Machine Learning

A. M. Sadeghzadeh, Ph.D.

Sharif University of Technology
Computer Engineering Department (CE)
Trustworthy and Secure AI Lab (TSAIL)

October 24, 2025

Most slides have been adapted from Bhiksha Raj, 11-785, CMU 2020, Fei Fei Li, cs231n, Stanford 2017, and Soleymani, CE40717, Sharif 2020.

Schedule

Machine Learning

Definition of machine learning

Tom Mitchell (1998)

A computer program is said to **learn** from **experience E** with respect to some class of **tasks T** and **performance measure P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**.

Example

Task: Classifying emails as spam or not spam
Experience: Watching label of emails as spam or not spam
Performance metric: The number (or fraction) of emails correctly classified as spam

Type of Learning

- **Supervised learning**
  Predicting a target variable for which we get to see examples

Type of Learning

- **Supervised learning**
  Predicting a target variable for which we get to see examples
- **Unsupervised learning**
  Revealing structure in the observed data

Type of Learning

- **Supervised learning**
  Predicting a target variable for which we get to see examples
- **Unsupervised learning**
  Revealing structure in the observed data
- **Reinforcement learning**
  Partial (indirect) feedback, no explicit guidance
  Given rewards for a sequence of moves to learn a policy and utility functions

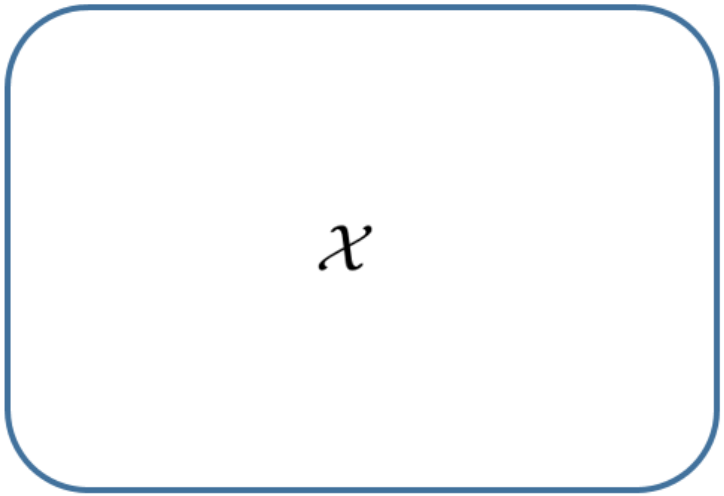
Supervised Learning

Components of supervised learning (function approximation)

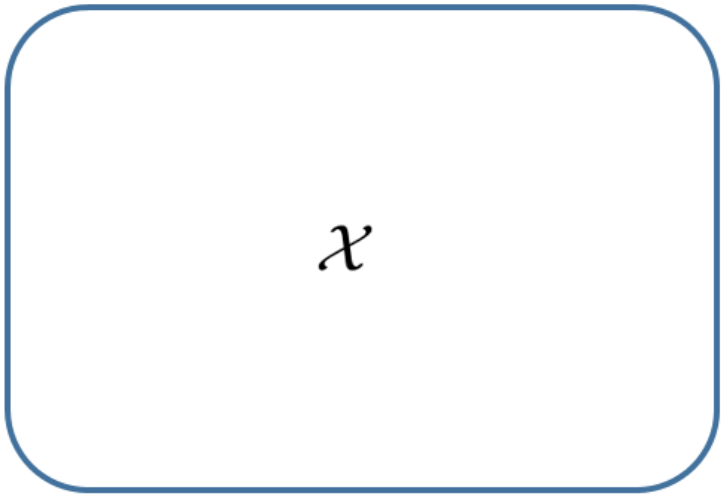
Concept

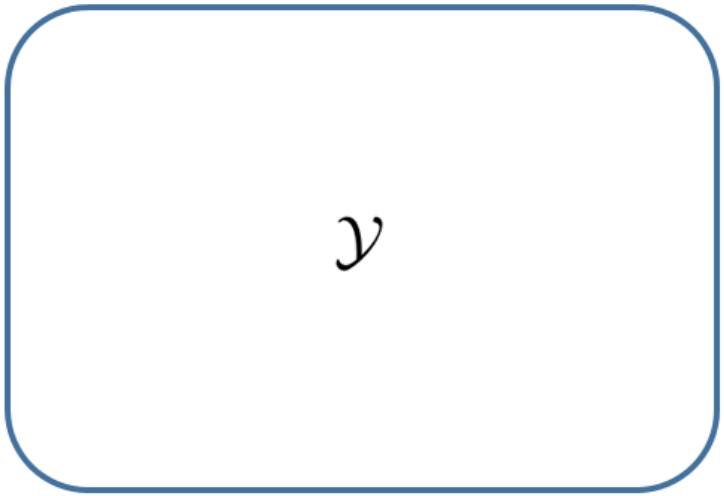
- Unknown target function $f : \mathcal{X} \to \mathcal{Y}$
- $\mathcal{X}$: Input space
- $\mathcal{Y}$: Output space

Components of supervised learning (function approximation)

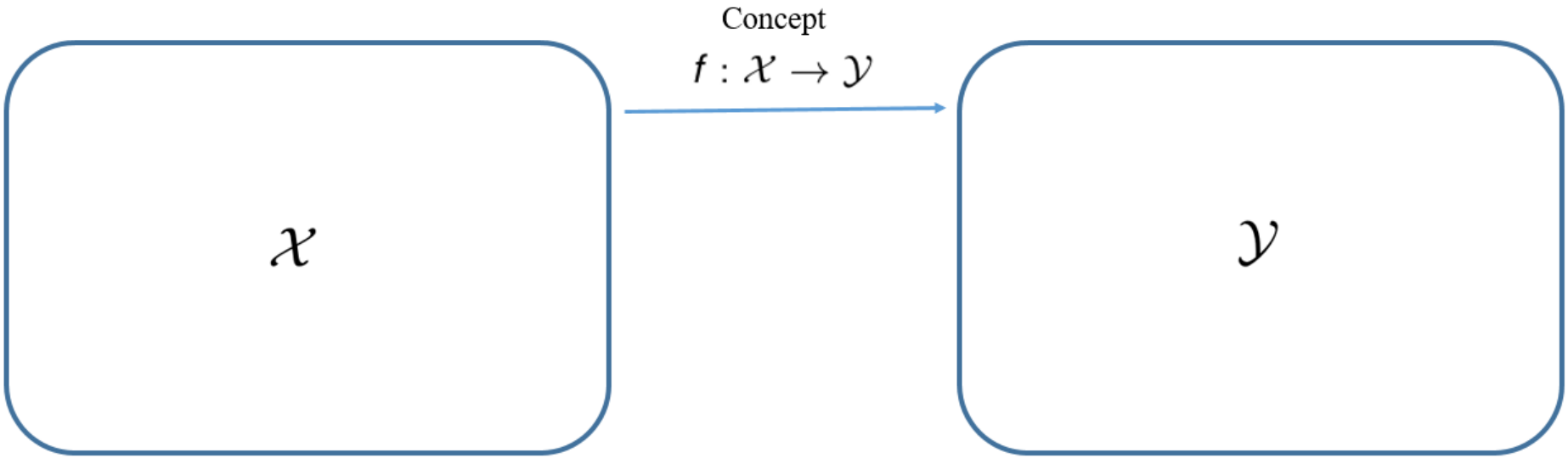


Components of supervised learning (function approximation)

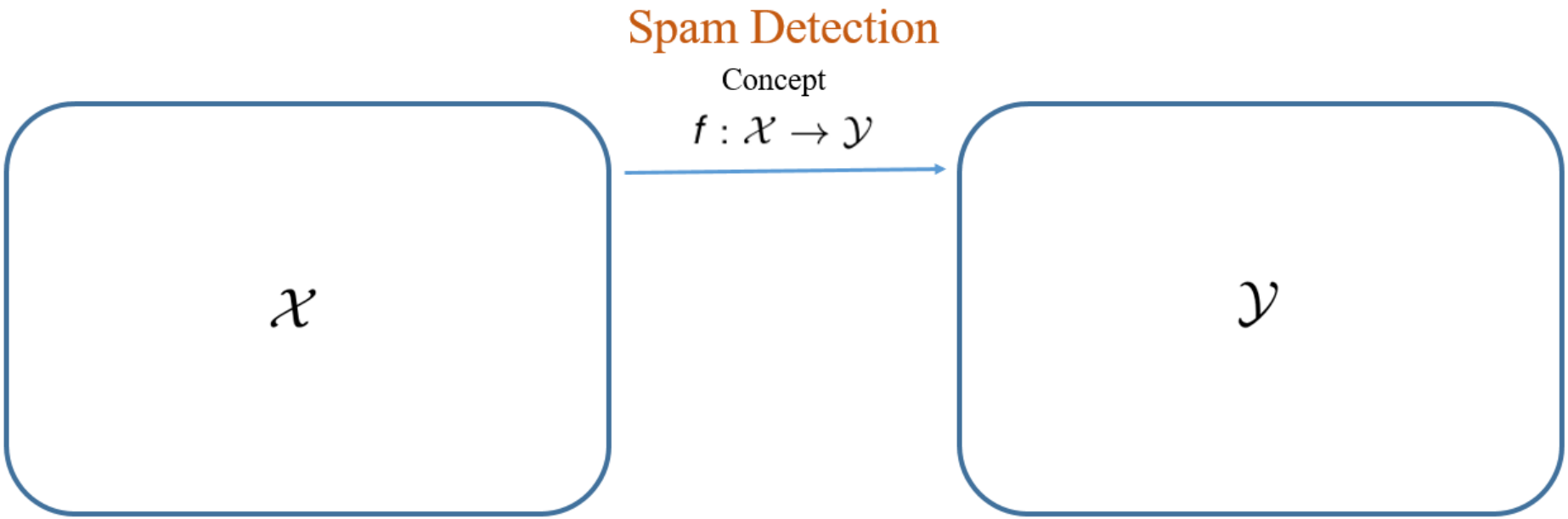




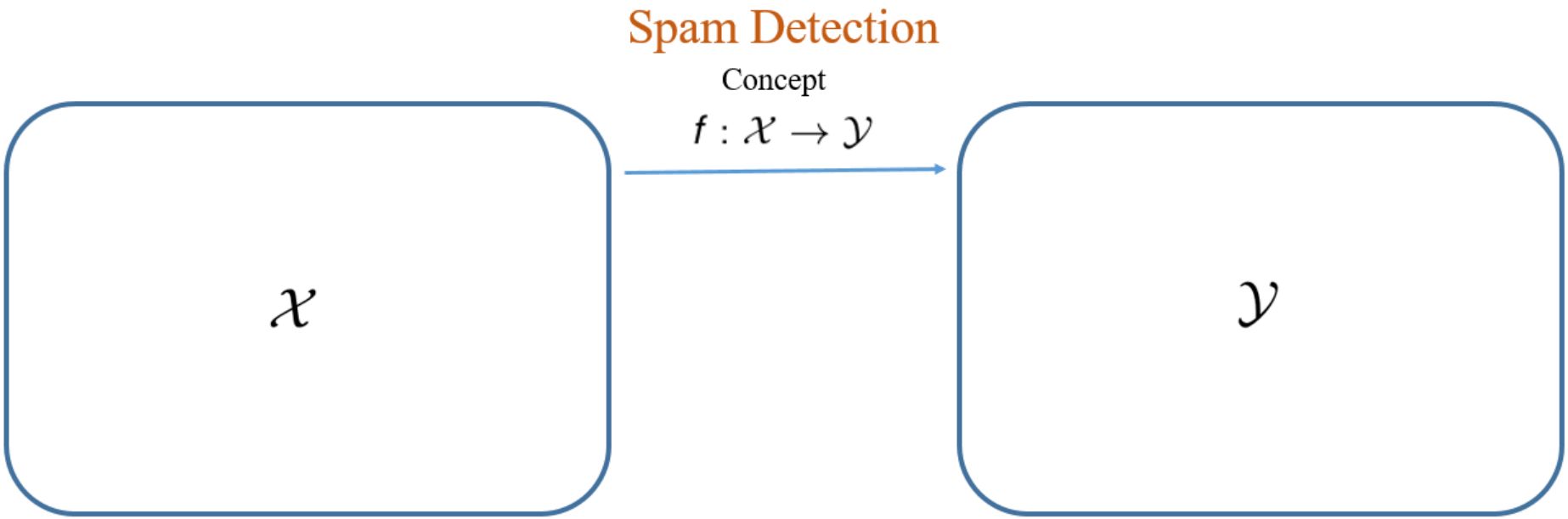
Components of supervised learning (function approximation)



Components of supervised learning (function approximation)

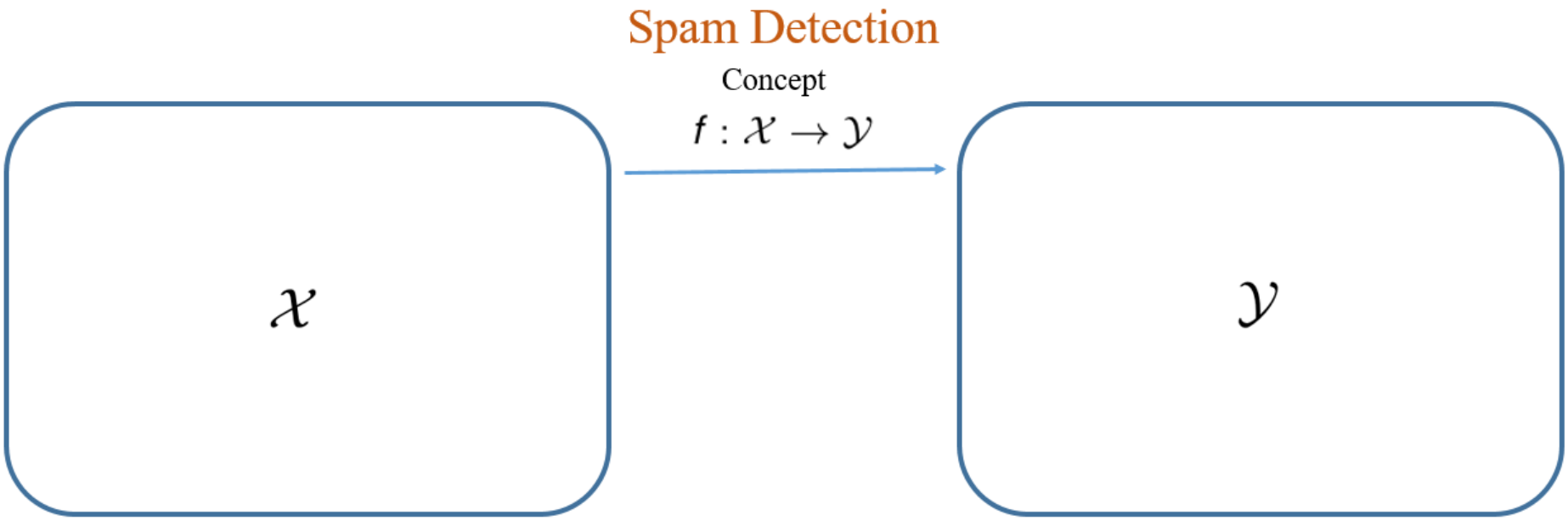


Components of supervised learning (function approximation)



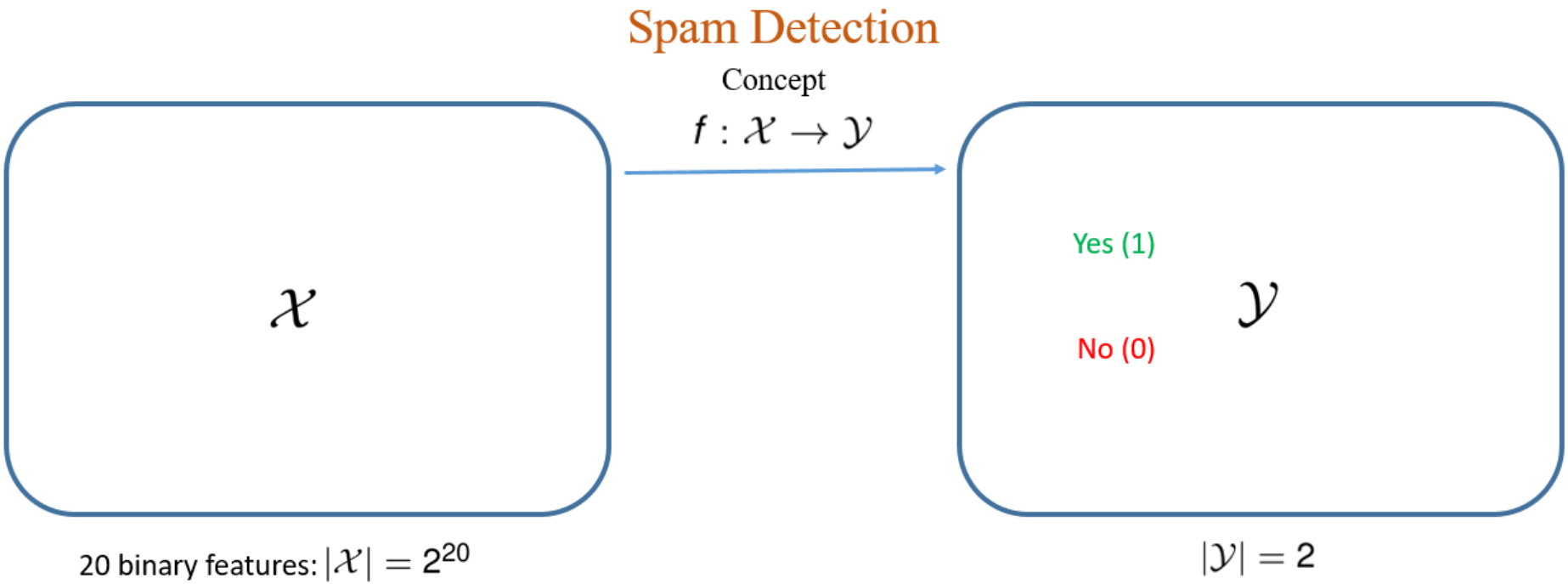
20 binary features:

Components of supervised learning (function approximation)

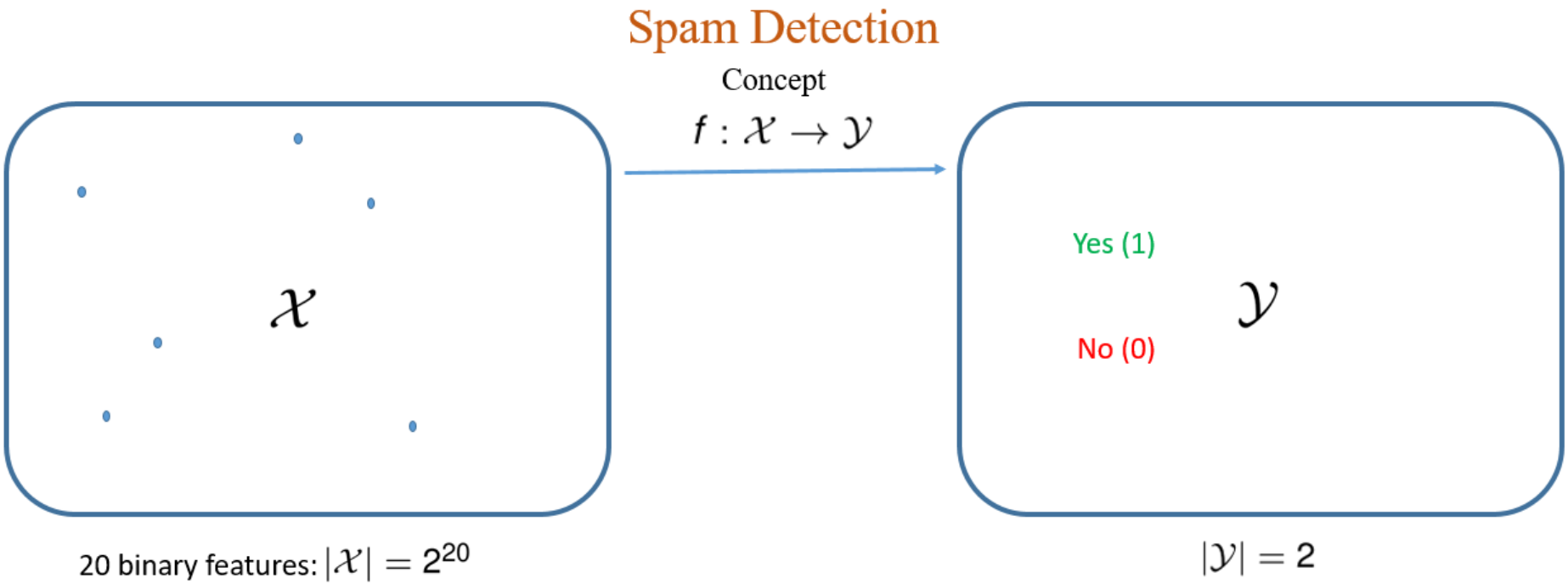


20 binary features: $|\mathcal{X}| = 2^{20}$

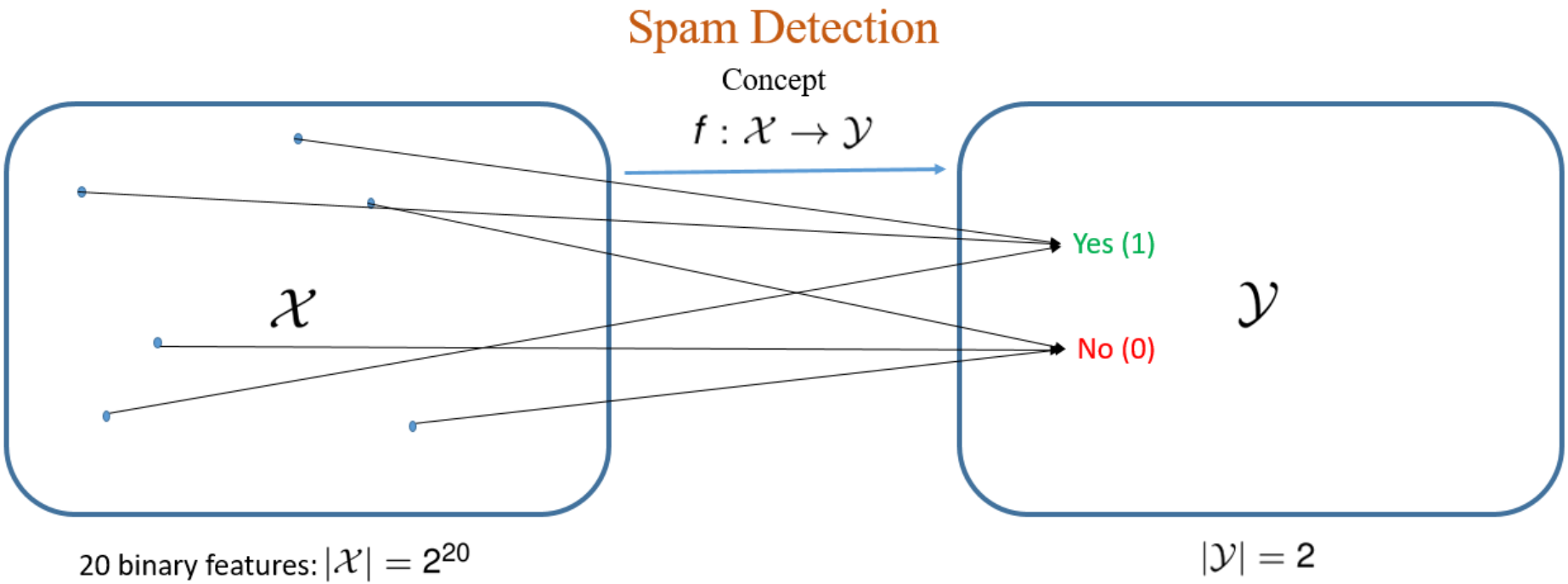
Components of supervised learning (function approximation)



