# Effective Theory of Deep Learning

## Beyond the Infinite-Width Limit

Dan Roberts[a] and Sho Yaida[b]

[a]MIT, IAIFI, & Salesforce, [b]Facebook AI Research

*Deep Learning Theory Summer School at Princeton*
July 27, 2021 – August 8, 2021

Based on The Principles of Deep Learning Theory, also with Boris Hanin: 2106.10165.

# Initialization

> *The simulation is such that [one] generally perceives the sum of many billions of elementary processes simultaneously, so that the leveling law of large numbers completely obscures the real nature of the individual processes.*
>
> John von Neumann

Thanks to substantial investments into computer technology, modern **artificial intelligence** (AI) systems can now come equipped with many billions of elementary components.

# Initialization

> *The simulation is such that [one] generally perceives the sum of many billions of elementary processes simultaneously, so that the leveling law of large numbers completely obscures the real nature of the individual processes.*
>
> John von Neumann

Thanks to substantial investments into computer technology, modern **artificial intelligence** (AI) systems can now come equipped with many billions of elementary components.

- Behind much of this success is **deep learning**: deep learning uses artificial **neural networks** as an underlying model for AI.

# Initialization

> *The simulation is such that [one] generally perceives the sum of many billions of elementary processes simultaneously, so that the leveling law of large numbers completely obscures the real nature of the individual processes.*
>
> John von Neumann

Thanks to substantial investments into computer technology, modern **artificial intelligence** (AI) systems can now come equipped with many billions of elementary components.

- Behind much of this success is **deep learning**: deep learning uses artificial **neural networks** as an underlying model for AI.
- The real power of this framework comes from **representation learning**: *deep* neural networks, with many neurons organized into sequential computational layers, that *learn* useful representations of the world.

# Initialization: Goals

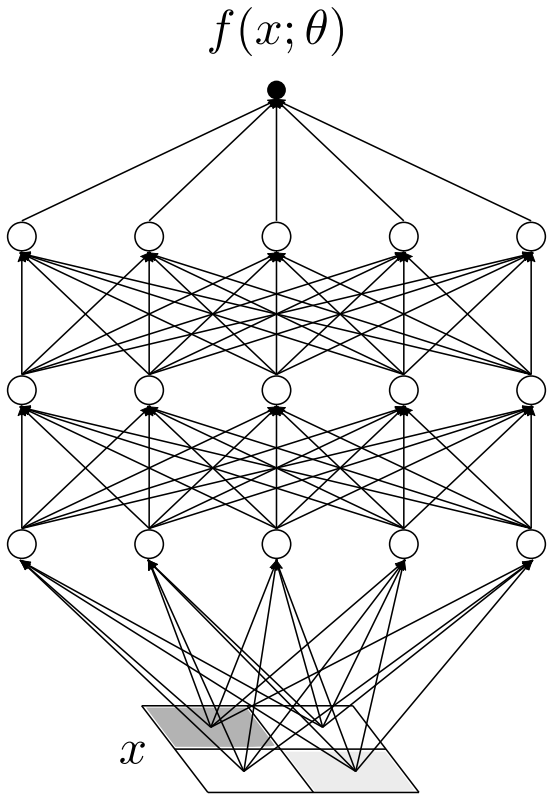
The goal of these lectures is to put forth a set of **principles** that enable us to theoretically analyze *deep* neural networks of *actual relevance*. To initialize you to this task, in the beginning of this lecture we'll explain at a very high-level both

*(i)* why such a goal is even attainable in theory, and

*(ii)* how we are able to get there in practice.

# Neural Networks

A **neural network** is a recipe for computing a function built out of many computational units called **neurons**:




Neurons are then organized in parallel into **layers**, and *deep* neural networks are those composed of multiple layers in sequence.

# Neural Networks Abstracted

For a moment, let's ignore all that structure and simply think of a neural network as a parameterized function

$$f(x;\theta)\,,$$

where $x$ is the input to the function and $\theta$ is a vector of a large number of **parameters** controlling the shape of the function.

# Neural Networks Abstracted: Function Approximation

For such a function to be useful, we need to tune the high-dimensional parameter vector $\theta$:

# Neural Networks Abstracted: Function Approximation

For such a function to be useful, we need to tune the high-dimensional parameter vector $\theta$:

- First, we choose an **initialization distribution** by randomly sampling the parameter vector $\theta$ from a computationally simple probability distribution,

$$p(\theta)\,.$$

# Neural Networks Abstracted: Function Approximation

For such a function to be useful, we need to tune the high-dimensional parameter vector $\theta$:

- First, we choose an **initialization distribution** by randomly sampling the parameter vector $\theta$ from a computationally simple probability distribution,

$$p(\theta)\,.$$

- Second, we adjust the parameter vector as $\theta \to \theta^\star$, such that the resulting *network function* $f(x;\theta^\star)$ is as close as possible to a desired *target function* $f(x)$:

$$f(x;\theta^\star) \approx f(x)\,.$$

This is called **function approximation**.

# Neural Networks Abstracted: Training

To find these tunings $\theta^\star$, we fit the network function $f(x;\theta)$ to **training data**, consisting of many pairs of the form $(x, f(x))$ observed from the desired – but only partially observable – target function $f(x)$.

- Making these adjustments is called **training**.
- The particular procedure used to tune them is called a **learning algorithm**.

# The Theoretical Minimum

Our goal is to understand this *trained* network function:

$$f(x; \theta^\star) \, .$$

# The Theoretical Minimum

Our goal is to understand this *trained* network function:

$$f(x;\theta^\star)\,.$$

One way to see the kinds of technical problems that we'll encounter in pursuit of this goal is to *Taylor expand* our trained network function $f(x;\theta^\star)$ around the initialized value of the parameters $\theta$

$$f(x;\theta^\star) = f(x;\theta) + (\theta^\star - \theta)\,\frac{df}{d\theta} + \frac{1}{2}\,(\theta^\star - \theta)^2\,\frac{d^2 f}{d\theta^2} + \ldots\,,$$

where $f(x;\theta)$ and its derivatives on the right-hand side are all evaluated at initialized value of the parameters.

# The Theoretical Minimum: Problem 1

In general, the Taylor series contains an infinite number of terms

$$f\,, \quad \frac{df}{d\theta}\,, \quad \frac{d^2f}{d\theta^2}\,, \quad \frac{d^3f}{d\theta^3}\,, \quad \frac{d^4f}{d\theta^4}\,, \quad \ldots\,,$$

and in principle we need to compute them all.

# The Theoretical Minimum: Problem 2

Since the parameters $\theta$ are randomly sampled from $p(\theta)$, each time we initialize our network we get a different function $f(x;\theta)$, and we need to determine the mapping:

$$p(\theta) \to p\left(f, \frac{df}{d\theta}, \frac{d^2f}{d\theta^2}, \ldots\right) .$$

This means that each term $f$, $df/d\theta$, $d^2f/d\theta^2$, ... , in the Taylor expansion is really a *random function* of the input $x$, and this joint distribution will have intricate statistical dependencies.

# The Theoretical Minimum: Problem 3

The learned value of the parameters, $\theta^\star$, is the result of a complicated training process. In general, $\theta^\star$ is not unique and can depend on *everything*:

$$\theta^\star \equiv [\theta^\star]\left(\theta,\, f,\, \frac{df}{d\theta},\, \frac{d^2 f}{d\theta^2},\, \ldots;\, \text{learning algorithm};\, \text{training data}\right) .$$

Determining an *analytical* expression for $\theta^\star$ must take "*everything*" into account.

# Goal, restated

If we could solve all three of these problems, then we'd have a *distribution* over trained network functions

$$p(f^\star) \equiv p\Big(f(x;\theta^\star)\Big|\,\text{learning algorithm; training data}\Big)\,,$$

now conditioned in a simple way on the learning algorithm and the data we used for training.

