Modern Techniques

CISC 7026: Introduction to Deep Learning

University of Macau

Admin

Quiz 2 grades online, mean 3.5/4

Admin

Quiz 2 grades online, mean 3.5/4

Assignment 2 grades online, mean 82/100

Admin

Quiz 2 grades online, mean 3.5/4

Assignment 2 grades online, mean 82/100

Assignment 3 due next week

Admin

Quiz 2 grades online, mean 3.5/4

Assignment 2 grades online, mean 82/100

Assignment 3 due next week

Syllabus has optional reading for most lectures

Admin

Quiz 2 grades online, mean 3.5/4

Assignment 2 grades online, mean 82/100

Assignment 3 due next week

Syllabus has optional reading for most lectures

d2l.ai Section 5.1-5.5, 6.1-6.3, 12.1-12.10

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. Coding

Agenda

1. **Review**
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. Coding

Review

Many problems in ML can be reduced to **regression** or **classification**

Review

Many problems in ML can be reduced to **regression** or **classification**

Regression asks how many

Review

Many problems in ML can be reduced to **regression** or **classification**

Regression asks how many

- How long will I live?

Review

Many problems in ML can be reduced to **regression** or **classification**

Regression asks how many

- How long will I live?
- How much rain will there be tomorrow?

Review

Many problems in ML can be reduced to **regression** or **classification**

Regression asks how many

- How long will I live?
- How much rain will there be tomorrow?
- How far away is this object?

Review

Many problems in ML can be reduced to **regression** or **classification**

Regression asks how many

- How long will I live?
- How much rain will there be tomorrow?
- How far away is this object?

Classification asks which one

Review

Many problems in ML can be reduced to **regression** or **classification**

Regression asks how many

- How long will I live?
- How much rain will there be tomorrow?
- How far away is this object?

Classification asks which one

- Is this a dog or muffin?

Review

Many problems in ML can be reduced to **regression** or **classification**

Regression asks how many

- How long will I live?
- How much rain will there be tomorrow?
- How far away is this object?

Classification asks which one

- Is this a dog or muffin?
- Will it rain tomorrow? Yes or no?

Review

Many problems in ML can be reduced to **regression** or **classification**

Regression asks how many

- How long will I live?
- How much rain will there be tomorrow?
- How far away is this object?

Classification asks which one

- Is this a dog or muffin?
- Will it rain tomorrow? Yes or no?
- What color is this object?

Review

Many problems in ML can be reduced to **regression** or **classification**

Regression asks how many

- How long will I live?
- How much rain will there be tomorrow?
- How far away is this object?

Classification asks which one

- Is this a dog or muffin?
- Will it rain tomorrow? Yes or no?
- What color is this object?

So far, we only looked at regression. Now, let us look at classification

Review

Task: Given a picture of clothes, predict the text description

Review

Task: Given a picture of clothes, predict the text description

$X : \mathbb{Z}_{0,255}^{32\times 32}$

Review

Task: Given a picture of clothes, predict the text description

$X : \mathbb{Z}_{0,255}^{32\times 32}$

Y : {T-shirt, Trouser, Pullover, Dress, Coat,

Sandal, Shirt, Sneaker, Bag, Ankle boot}

Review

Task: Given a picture of clothes, predict the text description

$X : \mathbb{Z}_{0,255}^{32\times 32}$

Y : {T-shirt, Trouser, Pullover, Dress, Coat,

Sandal, Shirt, Sneaker, Bag, Ankle boot}

Approach: Learn $\boldsymbol{\theta}$ that produce **conditional probabilities**

Review

Task: Given a picture of clothes, predict the text description

$X : \mathbb{Z}_{0,255}^{32\times 32}$

Y : {T-shirt, Trouser, Pullover, Dress, Coat,

Sandal, Shirt, Sneaker, Bag, Ankle boot}

Approach: Learn $\boldsymbol{\theta}$ that produce **conditional probabilities**

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = P(\boldsymbol{y} \mid \boldsymbol{x}) = P\left(\begin{bmatrix} \text{T-Shirt} \\ \text{Trouser} \\ \vdots \end{bmatrix} \middle| \right. \quad \left. \right) = \begin{bmatrix} 0.2 \\ 0.01 \\ \vdots \end{bmatrix}$$

Review

If events A, B are not disjoint, they are **conditionally dependent**

Review

If events A, B are not disjoint, they are **conditionally dependent**

$$P(\text{cloud}) = 0.2, P(\text{rain}) = 0.1$$

Review

If events A, B are not disjoint, they are **conditionally dependent**

$$P(\text{cloud}) = 0.2, P(\text{rain}) = 0.1$$

$$P(\text{rain} \mid \text{cloud}) = 0.5$$

Review

If events A, B are not disjoint, they are **conditionally dependent**

$$P(\text{cloud}) = 0.2, P(\text{rain}) = 0.1$$

$$P(\text{rain} \mid \text{cloud}) = 0.5$$

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Review

If events A, B are not disjoint, they are **conditionally dependent**

$$P(\text{cloud}) = 0.2, P(\text{rain}) = 0.1$$

$$P(\text{rain} \mid \text{cloud}) = 0.5$$

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Walk outside

$$P(\text{Rain} \cap \text{Cloud}) = 0.1$$

$$P(\text{Cloud}) = 0.2$$

$$P(\text{Rain} \mid \text{Cloud}) = \frac{0.1}{0.2} = 0.5$$

Review

How can we represent a probability distribution for a neural network?

Review

How can we represent a probability distribution for a neural network?

$$\boldsymbol{v} = \left\{ \begin{bmatrix} v_1 \\ \vdots \\ v_{d_y} \end{bmatrix} \middle| \quad \sum_{i=1}^{d_y} v_i = 1; \quad v_i \in (0, 1) \right\}$$

Review

How can we represent a probability distribution for a neural network?

$$\boldsymbol{v} = \left\{ \begin{bmatrix} v_1 \\ \vdots \\ v_{d_y} \end{bmatrix} \middle| \quad \sum_{i=1}^{d_y} v_i = 1; \quad v_i \in (0, 1) \right\}$$

There is special notation for this vector, called the **simplex**

Review

How can we represent a probability distribution for a neural network?

$$\boldsymbol{v} = \left\{ \begin{bmatrix} v_1 \\ \vdots \\ v_{d_y} \end{bmatrix} \middle| \quad \sum_{i=1}^{d_y} v_i = 1; \quad v_i \in (0, 1) \right\}$$