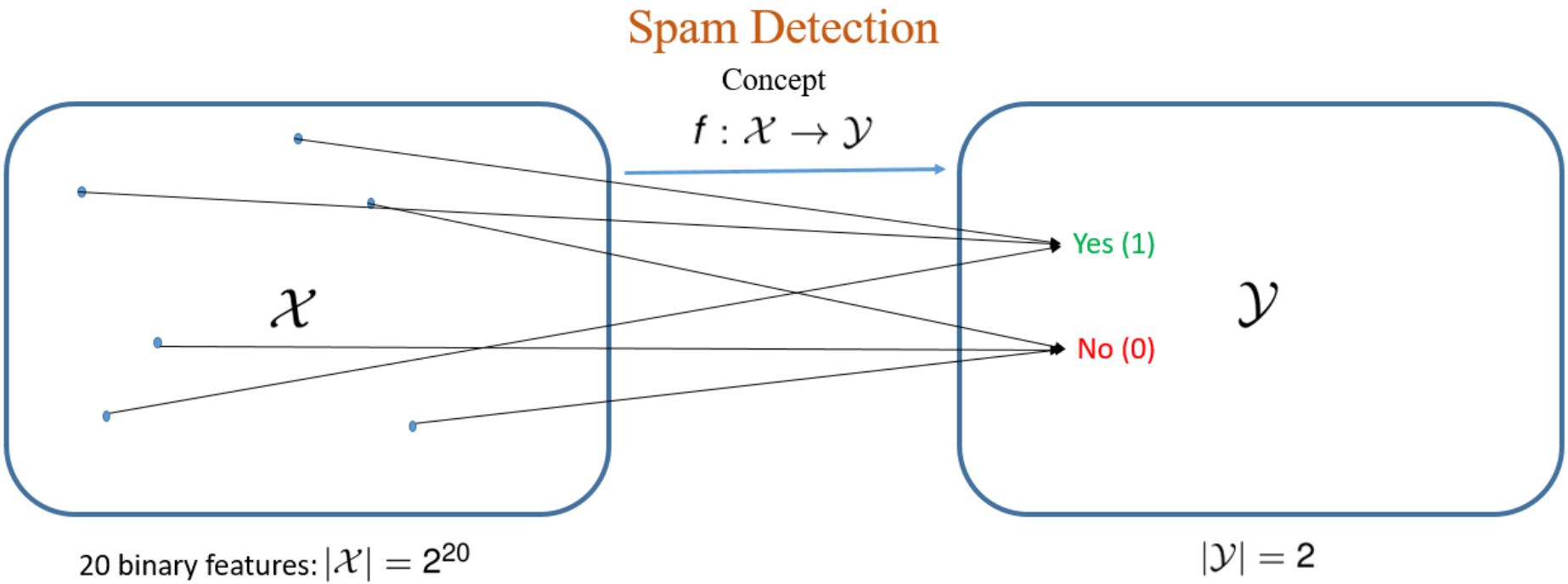
Components of supervised learning (function approximation)



Components of supervised learning (function approximation)

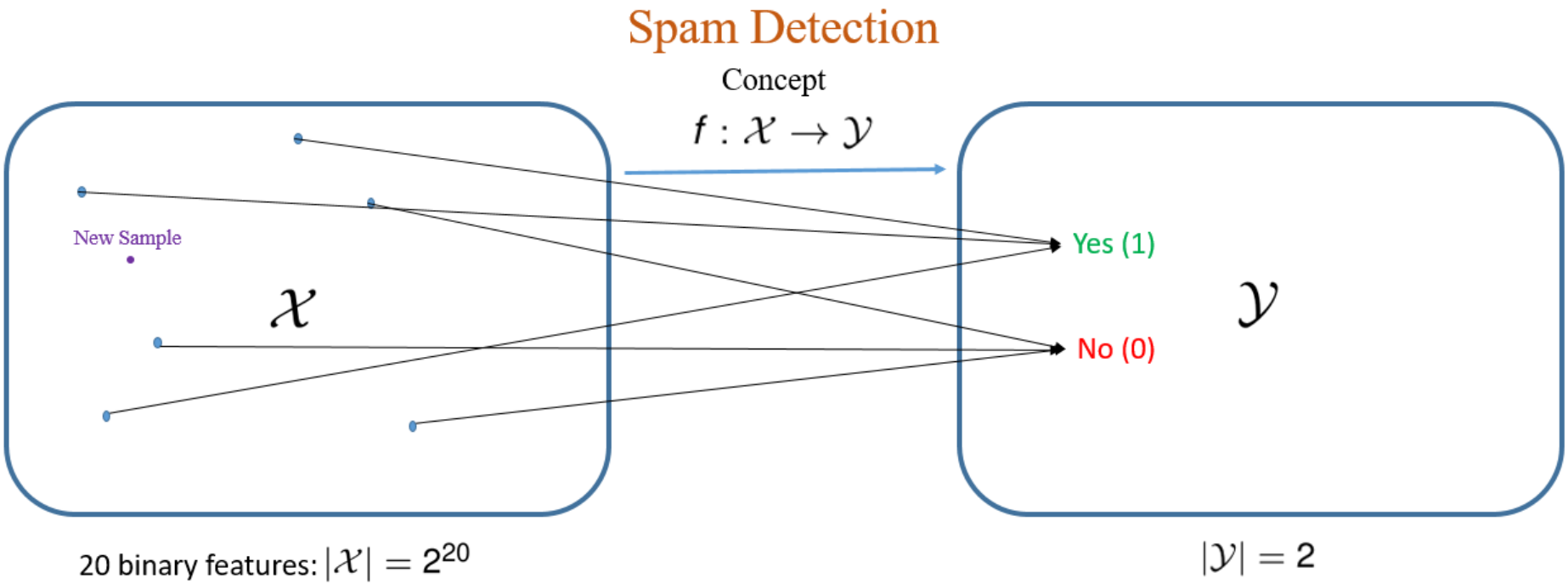


Components of supervised learning (function approximation)



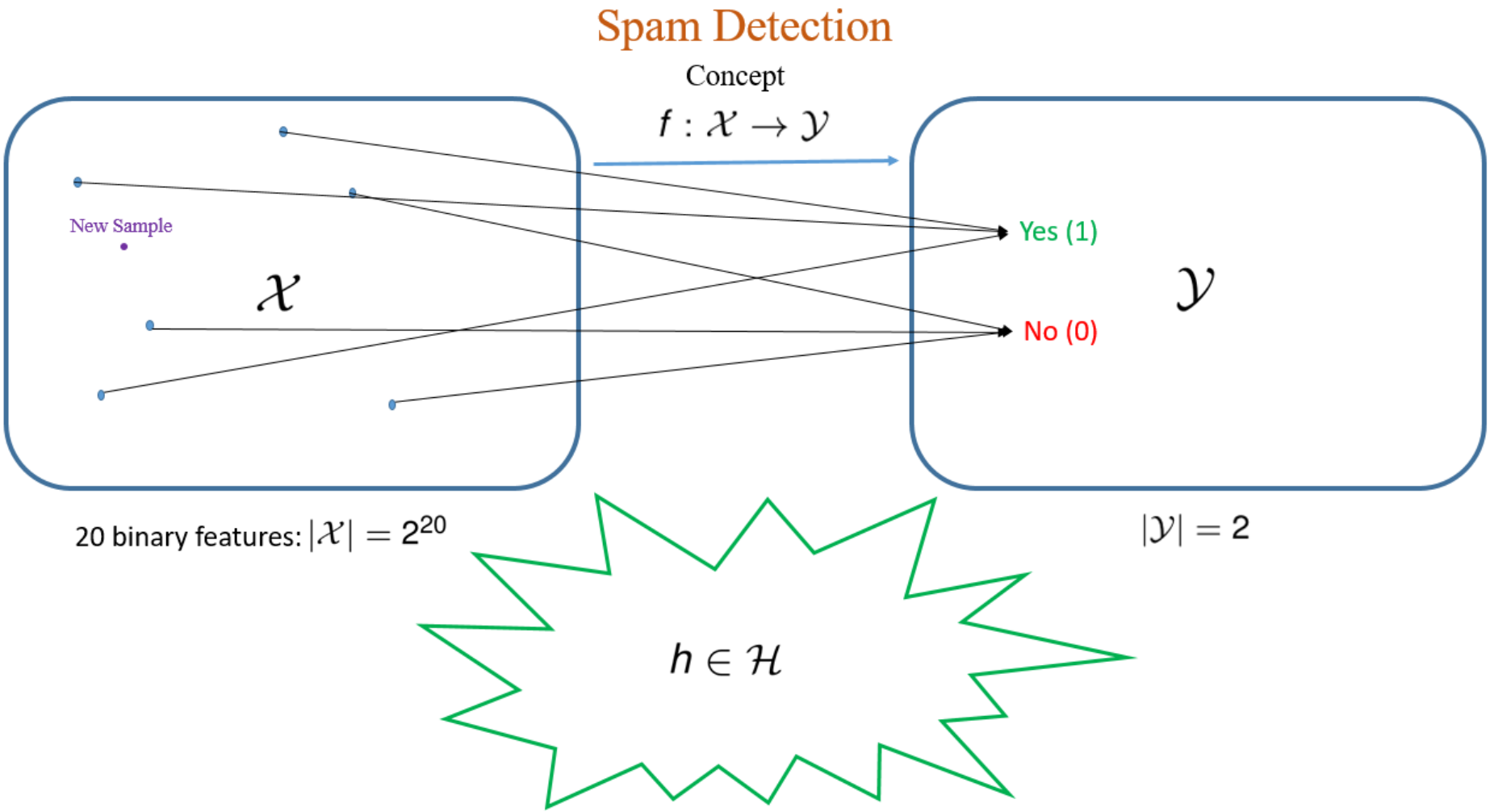
Goal: finding function $h : \mathcal{X} \to \mathcal{Y}$ that approximates the target function f

Components of supervised learning (function approximation)



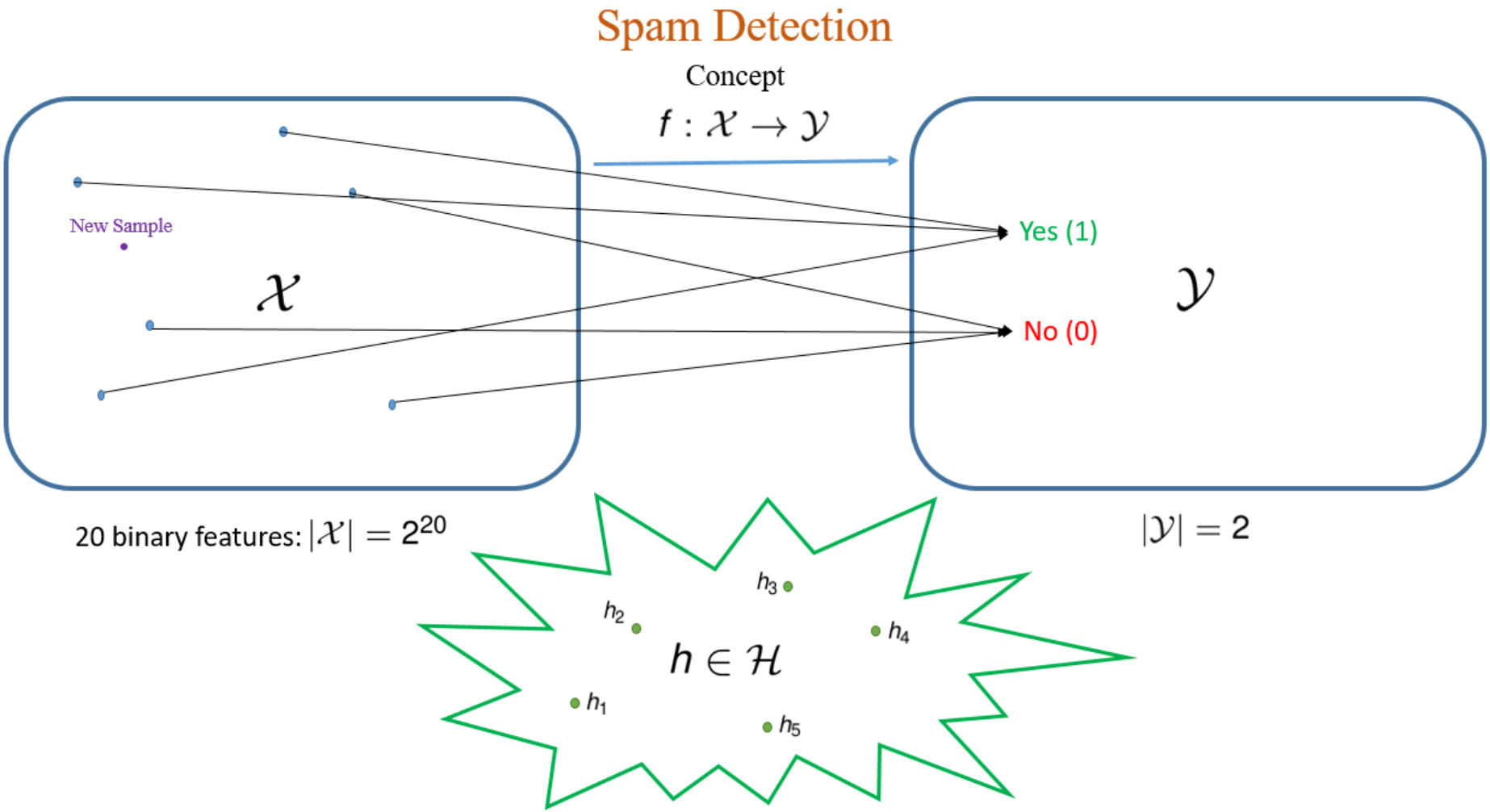
Goal: finding function $h : \mathcal{X} \to \mathcal{Y}$ that approximates the target function f (Generalization)

Components of supervised learning (function approximation)

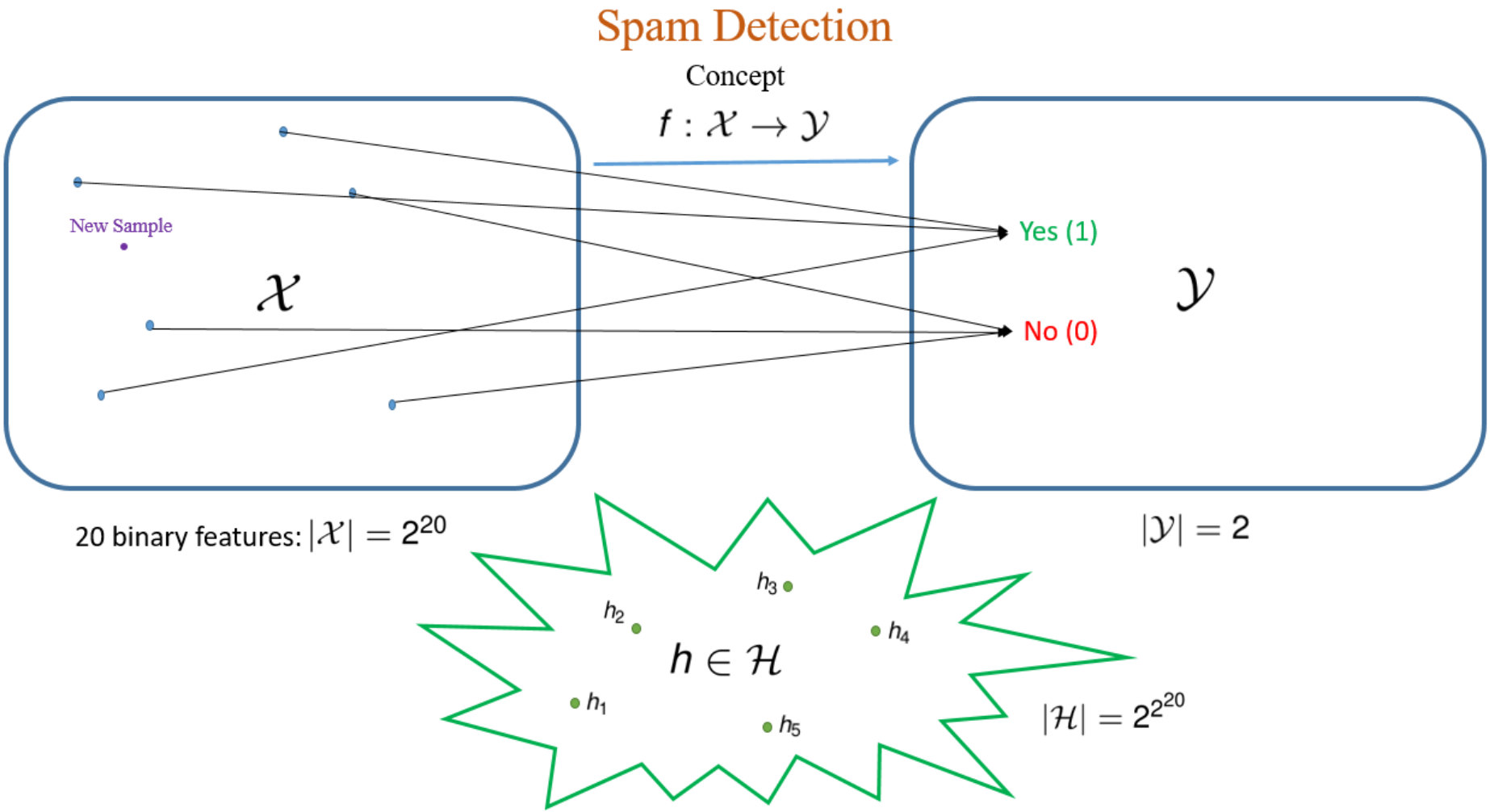


Goal: finding function $h : \mathcal{X} \to \mathcal{Y}$ that approximates the target function f (Generalization)

Components of supervised learning (function approximation)