# Goal, restated

If we could solve all three of these problems, then we'd have a *distribution* over trained network functions

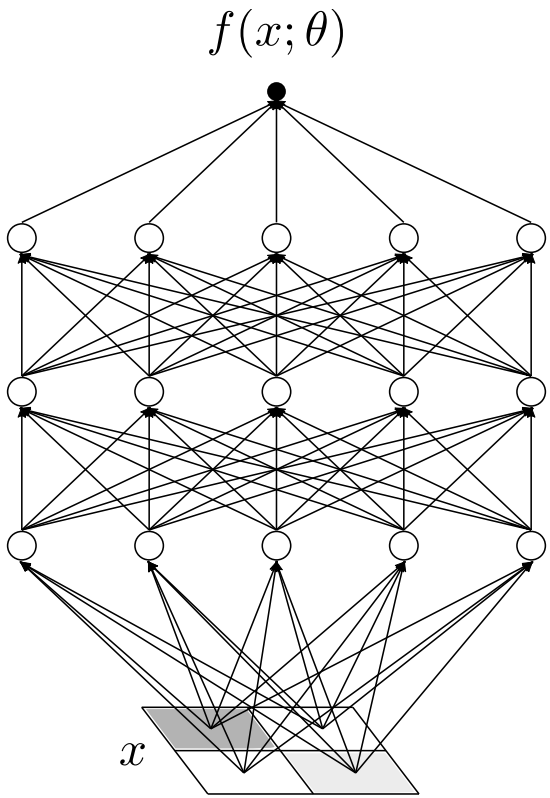
$$p(f^\star) \equiv p\Big(f(x;\theta^\star)\Big|\,\text{learning algorithm; training data}\Big)\,,$$

now conditioned in a simple way on the learning algorithm and the data we used for training.

The development of a method for the analytical computation of $p(f^\star)$ is the principle goal of these lectures.

# Fine, Structure

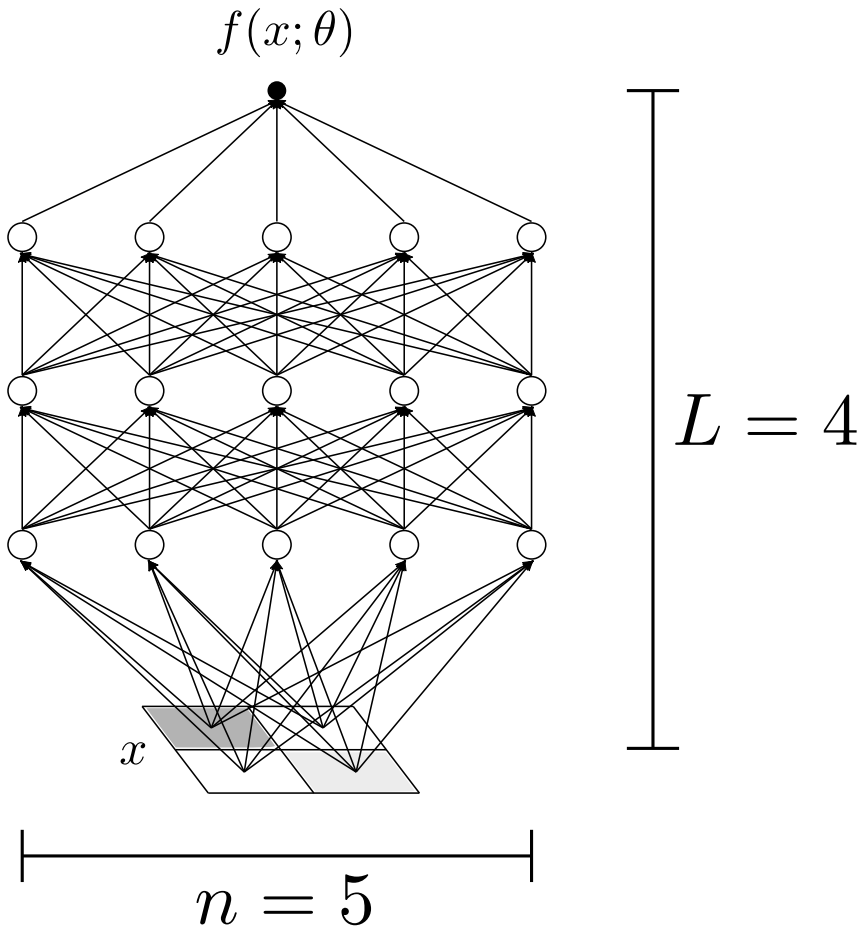
Solving our three problems for a general parameterized function $f(x;\theta)$ is not tractable. However, we only care about the functions that are deep neural networks:




To make progress we will have to make use of the particular **structure** of neural-network function.

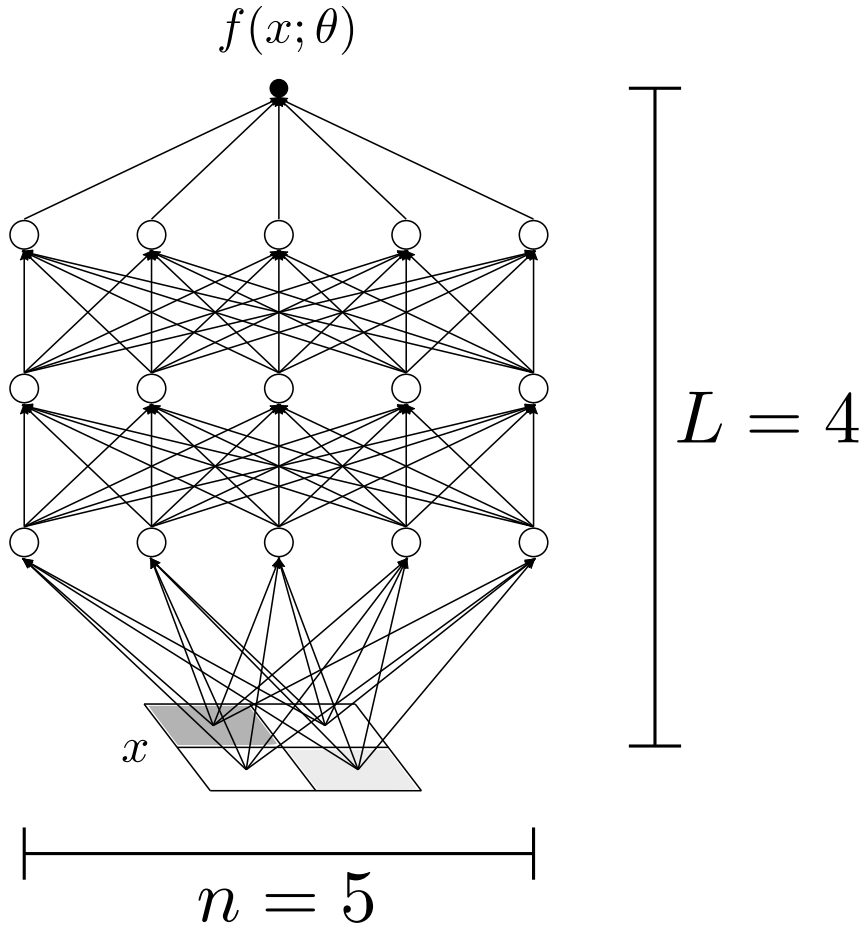
# Fine, Structure

Two essential aspects of a neural network *architecture* are its **width**, $n$, and its **depth**, $L$.