There is special notation for this vector, called the **simplex**

$$\Delta^{d_y - 1}$$

Review

The simplex Δ^k is an $k-1$-dimensional triangle in k-dimensional space

Review

The simplex Δ^k is an $k-1$-dimensional triangle in k-dimensional space

It has only $k-1$ free variables, because $x_k = 1 - \sum_{i=1}^{k-1} x_i$

Review

The softmax function maps real numbers to the simplex (probabilities)

$$\text{softmax} : \mathbb{R}^k \mapsto \Delta^{k-1}$$

Review

The softmax function maps real numbers to the simplex (probabilities)

$$\text{softmax} : \mathbb{R}^k \mapsto \Delta^{k-1}$$

$$\text{softmax}\left(\begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix}\right) = \frac{e^{\boldsymbol{x}}}{\sum_{i=1}^{k} e^{x_i}} = \begin{bmatrix} \frac{e^{x_1}}{e^{x_1}+e^{x_2}+\ldots e^{x_k}} \\ \frac{e^{x_2}}{e^{x_1}+e^{x_2}+\ldots e^{x_k}} \\ \vdots \\ \frac{e^{x_k}}{e^{x_1}+e^{x_2}+\ldots e^{x_k}} \end{bmatrix}$$

Review

The softmax function maps real numbers to the simplex (probabilities)

$$\text{softmax} : \mathbb{R}^k \mapsto \Delta^{k-1}$$

$$\text{softmax}\left(\begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix}\right) = \frac{e^{\boldsymbol{x}}}{\sum_{i=1}^{k} e^{x_i}} = \begin{bmatrix} \frac{e^{x_1}}{e^{x_1}+e^{x_2}+\ldots e^{x_k}} \\ \frac{e^{x_2}}{e^{x_1}+e^{x_2}+\ldots e^{x_k}} \\ \vdots \\ \frac{e^{x_k}}{e^{x_1}+e^{x_2}+\ldots e^{x_k}} \end{bmatrix}$$

If we attach it to our model, we can output probabilities!

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \text{softmax}(\boldsymbol{\theta}^\top \overline{\boldsymbol{x}})$$

Review

Question: Why do we output probabilities instead of a binary values

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \begin{bmatrix} P\big(\text{Shirt} \mid \text{ }\big) \\ P\big(\text{Bag} \mid \text{ }\big) \\ \vdots \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.08 \\ \vdots \end{bmatrix}; \quad f(\boldsymbol{x}, \boldsymbol{\theta}) = \begin{bmatrix} 1 \\ 0 \\ \vdots \end{bmatrix}$$

Review

Question: Why do we output probabilities instead of a binary values

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \begin{bmatrix} P\big(\text{Shirt} \mid \text{ }\big) \\ P\big(\text{Bag} \mid \text{ }\big) \\ \vdots \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.08 \\ \vdots \end{bmatrix}; \quad f(\boldsymbol{x}, \boldsymbol{\theta}) = \begin{bmatrix} 1 \\ 0 \\ \vdots \end{bmatrix}$$

Answer 1: Outputting probabilities results in differentiable functions

Review

Question: Why do we output probabilities instead of a binary values

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \begin{bmatrix} P\big(\text{Shirt} \mid \text{ }\big) \\ P\big(\text{Bag} \mid \text{ }\big) \\ \vdots \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.08 \\ \vdots \end{bmatrix}; \quad f(\boldsymbol{x}, \boldsymbol{\theta}) = \begin{bmatrix} 1 \\ 0 \\ \vdots \end{bmatrix}$$

Answer 1: Outputting probabilities results in differentiable functions

Answer 2: We report uncertainty, which is useful in many applications

Review

Class Probability
Ankle Boot
Bag
Sneaker
Shirt
Sandal
Coat
Dress
Pullover
Trouser
T-shirt/top
0.00
0.25
0.50
0.75
1.00

Review

We consider the label $\boldsymbol{y}_{[i]}$ as a conditional distribution

$$P\left(\boldsymbol{y}_{[i]} \mid \boldsymbol{x}_{[i]}\right) = \begin{bmatrix} P\left(\text{Shirt} \mid \right. \\ P\left(\text{Bag} \mid \right. \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Review

We consider the label $\boldsymbol{y}_{[i]}$ as a conditional distribution

$$P\left(\boldsymbol{y}_{[i]} \mid \boldsymbol{x}_{[i]}\right) = \begin{bmatrix} P\left(\text{Shirt} \mid \right.\left.\right) \\ P\left(\text{Bag} \mid \right.\left.\right) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

We approximate this with our model $f(\boldsymbol{x}, \boldsymbol{\theta})$

Review

We consider the label $\boldsymbol{y}_{[i]}$ as a conditional distribution

$$P\left(\boldsymbol{y}_{[i]} \mid \boldsymbol{x}_{[i]}\right) = \begin{bmatrix} P\left(\text{Shirt} \mid \right. \\ P\left(\text{Bag} \mid \right. \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

We approximate this with our model $f(\boldsymbol{x}, \boldsymbol{\theta})$

Our loss function measures the difference between two distributions

Review

We consider the label $\boldsymbol{y}_{[i]}$ as a conditional distribution

$$P\left(\boldsymbol{y}_{[i]} \mid \boldsymbol{x}_{[i]}\right) = \begin{bmatrix} P\left(\text{Shirt} \mid \right. \\ P\left(\text{Bag} \mid \right. \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

We approximate this with our model $f(\boldsymbol{x}, \boldsymbol{\theta})$

Our loss function measures the difference between two distributions

$$P\left(\boldsymbol{y}_{[i]} \mid \boldsymbol{x}_{[i]}\right), \quad f(\boldsymbol{x}, \boldsymbol{\theta})$$

Review

The **Kullback-Leibler Divergence (KL)** measures the difference between distributions

Review

The **Kullback-Leibler Divergence (KL)** measures the difference between distributions

Review

From the KL divergence, we derived the **cross-entropy loss** function, which we use for classification

Review

From the KL divergence, we derived the **cross-entropy loss** function, which we use for classification

$$= -\sum_{i=1}^{d_y} P(y_i \mid \boldsymbol{x}) \log f(\boldsymbol{x}, \boldsymbol{\theta})_i$$

Review

From the KL divergence, we derived the **cross-entropy loss** function, which we use for classification

$$= -\sum_{i=1}^{d_y} P(y_i \mid \boldsymbol{x}) \log f(\boldsymbol{x}, \boldsymbol{\theta})_i$$

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}) = \left[-\sum_{j=1}^{n} \sum_{i=1}^{d_y} P\left(y_{[j],i} \mid \boldsymbol{x}_{[j]}\right) \log f\left(\boldsymbol{x}_{[j]}, \boldsymbol{\theta}\right)_i \right]$$

Review

Finish coding exercise

https://colab.research.google.com/drive/1BGMIE2CjlLJOH-D2r9AariPDVgxjWlqG#scrollTo=AnHP-PHVhpW_

