matrix $\boldsymbol{X}$. But is this $\boldsymbol{\beta}$ the only global minima of $\hat{R}$? No! And in the next claim we shall realize how there is a multiplicity of global minima.

Claim 3. *Let $\boldsymbol{x}_i^\top \in \mathbb{R}^{1\times d}$ be the rows of the data matrix $\boldsymbol{X}$. Given $\boldsymbol{y}$ and $\boldsymbol{X}$ as above, $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2 = 0$ iff $\boldsymbol{\beta} = \boldsymbol{X}^\dagger\boldsymbol{y} + \boldsymbol{\xi}$ where $\boldsymbol{\xi}$ is s.t $\boldsymbol{\xi}^\top\boldsymbol{x}_i = 0$ for all i.*

Proof. Given $\boldsymbol{\beta}$ as in the above claim we can easily note that,

$$\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\| = \|\boldsymbol{y} - \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\xi}\| = \|\boldsymbol{X}\boldsymbol{\xi}\| = 0$$

The last equality follows since it has been assumed that every row of $\boldsymbol{X}$ is orthogonal to $\boldsymbol{\xi}$.

Conversely, suppose being given a $\boldsymbol{\beta}$ s.t $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\| = 0$. Then we can define $\boldsymbol{\eta} \coloneqq \boldsymbol{\beta} - \boldsymbol{X}^\dagger\boldsymbol{y}$ and repeat the same calculation as above to get that $\|\boldsymbol{X}\boldsymbol{\eta}\| = 0$. Since the zero vector is the only vector with a zero norm it follows that every coordinate of the vector $\boldsymbol{X}\boldsymbol{\eta}$ must be 0 and hence $\boldsymbol{\eta}$ is orthogonal to every row of the data matrix. This $\boldsymbol{\eta}$ satisfies all the conditions to be the $\boldsymbol{\xi}$ given in the claim statement. □

Recall that in the overparameterized regime we have, $n < d$. Hence the set of vectors orthogonal to n rows of the data matrix form a "vector space" inside $\mathbb{R}^d$ or more specifically a "vector subspace" of $\mathbb{R}^d$, the data space. (Prove!) Let's call this subspace, "Orth(Rows($\boldsymbol{X}$))". Thus we have discovered that there are uncountably infinite global minima of our objective $\hat{R}$ which can be described as $\boldsymbol{X}^\dagger\boldsymbol{y} + \text{Orth}(\text{Rows}(\boldsymbol{X}))$ - which is an instance of an "affine space" i.e a translate of a vector space.

But the solution $\boldsymbol{X}^\dagger\boldsymbol{y}$ is still special and that is because it is the global minima of our loss with the least norm -i.e the solution of the linear regression problem that lies closest to the origin in the parameter space. And now we shall prove that fact.

Claim 4.

$$\boldsymbol{X}^\dagger\boldsymbol{y} = \underset{\boldsymbol{\beta}|\hat{R}(\boldsymbol{\beta})=0}{\text{argmin}} \|\boldsymbol{\beta}\|_2$$

Proof. From the previous claim we know that if any $\boldsymbol{\beta}$ is s.t $\hat{R}(\boldsymbol{\beta}) = 0$ then we can write it as $\boldsymbol{\beta} = \boldsymbol{X}^\dagger\boldsymbol{y} + \boldsymbol{\xi}$ s.t $\boldsymbol{\xi}$ is orthogonal to each of the rows of $\boldsymbol{X}$. Now we can compute the norm of such a solution to get,

$$\begin{aligned}
\|\boldsymbol{\beta}\|^2 &= \left\|\boldsymbol{X}^\dagger\boldsymbol{y}\right\|^2 + \|\boldsymbol{\xi}\|^2 + 2\left\langle\boldsymbol{X}^\dagger\boldsymbol{y}, \boldsymbol{\xi}\right\rangle \\
&= \left\|\boldsymbol{X}^\dagger\boldsymbol{y}\right\|^2 + \|\boldsymbol{\xi}\|^2 + 2\left\langle\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{y}, \boldsymbol{\xi}\right\rangle \\
&= \left\|\boldsymbol{X}^\dagger\boldsymbol{y}\right\|^2 + \|\boldsymbol{\xi}\|^2 + 2\left\langle(\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{y}, \boldsymbol{X}\boldsymbol{\xi}\right\rangle
\end{aligned} \tag{76}$$

We can set the last term above to 0 since $\boldsymbol{X}\boldsymbol{\xi} = 0$ by our previous proof about $\boldsymbol{\xi}$ being orthogonal to the rows of the data matrix. Thus we get,

$$\|\boldsymbol{\beta}\|^2 = \left\|\boldsymbol{X}^\dagger\boldsymbol{y}\right\|^2 + \|\boldsymbol{\xi}\|^2 \geq \left\|\boldsymbol{X}^\dagger\boldsymbol{y}\right\|_2^2 \tag{77}$$

In the last step above its clear that the equality holds only when $\boldsymbol{\xi} = 0$ and thus we have proven that the 2−norm of all global minimizers of our linear regression loss $\hat{R}$ is bigger than that of $\boldsymbol{X}^\dagger\boldsymbol{y}$. □

Now lets consider doing Gradient Descent (G.D.) on $\hat{R}(\boldsymbol{\beta}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2$ starting from an initial weight β_0 and using a constant step-length η. Stated precisely we consider the following algorithm,

Algorithm 3 Linear Regression on the Loss $\hat{R}$ (Equation 75) By Constant Step-Size G.D.

1: **Input:** A starting point for the algorithm, $\boldsymbol{\beta}_0 \in \mathbb{R}^d$
2: **Input:** A step-size $\eta > 0$.
3: **for** $t = 1, \ldots$ **do**
4: $\quad \boldsymbol{\beta}_t = \boldsymbol{\beta}_{t-1} - \eta \nabla \hat{R}(\boldsymbol{\beta}_{t-1})$
5: **end for**

We note the following preliminary observation,

Lemma 9.1. *If the data matrix $\boldsymbol{X}$ is full rank, $\exists$ a continuum of choices of η s.t Algorithm 3 converges to the global minimizer of $\hat{R}$.*

The above follows from the fact that $\hat{R}$ has global minima (as shown in Claim 3), is convex and is Lipschitz smooth and hence one can invoke Theorem 3.4 in these notes, arXiv:2301.11235. Recall that in Theorem 8.1, we have seen an explicit special case of this more general result - but note that in the more general scenario of Theorem 3.4 in arXiv:2301.11235 one isn't able to get as fast a rate of convergence as was obtained in Theorem 8.1. *A more subtle fact is that one can prove that these speeds of convergence are essentially non-improvable.*

Extra Reading (Non-Assessed)

- The reader is strongly advised to understand the proof of Theorem 3.4 in arXiv:2301.11235.
- Readers who are familiar with differential equations, are encouraged to see Appendix B for an alternative view of the above referenced theorem.

Concept Check...
Prove that our loss function $\mathcal{L}$ under consideration here satisfies the conditions for the above referenced Theorem 3.4 in these notes to be invoked on it.

But we have seen that our assumptions on the data matrix being full rank and overparameterized leads us to conclude that $\hat{R}$ has multiple global minima. Hence naturally a question arises as to which one of them is found by the above algorithm. For this very controlled setting we answer this question in the following theorem - our main result in this section,

Theorem 9.2. **[(Main Result)Full-Rank Overparameterized Linear Regression by G.D. Can Be Setup s.t It Finds The Minimum 2–Norm Interpolant]** *If the data matrix $\boldsymbol{X}$ is full rank and $\boldsymbol{\beta}_0 = 0$ then $\exists$ a continuum of choices of η s.t Algorithm 3 implementing constant step-length G.D. on the overparameterized linear regression loss $\mathcal{L}$ (Equation 75), converges to the global minimizer with the minimum 2–norm i.e $\boldsymbol{X}^{\dagger}\boldsymbol{y}$.*

Proof of Theorem 9.2. Recall that we have denoted $\mathbf{x}_1^\top, \ldots, \mathbf{x}_n^\top$ as the rows of the data matrix $\boldsymbol{X}$ and each $\mathbf{x}_i \in \mathbb{R}^{d\times 1}$. Then, via induction, we shall first try to prove that any of the iterates of G.D. as given in Algorithm 3 is a linear combination of the data.

To start the induction, note that trivially we have, $\boldsymbol{\beta}_0 = 0 \in \text{Span}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. Next, suppose that, $\boldsymbol{\beta}_{t-1} \in \text{Span}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. Now recall that we had,

$$\hat{R}(\boldsymbol{\beta}) = \frac{1}{2} \cdot \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 = \frac{1}{2} \cdot \sum_{i=1}^{n} (y_i - (\boldsymbol{X}\boldsymbol{\beta})_i)^2 = \frac{1}{2} \cdot \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{d} X_{ij}\boldsymbol{\beta}_j \right)^2$$

So, we get, $\frac{\partial \hat{R}}{\partial \beta_j} = -\sum_{i=1}^{n}(y_i - \sum_{k=1}^{d} X_{ik}\beta_k) \cdot X_{ij}$. Stacking these into a column vector of d dimensions we have the derivative of the loss function w.r.t the weight vector being writable succinctly as,

$$\frac{\partial \hat{R}}{\partial \boldsymbol{\beta}} := \begin{bmatrix} \frac{\partial \hat{R}}{\partial \beta_1} \\ \vdots \\ \frac{\partial \hat{R}}{\partial \beta_d} \end{bmatrix} = -\boldsymbol{X}^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$$

Recall that the i^{th}–column of $\boldsymbol{X}^\top$ is the vector $\mathbf{x}_i$. So the matrix multiplication that is done in the above expression of $\frac{\partial \hat{R}}{\partial \boldsymbol{\beta}}$ can be re-written as the following weighted sum of vectors,

$$\frac{\partial \hat{R}}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{n}(-y_i + (\boldsymbol{X}\boldsymbol{\beta})_i) \cdot \mathbf{x}_i$$

Hence specific to $\boldsymbol{\beta}_{t-1}$, we can conclude from above that, $\frac{\partial \hat{R}}{\partial \boldsymbol{\beta}_{t-1}} \in \text{Span}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. Further, from the update rule being, $\boldsymbol{\beta}_t = \boldsymbol{\beta}_{t-1} - \eta \nabla \hat{R}(\boldsymbol{\beta}_{t-1})$ and recalling the induction hypothesis we can conclude what we set out to prove, that,

$$\boldsymbol{\beta}_t \in \text{Span}(\mathbf{x}_1, \ldots, \mathbf{x}_n), \ \forall t = 1, 2, \ldots$$

Thus we have demonstrated that the iterates of G.D. in our linear regression setup remain restricted to the linear span of the training data.

Note that for any $\hat{\boldsymbol{\beta}} \in \text{Span}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ it follows from definition that $\exists\ \boldsymbol{v} \in \mathbb{R}^n$ s.t $\hat{\boldsymbol{\beta}} = \boldsymbol{X}^\top \boldsymbol{v}$. From Claim 2 we know that our assumptions on the data imply that $\boldsymbol{X}\boldsymbol{X}^\top$ is an invertible matrix. Hence we can choose $\boldsymbol{v} = (\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{y}$ and for the corresponding $\boldsymbol{\beta}$ we have, $\hat{\boldsymbol{\beta}} = \boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{y} = \boldsymbol{X}^\dagger \boldsymbol{y} \implies \hat{R}(\hat{\boldsymbol{\beta}}) = 0$. Note that this is the same weight vector that was shown to be the minimum norm interpolant in Claim 4.

Thus the above paragraph shows the existence of a $\hat{\boldsymbol{\beta}} \in \text{Span}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ s.t $\hat{R}(\hat{\boldsymbol{\beta}}) = 0$. Now towards showing its uniqueness, suppose we have a $\hat{\boldsymbol{\beta}} \in \text{Span}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ (i.e $\exists\ \boldsymbol{v} \in \mathbb{R}^n$ s.t $\hat{\boldsymbol{\beta}} = \boldsymbol{X}^\top \boldsymbol{v}$) s.t $\hat{R}(\hat{\boldsymbol{\beta}}) = 0$. Then it would follow that $\left\|\boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}\right\| = 0$ which is equivalent to having $\boldsymbol{y} = \boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{v}$. It is easy to see that using the fact proven earlier that $\boldsymbol{X}\boldsymbol{X}^\top$ is an invertible matrix, it follows that this equation can be solved to get its unique solution, $\boldsymbol{v} = (\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{y}$.

Thus we have established that there exists a unique $\hat{\boldsymbol{\beta}}$ in the span of the data such that $\hat{R}(\hat{\boldsymbol{\beta}}) = 0$ and since 0 is the lower bound on $\mathcal{L}$ *it follows that our linear regression loss has a unique global minima in the span of its data*. Since G.D. is confined to be in this span and from Claim 9.1 we know that for a choice of step-lengths G.D. converges to its global minima, it follows that this convergence is to the minimum norm solution $\hat{\boldsymbol{\beta}}$.

□

Thus we have seen an explicit example of how having a large model size compared to the number of data leads to a multiplicity of global minima for the loss function and a surprising selectivity to creep into the G.D. algorithm as to which global minima it will converge to. This is only the tip of the iceberg of this phenomenon which gets called "implicit bias" or "algorithmic regularization". Such analysis gets extremely more involved for anything of practical interest. Almost everything remains unknown in this regard for neural nets - where this phenomenon is most pertinent and vividly visible.

Extra Reading (Non-Assessed)

The curious reader is strongly urged to read further about what are the most recent insights with regards to understanding such phenomenon with neural nets, like see this short survey of the state of this field, and the following interesting new papers, Implicit Bias in Leaky ReLU Networks Trained on High-Dimensional Data, Self-Stabilization: The Implicit Bias of Gradient Descent at the Edge of Stability, Implicit Regularization in Hierarchical Tensor Factorization and Deep Convolutional Neural Networks etc.

10 A Measure of Capacity : Rademacher Complexity

Motivation. In these last two lectures we return to the core question of machine learning of being able to theoretically bound the population risk in a given ML scenario. This lecture will involve going through a lot of setting-up with notation and formalism as we realize that naive ways of framing this target of bounding the true risk may make the question essentially unanswerable. Eventually, we will get to the somewhat sophisticated idea of "Rademacher Complexity" which we will see is a potent way to measure out-of-sample performance of a ML setup in a way that does not tie itself to any specific choice of a training algorithm. And this sets the stage for the last lecture where we will see explicit examples of how Rademacher complexity helps us understand in a precise quantitative way that the capacity of learning may not be tied to the size of the model.

Recall that in Section 2 you have seen the notions of empirical risk in Definition 11 and population risk in Definition 12. Any such risk definition depends on three key components, (a) the sampled data for the empirical risk and the data distribution for the population risk, (b) the loss function and (c) the predictor being used. And when the approximation-estimation error decomposition had to be defined in Definition 7, we had to further introduce the idea of minimizing the population risk in a specific function class $\mathcal{F}$.