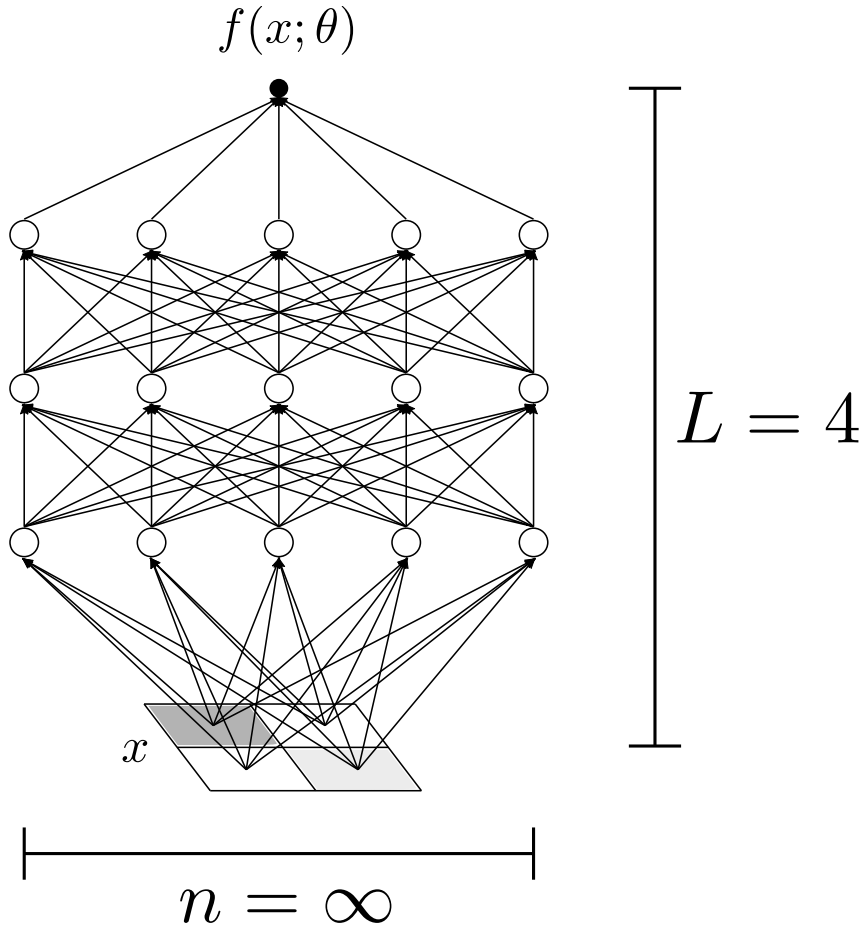
# A Principle of Sparsity

There are often simplifications to be found in the limit of a large number of components.



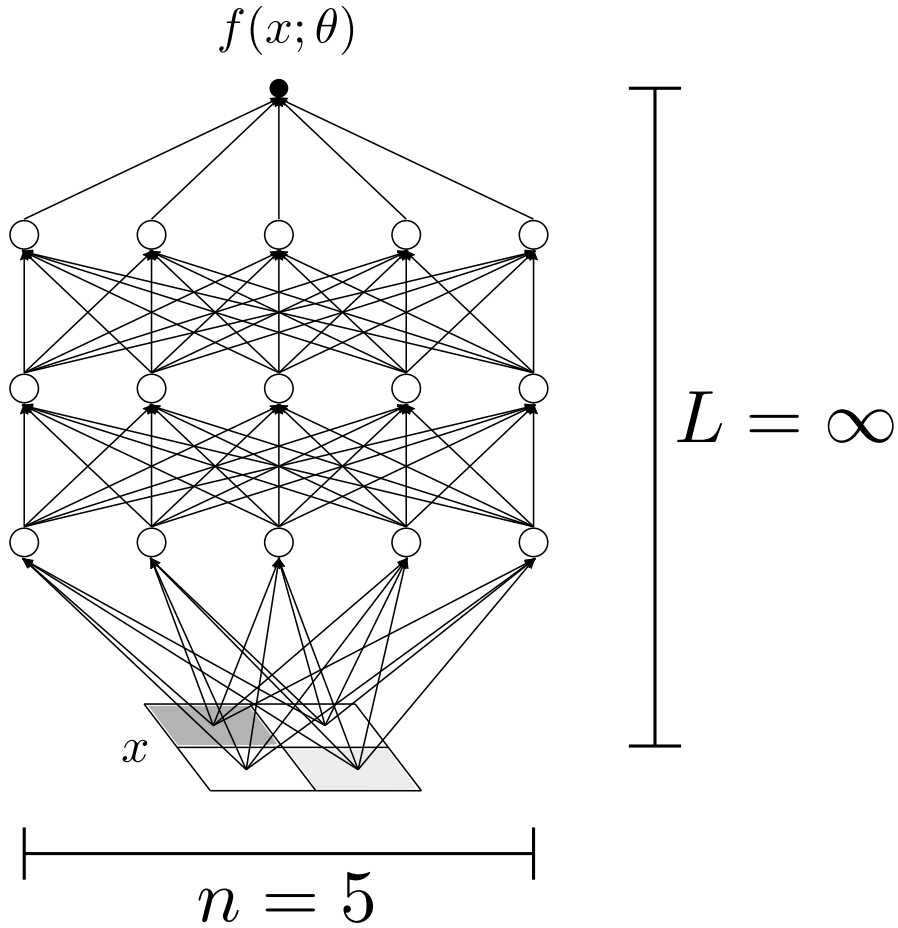
# A Principle of Sparsity

There are often simplifications to be found in the limit of a large number of components.



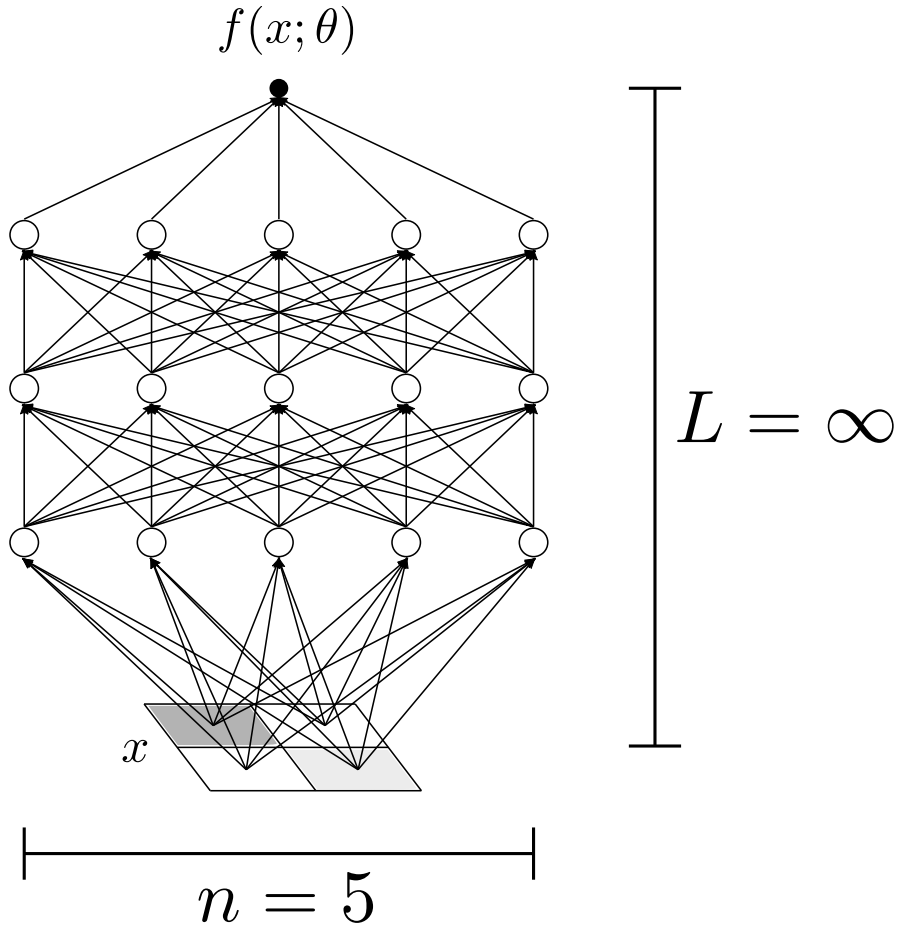
# A Principle of Sparsity

There are often simplifications to be found in the limit of a large number of components.