Review

Agenda

1. **Review**
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. Coding

Agenda

1. Review
2. **Dirty Secret of Deep Learning**
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. Coding

Dirty Secret of Deep Learning

So far, I gave you the impression that deep learning is rigorous

Dirty Secret of Deep Learning

So far, I gave you the impression that deep learning is rigorous

Biological inspiration, theoretical bounds and mathematical guarantees

Dirty Secret of Deep Learning

So far, I gave you the impression that deep learning is rigorous

Biological inspiration, theoretical bounds and mathematical guarantees

For complex neural networks, deep learning is a **science** not **math**

Dirty Secret of Deep Learning

So far, I gave you the impression that deep learning is rigorous

Biological inspiration, theoretical bounds and mathematical guarantees

For complex neural networks, deep learning is a **science** not **math**

There is no widely-accepted theory for why deep neural networks are so effective

Dirty Secret of Deep Learning

So far, I gave you the impression that deep learning is rigorous

Biological inspiration, theoretical bounds and mathematical guarantees

For complex neural networks, deep learning is a **science** not **math**

There is no widely-accepted theory for why deep neural networks are so effective

In modern deep learning, we progress using trial and error

Dirty Secret of Deep Learning

So far, I gave you the impression that deep learning is rigorous

Biological inspiration, theoretical bounds and mathematical guarantees

For complex neural networks, deep learning is a **science** not **math**

There is no widely-accepted theory for why deep neural networks are so effective

In modern deep learning, we progress using trial and error

Today we experiment, and maybe tomorrow we discover the theory

Dirty Secret of Deep Learning

Scientific method:

Dirty Secret of Deep Learning

Scientific method:

1. Collect observations

Dirty Secret of Deep Learning

Scientific method:

1. Collect observations
2. Form hypothesis

Dirty Secret of Deep Learning

Scientific method:

1. Collect observations
2. Form hypothesis
3. Run experiment

Dirty Secret of Deep Learning

Scientific method:

1. Collect observations
2. Form hypothesis
3. Run experiment
4. Publish theory

Dirty Secret of Deep Learning

Scientific method:

1. Collect observations
2. Form hypothesis
3. Run experiment
4. Publish theory

However, there is a second part:

Dirty Secret of Deep Learning

Scientific method:

1. Collect observations
2. Form hypothesis
3. Run experiment
4. Publish theory

However, there is a second part:

1. Find existing theory

Dirty Secret of Deep Learning

Scientific method:

1. Collect observations
2. Form hypothesis
3. Run experiment
4. Publish theory

However, there is a second part:

1. Find existing theory
2. Show counterexample through experiment

Dirty Secret of Deep Learning

Scientific method:

1. Collect observations
2. Form hypothesis
3. Run experiment
4. Publish theory

However, there is a second part:

1. Find existing theory
2. Show counterexample through experiment
3. Publish counterexample

Dirty Secret of Deep Learning

Scientific method:

1. Collect observations
2. Form hypothesis
3. Run experiment
4. Publish theory

However, there is a second part:

1. Find existing theory
2. Show counterexample through experiment
3. Publish counterexample
4. Update or falsify theory

Deep learning is new, so much of part 2 has not happened yet!

Dirty Secret of Deep Learning

For many concepts, the **observations** are stronger than the **theory**

Dirty Secret of Deep Learning

For many concepts, the **observations** are stronger than the **theory**

Observe that a concept improves many types of neural networks

Dirty Secret of Deep Learning

For many concepts, the **observations** are stronger than the **theory**

Observe that a concept improves many types of neural networks

Concepts we discuss today are known to improve neural networks

Dirty Secret of Deep Learning

For many concepts, the **observations** are stronger than the **theory**

Observe that a concept improves many types of neural networks

Concepts we discuss today are known to improve neural networks

We will discuss theories about why they improve neural networks

Dirty Secret of Deep Learning

For many concepts, the **observations** are stronger than the **theory**

Observe that a concept improves many types of neural networks

Concepts we discuss today are known to improve neural networks

We will discuss theories about why they improve neural networks

If you do not believe the theory, prove it wrong and be famous!

Dirty Secret of Deep Learning

For many concepts, the **observations** are stronger than the **theory**

Observe that a concept improves many types of neural networks

Concepts we discuss today are known to improve neural networks

We will discuss theories about why they improve neural networks

If you do not believe the theory, prove it wrong and be famous!

Even if we do not agree on **why** a concept works, if we **observe** that it helps, we can still use it

Dirty Secret of Deep Learning

For many concepts, the **observations** are stronger than the **theory**

Observe that a concept improves many types of neural networks

Concepts we discuss today are known to improve neural networks

We will discuss theories about why they improve neural networks

If you do not believe the theory, prove it wrong and be famous!

Even if we do not agree on **why** a concept works, if we **observe** that it helps, we can still use it

This is how medicine works (e.g., Anesthetics)!

Dirty Secret of Deep Learning

Theoretical advancements are still very important!

Dirty Secret of Deep Learning

Theoretical advancements are still very important!

Imagine the 40 years of neural network research before we understood how to train neural networks

Dirty Secret of Deep Learning

Theoretical advancements are still very important!

Imagine the 40 years of neural network research before we understood how to train neural networks

Dirty Secret of Deep Learning

Theoretical advancements are still very important!

Imagine the 40 years of neural network research before we understood how to train neural networks

Are modern networks too complex for humans to understand?

Agenda

1. Review
2. **Dirty Secret of Deep Learning**
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. Coding

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. **Optimization is Hard**
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. Coding

Optimization is Hard

A 2-layer neural network can represent **any** continuous function to arbitrary precision

Optimization is Hard

A 2-layer neural network can represent **any** continuous function to arbitrary precision

$$\mid f(\boldsymbol{x}, \boldsymbol{\theta}) - g(\boldsymbol{x}) \mid < \varepsilon$$

Optimization is Hard

A 2-layer neural network can represent **any** continuous function to arbitrary precision

$$\mid f(\boldsymbol{x}, \boldsymbol{\theta}) - g(\boldsymbol{x}) \mid < \varepsilon$$

$$\lim_{d_h \to \infty} \varepsilon = 0$$

Optimization is Hard

A 2-layer neural network can represent **any** continuous function to arbitrary precision

$$\mid f(\boldsymbol{x}, \boldsymbol{\theta}) - g(\boldsymbol{x}) \mid < \varepsilon$$

$$\lim_{d_h \to \infty} \varepsilon = 0$$

However, finding such $\boldsymbol{\theta}$ is a much harder problem

Optimization is Hard

Gradient descent only guarantees convergence to a **local** optima

Optimization is Hard

Gradient descent only guarantees convergence to a **local** optima

Optimization is Hard

Harder tasks can have millions of local optima, and many of the local optima are not very good!

Optimization is Hard

Many of the concepts today create a **flat** loss landscape

Optimization is Hard

Many of the concepts today create a **flat** loss landscape

Optimization is Hard

Many of the concepts today create a **flat** loss landscape

Gradient descent reaches a better optimum more quickly in these cases

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. **Optimization is Hard**
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. Coding

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. **Deeper Neural Networks**
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. Coding

Deeper Neural Networks

A two-layer neural network is sufficient to approximate any continuous function to arbitrary precision

Deeper Neural Networks

A two-layer neural network is sufficient to approximate any continuous function to arbitrary precision

But only with infinite width $d_h \to \infty$

Deeper Neural Networks

A two-layer neural network is sufficient to approximate any continuous function to arbitrary precision

But only with infinite width $d_h \to \infty$

For certain problems, adding one more layer is equivalent to **exponentially** increasing the width

- Eldan, Ronen, and Ohad Shamir. "The power of depth for feedforward neural networks." Conference on learning theory. PMLR, 2016.

