Lecture 1: Introduction, Course Overview, and Preliminaries

Adityanarayanan Radhakrishnan and George Stefanakis

Edited by: Max Ruiz Luyten and Cathy Cai

January 8, 2023

1 Introduction

Over the past decade, interest in machine learning research has spiked dramatically, with advancements in deep learning being a significant driving force. Indeed, deep learning has transformed many areas in computer science including computer vision [7, 11], natural language processing [3], and reinforcement learning [13]. Unfortunately, given the rapid pace of progress in deep learning, a newcomer looking for a simple set of guiding principles for building machine learning applications can be easily overwhelmed by the nuances of training deep networks.

In particular, the flexibility provided by the vast number of choices of network hyper-parameters and architectures in deep learning can be a double-edged sword. When trying to construct a neural network from scratch for any given task, the newcomer may naturally ask (1) how many layers to use or (2) what type of layers to use. There remains no clear, simple answer to these questions, and suggestions for these choices seem to be governed by rapidly evolving results from empirical successes. Prominent examples appear in the domain of image classification: (1) after iterating through several state-of-the-art models over the past 5 years [7, 8, 21, 25], the impact of network architecture on performance seems to remain unclear; (2) recent experiments question whether the ubiquitous convolutional layer is even necessary for producing effective image classifiers [4, 22].

On the other hand, classical kernel-based machine learning methods (e.g. support vector machines (SVM) or kernel regression) while conceptually simple and easy to implement, appear to lack both the flexibility and empirical successes of deep neural networks. Indeed, there is no simple method for constructing a kernel function based on the application domain, and classical approaches often involved applying standard kernels (e.g. Gaussian kernels) to features crafted from the data. For example, while SVM approaches based on hand-crafted features [20] were the state-of-the-art approach for achieving competitive performance on ImageNet [18], kernel methods have since then been outperformed by neural networks in these tasks since AlexNet in 2012 [11]. If kernel methods had the flexibility and effectiveness of neural networks, then given their simplicity, these models would offer a newcomer-friendly alternative to neural networks.

We begin this course by utilizing the recently-discovered connection between neural networks and kernel methods given by the Neural Tangent Kernel (NTK) [9] to present a class of kernel methods that are simple, flexible, and competitive in practice. At a high level, the NTK serves as a tool for computing a kernel corresponding to nearly any neural network architecture, and importantly, solving kernel regression with the NTK is equivalent to training the corresponding neural network provided the layer-wise widths are sufficiently large (e.g. approach infinity). Below, we highlight some recent work demonstrating the effectiveness of this approach.

Example 1 (NTK Regression and Classification)**.** *Recent work [2] demonstrated that using SVMs with the NTK corresponding to ReLU networks with at most 5 hidden layers generally outperformed random forests, SVMs with the Gaussian kernel, and finite width neural networks on 90 tasks from the UCI database [6]. This work also demonstrated that the NTK for convolutional networks, i.e. the CNTK, outperformed convolutional networks (including ResNets) on datasets with at most 640 images.*

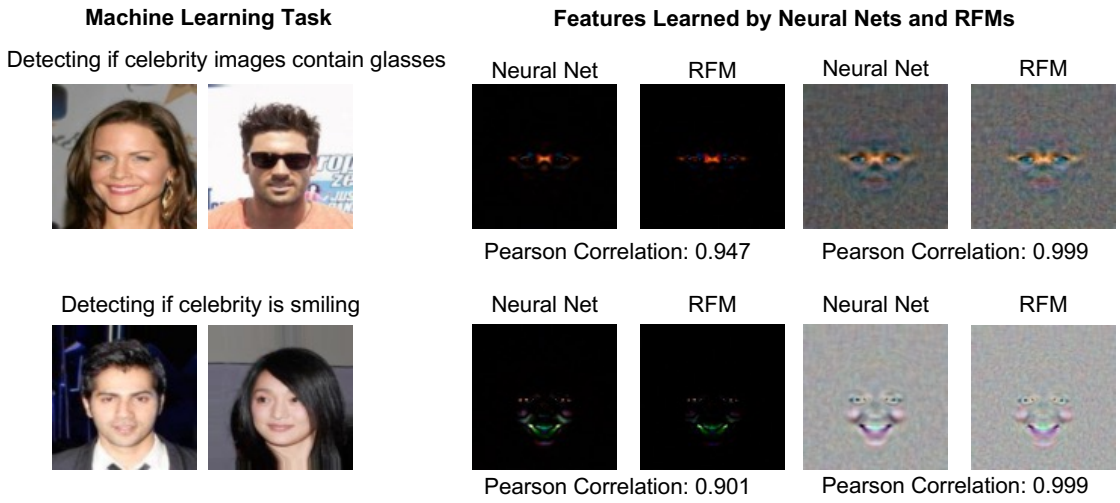

Figure 1: Examples of feature learning in neural networks and recursive feature machines (RFMs). When trained to classify whether celebrity images containing glasses or a smiling celebrity, both models learn features associated with the prediction task. Moreover, both models identify nearly identical features (Pearson correlations greater than 0.9.