In this section we will undertake a detailed study of the relationship between empirical and population risks when the predictors are being chosen from a specific function class $\mathcal{F}$ – and our focus will be on understanding how this relationship (more specifically their proximity) is affected by the chosen size of the sample n and the data distribution $\mathcal{D}$. Given any loss function, there is a very subtle interplay between n, $\mathcal{D}$ and $\mathcal{F}$ which we shall begin to slowly uncover here - in particular, we will see how the "size of the predictors" i.e number of parameters in the specification of a predictor (like the number of weights in a neural net) *may not* directly affect this relationship!

Recall that we had defined $f : \mathbb{R}^d \to \mathbb{R}^k$ as the predictor (the model at hand, say a neural net) and the loss function $\ell : \mathbb{R}^k \times \mathbb{R}^k \to [0, \infty)$ and the data $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^d \times \mathbb{R}^k$ sampled from a data distribution say $\mathcal{D}$ on $\mathbb{R}^d \times \mathbb{R}^k$. And we had defined the "population risk" of f as,

$$R(f) \coloneqq \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}} \left[\ell(\boldsymbol{y}, f(\boldsymbol{x}))\right] \tag{78}$$

It is easy to see from the above, that the above expression captures the idea of the error in prediction the model f makes, on an average, over all data that it can be expected to face. Note that the above quantity does not depend on the choice of training data, even if the f being used is instantiated as the output of a training algorithm which consumed some particular training data. Further, given a set of training data $\mathcal{S}_n = \{(\boldsymbol{x}_i, y_i), i = 1, \ldots, n\}$, each data being sampled from the distribution $\mathcal{D}$ specified above, we had denoted the empirical risk of f as,

$$\hat{R}(f, \mathcal{S}_n) \coloneqq \frac{1}{n} \sum_{i=1}^{n} \left[\ell(\boldsymbol{y}_i, f(\boldsymbol{x}_i))\right] \tag{79}$$

Looking at the two equations above, side-by-side, a discerning reader may wonder if it is true that choosing $\mathcal{S}_n$ corresponding to larger and larger n will make the quantity in equation 79 get closer and closer (and eventually converge) to the quantity in equation 78. Such an intuition although seemingly natural turns out to be exceptionally difficult to make rigorous in such generality. *As a corollary to the main point being made in this section, we will see some special families of f where one can establish this intuition.* A more general answer to this question is beyond the scope of this course.

Extra Reading (Non-Assessed)

- It is instructive to note that for certain neural nets and for carefully designed data, instances of Equation 78 can be exactly computed. See, Appendix A
- At this point the curious reader is strongly encouraged to read about the concept of Glivenko-Cantelli Class - say in Lecture 5 here.

In light of the above definitions, we realize that given training data $\mathcal{S}_n$, our usual machine learning algorithms use this training data to help search through a set of f and find one that minimizes the above quantity in equation 79. Usually in M.L., the set of f accessible to a algorithm is a parametric set of functions - like the set of neural nets at different weights on a fixed architecture. Thus, the algorithm effectively searches through the set of f by searching through the space of parameters of it - like the space of weights of a net, which is a finite dimensional vector space. *The reader is urged to carefully note that the space of all possible f that an algorithm is effectively searching through can be infinite dimensional space (and is common so) while the space of its parameters is not.*

Let us recall that we had defined $\mathcal{F}$ as the set of all predictor functions f. More specifically let us define,

$$\begin{aligned}\mathcal{F} \coloneqq &\text{ the set of all possible models(predictors) mapping } \mathbb{R}^d \to \mathbb{R}^k, \\ &\text{that a given training algorithm is searching over.}\end{aligned} \tag{80}$$

Given a fixed neural architecture, the set of all possible neural functions that can be obtained from every possible weight assignment to its edges, is an example of the set $\mathcal{F}$ that is associated to a neural training situation w.r.t a fixed architecture.

Further, we define ALG to be any chosen training algorithm which takes as input the 3–tuple $(\ell, \mathcal{S}_n, \mathcal{F})$ and searches through the above $\mathcal{F}$ for the "best" predictor - as defined in Definition 4. Thus at this point we have met all the main characters of the play i.e ALG, ℓ, $\mathcal{S}_n, \mathcal{D}$ and $\mathcal{F}$.

Towards making the above expectation from the algorithm precise, we want to capture the idea that on-an-average the algorithmically obtained model has low errors on *all possible data*. That is what is captured by wanting that $R(\text{ALG}(\ell, \mathcal{S}_n, \mathcal{F}))$ - an instance of the quantity defined in equation 78 - be as low as possible - ideally it be precisely $= \inf_{f\in\mathcal{F}} R(f)$. *Note that in computing $R(\text{ALG}(\ell, \mathcal{S}_n, \mathcal{F}))$, the expectation is not being taken over all the sources of randomness which in general can include sources of stochasticity in the training algorithm as well as the randomness in sampling $\mathcal{S}_n$ from the whole data distribution $\mathcal{D}$.*

Thus we are led to realize the following important facets of how machine learning happens,

- The eventual goal of "Machine Learning" is not just about finding f that can minimize $\hat{R}(f, \mathcal{S}_n)$ for some (possibly randomly sampled) training data $\mathcal{S}_n$. M.L. is the process of finding $\text{arginf}_{f\in\mathcal{F}} R(f)$ – which does not depend on any choice of training data.

  Just that often enough we accomplish finding a good approximation to $\text{arginf}_{f\in\mathcal{F}} R(f)$ by finding a good approximation to $\text{arginf}_{f\in\mathcal{F}} \hat{R}(f, \mathcal{S}_n)$. And the ubiquity of this "miracle" is very hard to explain - and mostly not known!

- Even before we begin to address the above mystery we need to confront the fact that since the learning algorithms can only ever at most access $\hat{R}(f, \mathcal{S}_n)$ for various choices of $f \in \mathcal{F}$, even if they manage to find $f_{erm} \coloneqq \text{arginf}_{f\in\mathcal{F}} \hat{R}(f, \mathcal{S}_n)$ (which is typically a better predictor than the $\text{ALG}(\ell, \mathcal{S}_n, \mathcal{F})$ that we actually find via the algorithm) – it is entirely unclear as to when would this f_{erm} have a low out-of-sample risk i.e low $R(f_{erm})$.

- Recall that the dataset $\mathcal{S}_n$ that the algorithm feeds on is itself randomly sampled from the full distribution over data – and thus it is a source of stochasticity in the algorithm's output. Also, the algorithm may have further sources of randomness stemming from the implementation details of the

algorithm - like having other auxiliary random variables (as in a Variational Autoencoder) or from using random approximations of the gradients of the loss/stochastic gradients.

Thus, we have no escape from having to live with the fact that "$\text{ALG}(\ell, \mathcal{S}_n, \mathcal{F})$" - the model output by the training algorithm - is itself random!

One might want to define, a notion of "cumulative risk at the population/empirical risk infimum" as, $\mathbb{E}_{\mathcal{S}_n \sim \mathcal{D}}\left[R(f_{erm})\right]$ and $\mathbb{E}_{\mathcal{S}_n \sim \mathcal{D}}\left[\hat{R}(f_{erm}, \mathcal{S}_n)\right]$ respectively. It's easy to see how to generalize this to account for all sources of randomness in the algorithm by replacing f_{erm} by $\text{ALG}(\ell, \mathcal{S}_n, \mathcal{F})$ and taking a further expectation over the sources of randomness in the algorithm. In these lectures we will not explore this idea.

10.1 Expectation Over Data of the Worst Generalization Gap & Rademacher Complexity

From the discussion so far, we have learnt that there is a vast lack of information about the obtained model in any training situation. And this lack of mathematical control largely stems from the fact that the distribution of $\text{ALG}(\ell, \mathcal{S}_n, \mathcal{F})$ is essentially unknowable for anything practically useful. So we realize that there is an *almost insurmountable challenge* in being able to control the quantity that we are really after i.e $R(\text{ALG}(\ell, \mathcal{S}_n, \mathcal{F}))$ - which we now realize is a random variable about whose distribution we can't hope to know much!

The deeply fundamental idea that takes us beyond this conundrum we find ourselves in, will be to relax our goals from wanting to control the above to want to control *only* the worst possible difference between population and empirical risk, in expectation over all possible training data $\mathcal{S}_n$. *We could also do this control in high probability over sampling $\mathcal{S}_n$ but we shall ignore that aspect in this course as that needs significantly more mathematics and it does not lead to any concept that we deem to be of immediate interest.*

Stated precisely, the data averaged worst (over all possible predictors) "generalization gap" between the population and empirical risk is,

$$\mathbb{E}_{\substack{(\boldsymbol{x}_i, \boldsymbol{y}_i) \sim \mathcal{D} \\ \forall i = 1, \ldots, n}} \left[\sup_{f \in \mathcal{F}} \left(\hat{R}(f, \mathcal{S}_n) - R(f) \right) \right]$$

It is to be noted that trying to control the above frees us from the details of any particular M.L. experiment that is trying to find $\text{arginf}_{f \in \mathcal{F}} R(f)$. Seen differently, by trying to understand the above quantity we shall become blind to the details of any algorithm and data being used in any experiment that is trying to find $\text{arginf}_{f \in \mathcal{F}} R(f)$. But this is the price we must pay to gain any insight at all into how the choice of $\mathcal{F}$ and $\mathcal{D}$ interact in our main goal towards understanding the quantity, $\text{arginf}_{f \in \mathcal{F}} R(f, \mathcal{D})$ - particularly when $\mathcal{F}$ is very complicated, like a set of deep neural nets!

Despite the loss of precision that is happening by focussing on the above quantity, we crucially note that *any condition we find on $n, \mathcal{F}$ and $\mathcal{D}$ which makes the above quantity small is a sufficient condition for the empirical and the population risk to be as much close for any predictor in the class $\mathcal{F}$.* Thus, at a fixed $\mathcal{F}$ and $\mathcal{D}$, we realize that if our algorithm is able to get very close to f_{erm} and if the data-size n used therein happens to be such that it makes the above quantity small then we know that the risk of the algorithm oputput too isn't very big - which we realize is a guarantee about the performance over unseen data for the predictor found by the training algorithm!

And such a condition which ensures proximity of empirical and population risk becomes particularly interesting if it does *not* depend on the number of parameters in the function of the set $\mathcal{F}$ - assuming these are parameterized functions like neural nets. Towards understanding such deatails we need the following notion of "Rademacher Complexity",

Definition 32 (Empirical and Average Rademacher complexity)**.** *For a scalar/real valued function class $\mathcal{H}$, suppose being given a $n-$sized data-set $\mathcal{S}_n$ of points $\{\boldsymbol{z}_i \mid i = 1, \ldots, n\}$ in the domain of the elements in $\mathcal{F}$. Let $\boldsymbol{\epsilon}$ be a $n-$dimensional random vector s.t its coordinates are sampled as $\epsilon_i \sim \pm 1$ with equal probability and independently. Then the corresponding empirical Rademacher complexity is defined as,*

$$\hat{\mathcal{R}}_n(\mathcal{H}) \coloneqq \mathbb{E}_{\boldsymbol{\epsilon}}\left[\sup_{h\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \epsilon_i h(\boldsymbol{z}_i)\right] \tag{81}$$

If the elements of $\mathcal{S}_n$ above are sampled independently and identically from a common distribution $\mathcal{D}$ then we shall denote this using the notation $\mathcal{S}_n \sim \mathcal{D}^n$. Correspondingly the "average Rademacher complexity" is defined as,

$$\mathcal{R}_n(\mathcal{H}) \coloneqq \mathbb{E}_{\mathcal{S}_n\sim\mathcal{D}^n}\left(\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sup_{h\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \epsilon_i h(\boldsymbol{z}_i)\right]\right) \tag{82}$$

Recalling the data-label pairs $(\boldsymbol{x}, \boldsymbol{y})$ from the beginning of the section we note that in the supervised setting we would typically invoke the above definition for $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$ and $h(\boldsymbol{z}) = \ell(\boldsymbol{y}, f(\boldsymbol{x}))$. In such cases, if we intend to be very explicit we shall sometimes expand the short-hand of "$\mathbb{E}_{\mathcal{S}_n\sim\mathcal{D}^n}$" to "$\mathbb{E}_{\substack{(\boldsymbol{x}_i,\boldsymbol{y}_i)\sim\mathcal{D} \\ \forall i=1,\ldots,n}}$"

10.1.1 A Foundational Property of Rademacher Complexity

And now we have all the setup to state what is possibly the most important property about Rademacher complexity,

Theorem 10.1 (**Rademacher Complexity & the Expectation of the Worst Generalization Gap**). *Suppose $\mathcal{H}$ is a set (also called a "class") of real valued functions on a common domain and let $\mathcal{S}_n$ denote $n-$sized samples being drawn from a distribution $\mathcal{D}$ on that domain. Then we have,*

$$\mathbb{E}_{\mathcal{S}_n\sim\mathcal{D}^n}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n} h(\boldsymbol{z}_i) - \mathbb{E}_{\boldsymbol{z}\sim\mathcal{D}}\left[h(\boldsymbol{z})\right]\right)\right] \leq 2\mathcal{R}_n(\mathcal{H})$$

The proof of the above theorem needs certain techniques in probability theory that the reader might not have seen earlier. The full proof is explained in detail in Section 10.1.2. In the rest of this section, we shall try to develop an understanding of how such an inequality as above reveals a whole lot of deep things about how machine learning happens.