Deeper Neural Networks

A two-layer neural network is sufficient to approximate any continuous function to arbitrary precision

But only with infinite width $d_h \to \infty$

For certain problems, adding one more layer is equivalent to **exponentially** increasing the width

- Eldan, Ronen, and Ohad Shamir. "The power of depth for feedforward neural networks." Conference on learning theory. PMLR, 2016.

$$2 \times 32 \times 32 \Rightarrow 2^{2\times 32\times 32} \approx 10^{616}; \quad \text{universe has } 10^{80} \text{ atoms}$$

Deeper Neural Networks

A two-layer neural network is sufficient to approximate any continuous function to arbitrary precision

But only with infinite width $d_h \to \infty$

For certain problems, adding one more layer is equivalent to **exponentially** increasing the width

- Eldan, Ronen, and Ohad Shamir. "The power of depth for feedforward neural networks." Conference on learning theory. PMLR, 2016.

$$2 \times 32 \times 32 \Rightarrow 2^{2\times 32\times 32} \approx 10^{616}; \quad \text{universe has } 10^{80} \text{ atoms}$$

One more layer can solve this problem $3 \times 32 \times 32$

Deeper Neural Networks

A two-layer neural network is sufficient to approximate any continuous function to arbitrary precision

But only with infinite width $d_h \to \infty$

For certain problems, adding one more layer is equivalent to **exponentially** increasing the width

- Eldan, Ronen, and Ohad Shamir. "The power of depth for feedforward neural networks." Conference on learning theory. PMLR, 2016.

$$2 \times 32 \times 32 \Rightarrow 2^{2\times 32\times 32} \approx 10^{616}; \quad \text{universe has } 10^{80} \text{ atoms}$$

One more layer can solve this problem $3 \times 32 \times 32$

We need more layers for harder problems

Deeper Neural Networks

Low-level features

Edges, dark spots

Mid-level features

Eyes, ears, nose

High-level features

Facial structure

Deeper Neural Networks

In fact, we do not just need **deeper** networks, but also **wider** networks

Deeper Neural Networks

In fact, we do not just need **deeper** networks, but also **wider** networks

The number of neurons in a deep neural network affects the quality of local optima

Deeper Neural Networks

In fact, we do not just need **deeper** networks, but also **wider** networks

The number of neurons in a deep neural network affects the quality of local optima

From Choromanska, Anna, et al. *The loss surfaces of multilayer networks.* (2014):

Deeper Neural Networks

In fact, we do not just need **deeper** networks, but also **wider** networks

The number of neurons in a deep neural network affects the quality of local optima

From Choromanska, Anna, et al. *The loss surfaces of multilayer networks.* (2014):

- "For large-size networks, most local minima are equivalent and yield similar performance on a test set."

Deeper Neural Networks

In fact, we do not just need **deeper** networks, but also **wider** networks

The number of neurons in a deep neural network affects the quality of local optima

From Choromanska, Anna, et al. *The loss surfaces of multilayer networks.* (2014):

- "For large-size networks, most local minima are equivalent and yield similar performance on a test set."
- "The probability of finding a "bad" (high value) local minimum is non-zero for small-size networks and decreases quickly with network size"

Deeper Neural Networks

This is a difficult finding to conceptualize

Deeper Neural Networks

This is a difficult finding to conceptualize

The plots I show you are always 3D (2 parameters)

Deeper Neural Networks

This is a difficult finding to conceptualize

The plots I show you are always 3D (2 parameters)

For neural networks with 1,000,000+ parameters, our geometric intuition begins to fail

Deeper Neural Networks

This is a difficult finding to conceptualize

The plots I show you are always 3D (2 parameters)

For neural networks with 1,000,000+ parameters, our geometric intuition begins to fail

We understand bigger/deeper networks often perform better, but we are missing deeper theory

Deeper Neural Networks

This is a difficult finding to conceptualize

The plots I show you are always 3D (2 parameters)

For neural networks with 1,000,000+ parameters, our geometric intuition begins to fail

We understand bigger/deeper networks often perform better, but we are missing deeper theory

We call such networks **overparameterized** neural networks

Deeper Neural Networks

To summarize, deeper and wider neural networks tend to produce better results

Deeper Neural Networks

To summarize, deeper and wider neural networks tend to produce better results

Add more layers to your network

Deeper Neural Networks

To summarize, deeper and wider neural networks tend to produce better results

Add more layers to your network

Increase the width of each layer

Deeper Neural Networks

```python
# Deep neural network
from torch import nn

d_x, d_y, d_h = 1, 1, 16
# Linear(input, output)
l1 = nn.Linear(d_x, d_h)
l2 = nn.Linear(d_h, d_y)
```

Deeper Neural Networks

```python
# Deep neural network
from torch import nn

d_x, d_y, d_h = 1, 1, 16
# Linear(input, output)
l1 = nn.Linear(d_x, d_h)
l2 = nn.Linear(d_h, d_y)
```

```python
# Deeper and wider neural network
from torch import nn

d_x, d_y, d_h = 1, 1, 256
# Linear(input, output)
l1 = nn.Linear(d_x, d_h)
l2 = nn.Linear(d_h, d_h)
l3 = nn.Linear(d_h, d_h)
...
l6 = nn.Linear(d_h, d_y)
```

Deeper Neural Networks

```
import torch
d_x, d_y, d_h = 1, 1, 256
net = torch.nn.Sequential(
    torch.nn.Linear(d_x, d_h),
    torch.nn.Sigmoid(),
    torch.nn.Linear(d_h, d_h),
    torch.nn.Sigmoid(),
    ...
    torch.nn.Linear(d_h, d_y),
)

x = torch.ones((d_x,))
y = net(x)
```

Deeper Neural Networks

```
import jax, equinox
d_x, d_y, d_h = 1, 1, 256
net = equinox.nn.Sequential([
    equinox.nn.Linear(d_x, d_h),
    equinox.nn.Lambda(jax.nn.sigmoid),
    equinox.nn.Linear(d_h, d_h),
    equinox.nn.Lambda(jax.nn.sigmoid),
    ...
    equinox.nn.Linear(d_h, d_y),
])

x = jax.numpy.ones((d_x,))
y = net(x)
```

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. **Deeper Neural Networks**
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. Coding

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. **Activation Functions**
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. Coding

Activation Functions

The sigmoid function was the standard activation function until ~ 2012

Activation Functions

The sigmoid function was the standard activation function until ~ 2012

Activation Functions

The sigmoid function was the standard activation function until ~ 2012

In 2012, people realized that ReLU activation performed much better

Activation Functions

Activation Functions

The sigmoid function can result in a **vanishing gradient**

Activation Functions

The sigmoid function can result in a **vanishing gradient**

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})))$$

Activation Functions

The sigmoid function can result in a **vanishing gradient**

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})))$$

$$\nabla_{\boldsymbol{\theta}_1} f(\boldsymbol{x}, \boldsymbol{\theta}) = \nabla[\sigma](\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}}))) \cdot \nabla[\sigma](\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})) \cdot \nabla[\sigma](\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})$$