Example 2 (CNTK for Image Classification)**.** *The work [12] presents a thorough empirical comparison between the performance of the CNTK and corresponding finite width convolutional networks on the CIFAR10 image classification task [10]. This work generally demonstrated that the CNTK is competitive with training convolutional networks. In particular, the CNTK corresponding to a convolutional ReLU network with a fully connected last layer consistently outperforms the corresponding finite width network, and the CNTK for a convolutional ReLU network with global average pooling [1] is roughly* 3% *worse than the corresponding finite width network (*77% *vs.* 80.61% *test accuracy in Supplementary Table S1).*

Example 3 (NTK/CNTK for Matrix/Image Completion)**.** *Lastly, the work [17] demonstrates the effectiveness of the NTK and CNTK for matrix and image completion tasks. In particular, this work demonstrates that the NTK outperforms prior methods for virtual drug screening, and that the CNTK generally outperforms modern neural network architectures [23] for image inpainting while also having a run-time advantage.*

Given its simplicity and effectiveness, the NTK serves as a powerful alternative to neural networks in several applications. Yet, as shown in recent work [5, 19], there are certain classes of problems for which neural networks have a significant advantage over NTKs. What are key properties of these problems that lead to a gap between neural networks and NTKs, and what is the mechanism driving the success of neural networks on these problems?

Feature learning in neural networks. A mechanism of particular interest is the one by which neural networks "learn features." Indeed, ability of neural networks to learn features from data is thought to be a central contributor to their success [19, 24]. While we defer a mathematical definition of feature learning, we use this terminology to refer to the ability of neural networks to identify and reduce to features most relevant for prediction. As illustrative examples, when classifying whether a celebrity has a glasses or is smiling, fully connected neural networks subset to the region of pixels corresponding to these features (See Fig. 1) .

Recursive Feature Machines (RFMs). Toward the end of this course, we will discuss recent work that identifies a mechanism driving feature learning in neural networks [16]. In particular, we will demonstrate that feature learning in neural networks is connected to a statistical estimator known as the expected gradient outer product. We will subsequently leverage this insight to develop *Recursive Feature Machines* (RFMs), which are kernel methods that learn features. We will show that RFMs are simple, interpretable, and effective machine learning models. In particular, we demonstrate that:

1. Features learned by RFMs accurately capture those learned by fully connected neural networks.
2. RFMs produce state-of-the-art performance on tabular datasets.
3. RFMs shed light on remarkable deep learning phenomena, including identifying spurious features and biases learned by neural networks.

General Course Remarks. While the goal of the course is to demonstrate that the NTK and RFMs are simple and easy-to-use methods in practice, we will simultaneously present a full theoretical development of these methods starting from the basics of linear and kernel regression.

2 Course Overview

Below, we breakdown the course with an overview of the content in each lecture. We note that each lecture will have a theoretical component followed by empirical examples to make the theory concrete.

Lecture 2: Linear Regression. We begin the course with a review of linear regression, i.e. finding a line of best fit to data. The ideas from linear regression lay the foundation for kernel regression and will be crucial to understanding why kernel regression with the NTK is a conceptually simple and easy to use method. In particular, we will cover key aspects of the linear regression framework including a derivation of the fact that gradient descent leads to a minimum norm solution for linear regression given by the Moore-Penrose pseudoinverse.

Lecture 3: Kernel Regression. We extend the linear regression framework to kernel regression by performing linear regression after applying a nonlinear transformation, i.e. a feature map, to the data. We demonstrate that kernel regression is tractable even when the feature map has an infinite dimensional range (corresponding to a Hilbert space). In particular, we review key results including the Representer theorem and the kernel trick that enable kernel regression to be easily implemented. We lastly review and present examples of kernel regression with popular kernels including the Gaussian and Laplace kernels.

Lecture 4: Neural Networks. We will provide a brief introduction to neural networks. While this is a vast area, we will cover the following fundamental aspects of neural networks: (1) Basic architectures (fully connected and convolutional networks) ; (2) network width and depth ; (3) layer initialization schemes ; (4) training methodology (gradient descent, stochastic gradient descent). Given the complexity of modeling and training choices, we will provide a short checklist of guiding principles for training neural networks. We will also provide some code in PyTorch [15] to familiarize the reader with a typical workflow for training neural networks.

Lecture 5: NNGP, Dual Activations, and Over-parameterization. After reviewing the fundamentals of kernel regression and neural networks, we connect the kernel regression framework to infinitely wide neural networks where only the last layer is trained. In particular, under certain conditions on the initialization, training the last layer of an infinite width network corresponds to solving kernel regression with the Neural Network Gaussian Process (NNGP). We present analytical tools to compute the NNGP in practice and utilize the theory of dual activations to simplify the form of the NNGP. We then provide a first example of the double descent phenomenon in machine learning. Namely, we show that increasing network width when training only the last layer leads to improved generalization with the infinite width limit given by the NNGP yielding the best performance.

Lecture 6: NTK Origin and Derivation. With the tools from previous lectures in hand, we finally present the main connection between training all layers of an infinitely wide neural network and solving kernel regression the Neural Tangent Kernel (NTK). We first show that the NTK arises as a linearization of a neural network around its initial weights, and this approximation becomes increasingly accurate as network width increases. We then present a simple argument demonstrating that this approximation remains accurate throughout training, which implies that solving kernel regression with the NTK is remarkably equivalent

to training an infinitely wide neural network. Utilizing the tools for computing the NNGP, we compute a closed for the NTK in terms of dual activation functions. As we did for the NNGP, we then lastly present empirical examples highlighting the double descent phenomenon by comparing the performance of networks of increasing width to that of the NTK.