Considering $\mathcal{H}$ to be $\ell \circ \mathcal{F}$ for some class of predictors $\mathcal{F}$, we can see that the above theorems gives a bound on the expectation (over training data) of the worst generalization error among all possible predictors – an upper bound that is dependent on the loss, the predictor class, the distribution, and the number of samples. *Being able to incorporate all these dependencies in such a succinct inequality as above is a prime reason that makes this such an attractive approach to understanding machine learning.*

Although the above interpretation needed invoking the given theorem for $\mathcal{H} = \ell \circ \mathcal{F}$, mathematically it is typically easier to directly control the Rademacher complexity of the predictor class $\mathcal{F}$. Relating the Rademacher complexity of the class $\ell \circ \mathcal{F}$ and $\mathcal{F}$ is a somewhat subtle issue. In the examples class this week, a simple example would be demonstrated where $\mathcal{R}_n(\mathcal{F})$ and $\mathcal{R}_n(\mathcal{H})$ are equal upto a constant. That example would use the following setup - suppose one is doing binary classification and any arbitrary set of n training data be given as $\boldsymbol{z}_i = (\boldsymbol{x}_i, y_i)$ with $y_i = \pm 1$ for $i = 1, \ldots, n$. Then for any $f \in \mathcal{F}$ we consider the following loss on data $(\boldsymbol{x}, y)$,

$$\ell_{0/1}((\boldsymbol{x}, y), f) = \mathbf{1}_{y\neq f(\boldsymbol{x})} = \frac{1 - yf(\boldsymbol{x})}{2}$$

Note that we shall restrict our analysis to the case when every $f \in \mathcal{F}$ is ± 1 valued (could be neural nets composed with a ± 1-valued threshold gate) – as this is the class where the set of possible output values of the predictor matches the set of possible values of the true labels and hence above loss function is sensible. Thus

given the function class of $\mathcal{F}$, we have corresponding to it a loss class $\mathcal{H}$ whose elements could be imagined to be indexed by the elements of $\mathcal{F}$ i.e,

$$\mathcal{H} \ni h_f \text{ s.t } h_f(\boldsymbol{z}) \coloneqq \ell_{0/1}((\boldsymbol{x}, y)f) \tag{83}$$

Thus corresponding to a data distribution $\mathcal{D}$ the averaged Rademacher complexity for this loss class would be,

$$\mathcal{R}_n(\mathcal{H}) = \mathbb{E}_{\substack{(\boldsymbol{x}_i, y_i)\sim\mathcal{D} \\ \forall i=1,\ldots,n}} \left[\sup_{f\in\mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \epsilon_i \cdot \ell_{0/1}((\boldsymbol{x}_i, y_i), f) \right] \tag{84}$$

And the averaged Rademacher complexity for the predictor class would be,

$$\mathcal{R}_n(\mathcal{F}) = \mathbb{E}_{\substack{(\boldsymbol{x}_i, y_i)\sim\mathcal{D} \\ \forall i=1,\ldots,n}} \left[\sup_{f\in\mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \epsilon_i \cdot f(\boldsymbol{x}_i) \right] \tag{85}$$

Note that in above the sampled labels y_i are not being used in the computation. In the examples class, the reader would work through a proof that shows that,

$$\mathcal{R}_n(\mathcal{H}) = \frac{1}{2} \cdot \mathcal{R}_n(\mathcal{F}) \tag{86}$$

We note that such simple equality relations between the Rademacher complexities of the loss class and the predictor class are not commonplace. But one of them bounding the other upto a constant is a more generic phenomenon - and explaining the reason for that is not within the scope of this course.

We shall often try to find a function $\mathcal{C}(\mathcal{F}, \mathcal{D})$ s.t we can write,

$$\mathcal{R}_n(\mathcal{H}) \leq \frac{\mathcal{C}(\mathcal{H}, \mathcal{D})}{\sqrt{n}} \tag{87}$$

The reader is urged to note that to the best of our knowledge there is no general principle as to when such an inequality as above is possible. It seems extremely fortuitous that so many of the natural use-cases in machine learning practice have an inequality of the above form being true!

Consequentially, from Theorem 10.1 we would have,

$$\mathbb{E}_{\substack{(\boldsymbol{x}_i, \boldsymbol{y}_i)\sim\mathcal{D} \\ \forall i=1,\ldots,n}} \left[\sup_{h\in\mathcal{H}} \left(\hat{R}(h, \mathcal{S}_n) - R(h) \right) \right] \leq \frac{2 \cdot \mathcal{C}(\mathcal{H}, \mathcal{D})}{\sqrt{n}} \tag{88}$$

It is clear, that in the above we have used an explicit representation of the data as appropriate in the supervised setting $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$ and invoked Theorem 10.1 on $h(\boldsymbol{z}) = \ell(\boldsymbol{y}, f(\boldsymbol{x}))$ and rewritten the LHS of Theorem 10.1 in the notation of the population and the empirical risks as introduced in the beginning.

An immediate consequence of such a bound is that we can read off from the above, that by training using $n \geq \left(\frac{2 \cdot \mathcal{C}(\mathcal{H}, \mathcal{D})}{\epsilon} \right)^2$ samples we can have the worst/maximum generalization error be at most ϵ, for any $\epsilon > 0$ arbitrarily small. One of the immediate roles of the idea of Rademacher complexity is to be able to provide

such estimates of the number of samples needed to attain a target accuracy of learning - and such estimates are often called "sample complexity".

It is to be noted that as the function class $\mathcal{F}$ get more and more complicated our sample complexity estimates obtained via the above machinery often start to wildly overestimate the number of required training data than what we see to be needed in practice. Resolving this theory vs experiment gap is a key direction at the cutting-edge of research!

When an inequality of the form as in equation 88 is obtainable *and* $\mathcal{C}$ *not depend on the number of parameters in the functions in* $\mathcal{H}$ then we can see how using a large number of parameters in our models may not hurt anything seen from this lens. In such examples, we are essentially led to realize that the effective "capacity" of the setup (independent of any n) i.e $\mathcal{C}$ is controlled by properties of the $\mathcal{H}$ and $\mathcal{D}$ which can be set s.t they do not scale with the number of parameters in $\mathcal{H}$. And this possibility shall be our motivation towards going forward with the program to understand the average of the worst generalization gap.

10.1.2 Proof of Theorem 10.1

Proof. The proof uses a technique called "symmetrization" - which has far-reaching uses than just this. Given a data set $\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n$, we imagine sampling another training data set (sometimes called the "ghost sample") $\boldsymbol{z}'_1, \boldsymbol{z}'_2, \ldots, \boldsymbol{z}'_n$ s.t as earlier, each is chosen independently of the other and from the same distribution $\mathcal{D}$. Then we can implement the following re-writing,

$$\begin{aligned}
&\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}h(\boldsymbol{z}_i)-\mathbb{E}_{\boldsymbol{z}\sim\mathcal{D}}\left[h(\boldsymbol{z})\right]\right)\\
=&\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}h(\boldsymbol{z}_i)-\mathbb{E}_{\boldsymbol{z}'_1,\boldsymbol{z}'_2,\ldots,\boldsymbol{z}'_n}\left[\frac{1}{n}\sum_{i=1}^{n}h(\boldsymbol{z}'_i)\right]\right)
\end{aligned}$$

Now the key idea is that since the first term in the sup above and the second term therein depend on two different sets of data one can just take the expectation "$\mathbb{E}_{\boldsymbol{z}'_1,\boldsymbol{z}'_2,\ldots,\boldsymbol{z}'_n}$" also over the first term without changing its value - that's now essentially a dummy operation! Thus we get,

$$\begin{aligned}
&\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}h(\boldsymbol{z}_i)-\mathbb{E}_{\boldsymbol{z}\sim\mathcal{D}}\left[h(\boldsymbol{z})\right]\right)\\
=&\sup_{h\in\mathcal{H}}\left(\mathbb{E}_{\boldsymbol{z}'_1,\boldsymbol{z}'_2,\ldots,\boldsymbol{z}'_n}\left[\frac{1}{n}\sum_{i=1}^{n}h(\boldsymbol{z}_i)-\frac{1}{n}\sum_{i=1}^{n}h(\boldsymbol{z}'_i)\right]\right)
\end{aligned}$$

Now we notice a general property w.r.t any two variables x and y, the later being random, and a bivariate function g,

$$\sup_x\left(\mathbb{E}_y[g(x,y)]\right)\leq\sup_x\left(\mathbb{E}_y[\sup_{x'}g(x',y)]\right)=\mathbb{E}_y[\sup_x g(x,y)]$$

In the last equality above we realize that the outermost sup in the middle term has become redundant since nothing inside it anymore depends on x. Thus we can just remove it and rename the x' as x.

We use the above observation to get,

$$\begin{aligned}
&\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}h(\boldsymbol{z}_i)-\mathbb{E}_{\boldsymbol{z}\sim\mathcal{D}}\left[h(\boldsymbol{z})\right]\right)\\
\leq&\mathbb{E}_{\boldsymbol{z}'_1,\boldsymbol{z}'_2,\ldots,\boldsymbol{z}'_n}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}h(\boldsymbol{z}_i)-\frac{1}{n}\sum_{i=1}^{n}h(\boldsymbol{z}'_i)\right)\right]
\end{aligned}$$

Now we take expectation over the data $\boldsymbol{z}_1,\ldots,\boldsymbol{z}_n$ to get,

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_n}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}h(\boldsymbol{z}_i)-\mathbb{E}_{\boldsymbol{z}\sim\mathcal{D}}\left[h(\boldsymbol{z})\right]\right)\right]\\
\leq&\mathbb{E}_{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_n}\left[\mathbb{E}_{\boldsymbol{z}'_1,\boldsymbol{z}'_2,\ldots,\boldsymbol{z}'_n}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}(h(\boldsymbol{z}_i)-h(\boldsymbol{z}'_i))\right)\right]\right]\\
\leq&\mathbb{E}_{\substack{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_n\\ \boldsymbol{z}'_1,\boldsymbol{z}'_2,\ldots,\boldsymbol{z}'_n}}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}(h(\boldsymbol{z}_i)-h(\boldsymbol{z}'_i))\right)\right]
\end{aligned}$$

We shall make another critical observation at this point. Since $\boldsymbol{z}_i$ and $\boldsymbol{z}'_i$ are sampled independently from the same distribution $\mathcal{D}$, they have the same set of possible values with equal likelihood (or densities to be more specific). Thus "$q(h)_i \coloneqq h(\boldsymbol{z}_i)-h(\boldsymbol{z}'_i)$" is a symmetric random variable i.e $q(h)_i$ and $-q(h)_i$ are equally likely - in an intuitive sense. Now suppose σ_i for $i=1,\ldots,n$ are independent random variables s.t each is 1 or -1 with equal probability. Then we have,

$$\mathbb{E}_{\substack{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_n\\ \boldsymbol{z}'_1,\boldsymbol{z}'_2,\ldots,\boldsymbol{z}'_n}}\left[\mathbb{E}_{\sigma_1,\ldots,\sigma_n}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}\sigma_i\cdot q(h)_i\right)\right]\right]=\frac{1}{2^n}\sum_{\sigma_1=\pm1}\cdots\sum_{\sigma_n=\pm1}\mathbb{E}_{\substack{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_n\\ \boldsymbol{z}'_1,\boldsymbol{z}'_2,\ldots,\boldsymbol{z}'_n}}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}\sigma_i\cdot q(h)_i\right)\right]$$

Given that we have realized that the density at $q(h)_i$ and $-q(h)_i$ are the same we realize that the value of each of the 2^n expectations in the R.H.S. above is the same - irrespective of the values assumed by σ_is. Thus each term in the above summand is identical irrespective of the values assumed by the σ_is. Thus the introduction of these auxiliary σ random variables is redundant and we can write,

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_n}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}h(\boldsymbol{z}_i)-\mathbb{E}_{\boldsymbol{z}\sim\mathcal{D}}\left[h(\boldsymbol{z})\right]\right)\right]\\
\leq&\mathbb{E}_{\substack{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_n\\ \boldsymbol{z}'_1,\boldsymbol{z}'_2,\ldots,\boldsymbol{z}'_n}}\left[\mathbb{E}_{\sigma_1,\ldots,\sigma_n}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}\sigma_i\cdot(h(\boldsymbol{z}_i)-h(\boldsymbol{z}'_i))\right)\right]\right]\\
\leq&\mathbb{E}_{\substack{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_n\\ \boldsymbol{z}'_1,\boldsymbol{z}'_2,\ldots,\boldsymbol{z}'_n}}\left[\mathbb{E}_{\sigma_1,\ldots,\sigma_n}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}\sigma_i\cdot h(\boldsymbol{z}_i)\right)+\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}-\sigma_i h(\boldsymbol{z}'_i)\right)\right]\right]\\
\leq&\mathbb{E}_{\substack{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_n\\ \boldsymbol{z}'_1,\boldsymbol{z}'_2,\ldots,\boldsymbol{z}'_n}}\left[\mathbb{E}_{\sigma_1,\ldots,\sigma_n}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}\sigma_i\cdot h(\boldsymbol{z}_i)\right)\right]\right]+\mathbb{E}_{\substack{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_n\\ \boldsymbol{z}'_1,\boldsymbol{z}'_2,\ldots,\boldsymbol{z}'_n}}\left[\mathbb{E}_{\sigma_1,\ldots,\sigma_n}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}-\sigma_i\cdot h(\boldsymbol{z}'_i)\right)\right]\right]\\
\leq&\mathbb{E}_{\boldsymbol{z}_1,\boldsymbol{z}_2,\ldots,\boldsymbol{z}_n}\left[\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}\epsilon_i\cdot h(\boldsymbol{z}_i)\right)\right]\right]+\mathbb{E}_{\boldsymbol{z}'_1,\boldsymbol{z}'_2,\ldots,\boldsymbol{z}'_n}\left[\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n}\epsilon_i\cdot h(\boldsymbol{z}'_i)\right)\right]\right]\\
\leq&2\cdot\mathcal{R}_n(\mathcal{H})
\end{aligned}$$

In the second inequality above we have used super-additive property of the supremum. In the third inequality above we have used linearity of expectations. In the last inequality above we have evaluated the redundant

expectations (to 1) – and we have renamed the σ_is to ϵ_is to match to the original variable naming used in the definition of the Rademacher complexity.

And thus the proof is done. □

11 Elementary Examples of Bounding Rademacher Complexity

Motivation. In this last lecture we will see explicit examples of how Rademacher complexity helps us understand in a precise quantitative way that the capacity of learning may not be tied to the size of the model. To achieve this demonstration we will start from the simple case of computing bounds on the Rademacher complexity of a finite set of real-valued functions. Then we will move to our most important case of an (uncountably infinite) set of linear predictors with bounded weight. Both these proofs will make the reader familiar with various kinds of important techniques with probability theory - which shall continue to be important when the reader sees more advanced machine learning beyond this course.

Theorem 11.1. (Massart's Finite Class Lemma) *Let $\mathcal{H}$ be a finite set of real-valued functions over a common domain and let $|\mathcal{H}|$ be the number of functions in this set. Let $\mathcal{S}_n \coloneqq \{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n\}$ be a set of n samples of data in the domain of the functions in $\mathcal{H}$. For all $h \in \mathcal{H}$, we define, $h(\mathcal{S}_n)$ as the vector of evaluations of h on the data i.e the $n-$dimensional vector, $(h(\boldsymbol{z}_1), h(\boldsymbol{z}_2), \ldots, h(\boldsymbol{z}_n))$. Suppose we define a constant M s.t $\frac{1}{\sqrt{n}}\|h(\mathcal{S}_n)\|_2 \leq M \ \ \forall h \in \mathcal{H}$. Then:*

$$\hat{\mathcal{R}}_n(\mathcal{H}) \coloneqq \mathbb{E}_{\boldsymbol{\epsilon}}\left[\sup_{h\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \epsilon_i h(\boldsymbol{z}_i)\right] \leq \sqrt{\frac{2M^2 \ln|\mathcal{H}|}{n}}.$$

Note that corresponding to the function class above, if we further denote as $\mathcal{D}$ as the distribution on the domain from which each of the data $\boldsymbol{z}_i$s are being sampled independently then we can take expectation over data sampling on both sides of the above inequality and realize that we have proven $\mathcal{R}_n(\mathcal{H}) \leq \sqrt{\frac{2M^2 \ln|\mathcal{H}|}{n}}$. Then we can invoke Theorem 10.1 to get,

$$\mathbb{E}_{\mathcal{S}_n\sim\mathcal{D}^n}\left[\sup_{f\in\mathcal{H}}\left(\frac{1}{n}\sum_{i=1}^{n} h(\boldsymbol{z}_i) - \mathbb{E}_{\boldsymbol{z}\sim\mathcal{D}}\left[h(\boldsymbol{z})\right]\right)\right] \leq 2\mathcal{R}_n(\mathcal{H}) \leq \sqrt{\frac{8\cdot M^2 \ln|\mathcal{H}|}{n}}$$

The reader is urged to note how we have rediscovered the essential insight of equation 36 as a special case of the method of Rademacher complexity. The above is a generalization bound which holds on average over sampling of data while the bound in equation 36 was proven to hold with high probability. One can use Rademacher techniques to get such high probability guarantees too but that we shall not get into this course.