# A Principle of Sparsity

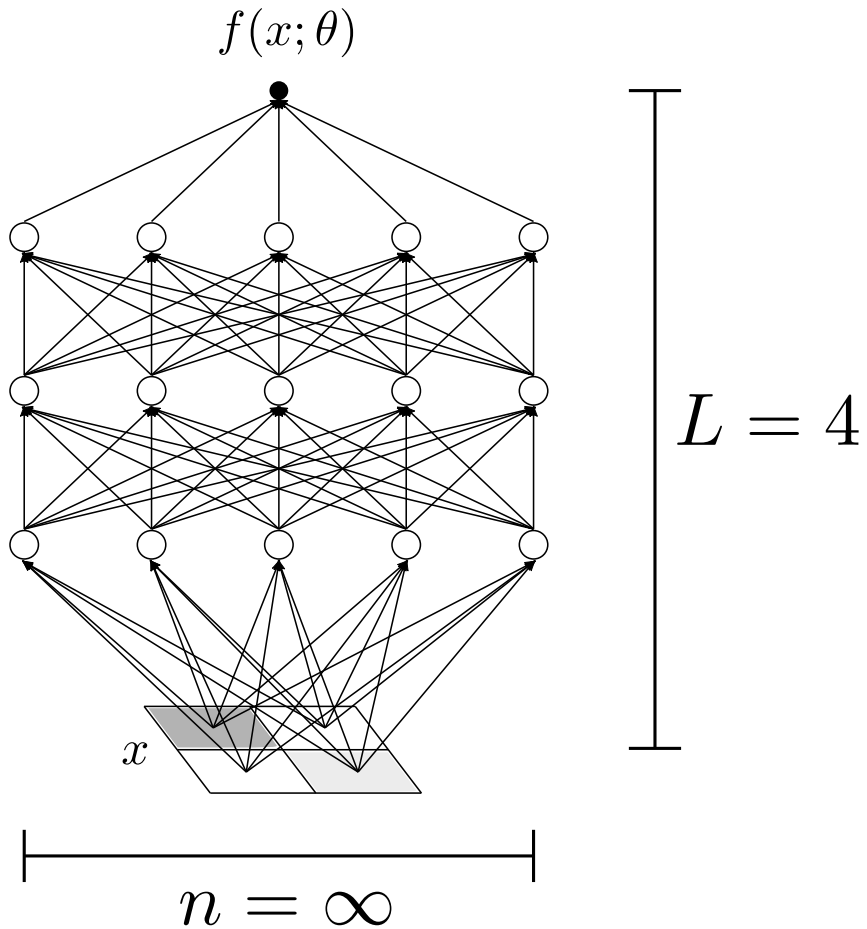
There are often simplifications to be found in the limit of a large number of components.




It's not enough to consider any massive macroscopic system, and taking the right limit often requires some care.

# A Principle of Sparsity

In this case, the $n = \infty$ will make everything really simple, while the $L = \infty$ will be hopelessly complicated and useless in practice.



# The Infinite-Width Limit

Let's begin by formally taking the limit

$$\lim_{n \to \infty} p(f^\star) \,,$$

and studying an *idealized* neural network in this limit.

# The Infinite-Width Limit

Let's begin by formally taking the limit

$$\lim_{n\to\infty} p(f^\star)\,,$$

and studying an *idealized* neural network in this limit.

- This is known as the **infinite-width limit** of the network, and as a strict limit it's rather *unphysical* for a network: obviously you cannot directly program a function to have an infinite number of components on a finite computer.

# The Infinite-Width Limit

Let's begin by formally taking the limit

$$\lim_{n\to\infty} p(f^\star)\,,$$

and studying an *idealized* neural network in this limit.

- This is known as the **infinite-width limit** of the network, and as a strict limit it's rather *unphysical* for a network: obviously you cannot directly program a function to have an infinite number of components on a finite computer.
- However, this extreme limit does massively simplify the distribution over trained networks $p(f^\star)$, rendering each of our three problems completely benign.

# Simplicity at Infinite Width

- Addressing **Problem 1**, higher derivative terms will effectively vanish, and we only need to keep track of two terms:

$$f\,, \quad \frac{df}{d\theta}\,.$$

# Simplicity at Infinite Width

- Addressing **Problem 1**, higher derivative terms will effectively vanish, and we only need to keep track of two terms:

$$f\,, \quad \frac{df}{d\theta}\,.$$

- Addressing **Problem 2**, the distributions of these random functions will be independent,

$$\lim_{n\to\infty} p\left(f, \frac{df}{d\theta}, \frac{d^2 f}{d\theta^2}, \ldots\right) = p(f)\, p\left(\frac{df}{d\theta}\right)\,,$$

with each marginal distribution factor taking a simple form.

# Simplicity at Infinite Width

- Addressing **Problem 1**, higher derivative terms will effectively vanish, and we only need to keep track of two terms:

$$f\,, \quad \frac{df}{d\theta}\,.$$

- Addressing **Problem 2**, the distributions of these random functions will be independent,

$$\lim_{n\to\infty} p\left(f, \frac{df}{d\theta}, \frac{d^2 f}{d\theta^2}, \ldots\right) = p(f)\, p\left(\frac{df}{d\theta}\right)\,,$$

  with each marginal distribution factor taking a simple form.
- Addressing **Problem 3**, the training dynamics become *linear* and *independent* of the details of the learning algorithm, giving $\theta^\star$ in a *closed form* analytical solution:

$$\lim_{n\to\infty} \theta^\star = [\theta^\star]\left(\theta,\, f,\, \frac{df}{d\theta};\, \text{training data}\right)\,.$$

# Simplicity at Infinite-Width

These simplifications are the consequence of a **principle of sparsity**, and the fully-trained distribution,

$$\lim_{n \to \infty} p(f^\star) \,,$$

is a simple **Gaussian distribution** with a nonzero mean.

# *Too Much* Simplicity at Infinite-Width

The formal infinite-width limit, $n \to \infty$, leads to a poor model of *deep* neural networks in practice:

# *Too Much* Simplicity at Infinite-Width

The formal infinite-width limit, $n \to \infty$, leads to a poor model of *deep* neural networks in practice:

- The distribution over *real* trained networks *does* depend on the properties of the learning algorithm used to train them.

# *Too Much* Simplicity at Infinite-Width

The formal infinite-width limit, $n \to \infty$, leads to a poor model of *deep* neural networks in practice:

- The distribution over *real* trained networks *does* depend on the properties of the learning algorithm used to train them.
- Infinite-width networks don't have **representation learning**: for any input $x$, its transformations in the hidden layers, $z^{(1)}$, $z^{(2)}$, $\ldots$, $z^{(L-1)}$, will remain unchanged from initialization.

# *Too Much* Simplicity at Infinite-Width

The formal infinite-width limit, $n \to \infty$, leads to a poor model of *deep* neural networks in practice:

- The distribution over *real* trained networks *does* depend on the properties of the learning algorithm used to train them.
- Infinite-width networks don't have **representation learning**: for any input $x$, its transformations in the hidden layers, $z^{(1)}$, $z^{(2)}$, $\ldots$, $z^{(L-1)}$, will remain unchanged from initialization.

The *central limiting* problem is that the input of an infinite number of signals is such that the leveling law of large numbers completely obscures the subtle correlations between neurons that get amplified over the course of training for representation learning.

# Interacting Neurons

We'll need to find a way to restore and then study the **interactions** between neurons that are present in realistic *finite-width* networks.

# Interacting Neurons

We'll need to find a way to restore and then study the **interactions** between neurons that are present in realistic *finite-width* networks.

To do so, we can use **perturbation theory** and study deep learning using a $\mathbf{1/n}$ **expansion**, treating the inverse layer width, $\epsilon \equiv 1/n$, as our small parameter of expansion:

$$p(f^\star) \equiv \left\{ \lim_{n\to\infty} p(f^\star) \right\} + \frac{p^{\{1\}}(f^\star)}{n} + \frac{p^{\{2\}}(f^\star)}{n^2} + \ldots ,$$

# Interacting Neurons

We'll need to find a way to restore and then study the **interactions** between neurons that are present in realistic *finite-width* networks.