Activation Functions

The sigmoid function can result in a **vanishing gradient**

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})))$$

$$\nabla_{\boldsymbol{\theta}_1} f(\boldsymbol{x}, \boldsymbol{\theta}) = \underbrace{\nabla[\sigma](\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})))}_{<0.5} \cdot \underbrace{\nabla[\sigma](\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}}))}_{<0.5} \cdot \underbrace{\nabla[\sigma](\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})}_{<0.5}$$

Activation Functions

The sigmoid function can result in a **vanishing gradient**

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})))$$

$$\nabla_{\boldsymbol{\theta}_1} f(\boldsymbol{x}, \boldsymbol{\theta}) = \underbrace{\nabla[\sigma](\boldsymbol{\theta}_3^\top \sigma(\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})))}_{<0.5} \cdot \underbrace{\nabla[\sigma](\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}}))}_{<0.5} \cdot \underbrace{\nabla[\sigma](\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})}_{<0.5}$$

$$\nabla_{\boldsymbol{\theta}_1} f(\boldsymbol{x}, \boldsymbol{\theta}) \approx 0$$

Activation Functions

To fix the vanishing gradient, researchers use the **rectified linear unit (ReLU)**

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

Activation Functions

To fix the vanishing gradient, researchers use the **rectified linear unit (ReLU)**

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

Activation Functions

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

Looks nothing like a biological neuron

Activation Functions

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

Looks nothing like a biological neuron

However, it works much better than sigmoid in practice

Activation Functions

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

Looks nothing like a biological neuron

However, it works much better than sigmoid in practice

Via chain rule, gradient is $1 \cdot 1 \cdot 1 \ldots$ which does not vanish

Activation Functions

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

However, it works much better than sigmoid in practice

Via chain rule, gradient is $1 \cdot 1 \cdot 1 \ldots$ which does not vanish

If layers outputs one positive number, gradient will not vanish

Activation Functions

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

Via chain rule, gradient is $1 \cdot 1 \cdot 1 \ldots$ which does not vanish

If layers outputs one positive number, gradient will not vanish

Question: Any problems?

Activation Functions

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

If layers outputs one positive number, gradient will not vanish

Question: Any problems?

Answer: Zero gradient region!

Activation Functions

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

Question: Any problems?

Answer: Zero gradient region!

Neurons can get "stuck", always output 0

Activation Functions

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

Answer: Zero gradient region!

Neurons can get "stuck", always output 0

These neurons cannot recover, they are **dead neurons**

Activation Functions

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

These neurons cannot recover, they are **dead neurons**

Activation Functions

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

These neurons cannot recover, they are **dead neurons**

Training for longer results in more dead neurons

Activation Functions

$$\sigma(x) = \max(0, x)$$

$$\nabla\sigma(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

These neurons cannot recover, they are **dead neurons**

Training for longer results in more dead neurons

Dead neurons hurt your network!

Activation Functions

To fix dying neurons, use **leaky ReLU**

Activation Functions

To fix dying neurons, use **leaky ReLU**

$$\sigma(x) = \max(0.1x, x)$$

$$\nabla\sigma(x) = \begin{cases} 0.1 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

Activation Functions

To fix dying neurons, use **leaky ReLU**

$$\sigma(x) = \max(0.1x, x)$$

$$\nabla\sigma(x) = \begin{cases} 0.1 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

Activation Functions

To fix dying neurons, use **leaky ReLU**

$$\sigma(x) = \max(0.1x, x)$$

$$\nabla\sigma(x) = \begin{cases} 0.1 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$

Small negative slope allows dead neurons to recover

Activation Functions

There are other activation functions that are better than leaky ReLU

Activation Functions

There are other activation functions that are better than leaky ReLU

- Mish

Activation Functions

There are other activation functions that are better than leaky ReLU

- Mish
- Swish

Activation Functions

There are other activation functions that are better than leaky ReLU

- Mish
- Swish
- ELU

Activation Functions

There are other activation functions that are better than leaky ReLU

- Mish
- Swish
- ELU
- GeLU

Activation Functions

There are other activation functions that are better than leaky ReLU

- Mish
- Swish
- ELU
- GeLU
- SeLU

Activation Functions

There are other activation functions that are better than leaky ReLU

- Mish
- Swish
- ELU
- GeLU
- SeLU

Activation Functions

There are other activation functions that are better than leaky ReLU

- Mish
- Swish
- ELU
- GeLU
- SeLU

They are all very similar

Activation Functions

There are other activation functions that are better than leaky ReLU

- Mish
- Swish
- ELU
- GeLU
- SeLU

They are all very similar

I usually use leaky ReLU because it works well enough

Activation Functions

https://pytorch.org/docs/stable/nn.html#non-linear-activations-weighted-sum-nonlinearity

https://jax.readthedocs.io/en/latest/jax.nn.html#activation-functions

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. **Activation Functions**
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. Coding

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. **Parameter Initialization**
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. Coding

Parameter Initialization

Recall the gradient descent algorithm

```
function GRADIENT DESCENT(X, Y, ℒ, t, α)
    ▷ Randomly initialize parameters
    θ ← 𝒩(0, 1)
    for i ∈ 1...t do
        ▷ Compute the gradient of the loss
        J ← ∇_θ ℒ(X, Y, θ)
        ▷ Update the parameters using the negative gradient
        θ ← θ − αJ
    return θ
```

Parameter Initialization

Initial $\boldsymbol{\theta}$ is starting position for gradient descent

Parameter Initialization

Initial $\boldsymbol{\theta}$ is starting position for gradient descent

Parameter Initialization

Initial $\boldsymbol{\theta}$ is starting position for gradient descent

Pick $\boldsymbol{\theta}$ that results in good local minima

Parameter Initialization

Start simple, initialize all parameters to 0

$$\boldsymbol{\theta} = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}, \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}, \dots$$

Parameter Initialization

Start simple, initialize all parameters to 0

$$\boldsymbol{\theta} = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}, \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}, \dots$$

Question: Any issues?

Parameter Initialization

Start simple, initialize all parameters to 0

$$\boldsymbol{\theta} = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}, \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}, \dots$$

Question: Any issues?

Answer: For ReLU activation, the gradient will always be zero

$$\nabla_{\boldsymbol{\theta}_1} f = \nabla[\sigma](\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}}))\ \nabla[\sigma](\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})\ \overline{\boldsymbol{x}}$$

Parameter Initialization

Start simple, initialize all parameters to 0

$$\boldsymbol{\theta} = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}, \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}, \dots$$

Question: Any issues?