We shall go over the proof of Theorem 11.1 in the examples session this week. Currently we shall try to get a feel for this bound above using an example about doing linear regression over quantized weights.

Suppose we restrict the weight vectors of the linear predictors to be searched over to be such that each coordinate can only be an integer representable by 4 bits, inclusive of the first bit which would be 0 or 1 to indicate whether the integer is positive or negative respectively. Recall that in this "2's complement" notation the smallest integer representable by 4 bits is "1000" which evaluates to $(-1)\cdot 2^3 = -8$ and the largest integer representable by 4 bits is "0111" which evaluates to $0\cdot 2^3 + 2^2 + 2^1 + 2^0 = 7$. Thus we can represent a total of 16 integers using 4 bits and hence the space of all possible $d-$dimensional weights representable using 4 bits per coordinate is 16^d. For each such "quantized" weight "$\boldsymbol{w} \in \mathbb{R}^d$" we have one linear predictor mapping $\boldsymbol{x} \mapsto \langle \boldsymbol{w}, \boldsymbol{x}\rangle$ and hence our search space consists of 16^d linear predictors.

Further, for each such quantized weight vector he have one loss function as follows, as a candidate h for Theorem 11.1 – different functions being now distinguished by their subscript,

$$\mathbb{R}^p \ni \boldsymbol{w} \mapsto h_{\boldsymbol{w}}(\boldsymbol{x}, y) = \frac{1}{2}\cdot(y - \langle \boldsymbol{w}, \boldsymbol{x}\rangle)^2 \in \mathbb{R}$$

Thus we can see that Theorem 11.1 can be invoked on the above set of hs with $\boldsymbol{z}_i = (\boldsymbol{x}_i, y_i)$, the i indexing over the n training data on which linear regression is to be done. And, the set $\mathcal{H}$, of all possible fs in this

instance is s.t $|\mathcal{H}| = 16^d$. Given that the weight vectors have bounded coordinate entries and because of the finiteness of the data it follows that a $M(d)$ as required exists and we can write for this case,

$$\hat{\mathcal{R}}_n(\mathcal{H}) \leq M(d)\sqrt{\frac{2d\ln(16)}{n}}$$

Extra Reading (Non-Assessed)
The curious reader is strongly urged to read further about the idea of model quantization - maybe one can start from seeing some of the state-of-the-art implementations of this philosophy as in here. Understanding the quantization-performance trade-offs require very deep ideas in mathematics - far beyond the scope of this course.

11.1 Rademacher Complexity for Linear Predictors

The next simplest class of predictors that is of concern in M.L. is that of linear functions. In this segment we shall get bounds on the Rademacher complexity of the set of linear functions whose weight vectors are bounded in 2−norm – which is a constraint that is often weakly implemented in the real world via regularization terms in the loss function. (And we have already seen a basic example of such a regularization in the set up of training a simple neural net – the term "λw^2" in the loss function in equation 66.)

Towards the above goal, first we need to know a simple but very useful piece of mathematics, "The Jensen's Lemma" - of which we shall now show a proof in the restricted setting that shall be of immediate use to us.

Lemma 11.2 (**Jensen's Lemma (Version 1)**). *Let $F : \mathbb{R} \to \mathbb{R}$ be a convex function and let X be a real valued random variable which takes finitely many values. Then we have,*

$$F(\mathbb{E}[X]) \leq \mathbb{E}[F(X)]$$

In Subsection 11.2 we shall give a full proof of the above statement. For an immediate intuition about what the Jensen's Lemma says, consider $F(x) = x^2$ – the F that we shall need in our upcoming use cases – and then we have the statement that, $(\mathbb{E}[X])^2 \leq \mathbb{E}[X^2]$.

Note that the specific form of the Jensen's Lemma that we shall need is stated below and its proof is essentially identical to that of the above.

Lemma 11.3 (**Jensen's Lemma (Version 2)**). *Let $f : \mathbb{R} \to \mathbb{R}$ be a convex function and let $\boldsymbol{X}$ be a $n-$dimensional vector valued random variable which takes finitely many values and let $T : \mathbb{R}^n \to \mathbb{R}$ be a given function. Then we have,*

$$F(\mathbb{E}[T(\boldsymbol{X})]) \leq \mathbb{E}[F(T(\boldsymbol{X}))]$$

Concept Check...
Prove the above lemma!

Now we have all the tools necessary to prove an interesting upper bound on the Rademacher complexity of weight-bounded linear predictors. That is, we want to bound the Rademacher complexity of the following set of functions corresponding to some arbitrary input dimension d and a weight bound B,

$$\mathcal{F} = \{\mathbb{R}^d \ni \boldsymbol{x} \mapsto \langle \boldsymbol{w}, \boldsymbol{x} \rangle \in \mathbb{R} \mid \|\boldsymbol{w}\|_2 \leq B\}$$

Also, for getting clean results for the average Rademacher complexity, we need to assume some bounds on the data distribution. We shall assume that $\mathbb{E}[\|\boldsymbol{x}\|_2^2]$ is bounded.

Now we can state the main theorem that we want to establish,

Theorem 11.4 (**Rademacher Complexity of 2-Norm Bounded Linear Predictors**). *Given any $B > 0$, let the class of linear predictors we consider be $\mathcal{F} = \{\mathbb{R}^d \ni \boldsymbol{x} \mapsto \langle \boldsymbol{w}, \boldsymbol{x} \rangle \in \mathbb{R} \mid \|\boldsymbol{w}\|_2 \leq B\}$. Then for any set of n data in $d-$dimensions, say $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$, we have,*

$$\hat{\mathcal{R}}_n(\mathcal{F}) \leq \frac{B}{n} \cdot \sqrt{\sum_{i=1}^{n} \|\boldsymbol{x}_i\|_2^2}$$

Further, if we assume that the data distribution be s.t $\exists\ C > 0$ s.t, $\mathbb{E}[\|\boldsymbol{x}\|_2^2] \leq C^2$, then it holds that,

$$\mathcal{R}_n(\mathcal{F}) \leq \frac{B \cdot C}{\sqrt{n}}$$

The reader is urged to notice the key feature of the above result - that we have a bound on the Rademacher complexity that does *not* scale with the dimension of the data d - which in this case also happens to be equal to the number of trainable parameters i.e dimension$(\boldsymbol{w})$.

The quantity B in above – the 2–norm bound on the weights of the linear functions – is the key quantity on which the Rademacher complexity of linear predictors is seen to depend. Thus, recalling Theorem 10.1 and from above it is easy to believe that when we are learning using linear functions with a fixed norm bound on their weight vectors, then irrespective of whether we are working in high or low dimensions, our ability to generalize is critically dependent on this norm bound and *not* on the number of trainable parameters.

Thus we are at the cusp of being able to see an explicit example of how the "size of the model" does not necessarily directly affect the learning properties of a function/model class. The remaining gap between this intuition and a proof is to figure out how to transfer any bound on the Rademacher complexity of the predictor class to a corresponding loss class with which the actual minimization/training would happen.

Recall that in the example session last week, we had gone through the steps of proving equation 86, which pertains to a specific predictor class and a corresponding loss class. Last lecture we had pointed out that it is outside the ambit of this module to explain how its not uncommon to be able to upper bound the Rademacher complexity of a loss class by a constant multiple of the Rademacher complexity of a predictor class. When such relationships hold then it can be imagined that results analogous to Theorem 11.4, but proven for the $\mathcal{H}$ under consideration, lead to an upper bound on the expected value of the worst case generalization error, as explained in Theorem 10.1. Thus a viable path is paved to help us conclude that for certain classes of predictors and corresponding losses their generalization error may not scale with the number of trainable parameters.

Also note, how the average Rademacher, as well as the empirical Rademacher complexity, both, vanish as the number of samples, n, goes to infinity - but they vanish at different rates. If this vanishing can be shown to continue to hold for the Rademacher complexity of a corresponding loss class too, that would be tantamount to proving that if we arbitrarily increase the number of training samples then the generalization gap closes. Further note that this $\mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$ rate for the average Rademacher complexity is extremely generic across many function classes - *it is called a "slow rate" for reasons that lie much outside the scope of this course.* As the function classes get increasingly more complicated beyond just the linear example considered here, the techniques get increasingly more involved for proving such bounds as given above - but much of the key ideas we will see in the upcoming proof do continue to remain fruitful.

Proof of Theorem 11.4. *Note that in this proof all norms shall mean 2–norms and we shall not explicitly state that every time.* From the definition of empirical Rademacher complexity it follows that,

$$\begin{aligned}\hat{\mathcal{R}}_n(\mathcal{F}) &= \mathbb{E}_{\boldsymbol{\epsilon}}\left[\sup_{\|\boldsymbol{w}\|\leq B}\frac{1}{n}\sum_{i=1}^{n}\epsilon_i\left\langle \boldsymbol{w}, \boldsymbol{x}_i\right\rangle\right] \\ &= \frac{1}{n}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sup_{\|\boldsymbol{w}\|\leq B}\left\langle \boldsymbol{w}, \sum_{i=1}^{n}\epsilon_i\boldsymbol{x}_i\right\rangle\right]\end{aligned}$$

Now we recall the Cauchy-Schwartz inequality which states that for any two finite dimensional vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ we have, $\langle \boldsymbol{a}, \boldsymbol{b}\rangle \leq \|\boldsymbol{a}\|\|\boldsymbol{b}\|$. We invoke this inequality in the argument of the sup above to get,

$$\hat{\mathcal{R}}_n(\mathcal{F}) \leq \frac{1}{n}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sup_{\|\boldsymbol{w}\|\leq B}\|\boldsymbol{w}\|\left\|\sum_{i=1}^{n}\epsilon_i\boldsymbol{x}_i\right\|\right] \leq \frac{B}{n}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\left\|\sum_{i=1}^{n}\epsilon_i\boldsymbol{x}_i\right\|\right]$$

In the last inequality above we have invoked the assumption on the norms of the weights to compute the supremum.

Now recall the relationship between inner-products and 2−norms of vectors i.e $\|\boldsymbol{a}\| = \sqrt{\langle \boldsymbol{a}, \boldsymbol{a}\rangle}$.

Thus we can write the last step as,

$$\hat{\mathcal{R}}_n(\mathcal{F}) \leq \frac{B}{n}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sqrt{\left\langle \sum_{i=1}^{n}\epsilon_i\boldsymbol{x}_i, \sum_{i=1}^{n}\epsilon_i\boldsymbol{x}_i\right\rangle}\right]$$

Now we can invoke Lemma 11.3 for the convex function $F(x) = x^2$, $\boldsymbol{X} = \boldsymbol{\epsilon}$ and $T(\boldsymbol{X}) = \langle \sum_{i=1}^{n}\epsilon_i\boldsymbol{x}_i, \sum_{i=1}^{n}\epsilon_i\boldsymbol{x}_i\rangle$ to get,

$$\left(\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sqrt{\left\langle \sum_{i=1}^{n}\epsilon_i\boldsymbol{x}_i, \sum_{i=1}^{n}\epsilon_i\boldsymbol{x}_i\right\rangle}\right]\right)^2 \leq \mathbb{E}_{\boldsymbol{\epsilon}}\left[\left\langle \sum_{i=1}^{n}\epsilon_i\boldsymbol{x}_i, \sum_{i=1}^{n}\epsilon_i\boldsymbol{x}_i\right\rangle\right]$$

Thus we have,

$$\hat{\mathcal{R}}_n(\mathcal{F}) \leq \frac{B}{n}\sqrt{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\left\langle \sum_{i=1}^{n}\epsilon_i\boldsymbol{x}_i, \sum_{i=1}^{n}\epsilon_i\boldsymbol{x}_i\right\rangle\right]}$$

Recalling that $\epsilon_i^2 = 1$, we can evaluate the inner-product inside the expectation above to get,

$$\hat{\mathcal{R}}_n(\mathcal{F}) \leq \frac{B}{n}\sqrt{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{i=1}^{n}\|\boldsymbol{x}_i\|^2 + \sum_{\substack{i,j=1 \\ i\neq j}}^{n}\epsilon_i\epsilon_j\left\langle \boldsymbol{x}_i, \boldsymbol{x}_j\right\rangle\right]} \tag{89}$$

From definition of the random variables ϵ_is it follows that they are of mean 0 and we have already assumed that they are independently sampled. So each term in the double sum above can be evaluated as,

$$\mathbb{E}\left[\epsilon_i\epsilon_j\left\langle \boldsymbol{x}_i, \boldsymbol{x}_j\right\rangle\right] = \left\langle \boldsymbol{x}_i, \boldsymbol{x}_j\right\rangle\mathbb{E}[\epsilon_i]\mathbb{E}[\epsilon_j] = 0$$

Substituting the above into equation 89 we get the intended upper bound,

$$\hat{\mathcal{R}}_n(\mathcal{F}) \leq \frac{B}{n}\sqrt{\sum_{i=1}^{n}\|\boldsymbol{x}_i\|^2}$$

Further, we can again use the same ideas as earlier to say that,

$$\left(\mathbb{E}\left[\sqrt{\sum_{i=1}^{n}\|\boldsymbol{x}_i\|^2}\right]\right)^2 \leq \mathbb{E}\left[\sum_{i=1}^{n}\|\boldsymbol{x}_i\|^2\right] = \sum_{i=1}^{n}\mathbb{E}\left[\|\boldsymbol{x}_i\|^2\right]$$

In the last equality above we have used the linearity of expectation. Then we recall that each random variable $\boldsymbol{x}_i$ is sampled from the same distribution and hence the definition of C can be invoked on each of the expectations in the sum above to get,

$$\mathbb{E}\left[\sqrt{\sum_{i=1}^{n}\|\boldsymbol{x}_i\|^2}\right] \leq \sqrt{n \cdot C^2}$$

Hence we can taken an expectation over data for the upper bound we have obtained on $\hat{\mathcal{R}}_n(\mathcal{F})$ to get,

$$\mathcal{R}_n(\mathcal{F}) \leq \frac{B \cdot C}{\sqrt{n}}$$

□

11.2 Proof of Lemma 11.2

Proof. We will prove the lemma by induction on the number of values that X can take. Suppose X takes only two values x_1 and x_2 with probabilities $p(x_1), p(x_2) \in [0,1]$ s.t $p(x_1) + p(x_2) = 1$. Then explicitly we have,

$$F(\mathbb{E}[X]) - \mathbb{E}[F(X)] = f(p(x_1)x_1 + p(x_2)x_2) - p(x_1)F(x_1) + p(x_2)F(x_2)$$

Then it immediately follows from the definition of convex functions that the above is non-positive and hence the lemma gets proven for this case.