To do so, we can use **perturbation theory** and study deep learning using a $\mathbf{1/n}$ **expansion**, treating the inverse layer width, $\epsilon \equiv 1/n$, as our small parameter of expansion:

$$p(f^\star) \equiv \left\{ \lim_{n\to\infty} p(f^\star) \right\} + \frac{p^{\{1\}}(f^\star)}{n} + O\left(\frac{1}{n^2}\right) .$$

# Near-Simplicity at Finite Width

- Addressing **Problem 1**, most derivatives will contribute as $O(1/n^2)$ or smaller, so we only need to keep track of 4 terms:

$$f\,, \quad \frac{df}{d\theta}\,, \quad \frac{d^2f}{d\theta^2}\,, \quad \frac{d^3f}{d\theta^3}\,.$$

# Near-Simplicity at Finite Width

- Addressing **Problem 1**, most derivatives will contribute as $O(1/n^2)$ or smaller, so we only need to keep track of 4 terms:

$$f\,, \quad \frac{df}{d\theta}\,, \quad \frac{d^2f}{d\theta^2}\,, \quad \frac{d^3f}{d\theta^3}\,.$$

- Addressing **Problem 2**, the distribution of these random functions at initialization will be *nearly* simple at order $1/n$:

$$p\left(f, \frac{df}{d\theta}, \frac{d^2f}{d\theta^2}, \frac{d^3f}{d\theta^3}\right)\,,$$

# Near-Simplicity at Finite Width

- Addressing **Problem 1**, most derivatives will contribute as $O(1/n^2)$ or smaller, so we only need to keep track of 4 terms:

$$f\,, \quad \frac{df}{d\theta}\,, \quad \frac{d^2f}{d\theta^2}\,, \quad \frac{d^3f}{d\theta^3}\,.$$

- Addressing **Problem 2**, the distribution of these random functions at initialization will be *nearly* simple at order $1/n$:

$$p\left(f, \frac{df}{d\theta}, \frac{d^2f}{d\theta^2}, \frac{d^3f}{d\theta^3}\right)\,,$$

- Addressing **Problem 3**, the *nonlinear* training dynamics can be tamed with *dynamical perturbation theory*, giving $\theta^\star$ in a *closed form* analytical solution:

$$\theta^\star = [\theta^\star]\left(\theta, f, \frac{df}{d\theta}\,, \frac{d^2f}{d\theta^2}\,\frac{d^3f}{d\theta^3}; \text{ learning algorithm; training data}\right)\,.$$

# Near-Simplicity at Finite Width

These near-simplifications are a further consequence of the **principle of sparsity**, and our *dual* **effective theory** description of the fully-trained distribution at order $1/n$,

$$p(f^\star) \equiv \left\{ \lim_{n\to\infty} p(f^\star) \right\} + \frac{p^{\{1\}}(f^\star)}{n} + O\left(\frac{1}{n^2}\right) ,$$

will be a **nearly-Gaussian distribution**.

# Subleading Corrections?

An important byproduct of studying the leading corrections is actually a deeper understanding of the truncation error. Defining the **aspect ratio**

$$r \equiv L/n\,,$$

we will see that we can recast our understanding of *infinite-width vs. finite-width* and *shallow vs. deep*:

# Subleading Corrections?

An important byproduct of studying the leading corrections is actually a deeper understanding of the truncation error. Defining the **aspect ratio**

$$r \equiv L/n\,,$$

we will see that we can recast our understanding of *infinite-width vs. finite-width* and *shallow vs. deep*:

- In the strict limit $r \to 0$, the interactions between neurons turn off: the infinite-width limit is actually a decent description, but these networks are **not really deep**, as their relative depth is zero: $L/n = 0$.

# Subleading Corrections?

An important byproduct of studying the leading corrections is actually a deeper understanding of the truncation error. Defining the **aspect ratio**

$$r \equiv L/n\,,$$

we will see that we can recast our understanding of *infinite-width vs. finite-width* and *shallow vs. deep*:

- In the regime $0 < r \ll 1$, there are nontrivial interactions between neurons: the finite-width effective theory truncated at order $1/n$ gives an accurate accounting $p(f^\star)$. These networks are **effectively deep**.

# Subleading Corrections?

An important byproduct of studying the leading corrections is actually a deeper understanding of the truncation error. Defining the **aspect ratio**

$$r \equiv L/n\,,$$

we will see that we can recast our understanding of *infinite-width vs. finite-width* and *shallow vs. deep*:

- In the regime $r \gg 1$, the neurons are strongly coupled: networks will behave chaotically, and there is no effective description due to large fluctuations from instantiation to instantiation. These networks are **overly deep**.

# Subleading Corrections?

An important byproduct of studying the leading corrections is actually a deeper understanding of the truncation error. Defining the **aspect ratio**

$$r \equiv L/n\,,$$

we will see that we can recast our understanding of *infinite-width vs. finite-width* and *shallow vs. deep*:

- In the regime $r \gg 1$, the neurons are strongly coupled: networks will behave chaotically, and there is no effective description due to large fluctuations from instantiation to instantiation. These networks are **overly deep**.

Networks of *practical use* have small aspect ratios: $r \sim r^\star \ll 1$.

# End of Initialization

To really describe the properties of **multilayer neural networks**, i.e. to understand **deep learning**, we need to study *large-but-finite-width networks*.

# Course Plan

Lecture 1 ~~**Initialization**~~**, Linear Models**

- §0 + §7.1 + §10.4

Lecture 2 **Gradient Descent & Nearly-Kernel Methods**

- §7.2 + §∞.2.2 + §11.4

Lecture 3 **The Principle of Sparsity (Recursing)**

- §4, §8, §11.2, §∞.3

Lecture 4 **The Principle of Criticality**

- §5, §9, §11.3, §∞.1, +§10.3

Lecture 5 **The End of Training & More**

- §∞.2.3 + Maybe { §A, §B, . . . }

# Problem 3: Training Dynamics

The learned value of the parameters, $\theta^\star$, is the result of a complicated training process. In general, $\theta^\star$ is not unique and can depend on *everything*:

$$\theta^\star \equiv [\theta^\star]\left(\theta,\, f,\, \frac{df}{d\theta},\, \frac{d^2 f}{d\theta^2},\, \ldots;\, \text{learning algorithm};\, \text{training data}\right) .$$

Determining an *analytical* expression for $\theta^\star$ must take "*everything*" into account.

# Problem 3: Training Dynamics

The learned value of the parameters, $\theta^\star$, is the result of a complicated training process. In general, $\theta^\star$ is not unique and can depend on *everything*:

$$\theta^\star \equiv [\theta^\star]\left(\theta,\, f,\, \frac{df}{d\theta},\, \frac{d^2 f}{d\theta^2},\, \ldots;\, \text{learning algorithm;}\, \text{training data}\right) .$$

Determining an *analytical* expression for $\theta^\star$ must take "*everything*" into account.

Goal right now is to understand the **algorithm dependence** and **data dependence** of solutions for a very general class of machine learning models.

# Machine Learning Models

$$f(x; \theta)$$

# Machine Learning Models

$$f(x;\theta) \equiv z(x;\theta)$$

# Machine Learning Models

$$f(x;\theta) \equiv z_i(x;\theta)$$

- $i = 1, \ldots, n_{\text{out}}$ is a **vectorial index**

# Machine Learning Models

$$f(x;\theta) \equiv z_i(x_\delta;\theta)$$

- $i = 1, \ldots, n_{\text{out}}$ is a **vectorial index**
- $\delta \in \mathcal{D}$ is a **sample index**

# Machine Learning Models

$$f(x;\theta) \equiv z_{i;\delta}(\theta)$$

- $i = 1, \ldots, n_{\text{out}}$ is a **vectorial index**
- $\delta \in \mathcal{D}$ is a **sample index**