Answer: For ReLU activation, the gradient will always be zero

$$\nabla_{\boldsymbol{\theta}_1} f = \nabla[\sigma](\boldsymbol{\theta}_2^\top \sigma(\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}}))\ \nabla[\sigma](\boldsymbol{\theta}_1^\top \overline{\boldsymbol{x}})\ \overline{\boldsymbol{x}}$$

$$\nabla_{\boldsymbol{\theta}_1} f = \nabla[\sigma](\mathbf{0}^\top \sigma(\mathbf{0}^\top \overline{\boldsymbol{x}}))\ \nabla[\sigma](\mathbf{0}_1^\top \overline{\boldsymbol{x}})\ \overline{\boldsymbol{x}} = 0$$

Parameter Initialization

Ok, so initialize $\boldsymbol{\theta} = \mathbf{1}$

Parameter Initialization

Ok, so initialize $\boldsymbol{\theta} = \mathbf{1}$

$$\boldsymbol{\theta} = \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}, \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}, \dots$$

Parameter Initialization

Ok, so initialize $\boldsymbol{\theta} = \mathbf{1}$

$$\boldsymbol{\theta} = \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}, \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}, \dots$$

Question: Any issues?

Parameter Initialization

Ok, so initialize $\boldsymbol{\theta} = \mathbf{1}$

$$\boldsymbol{\theta} = \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}, \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}, \dots$$

Question: Any issues?

$$z_i = \sigma\left(\sum_{j=1}^{d_x} \theta_j \cdot \overline{x}_j\right) = \sigma\left(\sum_{j=1}^{d_x} \overline{x}_j\right)$$

Parameter Initialization

Ok, so initialize $\boldsymbol{\theta} = \mathbf{1}$

$$\boldsymbol{\theta} = \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}, \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}, \dots$$

Question: Any issues?

$$z_i = \sigma\left(\sum_{j=1}^{d_x} \theta_j \cdot \overline{x}_j\right) = \sigma\left(\sum_{j=1}^{d_x} \overline{x}_j\right)$$

All neurons in layer have the same gradient,and will receive the same update. Equivalent to a network of width 1, not good!

Parameter Initialization

We randomly initialize $\boldsymbol{\theta}$ to break symmetry

$$\boldsymbol{\theta} = \begin{bmatrix} -0.5 & \dots & 2 \\ \vdots & \ddots & \vdots \\ 0.1 & \dots & 0.6 \end{bmatrix}, \begin{bmatrix} 1.3 & \dots & 1.2 \\ \vdots & \ddots & \vdots \\ -0.8 & \dots & -1.1 \end{bmatrix}, \dots$$

Parameter Initialization

We randomly initialize $\boldsymbol{\theta}$ to break symmetry

$$\boldsymbol{\theta} = \begin{bmatrix} -0.5 & \dots & 2 \\ \vdots & \ddots & \vdots \\ 0.1 & \dots & 0.6 \end{bmatrix}, \begin{bmatrix} 1.3 & \dots & 1.2 \\ \vdots & \ddots & \vdots \\ -0.8 & \dots & -1.1 \end{bmatrix}, \dots$$

But what scale? If $\boldsymbol{\theta} \ll 0$ the gradient will vanish to zero, if $\boldsymbol{\theta} \gg 0$ the gradient explode to infinity

Parameter Initialization

We randomly initialize $\boldsymbol{\theta}$ to break symmetry

$$\boldsymbol{\theta} = \begin{bmatrix} -0.5 & \dots & 2 \\ \vdots & \ddots & \vdots \\ 0.1 & \dots & 0.6 \end{bmatrix}, \begin{bmatrix} 1.3 & \dots & 1.2 \\ \vdots & \ddots & \vdots \\ -0.8 & \dots & -1.1 \end{bmatrix}, \dots$$

But what scale? If $\boldsymbol{\theta} \ll 0$ the gradient will vanish to zero, if $\boldsymbol{\theta} \gg 0$ the gradient explode to infinity

Almost everyone initializes following a single paper from 2010:

Parameter Initialization

We randomly initialize $\boldsymbol{\theta}$ to break symmetry

$$\boldsymbol{\theta} = \begin{bmatrix} -0.5 & \dots & 2 \\ \vdots & \ddots & \vdots \\ 0.1 & \dots & 0.6 \end{bmatrix}, \begin{bmatrix} 1.3 & \dots & 1.2 \\ \vdots & \ddots & \vdots \\ -0.8 & \dots & -1.1 \end{bmatrix}, \dots$$

But what scale? If $\boldsymbol{\theta} \ll 0$ the gradient will vanish to zero, if $\boldsymbol{\theta} \gg 0$ the gradient explode to infinity

Almost everyone initializes following a single paper from 2010:

- Glorot, Xavier, and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks."

Parameter Initialization

We randomly initialize $\boldsymbol{\theta}$ to break symmetry

$$\boldsymbol{\theta} = \begin{bmatrix} -0.5 & \dots & 2 \\ \vdots & \ddots & \vdots \\ 0.1 & \dots & 0.6 \end{bmatrix}, \begin{bmatrix} 1.3 & \dots & 1.2 \\ \vdots & \ddots & \vdots \\ -0.8 & \dots & -1.1 \end{bmatrix}, \dots$$

But what scale? If $\boldsymbol{\theta} \ll 0$ the gradient will vanish to zero, if $\boldsymbol{\theta} \gg 0$ the gradient explode to infinity

Almost everyone initializes following a single paper from 2010:

- Glorot, Xavier, and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks."
- Maybe there are better options?

Parameter Initialization

Here is the magic equation, given the input and output size of the layer is d_h

$$\boldsymbol{\theta} \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{2d_h}}, \frac{\sqrt{6}}{\sqrt{2d_h}}\right]$$

Parameter Initialization

Here is the magic equation, given the input and output size of the layer is d_h

$$\boldsymbol{\theta} \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{2d_h}}, \frac{\sqrt{6}}{\sqrt{2d_h}}\right]$$

If you have different input or output sizes, such as d_x, d_y, then the equation is

$$\boldsymbol{\theta} \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{d_x + d_y}}, \frac{\sqrt{6}}{\sqrt{d_x + d_y}}\right]$$

Parameter Initialization

These equations are designed for ReLU and similar activation functions

Parameter Initialization

These equations are designed for ReLU and similar activation functions

They prevent vanishing or exploding gradients

Parameter Initialization

Usually `torch` and `jax/equinox` will automatically use this initialization when you create `nn.Linear`

```
layer = nn.Linear(d_x, d_h) # Uses Glorot init
```

You can find many initialization functions at https://pytorch.org/docs/stable/nn.init.html

For JAX it is https://jax.readthedocs.io/en/latest/jax.nn.initializers.html

Parameter Initialization

```python
import torch
d_h = 10
# Manually
theta = torch.zeros((d_h + 1, d_h))
torch.nn.init.xavier_uniform_(theta)
theta = torch.nn.Parameter(theta)

# Using nn.Linear
layer = torch.nn.Linear(d_h, d_h)
# Use .data, to bypass autograd security
torch.nn.init.xavier_uniform_(layer.weight.data)
torch.nn.init.xavier_uniform_(layer.bias.data) # Or zero
```

Parameter Initialization

```python
import jax
d_h = 10

init = jax.nn.initializers.glorot_uniform()
theta = init(jax.random.key(0), (d_h + 1, d_h))
```

Parameter Initialization

```
import jax, equinox
d_h = 10

layer = equinox.nn.Linear(d_h, d_h, key=jax.random.key(0))
# Create new bias and weight
init = jax.nn.initializers.glorot_uniform()
new_weight = init(jax.random.key(1), (d_h, d_h))
new_bias = init(jax.random.key(2), (d_h,))
# Use a lambda function to save space
# tree_at creates a new layer with the new weight
layer = equinox.tree_at(lambda l: l.weight, layer,
new_weight)
layer = equinox.tree_at(lambda l: l.bias, layer, new_weight)
```

Parameter Initialization

Remember, in `equinox` and `torch`, `nn.Linear` is already initialized correctly!