Now suppose that the random variable X takes n values $x_1, x_2, \ldots, x_n$ with probabilities, $p(x_1), p(x_2), \ldots, p(x_n) \in [0,1]$ s.t $\sum_{i=1}^{n} p(x_i) = 1$.

Without loss of generality, we can assume that $p(x_1) \neq 1$ and assume that we have already proven the theorem for random variables taking $n-1$ values - that being the induction hypothesis. Then, in this case we have,

$$\begin{aligned}\mathbb{E}[F(X)] &= p(x_1)F(x_1) + \sum_{i=2}^{n} p(x_i)F(x_i) \\ &= p(x_1)F(x_1) + (1-p(x_1)) \cdot \sum_{i=2}^{n} \frac{p(x_i)}{1-p(x_1)}F(x_i)\end{aligned}$$

Note that $\sum_{i=2}^{n} \frac{p(x_i)}{1-p(x_1)} = 1$ and hence the weights in the second term above are themselves specifying a probability distribution. Hence we can invoke the induction hypothesis on those $n-1$ variables to get,

$$\begin{aligned}
\mathbb{E}[F(X)] &= p(x_1)F(x_1) + \sum_{i=2}^{n} p(x_i)F(x_i) \\
&= p(x_1)F(x_1) + (1-p(x_1)) \cdot \sum_{i=2}^{n} \frac{p(x_i)}{1-p(x_1)} F(x_i) \\
&\geq p(x_1)F(x_1) + (1-p(x_1)) \cdot F\left(\sum_{i=2}^{n} \frac{p(x_i)}{1-p(x_1)} \cdot x_i\right)
\end{aligned}$$

Now we can again invoke the definition of convexity at the last step above to get,

$$\begin{aligned}
\mathbb{E}[F(X)] &\geq F\left(p(x_1) \cdot x_1 + (1-p(x_1)) \cdot \sum_{i=2}^{n} \frac{p(x_i)}{1-p(x_1)} \cdot x_i\right) \\
&\geq F(\mathbb{E}[X])
\end{aligned}$$

And hence induction follows and we have proven what we had set out to prove. □

11.3 Looking Ahead With Neural Nets

A fundamental mystery with neural nets can be stated by noting the following experimental fact about them. Firstly, it is possible to fix training data and a value of depth and choose increasingly wide nets s.t all of them have been trained to have training errors below a certain low threshold (like, imagine having all their squared loss errors being less than 10^{-3} – a situation that is not very uncommon). Thus, we start from having been given a sequence of nets of increasingly large number of parameters at a fixed depth, all of which have been trained as best as possible on a fixed training data. Then it turns out that for *many* such sequences of well trained nets that can be constructed in experiments, we observe 3 properties to hold in tandem,

- The (estimated) Lipschitz constant of the trained nets will neither be very small nor be decreasing – might even be moderately increasing as the number of parameters increase.
- Such good nets can be found with as many parameters as,
  $\text{data-dimension} \times \text{number-of-training-data} = d \times n$, or even more.
- The test error would not be increasing and might even be improving as the number of parameters increase (while all of them have nearly identical and nearly zero training errors)

To the best of our knowledge, there is no mathematical formalism known till date which can explain the coexistence of these 3 *above-stated critical features in very many such experiments.* In this last section of the course we have made a modest attempt at trying to explain *some* small parts of this gargantuan mystery.

References

Amir Ali Ahmadi, Alex Olshevsky, Pablo A Parrilo, and John N Tsitsiklis. Np-hardness of deciding convexity of quartic polynomials and related problems. *Mathematical Programming*, 137:453–476, 2013.

Leo Breiman. Bagging predictors. In *Machine Learning*, pp. 123–140, 1996.

Gavin Brown and Riccardo Ali. Bias/Variance is not the same as Approximation/Estimation. *Transactions on Machine Learning Research*, 2024. URL https://TO~APPEAR-SOON.

Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pp. 1–15. Springer, 2000.

Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

Furong Huang, Jordan Ash, John Langford, and Robert Schapire. Learning deep resnet blocks sequentially using boosting theory. *arXiv:1706.04964*, 2017.

Anders Krogh, Jesper Vedelsby, et al. Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, 7:231–238, 1995.

R Tyrrell Rockafellar. Lagrange multipliers and optimality. *SIAM review*, 35(2):183–238, 1993.

P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, 2001.

Zitong Yang, Yaodong Yu, Chong You, Jacob Steinhardt, and Yi Ma. Rethinking bias-variance trade-off for generalization of neural networks. In *International Conference on Machine Learning*, pp. 10767–10777. PMLR, 2020. URL https://arxiv.org/abs/2002.11328.

Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

A Example Of An Exact Neural Risk (Non-Assessed)

For the most basic kind of depth 2 ReLU nets, consider the following risk function arising from ℓ_2-loss,

$$R(\boldsymbol{W}) \coloneqq \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\frac{1}{2}\left(y - \langle \boldsymbol{a}, \max\{0, \boldsymbol{W}\mathbf{x}\}\rangle\right)^2\right]$$

For computational tractability we shall consider the special case where, $y = \epsilon + \max\{0, \langle \boldsymbol{w}_0, \mathbf{x}\rangle\}$ and for some $\sigma_1, \sigma_2 > 0$ we have, $\epsilon \sim \mathcal{N}(0, \sigma_1)$, $\mathbf{x} \sim \mathcal{N}(0, \sigma_2 \boldsymbol{I}_d)$ and ϵ and $\mathbf{x}$ are uncorrelated. This is often said to be a "semi-agnostic" model indicating the fact that upto an additive noise the true labels are reproducible by a member of the function class, $\mathbf{x} \mapsto \langle \boldsymbol{a}, \max\{0, \boldsymbol{W}\mathbf{x}\}\rangle$, that is being trained over.

Firstly, we shall prove a lowerbound on this kind of a risk function,

Lemma A.1. *For some data distribution $\mathcal{D}_x$, let $\mathbf{x} \sim \mathcal{D}_x$, $\epsilon \sim \mathcal{N}(0, \sigma_1)$ and $y = \epsilon + \max\{0, \langle \boldsymbol{w}_0, \mathbf{x}\rangle\}$. Then,*

$$R_{\substack{\text{semi-agnostic labels}\\ \text{with Gaussian noise}}}(\boldsymbol{W}) \coloneqq \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma_1), \mathbf{x}\sim\mathcal{D}_x}\left[\frac{1}{2}\left(y - \langle \boldsymbol{a}, \max\{0, \boldsymbol{W}\mathbf{x}\}\rangle\right)^2\right] \geq \frac{\sigma_1}{2}$$

Proof. Check! □

The above lower bound argument is already independent of the data distribution and its easy to see how to write it for arbitrary label noise distributions. There are 3 structural questions that arise here immediately,

- Is there a matrix $\boldsymbol{W}$ where the above lowerbound is attained ?
  (Yes! And such $\boldsymbol{W}$ would be candidate global minima of the risk function.)
- Are there multiple values of $\boldsymbol{W}$ at which the global minima is attained?
- How well does a finite sample approximation of $\mathcal{R}(\boldsymbol{W})$ approximate $\mathcal{R}(\boldsymbol{W})$ and does it also minimize at the same $\boldsymbol{W}$?

In this short demonstration, we shall not be able to answer the above two complicated questions. But we shall use this motivating example to show that this gives an example of a non-trivial population risk function which has a closed-form expression – at least if we assume that $\mathcal{D}_x$ is a Gaussian distribution. This calculation shown here can be the starting point for developing a number of intuitions about neural nets.

Simplifying under the above-mentioned assumption we have,

$$\begin{aligned}
&\mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[\frac{1}{2}\left(y - \langle \boldsymbol{a}, \max\{0, \boldsymbol{W}\mathbf{x}\}\rangle\right)^2\right]\\
=&\mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma_2\boldsymbol{I}_d),\epsilon\sim\mathcal{N}(0,\sigma_1)}\left[\frac{1}{2}\left(\epsilon + \max\{0, \langle \boldsymbol{w}_0, \mathbf{x}\rangle\} - \langle \boldsymbol{a}, \max\{0, \boldsymbol{W}\mathbf{x}\}\rangle\right)^2\right]\\
=&\mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma_1)}\left[\frac{\epsilon^2}{2}\right] + \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma_2\boldsymbol{I}_d)}\left[\max\{0, \langle \boldsymbol{w}_0, \mathbf{x}\rangle\}^2\right] + \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma_2\boldsymbol{I}_d)}\left[\left(\sum_{i=1}^{w} a_i \max\{0, \langle \boldsymbol{w}_i, \mathbf{x}\rangle\}\right)^2\right]\\
&-\mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma_2\boldsymbol{I}_d)}\left[\max\{0, \langle \boldsymbol{w}_0, \mathbf{x}\rangle\} \cdot \left(\sum_{i=1}^{w} a_i \max\{0, \langle \boldsymbol{w}_i, \mathbf{x}\rangle\}\right)\right]\\
=&\frac{\sigma_1}{2} + \frac{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{N}(0,\sigma_2\boldsymbol{I}_d)}\left[\max\{0, \langle \boldsymbol{w}_0, \mathbf{x}\rangle\}^2\right]}{2} + \frac{1}{2}\cdot\sum_{i,j=1}^{w} a_i a_j \mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma_2\boldsymbol{I}_d)}\left[\max\{0, \langle \boldsymbol{w}_i, \mathbf{x}\rangle\} \cdot \max\{0, \langle \boldsymbol{w}_j, \mathbf{x}\rangle\}\right]\\
&-\sum_{i=1}^{w} a_i \mathbb{E}_{\mathbf{x}\sim\mathcal{N}(0,\sigma_2\boldsymbol{I}_d)}\left[\max\{0, \langle \boldsymbol{w}_0, \mathbf{x}\rangle\} \cdot \max\{0, \langle \boldsymbol{w}_i, \mathbf{x}\rangle\}\right]
\end{aligned} \tag{90}$$

For any $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{R}^n$ if $\theta \coloneqq \arccos(\boldsymbol{w}_1, \boldsymbol{w}_2)$ then as a special case of equation 3 of https://papers.nips.cc/paper/3628-kernel-methods-for-deep-learning.pdf we have,

$$\int \mathrm{d}\boldsymbol{x} \frac{e^{-\|\boldsymbol{x}\|^2}}{(2\pi)^{\frac{n}{2}}} \max\{0, \langle \boldsymbol{w}_1, \boldsymbol{x}\rangle\} \cdot \max\{0, \langle \boldsymbol{w}_2, \boldsymbol{x}\rangle\} = \frac{1}{2\pi} \cdot \|\boldsymbol{w}_1\|\|\boldsymbol{w}_2\| \cdot (\sin(\theta) + (\pi - \theta)\cos(\theta))$$

Now for some $\sigma > 0$ if we define $\boldsymbol{x} = \frac{1}{\sqrt{2\sigma}}\boldsymbol{y}$ the the above implies,

$$\begin{aligned}
&\int \mathrm{d}\boldsymbol{y} \cdot \frac{1}{(\sqrt{2})^n} \cdot \frac{e^{-\frac{\|\boldsymbol{y}\|^2}{2\sigma}}}{\sqrt{(2\pi\sigma)^n}} \cdot \frac{1}{2\sigma} \max\{0, \langle \boldsymbol{w}_1, \boldsymbol{y}\rangle\} \cdot \max\{0, \langle \boldsymbol{w}_2, \boldsymbol{y}\rangle\} = \frac{1}{2\pi} \cdot \|\boldsymbol{w}_1\|\|\boldsymbol{w}_2\| \cdot (\sin(\theta) + (\pi - \theta)\cos(\theta)) \\
\implies &\mathbb{E}_{\boldsymbol{y}\sim\mathcal{N}(0,\sigma\boldsymbol{I}_d)}\Big[\max\{0, \langle \boldsymbol{w}_1, \boldsymbol{y}\rangle\} \cdot \max\{0, \langle \boldsymbol{w}_2, \boldsymbol{y}\rangle\}\Big] = (\sqrt{2})^d \cdot \frac{\sigma}{\pi} \cdot \|\boldsymbol{w}_1\|\|\boldsymbol{w}_2\| \cdot (\sin(\theta) + (\pi - \theta)\cos(\theta))
\end{aligned} \tag{91}$$

Defining $\theta_{ij} \coloneqq \arccos(\boldsymbol{w}_i, \boldsymbol{w}_j)$ and $\theta_{0i} \coloneqq \arccos(\boldsymbol{w}_0, \boldsymbol{w}_i)$, we can substitute the above into the RHS of equation (90) to get an example of an exactly knowable neural risk function,

$$\begin{aligned}
&\mathbb{E}_{\substack{y=\epsilon+\max\{0,\langle \boldsymbol{w}_0,\mathbf{x}\rangle\} \\ \epsilon\sim\mathcal{N}(0,\sigma_1) \\ \mathbf{x}\sim\mathcal{N}(0,\sigma_2\boldsymbol{I}_d)}} \left[\frac{1}{2}\left(y - \langle \boldsymbol{a}, \max\{0, \boldsymbol{W}\mathbf{x}\}\rangle\right)^2\right] \\
=&\frac{\sigma_1}{2} + \frac{\sigma_2(\sqrt{2})^d\|\boldsymbol{w}_0\|^2}{2} + \frac{(\sqrt{2})^d\sigma_2}{2\pi} \cdot \sum_{i,j=1}^{w} a_i a_j \|\boldsymbol{w}_i\|\|\boldsymbol{w}_j\| \cdot (\sin(\theta_{ij}) + (\pi - \theta_{ij})\cos(\theta_{ij})) \\
-&\frac{(\sqrt{2})^d\sigma_2}{\pi} \sum_{i=1}^{w} a_i \|\boldsymbol{w}_0\|\|\boldsymbol{w}_i\| \cdot (\sin(\theta_{0i}) + (\pi - \theta_{0i})\cos(\theta_{0i}))
\end{aligned} \tag{92}$$

Now for intuition lets consider a special case of the above where $w = 1$ i.e there is only 1 ReLU gate. And in this case we shall rename $\boldsymbol{w}_1$ as just $\boldsymbol{w}$, θ_{0i} to just θ and the vector $\boldsymbol{a}$ will be replaced by just a scalar a to get,

$$\begin{aligned}
&\mathbb{E}_{\substack{y=\epsilon+\max\{0,\langle \boldsymbol{w}_0,\mathbf{x}\rangle\} \\ \epsilon\sim\mathcal{N}(0,\sigma_1) \\ \mathbf{x}\sim\mathcal{N}(0,\sigma_2\boldsymbol{I}_d)}} \left[\frac{1}{2}\left(y - a\max\left\{0, \boldsymbol{w}^\top\mathbf{x}\right\}\right)^2\right] \\
=&\frac{\sigma_1}{2} + \frac{\sigma_2(\sqrt{2})^d\|\boldsymbol{w}_0\|^2}{2} + \frac{(\sqrt{2})^d\sigma_2}{2\pi} \cdot a^2\|\boldsymbol{w}\|^2 \cdot \pi - \frac{(\sqrt{2})^d\sigma_2}{\pi} \cdot a\|\boldsymbol{w}_0\|\|\boldsymbol{w}\| \cdot (\sin(\theta) + (\pi - \theta)\cos(\theta))
\end{aligned} \tag{93}$$

We notice that there is a conical symmetry in this risk function : that the risk of the ReLU gate $\max\{0, \langle \boldsymbol{w}, \mathbf{x}\rangle$ remains invariant for all weights which lie on a cone about the generating weight $\boldsymbol{w}_0$ with half-angle θ and which have the same norm.