# Machine Learning Models: Taylor Expanded

$z_{i;\delta}(\theta)$

# Machine Learning Models: Taylor Expanded

$$z_{i;\delta}(\theta) = z_{i;\delta}(\theta = 0) + \sum_{\mu=1}^{P} \theta_\mu \frac{dz_{i;\delta}}{d\theta_\mu}\bigg|_{\theta=0} + \frac{1}{2} \sum_{\mu_1,\mu_2=1}^{P} \theta_{\mu_1}\theta_{\mu_2} \frac{d^2 z_{i;\delta}}{d\theta_{\mu_1} d\theta_{\mu_2}}\bigg|_{\theta=0}$$
$$+ \frac{1}{3!} \sum_{\mu_1,\mu_2,\mu_3=1}^{P} \theta_{\mu_1}\theta_{\mu_2}\theta_{\mu_3} \frac{d^3 z_{i;\delta}}{d\theta_{\mu_1} d\theta_{\mu_2} d\theta_{\mu_3}}\bigg|_{\theta=0} + \dots$$

# Machine Learning Models: Taylor Expanded

$$z_{i;\delta}(\theta) = z_{i;\delta}(\theta = 0) + \sum_{\mu=1}^{P} \theta_\mu \frac{dz_{i;\delta}}{d\theta_\mu}\bigg|_{\theta=0} + \frac{1}{2} \sum_{\mu_1,\mu_2=1}^{P} \theta_{\mu_1}\theta_{\mu_2} \frac{d^2 z_{i;\delta}}{d\theta_{\mu_1} d\theta_{\mu_2}}\bigg|_{\theta=0}$$
$$+ \frac{1}{3!} \sum_{\mu_1,\mu_2,\mu_3=1}^{P} \theta_{\mu_1}\theta_{\mu_2}\theta_{\mu_3} \frac{d^3 z_{i;\delta}}{d\theta_{\mu_1} d\theta_{\mu_2} d\theta_{\mu_3}}\bigg|_{\theta=0} + \dots$$

- $\mu_i$ is **parameter index**, with $P$ total model parameters

# Machine Learning Models: Taylor Expanded

$$z_{i;\delta}(\theta) = z_{i;\delta}(\theta = 0) + \sum_{\mu=1}^{P} \theta_\mu \frac{dz_{i;\delta}}{d\theta_\mu}\bigg|_{\theta=0} + \frac{1}{2} \sum_{\mu_1,\mu_2=1}^{P} \theta_{\mu_1}\theta_{\mu_2} \frac{d^2 z_{i;\delta}}{d\theta_{\mu_1} d\theta_{\mu_2}}\bigg|_{\theta=0}$$
$$+ \frac{1}{3!} \sum_{\mu_1,\mu_2,\mu_3=1}^{P} \theta_{\mu_1}\theta_{\mu_2}\theta_{\mu_3} \frac{d^3 z_{i;\delta}}{d\theta_{\mu_1} d\theta_{\mu_2} d\theta_{\mu_3}}\bigg|_{\theta=0} + \dots$$

- $\mu_i$ is **parameter index**, with $P$ total model parameters
- generic **nonlinear model** will be intractable.

# Machine Learning Models: Taylor Expanded

$$z_{i;\delta}(\theta) = z_{i;\delta}(\theta = 0) + \sum_{\mu=1}^{P} \theta_\mu \frac{dz_{i;\delta}}{d\theta_\mu}\bigg|_{\theta=0} + \frac{1}{2}\sum_{\mu_1,\mu_2=1}^{P} \theta_{\mu_1}\theta_{\mu_2} \frac{d^2 z_{i;\delta}}{d\theta_{\mu_1} d\theta_{\mu_2}}\bigg|_{\theta=0}$$
$$+ \frac{1}{3!}\sum_{\mu_1,\mu_2,\mu_3=1}^{P} \theta_{\mu_1}\theta_{\mu_2}\theta_{\mu_3} \frac{d^3 z_{i;\delta}}{d\theta_{\mu_1} d\theta_{\mu_2} d\theta_{\mu_3}}\bigg|_{\theta=0}$$

- $\mu_i$ is **parameter index**, with $P$ total model parameters
- generic **nonlinear model** will be intractable.
- truncation to **cubic model** describes deep MLPs at $O(1/n)$

# Machine Learning Models: Taylor Expanded

$$z_{i;\delta}(\theta) = z_{i;\delta}(\theta = 0) + \sum_{\mu=1}^{P} \theta_\mu \frac{dz_{i;\delta}}{d\theta_\mu}\bigg|_{\theta=0} + \frac{1}{2} \sum_{\mu_1,\mu_2=1}^{P} \theta_{\mu_1}\theta_{\mu_2} \frac{d^2 z_{i;\delta}}{d\theta_{\mu_1} d\theta_{\mu_2}}\bigg|_{\theta=0}$$

- $\mu_i$ is **parameter index**, with $P$ total model parameters
- generic **nonlinear model** will be intractable.
- truncation to **cubic model** describes deep MLPs at $O(1/n)$
- truncation to **quadratic model** illustrates nonlinear dynamics

# Machine Learning Models: Taylor Expanded

$$z_{i;\delta}(\theta) = z_{i;\delta}(\theta = 0) + \sum_{\mu=1}^{P} \theta_\mu \frac{dz_{i;\delta}}{d\theta_\mu}\bigg|_{\theta=0}$$

- $\mu_i$ is **parameter index**, with $P$ total model parameters
- generic **nonlinear model** will be intractable.
- truncation to **cubic model** describes deep MLPs at $O(1/n)$
- truncation to **quadratic model** illustrates nonlinear dynamics
- truncation to **linear model** describes infinite-width networks

# Linear Models

The simplest linear model is a *one-layer (zero-hidden-layer) network*

$$z_{i_\delta}(\theta) = b_i + \sum_{j=1}^{n_0} W_{ij}\, x_{j;\delta} \,.$$

# Linear Models

The simplest linear model is a *one-layer (zero-hidden-layer) network*

$$z_{i_\delta}(\theta) = b_i + \sum_{j=1}^{n_0} W_{ij}\, x_{j;\delta}\,.$$

- While this model is linear in both the parameters $\theta = \{b_i, W_{ij}\}$ and the input $x_{j;\delta}$, the *linear* in **linear model** takes its name from the dependence on the parameters $\theta$ and not the input $x$.

# Linear Models

The simplest linear model is a *one-layer (zero-hidden-layer) network*

$$z_{i_\delta}(\theta) = b_i + \sum_{j=1}^{n_0} W_{ij}\, x_{j;\delta}\,.$$

- While this model is linear in both the parameters $\theta = \{b_i, W_{ij}\}$ and the input $x_{j;\delta}$, the *linear* in **linear model** takes its name from the dependence on the parameters $\theta$ and not the input $x$.
- While $x_{j;\delta}$ can serve as a reasonable set of **features** for function approximation, in general they do not.

# (Generalized) Linear Models

Instead, we might design a fixed set of **feature functions** $\phi_j(x)$ that's meant to work well for underlying task at hand:

$$z_{i;\delta}(\theta) = b_i + \sum_{j=1}^{n_f} W_{ij}\, \phi_j(x_\delta)$$

# (Generalized) Linear Models

Instead, we might design a fixed set of **feature functions** $\phi_j(x)$ that's meant to work well for underlying task at hand:

$$z_{i;\delta}(\theta) = b_i + \sum_{j=1}^{n_f} W_{ij}\, \phi_j(x_\delta)$$

- All the complicated modeling work goes into the construction of these feature functions $\phi_j(x)$.

# (Generalized) Linear Models

Instead, we might design a fixed set of **feature functions** $\phi_j(x)$ that's meant to work well for underlying task at hand:

$$z_{i;\delta}(\theta) = \sum_{j=0}^{n_f} W_{ij}\, \phi_j(x_\delta)$$

- All the complicated modeling work goes into the construction of these feature functions $\phi_j(x)$.
- Here, we've subsumed the bias vector into the weight matrix by setting $\phi_0(x) \equiv 1$ and $W_{i0} \equiv b_i$.

# (Generalized) Linear Models

Instead, we might design a fixed set of **feature functions** $\phi_j(x)$ that's meant to work well for underlying task at hand:

$$z_{i;\delta}(\theta) = \sum_{j=0}^{n_f} W_{ij}\, \phi_j(x_\delta)$$

- All the complicated modeling work goes into the construction of these feature functions $\phi_j(x)$.
- Here, we've subsumed the bias vector into the weight matrix by setting $\phi_0(x) \equiv 1$ and $W_{i0} \equiv b_i$.
- We can still think of this model as a one-layer neural network, but now we pre-process $x$ with the function $\phi_j(x)$.

