

# Diving Into Deep Learning: Part 1 – A Deep Learning Crash Course

**Elizabeth Newman**  
Emory University

# What Is Deep Learning?

Let's ask Chat GPT!

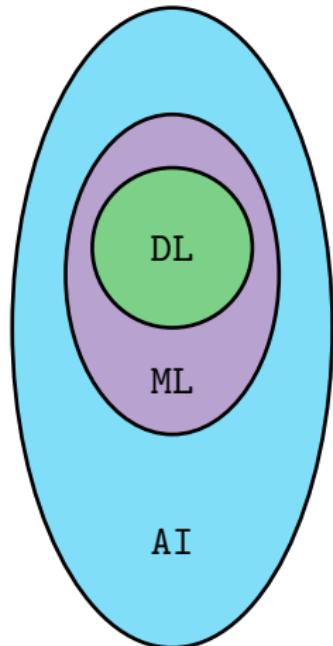


*Deep learning is a subfield of machine learning and artificial intelligence (AI) that focuses on the development of **artificial neural networks** that are capable of learning and making predictions or decisions on their own. The term "deep" in deep learning refers to the fact that these neural networks are typically designed with multiple layers, allowing them to learn representations of data at different levels of abstraction.*

*Deep learning algorithms learn from large amounts of labeled data, and their architecture allows them to automatically extract features and patterns from raw data, such as images, speech, text, and other types of data. These learned representations can then be used for tasks such as **image and speech recognition, natural language processing, autonomous driving, and many other applications.***

*Deep learning has revolutionized many fields, including computer vision, speech recognition, natural language processing, and robotics, among others. It has achieved state-of-the-art performance in various tasks, surpassing traditional machine learning approaches in many cases. Deep learning has also been **enabled by advances in computing power, particularly in graphics processing units (GPUs) and distributed computing, which have made it possible to train large neural networks on vast amounts of data.***

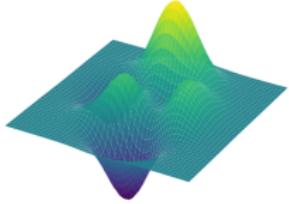
*Overall, deep learning has become a powerful and widely used approach in modern AI, allowing machines to automatically **learn complex patterns and make predictions or decisions with high accuracy, opening up new possibilities for applications across numerous domains.***



# What Can Deep Neural Networks Do?



# What Can Deep Neural Networks Do?



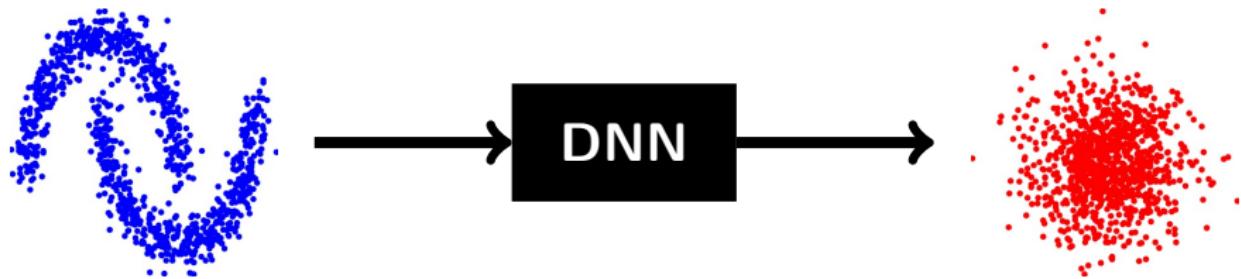
# What Can Deep Neural Networks Do?



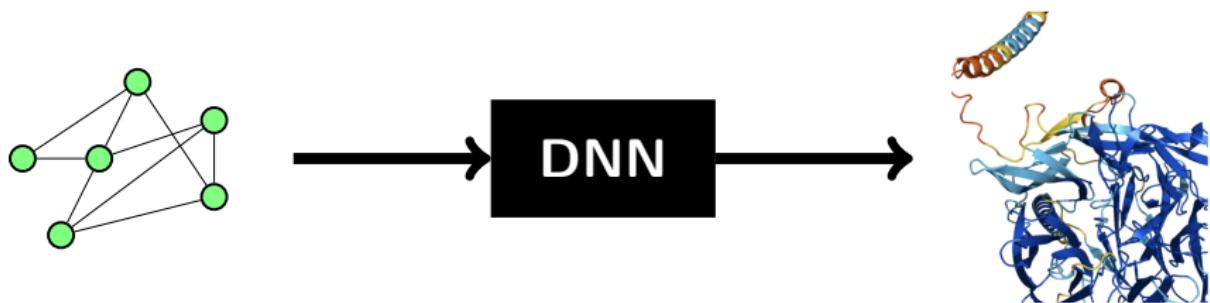
# What Can Deep Neural Networks Do?



# What Can Deep Neural Networks Do?



# What Can Deep Neural Networks Do?



# The Goal of Deep Learning

Find a parameterized mapping  $F(\mathbf{y}, \boldsymbol{\theta}) \approx \mathbf{c}$  for all input-target pairs  $(\mathbf{y}, \mathbf{c})$ .

# The Goal of Deep Learning

Find a parameterized mapping  $F(\mathbf{y}, \boldsymbol{\theta}) \approx \mathbf{c}$  for all input-target pairs  $(\mathbf{y}, \mathbf{c})$ .

## The Learning Problem

$$\min_{\boldsymbol{\theta}} \mathbb{E} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$

# The Goal of Deep Learning

Find a parameterized mapping  $F(\mathbf{y}, \boldsymbol{\theta}) \approx \mathbf{c}$  for all input-target pairs  $(\mathbf{y}, \mathbf{c})$ .

## The Training Problem

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T}} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$

## The Learning Problem

$$\min_{\boldsymbol{\theta}} \mathbb{E} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$

# The Goal of Deep Learning

Find a parameterized mapping  $F(\mathbf{y}, \boldsymbol{\theta}) \approx \mathbf{c}$  for all input-target pairs  $(\mathbf{y}, \mathbf{c})$ .

## The Training Problem

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T}} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$

## The Learning Problem

$$\min_{\boldsymbol{\theta}} \mathbb{E} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$

$$F(\cdot, \hat{\boldsymbol{\theta}}_{\mathcal{T}})$$



# The Goal of Deep Learning

Find a parameterized mapping  $F(\mathbf{y}, \boldsymbol{\theta}) \approx \mathbf{c}$  for all input-target pairs  $(\mathbf{y}, \mathbf{c})$ .

## The Training Problem

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T}} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$

## The Learning Problem

$$\min_{\boldsymbol{\theta}} \mathbb{E} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$



# The Goal of Deep Learning

Find a parameterized mapping  $F(\mathbf{y}, \boldsymbol{\theta}) \approx \mathbf{c}$  for all input-target pairs  $(\mathbf{y}, \mathbf{c})$ .

## The Training Problem

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T}} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$

## The Learning Problem

$$\min_{\boldsymbol{\theta}} \mathbb{E} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$



# The Goal of Deep Learning

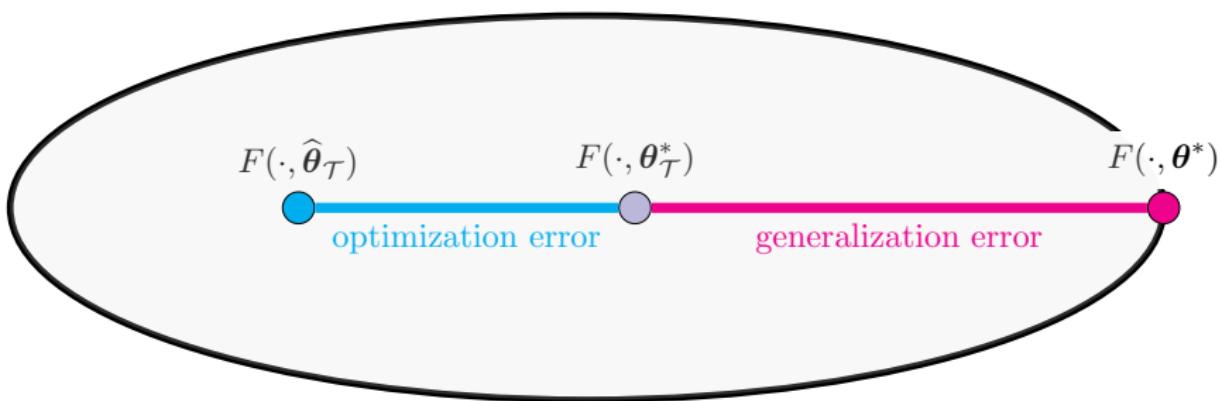
Find a parameterized mapping  $F(\mathbf{y}, \boldsymbol{\theta}) \approx \mathbf{c}$  for all input-target pairs  $(\mathbf{y}, \mathbf{c})$ .

## The Training Problem

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T}} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$

## The Learning Problem

$$\min_{\boldsymbol{\theta}} \mathbb{E} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$



Bottou, Curtis, and Nocedal, "Optimization Methods for Large-Scale Machine Learning"; Lu et al., "DeepXDE: A deep learning library for solving differential equations"

# The Goal of Deep Learning

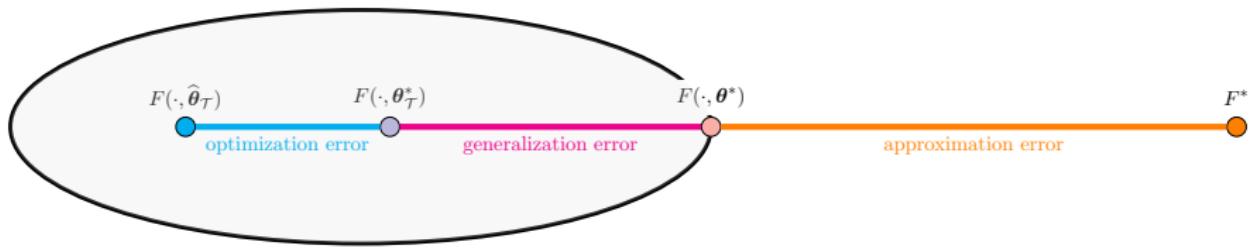
Find a parameterized mapping  $F(\mathbf{y}, \boldsymbol{\theta}) \approx \mathbf{c}$  for all input-target pairs  $(\mathbf{y}, \mathbf{c})$ .

## The Training Problem

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T}} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$

## The Learning Problem

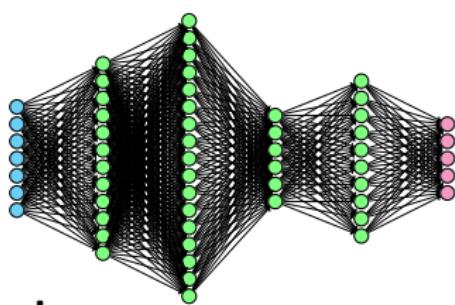
$$\min_{\boldsymbol{\theta}} \mathbb{E} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) + R(\boldsymbol{\theta})$$



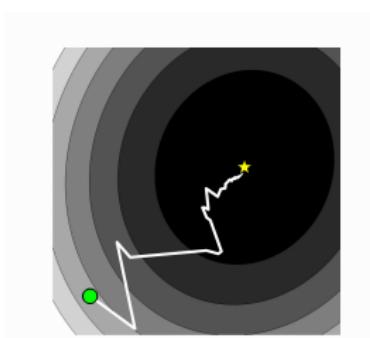
Bottou, Curtis, and Nocedal, "Optimization Methods for Large-Scale Machine Learning"; Lu et al., "DeepXDE: A deep learning library for solving differential equations"

# A Recipe for Deep Learning

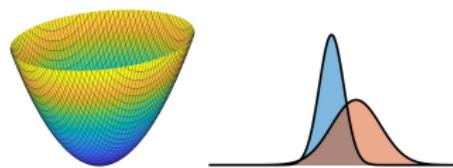
## Architecture



## Optimizer



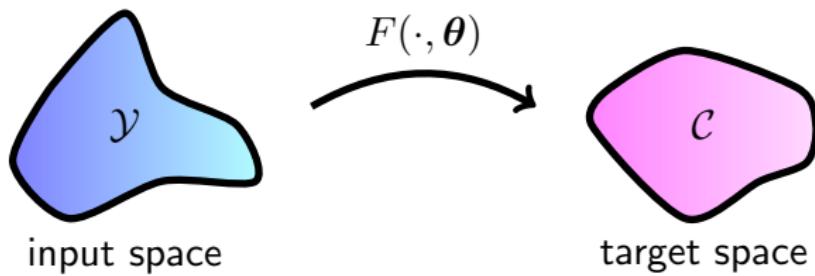
## Objective Function



## Data



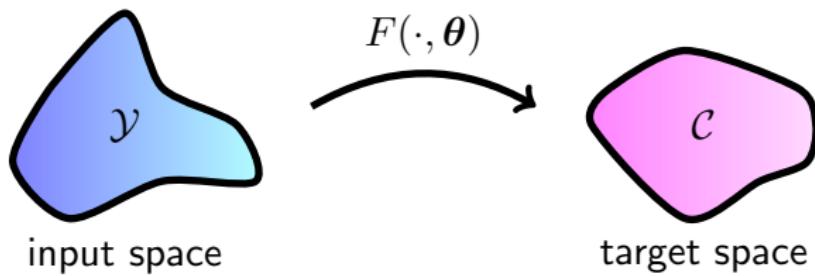
# Architectures 101



A **neural network**  $F : \mathcal{Y} \times \Theta \rightarrow \mathcal{C}$  is a **composite function**