Advanced Topics (Subject to change based on student interest).

Lecture 7: NTK of Deep Neural Networks and the CNTK. Thus far, we have presented tools to compute the NTK/NNGP for fully connected networks with 1 hidden layer. We now present a derivation of the NTK/NNGP for networks of arbitrary depth. In particular, we show that the NTK/NNGP for a network of depth L can be written recursively in terms of the NTK/NNGP for a network of depth $L-1$ and are thus simple to compute in practice. In order to showcase the flexibility of the NTK framework, we then present a simple example of computing the NTK for a 1 hidden layer convolutional network, i.e. the CNTK. We highlight the impact of changing the last layer from a fully connected layer to a global average pooling layer, and indicate why the latter is more expensive to compute. We lastly present experiments utilizing the Neural Tangents library [14] demonstrating the effectiveness of the NTK/CNTK in practice.

Lecture 8: NTK and CNTK for Matrix Completion. We will review the contents of [17], which derives the NTK and CNTK for matrix completion problems. This work provides two applications of these derivations: (1) virtual drug screening and (2) image inpainting. This work demonstrates that the NTK is state-of-the-art for virtual drug screening and that the CNTK generally outperforms a variety of convolutional networks used for image inpainting.

Lecture 9: Recursive Feature Machines (RFMs). We will review the contents of [16]. This work identifies a mechanism driving feature learning in neural networks and subsequently uses this mechanism to develop RFMs, which are kernel methods that learn features. This work demonstrates that (1) RFMs accurately caputre features learned by fully connected neural networks ; (2) RFMs produce state-of-the-art-results on tabular datasets ; and (3) RFMs shed light on a variety of remarkable deep learning phenomena.

3 Mathematical Background and Review

We briefly review of mathematical background that will be helpful in understanding core theoretical concepts of the course. While the methods covered in the course can be coded up and used with minimal knowledge of the material below, a better understanding of the theoretical concepts serves especially useful for debugging, should any code happen to go awry.

The general topics that will be covered span linear algebra, analysis, probability, and statistics. We begin with mathematical notation that will be used commonly throughout the course. We will place blue text to indicate the lectures where the material will be used.

3.1 Notation

In this course, we abide by the following notation:

- $\mathbb{Z}$: the space of integers. $\mathbb{Z}_+$: the space of positive real numbers.
- $\mathbb{R}$: the space of real numbers. $\mathbb{R}_+$: the space of positive real numbers.
- $\mathbb{C}$: the space of complex numbers
- $\mathbb{R}^d$: the space of d-dimensional real-valued (column) vectors
- $\mathcal{S}^{d-1}$: the unit sphere in d dimensions (i.e. $\mathcal{S}^{d-1} = \{x \in \mathbb{R}^d \ ; \ \|x\|_2 = 1\}$)
- $\mathbb{R}^{m \times n}$: the space of real-valued $m \times n$ matrices
- If $A \in \mathbb{R}^{m \times n}$, we let $A_{i,j}$ denote the i, j entry of A. We let $A_{i,:} \in \mathbb{R}^{1 \times n}$ denote row i of A. Similarly, $A_{:,j} \in \mathbb{R}^m$ is column j of A.
- Given $A \in \mathbb{R}^{m \times n}$, the matrix $A^T \in \mathbb{R}^{n \times m}$ is the transpose of A, with $A^T_{i,j} = A_{j,i}$.
- To iterate through a matrix in *row-major order* is to observe consecutive entries from left to right, reading an entire row before proceeding to the next.
- $\mathbb{R}^m \times \mathbb{R}^n = \{(x, y) \ ; \ x \in \mathbb{R}^m, y \in \mathbb{R}^n\}$
- $f : \mathbb{R}^m \to \mathbb{R}^n$ denotes a function f from $\mathbb{R}^m$ to $\mathbb{R}^n$
- A function $\phi : \mathbb{R} \to \mathbb{R}$ acts element-wise on $x \in \mathbb{R}^d$ when:

$$\phi(x) = \begin{bmatrix} \phi(x_1) \\ \phi(x_2) \\ \vdots \\ \phi(x_d) \end{bmatrix}$$

- Given $f : \mathbb{R}^d \to \mathbb{R}$, we let $\nabla f : \mathbb{R}^d \to \mathbb{R}^d$ and $H_f \in \mathbb{R}^d \to \mathbb{R}^{d \times d}$ denote the gradient and Hessian of f respectively, where:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix} \quad ; \quad H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix}$$

3.2 Linear Algebra

3.2.1 Orthogonal Matrices

Orthogonal matrices are particularly important in the analyses in this course. They will arise in Eigendecompositions and Singular Value Decompositions, which will be used to simplify the analysis of linear and kernel regression (Lectures 2 and 3).

Definition 1. *An **orthonormal basis**, $\{v_1, ..., v_n\}$ of a vector space with inner product $\langle v_i, v_j \rangle$, satisfies:*

$$\langle v_i, v_j \rangle = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}.$$

Namely, any pair of vectors in the basis must be perpendicular (have inner product 0), and every vector in the space must be a unit vector.

Definition 2. *A matrix $A \in \mathbb{R}^{n \times n}$ is **orthogonal** if its rows and columns form an orthonormal basis.*

Definition 1 implies that for a real orthogonal matrix A: $A^T A = A A^T = I$. Hence, $A^T = A^{-1}$.