# *Aside*: Supervised Learning

For any pair $(x, y)$ that is jointly sampled from a **data distribution**

$$p(x, y) = p(y|x)p(x)\,,$$

the goal is to predict a **label** $y$ given an input $x$. The better the *probabilistic model* learns the distribution $p(y|x)$, the more accurately it can predict a true label $y$ for a novel input example $x$.

# *Aside*: Supervised Learning

Before we understand how to *fit* the model parameters, we need to understand what we mean by making good predictions:

# *Aside*: Supervised Learning

Before we understand how to *fit* the model parameters, we need to understand what we mean by making good predictions:

- For a *typical* $(x_\delta, y_\delta)$ sampled from $p(x, y)$, we want $z_\delta(\theta)$ to be as close to $y_\delta$ as possible on average.

# *Aside*: Supervised Learning

Before we understand how to *fit* the model parameters, we need to understand what we mean by making good predictions:

- For a *typical* $(x_\delta, y_\delta)$ sampled from $p(x, y)$, we want $z_\delta(\theta)$ to be as close to $y_\delta$ as possible on average.
- To measure this proximity, we need to define an auxiliary function called the **loss**,

$$\mathcal{L}\Big(z_\delta(\theta),\, y_\delta\Big)\,,$$

such that the closer $z_\delta(\theta)$ is to $y_\delta$, the smaller the loss.

# *Aside*: Supervised Learning

Before we understand how to *fit* the model parameters, we need to understand what we mean by making good predictions:

- For a *typical* $(x_\delta, y_\delta)$ sampled from $p(x, y)$, we want $z_\delta(\theta)$ to be as close to $y_\delta$ as possible on average.
- To measure this proximity, we need to define an auxiliary function called the **loss**,

$$\mathcal{L}\Big(z_\delta(\theta),\, y_\delta\Big)\,,$$

  such that the closer $z_\delta(\theta)$ is to $y_\delta$, the smaller the loss.
- One intuitive choice for this is the **MSE loss**:

$$\mathcal{L}_{\text{MSE}}\Big(z_\delta(\theta),\, y_\delta\Big) \equiv \frac{1}{2}\Big[z_\delta(\theta) - y_\delta\Big]^2\,.$$

# *Aside*: Supervised Learning

To train our model, we try to find a configuration of the model parameters that minimizes the **training loss** $\mathcal{L}_{\mathcal{A}}(\theta)$:

$$\theta^{\star} = \arg\min_{\theta} \mathcal{L}_{\mathcal{A}}(\theta) = \arg\min_{\theta} \left[ \sum_{\tilde{\alpha} \in \mathcal{A}} \mathcal{L}\Big( z_{\tilde{\alpha}}(\theta),\, y_{\tilde{\alpha}} \Big) \right] ,$$

where $\mathcal{A}$ is the **training set**.

# *Aside*: Supervised Learning

To train our model, we try to find a configuration of the model parameters that minimizes the **training loss** $\mathcal{L}_{\mathcal{A}}(\theta)$:

$$\theta^{\star} = \arg\min_{\theta} \mathcal{L}_{\mathcal{A}}(\theta) = \arg\min_{\theta} \left[ \sum_{\tilde{\alpha} \in \mathcal{A}} \mathcal{L}\Big( z_{\tilde{\alpha}}(\theta),\, y_{\tilde{\alpha}} \Big) \right],$$

where $\mathcal{A}$ is the **training set**.

- To assess the **generalization** of a model, a **test set**, $(x_{\dot{\beta}}, y_{\dot{\beta}})_{\dot{\beta} \in \mathcal{B}}$, is used to evaluate a model after training.

# *Aside*: Supervised Learning

To train our model, we try to find a configuration of the model parameters that minimizes the **training loss** $\mathcal{L}_{\mathcal{A}}(\theta)$:

$$\theta^{\star} = \arg\min_{\theta} \mathcal{L}_{\mathcal{A}}(\theta) = \arg\min_{\theta} \left[ \sum_{\tilde{\alpha} \in \mathcal{A}} \mathcal{L}\Big( z_{\tilde{\alpha}}(\theta),\, y_{\tilde{\alpha}} \Big) \right],$$

where $\mathcal{A}$ is the **training set**.

- To assess the **generalization** of a model, a **test set**, $(x_{\dot{\beta}}, y_{\dot{\beta}})_{\dot{\beta} \in \mathcal{B}}$, is used to evaluate a model after training.
- We will denote *training set* inputs as $\tilde{\alpha} \in \mathcal{A}$, *generic* inputs as $\delta \in \mathcal{D}$, and *test set* inputs as $\dot{\beta} \in \mathcal{B}$.

# Linear Regression

Supervised learning with a *linear model* is **linear regression**

$$\mathcal{L}_A(\theta) = \frac{1}{2} \sum_{\tilde{\alpha} \in \mathcal{A}} \sum_{i=1}^{n_{\text{out}}} \left[ y_{i;\tilde{\alpha}} - \sum_{j=0}^{n_f} W_{ij} \phi_j(x_{\tilde{\alpha}}) \right]^2 .$$

# Linear Regression

Supervised learning with a *linear model* is **linear regression**

$$\mathcal{L}_A(\theta) = \frac{1}{2} \sum_{\tilde{\alpha} \in \mathcal{A}} \sum_{i=1}^{n_{\text{out}}} \left[ y_{i;\tilde{\alpha}} - \sum_{j=0}^{n_f} W_{ij} \phi_j(x_{\tilde{\alpha}}) \right]^2 .$$

Being entirely naive about it, let's try **direct optimization**:

$$0 = \left. \frac{d\mathcal{L}_{\mathcal{A}}}{dW_{ij}} \right|_{W=W^\star} ,$$

which gives an implicit equation for the optimal weight matrix $W^\star_{ij}$:

$$\sum_{k=0}^{n_f} W^\star_{ik} \left[ \sum_{\tilde{\alpha} \in \mathcal{A}} \phi_k(x_{\tilde{\alpha}}) \phi_j(x_{\tilde{\alpha}}) \right] = \sum_{\tilde{\alpha} \in \mathcal{A}} y_{i;\tilde{\alpha}} \phi_j(x_{\tilde{\alpha}}) .$$

# Linear Regression

To solve implicit equation,

$$\sum_{k=0}^{n_f} W^{\star}_{ik} \left[ \sum_{\tilde{\alpha} \in \mathcal{A}} \phi_k(x_{\tilde{\alpha}}) \phi_j(x_{\tilde{\alpha}}) \right] = \sum_{\tilde{\alpha} \in \mathcal{A}} y_{i;\tilde{\alpha}} \phi_j(x_{\tilde{\alpha}}) \,,$$

let's define a symmetric $(n_f + 1)$-by-$(n_f + 1)$ matrix of features:

$$M_{ij} \equiv \sum_{\tilde{\alpha} \in \mathcal{A}} \phi_i(x_{\tilde{\alpha}}) \phi_j(x_{\tilde{\alpha}}) \,.$$

# Linear Regression

To solve implicit equation,

$$\sum_{k=0}^{n_f} W^{\star}_{ik} \left[ \sum_{\tilde{\alpha} \in \mathcal{A}} \phi_k(x_{\tilde{\alpha}}) \phi_j(x_{\tilde{\alpha}}) \right] = \sum_{\tilde{\alpha} \in \mathcal{A}} y_{i;\tilde{\alpha}} \phi_j(x_{\tilde{\alpha}}) \,,$$

let's define a symmetric $(n_f + 1)$-by-$(n_f + 1)$ matrix of features:

$$M_{ij} \equiv \sum_{\tilde{\alpha} \in \mathcal{A}} \phi_i(x_{\tilde{\alpha}}) \phi_j(x_{\tilde{\alpha}}) \,.$$