$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d(f_{d-1}(\cdots(f_2(f_1(\mathbf{y}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2), \cdots), \boldsymbol{\theta}_{d-1}), \boldsymbol{\theta}_d)$$

# Architectures 101

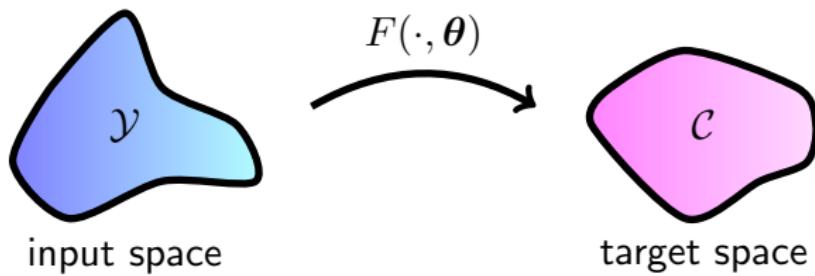


A **neural network**  $F : \mathcal{Y} \times \Theta \rightarrow \mathcal{C}$  is a **composite function**

$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d(f_{d-1}(\cdots(f_2(f_1(\mathbf{y}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2), \cdots), \boldsymbol{\theta}_{d-1}), \boldsymbol{\theta}_d)$$

- **weights:**  $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_d)$  lying in parameter space  $\Theta$

# Architectures 101

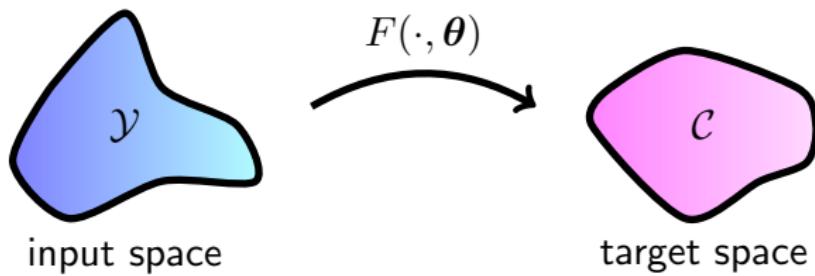


A **neural network**  $F : \mathcal{Y} \times \Theta \rightarrow \mathcal{C}$  is a **composite function**

$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d(f_{d-1}(\cdots(f_2(f_1(\mathbf{y}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2), \cdots), \boldsymbol{\theta}_{d-1}), \boldsymbol{\theta}_d)$$

- **weights:**  $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_d)$  lying in parameter space  $\Theta$
- **layers:**  $f_i : \mathbb{R}^{n_i} \times \Theta_i \rightarrow \mathbb{R}^{n_{i+1}}$  **deep** = many layers

# Architectures 101

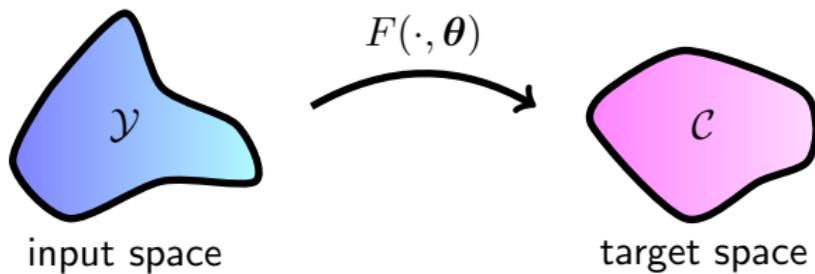


A **neural network**  $F : \mathcal{Y} \times \Theta \rightarrow \mathcal{C}$  is a **composite function**

$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d(f_{d-1}(\cdots(f_2(f_1(\mathbf{y}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2), \cdots), \boldsymbol{\theta}_{d-1}), \boldsymbol{\theta}_d)$$

- **weights:**  $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_d)$  lying in parameter space  $\Theta$
  - **layers:**  $f_i : \mathbb{R}^{n_i} \times \Theta_i \rightarrow \mathbb{R}^{n_{i+1}}$
  - **input features:**  $\mathbf{y} \in \mathcal{Y} \subseteq \mathbb{R}^n$
- deep** = many layers

# Architectures 101

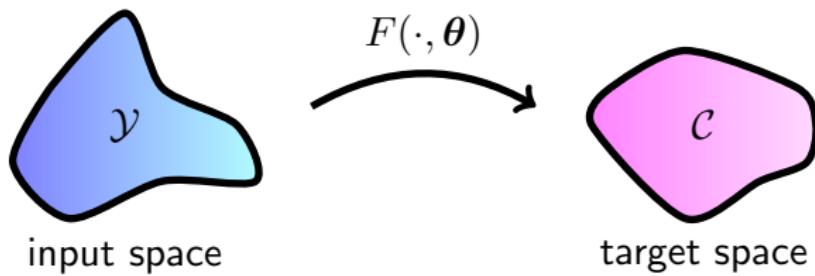


A **neural network**  $F : \mathcal{Y} \times \Theta \rightarrow \mathcal{C}$  is a **composite function**

$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d(f_{d-1}(\cdots(f_2(f_1(\mathbf{y}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2), \cdots), \boldsymbol{\theta}_{d-1}), \boldsymbol{\theta}_d)$$

- **weights:**  $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_d)$  lying in parameter space  $\Theta$
- **layers:**  $f_i : \mathbb{R}^{n_i} \times \Theta_i \rightarrow \mathbb{R}^{n_{i+1}}$  **deep** = many layers
- **input features:**  $\mathbf{y} \in \mathcal{Y} \subseteq \mathbb{R}^n$
- **hidden features:**  $\mathbf{u}_i = f_i(\mathbf{u}_{i-1}, \boldsymbol{\theta}_i)$  for  $i = 1, \dots, d$  with  $\mathbf{u}_0 = \mathbf{y}$ .

# Architectures 101



A **neural network**  $F : \mathcal{Y} \times \Theta \rightarrow \mathcal{C}$  is a **composite function**

$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d(f_{d-1}(\cdots(f_2(f_1(\mathbf{y}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2), \cdots), \boldsymbol{\theta}_{d-1}), \boldsymbol{\theta}_d)$$

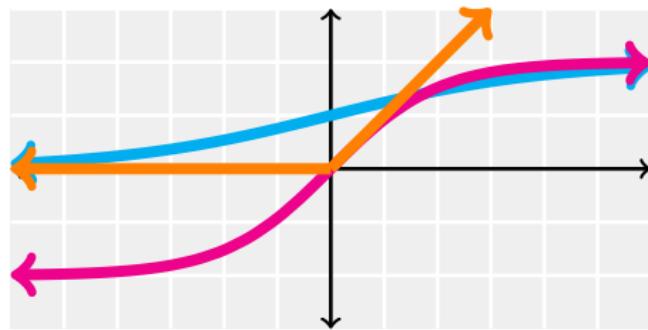
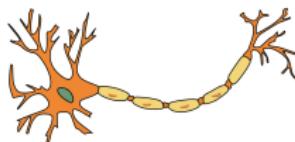
- **weights:**  $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_d)$  lying in parameter space  $\Theta$
- **layers:**  $f_i : \mathbb{R}^{n_i} \times \Theta_i \rightarrow \mathbb{R}^{n_{i+1}}$  **deep** = many layers
- **input features:**  $\mathbf{y} \in \mathcal{Y} \subseteq \mathbb{R}^n$
- **hidden features:**  $\mathbf{u}_i = f_i(\mathbf{u}_{i-1}, \boldsymbol{\theta}_i)$  for  $i = 1, \dots, d$  with  $\mathbf{u}_0 = \mathbf{y}$ .
- **target features:**  $\mathbf{c} \in \mathcal{C} \subseteq \mathbb{R}^m$

# A Biological Perspective on Layers

$$f_j(\mathbf{u}_{j-1}, \underbrace{(\mathbf{K}_j, \mathbf{b}_j)}_{\theta_j}) = \sigma(\mathbf{K}_j \mathbf{u}_{j-1} + \mathbf{b}_j)$$

affine transformation + activation

- **weight** matrix  $\mathbf{K}_j$
- **bias** vector  $\mathbf{b}_j$
- **activation function**  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$   
(pointwise)



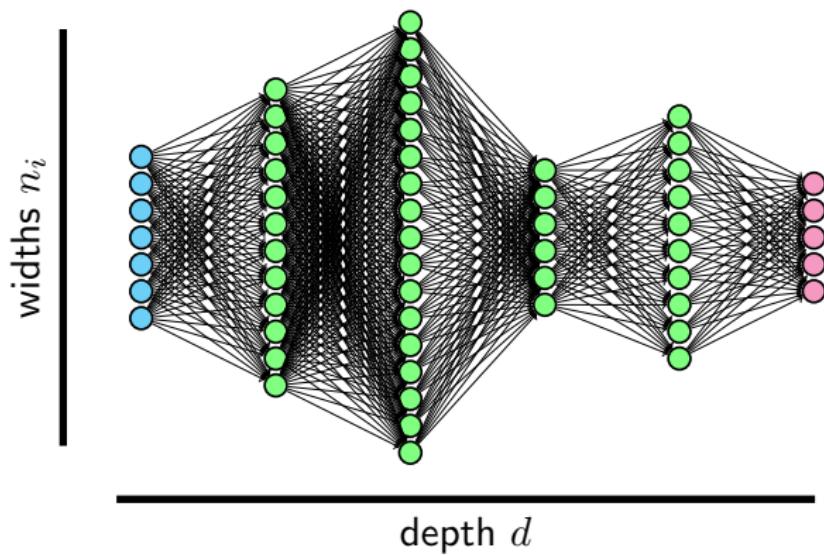
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\sigma(z) = \tanh(z)$$

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

# Putting It Together

$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d(f_{d-1}(\cdots(f_2(f_1(\mathbf{y}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2), \cdots), \boldsymbol{\theta}_{d-1}), \boldsymbol{\theta}_d)$$



# Universal Approximation

## Theorem (slightly simplified)

Let  $f : \mathcal{C}([0, 1]^n) \rightarrow \mathbb{R}$  and let  $\sigma$  be a sigmoidal function. Given  $\varepsilon > 0$ , there exists a dimension  $m \in \mathbb{N}$  such that for  $\mathbf{K} \in \mathbb{R}^{m \times n}$  and  $\mathbf{w}, \mathbf{b} \in \mathbb{R}^m$ , we have

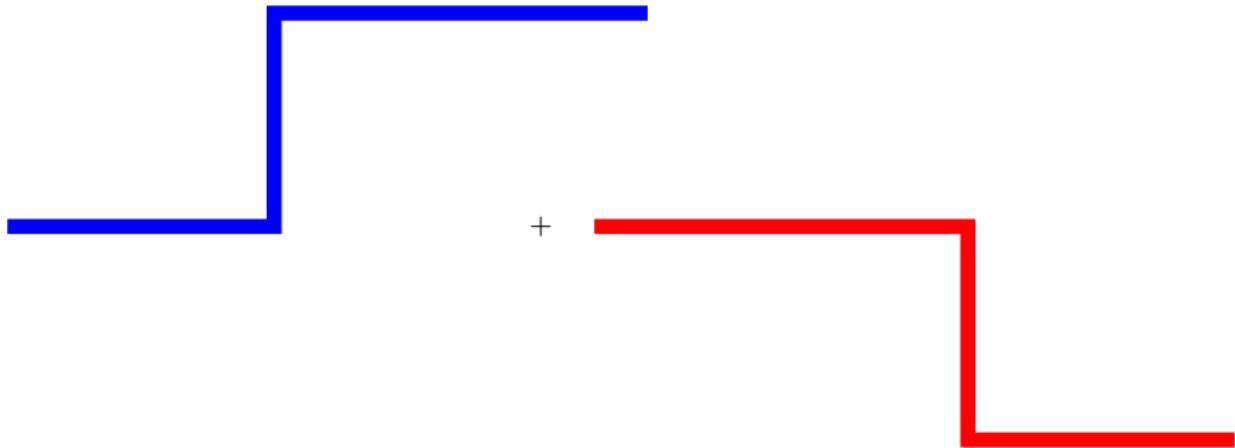
$$\left| \mathbf{w}^\top \sigma(\mathbf{K}\mathbf{x} + \mathbf{b}) - f(\mathbf{x}) \right| < \varepsilon \quad \text{for all } \mathbf{x} \in [0, 1]^n.$$

# Universal Approximation

## Theorem (slightly simplified)

Let  $f : \mathcal{C}([0, 1]^n) \rightarrow \mathbb{R}$  and let  $\sigma$  be a sigmoidal function. Given  $\varepsilon > 0$ , there exists a dimension  $m \in \mathbb{N}$  such that for  $\mathbf{K} \in \mathbb{R}^{m \times n}$  and  $\mathbf{w}, \mathbf{b} \in \mathbb{R}^m$ , we have

$$\left| \mathbf{w}^\top \sigma(\mathbf{K}\mathbf{x} + \mathbf{b}) - f(\mathbf{x}) \right| < \varepsilon \quad \text{for all } \mathbf{x} \in [0, 1]^n.$$