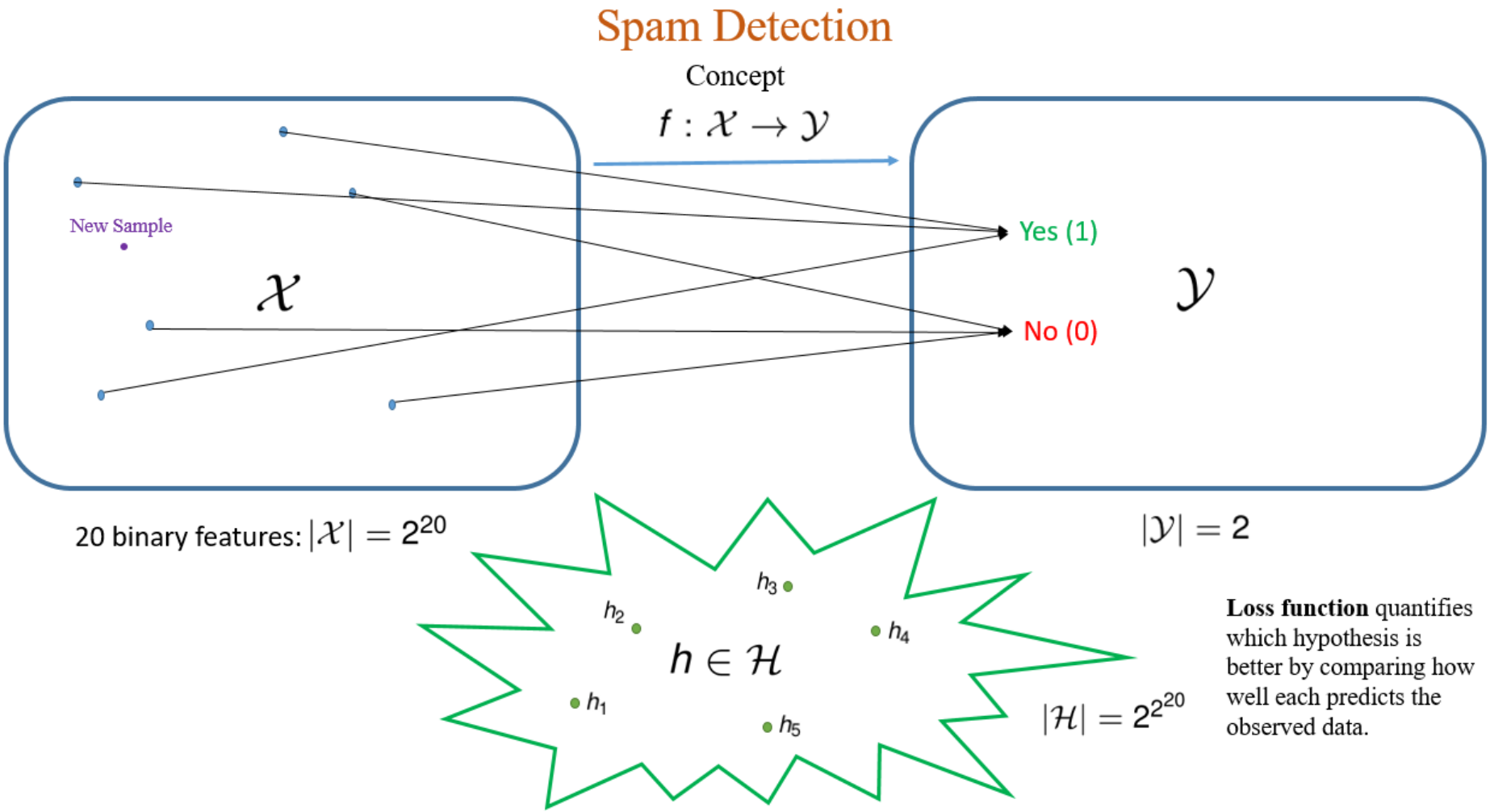


Components of supervised learning (function approximation)

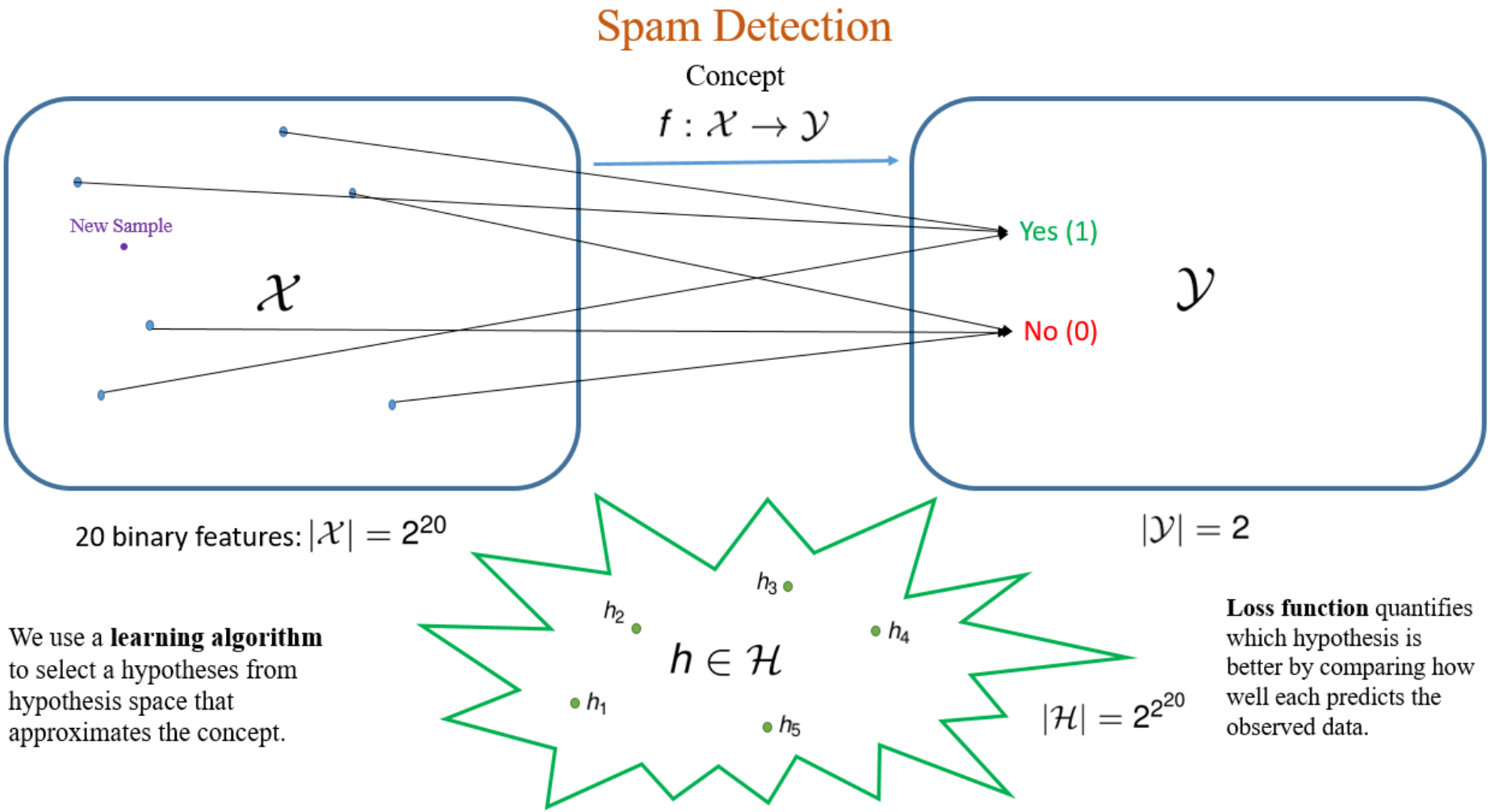


Goal: finding function $h : \mathcal{X} \to \mathcal{Y}$ that approximates the target function f (Generalization)

Components of supervised learning (function approximation)



Components of supervised learning (function approximation)



We use a **learning algorithm** to select a hypotheses from hypothesis space that approximates the concept.

Loss function quantifies which hypothesis is better by comparing how well each predicts the observed data.

Goal: finding function $h : \mathcal{X} \to \mathcal{Y}$ that approximates the target function f (Generalization)

Components of supervised learning (function approximation)

Concept

- Unknown target function $f : \mathcal{X} \to \mathcal{Y}$
- $\mathcal{X}$: Input space
- $\mathcal{Y}$: Output space

Training data

- Samples from the concept $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(N)}, y^{(N)})\}$

Components of supervised learning (function approximation)

Concept

- Unknown target function $f : \mathcal{X} \to \mathcal{Y}$
- $\mathcal{X}$: Input space
- $\mathcal{Y}$: Output space

Training data

- Samples from the concept $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(N)}, y^{(N)})\}$

Hypotheses space

- Pick a hypotheses $h : \mathcal{X} \to \mathcal{Y}$ from hypotheses space $h \in \mathcal{H}$ (e.g., linear functions) that approximates the target function f

Components of supervised learning (function approximation)

Concept

- Unknown target function $f : \mathcal{X} \to \mathcal{Y}$
- $\mathcal{X}$: Input space
- $\mathcal{Y}$: Output space

Training data

- Samples from the concept $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(N)}, y^{(N)})\}$

Hypotheses space

- Pick a hypotheses $h : \mathcal{X} \to \mathcal{Y}$ from hypotheses space $h \in \mathcal{H}$ (e.g., linear functions) that approximates the target function f

Learning algorithm

- We use a learning algorithm to select a hypotheses from hypothesis space that approximates the concept

Components of supervised learning

Learning model composed of

- A hypothesis set
- A learning algorithm

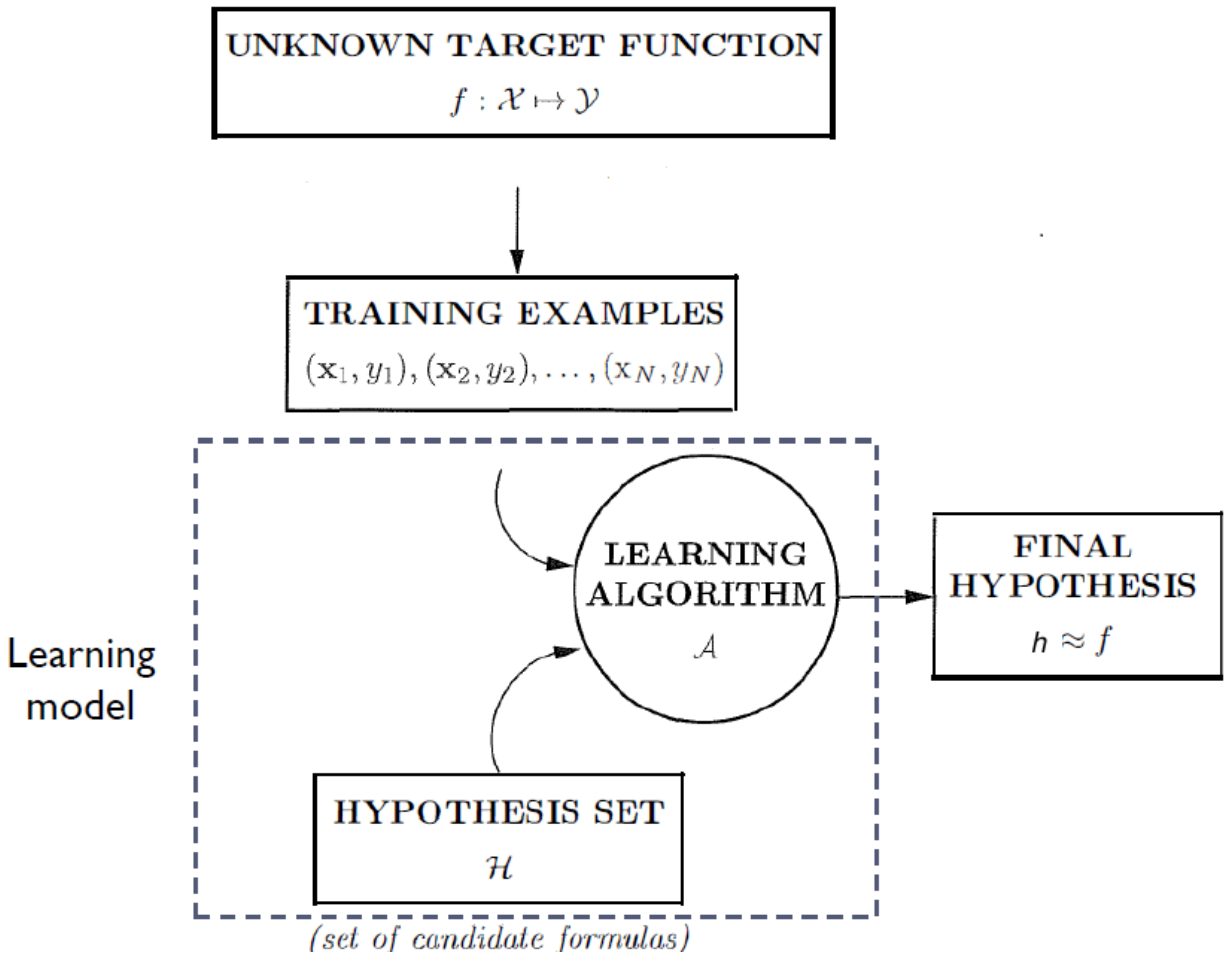


Figure: https://work.caltech.edu/telecourse. html

Generalization

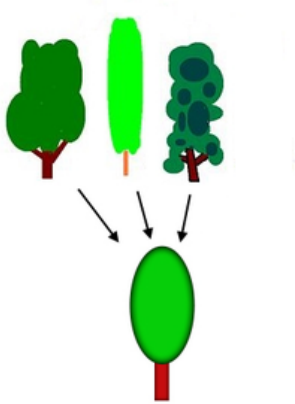
Hypotheses evaluation

How well h generalizes to unseen examples.

Test data

Samples from the concept $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(N)}, y^{(N)})\}$

Do not exist in the training data



Generalization

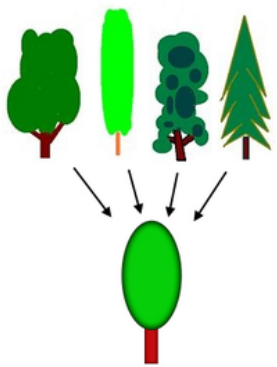
Hypotheses evaluation

How well h generalizes to unseen examples.

Test data

Samples from the concept $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(N)}, y^{(N)})\}$

Do not exist in the training data



Independent and identically distributed (i.i.d.)

A collection of random variables is independent and identically distributed if each random variable has the same probability distribution as the others and all are mutually independent.

- $\forall i, x^{(i)} \sim \mathcal{D}$ (Identically Distributed)
- $\forall i \neq j, \mathcal{P}(x^{(i)}, x^{(j)}) = \mathcal{P}(x^{(i)})\mathcal{P}(x^{(j)})$ (Independently Distributed)

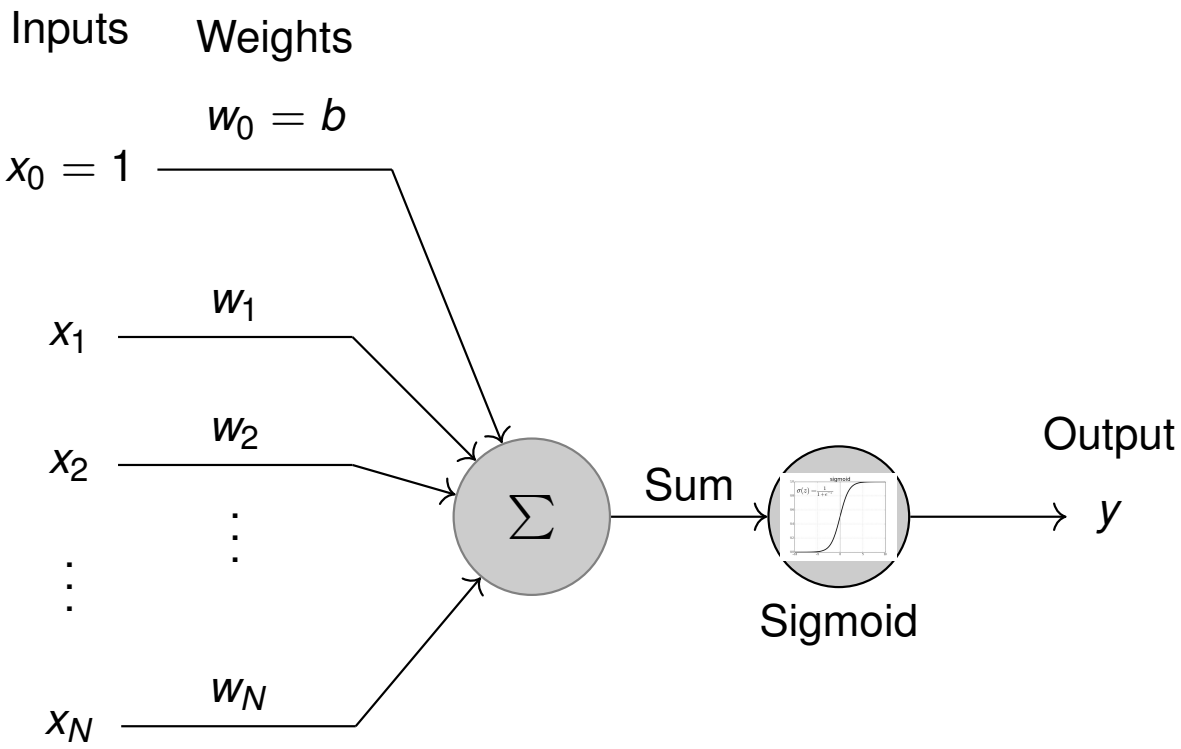
Machine learning algorithms have focused primarily on sets of data points that were assumed to be independent and identically distributed (i.i.d.).

Logistic Regression

Logistic regression

It is the perceptron with a sigmoid activation

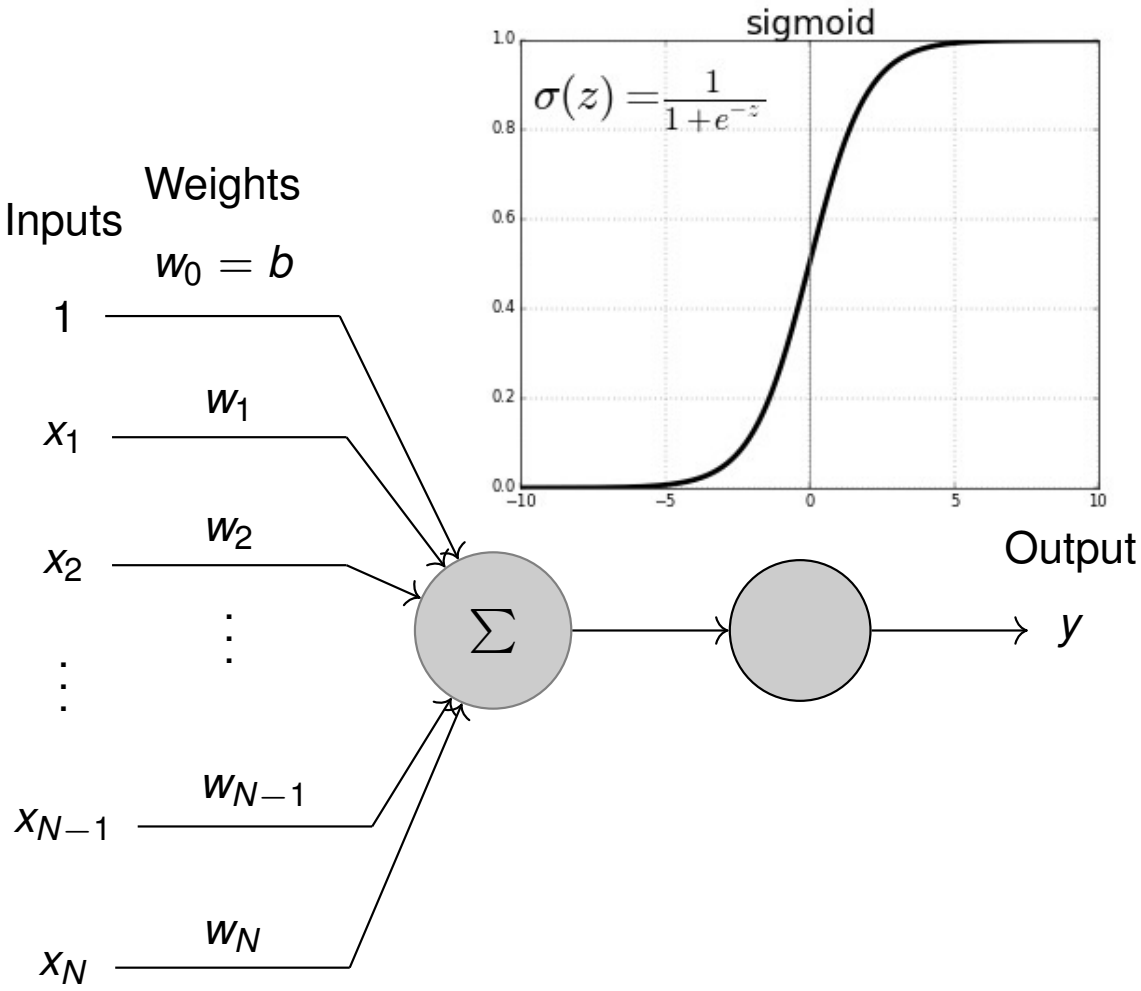
- It actually computes the probability that the input belongs to class 1



Logistic regression

It is the perceptron with a sigmoid activation

- It actually computes the probability that the input belongs to class 1



$$z = \sum_i w_i x_i$$

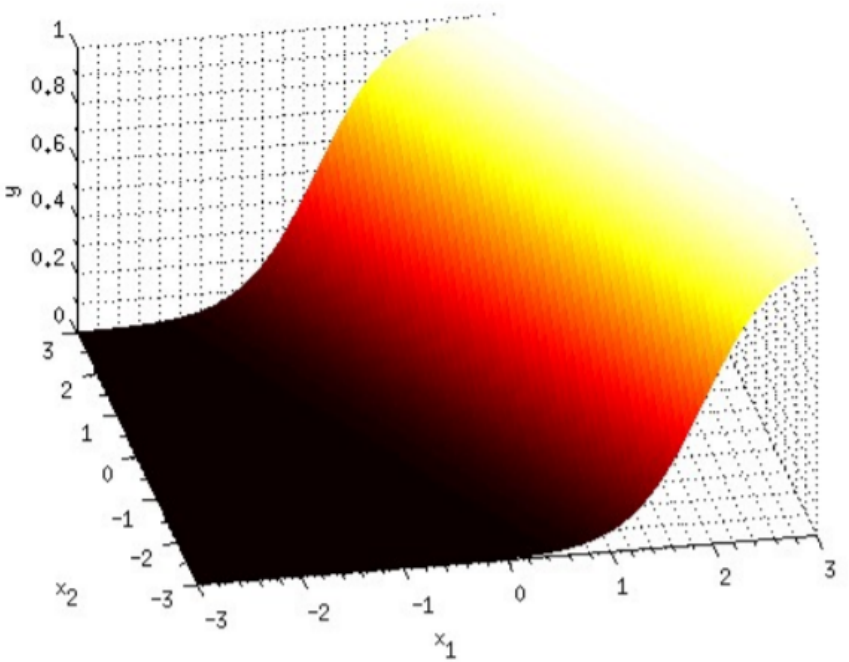
$$p(y = 1|x) = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

$$y = \begin{cases} 1 & \text{if} \quad \sigma(z) \geq \frac{1}{2} \\ 0 & \text{if} \quad \sigma(z) < \frac{1}{2} \end{cases}$$

Logistic regression

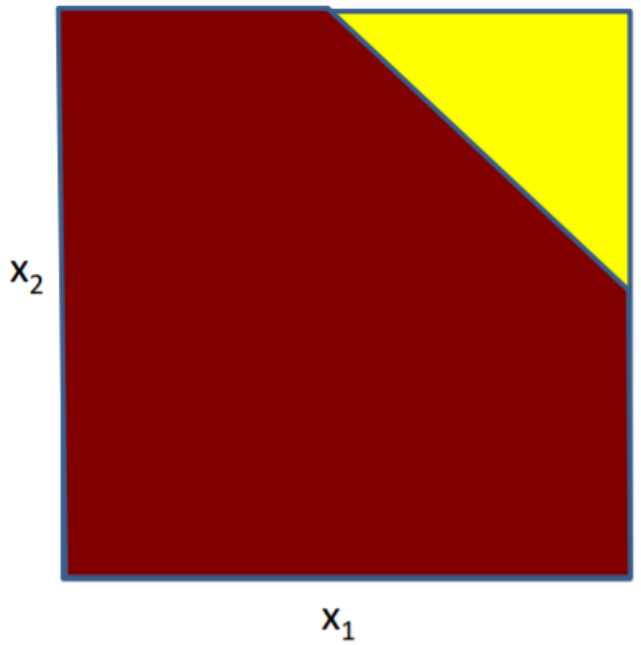
This the perceptron with a sigmoid activation

- It actually computes the probability that the input belongs to class 1



When X is a 2-D variable

Decision: y > 0.5?



$$P(Y = 1|X) = \frac{1}{1 + \exp\left(-\sum_i w_i x_i - b\right)}$$

Vectorization

$$z = \sum_i w_i x_i = \boldsymbol{w}^T \boldsymbol{x} = \begin{bmatrix} w_0 = b & w_1 & w_2 & \dots & w_{N-1} & w_N \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}$$

$$f(\boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})}$$

Loss Function

Loss function

- Loss function quantifies our unhappiness with the scores across the training data.