Secondly, we note that for $\boldsymbol{w} = \boldsymbol{w}_0$ we have $\theta = 0$ and further if we have $a = 1$ then in that case the above reduces to,

$$\begin{aligned}
&\mathbb{E}_{\substack{y=\epsilon+\max\{0,\langle \boldsymbol{w}_0,\mathbf{x}\rangle\} \\ \epsilon\sim\mathcal{N}(0,\sigma_1) \\ \mathbf{x}\sim\mathcal{N}(0,\sigma_2\boldsymbol{I}_d)}} \left[\frac{1}{2}\left(y - a\max\left\{0, \boldsymbol{w}^\top\mathbf{x}\right\}\right)^2\right] \\
=&\frac{\sigma_1}{2} + \frac{\sigma_2(\sqrt{2})^d\|\boldsymbol{w}_0\|^2}{2} + \frac{(\sqrt{2})^d\sigma_2}{2\pi} \cdot \|\boldsymbol{w}_0\|^2 \cdot \pi - \frac{(\sqrt{2})^d\sigma_2}{\pi} \cdot \|\boldsymbol{w}_0\|^2 \cdot \pi \\
=&\frac{\sigma_1}{2}
\end{aligned} \tag{94}$$

Recalling Lemma A.1 we realize that that the predictor $\mathbf{x} \mapsto \max\{0, \langle \boldsymbol{w}_0, \mathbf{x}\rangle\}$ is a global minima of the single gate risk function given in equation, 93. Notice that because of the label noise, the global minima of the risk is not 0.

But, is the predictor $\mathbf{x} \mapsto \max\{0, \langle \boldsymbol{w}_0, \mathbf{x}\rangle\}$ an unique global minima of the risk in equation, 93 ?

B Lyapunov Functions and Gradient Descent in Continuous Time(Non-Assessed)

We recall the notion of the gradient descent for a differentiable function as defined earlier in Algorithm 2. By staring at that for some time, it's not very hard to imagine that as being a discretization of the following "gradient flow" Ordinary Differential Equation (ODE),

$$\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} = -\eta \nabla F(\boldsymbol{w})$$

which starts from the point $\boldsymbol{w}(t = 0) = \boldsymbol{w}(0)$. In this section, we will learn of a slick technique of arguing about the convergence of such ODEs for nice F i.e determining the limit $\lim_{t\to\infty} \boldsymbol{w}(t)$ for the above ODE when this limit exists. This technique that we will see has far-reaching consequences. In particular, it often turns out that this gradient flow way of thinking is far easily tractable for the kind of things that we want to understand in research in this domain, like, determining the end points of the gradient descent algorithm when there is a multiplicity of global minima.

Theorem B.1. *For any convex and at least one differentiable F s.t $\boldsymbol{w}_* \in \operatorname{argmin} F(\boldsymbol{w})$ is well-defined and for any $\eta > 0$, it follows that an $\epsilon-$sub-optimal point i.e $\boldsymbol{w}(t)$ s.t $F(\boldsymbol{w}(t)) - \min F \leq \epsilon$ is reached if $t \geq \frac{\|\boldsymbol{w}(0)-\boldsymbol{w}_*\|^2}{2\cdot\eta\cdot\varepsilon}$.*

Proof. Let $F_* = \min F$. If $\boldsymbol{w}(t)$ is a solution to the given ODE, then corresponding to it we shall define the following function,

$$\mathcal{L}(t) \coloneqq t(F(\boldsymbol{w}(t)) - F_*) + \frac{1}{2\eta} \cdot \|\boldsymbol{w}(t) - \boldsymbol{w}_*\|^2$$

We immediately observe that this $\mathcal{L}$ is always non-negative for $t \geq 0$. And at $\boldsymbol{w}(t) = \boldsymbol{w}_*$ it attains that global minima. Further taking its time derivative (using the chain-rule of differentiation) we have,

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}t} = (F(\boldsymbol{w}(t)) - F_*) + t\left\langle \nabla F(\boldsymbol{w}(t)), \frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t}\right\rangle + \frac{1}{\eta}\left\langle \boldsymbol{w}(t) - \boldsymbol{w}_*, \frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t}\right\rangle \tag{95}$$

In the above, we now substitute the defining equation of $\boldsymbol{w}(t)$ i.e the ODE that it satisfies to get,

$$\begin{aligned}\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}t} &= (F(\boldsymbol{w}(t)) - F_*) - \eta t \cdot \|\nabla F(\boldsymbol{w}(t))\|^2 - \langle \nabla F(\boldsymbol{w}(t)), \boldsymbol{w}(t) - \boldsymbol{w}_*\rangle \\ &= (F(\boldsymbol{w}(t)) + \langle \boldsymbol{w}_* - \boldsymbol{w}(t), \nabla F(\boldsymbol{w}(t))\rangle - f_*) - \eta t \cdot \|\nabla F(\boldsymbol{w}(t))\|^2\end{aligned} \tag{96}$$

Now we invoke the property that F is convex to see that,

$$F_* \geq F(\boldsymbol{w}(t)) + \langle \boldsymbol{w}_* - \boldsymbol{w}(t), \nabla F(\boldsymbol{w}(t))\rangle \tag{97}$$

From here we can conclude that the first term on the RHS of equation 96 is non-positive, and as $-\eta t\|\nabla f(\boldsymbol{w}(t))\|^2 \leq 0$, we conclude that:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}t} \leq 0 \tag{98}$$

Hence, $\forall t > 0$ we can conclude that the following sequence of implications hold,

$$\mathcal{L}(t) \leq \mathcal{L}(0) \implies \frac{\mathcal{L}(t)}{t} \leq \frac{\mathcal{L}(0)}{t} \implies (F(\boldsymbol{w}(t)) - F_*) + \frac{1}{2\eta t}\|\boldsymbol{w}(t) - \boldsymbol{w}_*\|^2 \leq \frac{\|\boldsymbol{w}(0) - \boldsymbol{w}_*\|^2}{2\eta t} \tag{99}$$

As the term $\frac{1}{2\eta t}\|\boldsymbol{w}(t) - \boldsymbol{w}_*\|^2$ is non-negative we can weaken the last inequality above to get,

$$F(\boldsymbol{w}(t)) - F_* \leq \frac{\|\boldsymbol{w}(0) - \boldsymbol{w}_*\|^2}{2\eta t} \tag{100}$$

Given the definition of F_* it follows that, $F(\boldsymbol{w}(t)) - f_* \geq 0$ and hence combining with the above we have,

$$0 \leq F(\boldsymbol{w}(t)) - F_* \leq \frac{\|\boldsymbol{w}(0) - \boldsymbol{w}_*\|^2}{2\eta t} \tag{101}$$

Now note that as $t \to \infty$, both the sides of the above inequality tend to 0 and hence we can conclude that,

$$\lim_{t\to\infty}(F(\boldsymbol{w}(t)) - F_*) = 0 \tag{102}$$

In other words, we have shown that $F(\boldsymbol{w}(t))$ converges to the global minimum F_* as $t \to \infty$. Thus we have obtained an "asymptotic convergence" convergence result. But we aim higher - we realize that we actually have enough control here to get a "non-asymptotic" result whereby we can compute how long it takes for this gradient flow to get ε close to the global minima in function value for any $\varepsilon > 0$. That is we can aim to solve the following inequality $F(\boldsymbol{w}(t)) - F_* \leq \varepsilon$.

In light of equation 101 we realize that a sufficient condition for the above proximity in function value, is to have, $\frac{\|\boldsymbol{w}(0)-\boldsymbol{w}_*\|^2}{2\eta t} \leq \varepsilon$. This sufficient condition can be rearranged to get,

$$t \geq \frac{\|\boldsymbol{w}(0) - \boldsymbol{w}_*\|^2}{2 \cdot \eta \cdot \varepsilon} \tag{103}$$

We read the above inequality as leading us to the conclusion that $\forall \varepsilon > 0$, gradient flow is guaranteed to converge to function values at most ε more than global minimum for any differentiable convex function f with a global minimum after $\mathcal{O}\left(\frac{1}{\varepsilon}\right)$ time.

□

We note a few salient points about the above argument,

1. Note that the condition $t \geq \frac{\|\boldsymbol{w}(0)-\boldsymbol{w}_*\|^2}{2\cdot\eta\cdot\varepsilon}$ obtained above for reaching function values at most ε above the global minima is only a *sufficient* condition. Its entirely possible that for a certain function(s) within the ambit of the above proof this proximity to global minima is attained much before this. If we accept the equivalence of guarantees between gradient flow and gradient descent (at least for the simple functions we are considering here) then we can relate this possibility to what we have seen in Theorem 8.1 where the required time was $\mathcal{O}\left(\log\left(\frac{1}{\varepsilon}\right)\right)$, an exponentially lower time requirement than what the above proof would predict.

   Also note that in Theorem 8.1 the convergence was proven for the iterates to the global minima while in the above scenario the convergence was for the value of the objective function. Its a very subtle issue as to which kind of proofs would be possible under what assumptions about F.

   It's a separate fact that this ε scaling of the required time obtained above is not improvable in the worst-case – and that discussion might be considered far too advanced for this course.

2. Notice that we never had to solve the ODE to derive a fundamental property of *any* solution of it! This is reflective of the deep power that this method holds and how it opens a while new doorway into understanding such systems.

3. One might wonder as to how the function $\mathcal{L}$ used in the above proof was obtained. Fact is that there is no known systematic method of finding these functions given a particular ODE/dynamical system. A general discussion of the idea of "Lyapunov functions" (of which this is an example) can be seen in in these lecture notes of Stephen Boyd.

C A Brief Reminder of Some Background Mathematics

C.1 Algebra

C.1.1 Sums and products

We write a sum of N terms as,

$$\sum_{i=1}^{N} f(i) = f(1) + f(2) + \ldots + f(N) \ .$$

Sometimes we will write sums over a set of values which aren't in a natural sequence from 1 to N. For example, we might wish to sum up the number of each nucleotide (letter) in a DNA sequence. Call this number $n(x)$ where $x \in \{A, C, T, G\}$ is one of the letters in the DNA alphabet. We could then write the total number of nucleotides in the DNA sequence as,

$$\sum_{x \in \{A,C,T,G\}} n(x) \ = \ n(A) + n(C) + n(T) + n(G) \ .$$

In some cases it will be cumbersome to write the terms which the sum is over and we may just write,

$$\sum_{x} n(x) \ = \ n(A) + n(C) + n(T) + n(G)$$

where it is understood that the sum is over all possible values that x may take. This allows us to write general expressions where the range of the sum will depend on the context and is particularly useful when writing mathematical expressions which hold for many different cases.

In a similar way to sums of many terms we can write a product of many terms as,

$$\prod_{i=1}^{N} f(i) \quad = \quad f(1)f(2)f(3)\ldots f(N) \ .$$

C.1.2 Logarithms

A very important function is the logarithm. In general we define this function by the equation,

$$\log_a a^n = n$$

where $\log_a x$ is called the logarithm of x to base a. The most usual logarithms are base 2 ($\log_2 x$), base 10 ($\log_{10} x$) and base e ($\log_e x$, also written as $\ln x$).

All logarithms are proportional to one another, i.e. they only differ by a constant factor (e.g. $\log_2 x \propto \ln x$). In many formulas we will therefore not need to specify the base of logarithm used, since this will not usually affect the results. If one logged number is larger than another, this will remain the case when we change base.

Important formulas for manipulating powers and logarithms (to any base) are,

$$\begin{aligned}
(a^x)(a^y) &= a^{x+y} \ , \quad \frac{1}{a^x} = a^{-x} \ , \quad \frac{a^x}{a^y} = a^{x-y} \ , \\
\log(xy) &= \log x + \log y \ , \quad \log(x^n) = n \log x \ , \\
\log\left(\frac{x}{y}\right) &= \log x - \log y \ , \quad \log \prod_{i=1}^{N} f(i) = \sum_{i=1}^{N} \log f(i) \ .
\end{aligned}$$

Notice the last equation shows that a product of numbers can be converted into a sum of logarithms. Before the advent of calculators, log tables could be used to help with calculations involving the multiplication and division of large numbers. Logarithms are still useful now to help computers deal with arithmetic involving very large or very small numbers.

C.1.3 Vectors and matrices

This course deals extensively with vectors and matrices. We will mostly use bold fonts to denote vectors (lower case bold) and matrices (upper case bold).

For example, we could write a 3-dimensional vector $\boldsymbol{x}$ as,

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

where x_i is the value in the ith row of the vector. We write a 3×2 matrix $\boldsymbol{A}$ as,

$$\boldsymbol{A} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{pmatrix}$$

where A_{ij} is the value in the ith row and jth column of the matrix. Notice that a column vector is just a matrix with a single column. This is important, because it means we can multiply vectors and matrices together and we can use matrix operations like transposition on vectors.