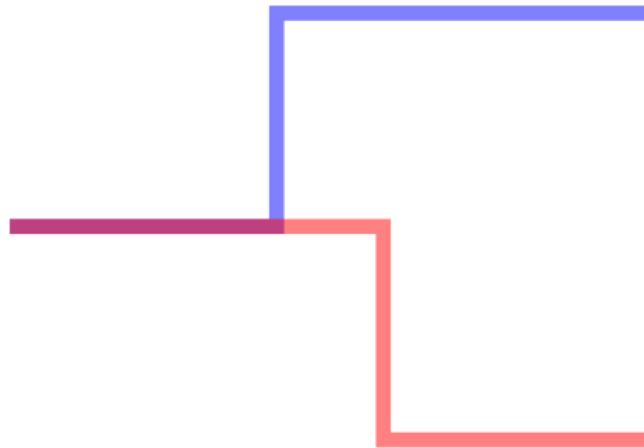


# Universal Approximation

## Theorem (slightly simplified)

Let  $f : \mathcal{C}([0, 1]^n) \rightarrow \mathbb{R}$  and let  $\sigma$  be a sigmoidal function. Given  $\varepsilon > 0$ , there exists a dimension  $m \in \mathbb{N}$  such that for  $\mathbf{K} \in \mathbb{R}^{m \times n}$  and  $\mathbf{w}, \mathbf{b} \in \mathbb{R}^m$ , we have

$$\left| \mathbf{w}^\top \sigma(\mathbf{K}\mathbf{x} + \mathbf{b}) - f(\mathbf{x}) \right| < \varepsilon \quad \text{for all } \mathbf{x} \in [0, 1]^n.$$

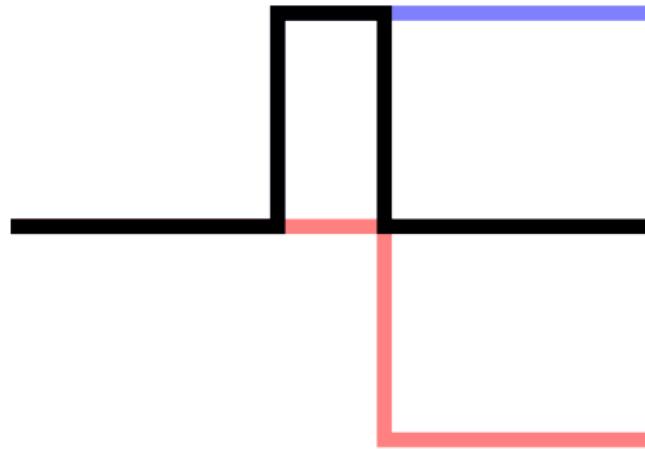


# Universal Approximation

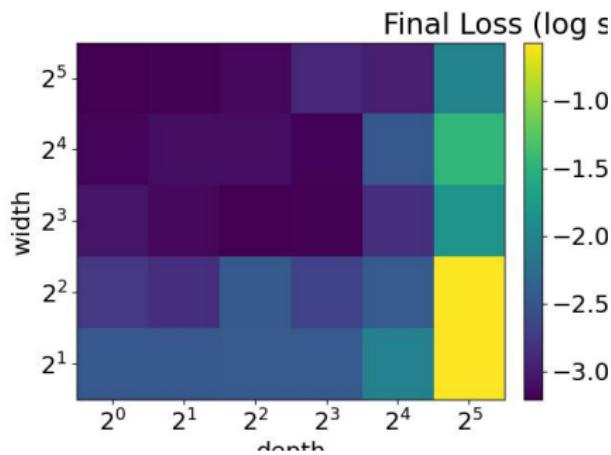
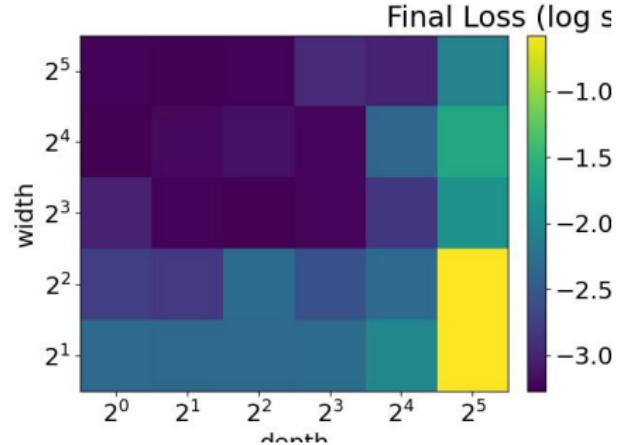
## Theorem (slightly simplified)

Let  $f : \mathcal{C}([0, 1]^n) \rightarrow \mathbb{R}$  and let  $\sigma$  be a sigmoidal function. Given  $\varepsilon > 0$ , there exists a dimension  $m \in \mathbb{N}$  such that for  $\mathbf{K} \in \mathbb{R}^{m \times n}$  and  $\mathbf{w}, \mathbf{b} \in \mathbb{R}^m$ , we have

$$\left| \mathbf{w}^\top \sigma(\mathbf{K}\mathbf{x} + \mathbf{b}) - f(\mathbf{x}) \right| < \varepsilon \quad \text{for all } \mathbf{x} \in [0, 1]^n.$$



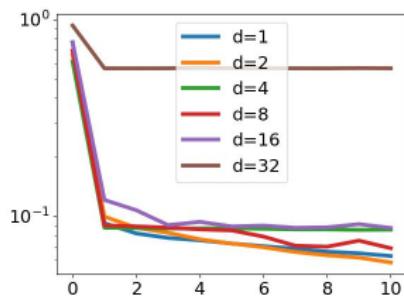
# Width vs. Depth

**Final Training Loss****Final Validation Loss**

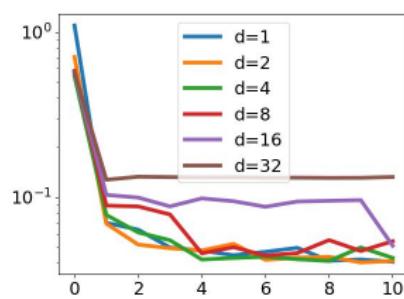
Setup:  $F : \mathbb{R} \times \Theta \rightarrow \mathbb{R}$ , fully-connected, trained with Adam for fixed number of epochs.

# Width vs. Depth

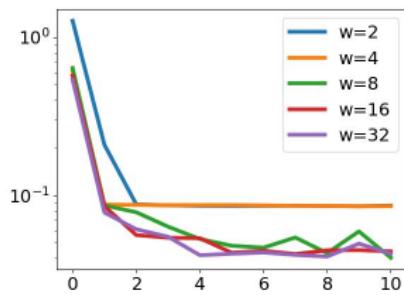
width = 4



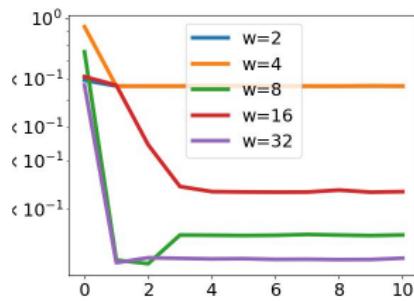
width = 32



depth = 4



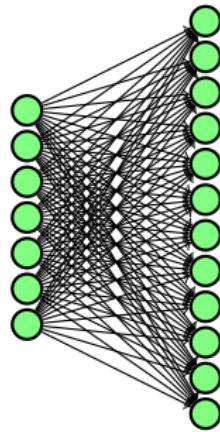
depth = 32



# Many Types of Layers

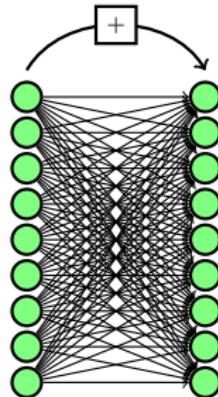
## Fully-Connected

$$\mathbf{u}_j = \sigma(\mathbf{K}_j \mathbf{u}_{j-1} + \mathbf{b}_j)$$



## Residual Layer

$$\mathbf{u}_j = \mathbf{u}_{j-1} + \sigma(\mathbf{K}_j \mathbf{u}_{j-1} + \mathbf{b}_j)$$



Krizhevsky, Sutskever, and Hinton, "ImageNet Classification with Deep Convolutional Neural Networks"; He et al., "Deep Residual Learning for Image Recognition"

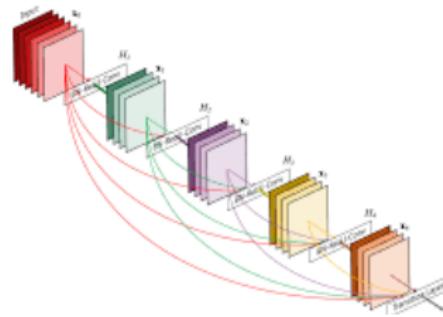
# Many Types of Layers

## Convolutional Layers

$$\mathbf{u}_j = \sigma(\mathbf{K}(\theta_j)\mathbf{u}_{j-1} + \mathbf{b}_j)$$

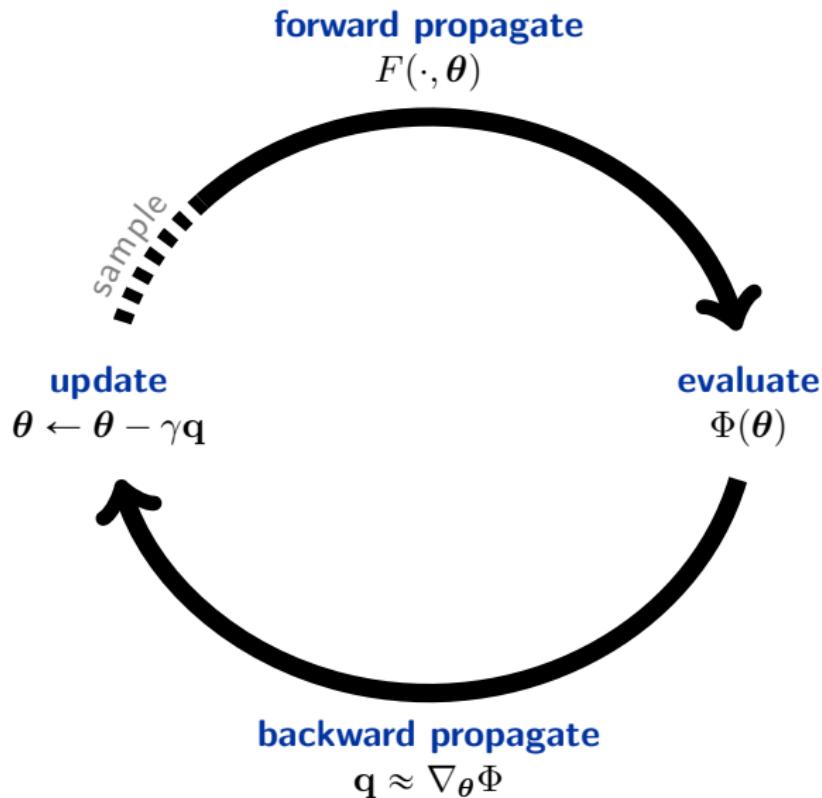
## Dense Layers

$$\mathbf{u}_j = f(\mathbf{u}_{j-1}, \mathbf{u}_{j-2}, \dots, \mathbf{u}_0, \theta)$$

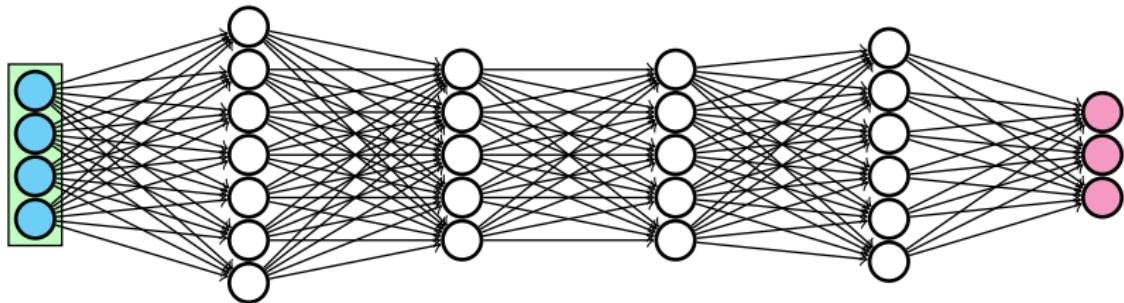


Krizhevsky, Sutskever, and Hinton, "ImageNet Classification with Deep Convolutional Neural Networks"; Huang et al., "Densely Connected Convolutional Networks"