Loss function

- Loss function quantifies our unhappiness with the scores across the training data.
- A loss function tells how good our current classifier is
  - Given a dataset of examples $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$
  - $x^{(i)}$ is data and $y^{(i)}$ is binary label.

Loss function

- Loss function quantifies our unhappiness with the scores across the training data.
- A loss function tells how good our current classifier is
  - Given a dataset of examples $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$
  - $x^{(i)}$ is data and $y^{(i)}$ is binary label.
- Loss over the dataset is a average of loss over examples:

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_{i} L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)})$$

Loss function

- Loss function quantifies our unhappiness with the scores across the training data.
- A loss function tells how good our current classifier is
  - Given a dataset of examples $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$
  - $x^{(i)}$ is data and $y^{(i)}$ is binary label.
- Loss over the dataset is a average of loss over examples:

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_{i} L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)})$$

- Empirical loss L is only an empirical approximation of the true loss

Loss function

- Loss function quantifies our unhappiness with the scores across the training data.
- A loss function tells how good our current classifier is
  - Given a dataset of examples $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$
  - $x^{(i)}$ is data and $y^{(i)}$ is binary label.
- Loss over the dataset is a average of loss over examples:

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_{i} L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)})$$

- Empirical loss L is only an empirical approximation of the true loss
- **Learning**
  - Estimate the parameters to minimize the empirical loss L

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}}\ L(f(X, \boldsymbol{w}), \boldsymbol{y})$$

Cross entropy

If we have two separate probability distributions $P(x)$ and $Q(x)$ over the same random variable x , we can measure how different these two distributions are using the Kullback-Leibler (KL) divergence:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}[\log \frac{P(x)}{Q(x)}] = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)]$$

$$D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

Cross entropy

If we have two separate probability distributions $P(x)$ and $Q(x)$ over the same random variable x , we can measure how different these two distributions are using the Kullback-Leibler (KL) divergence:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}[\log \frac{P(x)}{Q(x)}] = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)]$$

$$D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

- It is the extra amount of information needed to send a message containing symbols drawn from probability distribution P , when we use a code that was designed to minimize the length of messages drawn from probability distribution Q.

Cross entropy

If we have two separate probability distributions $P(x)$ and $Q(x)$ over the same random variable x , we can measure how different these two distributions are using the Kullback-Leibler (KL) divergence:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}[\log \frac{P(x)}{Q(x)}] = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)]$$

$$D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

- It is the extra amount of information needed to send a message containing symbols drawn from probability distribution P , when we use a code that was designed to minimize the length of messages drawn from probability distribution Q.
- KL divergence is non-negative and measures the difference between two distributions, it is often conceptualized as measuring some sort of distance between these distributions. However, it is not a true distance measure because it is not symmetric

Cross entropy

- Minimizing loss function

$$\underset{Q(x)}{\operatorname{argmin}} D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

Cross entropy

- Minimizing loss function

$$\underset{Q(x)}{\operatorname{argmin}} D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

- P(x) is the true distribution of the random variable *x* and it is fixed.

Cross entropy

- Minimizing loss function

$$\underset{Q(x)}{\operatorname{argmin}} D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

- P(x) is the true distribution of the random variable *x* and it is fixed.

$$\underset{Q(x)}{\operatorname{argmin}} - \sum_i p_i(x) \log q_i(x)$$

Cross entropy

- Minimizing loss function

$$\underset{Q(x)}{\operatorname{argmin}} D_{KL}(P||Q) = \sum_i p_i(x) \log p_i(x) - \sum_i p_i(x) \log q_i(x)$$

- P(x) is the true distribution of the random variable *x* and it is fixed.

$$\underset{Q(x)}{\operatorname{argmin}} -\sum_i p_i(x) \log q_i(x)$$

$$CrossEntropy(P(x), Q(x)) = -\sum_i p_i(x) \log q_i(x)$$

Logistic regression loss function

- Logistic Regression

$$P(y = 1|\boldsymbol{x}^{(i)}; \boldsymbol{w}) = f(\boldsymbol{x}^{(i)}, \boldsymbol{w})$$
$$P(y = 0|\boldsymbol{x}^{(i)}; \boldsymbol{w}) = 1 - f(\boldsymbol{x}^{(i)}, \boldsymbol{w})$$
$$y^{(i)} \in \{0, 1\}$$

Logistic regression loss function

- Logistic Regression

$$P(y = 1|\boldsymbol{x}^{(i)}; \boldsymbol{w}) = f(\boldsymbol{x}^{(i)}, \boldsymbol{w})$$
$$P(y = 0|\boldsymbol{x}^{(i)}; \boldsymbol{w}) = 1 - f(\boldsymbol{x}^{(i)}, \boldsymbol{w})$$
$$y^{(i)} \in \{0, 1\}$$

- Cross entropy as the loss function

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)})$$
$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i -y^{(i)} \log f(\boldsymbol{x}^{(i)}, \boldsymbol{w}) - (1 - y^{(i)}) \log(1 - f(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

Logistic regression loss function

- Logistic Regression

$$P(y = 1|\boldsymbol{x}^{(i)}; \boldsymbol{w}) = f(\boldsymbol{x}^{(i)}, \boldsymbol{w})$$
$$P(y = 0|\boldsymbol{x}^{(i)}; \boldsymbol{w}) = 1 - f(\boldsymbol{x}^{(i)}, \boldsymbol{w})$$
$$y^{(i)} \in \{0, 1\}$$

- Cross entropy as the loss function

$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)})$$
$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i -y^{(i)} \log f(\boldsymbol{x}^{(i)}, \boldsymbol{w}) - (1 - y^{(i)}) \log(1 - f(\boldsymbol{x}^{(i)}, \boldsymbol{w}))$$

How do we find the best w?

Optimization

Derivative

- In 1-dimension, the derivative of a function gives the slope:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- In multiple dimensions, the gradient is the vector of (partial derivatives) along each dimension
- The direction of steepest descent is the negative gradient

- To decrease f at x:

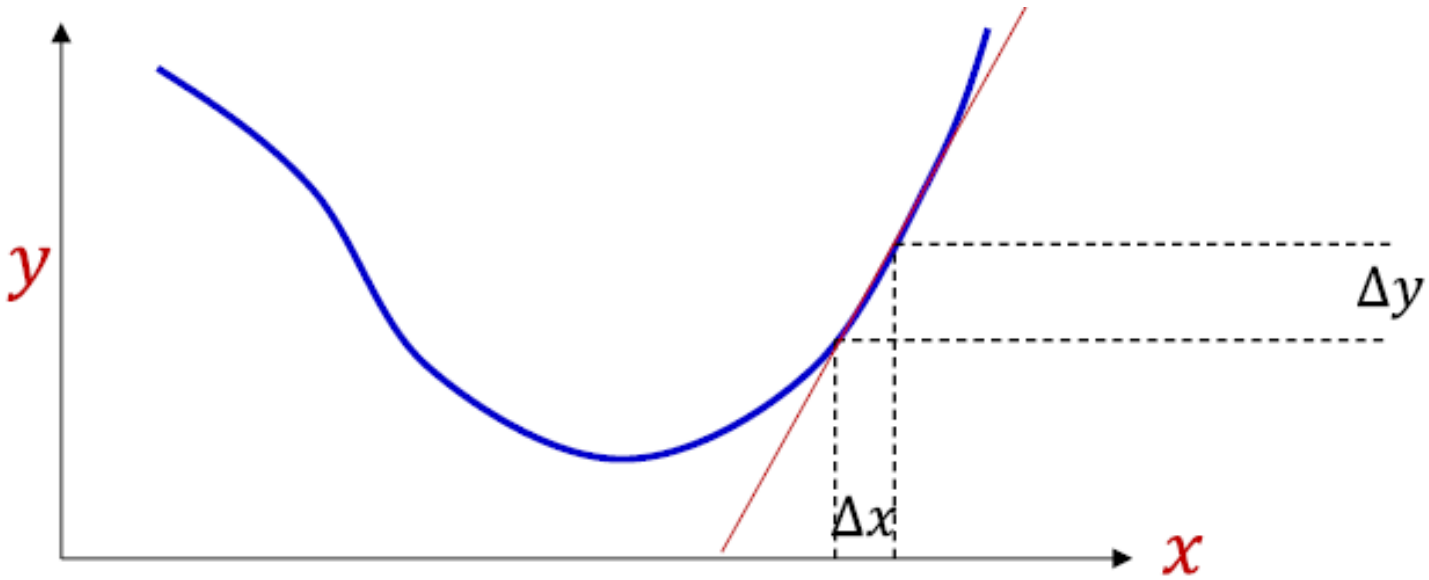
$$if \ \frac{df(x)}{dx} < 0 \Rightarrow f(x+h) < f(x) \Rightarrow f \text{ is decreasing in } x \Rightarrow \text{take a step in } \textbf{positive} \text{ direction}$$

$$if \ \frac{df(x)}{dx} > 0 \Rightarrow f(x+h) > f(x) \Rightarrow f \text{ is increasing in } x \Rightarrow \text{take a step in } \textbf{negative} \text{ direction}$$

  - Take a step in the reverse direction of derivative at x.

A brief note on derivatives...

- A derivative of a function at any point tells us how much a minute increment to the argument of the function will increment the value of the function
  - For any $y = f(x)$, expressed as a multiplier α to a tiny increment Δx to obtain the increments Δy to the output
    $$\Delta y = \alpha \Delta x$$
  - Based on the fact that at a fine enough resolution, any smooth, continuous function is locally linear at any point



Multivariate scalar function

- Scalar function of vector argument
  $y = f(\boldsymbol{x})$ ($f : \mathbb{R}^d \to \mathbb{R}$)

$$\Delta y = \alpha \Delta \boldsymbol{x}$$

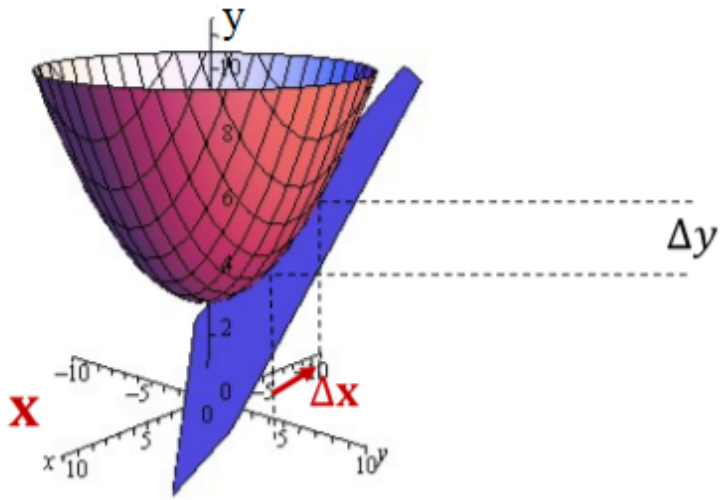
Multivariate scalar function

- Scalar function of vector argument
  $y = f(\boldsymbol{x})$ ($f : \mathbb{R}^d \rightarrow \mathbb{R}$)

$$\Delta y = \alpha \Delta \boldsymbol{x}$$

- Δx is now a vector

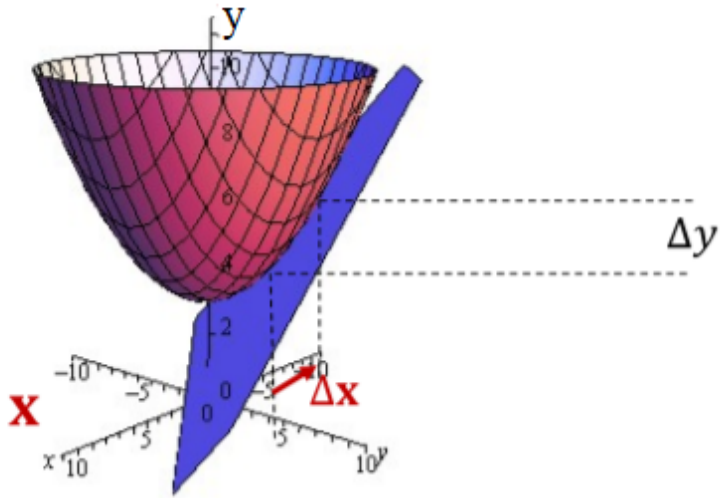
$$\Delta \boldsymbol{x} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_d \end{bmatrix}$$



Multivariate scalar function

- Scalar function of vector argument
  $y = f(\boldsymbol{x})$ $(f : \mathbb{R}^d \to \mathbb{R})$

$$\Delta y = \alpha \Delta \boldsymbol{x}$$



- Δx is now a vector

$$\Delta \boldsymbol{x} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_d \end{bmatrix}$$

  - Giving us that α is a row vector: $\alpha = \begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_d \end{bmatrix}$

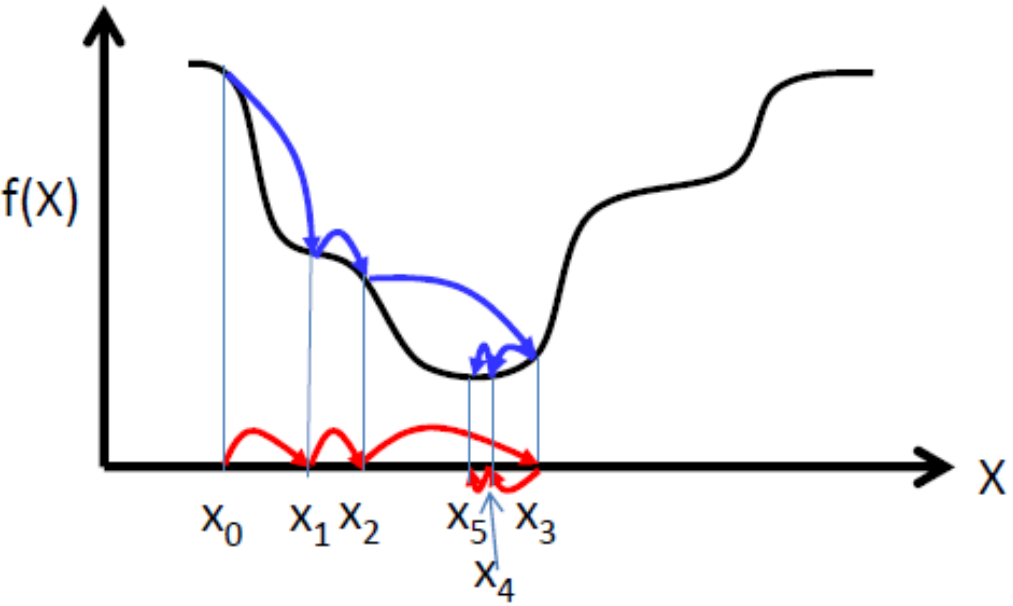
$$\Delta y = \alpha_1 \Delta x_1 + \alpha_2 \Delta x_2 + \dots + \alpha_d \Delta x_d$$

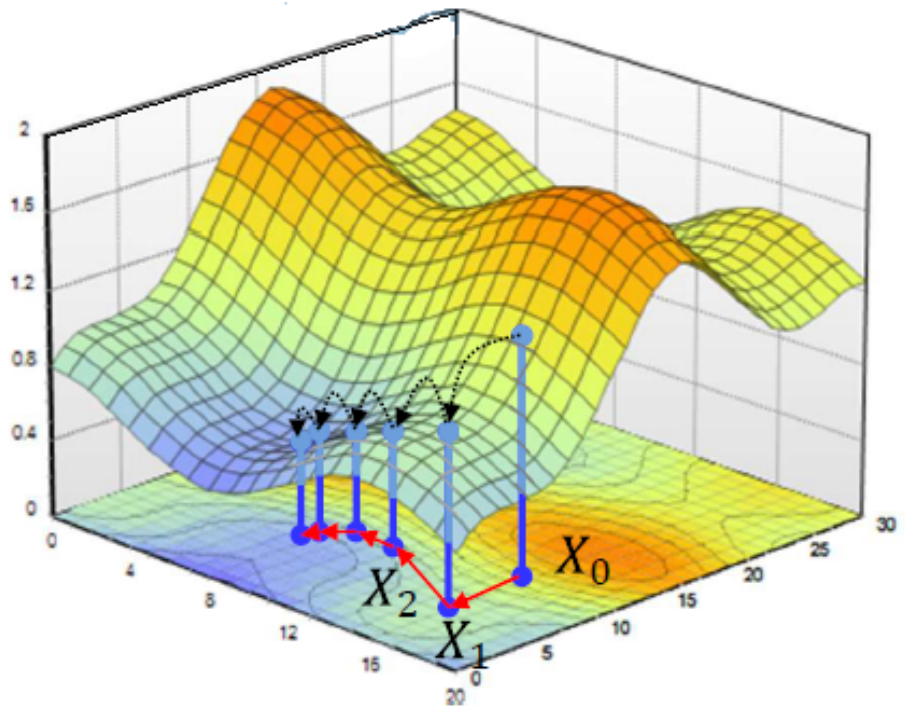
  - The partial derivative α_i gives us how y increments when only x_i is incremented.
  - Often represented as $\frac{\partial y}{\partial x_i}$

$$\Delta y = \frac{\partial y}{\partial x_1} \Delta x_1 + \frac{\partial y}{\partial x_2} \Delta x_2 + \dots + \frac{\partial y}{\partial x_d} \Delta x_d$$

Iterative solutions (Gradient Descent)

- Iterative solutions
  - Start from an initial guess X_0 for the optimal X
  - Update the guess towards a (hopefully) "better" value of $f(X)$
  - Stop when $f(X)$ no longer decreases
- Problems
  - Which direction to step in
  - How big must the steps be





Gradient descent

- The gradient descent method to find the minimum of a function iteratively

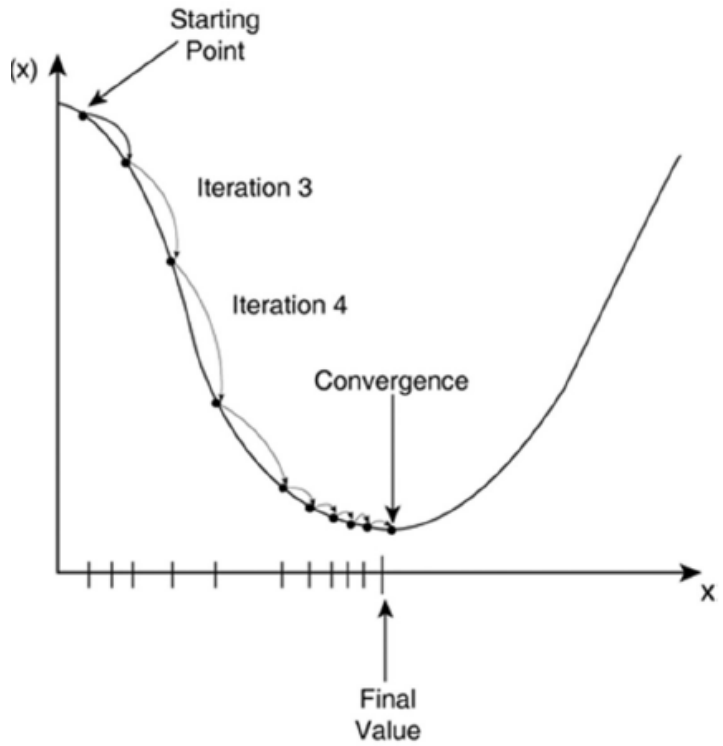
$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - \eta \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^t)$$

  - η is the "step size" (Also called "learning rate")

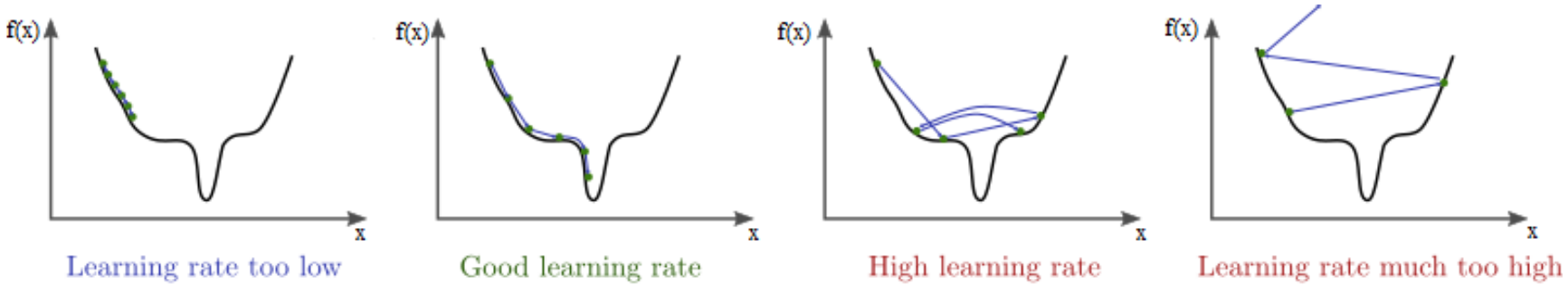
- The gradient descent algorithm converges when the following criterion is satisfied.