3.2.2 Rank, Eigendecomposition, Singular Value Decomposition

The Eigendecomposition and Singular Value Decomposition (SVD) are matrix decompositions commonly used in machine learning and statistics. We will utilize this decomposition when analyzing convergence of linear regression (Lecture 2) and when visualizing top eigenvectors of feature matrices (Lecture 9).

Definition 3. *The **rank** of a matrix is the dimension of the space spanned by its rows (or columns).*

Definition 4. *A vector $\mathbf{v}$ is an **eigenvector** of matrix $\mathbf{A}$, with **eigenvalue** λ, if*

$$\mathbf{A}\mathbf{v} = \lambda \mathbf{v}.$$

Two useful facts about eigenvalues are that the trace of a matrix corresponds to the sum of its eigenvalues, and the determinant corresponds to the product of the eigenvalues.

For square matrices, there exists a specific decomposition called the *eigendecomposition*, which is a factorization of the matrix into the product of three matrices containing its corresponding eigenvalues and eigenvectors as follows.

Definition 5. *The **eigendecomposition** of $\mathbf{A} \in \mathbb{C}^{n \times n}$, with eigenvectors v_i and corresponding eigenvalues λ_i, for $i \in [n]$, is given by:*

$$\mathbf{A} = \begin{bmatrix} | & | & \dots & | \\ v_1 & v_2 & \dots & v_n \\ | & | & \dots & | \end{bmatrix} \begin{bmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_n \end{bmatrix} \begin{bmatrix} | & | & \dots & | \\ v_1 & v_2 & \dots & v_n \\ | & | & \dots & | \end{bmatrix}^{-1} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}.$$

Definition 6. *The **singular value decomposition (SVD)** of a matrix is the factorization of a real matrix, $\mathbf{A} \in \mathbb{R}^{m \times n}$ into a product of three matrices, $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$, such that:*

- *$\mathbf{U} \in \mathbb{R}^{m \times r}$ has columns forming an orthonormal basis for the column space of $\mathbf{A}$.*
- *$\mathbf{\Sigma} \in \mathbb{R}^{r \times r}$ is a diagonal matrix whose diagonal elements are known as the singular values of $\mathbf{A}$.*
- *$\mathbf{V} \in \mathbb{R}^{r \times n}$ has columns forming an orthonormal basis for the row space of $\mathbf{A}$.*

Note that r corresponds to the rank of the matrix A. Then, we have:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}.$$

From the above, it follows that the rank of a matrix can also be defined as the number of nonzero singular values. We let $\sigma_i(A)$ be the i^{th} singular value of A for $i \in \{1, 2, \ldots, r\}$.

Key Properties of SVD: Note that the SVD exists for any real matrix A, unlike the eigendcomposition. As U, V are orthonormal matrices, we have $UU^T = I_{m\times m}$ and $V^TV = I_{n\times n}$. Hence, we have $AA^T = U\Sigma^2U^T$ and $A^TA = V\Sigma^2V^T$. In particular, these are also respectively the eigendecompositions of AA^T and A^TA, and so we conclude $\sigma_i(A)^2 = \lambda_i$. Naturally, given a matrix A, we can recover its left and right singular matrices U, V by computing the eigendecomposition of AA^T and A^TA.

3.2.3 Moore-Penrose Inverse

The Moore-Penrose inverse, often called pseudoinverse, is a generalization of the matrix inverse which can be applied to non-square matrices. Notably, while the regular inverse only exists under certain conditions, the pseudoinverse always uniquely exists for matrix, A. We will use the Moore-Penrose pseudoinverse to solve linear regression when the number of samples is not equal to the dimension of the data (See Lecture 2).

Definition 7. *The **Moore-Penrose Inverse** of a matrix $A \in \mathbb{R}^{m\times n}$, denoted $A^\dagger$, satisfies:*

- $AA^\dagger A = A$
- $A^\dagger AA^\dagger = A^\dagger$
- $(AA^\dagger)^T = AA^\dagger$
- $(A^\dagger A)^T = A^\dagger A$

The Moore-Penrose pseudoinverse is often convenient to write in terms of the SVD of a matrix.

Lemma 1. *Let $A \in \mathbb{R}^{m\times n}$ and let $A = U\Sigma V^T$ given by the SVD. Then, $A^\dagger = V\Sigma^\dagger U^T$.*

We leave it to the reader that $V\Sigma^\dagger U^T$ satisfies the conditions of the pseudoinverse above. There are three key special cases, for which a closed form of the pseudoinverse is easy to write out.

- $m = n$ and A is full-rank:

$$A^\dagger = A^{-1}.$$

- $m < n$ and rows of A are linearly independent:

$$A^\dagger = A^T(AA^T)^{-1}.$$

- $m > n$ and columns of A are linearly independent:

$$A^\dagger = (A^TA)^{-1}A^T.$$

3.2.4 Projection and Orthogonal Decomposition

A projection in linear algebra corresponds to the notion of restricting the representation of a vector from one space to a subspace. The notion of projection is useful for understanding the solutions to linear and kernel regression (See Lectures 2 and 3). An orthogonal projection matrix $\mathbf{P} \in \mathbb{R}^{n\times n}$ satisfies:

$$\mathbf{P} = \mathbf{P}^2 \qquad \text{and} \qquad \mathbf{P}^T = \mathbf{P}.$$

Orthogonality in projection implies that the projection matrix does not rescale the components being projected. Formally, this implies that an orthogonal projection matrix $\mathbf{P} : W \to W$ satisfies the property $\langle \mathbf{P}x, y\rangle = \langle x, \mathbf{P}y\rangle$ for all $x, y \in W$.