# The Training Cycle



# Forward Propagation: Composition

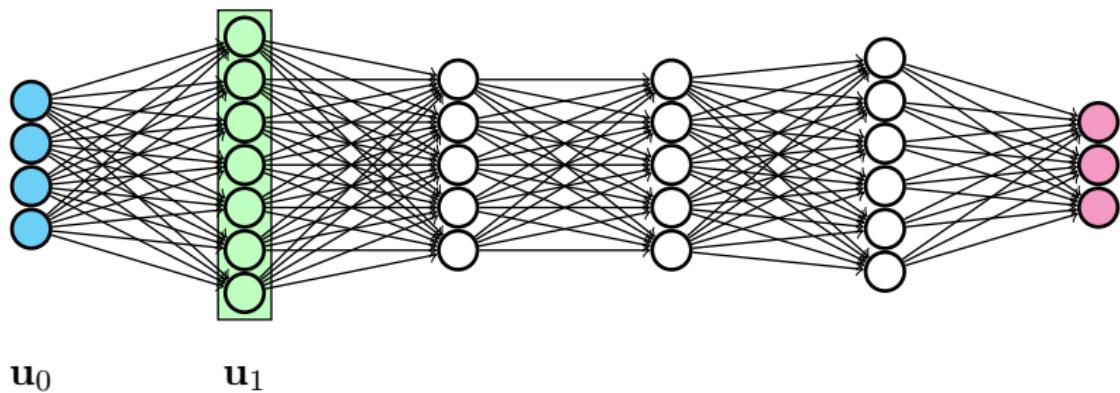


$\mathbf{u}_0$

$\mathbf{y}$

$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d ( f_{d-1} ( \cdots ( f_2 ( f_1 (\mathbf{y}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2 ), \cdots ), \boldsymbol{\theta}_{d-1} ), \boldsymbol{\theta}_d )$$

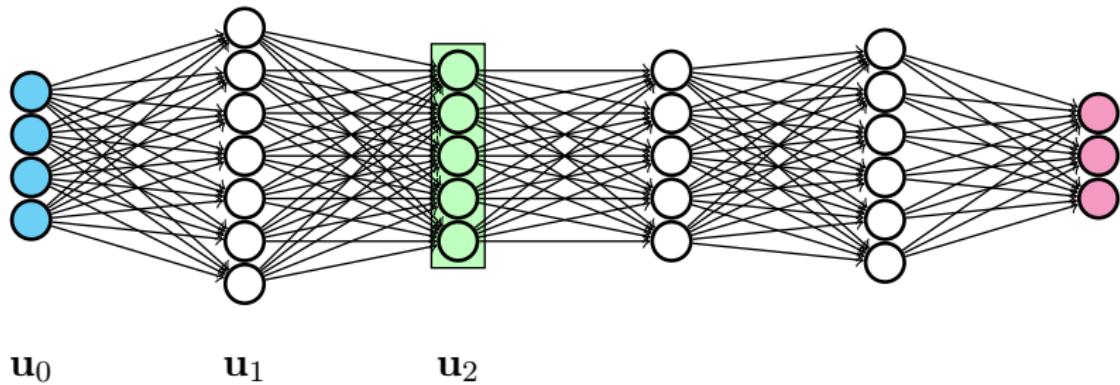
# Forward Propagation: Composition



$$\mathbf{y} \quad f_1(\mathbf{u}_0, \boldsymbol{\theta}_1)$$

$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d ( f_{d-1} ( \cdots ( f_2 ( f_1 (\mathbf{y}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2 ), \cdots ), \boldsymbol{\theta}_{d-1} ), \boldsymbol{\theta}_d )$$

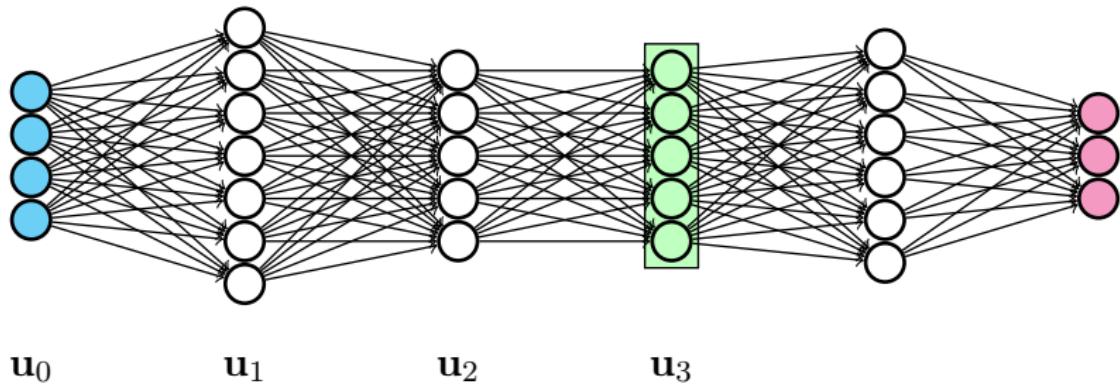
# Forward Propagation: Composition



$$\mathbf{y} \quad f_1(\mathbf{u}_0, \theta_1) \quad f_2(\mathbf{u}_1, \theta_2)$$

$$F(\mathbf{y}, \theta) = f_d ( f_{d-1} ( \cdots ( f_2 ( f_1 (\mathbf{y}, \theta_1), \theta_2 ), \cdots ), \theta_{d-1} ), \theta_d )$$

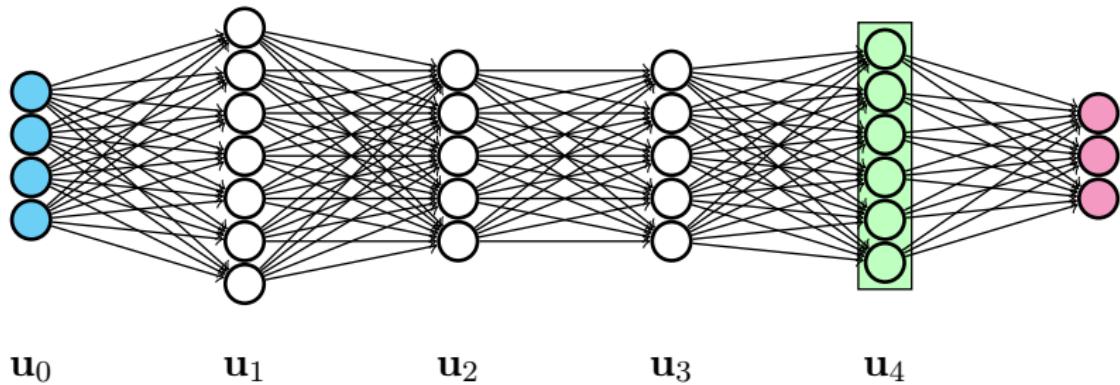
# Forward Propagation: Composition



$$\mathbf{y} \quad f_1(\mathbf{u}_0, \boldsymbol{\theta}_1) \quad f_2(\mathbf{u}_1, \boldsymbol{\theta}_2) \quad f_3(\mathbf{u}_2, \boldsymbol{\theta}_3)$$

$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d ( f_{d-1} ( \cdots ( f_2 ( f_1 (\mathbf{y}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2 ), \cdots ), \boldsymbol{\theta}_{d-1} ), \boldsymbol{\theta}_d )$$

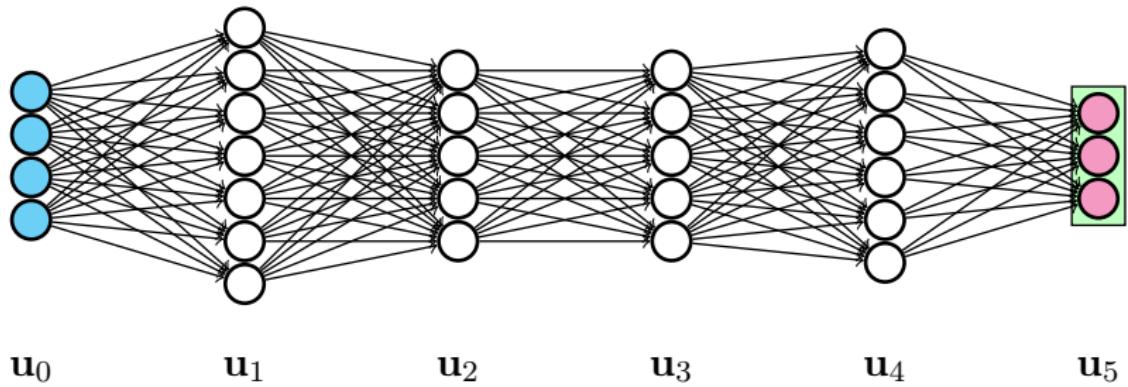
# Forward Propagation: Composition



$$\mathbf{y} \quad f_1(\mathbf{u}_0, \boldsymbol{\theta}_1) \quad f_2(\mathbf{u}_1, \boldsymbol{\theta}_2) \quad f_3(\mathbf{u}_2, \boldsymbol{\theta}_3) \quad f_4(\mathbf{u}_3, \boldsymbol{\theta}_4)$$

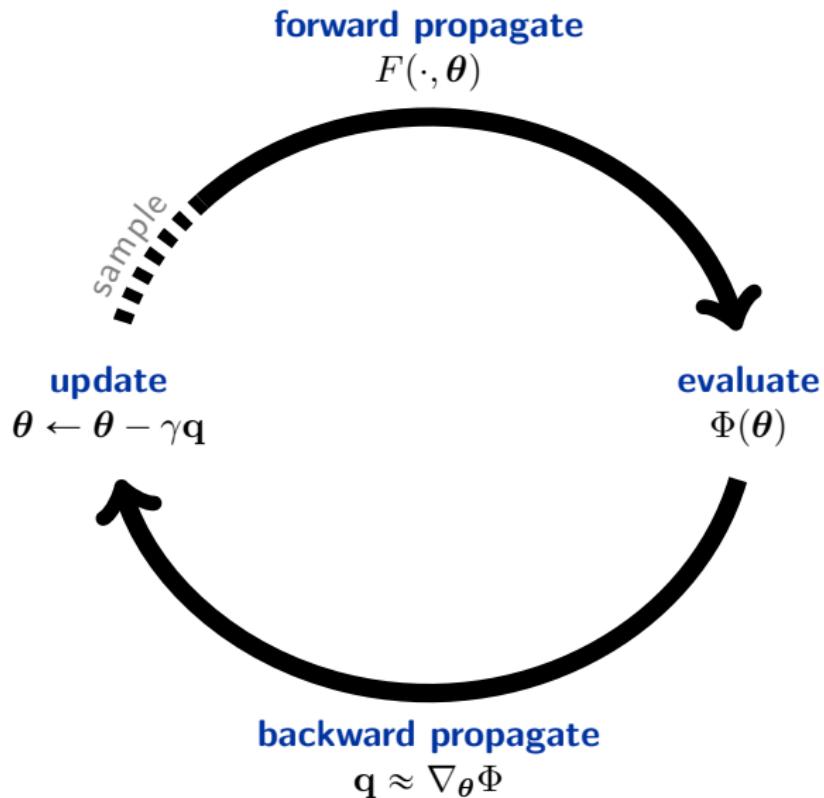
$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d ( f_{d-1} ( \cdots ( f_2 ( f_1 ( \mathbf{y}, \boldsymbol{\theta}_1 ), \boldsymbol{\theta}_2 ), \cdots ), \boldsymbol{\theta}_{d-1} ), \boldsymbol{\theta}_d )$$

# Forward Propagation: Composition



$$F(\mathbf{y}, \boldsymbol{\theta}) = f_d ( f_{d-1} ( \cdots ( f_2 ( f_1 (\mathbf{y}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2 ), \cdots ), \boldsymbol{\theta}_{d-1} ), \boldsymbol{\theta}_d )$$

# The Training Cycle



# Loss Functions

## Least Squares

$$L_{\text{LS}}(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) = \frac{1}{2} \|F(\mathbf{y}, \boldsymbol{\theta}) - \mathbf{c}\|_2^2$$

## Cross Entropy

$$L_{\text{CE}}(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) = -\mathbf{c}^\top \log \sigma(F(\mathbf{y}, \boldsymbol{\theta}))$$

where  $\sigma$  is the **softmax function**

$$\sigma(\mathbf{z}) = \frac{\exp(\mathbf{z})}{\mathbf{1}^\top \exp(\mathbf{z})}$$

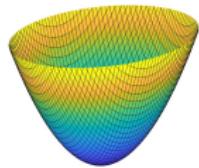
best approximation to data

distance between distributions

# Loss Functions

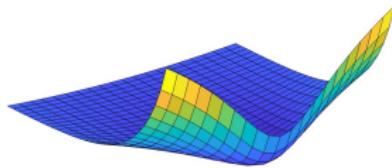
## Least Squares

$$L_{\text{LS}}(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) = \frac{1}{2} \|F(\mathbf{y}, \boldsymbol{\theta}) - \mathbf{c}\|_2^2$$



## Cross Entropy

$$L_{\text{CE}}(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c}) = -\mathbf{c}^\top \log \sigma(F(\mathbf{y}, \boldsymbol{\theta}))$$