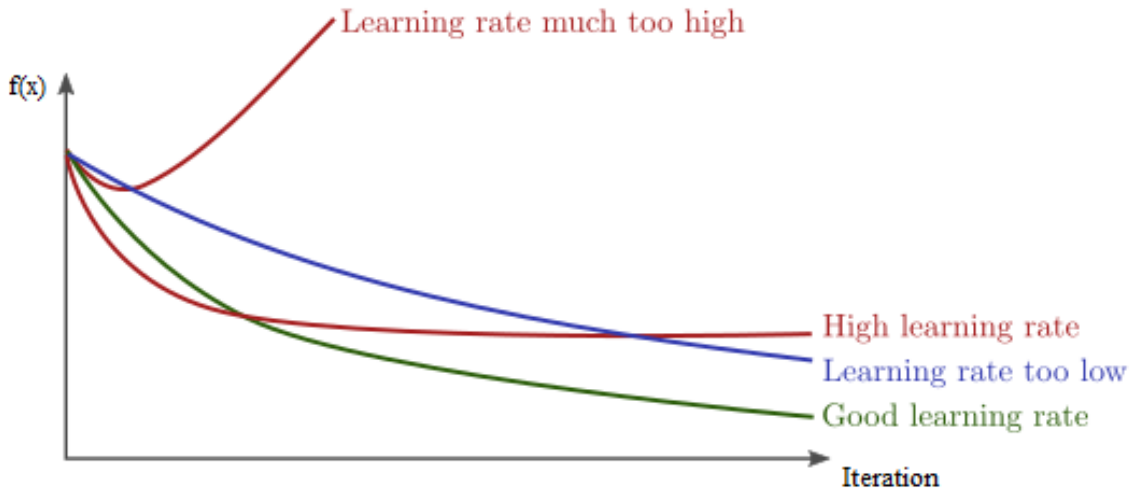
$$|f(\boldsymbol{x}^{t+k}) - f(\boldsymbol{x}^t)| < \epsilon$$

  - k is a hyperparameter.



Influence of step size





Regularization

Learning VS. Optimization

Generalization VS. Overfitting

- Ability to perform well on previously unobserved inputs is called **generalization**.

Learning VS. Optimization

Generalization VS. Overfitting

- Ability to perform well on previously unobserved inputs is called **generalization**.
- What separates machine learning from optimization is that we want the **generalization error**, also called the test error, to be low as well.

Learning VS. Optimization

Generalization VS. Overfitting

- Ability to perform well on previously unobserved inputs is called **generalization**.
- What separates machine learning from optimization is that we want the **generalization error**, also called the test error, to be low as well.
- We typically estimate the generalization error of a machine learning model by measuring its performance on a **test set** of examples that were collected separately from the training set.
- How can we affect performance on the test set when we get to observe only the training set? The field of **statistical learning theory** provides some answers.

Learning VS. Optimization

Generalization VS. Overfitting

- Ability to perform well on previously unobserved inputs is called **generalization**.
- What separates machine learning from optimization is that we want the **generalization error**, also called the test error, to be low as well.
- We typically estimate the generalization error of a machine learning model by measuring its performance on a **test set** of examples that were collected separately from the training set.
- How can we affect performance on the test set when we get to observe only the training set? The field of **statistical learning theory** provides some answers.
- If the training and the test set are collected arbitrarily, there is indeed little we can do.

Learning VS. Optimization

Generalization VS. Overfitting

- Ability to perform well on previously unobserved inputs is called **generalization**.
- What separates machine learning from optimization is that we want the **generalization error**, also called the test error, to be low as well.
- We typically estimate the generalization error of a machine learning model by measuring its performance on a **test set** of examples that were collected separately from the training set.
- How can we affect performance on the test set when we get to observe only the training set? The field of **statistical learning theory** provides some answers.
- If the training and the test set are collected arbitrarily, there is indeed little we can do.
- The train and test data are generated by a probability distribution over datasets called the data generating process. We typically make a set of assumptions known collectively as the **i.i.d.** assumptions.

Learning VS. Optimization

Generalization VS. Overfitting

- Ability to perform well on previously unobserved inputs is called **generalization**.
- What separates machine learning from optimization is that we want the **generalization error**, also called the test error, to be low as well.
- We typically estimate the generalization error of a machine learning model by measuring its performance on a **test set** of examples that were collected separately from the training set.
- How can we affect performance on the test set when we get to observe only the training set? The field of **statistical learning theory** provides some answers.
- If the training and the test set are collected arbitrarily, there is indeed little we can do.
- The train and test data are generated by a probability distribution over datasets called the data generating process. We typically make a set of assumptions known collectively as the **i.i.d.** assumptions.
- These assumptions are that the examples in each dataset are independent from each other, and that the train set and test set are identically distributed, drawn from the same probability distribution as each other.

(Deep Learning, Ian Goodfellow, MIT press, 2016)

Machine learning

underfitting and overfitting

- The factors determining how well a machine learning algorithm will perform are its ability to:
  - Make the training error small.
  - Make the gap between training and test error small.

Machine learning

underfitting and overfitting

- The factors determining how well a machine learning algorithm will perform are its ability to:
  - Make the training error small.
  - Make the gap between training and test error small.
- These two factors correspond to the two central challenges in machine learning: **underfitting** and **overfitting**. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large.

Machine learning

underfitting and overfitting

- The factors determining how well a machine learning algorithm will perform are its ability to:
  - Make the training error small.
  - Make the gap between training and test error small.
- These two factors correspond to the two central challenges in machine learning: **underfitting** and **overfitting**. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large.
- We can control whether a model is more likely to overfit or underfit by altering its **capacity**.

Machine learning

underfitting and overfitting

- The factors determining how well a machine learning algorithm will perform are its ability to:
  - Make the training error small.
  - Make the gap between training and test error small.
- These two factors correspond to the two central challenges in machine learning: **underfitting** and **overfitting**. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large.
- We can control whether a model is more likely to overfit or underfit by altering its **capacity**.
- Models with low capacity may struggle to fit the training set. Models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set.

Machine learning

underfitting and overfitting

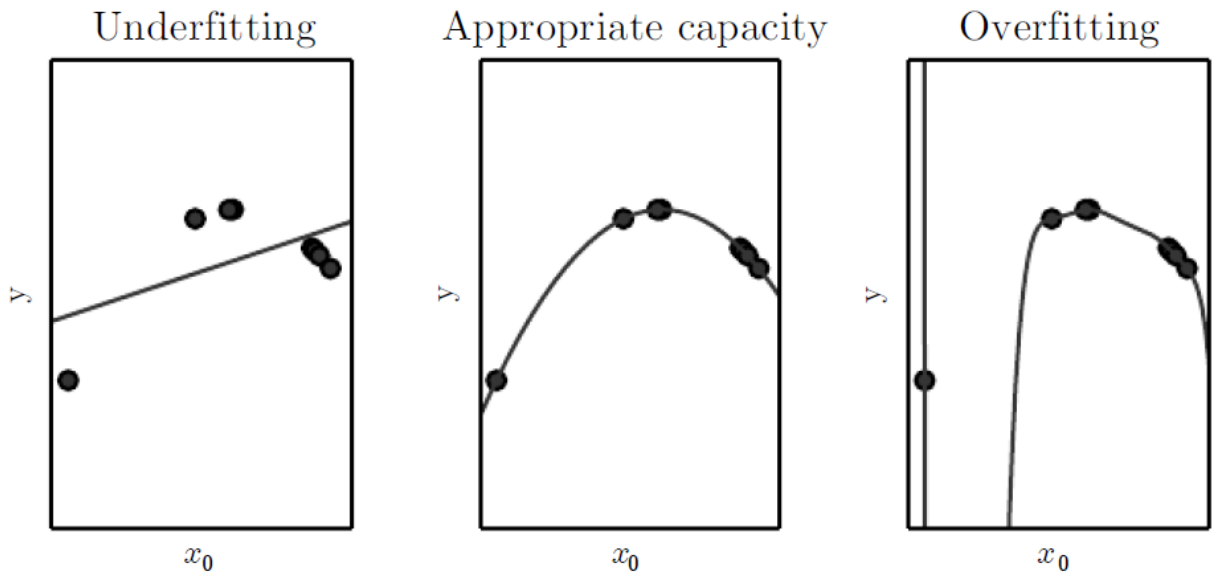
- The factors determining how well a machine learning algorithm will perform are its ability to:
  - Make the training error small.
  - Make the gap between training and test error small.
- These two factors correspond to the two central challenges in machine learning: **underfitting** and **overfitting**. Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large.
- We can control whether a model is more likely to overfit or underfit by altering its **capacity**.
- Models with low capacity may struggle to fit the training set. Models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set.
- One way to control the capacity of a learning algorithm is by choosing its **hypothesis space**, the set of functions that the learning algorithm is allowed to select as being the solution.

(Deep Learning, Ian Goodfellow, MIT press, 2016)

Machine learning

underfitting and overfitting

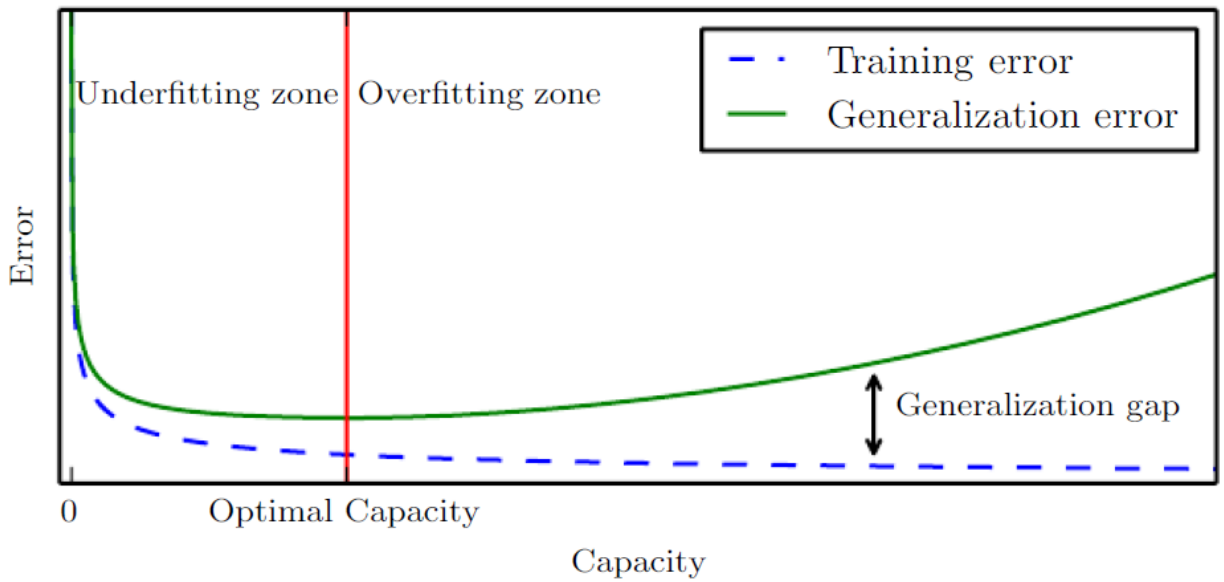
- Machine learning algorithms will generally perform best when their capacity is appropriate for the true complexity of the task they need to perform and the amount of training data they are provided with.



(Deep Learning, Ian Goodfellow, MIT press, 2016)

Capacity

Typically, generalization error has a U-shaped curve as a function of model capacity.

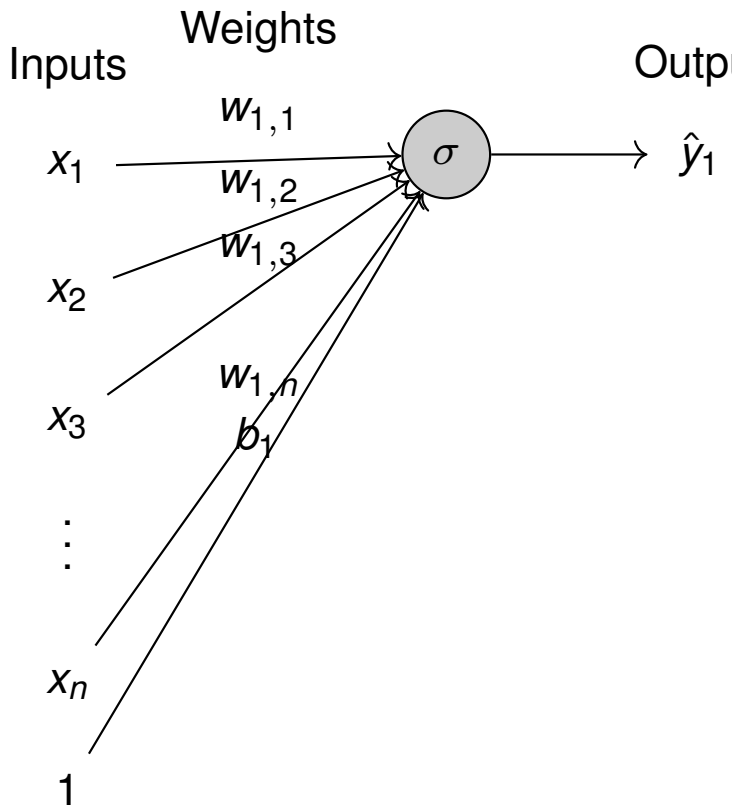


Regularization

- **Regularization** is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
  - We can give a learning algorithm a **preference** for one solution in its hypothesis space to another. This means that both functions are eligible, but one is preferred.

Softmax Classifier

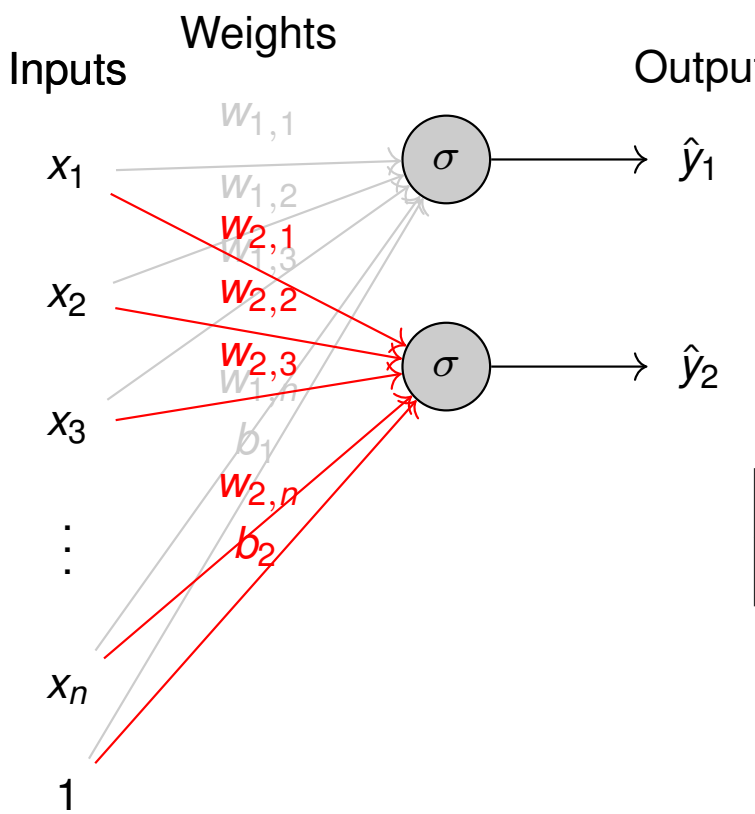
Multinomial logistic regression (softmax classifier)



$$z_1 = \sum_i w_{1,i} x_i + b_1 = \boldsymbol{w_1}^T \boldsymbol{x} + b_1, \quad \hat{y}_1 = \sigma(z_1)$$

$$z_1 = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \dots & w_{1,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + b_1$$

Multinomial logistic regression (softmax classifier)

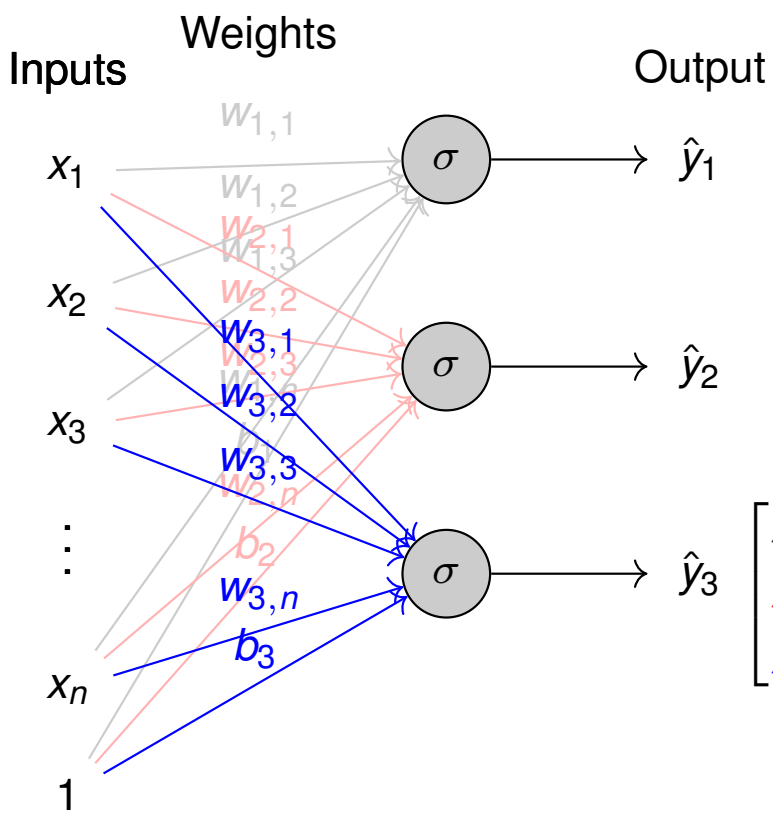


$$z_1 = \sum_i w_{1,i} x_i + b_1 = \boldsymbol{w_1}^T \boldsymbol{x} + b_1, \quad \hat{y}_1 = \sigma(z_1)$$

$$z_2 = \sum_i w_{2,i} x_i + b_2 = \boldsymbol{w_2}^T \boldsymbol{x} + b_2, \quad \hat{y}_2 = \sigma(z_2)$$

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \dots & w_{2,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Multinomial logistic regression (softmax classifier)



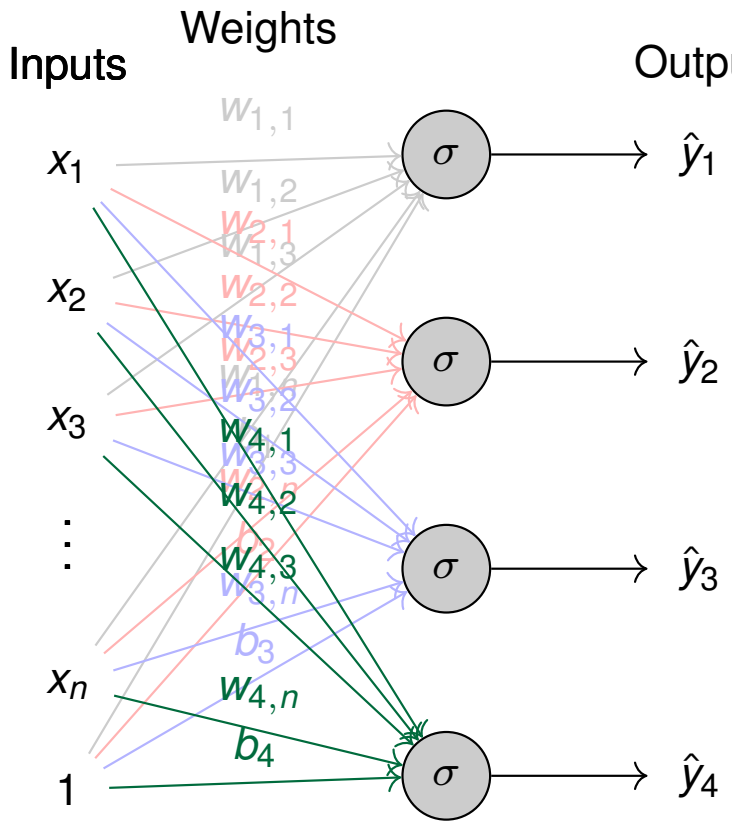
$$z_1 = \sum_i w_{1,i} x_i + b_1 = \boldsymbol{w_1}^T \boldsymbol{x} + b_1, \quad \hat{y}_1 = \sigma(z_1)$$

$$z_2 = \sum_i w_{2,i} x_i + b_2 = \boldsymbol{w_2}^T \boldsymbol{x} + b_2, \quad \hat{y}_2 = \sigma(z_2)$$

$$z_3 = \sum_i w_{3,i} x_i + b_3 = \boldsymbol{w_3}^T \boldsymbol{x} + b_3, \quad \hat{y}_3 = \sigma(z_3)$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \dots & w_{2,n} \\ w_{3,1} & w_{3,2} & w_{3,3} & \dots & w_{3,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Multinomial logistic regression (softmax classifier)



$$z_1 = \sum_i w_{1,i} x_i + b_1 = \boldsymbol{w_1}^T \boldsymbol{x} + b_1, \quad \hat{y}_1 = \sigma(z_1)$$