For a vector, $\mathbf{u} \in \mathbb{R}^n$, one can define the orthogonal decomposition with respect to a subspace $\mathcal{T} \subseteq \mathbb{R}^n$. Specifically, we can decompose $\mathbf{u}$ as:

$$\mathbf{u} = \underbrace{\mathbf{v}}_{\in\mathcal{T}} + \underbrace{\mathbf{w}}_{\in\mathcal{T}^\perp}.$$

Decompositions of this form are particularly useful for dimensionality reduction.

3.3 Analysis

3.3.1 Norms

We will be using norms throughout this course, but non-Euclidean norms will mostly arise in Lecture 3 and Lecture 9. We begin with the definition of a norm below and then list key examples of matrix and vector norms, which are helpful in theoretical analysis of machine learning methods.

Definition 8. *Let V be a vector space over $\mathbb{R}$. A norm $\|\cdot\| : V \to \mathbb{R}_+$ is a function such that for any $u, v \in V$ and $a \in \mathbb{R}$:*

(a) $\|u\| = 0$ iff $u = 0$ (Positive Definiteness).

(b) $\|au\| = |a|\|u\|$ (Homogeneity).

(c) $\|u + v\| \leq \|u\| + \|v\|$ (Triangle Inequality).

Below are several examples of norms that are commonly used in machine learning analyses. We encourage the reader to double check that the examples below are all norms.

Definition 9. *The **Frobenius norm** of an $m \times n$ matrix $\mathbf{A}$, denoted $\|\mathbf{A}\|_F$, is the square root of the sum of the magnitudes of its elements:*

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |\mathbf{A}_{i,j}|^2}.$$

From the above definition, it follows that the Frobenius norm can also be represented as the square root of the sum of the squares of the singular values of $\mathbf{A}$, as follows:

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{\min(m,n)} \sigma_i(\mathbf{A})^2}.$$

Another useful formulation of the Frobenius norm, for $\mathbf{A} \in \mathbb{R}^{m\times n}$, is the following:

$$\|\mathbf{A}\|_F = \sqrt{\text{Tr}(\mathbf{A}\mathbf{A}^T)}.$$

Definition 10. *The **nuclear norm** of an $m \times n$ matrix $\mathbf{A}$, denoted $\|\mathbf{A}\|_*$, is the sum of its singular values:*

$$\|\mathbf{A}\|_* = \sum_{i=1}^{\min(m,n)} \sigma_i(\mathbf{A}).$$

Definition 11. *The ℓ^2 **norm** of a length-n vector $\mathbf{x}$, denoted $\|\mathbf{x}\|_2$, is the square root of the sum of the magnitudes of its elements:*

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} |\mathbf{x}_i|^2}.$$

Definition 12. *The **spectral norm** of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, denoted $\|\mathbf{A}\|_2$, is the maximum singular value of $\mathbf{A}$:*

$$\|\mathbf{A}\|_2 = \sigma_1(\mathbf{A}).$$

We can also represent the spectral norm as follows, by relating it to the vector ℓ^2 norm:

$$\|\mathbf{A}\|_2 = \max_{\|\mathbf{x}\|_2=1} |\mathbf{A}\mathbf{x}|_2.$$

Essentially, the spectral norm represents the maximum amount by which the matrix $\mathbf{A}$ can "stretch" a unit vector, which exactly corresponds to its maximal singular value.

Definition 13. *The* $\mathbf{L}^\infty$ ***norm*** *of an* $m \times n$ *matrix* $\mathbf{A}$*, denoted* $\|\mathbf{A}\|_\infty$*, is the maximum row-wise magnitude sum of* $\mathbf{A}$*:*

$$\|\mathbf{A}\|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}|.$$

3.3.2 Inner Products

The inner product over a vector space V, often denoted $\langle \cdot, \cdot \rangle$, is a function which maps two elements in the vector space to a scalar. We will be utilizing inner products heavily throughout the course (See Lectures 2, 3, 5, and 6).

For real-values vectors, $x, y \in \mathbb{R}^n$, we can use the well-known *dot product* as the inner product, thus obtaining:

$$\langle x, y \rangle = x \cdot y = y^T x = \sum_{i=1}^{n} x_i y_i.$$

This inner product has the functional form: $\langle \cdot, \cdot \rangle : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$. Furthermore, the dot product over real-valued vectors has a few key properties. Notably:

$$x \cdot y = \|x\|_2 \|y\|_2 \cos\theta.$$

In the above equation, θ corresponds to the angle between the two vectors in Euclidean space. In machine learning, if x and y were, for example, the predicted and actual label for a specific model training example, $\cos\theta$ is commonly referred to as the cosine similarity metric (in statistics, this is just the correlation). In addition, for two real vectors x, y having nonzero ℓ_2 norm, if their dot product is 0, this corresponds to $\theta = \frac{\pi}{2}$, implying that the vectors are orthogonal. Finally, if $x = y$, we have:

$$\langle x, x \rangle = x \cdot x = x^T x = \|x\|_2^2.$$

For complex-valued vectors, $x, y \in \mathbb{C}^n$, we can use the dot product with complex conjugate to define an inner product as follows

$$\langle x, y \rangle = \sum_{i=1}^{n} x_i \overline{y_i} \ ;$$

where $\overline{y_i}$ denotes the complex conjugate of y_i. This inner product has the functional form: $\langle \cdot, \cdot \rangle : \mathbb{C}^n \times \mathbb{C}^n \to \mathbb{C}$. A similar angle-relation holds for the complex dot product: $\|x\|_2 \|y\|_2 \cos\theta = \mathrm{Re}(x \cdot y)$. This inner product will importantly be used in Lecture 5.

For matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$, the trace inner product is given by:

$$\langle \mathbf{A}, \mathbf{B} \rangle = \mathrm{Tr}(\mathbf{A}^T \mathbf{B}).$$

The above is often called the Frobenius inner product due to its relation to the Frobenius norm. More generally, inner products are naturally related to norms. Indeed a norm, $\|\cdot\|$ is induced by an inner product $\langle \cdot, \cdot \rangle$ via the relation $\|\cdot\| = \sqrt{\langle \cdot, \cdot \rangle}$.

As we will see in Lecture 5, there are also inner products on functions. Let f, g be real-valued, continuous functions defined on interval $[a, b] \subset \mathbb{R}$. The L^2 inner product over the space of such functions is given by:

$$\langle f, g \rangle = \int_a^b f(x) g(x) dx.$$

Correspondingly, the L^2 norm of this space, for a function f, is then given by:

$$\|f\|_{L^2} = \sqrt{\langle f, f \rangle} = \left(\int_a^b (f(x))^2 dx \right)^{\frac{1}{2}}.$$

3.3.3 Matrix Gradients

Matrix and vector gradients are commonplace in the analysis of neural networks. We will be commonly computing such gradients in Lectures 2, 3, 4, 5, 6. The key to taking matrix/vector gradients is to simply compute the partial derivative with respect to 1 entry at a time and then re-organize the entries into the right shape to see if a clear pattern emerges. The standard text to recommend for examples of such gradient computations is The Matrix Cookbook. Problem set 1 requires the computation of such a gradient since it will be used in the analysis of linear and kernel regression.

Below, we present a few key examples, for matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$.

- **Trace**: $\frac{\partial \text{Tr}(\mathbf{A})}{\partial \mathbf{A}}$ (let $m = n$ for this case)
  To calculate the gradient with respect to the trace, consider the element-wise gradient first. The trace is simply the sum of the diagonal elements of $\mathbf{A}$, or $\text{Tr}(\mathbf{A}) = a_{11} + a_{22} + \cdots + a_{nn}$. Note that the partial derivative of $\text{Tr}(\mathbf{A})$ is 0 with respect to any non-diagonal elements of $\mathbf{A}$, and 1 with respect to the diagonal. Thus, the gradient is simply the identity:

$$\frac{\partial \text{Tr}(\mathbf{A})}{\partial \mathbf{A}} = \mathbf{I}_{n \times n}.$$

- **Frobenius Norm Squared**: $\frac{\partial \|\mathbf{A}\|_F^2}{\partial \mathbf{A}}$
  First, we expand the norm as follows: $\|\mathbf{A}\|_F^2 = \text{Tr}(\mathbf{A}\mathbf{A}^T)$. Now, consider each term in the matrix $\mathbf{A}\mathbf{A}^T$. Each entry is formed as the dot product of two rows in $\mathbf{A}$. Specifically, $(\mathbf{A}\mathbf{A}^T)_{ij}$ is the dot product of the i-th and j-th rows in $\mathbf{A}$, so $(\mathbf{A}\mathbf{A}^T)_{ij} = a_{i1}a_{j1} + \cdots + a_{in}a_{jn}$. Since we take the trace of this matrix, we obtain $\text{Tr}(\mathbf{A}\mathbf{A}^T) = \sum_{i=1}^m \left[a_{i1}^2 + \cdots + a_{in}^2\right] = \sum_{i=1}^m \sum_{j=1}^n a_{ij}^2$. Note that the derivative of the trace with respect to any element in $\mathbf{A}$ is then simply given by:

$$\frac{\partial \text{Tr}(\mathbf{A}\mathbf{A}^T)}{\partial a_{ij}} = \frac{\partial \sum_{i=1}^m \sum_{j=1}^n a_{ij}^2}{\partial a_{ij}} = 2a_{ij}.$$

  Thus, it follows that the derivative of the original formulation is given by:

$$\frac{\partial \|\mathbf{A}\|_F^2}{\partial \mathbf{A}} = 2\mathbf{A}.$$

3.3.4 Taylor Series and Approximation

The Taylor Series is an invaluable function approximation tool, which in this course, is helpful in the derivation of the NTK (See Lecture 6). We build on the single-variable n-th order Taylor approximation when approximating $f(x)$ around a point y:

$$f(x) \approx f(y) + \frac{f'(y)}{1!}(x-y) + \frac{f''(y)}{2!}(x-y)^2 + \cdots + \frac{f^{(n)}(y)}{n!}(x-y)^n.$$

In the case of multivariate functions, we replace the derivative with a gradient, as follows. Consider a function $f : \mathbb{R}^n \to \mathbb{R}$. We can construct a first-order approximation to $f(x)$ around a vector $\mathbf{y} \in \mathbb{R}^n$, as follows:

$$f(\mathbf{x}) \approx f(\mathbf{y}) + \nabla f(\mathbf{y})^T(\mathbf{x} - \mathbf{y}).$$

While we can use higher-order gradients to obtain a more accurate expansion, the first order approximation is sufficient for the material in this course.