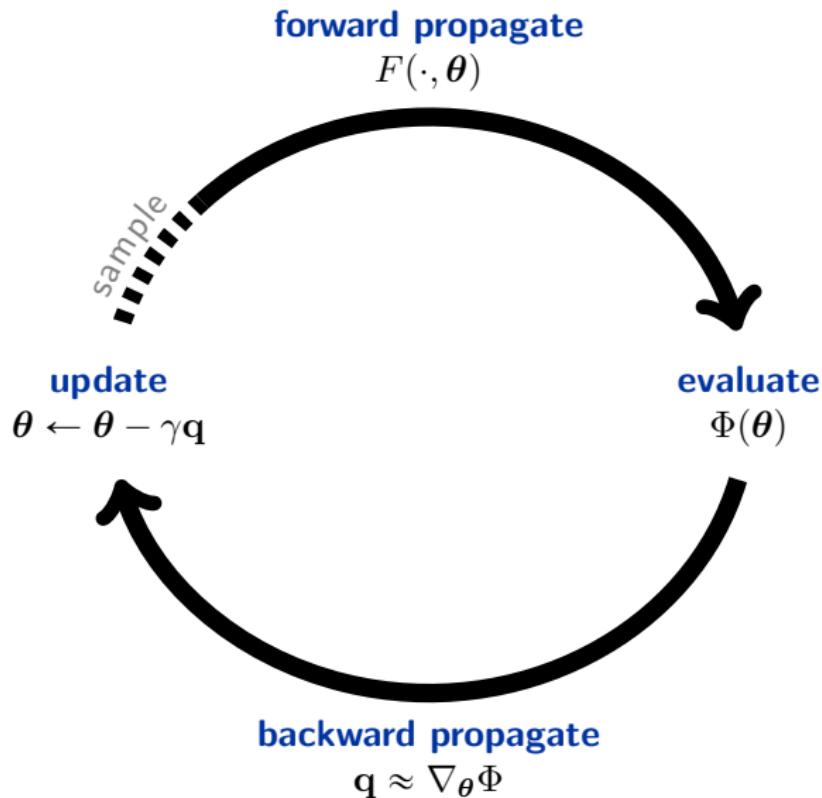


- $L_{\text{LS}} : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}_+$
- function approximation
- convex in first argument

- $L_{\text{CE}} : \mathbb{R}^m \times \Delta^m \rightarrow \mathbb{R}_+$
- classification
- convex in first argument

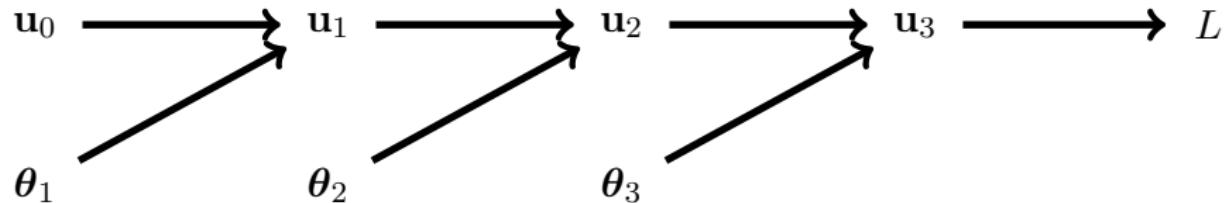
Here,  $\Delta^m = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{1}^\top \mathbf{x} = 0, x_i \geq 0 \text{ for } i = 1, \dots, m\}$  is the unit simplex.

# The Training Cycle



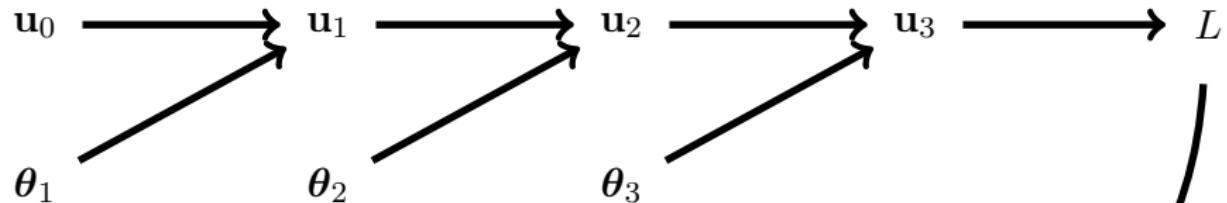
# Backward Propagation: Chain Rule

**Forward Propagate:**

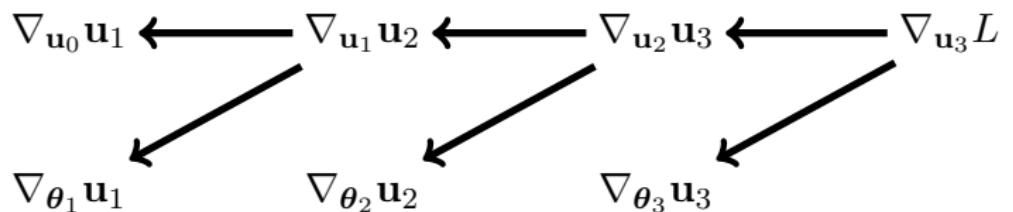


# Backward Propagation: Chain Rule

**Forward Propagate:**

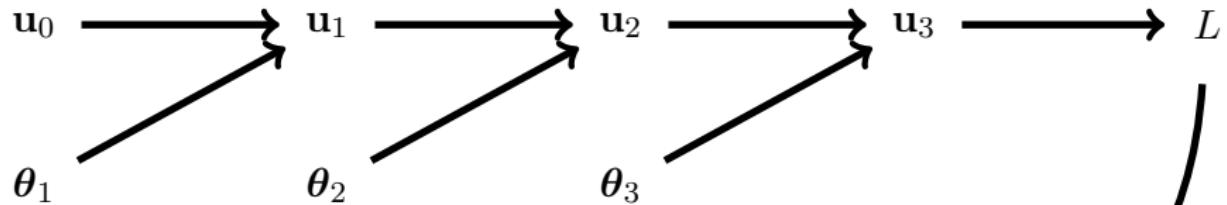


**Backward Propagate:**

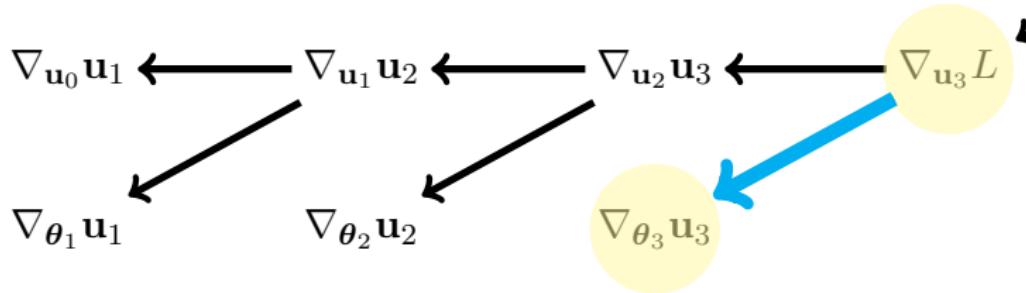


# Backward Propagation: Chain Rule

**Forward Propagate:**  $f_3(u_2, \theta_3)$

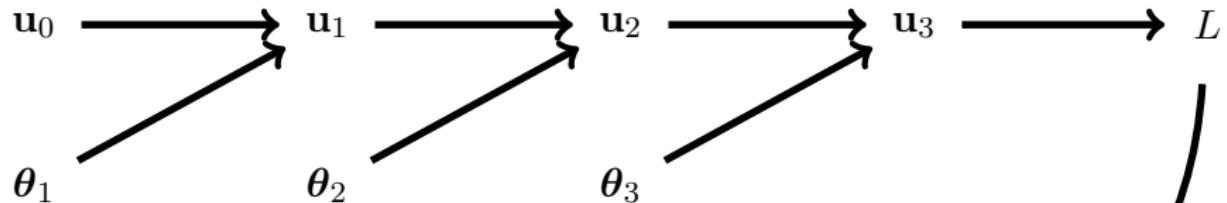


**Backward Propagate:**  $\nabla_{\theta_3} L = \nabla_{\theta_3} u_3 \nabla_{u_3} L$

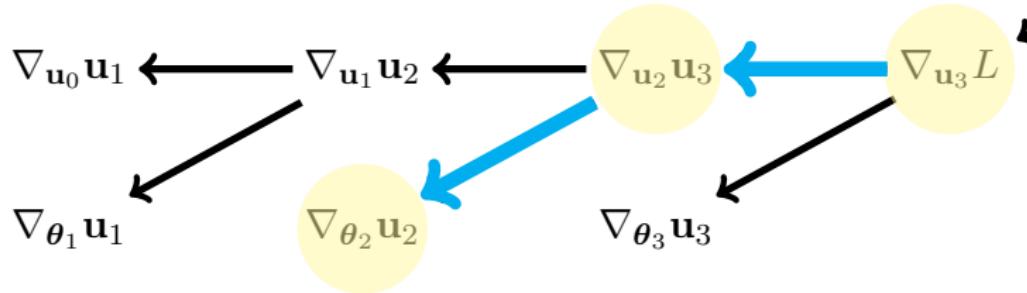


# Backward Propagation: Chain Rule

**Forward Propagate:**  $f_3(f_2(\mathbf{u}_1, \theta_2), \theta_3)$

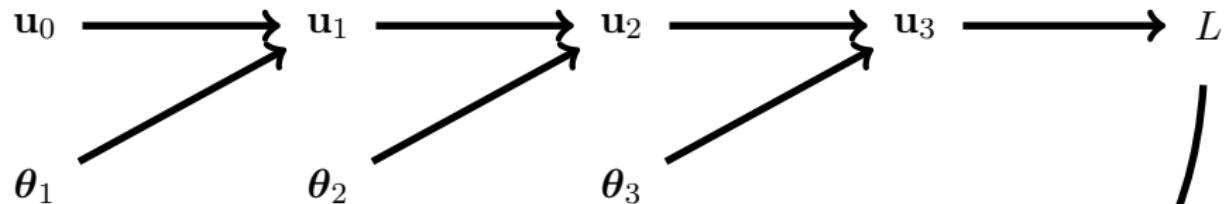


**Backward Propagate:**  $\nabla_{\theta_2} L = \nabla_{\theta_2} \mathbf{u}_2 \nabla_{\theta_3} \mathbf{u}_3 \nabla_{\mathbf{u}_3} L$

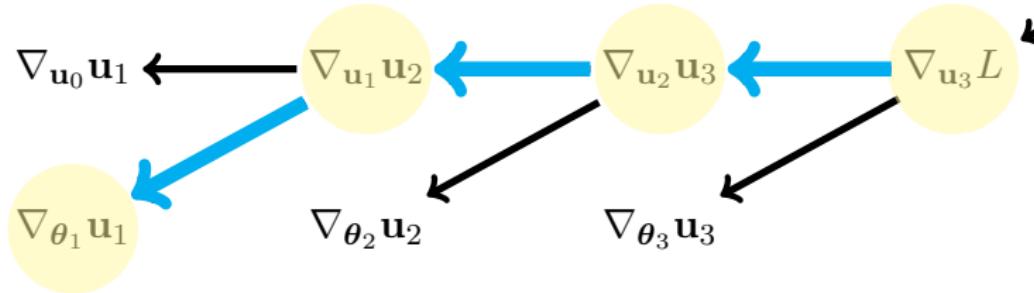


# Backward Propagation: Chain Rule

**Forward Propagate:**  $f_3(f_2(f_1(\mathbf{u}_0, \theta_1), \theta_2), \theta_3)$



**Backward Propagate:**  $\nabla_{\theta_2} L = \nabla_{\theta_1} \mathbf{u}_1 \nabla_{\mathbf{u}_1} \mathbf{u}_2 \nabla_{\mathbf{u}_2} \mathbf{u}_3 \nabla_{\mathbf{u}_3} L$



# Example: Backward Propagation

**Architecture:**  $F : \mathbb{R}^2 \times \Theta \rightarrow \mathbb{R}^2$

$$F(\mathbf{y}, \theta) = \mathbf{W}\mathbf{a}(\mathbf{K}\mathbf{y} + \mathbf{b})$$

---

The notation  $\mathbf{c} \in \Delta^2$  means  $\mathbf{c}$  is in the unit simplex; that is,  $c_1, c_2 \geq 0$  and  $c_1 + c_2 = 1$ .

# Example: Backward Propagation

**Architecture:**  $F : \mathbb{R}^2 \times \Theta \rightarrow \mathbb{R}^2$

$$F(\mathbf{y}, \theta) = \mathbf{W}a(\mathbf{K}\mathbf{y} + \mathbf{b})$$

**Forward:**

$$\mathbf{u}_1 = a(\mathbf{K}\mathbf{y} + \mathbf{b})$$

$$\mathbf{u}_2 = \mathbf{W}\mathbf{u}_1$$

---

The notation  $\mathbf{c} \in \Delta^2$  means  $\mathbf{c}$  is in the unit simplex; that is,  $c_1, c_2 \geq 0$  and  $c_1 + c_2 = 1$ .

# Example: Backward Propagation

**Architecture:**  $F : \mathbb{R}^2 \times \Theta \rightarrow \mathbb{R}^2$

$$F(\mathbf{y}, \theta) = \mathbf{W}a(\mathbf{K}\mathbf{y} + \mathbf{b})$$

**Forward:**

$$\mathbf{u}_1 = a(\mathbf{K}\mathbf{y} + \mathbf{b})$$

$$\mathbf{u}_2 = \mathbf{W}\mathbf{u}_1$$

**Loss:**  $L : \mathbb{R}^2 \times \Delta^2 \rightarrow \mathbb{R}_+$

$$L(\mathbf{u}_2, \mathbf{c}) = -\mathbf{c}^\top \log \sigma(\mathbf{u}_2)$$

$$\nabla_{\mathbf{u}_2} L(\mathbf{u}_2, \mathbf{c}) = \mathbf{c} - \sigma(\mathbf{u}_2)$$

---

The notation  $\mathbf{c} \in \Delta^2$  means  $\mathbf{c}$  is in the unit simplex; that is,  $c_1, c_2 \geq 0$  and  $c_1 + c_2 = 1$ .