Applying its inverse to both sides, we find a solution:

$$W^{\star}_{ij} = \sum_{k=0}^{n_f} \sum_{\tilde{\alpha} \in \mathcal{A}} y_{i;\tilde{\alpha}} \, \phi_k(x_{\tilde{\alpha}}) \left( M^{-1} \right)_{kj} \,.$$

# Linear Regression

To solve implicit equation,

$$\sum_{k=0}^{n_f} W^\star_{ik} \left[ \sum_{\tilde{\alpha} \in \mathcal{A}} \phi_k(x_{\tilde{\alpha}}) \phi_j(x_{\tilde{\alpha}}) \right] = \sum_{\tilde{\alpha} \in \mathcal{A}} y_{i;\tilde{\alpha}} \phi_j(x_{\tilde{\alpha}}) \,,$$

let's define a symmetric $(n_f + 1)$-by-$(n_f + 1)$ matrix of features:

$$M_{ij} \equiv \sum_{\tilde{\alpha} \in \mathcal{A}} \phi_i(x_{\tilde{\alpha}}) \phi_j(x_{\tilde{\alpha}}) \,.$$

Applying its inverse to both sides, we find a solution:

$$W^\star_{ij} = \sum_{k=0}^{n_f} \sum_{\tilde{\alpha} \in \mathcal{A}} y_{i;\tilde{\alpha}} \, \phi_k(x_{\tilde{\alpha}}) \left( M^{-1} \right)_{kj} \,.$$

- Notice that $W^\star_{ij}$ depends on $\mathcal{A}$, linearly on $y_{i;\tilde{\alpha}}$, and in a more complicated way on $\phi_k(x_{\tilde{\alpha}})$.

# Linear Regression

Finally, we can use this to make predictions on test-set inputs $x_{\dot{\beta}}$ as

$$z_i(x_{\dot{\beta}}; \theta^\star) = \sum_{j=0}^{n_f} W_{ij}^\star \phi_j(x_{\dot{\beta}})$$

# Kernel Methods

Finally, we can use this to make predictions on test-set inputs $x_{\dot{\beta}}$ as

$$
\begin{aligned}
z_i(x_{\dot{\beta}}; \theta^\star) &= \sum_{j=0}^{n_f} W_{ij}^\star \phi_j(x_{\dot{\beta}}) \\
&= \sum_{\tilde{\alpha} \in \mathcal{A}} \left[ \sum_{j,k=0}^{n_f} \phi_j(x_{\dot{\beta}}) \left( M^{-1} \right)_{jk} \phi_k(x_{\tilde{\alpha}}) \right] y_{i;\tilde{\alpha}}
\end{aligned}
$$

# Kernel Methods

Finally, we can use this to make predictions on test-set inputs $x_{\dot{\beta}}$ as

$$z_i(x_{\dot{\beta}}; \theta^\star) = \sum_{j=0}^{n_f} W_{ij}^\star \phi_j(x_{\dot{\beta}})$$

$$= \sum_{\tilde{\alpha} \in \mathcal{A}} \left[ \sum_{j,k=0}^{n_f} \phi_j(x_{\dot{\beta}}) \left( M^{-1} \right)_{jk} \phi_k(x_{\tilde{\alpha}}) \right] y_{i;\tilde{\alpha}}$$

- Note that this involves the inversion of $M_{ij}$.
- If $n_f \gg 1$, then this might be computationally difficult.

# *The Kernel*

Let us introduce a new $N_{\mathcal{D}} \times N_{\mathcal{D}}$-dimensional symmetric matrix:

$$k_{\delta_1 \delta_2} \equiv k(x_{\delta_1}, x_{\delta_2}) \equiv \sum_{i=0}^{n_f} \phi_i(x_{\delta_1})\, \phi_i(x_{\delta_2}) \ .$$

As an inner product of features, the **kernel** $k_{\delta_1 \delta_2}$ is a measure of similarity between two inputs $x_{i;\delta_1}$ and $x_{i;\delta_2}$ in *feature space*.

# The Kernel

Let us introduce a new $N_{\mathcal{D}} \times N_{\mathcal{D}}$-dimensional symmetric matrix:

$$k_{\delta_1\delta_2} \equiv k(x_{\delta_1}, x_{\delta_2}) \equiv \sum_{i=0}^{n_f} \phi_i(x_{\delta_1})\, \phi_i(x_{\delta_2}) \ .$$

As an inner product of features, the **kernel** $k_{\delta_1\delta_2}$ is a measure of similarity between two inputs $x_{i;\delta_1}$ and $x_{i;\delta_2}$ in *feature space*.

We'll also denote an $N_{\mathcal{A}}$-by-$N_{\mathcal{A}}$-dimensional submatrix of the kernel evaluated on the training set as $\widetilde{k}_{\tilde{\alpha}_1\tilde{\alpha}_2}$ with a tilde. This lets us write its **inverse** as $\widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_2}$, which satisfies

$$\sum_{\tilde{\alpha}_2 \in \mathcal{A}} \widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_2} \widetilde{k}_{\tilde{\alpha}_2\tilde{\alpha}_3} = \delta^{\tilde{\alpha}_1}_{\ \ \tilde{\alpha}_3} \ .$$

# Some Algebra

$$
\begin{aligned}
&\sum_{\tilde{\alpha}\in\mathcal{A}}\left[\sum_{j,k=0}^{n_f}\phi_j(x_{\dot{\beta}})\left(M^{-1}\right)_{jk}\phi_k(x_{\tilde{\alpha}})\right]\widetilde{k}_{\tilde{\alpha}\tilde{\alpha}_1}\\
&=\sum_{\tilde{\alpha}\in\mathcal{A}}\sum_{j,k=0}^{n_f}\phi_j(x_{\dot{\beta}})\left(M^{-1}\right)_{jk}\phi_k(x_{\tilde{\alpha}})\sum_{i=0}^{n_f}\phi_i(x_{\tilde{\alpha}})\,\phi_i(x_{\tilde{\alpha}_1})\\
&=\sum_{i,j,k=0}^{n_f}\phi_j(x_{\dot{\beta}})\left(M^{-1}\right)_{jk}M_{ki}\,\phi_i(x_{\tilde{\alpha}_1})\\
&=\sum_{i=0}^{n_f}\phi_i(x_{\dot{\beta}})\phi_i(x_{\tilde{\alpha}_1})=k_{\dot{\beta}\tilde{\alpha}_1}\,.
\end{aligned}
$$

# *Kernel* Methods

Multiplying the first and last expressions by the inverse submatrix $\widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_2}$, we get a *dual* representation

$$\left[\sum_{j,k=0}^{n_f} \phi_j(x_{\dot{\beta}}) \left(M^{-1}\right)_{jk} \phi_k(x_{\tilde{\alpha}_2})\right] = \sum_{\tilde{\alpha}_1 \in \mathcal{A}} k_{\dot{\beta}\tilde{\alpha}_1} \widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_2} \,,$$

which lets us rewrite the prediction of our linear model as

$$z_i(x_{\dot{\beta}}; \theta^\star) = \sum_{\tilde{\alpha}_1, \tilde{\alpha}_2 \in \mathcal{A}} k_{\dot{\beta}\tilde{\alpha}_1} \widetilde{k}^{\tilde{\alpha}_1\tilde{\alpha}_2} y_{i;\tilde{\alpha}_2} \,.$$

When the prediction of a linear model is computed in this way, it's known as a *kernel machine* or **kernel methods**.

# Course Plan

Lecture 1 ~~**Initialization, Linear Models**~~

- §0 + §7.1 + §10.4

Lecture 2 **Gradient Descent & Nearly-Kernel Methods**

- §7.2 + §∞.2.2 + §11.4

Lecture 3 **The Principle of Sparsity (Recursing)**

- §4, §8, §11.2, §∞.3

Lecture 4 **The Principle of Criticality**

- §5, §9, §11.3, §∞.1, +§10.3

Lecture 5 **The End of Training & More**

- §∞.2.3 + Maybe { §A, §B, . . . }