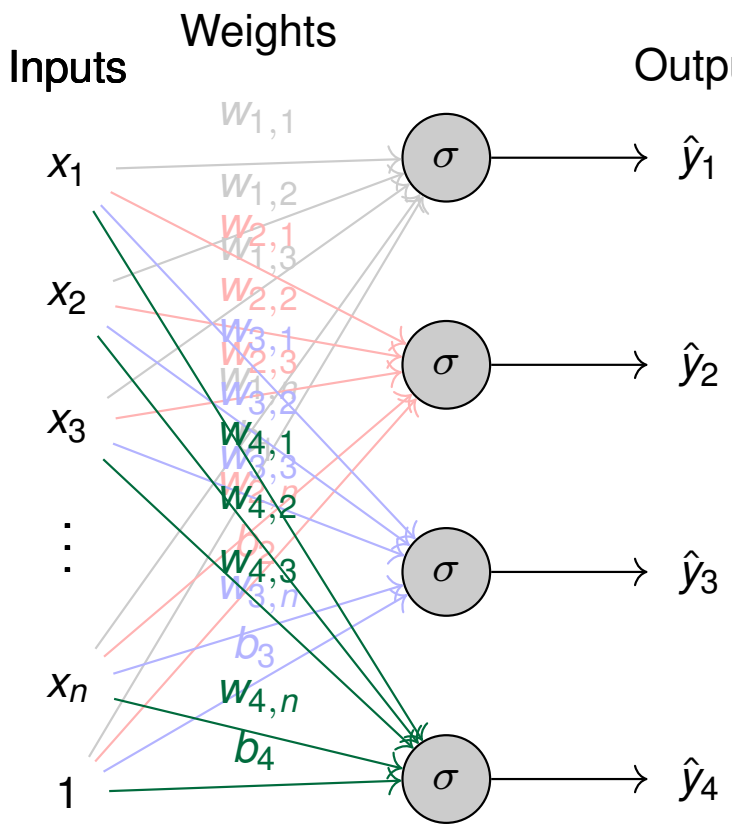
$$z_2 = \sum_i w_{2,i} x_i + b_2 = \boldsymbol{w_2}^T \boldsymbol{x} + b_2, \quad \hat{y}_2 = \sigma(z_2)$$

$$z_3 = \sum_i w_{3,i} x_i + b_3 = \boldsymbol{w_3}^T \boldsymbol{x} + b_3, \quad \hat{y}_3 = \sigma(z_3)$$

$$z_4 = \sum_i w_{4,i} x_i + b_4 = \boldsymbol{w_4}^T \boldsymbol{x} + b_4, \quad \hat{y}_4 = \sigma(z_4)$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \cdots & w_{2,n} \\ w_{3,1} & w_{3,2} & w_{3,3} & \cdots & w_{3,n} \\ w_{4,1} & w_{4,2} & w_{4,3} & \cdots & w_{4,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

Multinomial logistic regression (softmax classifier)



$$z_1 = \sum_i w_{1,i} x_i + b_1 = \boldsymbol{w_1}^T \boldsymbol{x} + b_1, \quad \hat{y}_1 = \sigma(z_1)$$

$$z_2 = \sum_i w_{2,i} x_i + b_2 = \boldsymbol{w_2}^T \boldsymbol{x} + b_2, \quad \hat{y}_2 = \sigma(z_2)$$

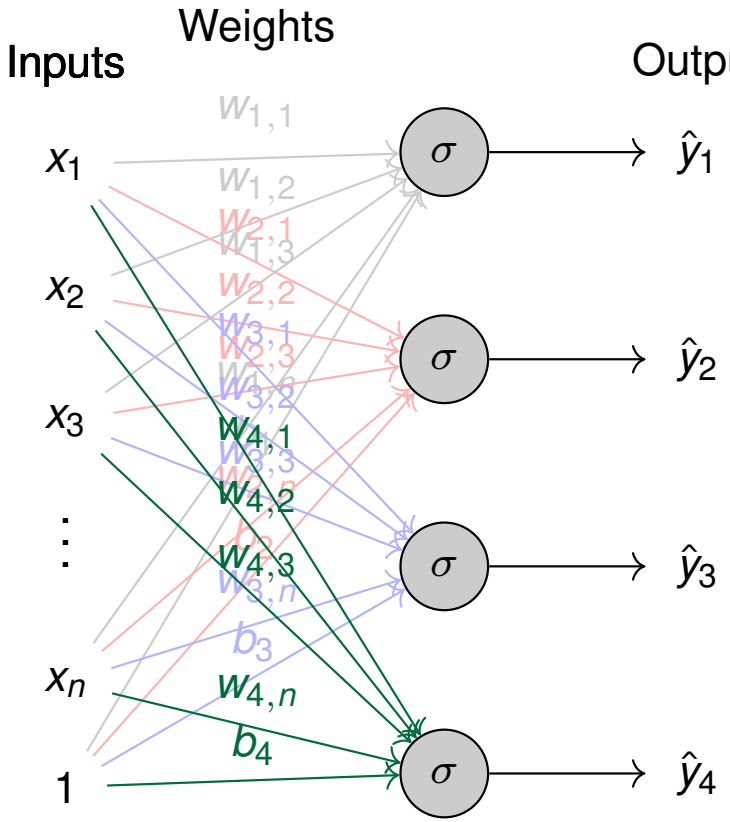
$$z_3 = \sum_i w_{3,i} x_i + b_3 = \boldsymbol{w_3}^T \boldsymbol{x} + b_3, \quad \hat{y}_3 = \sigma(z_3)$$

$$z_4 = \sum_i w_{4,i} x_i + b_4 = \boldsymbol{w_4}^T \boldsymbol{x} + b_4, \quad \hat{y}_4 = \sigma(z_4)$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \cdots & w_{2,n} \\ w_{3,1} & w_{3,2} & w_{3,3} & \cdots & w_{3,n} \\ w_{4,1} & w_{4,2} & w_{4,3} & \cdots & w_{4,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

$$\boldsymbol{z} = W^T \boldsymbol{x} + \boldsymbol{b}$$

Multinomial logistic regression (softmax classifier)



$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \cdots & w_{2,n} \\ w_{3,1} & w_{3,2} & w_{3,3} & \cdots & w_{3,n} \\ w_{4,1} & w_{4,2} & w_{4,3} & \cdots & w_{4,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

The output vector: $\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \hat{y}_4 \end{bmatrix} = \sigma\left(\begin{bmatrix} \hat{z}_1 \\ \hat{z}_2 \\ \hat{z}_3 \\ \hat{z}_4 \end{bmatrix} \right)$

- The output vector is not a probability vector (Nonzero and sum of elements must be one)
  - $\hat{y}_i \in [0, 1]$
  - Cross entropy is fed by a probability vector

Multi-class output: One-hot representations

- The labels are 0 and 1 in logistic regression
- Consider a network that must distinguish if an input is a cat, a dog, camel, a hat, or a flower
- For inputs of each of the four classes the desired output is a probability vector:

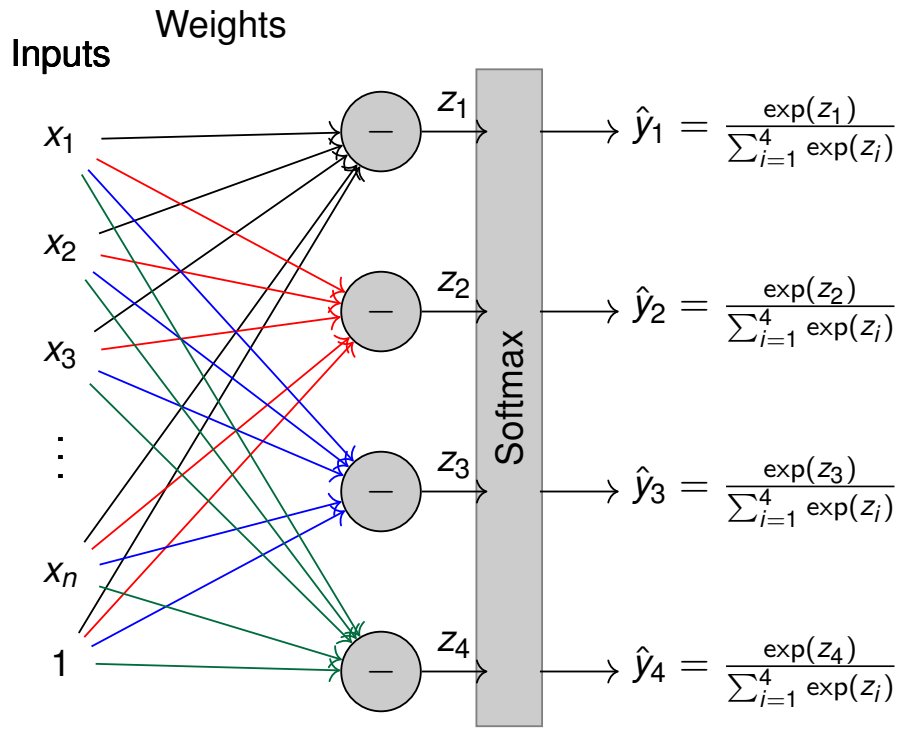
  Cat: $\begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}^T$

  Dog: $\begin{bmatrix}0 & 1 & 0 & 0\end{bmatrix}^T$

  Fish: $\begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix}^T$

  Frog: $\begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}^T$
- For input of any class, we will have a four-dimensional vector output with four zeros and a single 1 at the position of the class
- This is a one hot vector

Softmax classifier



Softmax function:

$$P(\hat{y} = j|\boldsymbol{x}, W) = \frac{\exp(z_j)}{\sum_{i=1}^{K}\exp(z_i)} = \hat{y}_j$$

The output vector: $\hat{\boldsymbol{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \hat{y}_4 \end{bmatrix}$

The output of classifier: $\underset{j}{\text{argmax}}\ \hat{y}_j$

Loss function

- Cross entropy:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{K} y_i \log \hat{y}_i$$

- $\boldsymbol{y}$ is a one-hot vector
- If the true class of $\boldsymbol{x}$ is j, only the the j^{th} element of $\boldsymbol{y}$ is one, and the others are zero. Therefore:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_j$$

Loss function

- Cross entropy:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{K} y_i \log \hat{y}_i$$

- $\boldsymbol{y}$ is a one-hot vector
- If the true class of $\boldsymbol{x}$ is *j*, only the the j^{th} element of $\boldsymbol{y}$ is one, and the others are zero. Therefore:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_j$$

example:

$$\boldsymbol{y} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} \textit{dog} \\ \textit{cat} \\ \textit{fish} \\ \textit{frog} \end{matrix} \quad \boldsymbol{z} = \begin{pmatrix} 3.2 \\ 5.1 \\ -1.7 \\ -3.7 \end{pmatrix} \quad softmax(\boldsymbol{z}) = \hat{\boldsymbol{y}} = \begin{pmatrix} 0.13 \\ 0.87 \\ 0 \\ 0 \end{pmatrix}$$

Loss function

- Cross entropy:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{K} y_i \log \hat{y}_i$$

- $\boldsymbol{y}$ is a one-hot vector
- If the true class of $\boldsymbol{x}$ is *j*, only the the j^{th} element of $\boldsymbol{y}$ is one, and the others are zero. Therefore:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_j$$

example:

$$\boldsymbol{y} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} \textit{dog} \\ \textit{cat} \\ \textit{fish} \\ \textit{frog} \end{matrix} \quad \boldsymbol{z} = \begin{pmatrix} 3.2 \\ 5.1 \\ -1.7 \\ -3.7 \end{pmatrix} \quad softmax(\boldsymbol{z}) = \hat{\boldsymbol{y}} = \begin{pmatrix} 0.13 \\ 0.87 \\ 0 \\ 0 \end{pmatrix}$$

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_2 = -\log 0.87 = 0.14$$

Loss function

- Cross entropy:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{K} y_i \log \hat{y}_i$$

- $\boldsymbol{y}$ is a one-hot vector
- If the true class of $\boldsymbol{x}$ is *j*, only the the j^{th} element of $\boldsymbol{y}$ is one, and the others are zero. Therefore:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_j$$

example:

$$\boldsymbol{y} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} \textit{dog} \\ \textit{cat} \\ \textit{fish} \\ \textit{frog} \end{matrix} \quad \boldsymbol{z} = \begin{pmatrix} 5.8 \\ 2.2 \\ -1.6 \\ 1.5 \end{pmatrix} \quad softmax(\boldsymbol{z}) = \hat{\boldsymbol{y}} = \begin{pmatrix} 0.96 \\ 0.03 \\ 0 \\ 0.01 \end{pmatrix}$$

Loss function

- Cross entropy:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{K} y_i \log \hat{y}_i$$

- $\boldsymbol{y}$ is a one-hot vector
- If the true class of $\boldsymbol{x}$ is *j*, only the the j^{th} element of $\boldsymbol{y}$ is one, and the others are zero. Therefore:

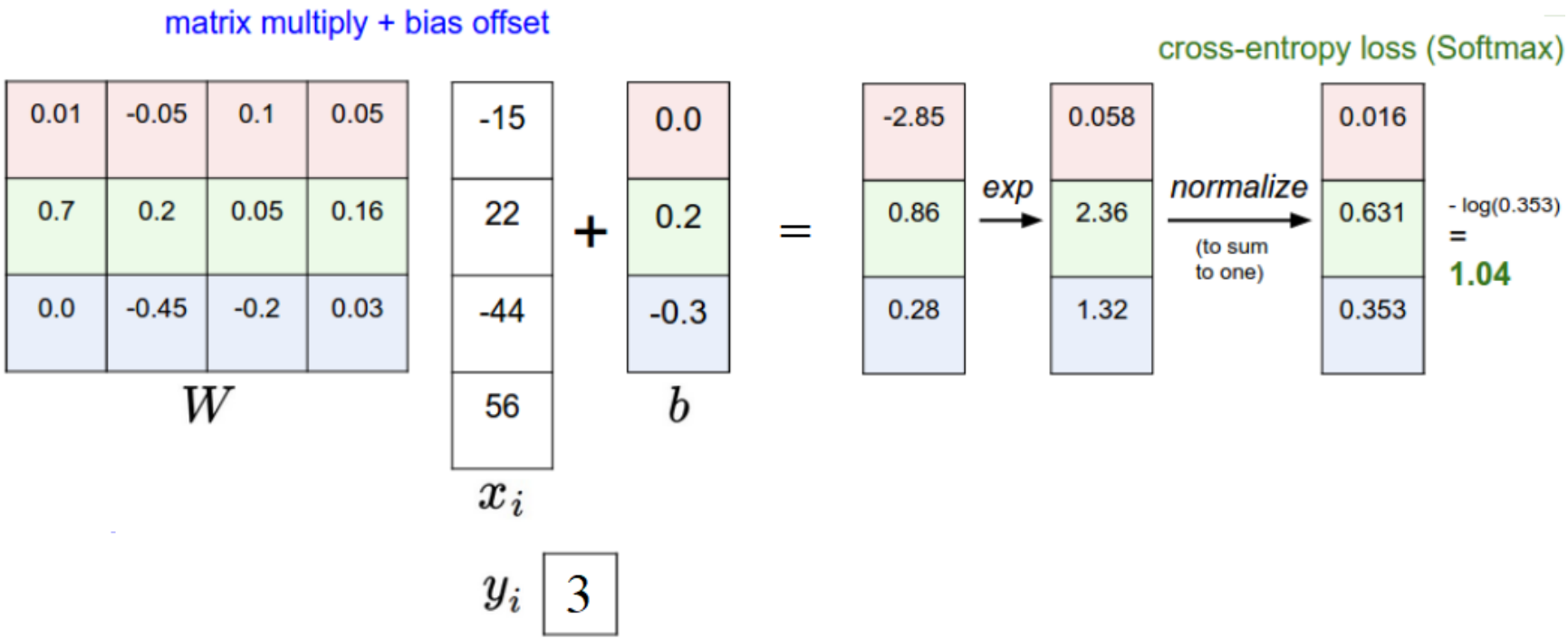
$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_j$$

example:

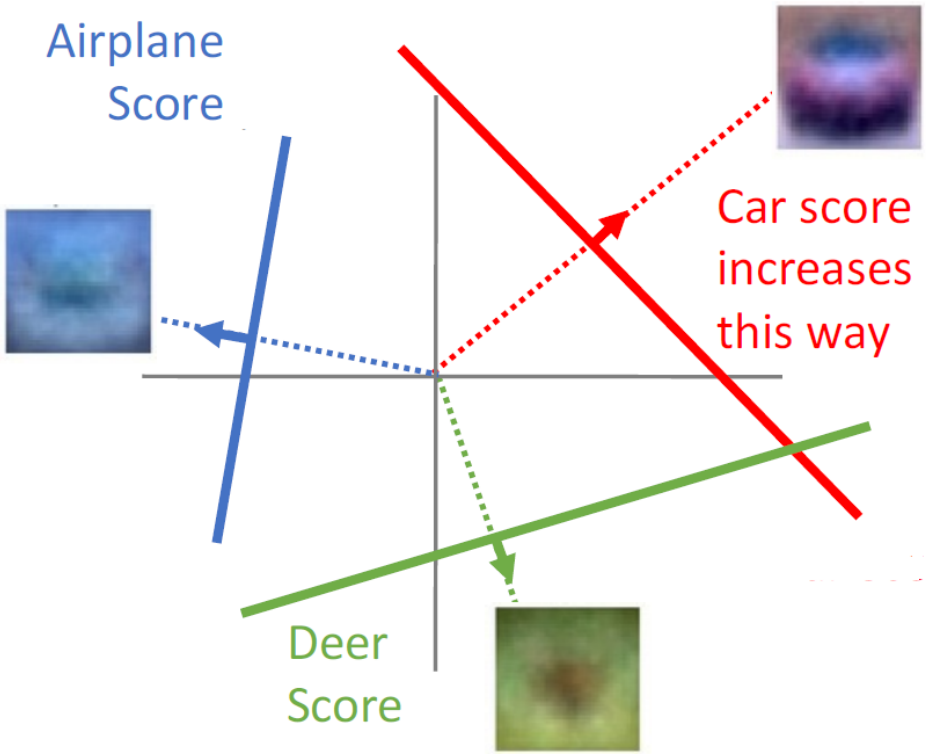
$$\boldsymbol{y} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} \textit{dog} \\ \textit{cat} \\ \textit{fish} \\ \textit{frog} \end{matrix} \quad \boldsymbol{z} = \begin{pmatrix} 5.8 \\ 2.2 \\ -1.6 \\ 1.5 \end{pmatrix} \quad softmax(\boldsymbol{z}) = \hat{\boldsymbol{y}} = \begin{pmatrix} 0.96 \\ 0.03 \\ 0 \\ 0.01 \end{pmatrix}$$

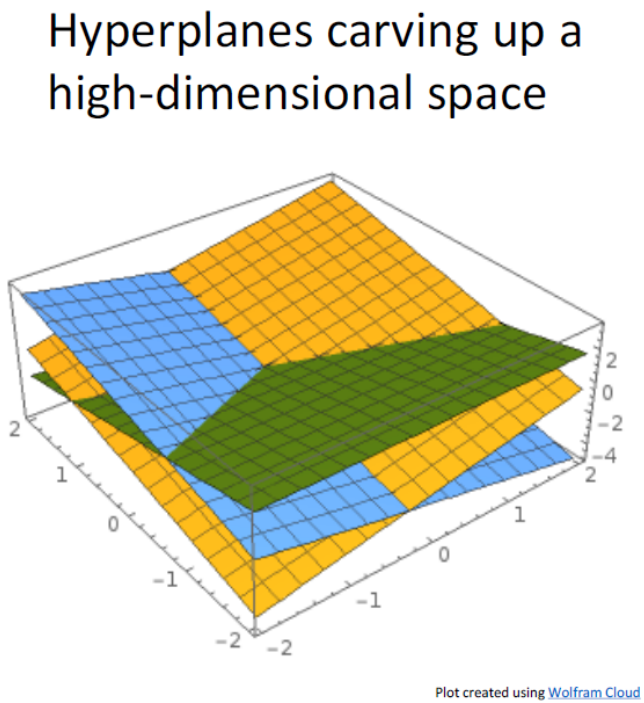
$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_2 = -\log 0.03 = 3.5$$

Example


matrix multiply + bias offset
cross-entropy loss (Softmax)
0.01 -0.05 0.1 0.05
0.7 0.2 0.05 0.16
0.0 -0.45 -0.2 0.03
W
-15
22
-44
56
x_i
+
0.0
0.2
-0.3
b
=
-2.85
0.86
0.28
exp
0.058
2.36
1.32
normalize
(to sum to one)
0.016
0.631
0.353
- log(0.353)
=
1.04
y_i
3

Softmax classifier



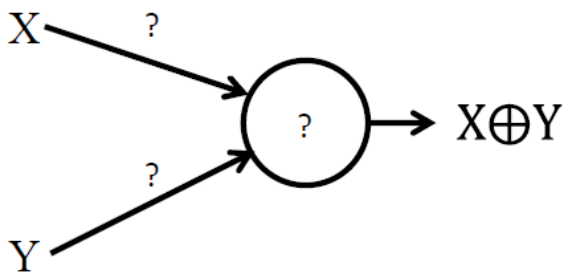


http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/

Deep Neural Networks

Xor

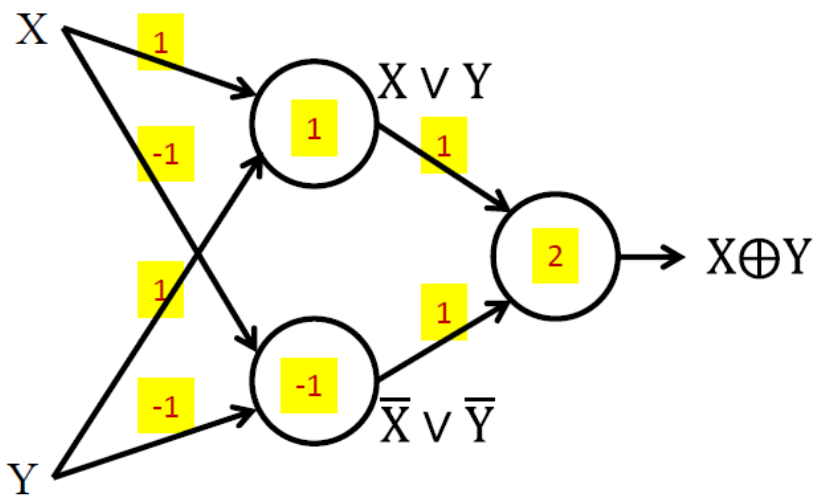
No solution for XOR!
Not universal!