# Example: Backward Propagation

**Architecture:**  $F : \mathbb{R}^2 \times \Theta \rightarrow \mathbb{R}^2$

$$F(\mathbf{y}, \theta) = \mathbf{W}a(\mathbf{K}\mathbf{y} + \mathbf{b})$$

**Forward:**

$$\mathbf{u}_1 = a(\mathbf{K}\mathbf{y} + \mathbf{b})$$

$$\mathbf{u}_2 = \mathbf{W}\mathbf{u}_1$$

**Loss:**  $L : \mathbb{R}^2 \times \Delta^2 \rightarrow \mathbb{R}_+$

$$L(\mathbf{u}_2, \mathbf{c}) = -\mathbf{c}^\top \log \sigma(\mathbf{u}_2)$$

$$\nabla_{\mathbf{u}_2} L(\mathbf{u}_2, \mathbf{c}) = \mathbf{c} - \sigma(\mathbf{u}_2)$$

**Backward:**

$$\nabla_{\mathbf{W}} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{W}\mathbf{u}_2} \nabla_{\mathbf{u}_2} L = (\mathbf{c} - \sigma(\mathbf{u}_2)) \mathbf{u}_1^\top$$

---

The notation  $\mathbf{c} \in \Delta^2$  means  $\mathbf{c}$  is in the unit simplex; that is,  $c_1, c_2 \geq 0$  and  $c_1 + c_2 = 1$ .

# Example: Backward Propagation

**Architecture:**  $F : \mathbb{R}^2 \times \Theta \rightarrow \mathbb{R}^2$

$$F(\mathbf{y}, \theta) = \mathbf{W}a(\mathbf{K}\mathbf{y} + \mathbf{b})$$

**Forward:**

$$\mathbf{u}_1 = a(\mathbf{K}\mathbf{y} + \mathbf{b})$$

$$\mathbf{u}_2 = \mathbf{W}\mathbf{u}_1$$

**Loss:**  $L : \mathbb{R}^2 \times \Delta^2 \rightarrow \mathbb{R}_+$

$$L(\mathbf{u}_2, \mathbf{c}) = -\mathbf{c}^\top \log \sigma(\mathbf{u}_2)$$

$$\nabla_{\mathbf{u}_2} L(\mathbf{u}_2, \mathbf{c}) = \mathbf{c} - \sigma(\mathbf{u}_2)$$

**Backward:**

$$\nabla_{\mathbf{W}} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{W}\mathbf{u}_2} \nabla_{\mathbf{u}_2} L = (\mathbf{c} - \sigma(\mathbf{u}_2))\mathbf{u}_1^\top$$

$$\nabla_{\mathbf{u}_1} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{u}_1} \mathbf{u}_2 \nabla_{\mathbf{u}_2} L = \mathbf{W}^\top (\mathbf{c} - \sigma(\mathbf{z}))$$

---

The notation  $\mathbf{c} \in \Delta^2$  means  $\mathbf{c}$  is in the unit simplex; that is,  $c_1, c_2 \geq 0$  and  $c_1 + c_2 = 1$ .

# Example: Backward Propagation

**Architecture:**  $F : \mathbb{R}^2 \times \Theta \rightarrow \mathbb{R}^2$

$$F(\mathbf{y}, \theta) = \mathbf{W}\mathbf{a}(\mathbf{K}\mathbf{y} + \mathbf{b})$$

**Forward:**

$$\mathbf{u}_1 = \mathbf{a}(\mathbf{K}\mathbf{y} + \mathbf{b})$$

$$\mathbf{u}_2 = \mathbf{W}\mathbf{u}_1$$

**Loss:**  $L : \mathbb{R}^2 \times \Delta^2 \rightarrow \mathbb{R}_+$

$$L(\mathbf{u}_2, \mathbf{c}) = -\mathbf{c}^\top \log \sigma(\mathbf{u}_2)$$

$$\nabla_{\mathbf{u}_2} L(\mathbf{u}_2, \mathbf{c}) = \mathbf{c} - \sigma(\mathbf{u}_2)$$

**Backward:**

$$\nabla_{\mathbf{W}} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{W}\mathbf{u}_2} \nabla_{\mathbf{u}_2} L = (\mathbf{c} - \sigma(\mathbf{u}_2))\mathbf{u}_1^\top$$

$$\nabla_{\mathbf{u}_1} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{u}_1 \mathbf{u}_2} \nabla_{\mathbf{u}_2} L = \mathbf{W}^\top (\mathbf{c} - \sigma(\mathbf{z}))$$

$$\nabla_{\mathbf{K}} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{K}\mathbf{u}_1} \nabla_{\mathbf{u}_1} L = \text{diag}(a'(\mathbf{K}\mathbf{y} + \mathbf{b}))(\mathbf{c} - \sigma(\mathbf{u}_2))\mathbf{y}^\top$$

---

The notation  $\mathbf{c} \in \Delta^2$  means  $\mathbf{c}$  is in the unit simplex; that is,  $c_1, c_2 \geq 0$  and  $c_1 + c_2 = 1$ .

# Example: Backward Propagation

**Architecture:**  $F : \mathbb{R}^2 \times \Theta \rightarrow \mathbb{R}^2$

$$F(\mathbf{y}, \theta) = \mathbf{W}\mathbf{a}(\mathbf{K}\mathbf{y} + \mathbf{b})$$

**Forward:**

$$\mathbf{u}_1 = a(\mathbf{K}\mathbf{y} + \mathbf{b})$$

$$\mathbf{u}_2 = \mathbf{W}\mathbf{u}_1$$

**Loss:**  $L : \mathbb{R}^2 \times \Delta^2 \rightarrow \mathbb{R}_+$

$$L(\mathbf{u}_2, \mathbf{c}) = -\mathbf{c}^\top \log \sigma(\mathbf{u}_2)$$

$$\nabla_{\mathbf{u}_2} L(\mathbf{u}_2, \mathbf{c}) = \mathbf{c} - \sigma(\mathbf{u}_2)$$

**Backward:**

$$\nabla_{\mathbf{W}} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{W}\mathbf{u}_2} \nabla_{\mathbf{u}_2} L = (\mathbf{c} - \sigma(\mathbf{u}_2))\mathbf{u}_1^\top$$

$$\nabla_{\mathbf{u}_1} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{u}_1 \mathbf{u}_2} \nabla_{\mathbf{u}_2} L = \mathbf{W}^\top (\mathbf{c} - \sigma(\mathbf{z}))$$

$$\nabla_{\mathbf{K}} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{K}\mathbf{u}_1} \nabla_{\mathbf{u}_1} L = \text{diag}(a'(\mathbf{K}\mathbf{y} + \mathbf{b}))(\mathbf{c} - \sigma(\mathbf{u}_2))\mathbf{y}^\top$$

$$\nabla_{\mathbf{b}} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{b}\mathbf{u}_1} \nabla_{\mathbf{u}_1} L = \text{diag}(a'(\mathbf{K}\mathbf{y} + \mathbf{b}))(\mathbf{c} - \sigma(\mathbf{u}_2))$$

---

The notation  $\mathbf{c} \in \Delta^2$  means  $\mathbf{c}$  is in the unit simplex; that is,  $c_1, c_2 \geq 0$  and  $c_1 + c_2 = 1$ .

# Example: Backward Propagation

**Architecture:**  $F : \mathbb{R}^2 \times \Theta \rightarrow \mathbb{R}^2$

$$F(\mathbf{y}, \theta) = \mathbf{W}\mathbf{a}(\mathbf{K}\mathbf{y} + \mathbf{b})$$

**Forward:**

$$\mathbf{u}_1 = \mathbf{a}(\mathbf{K}\mathbf{y} + \mathbf{b})$$

$$\mathbf{u}_2 = \mathbf{W}\mathbf{u}_1$$

**Loss:**  $L : \mathbb{R}^2 \times \Delta^2 \rightarrow \mathbb{R}_+$

$$L(\mathbf{u}_2, \mathbf{c}) = -\mathbf{c}^\top \log \sigma(\mathbf{u}_2)$$

$$\nabla_{\mathbf{u}_2} L(\mathbf{u}_2, \mathbf{c}) = \mathbf{c} - \sigma(\mathbf{u}_2)$$

**Backward:**

$$\nabla_{\mathbf{W}} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{W}\mathbf{u}_2} \nabla_{\mathbf{u}_2} L = (\mathbf{c} - \sigma(\mathbf{u}_2))\mathbf{u}_1^\top$$

$$\nabla_{\mathbf{u}_1} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{u}_1 \mathbf{u}_2} \nabla_{\mathbf{u}_2} L = \mathbf{W}^\top (\mathbf{c} - \sigma(\mathbf{z}))$$

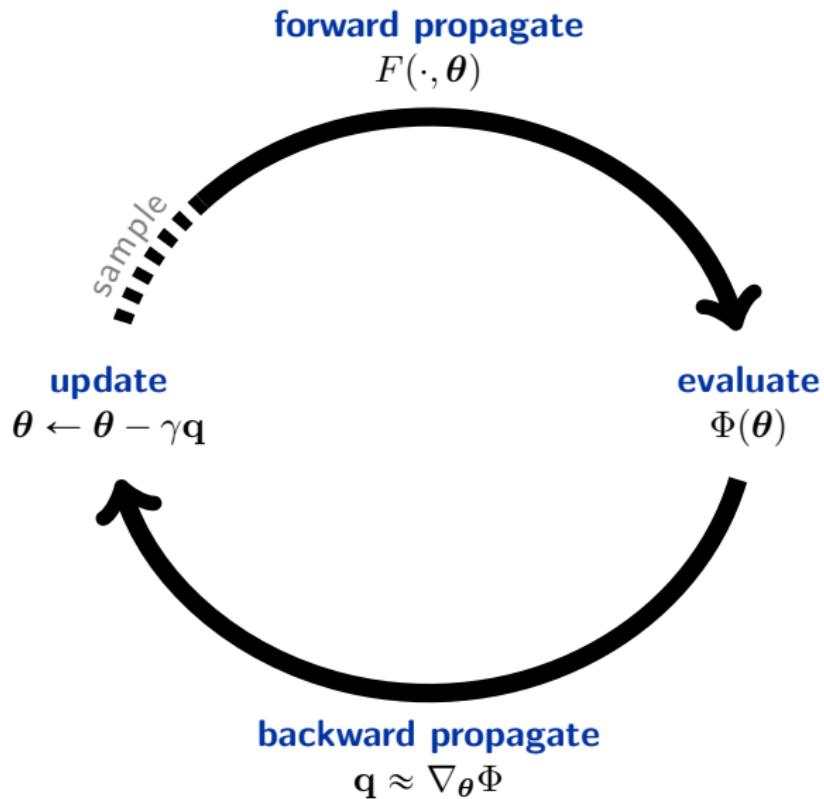
$$\nabla_{\mathbf{K}} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{K}\mathbf{u}_1} \nabla_{\mathbf{u}_1} L = \text{diag}(a'(\mathbf{K}\mathbf{y} + \mathbf{b}))(\mathbf{c} - \sigma(\mathbf{u}_2))\mathbf{y}^\top$$

$$\nabla_{\mathbf{b}} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{b}\mathbf{u}_1} \nabla_{\mathbf{u}_1} L = \text{diag}(a'(\mathbf{K}\mathbf{y} + \mathbf{b}))(\mathbf{c} - \sigma(\mathbf{u}_2))$$

$$\nabla_{\mathbf{y}} L(\mathbf{u}_2, \mathbf{c}) = \nabla_{\mathbf{y}\mathbf{u}_1} \nabla_{\mathbf{u}_1} L = \mathbf{K}^\top \text{diag}(a'(\mathbf{K}\mathbf{y} + \mathbf{b}))(\mathbf{c} - \sigma(\mathbf{u}_2))$$

The notation  $\mathbf{c} \in \Delta^2$  means  $\mathbf{c}$  is in the unit simplex; that is,  $c_1, c_2 \geq 0$  and  $c_1 + c_2 = 1$ .