3.3.5 Convexity

Convex analysis is the foundation of many well-known algorithms in machine learning, including gradient descent and its variants. A full review of optimization is out of scope for this course, but we expect the reader to understand that the loss landscape for linear and kernel regression is convex, implying the existence of a unique minimum norm solution (See Lectures 2 and 3). The definition of convexity that is simple to verify for the squared loss used in Lectures 2 and 3 is the following.

Definition 14. *A twice-differentiable, real-valued function $f : \mathbb{R}^d \to \mathbb{R}$ is said to be* ***convex*** *if and only if for all $\tilde{x}, x \in \mathbb{R}^d$:*

$$x^T H_f(\tilde{x}) x \geq 0 \ ,$$

i.e. the Hessian of f, H_f, is positive semi-definite at all points $\tilde{x} \in \mathbb{R}^d$.

3.4 Probability

Probability theory naturally arises in several analyses in machine learning. We will mainly use simple concepts from probability theory (e.g. computing expectations of functions of random variables) in Lectures 5, 6, 7, 8, 9. Below, we begin with a brief introduction to notation.

- $\mathbb{P}_X(x)$ denotes the probability, with respect to random variable X, that X takes the value x.
- $X \sim f$ indicates that random variable X is distributed with density function f.
- For probability density function f, $\mathbb{E}_{X \sim f}[X]$ denotes the expected value of X.
- $\mathcal{N}(\mu, \sigma^2)$ denotes the Gaussian distribution with mean μ and standard deviation σ.

3.4.1 Expectation

The expectation of a discrete random variable, X, taking n possible values, is given by:

$$\mathbb{E}_X[X] = \sum_{i=1}^{n} x_i \cdot \mathbb{P}_X[x_i].$$

Analogously, the expectation of a continuous random variable, X, with probability density function, f, is given by:

$$\mathbb{E}_X[X] = \int_{-\infty}^{\infty} x f(x) dx.$$

Consider, for example, a random variable X distributed as a Gaussian with mean 0 and standard deviation 1. The probability density function of this continuous random variable is given by:

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right).$$

We now evaluate the expectation of X and verify that $\mathbb{E}_{X \sim \mathcal{N}(0,1)}[X] = 0$.

$$\begin{aligned}
\mathbb{E}_{X \sim \mathcal{N}(0,1)}[X] &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} x e^{-x^2/2} dx \\
&= -\frac{1}{\sqrt{2\pi}} \int_{\infty}^{-\infty} e^u du \\
&= -\frac{1}{\sqrt{2\pi}} e^{-x^2/2} \Big|_{-\infty}^{\infty} \\
&= 0.
\end{aligned}$$

An important property of expectation is linearity, which states that for random variables X, Y and constants $\alpha, \beta, \gamma \in \mathbb{R}$:

$$\mathbb{E}[\alpha X + \beta Y + \gamma] = \alpha \mathbb{E}[X] + \beta \mathbb{E}[Y] + \gamma.$$

Remarkably, the above theorem makes no assumption about the independence of the random variables, and thus holds even if X and Y are dependent.

3.4.2 Covariance and Variance

The variance and covariance correspond to second-order terms which capture higher moments of the underlying distributions. These can be computed as follows.

Definition 15. *The **variance** of a random variable, X, is given by:*

$$Var[X] = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - \mathbb{E}^2[X].$$

Definition 16. *The **covariance** of random variables, X, Y, is given by:*

$$Cov[X, Y] = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y].$$

3.4.3 Properties of Gaussian Distributions

Gaussian distributions are of particular significance for this course since we will be considering the behavior of neural networks with weights sampled i.i.d. from a Gaussian distribution (See Lectures 5 and 6). We begin by integrating the Gaussian probability density function to show that the integral has value 1. To this end, we are interested in evaluating integrals of the form:

$$\int_{-\infty}^{\infty} e^{-cx^2} dx.$$

While the integral above is not straightforward to solve via classical methods, we solve it by instead considering a double integral. Namely, we begin by letting the integral be represented by a function, $g(c) = \int_{-\infty}^{\infty} e^{-cx^2} dx$. Then, we seek to evaluate $g(c)^2$, as follows:

$$\begin{aligned} g(c)^2 &= \left(\int_{-\infty}^{\infty} e^{-cx^2} dx \right)^2 \\ &= \left(\int_{-\infty}^{\infty} e^{-cx^2} dx \right) \cdot \left(\int_{-\infty}^{\infty} e^{-cy^2} dy \right) \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-c(x^2+y^2)} dx dy. \end{aligned}$$

Now, we can convert the above integral to polar coordinates by completing the square via the transformation $r^2 = x^2 + y^2$:

$$\begin{aligned} g(c)^2 &= \int_0^{2\pi} \int_0^{\infty} e^{-cr^2} r dr d\theta \\ &= 2\pi \int_0^{\infty} r e^{-cr^2} dr \\ &= -\frac{\pi}{c} \int_0^{-\infty} e^u du \\ &= \frac{\pi}{c} e^u \Big|_{-\infty}^{0} \\ &= \frac{\pi}{c}. \end{aligned}$$

Thus, it follows that $g(c) = \sqrt{\frac{\pi}{c}}$. Hence, for $c = \frac{1}{2}$, we recover the normalizing constant of $\frac{1}{\sqrt{2\pi}}$ for the standard Gaussian distribution.