Addition and multiplication

We can add vectors and matrices if they are the same size. We just add all the corresponding elements,

$$\begin{pmatrix} 1 & 3 \\ 4 & 5 \\ 2 & 6 \end{pmatrix} + \begin{pmatrix} 5 & 4 \\ 10 & 2 \\ 1 & 6 \end{pmatrix} = \begin{pmatrix} 6 & 7 \\ 14 & 7 \\ 3 & 12 \end{pmatrix} .$$

Two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ can be multiplied together if the number of rows in $\boldsymbol{B}$ equals the number of columns in $\boldsymbol{A}$. In this case we write,

$$\boldsymbol{AB} = \boldsymbol{C} \quad \text{where} \quad C_{ik} = \sum_{j=1}^{N} A_{ij} B_{jk}$$

where N is the number of columns in $\boldsymbol{A}$ and rows in $\boldsymbol{B}$. E.g.

$$\begin{aligned} \begin{pmatrix} 1 & 3 & 2 \\ 4 & 5 & 1 \end{pmatrix} \begin{pmatrix} 5 & 4 \\ 10 & 2 \\ 1 & 6 \end{pmatrix} &= \begin{pmatrix} (1\times 5 + 3\times 10 + 2\times 1) & (1\times 4 + 3\times 2 + 2\times 6) \\ (4\times 5 + 5\times 10 + 1\times 1) & (4\times 4 + 5\times 2 + 1\times 6) \end{pmatrix} \\ &= \begin{pmatrix} 37 & 22 \\ 71 & 32 \end{pmatrix} . \end{aligned}$$

Matrix multiplication is not commutative in general and $\boldsymbol{AB} \neq \boldsymbol{BA}$. One must therefore respect the order of multiplication, so that,

$$\begin{aligned} \boldsymbol{A}(\boldsymbol{B}+\boldsymbol{C}) &= \boldsymbol{AB} + \boldsymbol{AC} , \\ (\boldsymbol{B}+\boldsymbol{C})\boldsymbol{A} &= \boldsymbol{BA} + \boldsymbol{CA} . \end{aligned}$$

The top and bottom are not equal in general.

Identity and inverse

The identity matrix $\boldsymbol{I}$ is a square diagonal matrix (i.e. it has zeros off the diagonal) with ones on the diagonal,

$$\boldsymbol{I} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} .$$

The identity plays the role of 1 in scalar arithmetic,

$$\boldsymbol{A}\boldsymbol{I} = \boldsymbol{I}\boldsymbol{A} = \boldsymbol{A}$$

where $\mathbf{A}$ is any square matrix. If a square matrix $\boldsymbol{A}$ has an inverse then it is said to be invertible or *non-singular*,

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I} ,$$

and if $\boldsymbol{A}$ and $\boldsymbol{B}$ have inverses then,

$$(\boldsymbol{A}\boldsymbol{B})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1} .$$

A matrix is singular (non-invertible) if and only if it has a zero determinant (see below).

Transpose and dot-product

The transpose of a matrix is a matrix where the rows and columns have been switched. We put a capital T at the top right of a matrix to show that we are taking the transpose, e.g.

$$\begin{pmatrix} 1 & 3 & 2 \\ 4 & 5 & 1 \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} 1 & 4 \\ 3 & 5 \\ 2 & 1 \end{pmatrix} .$$

An important identity involves the transpose of a matrix product,

$$(\boldsymbol{A}\boldsymbol{B})^{T} = \boldsymbol{B}^{T}\boldsymbol{A}^{T} .$$

The transpose of a column vector is a row vector and from the previous section we see that it is possible to multiply together column and row vectors of the same length to get a single number (a scalar), e.g.

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \qquad \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} = x_1y_1 + x_2y_2 + x_3y_3 .$$

This particular form of multiplication is quite common and is given the name *dot-product* or *scalar product*. In general the dot-product of two length d column vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ with elements x_i and y_i is written,

$$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} = \sum_{i=1}^{d} x_i y_i .$$

The dot-product plays an important role in the definition of linear discriminant functions and neural networks.

Geometrical interpretation of vectors

It is sometimes helpful to think of vectors in geometrical terms. We can think of a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)^T$ as representing a point in a d-dimensional space which is connected to the origin of some axes. The values of each element in the vector are the displacements along each of the corresponding axes. In figure 34 we show an example of two 2-dimensional vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ with angle θ between them. One can show that the dot-product is given in geometrical terms by,

$$\boldsymbol{x} \cdot \boldsymbol{y} = |\boldsymbol{x}||\boldsymbol{y}| \cos\theta$$

where $|\boldsymbol{x}|$ is the the length (also known as the vector norm or L^2-norm) of vector $\boldsymbol{x}$, which is defined,

$$|\boldsymbol{x}| = \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}} = \sqrt{\sum_{i=1}^{d} x_i^2} \ .$$

This is often written using two bars, e.g. $\|\boldsymbol{x}\|$, to emphasize that it is the L^2-norm.

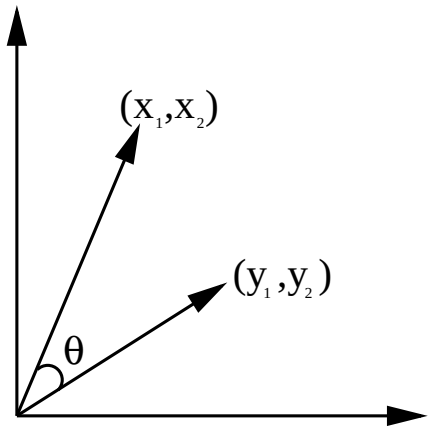


Figure 34: Vectors can be represented geometrically as arrows from the origin to a point with coordinates given by the vector elements. Here we show two 2-dimensional vectors $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{y} = (y_1, y_2)$. The *dot-product* of the two vectors is $\boldsymbol{x} \cdot \boldsymbol{y} = |\boldsymbol{x}||\boldsymbol{y}| \cos\theta$ where the length of vector $\boldsymbol{x}$ is defined $|\boldsymbol{x}| = \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}}$.

Eigenvalues and eigenvectors

An important construction in linear algebra is the eigen-decomposition of a square matrix $\boldsymbol{A}$. In the equation,

$$\boldsymbol{A}\boldsymbol{u} = \lambda \boldsymbol{u}$$

we say that $\boldsymbol{u}$ is an eigenvector of the matrix $\boldsymbol{A}$ with an associated eigenvalue (or singular value) λ. We will limit our discussion to symmetric matrices with real-valued entries, in which case the eigenvalues are also real valued.

Notice that the eigenvector can be multiplied by any constant scalar and the above equation remains true. Therefore it is common to normalise the eigenvector so that it has unit length, e.g. $\boldsymbol{u}^T\boldsymbol{u} = 1$. One can find d different (linearly independent) eigenvectors for a d-dimensional matrix,

$$\boldsymbol{A}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i \quad \text{for} \quad i = 1, 2, \ldots, d.$$

The number of non-zero eigenvalues is known as the rank of a matrix and a full-rank d-dimensional square matrix has d non-zero eigenvalues. It is useful to choose a set of orthogonal eigenvectors (a basis set),

$$\boldsymbol{u}_i \cdot \boldsymbol{u}_j = \delta_{ij} \quad \text{where} \quad \delta_{ij} = \begin{cases} 1 \text{ if } i = j \ , \\ 0 \text{ if } i \neq j \ . \end{cases}$$

Here, the function δ_{ij} is known as the Kronecker delta function or indicator function. If all the eigenvalues are distinct ($\lambda_i \neq \lambda_j \, \forall \, i \neq j$) then the eigenvectors are uniquely defined and are guaranteed to be orthogonal.

In matrix notation we can write the set of eigenvalue equations as,

$$\boldsymbol{A}\boldsymbol{U} = \boldsymbol{U}\boldsymbol{\Lambda} \tag{104}$$

where $\boldsymbol{U}$ is a matrix who's columns are the eigenvectors and the corresponding eigenvalues are in the diagonal matrix $\boldsymbol{\Lambda}$,

$$\boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{pmatrix} .$$

The orthogonality of the eigenvectors can also be expressed in matrix form,

$$\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{I}$$

and we say that $\boldsymbol{U}$ is an orthogonal matrix. For an orthogonal matrix $\boldsymbol{U}^T = \boldsymbol{U}^{-1}$ which should be obvious from above. By multiplying both sides of equation (104) by $\boldsymbol{U}^T$ on the right one obtains

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$$

which is known as the Singular Value Decomposition (SVD) of the matrix $\boldsymbol{A}$. This is the decomposition used in Principal Component Analysis (PCA) which you will meet during the course.

The covariance matrix $\boldsymbol{C}$ defined in equation (112) is an example of a symmetric matrix, because $C_{ij} = C_{ji}$. A full-rank covariance matrix is positive definite which means that it has all positive eigenvalues.

Determinant

The determinant of a square matrix can written in terms of the eigenvalues,

$$\det(\boldsymbol{A}) = |\boldsymbol{A}| = \prod_{i=1}^{d} \lambda_i \ .$$

This shows that a singular real symmetric matrix (i.e. with real-valued eigenvalues) must have at least one zero eigenvalue.

C.2 Differential calculus

Differentiation

The most fundamental quantity in differential calculus is the *derivative* of a function. The derivative of a function (of a single variable) at a point is the slope or *gradient* of the function at that point, i.e. how much y changes for a small constant increase in x. In fact it is the ratio of the change in y over the increase in x. Points which are in regions of the plot which slope up to the right are points with positive derivative (gradient). Points which are in regions of the plot which slope down to the right are points with negative derivative (gradient). Points which are in flat regions or lie between sloping regions have zero derivative (gradient).

In figure 35 we have plotted $y = f(x)$ with $f(x) = 2x - x^3$, a cubic polynomial. The gradient of the curve at $x = 1/\sqrt{3}$ is one, i.e. there is a slope of 45^o at this point. There are some equivalent notations for derivative,

$$f'(x) \quad \text{or} \quad \frac{d}{dx}f(x) \quad \text{or} \quad \frac{dy}{dx} \quad (\text{when } y = f(x)) \ .$$

The notation on the right reminds us that the gradient is the ratio of a small change in y, dy to a small change in x, dx. The notation on the left reminds us that the derivative is different at different parts of the function, so the derivative itself is a function of x.

One important use for derivatives is in determining the *stationary points* of a function. Examples of stationary points are the maxima and minima of the function. At these points the derivative vanishes. In figure 36 we show a maximum and minimum of $f(x) = 2x - x^3$ which are at $x = \pm\sqrt{2/3}$. These are actually *local* or *relative* minima and maxima because the function does take higher and lower values for other values of x. This should be contrasted with the minimum of the function $y = x^2$ at $x = 0$ which is a *global* minimum, i.e.

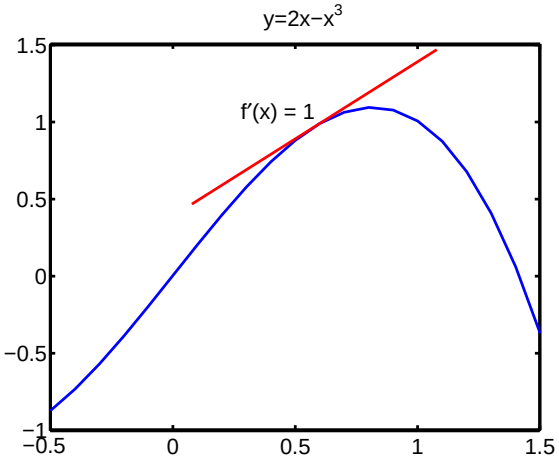


Figure 35: We plot a cubic polynomial. The gradient at the point $x = 1/\sqrt{3}$ is one ($f'(x) = 1$), corresponding to a 45^o slope to the right.

at $x = 0$ the function reaches its lowest point over the entire range of x. We can find stationary points by setting the derivative to zero and solving the corresponding equation for x.

We can calculate the derivative of any function using a small number of rules. For example,

$$\begin{aligned}
\frac{d}{dx}\left(af(x) + b\,g(x)\right) &= a\,\frac{d}{dx}f(x) + b\,\frac{d}{dx}g(x)\ , \\
\frac{d}{dx}\left(x^n\right) &= nx^{n-1}\ , \\
\frac{d}{dx}\exp(ax) &= a\exp(ax)\ , \\
\frac{d}{dx}\ln(x) &= \frac{1}{f(x)}\ ,
\end{aligned}$$

where a and b are numbers or possibly other functions which are independent of x. We can use these rules to calculate the derivative of many functions. For example, consider the polynomial plotted in figures 35 and 36, $f(x) = 2x - x^3$,

$$\frac{d}{dx}\left(2x - x^3\right) = 2\frac{d}{dx}x - \frac{d}{dx}x^3 = 2 - 3x^2$$

where we have used the top two rules above. We can now work out why the points in the figures have the derivatives shown. Setting the derivative to one we get,

$$\begin{aligned}
f'(x) &= 2 - 3x^2 = 1 \\
\rightarrow \quad x &= \pm\tfrac{1}{\sqrt{3}}\ .
\end{aligned}$$

The positive solution $x = 1/\sqrt{3}$ is the point highlighted in figure 35. A similar equation can be used to determine positions for the maximum and minimum shown in figure 36. We just set the derivative to zero in this case,

$$\begin{aligned}
f'(x) &= 2 - 3x^2 = 0 \\
\rightarrow \quad x &= \pm\sqrt{\tfrac{2}{3}}\ .
\end{aligned}$$

These are the points in figure 36. We can use the second derivative at these points to determine whether they are a minimum or a maximum,

$$f''(x) = \frac{d^2}{dx^2}\left(2x - x^3\right) = -6x$$

For $x = \sqrt{2/3}$ the second derivative is negative, indicating that this is a maximum. For $x = -\sqrt{2/3}$ we have a positive second derivative indicating that this is a minimum. These conditions should be familiar, but later we will generalise them to conditions to functions of more then one variable.

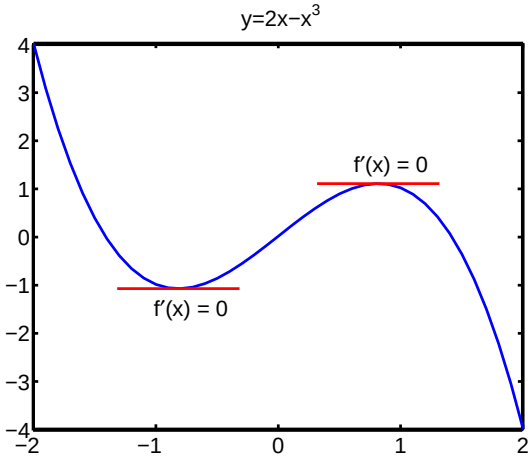


Figure 36: We plot a cubic polynomial which has a local maximum at $x = \sqrt{2/3}$ and a local minimum at $x = -\sqrt{2/3}$. The derivative is zero at these stationary points and the second derivative can be used to determine whether they are a relative maximum or a relative minimum.

Often we will want to take the derivative of a function containing another function. In this case it is useful to use the primed notation $f'(x)$ for the derivative,

$$\frac{d}{dx}g(f(x)) = f'(x)g'(f(x)) \ .$$

For example,

$$\frac{d}{dx}\exp(f(x)) = f'(x)\exp(f(x)) \quad \text{and} \quad \frac{d}{dx}\ln(f(x)) = \frac{f'(x)}{f(x)} \ .$$

Multivariate functions

We will often deal with functions of more than one variable – multivariate functions. In this case we will often use a *partial derivative* which is a derivative with respect to one variable assuming the others are constant, e.g.

$$\begin{aligned}
\frac{\partial}{\partial x}(x^2 - yxe^x) &= 2x - y(e^x + xe^x) \\
\frac{\partial}{\partial y}(x^2 - yxe^x) &= -xe^x \\
\frac{\partial^2}{\partial x \partial y}(x^2 - yxe^x) &= -(e^x + xe^x)
\end{aligned}$$

The last equation is obtained by taking the derivative of the top equation with respect to y or the middle equation with respect to x.