# The Training Cycle



# Gradient Descent vs. Stochastic Gradient

## Gradient Descent

$$\min_{\theta} \Phi_{\mathcal{T}}(\theta) \equiv \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{y}, \mathbf{c}) \in \mathcal{T}} L(F(\mathbf{y}, \theta), \mathbf{c})$$

## Stochastic Gradient

$$\min_{\theta} \Phi(\theta) \equiv \mathbb{E} L(F(\mathbf{y}, \theta), \mathbf{c})$$

# GD vs. SG

## Gradient Descent

$$\min_{\boldsymbol{\theta}} \Phi_{\mathcal{T}}(\boldsymbol{\theta}) \equiv \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{y}, \mathbf{c}) \in \mathcal{T}} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c})$$

## Stochastic Gradient

$$\min_{\boldsymbol{\theta}} \Phi(\boldsymbol{\theta}) \equiv \mathbb{E} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c})$$

---

Bottou, Curtis, and Nocedal, "Optimization Methods for Large-Scale Machine Learning"; Robbins and Monro, "A Stochastic Approximation Method"; Nocedal and Wright, *Numerical Optimization*

# GD vs. SG

## Gradient Descent

$$\min_{\boldsymbol{\theta}} \Phi_{\mathcal{T}}(\boldsymbol{\theta}) \equiv \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{y}, \mathbf{c}) \in \mathcal{T}} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c})$$

**Goal:** Find stationary point

$$\nabla \Phi_{\mathcal{T}}(\boldsymbol{\theta}^*) = \mathbf{0}$$

## Stochastic Gradient

$$\min_{\boldsymbol{\theta}} \Phi(\boldsymbol{\theta}) \equiv \mathbb{E} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c})$$

**Goal:** Let  $\Phi_i(\boldsymbol{\theta}) = L(F(\mathbf{y}_i, \boldsymbol{\theta}), \mathbf{c}_i)$ .

$$\nabla \Phi(\hat{\boldsymbol{\theta}}) = \mathbb{E}[\nabla \Phi_i(\hat{\boldsymbol{\theta}})] = \mathbf{0}$$

# GD vs. SG

## Gradient Descent

$$\min_{\boldsymbol{\theta}} \Phi_{\mathcal{T}}(\boldsymbol{\theta}) \equiv \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{y}, \mathbf{c}) \in \mathcal{T}} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c})$$

**Goal:** Find stationary point

$$\nabla \Phi_{\mathcal{T}}(\boldsymbol{\theta}^*) = \mathbf{0}$$

**Iteration:**

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla \Phi_{\mathcal{T}}(\boldsymbol{\theta}_k)$$

## Stochastic Gradient

$$\min_{\boldsymbol{\theta}} \Phi(\boldsymbol{\theta}) \equiv \mathbb{E} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c})$$

**Goal:** Let  $\Phi_i(\boldsymbol{\theta}) = L(F(\mathbf{y}_i, \boldsymbol{\theta}), \mathbf{c}_i)$ .

$$\nabla \Phi(\hat{\boldsymbol{\theta}}) = \mathbb{E}[\nabla \Phi_i(\hat{\boldsymbol{\theta}})] = \mathbf{0}$$

**Iteration:**

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \nabla \Phi_i(\boldsymbol{\theta}_i)$$

# GD vs. SG

## Gradient Descent

$$\min_{\boldsymbol{\theta}} \Phi_{\mathcal{T}}(\boldsymbol{\theta}) \equiv \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{y}, \mathbf{c}) \in \mathcal{T}} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c})$$

**Goal:** Find stationary point

$$\nabla \Phi_{\mathcal{T}}(\boldsymbol{\theta}^*) = \mathbf{0}$$

**Iteration:**

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla \Phi_{\mathcal{T}}(\boldsymbol{\theta}_k)$$

- deterministic (empirical risk/sample average approximation)
- descent (with well-chosen step sizes)
- parallelizable, yet computationally demanding iterates
- potential to overfit

## Stochastic Gradient

$$\min_{\boldsymbol{\theta}} \Phi(\boldsymbol{\theta}) \equiv \mathbb{E} L(F(\mathbf{y}, \boldsymbol{\theta}), \mathbf{c})$$

**Goal:** Let  $\Phi_i(\boldsymbol{\theta}) = L(F(\mathbf{y}_i, \boldsymbol{\theta}), \mathbf{c}_i)$ .

$$\nabla \Phi(\hat{\boldsymbol{\theta}}) = \mathbb{E}[\nabla \Phi_i(\hat{\boldsymbol{\theta}})] = \mathbf{0}$$

**Iteration:**

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \nabla \Phi_i(\boldsymbol{\theta}_i)$$

- stochastic (expected risk/stochastic approximation)
- no guaranteed decrease
- scalable, cheap iterates, yet hard to parallelize
- implicit regularization due to randomness

---

Bottou, Curtis, and Nocedal, "Optimization Methods for Large-Scale Machine Learning"; Robbins and Monro, "A Stochastic Approximation Method"; Nocedal and Wright, *Numerical Optimization*

# Stochastic Gradient Convergence

## Assumption 1

The objective function  $\Phi$  is **differentiable and bounded below**.

## Assumption 2

The objective function  $\Phi$  is **continuously differentiable** and  $\nabla\Phi$  is **Lipschitz continuous** with Lipschitz constant  $L > 0$ ; that is,

$$\|\nabla\Phi(\boldsymbol{\theta}) - \nabla\Phi(\tilde{\boldsymbol{\theta}})\|_2 \leq L\|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|_2 \quad \text{for all } \boldsymbol{\theta}, \tilde{\boldsymbol{\theta}} \in \Theta$$

## Assumption 3

The stochastic gradient is an unbiased estimate of the expected gradient; that is,

$$\mathbb{E}_i[\nabla\Phi_i(\boldsymbol{\theta})] = \nabla\Phi(\boldsymbol{\theta})$$

where  $\mathbb{E}_i$  is the expectation conditioned on current sample  $(\mathbf{y}_i, \mathbf{c}_i)$  and weights  $\boldsymbol{\theta}_i$ .

# Stochastic Gradient Convergence

Given the stochastic iterate

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \nabla \Phi_i(\boldsymbol{\theta}_i)$$

Bound the objective function value

$$\begin{aligned}\Phi(\boldsymbol{\theta}_{i+1}) &\leq \Phi(\boldsymbol{\theta}_i) + \nabla \Phi(\boldsymbol{\theta}_i)^\top (\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i) + \frac{1}{2} L \|\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i\|_2^2 \quad \text{by Lipschitz continuity} \\ &= \Phi(\boldsymbol{\theta}_i) - \alpha_i \nabla \Phi(\boldsymbol{\theta}_i)^\top \nabla \Phi_i(\boldsymbol{\theta}_i) + \frac{1}{2} \alpha_i^2 L \|\nabla \Phi_i(\boldsymbol{\theta}_i)\|_2^2\end{aligned}$$

# Stochastic Gradient Convergence

Given the stochastic iterate

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \nabla \Phi_i(\boldsymbol{\theta}_i)$$

Bound the objective function value

$$\begin{aligned}\Phi(\boldsymbol{\theta}_{i+1}) &\leq \Phi(\boldsymbol{\theta}_i) + \nabla \Phi(\boldsymbol{\theta}_i)^\top (\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i) + \frac{1}{2} L \|\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i\|_2^2 \quad \text{by Lipschitz continuity} \\ &= \Phi(\boldsymbol{\theta}_i) - \alpha_i \nabla \Phi(\boldsymbol{\theta}_i)^\top \nabla \Phi_i(\boldsymbol{\theta}_i) + \frac{1}{2} \alpha_i^2 L \|\nabla \Phi_i(\boldsymbol{\theta}_i)\|_2^2\end{aligned}$$

Take the conditional expectation

$$\mathbb{E}_i[\Phi(\boldsymbol{\theta}_{i+1})] \leq \mathbb{E}_i[\Phi(\boldsymbol{\theta}_i)] - \alpha_i \mathbb{E}_i[\nabla \Phi(\boldsymbol{\theta}_i)^\top \nabla \Phi_i(\boldsymbol{\theta}_i)] + \frac{1}{2} \alpha_i^2 L \mathbb{E}_i[\|\nabla \Phi_i(\boldsymbol{\theta}_i)\|_2^2]$$

# Stochastic Gradient Convergence

Given the stochastic iterate

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \nabla \Phi_i(\boldsymbol{\theta}_i)$$

Bound the objective function value

$$\begin{aligned}\Phi(\boldsymbol{\theta}_{i+1}) &\leq \Phi(\boldsymbol{\theta}_i) + \nabla \Phi(\boldsymbol{\theta}_i)^\top (\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i) + \frac{1}{2} L \|\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i\|_2^2 \quad \text{by Lipschitz continuity} \\ &= \Phi(\boldsymbol{\theta}_i) - \alpha_i \nabla \Phi(\boldsymbol{\theta}_i)^\top \nabla \Phi_i(\boldsymbol{\theta}_i) + \frac{1}{2} \alpha_i^2 L \|\nabla \Phi_i(\boldsymbol{\theta}_i)\|_2^2\end{aligned}$$

Take the conditional expectation

$$\begin{aligned}\mathbb{E}_i[\Phi(\boldsymbol{\theta}_{i+1})] &\leq \mathbb{E}_i[\Phi(\boldsymbol{\theta}_i)] - \alpha_i \mathbb{E}_i[\nabla \Phi(\boldsymbol{\theta}_i)^\top \nabla \Phi_i(\boldsymbol{\theta}_i)] + \frac{1}{2} \alpha_i^2 L \mathbb{E}_i[\|\nabla \Phi_i(\boldsymbol{\theta}_i)\|_2^2] \\ &= \mathbb{E}_i[\Phi(\boldsymbol{\theta}_i)] - \alpha_i \|\nabla \Phi(\boldsymbol{\theta}_i)\|_2^2 + \frac{1}{2} \alpha_i^2 L \mathbb{E}_i[\|\nabla \Phi_i(\boldsymbol{\theta}_i)\|_2^2] \quad \text{unbiased gradient}\end{aligned}$$

# Stochastic Gradient Convergence

Given the stochastic iterate

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \nabla \Phi_i(\boldsymbol{\theta}_i)$$

Bound the objective function value

$$\begin{aligned}\Phi(\boldsymbol{\theta}_{i+1}) &\leq \Phi(\boldsymbol{\theta}_i) + \nabla \Phi(\boldsymbol{\theta}_i)^\top (\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i) + \frac{1}{2} L \|\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i\|_2^2 \quad \text{by Lipschitz continuity} \\ &= \Phi(\boldsymbol{\theta}_i) - \alpha_i \nabla \Phi(\boldsymbol{\theta}_i)^\top \nabla \Phi_i(\boldsymbol{\theta}_i) + \frac{1}{2} \alpha_i^2 L \|\nabla \Phi_i(\boldsymbol{\theta}_i)\|_2^2\end{aligned}$$

Take the conditional expectation

$$\begin{aligned}\mathbb{E}_i[\Phi(\boldsymbol{\theta}_{i+1})] &\leq \mathbb{E}_i[\Phi(\boldsymbol{\theta}_i)] - \alpha_i \mathbb{E}_i[\nabla \Phi(\boldsymbol{\theta}_i)^\top \nabla \Phi_i(\boldsymbol{\theta}_i)] + \frac{1}{2} \alpha_i^2 L \mathbb{E}_i[\|\nabla \Phi_i(\boldsymbol{\theta}_i)\|_2^2] \\ &= \underbrace{\mathbb{E}_i[\Phi(\boldsymbol{\theta}_i)] - \alpha_i \|\nabla \Phi(\boldsymbol{\theta}_i)\|_2^2}_{\text{deterministic}} + \underbrace{\frac{1}{2} \alpha_i^2 L \mathbb{E}_i[\|\nabla \Phi_i(\boldsymbol{\theta}_i)\|_2^2]}_{\text{random noise}} \quad \text{unbiased gradient}\end{aligned}$$

# Stochastic Gradient Convergence

Given the stochastic iterate

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \nabla \Phi_i(\boldsymbol{\theta}_i)$$

Bound the objective function value

$$\begin{aligned}\Phi(\boldsymbol{\theta}_{i+1}) &\leq \Phi(\boldsymbol{\theta}_i) + \nabla \Phi(\boldsymbol{\theta}_i)^\top (\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i) + \frac{1}{2} L \|\boldsymbol{\theta}_{i+1} - \boldsymbol{\theta}_i\|_2^2 \quad \text{by Lipschitz continuity} \\ &= \Phi(\boldsymbol{\theta}_i) - \alpha_i \nabla \Phi(\boldsymbol{\theta}_i)^\top \nabla \Phi_i(\boldsymbol{\theta}_i) + \frac{1}{2} \alpha_i^2 L \|\nabla \Phi_i(\boldsymbol{\theta}_i)\|_2^2\end{aligned}$$

Take the conditional expectation

$$\begin{aligned}\mathbb{E}_i[\Phi(\boldsymbol{\theta}_{i+1})] &\leq \mathbb{E}_i[\Phi(\boldsymbol{\theta}_i)] - \alpha_i \mathbb{E}_i[\nabla \Phi(\boldsymbol{\theta}_i)^\top \nabla \Phi_i(\boldsymbol{\theta}_i)] + \frac{1}{2} \alpha_i^2 L \mathbb{E}_i[\|\nabla \Phi_i(\boldsymbol{\theta}_i)\|_2^2] \\ &= \underbrace{\mathbb{E}_i[\Phi(\boldsymbol{\theta}_i)] - \alpha_i \|\nabla \Phi(\boldsymbol{\theta}_i)\|_2^2}_{\text{deterministic}} + \underbrace{\frac{1}{2} \alpha_i^2 L \mathbb{E}_i[\|\nabla \Phi_i(\boldsymbol{\theta}_i)\|_2^2]}_{\text{random noise}} \quad \text{unbiased gradient}\end{aligned}$$

Choose step sizes to ensure decrease in the long run

$$\sum_i \alpha_i = \infty \quad \text{and} \quad \sum_i \alpha_i^2 < \infty$$

# Mini-Batch Stochastic Gradient

## Pseudocode

```

1: while not converged
2:   random partition  $\mathcal{T} = \bigsqcup_{i=1}^{n_b} \mathcal{T}_i$ 
3:   for  $i = 1, \dots, n_b$ 
4:     for  $(\mathbf{y}_j, \mathbf{c}_j) \in \mathcal{T}_i$ 
5:        $\hat{\mathbf{c}}_j = f(\mathbf{y}_j, \boldsymbol{\theta}_i)$ 
6:        $\phi_j = L(\hat{\mathbf{c}}_j, \mathbf{c}_j)$ 
    end for
7:        $\Phi_i = \frac{1}{|\mathcal{T}_i|} \sum_{j=1}^{|\mathcal{T}_i|} \phi_j$ 
8:        $\mathbf{g}_i = \nabla \Phi_i(\boldsymbol{\theta}_i)$ 
9:        $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{g}_i$ 
  end for
end while