Now that we have a technique for evaluating integrals of the form:

$$\int_{-\infty}^{\infty} e^{-ax^2} dx \ ,$$

we can similarly evaluate integrals of the form:

$$\int_{-\infty}^{\infty} e^{-ax^2+bx+c} dx \ ,$$

by recognizing that the integrand (after completing the square) is the integral over a Gaussian distribution with nonzero mean and variance. This technique will appear throughout the course and so, we walk through some examples in the homework.

Another key property of Gaussian distributions is that two independent Gaussian random variables, $X \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $Y \sim \mathcal{N}(\mu_2, \sigma_2^2)$, have a sum whose distribution is also Gaussian. That is:

$$X + Y \sim \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2).$$

Remarkably, the above also holds in the multivariate Gaussian case, wherein variables are distributed with mean vector μ and covariance matrix Σ. Furthermore, the product of a constant with a Gaussian random variable is still Gaussian, with a scaled mean and variance. Specifically, for $X \sim \mathcal{N}(\mu, \sigma^2)$, and $a \in \mathbb{R}$, we have $aX \sim \mathcal{N}(a\mu, a^2\sigma^2)$.

3.4.4 Law of Large Numbers

The final topic we review in these notes is the Law of Large Numbers, which is useful in proving properties of neural networks at the infinite-width limit (See Lectures 5 and 6). Intuitively, the law of large numbers states that as the number of trials of an experiment increases, the expected value and the observed value will converge. Formally, let $X_1, \ldots, X_n$ be independent, identically distributed random variables with mean $\mu = \mathbb{E}_{X_i}\left[\frac{1}{n}\sum_{i=1}^n X_i\right]$. If $Z = \frac{1}{n}\sum_{i=1}^n X_i$, then for any $\epsilon > 0$, there exists N such that for $n > N$:

$$\mathbb{P}_Z\left(|Z - \mu| < \epsilon\right) > 1 - \epsilon$$

References

[1] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov, and R. Wang. On Exact Computation with an Infinitely Wide Neural Net. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019.

[2] S. Arora, S. S. Du, Z. Li, R. Salakhutdinov, R. Wang, and D. Yu. Harnessing the Power of Infinitely Wide Deep Nets on Small-data Tasks. In *International Conference on Learning Representations*, 2020.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

[4] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever. Generative pretraining from pixels. In *International Conference on Machine Learning*, 2020.

[5] A. Damian, J. Lee, and M. Soltanolkotabi. Neural networks can learn representations with gradient descent. In *Conference on Learning Theory*, pages 5413–5452. PMLR, 2022.

[6] D. Dua and C. Graff. UCI machine learning repository, 2017.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition*. Institute of Electrical and Electronics Engineers, 2016.

[8] G. Huang, Z. Liu, L. van der Maaten, and K. Weinberger. Densely connected convolutional networks. In *Computer Vision and Pattern Recognition*, 2017.

[9] A. Jacot, F. Gabriel, and C. Hongler. Neural Tangent Kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2018.

[10] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

[11] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems*, 2012.

[12] J. Lee, S. S. Schoenholz, J. Pennington, B. Adlam, L. Xiao, R. Novak, and J. Shol-Dickstein. Finite Versus Infinite Neural Networks: an Empirical Study. In *Advances in Neural Information Processing Systems*, 2020.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *Neural Information Processing Systems Workshop onDeep Learning*, 2013.

[14] R. Novak, L. Xiao, J. Hron, J. Lee, A. A. Alemi, J. Sohl-Dickstein, and S. S. Schoenholz. Neural Tangents: Fast and easy infinite neural networks in Python. In *International Conference on Learning Representations*, 2020.

[15] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019.

[16] A. Radhakrishnan*, D. Beaglehole*, P. Pandit, and M. Belkin. Feature learning in neural networks and kernel machines that recursively learn features. *arXiv:2212.13881*, 2022.

[17] A. Radhakrishnan, G. Stefanakis, M. Belkin, and C. Uhler. Simple, fast, and flexible framework for matrix completion with infinite width neural networks. *arXiv:2108.00131*, 2021.

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and F.-F. Li. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015.

[19] Z. Shi, J. Wei, and Y. Lian. A theoretical analysis on feature learning in neural networks: Emergence from inputs and advantage over fixed features. In *International Conference on Learning Representations*, 2022.

[20] J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *Computer Vision and Pattern Recognition*, 2011.

[21] M. Tan and Q. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, 2019.

[22] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, D. Keysers, J. Uszkoreit, M. Lucic, and A. Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. *Computing Research Repository*, 2021.

[23] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Deep Image Prior. In *Computer Vision and Pattern Recognition*, pages 9446–9454. Institute of Electrical and Electronics Engineers, 2018.

[24] G. Yang and E. J. Hu. Tensor Programs IV: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, 2021.

[25] S. Zagoruyko and N. Komodakis. Wide residual networks. *Computing Research Repository*, 2016.