Values shown on edges are weights, numbers in the circles are thresholds

Solution: add layers

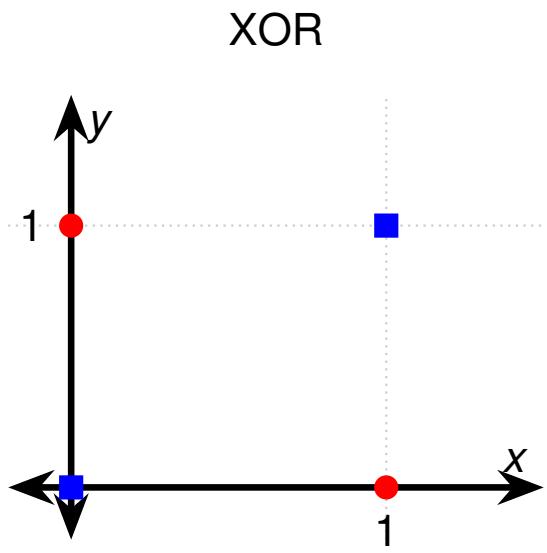
XOR

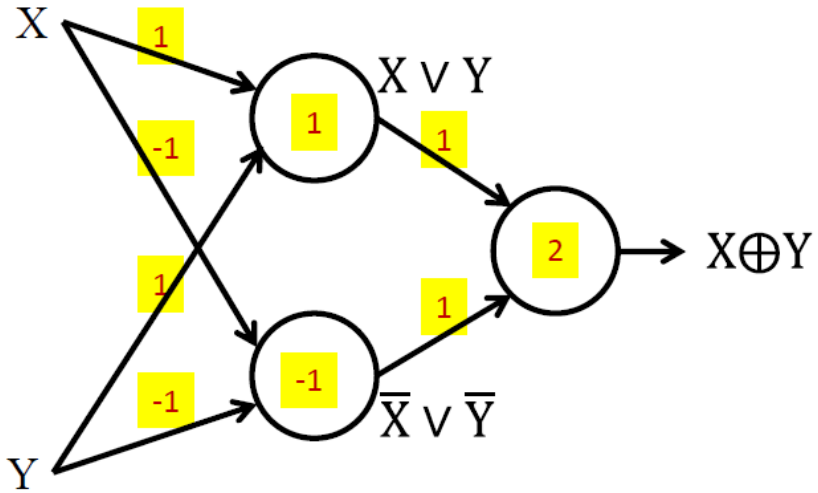


X	Y	A	B	Label
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

Transformation

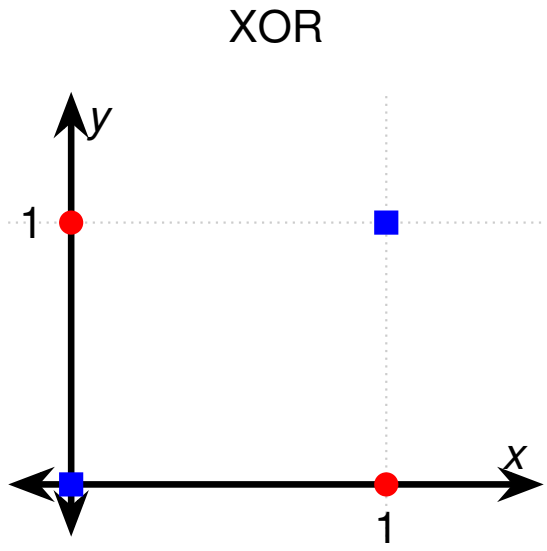
X	Y	A	B	Label
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

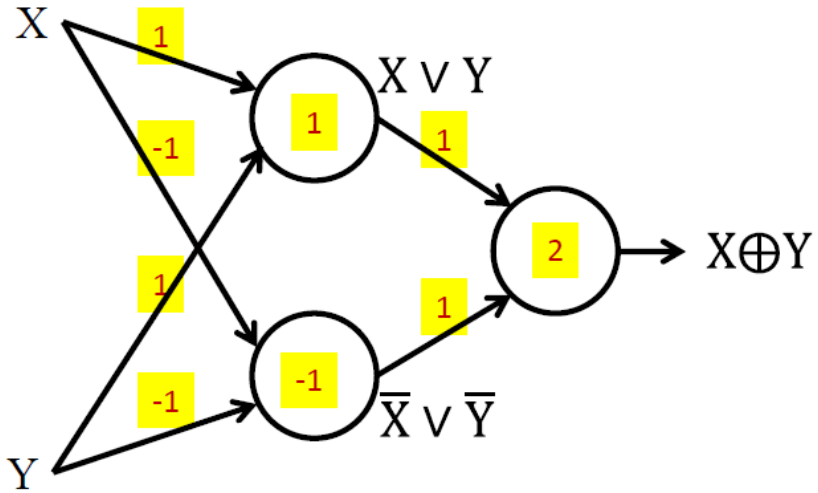


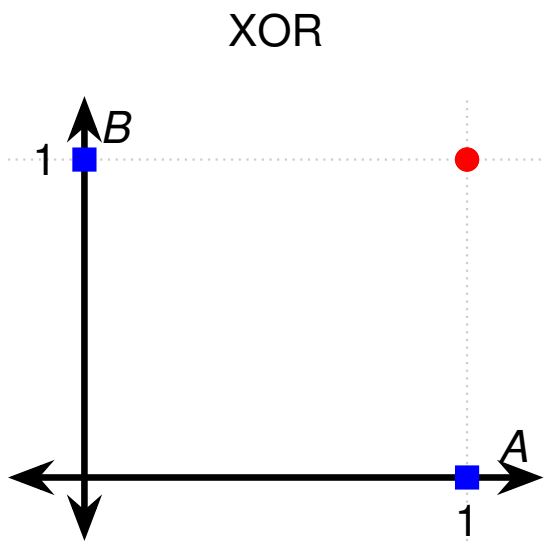


Transformation

X	Y	A	B	Label
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

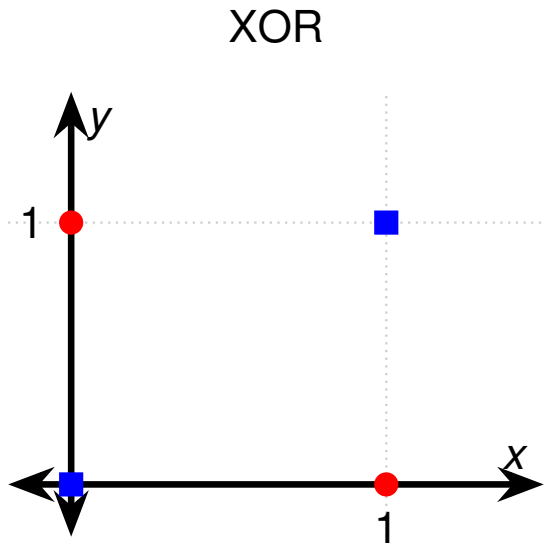


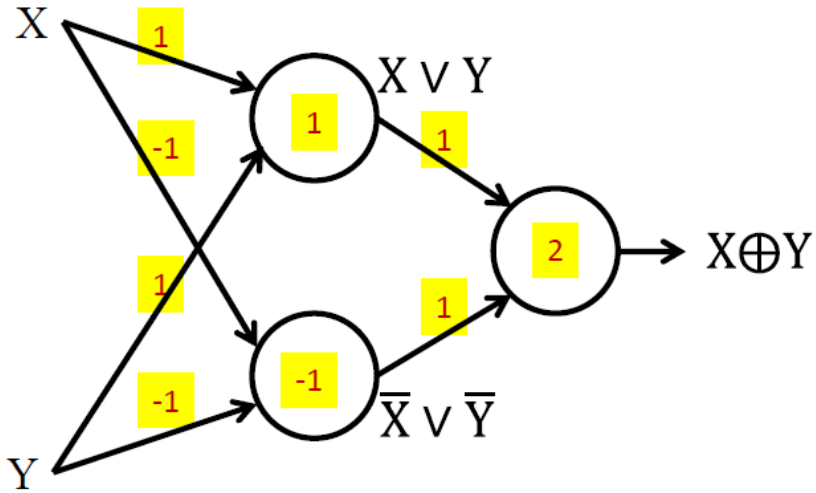


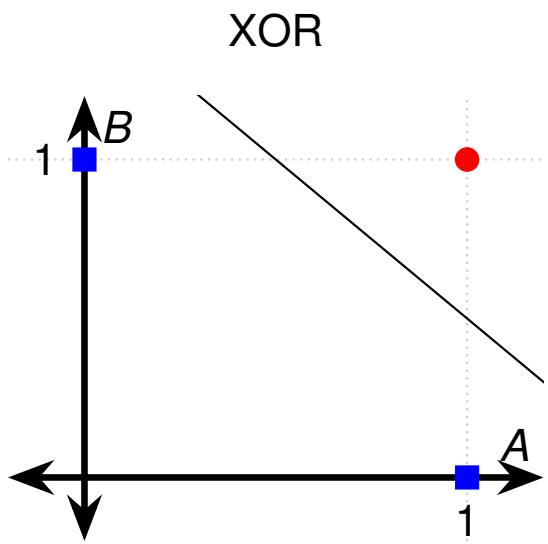


Transformation

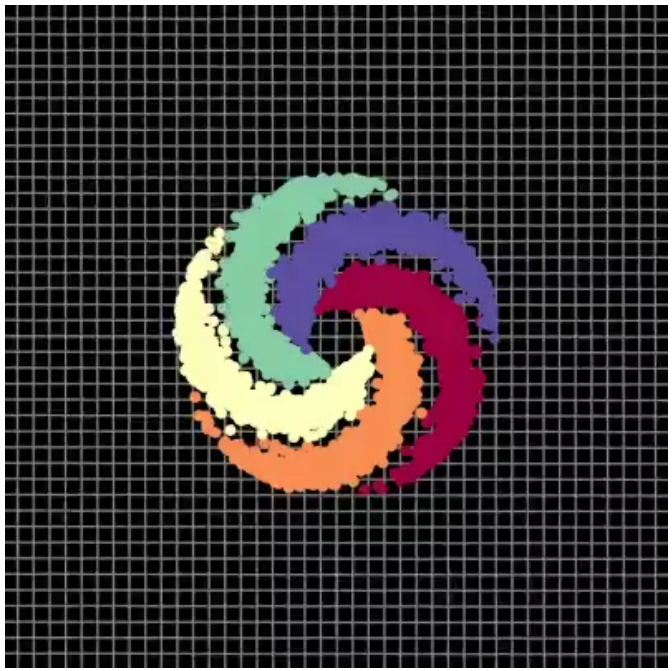
X	Y	A	B	Label
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0





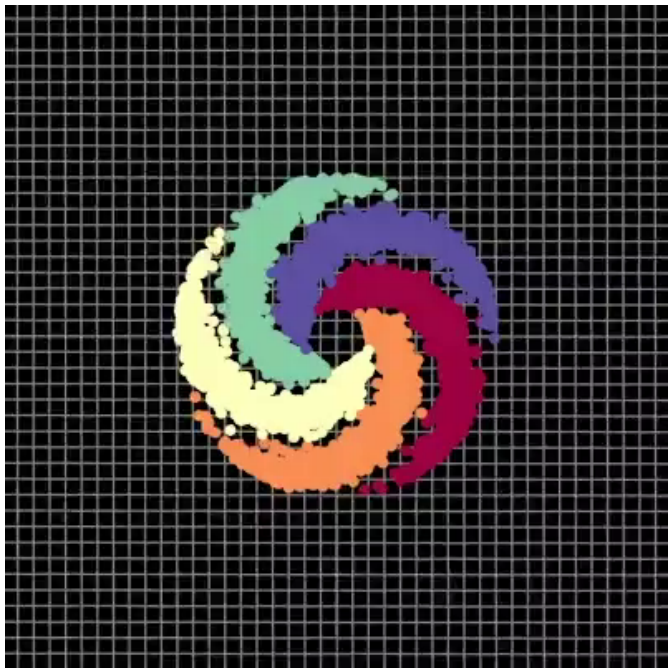


Needs more layers

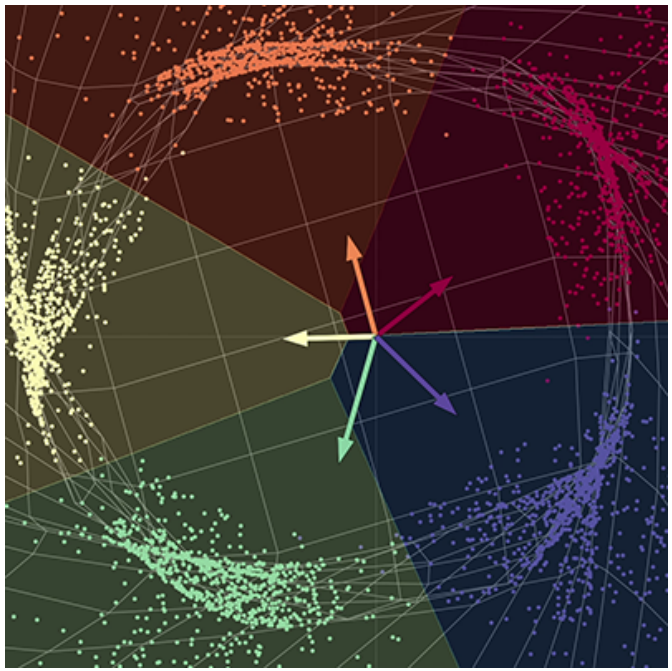


Input points, pre-network

Needs more layers



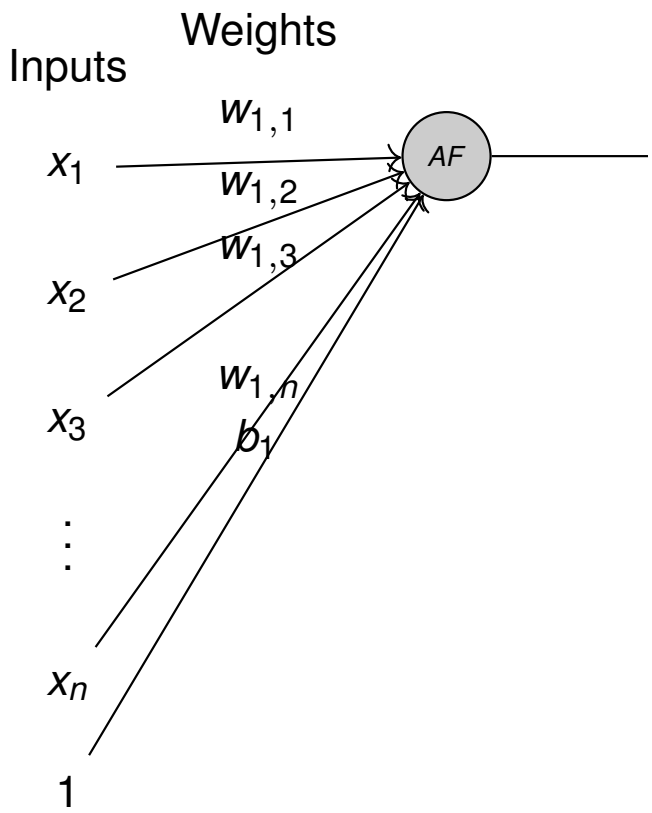
Input points, pre-network



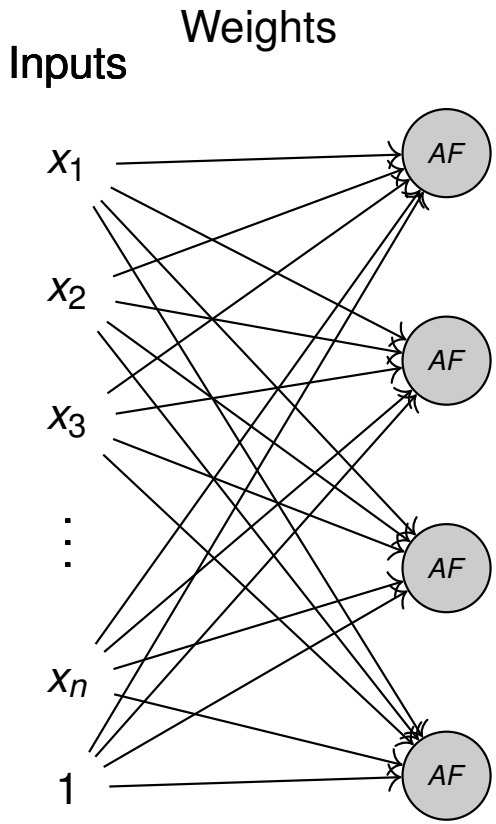
Output points, post-network

(Canziani, 2020)