```

## PyTorch Code

```

1: for epoch in range(max_epochs):
2:   # shuffle data
    shuffle_idx = randperm(n_samples)
3:   for epoch in range(nb):
4:     # select batch
      idx = shuffle_idx[i*nb:(i+1)*nb]
      yb, cb = y_train[idx], c_train[idx]
5:     # zero out gradients
      optimizer.zero_grad()
6:     # forward propagate
      cb_hat = net(yb)
7:     # evaluate
      phi = loss(cb_hat, cb)
8:     # backward propagate
      phi.backward()
9:     # update
      optimizer.step()

```

# Mini-Batch Stochastic Gradient

## Pseudocode

```

1: while not converged
2:   random partition  $\mathcal{T} = \bigsqcup_{i=1}^{n_b} \mathcal{T}_i$ 
3:   for  $i = 1, \dots, n_b$ 
4:     for  $(\mathbf{y}_j, \mathbf{c}_j) \in \mathcal{T}_i$ 
5:        $\hat{\mathbf{c}}_j = f(\mathbf{y}_j, \boldsymbol{\theta}_i)$ 
6:        $\phi_j = L(\hat{\mathbf{c}}_j, \mathbf{c}_j)$ 
    end for
7:      $\Phi_i = \frac{1}{|\mathcal{T}_i|} \sum_{j=1}^{|\mathcal{T}_i|} \phi_j$ 
8:      $\mathbf{g}_i = \nabla \Phi_i(\boldsymbol{\theta}_i)$ 
9:      $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{g}_i$ 
  end for
end while

```

## PyTorch Code

```

1: for epoch in range(max_epochs):
2:   # shuffle data
    shuffle_idx = randperm(n_samples)
3:   for epoch in range(nb):
4:     # select batch
      idx = shuffle_idx[i*nb:(i+1)*nb]
      yb, cb = y_train[idx], c_train[idx]
5:     # zero out gradients
      optimizer.zero_grad()
6:     # forward propagate
      cb_hat = net(yb)
7:     # evaluate
      phi = loss(cb_hat, cb)
8:     # backward propagate
      phi.backward()
9:     # update
      optimizer.step()

```

# Mini-Batch Stochastic Gradient

## Pseudocode

```

1: while not converged
2:   random partition  $\mathcal{T} = \bigsqcup_{i=1}^{n_b} \mathcal{T}_i$ 
3:   for  $i = 1, \dots, n_b$ 
4:     for  $(\mathbf{y}_j, \mathbf{c}_j) \in \mathcal{T}_i$ 
5:        $\hat{\mathbf{c}}_j = f(\mathbf{y}_j, \boldsymbol{\theta}_i)$ 
6:        $\phi_j = L(\hat{\mathbf{c}}_j, \mathbf{c}_j)$ 
    end for
7:      $\Phi_i = \frac{1}{|\mathcal{T}_i|} \sum_{j=1}^{|\mathcal{T}_i|} \phi_j$ 
8:      $\mathbf{g}_i = \nabla \Phi_i(\boldsymbol{\theta}_i)$ 
9:      $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{g}_i$ 
  end for
end while

```

## PyTorch Code

```

1: for epoch in range(max_epochs):
2:   # shuffle data
    shuffle_idx = randperm(n_samples)
3:   for epoch in range(nb):
4:     # select batch
      idx = shuffle_idx[i*nb:(i+1)*nb]
      yb, cb = y_train[idx], c_train[idx]
5:     # zero out gradients
      optimizer.zero_grad()
6:     # forward propagate
      cb_hat = net(yb)
7:     # evaluate
      phi = loss(cb_hat, cb)
8:     # backward propagate
      phi.backward()
9:     # update
      optimizer.step()

```

# Mini-Batch Stochastic Gradient

## Pseudocode

```

1: while not converged
2:   random partition  $\mathcal{T} = \bigsqcup_{i=1}^{n_b} \mathcal{T}_i$ 
3:   for  $i = 1, \dots, n_b$ 
4:     for  $(y_j, \mathbf{c}_j) \in \mathcal{T}_i$ 
5:        $\hat{\mathbf{c}}_j = f(\mathbf{y}_j, \boldsymbol{\theta}_i)$ 
6:        $\phi_j = L(\hat{\mathbf{c}}_j, \mathbf{c}_j)$ 
    end for
7:        $\Phi_i = \frac{1}{|\mathcal{T}_i|} \sum_{j=1}^{|\mathcal{T}_i|} \phi_j$ 
8:        $\mathbf{g}_i = \nabla \Phi_i(\boldsymbol{\theta}_i)$ 
9:        $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{g}_i$ 
  end for
end while

```

## PyTorch Code

```

1: for epoch in range(max_epochs):
2:   # shuffle data
    shuffle_idx = randperm(n_samples)
3:   for epoch in range(nb):
4:     # select batch
      idx = shuffle_idx[i*nb:(i+1)*nb]
      yb, cb = y_train[idx], c_train[idx]
5:     # zero out gradients
      optimizer.zero_grad()
6:     # forward propagate
      cb_hat = net(yb)
7:     # evaluate
      phi = loss(cb_hat, cb)
8:     # backward propagate
      phi.backward()
9:     # update
      optimizer.step()

```

# Mini-Batch Stochastic Gradient

## Pseudocode

```

1: while not converged
2:   random partition  $\mathcal{T} = \bigsqcup_{i=1}^{n_b} \mathcal{T}_i$ 
3:   for  $i = 1, \dots, n_b$ 
4:     for  $(y_j, c_j) \in \mathcal{T}_i$ 
5:        $\hat{c}_j = f(y_j, \theta_i)$ 
6:        $\phi_j = L(\hat{c}_j, c_j)$ 
    end for
7:        $\Phi_i = \frac{1}{|\mathcal{T}_i|} \sum_{j=1}^{|\mathcal{T}_i|} \phi_j$ 
8:        $\mathbf{g}_i = \nabla \Phi_i(\theta_i)$ 
9:        $\theta_{i+1} = \theta_i - \alpha_i \mathbf{g}_i$ 
end for
end while

```

## PyTorch Code

```

1: for epoch in range(max_epochs):
2:   # shuffle data
    shuffle_idx = randperm(n_samples)
3:   for epoch in range(nb):
4:     # select batch
      idx = shuffle_idx[i*nb:(i+1)*nb]
      yb, cb = y_train[idx], c_train[idx]
5:     # zero out gradients
      optimizer.zero_grad()
6:     # forward propagate
      cb_hat = net(yb)
7:     # evaluate
      phi = loss(cb_hat, cb)
8:     # backward propagate
      phi.backward()
9:     # update
      optimizer.step()

```

# Popular Stochastic Gradient Variants

```
1: while not converged
2:   random partition  $\mathcal{T} = \bigsqcup_{i=1}^{n_b} \mathcal{T}_i$ 
3:   for  $i = 1, \dots, n_b$ 
4:     for  $(\mathbf{y}_j, \mathbf{c}_j) \in \mathcal{T}_i$ 
5:        $\hat{\mathbf{c}}_j = F(\mathbf{y}_j, \boldsymbol{\theta}_i)$ 
6:        $\phi_j = L(\hat{\mathbf{c}}_j, \mathbf{c}_j)$ 
7:     end for
8:      $\Phi_i = \frac{1}{|\mathcal{T}_i|} \sum_{j=1}^{|\mathcal{T}_i|} \phi_j$ 
9:      $\mathbf{g}_i = \nabla \Phi_i$ 
       $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{g}_i$ 
end for
end while
```

# Popular Stochastic Gradient Variants

## Stochastic Gradient (SG)

$$1: \mathbf{g}_i = \nabla \Phi_i$$

$$2: \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{g}_i$$

## SG + Momentum (SGM)

$$1: \mathbf{g}_i = \nabla \Phi_i$$

$$2: \mathbf{m}_i = \beta \mathbf{m}_{i-1} + (1 - \beta) \mathbf{g}_i$$

$$3: \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{m}_i$$

# Popular Stochastic Gradient Variants

## Stochastic Gradient (SG)

$$1: \mathbf{g}_i = \nabla \Phi_i$$

$$2: \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{g}_i$$

## Adaptive Subgrad. (AdaGrad)

$$1: \mathbf{g}_i = \nabla \Phi_i$$

$$2: \mathbf{v}_i = \mathbf{v}_{i-1} + \mathbf{g}_i^2$$

$$3: \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \left( \frac{\mathbf{g}_i}{\sqrt{\mathbf{v}_i + \varepsilon}} \right)$$

# Popular Stochastic Gradient Variants

## Stochastic Gradient (SG)

$$1: \mathbf{g}_i = \nabla \Phi_i$$

$$2: \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \mathbf{g}_i$$

## Adaptive Momentum (Adam)

$$1: \mathbf{g}_i = \nabla \phi_i$$

$$2: \mathbf{m}_i = \beta_1 \mathbf{m}_{i-1} + (1 - \beta_1) \mathbf{g}_i$$

$$3: \mathbf{v}_i = \beta_2 \mathbf{v}_{i-1} + (1 - \beta_2) \mathbf{g}_i^2$$

$$6: \hat{\mathbf{m}}_i = \mathbf{m}_i / (1 + \hat{\beta}_1^i)$$

$$7: \hat{\mathbf{v}}_i = \mathbf{v}_i / (1 + \hat{\beta}_2^i)$$

$$8: \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha_i \left( \frac{\hat{\mathbf{m}}_i}{\sqrt{\hat{\mathbf{v}}_i} + \epsilon} \right)$$

# Popular Stochastic Gradient Variants

**SG**

**SGM**

**AdaGrad**

**GD**

**Adam**

# Your Turn!

Play with the notebooks in the repository!

<https://github.com/elizabethnewman/dnn101>

# References I

-  Bottou, Léon, Frank E. Curtis, and Jorge Nocedal. "Optimization Methods for Large-Scale Machine Learning". In: *SIAM Review* 60.2 (2018), pp. 223–311.
-  Cybenko, G. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314.
-  Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159.
-  He, Kaiming et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.
-  Huang, G. et al. "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, July 2017, pp. 2261–2269.
-  Kingma, Diederik P. and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017.
-  Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012.
-  Lu, Lu et al. "DeepXDE: A deep learning library for solving differential equations". In: *SIAM Review* 63 (Feb. 2021), pp. 208–228.

## References II

-  Nocedal, Jorge and Stephen J. Wright. *Numerical Optimization*. 2e. New York, NY, USA: Springer, 2006.
-  Robbins, Herbert and Sutton Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407.
-  Sutskever, Ilya et al. "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147.

# Cross Entropy + KL Divergence

## Entropy (Expected Surprise!)

$$H(p) = \sum_{x \in \mathcal{X}} p(x) \log \left( \frac{1}{p(x)} \right)$$

## Cross Entropy

$$H(p, q) = \sum_{x \in \mathcal{X}} p(x) \log \left( \frac{1}{q(x)} \right)$$

## Kullback-Leibler Divergence (Expected Relative Entropy)

$$\begin{aligned} D_{KL}(p||q) &= \sum_{x \in \mathcal{X}} p(x) \log \left( \frac{p(x)}{q(x)} \right) \\ &= H(p, q) - H(p) \end{aligned}$$

# Cross Entropy + Maximum Likelihood

## Setup

- parameterized estimated probability distribution  $q_{\theta}$  (e.g., DNN)
- empirical distribution  $p$  (e.g., labels)
- $N$  outcomes (e.g., training samples)

## Likelihood

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{x \in \mathcal{X}} q_{\boldsymbol{\theta}}(x)^{Np(x)}$$

## Cross Entropy

$$-\frac{1}{N} \log \mathcal{L}(\boldsymbol{\theta}) = -\sum_{x \in \mathcal{X}} p(x) \log q_{\boldsymbol{\theta}}(x) = H(p, q).$$

# Cross Entropy Minimization

**Claim:**  $H(p, q)$  is minimized when  $p$  and  $q$  are the same distributions; that is, show

$$H(p) \leq H(p, q).$$

This is also known as Gibb's inequality.

**Sketch of Proof:** significant simplifying assumption:  $p(x), q(x) > 0$  for all  $x \in \mathcal{X}$ .

(Step 1): Show  $D_{KL}(p, q) = H(q, p) - H(p) \geq 0$

$$\begin{aligned} D_{KL}(p, q) &= - \sum_{x \in \mathcal{X}} p(x) \log \left( \frac{q(x)}{p(x)} \right) \\ &\geq - \sum_{x \in \mathcal{X}} p(x) \left( \frac{q(x)}{p(x)} - 1 \right) \\ &= - \sum_{x \in \mathcal{X}} q(x) + \sum_{x \in \mathcal{X}} p(x) \\ &= 0. \end{aligned}$$

(Step 2): Show  $D_{KL}(p, q) = 0$  if and only if  $p = q$ . This is fairly straightforward with the simplifying assumption.