Parameter Initialization

Remember, in `equinox` and `torch`, `nn.Linear` is already initialized correctly!

In some cases, you may want to use different initializations

Parameter Initialization

Remember, in `equinox` and `torch`, `nn.Linear` is already initialized correctly!

In some cases, you may want to use different initializations

- Reinforcement learning

Parameter Initialization

Remember, in `equinox` and `torch`, `nn.Linear` is already initialized correctly!

In some cases, you may want to use different initializations

- Reinforcement learning
- Recurrent neural networks

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. **Parameter Initialization**
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. Coding

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. **Stochastic Gradient Descent**
8. Modern Optimization
9. Weight Decay
10. Coding

Stochastic Gradient Descent

```
function GRADIENT DESCENT(X, Y, ℒ, t, α)
    ▷ Randomly initialize parameters
    θ ← Glorot()
    for i ∈ 1...t do
        ▷ Compute the gradient of the loss
        J ← ∇_θ ℒ(X, Y, θ)
        ▷ Update the parameters using the negative gradient
        θ ← θ − α · J
    return θ
```

Stochastic Gradient Descent

```
function GRADIENT DESCENT(X, Y, ℒ, t, α)
    ▷ Randomly initialize parameters
    θ ← Glorot()
    for i ∈ 1...t do
        ▷ Compute the gradient of the loss
        J ← ∇_θ ℒ(X, Y, θ)
        ▷ Update the parameters using the negative gradient
        θ ← θ − α · J
    return θ
```

Gradient descent computes $\nabla \mathcal{L}$ over all $\boldsymbol{X}$

Stochastic Gradient Descent

This works for our small datasets, where $n = 1000$

Stochastic Gradient Descent

This works for our small datasets, where $n = 1000$

Question: How many GB are the LLM datasets?

Stochastic Gradient Descent

This works for our small datasets, where $n = 1000$

Question: How many GB are the LLM datasets?

Answer: About 774,000 GB according to *Datasets for Large Language Models: A Comprehensive Survey*

Stochastic Gradient Descent

This works for our small datasets, where $n = 1000$

Question: How many GB are the LLM datasets?

Answer: About 774,000 GB according to *Datasets for Large Language Models: A Comprehensive Survey*

This is just the dataset size, the gradient is orders of magnitude larger

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{x}_{[i]}, \boldsymbol{y}_{[i]}, \boldsymbol{\theta}\right) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_\ell}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_\ell}{\partial x_1} \end{bmatrix}_{[i]}$$

Stochastic Gradient Descent

Question: We do not have enough memory to compute the gradient. What can we do?

Stochastic Gradient Descent

Question: We do not have enough memory to compute the gradient. What can we do?

Answer: We approximate the gradient using a subset of the data

Stochastic Gradient Descent

Question: We do not have enough memory to compute the gradient. What can we do?

Answer: We approximate the gradient using a subset of the data

We call this **stochastic gradient descent** (SGD)

Stochastic Gradient Descent

function STOCHASTIC GRADIENT DESCENT($\boldsymbol{X}$, $\boldsymbol{Y}$, $\mathcal{L}$, t, α)

$\quad\boldsymbol{\theta} \leftarrow \text{Glorot}()$

$\quad$**for** $i \in 1 \ldots t$ **do**

$\quad\quad\boldsymbol{X}, \boldsymbol{Y} \leftarrow \text{Shuffle}(\boldsymbol{X}), \text{Shuffle}(\boldsymbol{Y})$

$\quad\quad$**for** $j \in 0 \ldots \frac{n}{B} - 1$ **do**

$\quad\quad\quad\boldsymbol{X}_j \leftarrow \begin{bmatrix} \boldsymbol{x}_{[jB]} & \boldsymbol{x}_{[jB+1]} & \cdots & \boldsymbol{x}_{[(j+1)B-1]} \end{bmatrix}^\top$

$\quad\quad\quad\boldsymbol{Y}_j \leftarrow \begin{bmatrix} \boldsymbol{y}_{[jB]} & \boldsymbol{y}_{[jB+1]} & \cdots & \boldsymbol{y}_{[(j+1)B-1]} \end{bmatrix}^\top$

$\quad\quad\quad\boldsymbol{J} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{X}_j, \boldsymbol{Y}_j, \boldsymbol{\theta}\right)$

$\quad\quad\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{J}$

$\quad$**return** $\boldsymbol{\theta}$

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is useful for saving memory

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is useful for saving memory

But it can also improve performance

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is useful for saving memory

But it can also improve performance

Since the “dataset” changes every update, so does the loss manifold

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is useful for saving memory

But it can also improve performance

Since the “dataset” changes every update, so does the loss manifold

This makes it less likely we get stuck in bad optima

Stochastic Gradient Descent

`torch.utils.data.DataLoader` helps with SGD

Stochastic Gradient Descent

`torch.utils.data.DataLoader` helps with SGD

```
import torch
dataloader = torch.utils.data.DataLoader(
    training_data,
    batch_size=32, # How many datapoints to sample
    shuffle=True, # Randomly shuffle each epoch
)
for epoch in number_of_epochs:
    for batch in dataloader:
        X_j, Y_j = batch
        loss = L(X_j, Y_j, theta)
        ...
```

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. **Stochastic Gradient Descent**
8. Modern Optimization
9. Weight Decay
10. Coding

Relax

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. **Modern Optimization**
9. Weight Decay
10. Coding

Modern Optimization

Gradient descent is a powerful tool, but it has issues

Modern Optimization

Gradient descent is a powerful tool, but it has issues

1. It can be slow to converge

Modern Optimization

Gradient descent is a powerful tool, but it has issues

1. It can be slow to converge
2. It can get stuck in poor local optima

Modern Optimization

Gradient descent is a powerful tool, but it has issues

1. It can be slow to converge
2. It can get stuck in poor local optima

Improve gradient descent convergence rate, while also preventing premature convergence

Modern Optimization

Gradient descent is a powerful tool, but it has issues

1. It can be slow to converge
2. It can get stuck in poor local optima

Improve gradient descent convergence rate, while also preventing premature convergence

It is hard to teach adaptive optimization solely through equations

Modern Optimization

Gradient descent is a powerful tool, but it has issues

1. It can be slow to converge
2. It can get stuck in poor local optima

Improve gradient descent convergence rate, while also preventing premature convergence

It is hard to teach adaptive optimization solely through equations

I want to show you a video to prepare you

https://www.youtube.com/watch?v=MD2fYip6QsQ&t=77s

Modern Optimization

The video simulations provide an intuitive understanding of adaptive optimizers

Modern Optimization

The video simulations provide an intuitive understanding of adaptive optimizers

The key behind modern optimizers is two concepts:

- Momentum

Modern Optimization

The video simulations provide an intuitive understanding of adaptive optimizers

The key behind modern optimizers is two concepts:

- Momentum
- Adaptive learning rate

Modern Optimization

The video simulations provide an intuitive understanding of adaptive optimizers

The key behind modern optimizers is two concepts:

- Momentum
- Adaptive learning rate

Let us discuss the algorithms more slowly

Modern Optimization

Review gradient descent again, because we will be making changes to it

Modern Optimization

Review gradient descent again, because we will be making changes to it

```
function GRADIENT DESCENT(X, Y, ℒ, t, α)
    ▷ Randomly initialize parameters
    θ ← Glorot()
    for i ∈ 1...t do
        ▷ Compute the gradient of the loss
        J ← ∇_θ ℒ(X, Y, θ)
        ▷ Update the parameters using the negative gradient
        θ ← θ − α · J
    return θ
```

Modern Optimization

Introduce **momentum** first

Modern Optimization

Introduce **momentum** first

```
function Momentum Gradient Descent(X, Y, ℒ, t, α, β)
    θ ← Glorot()
    M ← 0 # Init momentum
    for i ∈ 1...t do
        J ← ∇_θ ℒ(X, Y, θ) # Represents acceleration
        M ← β · M + (1 − β) · J # Momentum and acceleration
        θ ← θ − α · M
    return θ
```

Modern Optimization

Now **adaptive learning rate**

Modern Optimization

Now **adaptive learning rate**

function RMSProp$(\boldsymbol{X}, \boldsymbol{Y}, \mathcal{L}, t, \alpha, \beta, \varepsilon)$
    $\boldsymbol{\theta} \leftarrow \text{Glorot}()$
    $\boldsymbol{V} \leftarrow \boldsymbol{0}$ # Init variance
    for $i \in 1 \ldots t$ **do**
        $\boldsymbol{J} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$ # Represents acceleration
        $\boldsymbol{V} \leftarrow \beta \cdot \boldsymbol{V} + (1 - \beta) \cdot \boldsymbol{J} \odot \boldsymbol{J}$ # Magnitude
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \boldsymbol{J} \oslash \sqrt[\circ]{\boldsymbol{V} + \varepsilon}$ # Rescale grad by prev updates
    return $\boldsymbol{\theta}$

Modern Optimization

Combine **momentum** and **adaptive learning rate** to create **Adam**

Modern Optimization

Combine **momentum** and **adaptive learning rate** to create **Adam**

function ADAPTIVE MOMENT ESTIMATION($\boldsymbol{X}$, $\boldsymbol{Y}$, $\mathcal{L}$, t, α, β_1, β_2, ε)
    $\boldsymbol{\theta} \leftarrow \text{Glorot}()$
    $\boldsymbol{M}, \boldsymbol{V} \leftarrow \boldsymbol{0}$
    for $i \in 1 \ldots t$ **do**
        $\boldsymbol{J} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta})$
        $\boldsymbol{M} \leftarrow \beta_1 \cdot \boldsymbol{M} + (1 - \beta_1)\boldsymbol{J}$ # Compute momentum
        $\boldsymbol{V} \leftarrow \beta_2 \cdot \boldsymbol{V} + (1 - \beta_2) \cdot \boldsymbol{J} \odot \boldsymbol{J}$ # Magnitude
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \boldsymbol{M} \oslash \sqrt[\circ]{\boldsymbol{V} + \varepsilon}$ # Adaptive param update
    return $\boldsymbol{\theta}$ # Note, we use biased $\boldsymbol{M}, \boldsymbol{V}$ for clarity

Modern Optimization

```
import torch
betas = (0.9, 0.999)
net = ...
theta = net.parameters()

sgd = torch.optim.SGD(theta, lr=alpha)
momentum = torch.optim.SGD(
    theta, lr=alpha, momentum=betas[0])
rmsprop = torch.optim.RMSprop(
    theta, lr=alpha, momentum=betas[1])
adam = torch.optim.Adam(theta, lr=alpha, betas=betas)
...
sgd.step(), momentum.step(), rmsprop.step(), adam.step()
```

Modern Optimization

```
import optax
betas = (0.9, 0.999)
theta = ...

sgd = optax.sgd(lr=alpha)
momentum = optax.sgd(lr=alpha, momentum=betas[0])
rmsprop = optax.rmsprop(lr=alpha, decay=betas[1])
adam = optax.adam(lr=alpha, b1=betas[0], b2=betas[1])

v = rmsprop.init(theta)
theta, v = rmsprop.update(J, v, theta)
mv = adam.init(theta) # contains M and V
theta, mv = mv.update(J, mv, theta)
```

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. **Modern Optimization**
9. Weight Decay
10. Coding

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. **Weight Decay**
10. Coding

Weight Decay

Many modern optimizers also include **weight decay**

Weight Decay

Many modern optimizers also include **weight decay**

Weight decay penalizes large parameters

Weight Decay

Many modern optimizers also include **weight decay**

Weight decay penalizes large parameters

$$\mathcal{L}_{\text{decay}}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \mathcal{L}_{\text{decay}}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) + \lambda \sum_i \theta_i^2$$

Weight Decay

Many modern optimizers also include **weight decay**

Weight decay penalizes large parameters

$$\mathcal{L}_{\text{decay}}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \mathcal{L}_{\text{decay}}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) + \lambda \sum_i \theta_i^2$$

This results in a smoother, parabolic loss landscape that is easier to optimize

Weight Decay

Many modern optimizers also include **weight decay**

Weight decay penalizes large parameters

$$\mathcal{L}_{\text{decay}}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) = \mathcal{L}_{\text{decay}}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\theta}) + \lambda \sum_i \theta_i^2$$

This results in a smoother, parabolic loss landscape that is easier to optimize

(Unproven opinion) It reduces the number of saddle points and local minima

Weight Decay

$$z = \sin^2(x) + \sin^2(y)$$

Weight Decay

Weight decay is built into the Adam optimizer

Weight Decay

Weight decay is built into the Adam optimizer

But the implementation is wrong!

Weight Decay

Weight decay is built into the Adam optimizer

But the implementation is wrong!

Years later, someone noticed and published a new paper that fixes weight decay implementation

Weight Decay

Weight decay is built into the Adam optimizer

But the implementation is wrong!

Years later, someone noticed and published a new paper that fixes weight decay implementation

```
Lambda = 0.01
opt = torch.optim.adamw(..., weight_decay=Lambda)
opt = optax.adamw(..., weight_decay=Lambda)
```

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. **Weight Decay**
10. Coding

Agenda

1. Review
2. Dirty Secret of Deep Learning
3. Optimization is Hard
4. Deeper Neural Networks
5. Activation Functions
6. Parameter Initialization
7. Stochastic Gradient Descent
8. Modern Optimization
9. Weight Decay
10. **Coding**

Coding

https://colab.research.google.com/drive/1qTNSvB_JEMnMJfcAwsLJTuIlfxa_kyTD#scrollTo=YVkCyz78x4Rp