It is sometimes useful to think of a function of more than one variable as a function of a vector of variables $\boldsymbol{x}$ which we can write $f(\boldsymbol{x})$. The argument is a vector while the output is a scalar. Recall the geometrical interpretation of a vector in figure 34. In this two-dimensional example, one can think of the function $f(\boldsymbol{x})$ as a surface in a three-dimensional plot with the height of the third axis giving the value of the function. If this surface is smooth then it is useful to generalise the ideas from differential calculus to the multivariate setting.

The generalisation of a derivative in this context is the gradient function (a vector function) which is a vector of partial derivatives,

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f(\boldsymbol{x})}{x_1} \\ \frac{\partial f(\boldsymbol{x})}{x_2} \\ \vdots \\ \frac{\partial f(\boldsymbol{x})}{x_d} \end{pmatrix} .$$

The stationary points of a multivariate function are those for which the gradient is equal to a vector of zeros,

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} .$$

This could correspond to a maximum, minimum or a *saddle point*. Numerical optimisation methods exist to find such points – examples are gradient descent, conjugate gradient descent or quasi-newton methods; we will use these methods during the neural networks lab.

We can find out what type of stationary point we are dealing with by examining the matrix of second derivatives, which is known as the *Hessian matrix*,

$$\boldsymbol{H} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{pmatrix} .$$

The Hessian matrix plays an analogous role to the second derivative in the univariate case. The Hessian is a real symmetric matrix and therefore it has real-valued eigenvalues. The eigenvalues tell us what kind of stationary point we are at, generalising the univariate case described previously. For a relative maximum of $f(\boldsymbol{x})$ all the eigenvalues will be negative at the stationary point and the Hessian matrix is negative definite. For a relative minimum all the eigenvalues will be positive and the Hessian is positive definite. For a saddle point there will be both positive and negative eigenvalues. If any of the eigenvalues are zero then there are directions in which the function is flat (has zero curvature) and we may have to consider higher order derivatives to decide on the type of stationary point.

Lagrange multipliers

In some cases a function is constrained in some way, and we would like to find the maximum or minimum of the function $f(\boldsymbol{x})$ given a constraint $g(\boldsymbol{x}) = 0$. One can solve this problem by finding the stationary points of a function called a *Lagrangian*,

$$\mathcal{L}(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) - \lambda g(\boldsymbol{x})$$

where λ is known as a Lagrange multiplier[13]. We then solve,

$$\nabla_{\boldsymbol{x}} \mathcal{L} = 0 \ , \quad \frac{\partial \mathcal{L}}{\partial \lambda} = 0 \ .$$

The second equation just gives the constraint equation. The first equation becomes,

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) - \lambda \nabla_{\boldsymbol{x}} g(\boldsymbol{x}) \ = \ \boldsymbol{0} \ .$$

[13] not to be confused with an eigenvalue – we typically use λ for both

This equation must be solved for $\boldsymbol{x}$.

Let's consider a simple example to demonstrate how a Lagrange multiplier could be used. A packaging firm want to produce a box using the minimum amount of material (surface area) for a given volume. This type of problem is important in manufacturing, where we want to maximise utility while minimising cost. Let the sides of the box be x_1, x_2 and x_3. The volume V is fixed which provides us with the constraint $V = x_1x_2x_3$. We can write this constraint by defining the function,

$$g(\boldsymbol{x}) = x_1x_2x_3 - V \ ,$$

and writing the constraint as $g(\boldsymbol{x}) = 0$. We want to minimise the surface area which is the sum of the areas of each side,

$$f(\boldsymbol{x}) = 2(x_1x_2 + x_1x_3 + x_2x_3) \ .$$

The condition for the minimum surface area is then given by,

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) - \lambda \nabla_{\boldsymbol{x}} g(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f}{x_1} - \lambda \frac{\partial g}{x_1} \\ \frac{\partial f}{x_2} - \lambda \frac{\partial g}{x_2} \\ \frac{\partial f}{x_3} - \lambda \frac{\partial g}{x_3} \end{pmatrix} = \begin{pmatrix} 2(x_2 + x_3) - \lambda x_2 x_3 \\ 2(x_1 + x_3) - \lambda x_1 x_3 \\ 2(x_1 + x_2) - \lambda x_1 x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \ .$$

The solution to this system of equations is $x_1 = x_2 = x_3 = 4/\lambda$ (you should check this). We therefore get the intuitively obvious result that the shape of box that minimises surface area for a given volume is a cube. We can also find the value of λ from the constraint equation $g(\boldsymbol{x}) = 0$ but that is not necessary for us to determine the shape. It is often the case that the actual value of the Lagrange multipliers is not important.

Notice that there was an alternative way to cast the box problem. We could equally have maximised the volume for a fixed surface area, and that would have given us the same conclusion. There is often more than one way to solve an optimisation problem.

This idea can easily be extended to any number of linear constraints by adding a new Lagrange multiplier λ_i for each constraint $g_i(\boldsymbol{x}) = 0$. Lagrange multipliers will be an important tool used for inequality constraints and in this form they play an important role in the Support Vector Machine (SVM).

Integration

Integration is the inverse of differentiation. If $f(x)$ is a function of x with derivative $f'(x)$ then,

$$f(x) = \int f'(x)\, dx \ .$$

This is known as an *indefinite integral*. A *definite integral* is one where the limits of integration are specified. If we have a graph of a function $f(x)$ then we can integrate in the range a to b in order to calculate the area under the function in this range for a one-dimensional (univariate) function,

$$\text{Area under } f(x) \text{ between } a \text{ and } b \ = \ \int_a^b f(x) \ .$$

If the integral is over the whole range of integration then we often don't bother specifying the limits. For a multivariate function defined over the whole space we will often write,

$$\int f(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} = \int_{-\infty}^{\infty} \mathrm{d}x_1 \int_{-\infty}^{\infty} \mathrm{d}x_2 \cdots \int_{-\infty}^{\infty} \mathrm{d}x_d f(\boldsymbol{x}) \ .$$

where the range of the integrals is determined by the existence of the function $f(\boldsymbol{x})$ depending on the context. Working out integrals in a high-dimensional space is often very difficult and we usually resort to numerical or other approximate methods.

C.3 Probability theory

Much of this course will be concerned with making inferences from uncertain, or noisy, data sets. In this case the most principled way to proceed (arguably the only principled way to proceed) is by using the calculus of probability. Below we outline some of the main elements of probability theory, without too much detail or rigor. There are many textbooks available on statistics and probability theory which give a much more thorough introduction.

C.3.1 Laws of probability

Let A and B denote two events which can occur. Define the probability of seeing event A to be $P(A)$ and the probability of seeing both A and B by $P(A, B)$. Further let $P(A|B)$ denote the probability of seeing A once B is known to have occurred. The axioms of probability theory are then,

$$\begin{aligned} P(\text{not } A) + P(A) &= 1 \ , \\ P(A, B) &= P(A|B)P(B) \ . \end{aligned} \tag{105}$$

The first of these states that the probability of all possibilities will add to one (either A occurs or it doesn't). The second axiom shows how probabilities of different, non-exclusive events should be combined. Note that the second expression could equally well be written,

$$P(B, A) = P(B|A)P(A) \ ,$$

where we have simply exchanged the events A and B. Since $P(A, B) = P(B, A)$ (both are the probability of seeing A and B) we can equate the right hand sides of these equivalent expressions to get,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \ . \tag{106}$$

This is Bayes' theorem which plays a fundamental role in probabilistic inference problems. Because probabilities must add to one we know that,

$$P(A|B) + P(\text{not } A|B) = 1 \ ,$$

which gives us an expression for the denominator in Bayes' theorem,

$$P(B) = P(B|A)P(A) + P(B|\text{not } A)P(\text{not } A) \ .$$

C.3.2 Discrete random variables

Above we considered the probability of two events. In general we may be interested in a *random variable* X which maps a number to each possible outcome of an experiment involving many events. There may be a finite number of possible outcomes in which case X takes values from some finite set $\{x_1, x_2, \ldots, x_n\}$. In this case the *probability mass function* is defined $p(x) \equiv P(X = x)$ and gives the probability of each possible outcome. A simple example is the Bernoulli distribution (where $f \in [0, 1]$),

$$p(x) = \begin{cases} f & \text{if } x = 1 \\ 1 - f & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases} \tag{107}$$

For example, if $f = 1/2$ this is the appropriate distribution for describing the probability of the outcome of a fair coin toss.

The equivalent of the first axiom in equation (105) can now be written in terms of random variables as,

$$\sum_{i=1}^{n} p(x_i) \ = \ 1 \ .$$

This is called the *normalisation* condition, which ensures that probabilities add up to one. The mean (or expectation value) and variance of a discrete random variable are defined by,

$$\begin{aligned} \mu &= \sum_{i=1}^{n} p(x_i) x_i \ , \\ \sigma^2 &= \sum_{i=1}^{n} p(x_i)(x_i - \mu)^2 \ . \end{aligned}$$

Calculate the mean and variance for the Bernoulli distribution defined in equation (107).

A very useful probability distribution for discrete random variables is the binomial distribution, which is the probability of getting $X = 1$ r times out of n when sampling from the Bernoulli distribution defined in equation (107). In this case the probability mass function for r is,

$$p(r) = \frac{n!}{(n-r)!\, r!} f^r (1-f)^{n-r} \ . \tag{108}$$

The mean and variance of the binomial distribution are,

$$\begin{aligned} \mu &= nf \ , \\ \sigma^2 &= nf(1-f) \ . \end{aligned}$$

C.3.3 Continuous random variables

In some cases the random variable X maps onto a continuum and the probability of any particular value $P(X = x)$ will be zero for most probability distributions. In this case it makes more sense to define a probability density function rather than a probability mass function. A probability density function $p(x)$ is defined on a vanishingly small interval between x and $x + \mathrm{d}x$,

$$p(x) = \lim_{\mathrm{d}x \to 0} \frac{P(x < X < x + \mathrm{d}x)}{\mathrm{d}x} \ .$$

The probability density is similar to the probability mass function except that now we replace weighted sums by integrals (we use the same notation for both, since it should be clear which we mean by the context). For example, the normalisation condition above becomes,

$$\int_{-\infty}^{\infty} p(x) \mathrm{d}x = 1$$

and the definition of the mean and variance are now,

$$\begin{aligned} \mu &= \int_{-\infty}^{\infty} p(x) x \, \mathrm{d}x \ , \\ \sigma^2 &= \int_{-\infty}^{\infty} p(x)(x-\mu)^2 \, \mathrm{d}x \ . \end{aligned}$$

It should be noted that the integral will extend over the range of integration (not necessarily over the whole real line as in the above examples). A very useful probability density function is the normal distribution, which is defined in equation (109) (the multivariate generalisation is given in equation (110)).

Central limit theorem and the normal distribution

A frequency histogram will sometimes be well approximated by some standard distribution. A very useful example is the normal distribution (also known as a Gaussian distribution) which has the following density function,

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2} \right) \ , \tag{109}$$

where μ and σ^2 are the mean and variance respectively. The central limit theorem states that there are a class of random processes for which the associated frequency histograms will approach a normal distribution as the number of terms increases. Roughly speaking, these are problem in which terms contributing to the frequencies are randomly distributed according to some distribution with finite variance.

C.3.4 Multivariate data

In the previous examples each data instance is a scalar quantity, i.e. the number of sixes in a sequence of throws. In this case the data are said to be one-dimensional, since a scalar can be considered a one-dimensional vector. In contrast to this, all of the data we will deal with during the course will have dimension greater than one. In this case we will have to generalize some of the concepts introduced previously to higher dimensions. Data instances become data vectors and vectors are collected together to form the columns or rows of a matrix (see Appendix C.1.3). A particularly important multivariate probability distribution is the multivariate normal (or Gaussian) distribution. The Gaussian density function for column data vector $\boldsymbol{x}$ is given by,

$$p(\boldsymbol{x}) = \frac{\mathrm{e}^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{C}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}}{\sqrt{2\pi\det(\boldsymbol{C})}} \tag{110}$$

where the distribution has mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{C}$ defined by,

$$\boldsymbol{\mu} = \int p(\boldsymbol{x})\,\boldsymbol{x}\,\mathrm{d}\boldsymbol{x}\ , \tag{111}$$

$$\boldsymbol{C} = \int p(\boldsymbol{x})\,(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\,\mathrm{d}\boldsymbol{x}\ , \tag{112}$$

where the integral over the vector $\boldsymbol{x}$ is interpreted as,

$$\int f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \int_{-\infty}^{\infty}\mathrm{d}x_1\int_{-\infty}^{\infty}\mathrm{d}x_2\cdots\int_{-\infty}^{\infty}\mathrm{d}x_N f(\boldsymbol{x})\ .$$

>From the above expressions we see that $\boldsymbol{C}$ should be symmetric and invertible (non-singular). We will sometimes use the shorthand notation $\boldsymbol{x}\sim\mathcal{N}(\boldsymbol{\mu},\boldsymbol{C})$.

For formulas involving vectors and matrices we should think of N-dimensional column vectors (like $\boldsymbol{x}$) as $N\times 1$ matrices. Recall that if matrix $\boldsymbol{A}$ is $N\times K$ while matrix $\boldsymbol{B}$ is $K\times M$ then $\boldsymbol{AB}$ is $N\times M$. The transpose $\boldsymbol{A}^{\mathrm{T}}$ has columns of $\boldsymbol{A}$ as rows and is $K\times N$. The same rules hold for the N-dimensional column vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. For example,

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} = \sum_{i=1}^{N} x_i y_i = \boldsymbol{x}\cdot\boldsymbol{y} \quad \text{the dot or inner product which is } 1\times 1 \text{ (scalar)}\ , \tag{113}$$

$$\boldsymbol{x}\boldsymbol{y}^{\mathrm{T}} = \boldsymbol{Z} \quad \text{with} \quad Z_{ij} = x_i y_j \quad \text{the outer product which is } N\times N \text{ (matrix)}\ . \tag{114}$$

Similarly, the term in the exponent of equation (110) is a scalar,

$$(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{C}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) = \sum_{i=1}^{N}\sum_{j=1}^{N}(x_i-\mu_i)C_{ij}^{-1}(x_j-\mu_j)\ ,$$

while the components of the covariance matrix defined in equation (112) are written,

$$C_{ij} = \int p(\boldsymbol{x})\,(x_i-\mu_i)(x_j-\mu_j)\,\mathrm{d}\boldsymbol{x}\ .$$

The covariance matrix gives the correlation of the various dimensions in the data. If two components x_i and x_j are positively correlated then $C_{ij}>0$. If they are anti-correlated then $C_{ij}<0$.