Fully connected neural networks

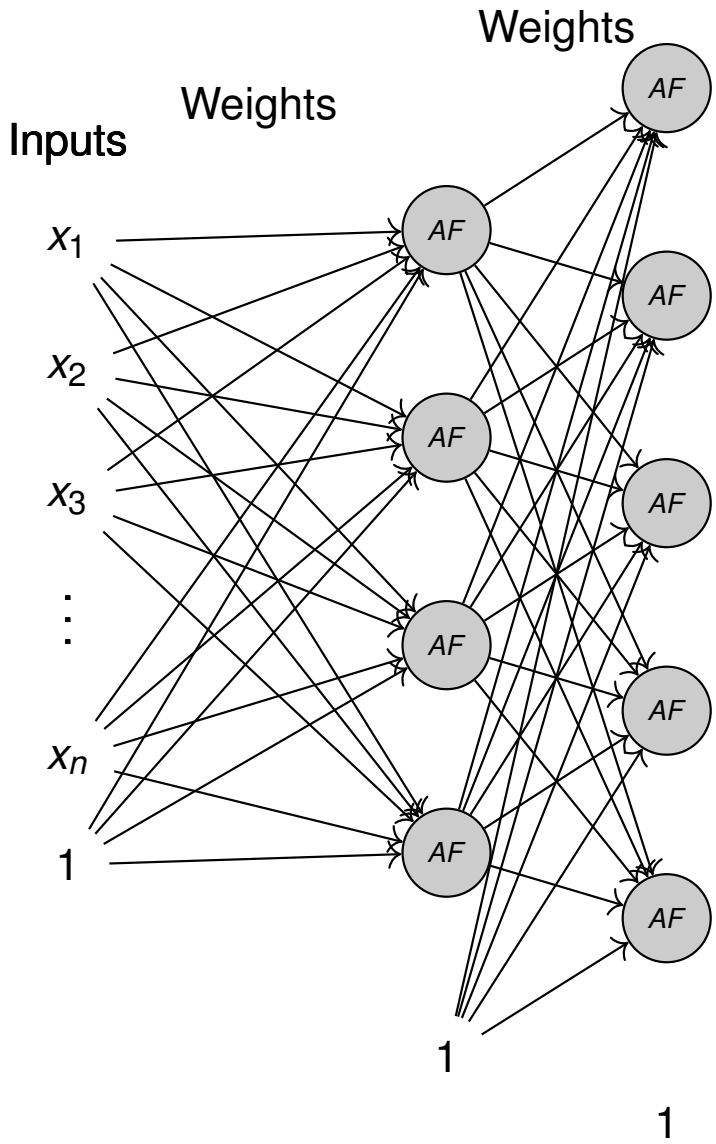


Fully connected neural networks

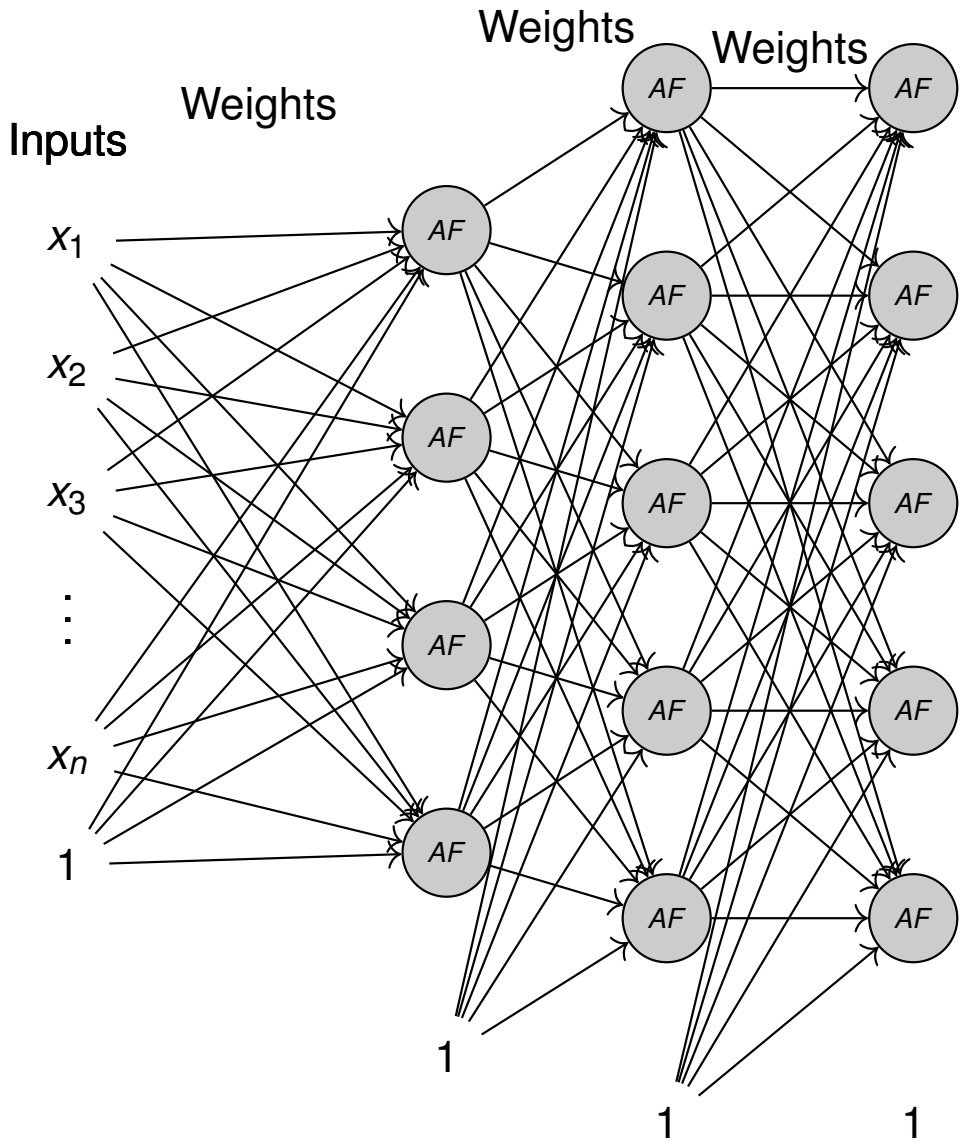


1

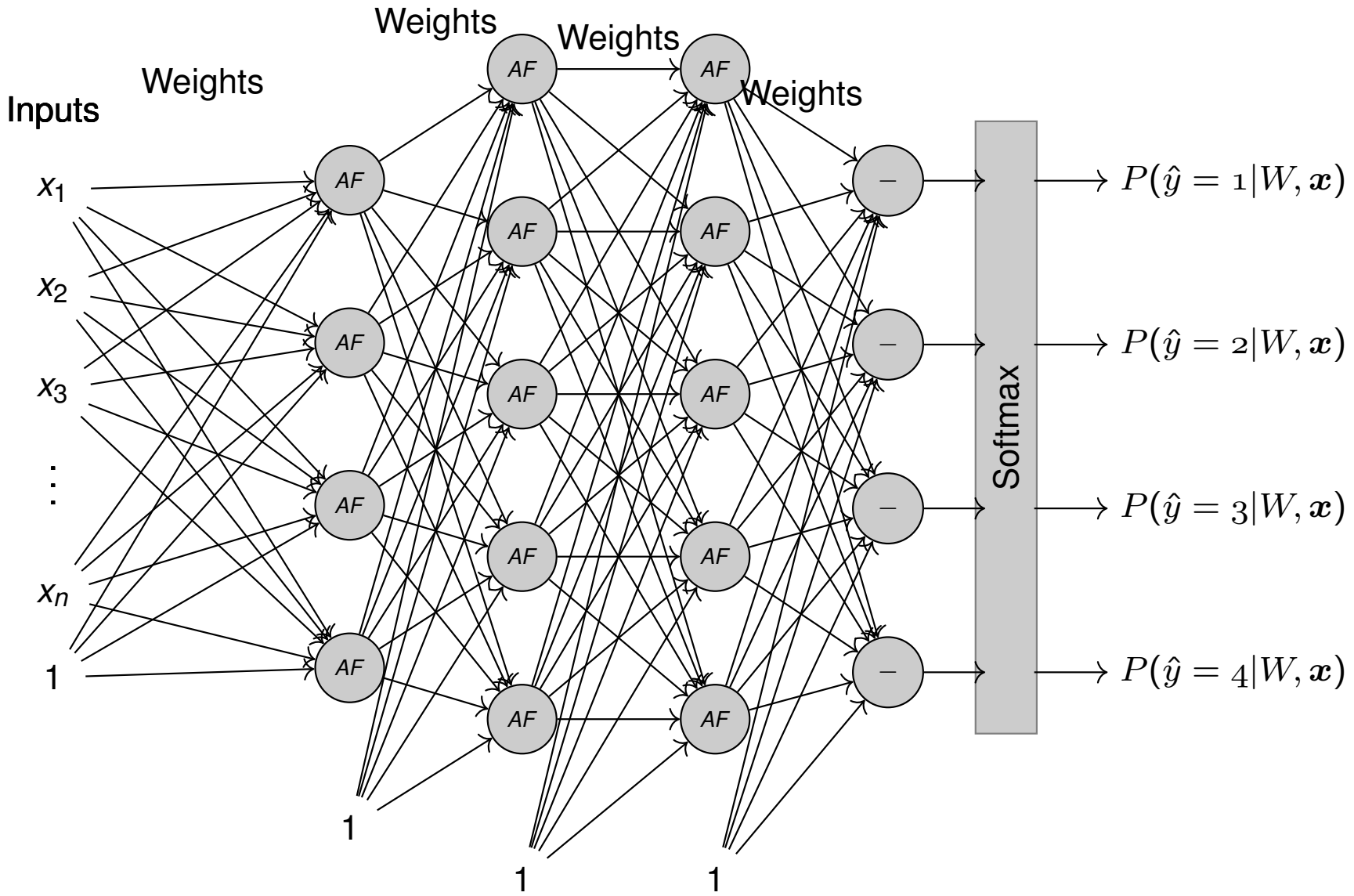
Fully connected neural networks



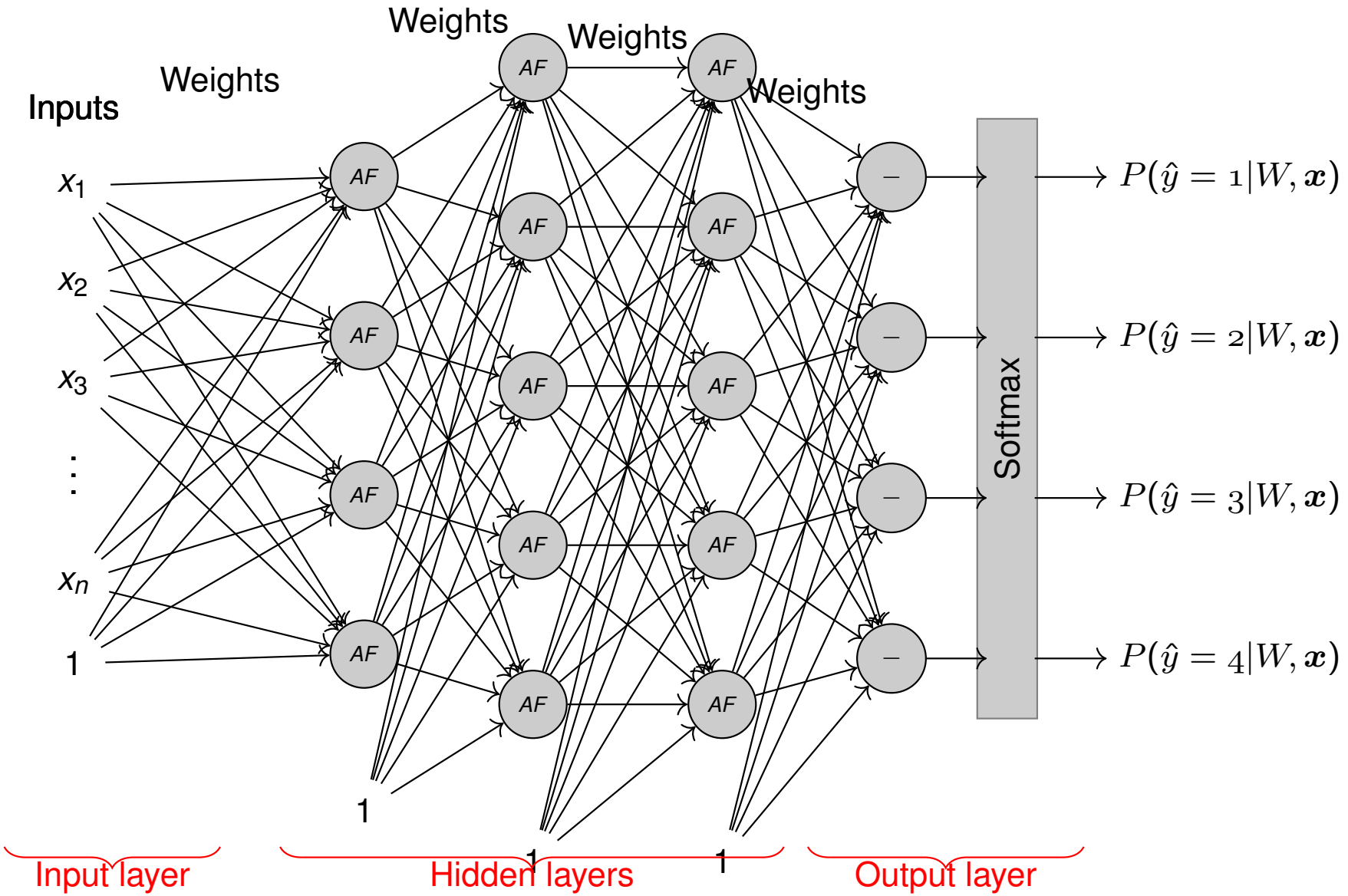
Fully connected neural networks



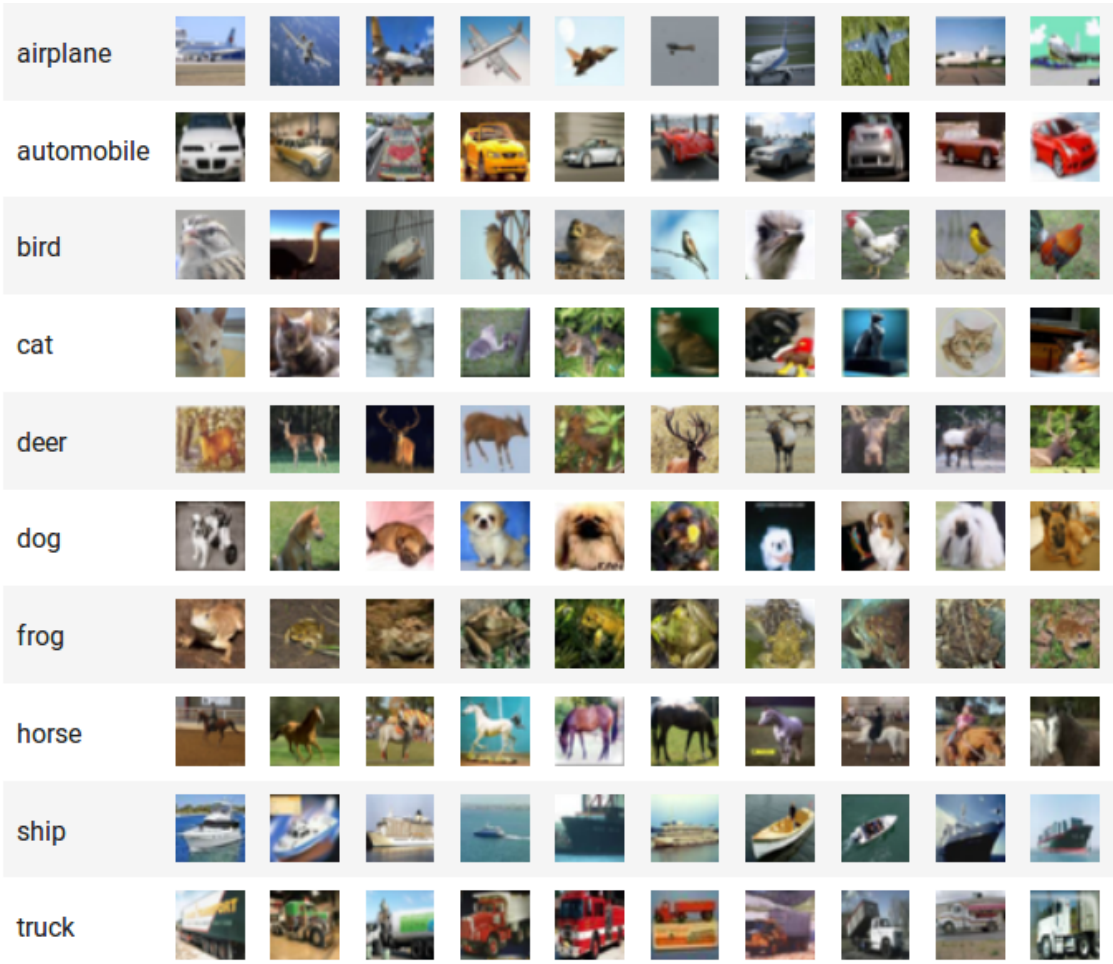
Fully connected neural networks



Fully connected neural networks

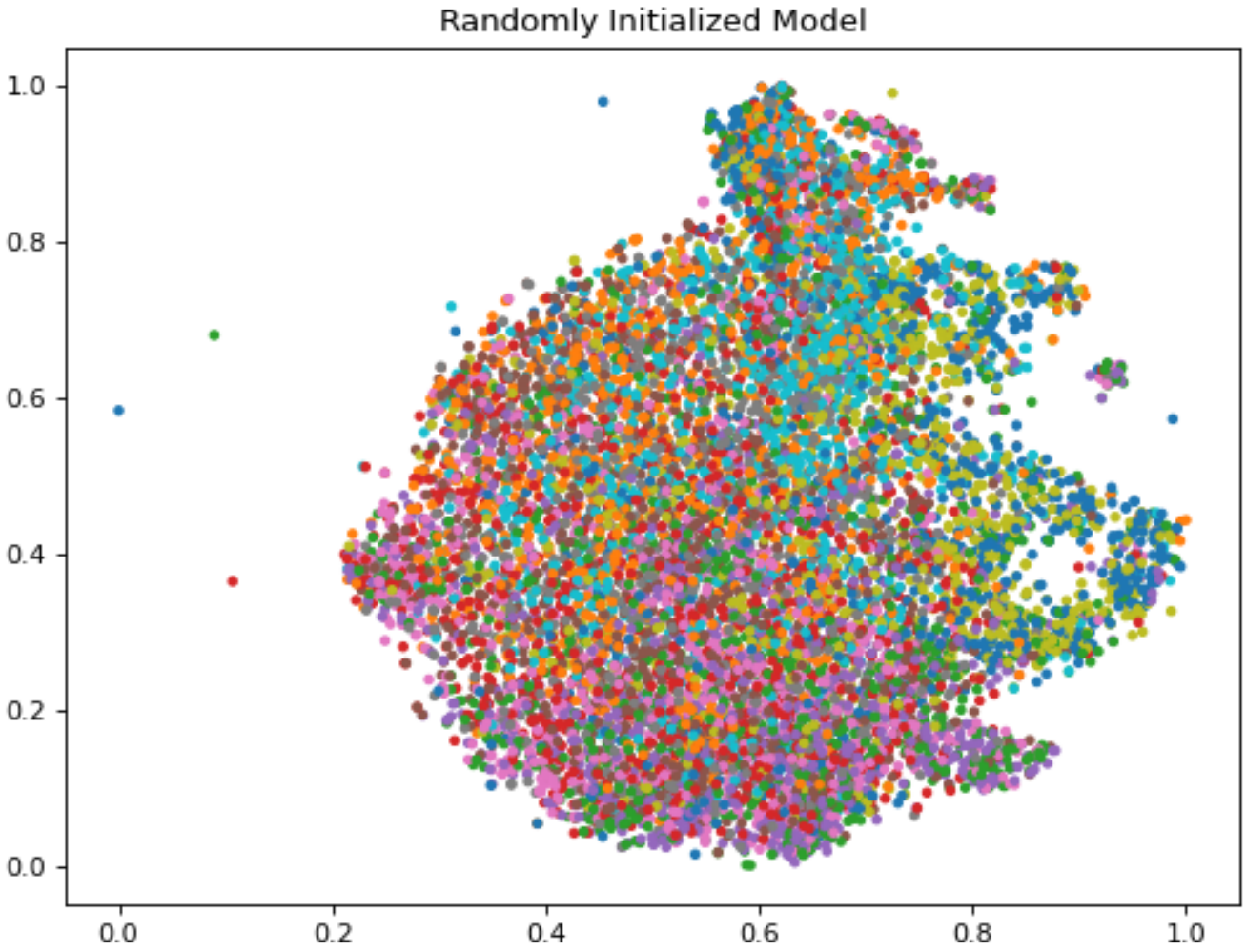


CIFAR10

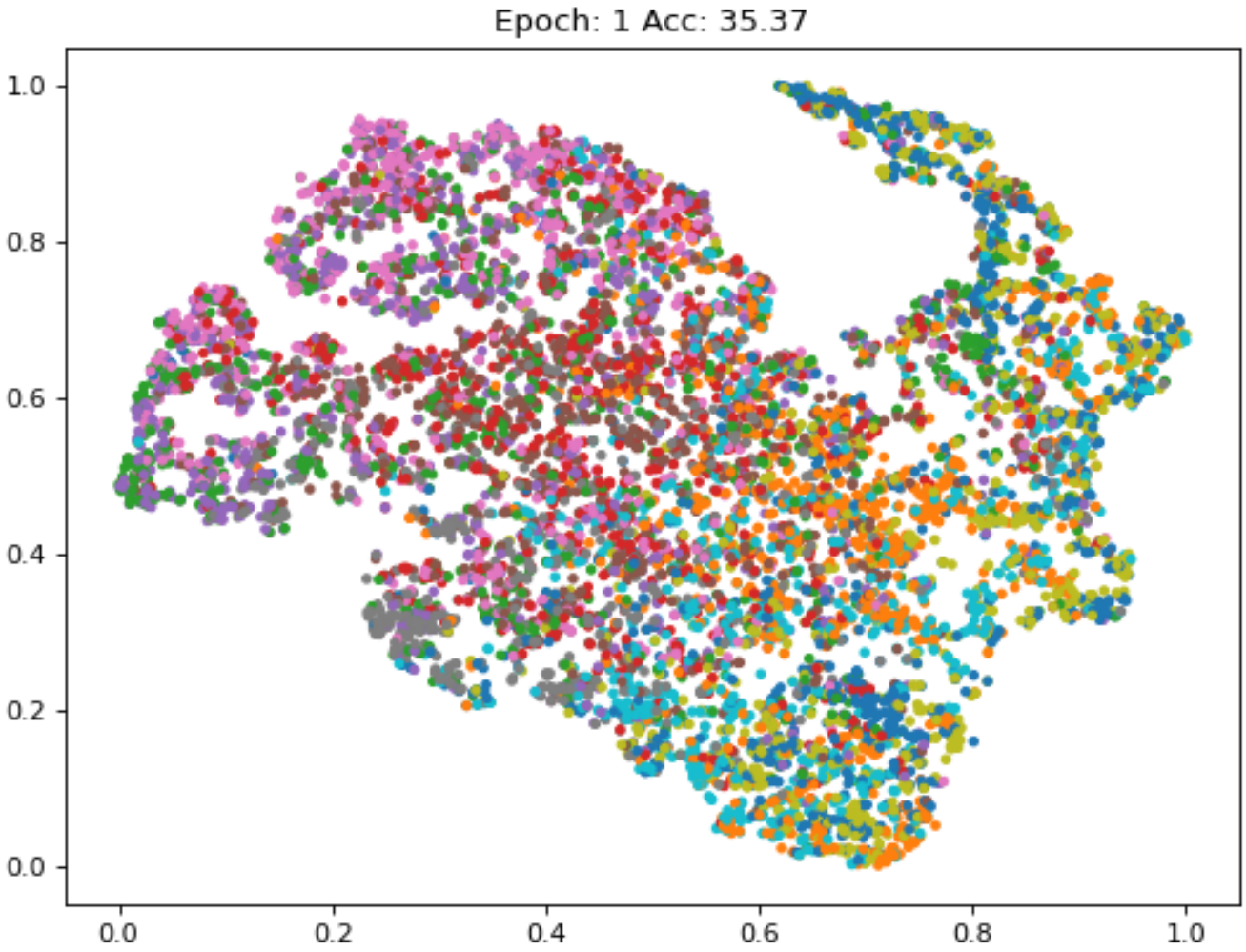


Example images from the CIFAR-10 dataset.

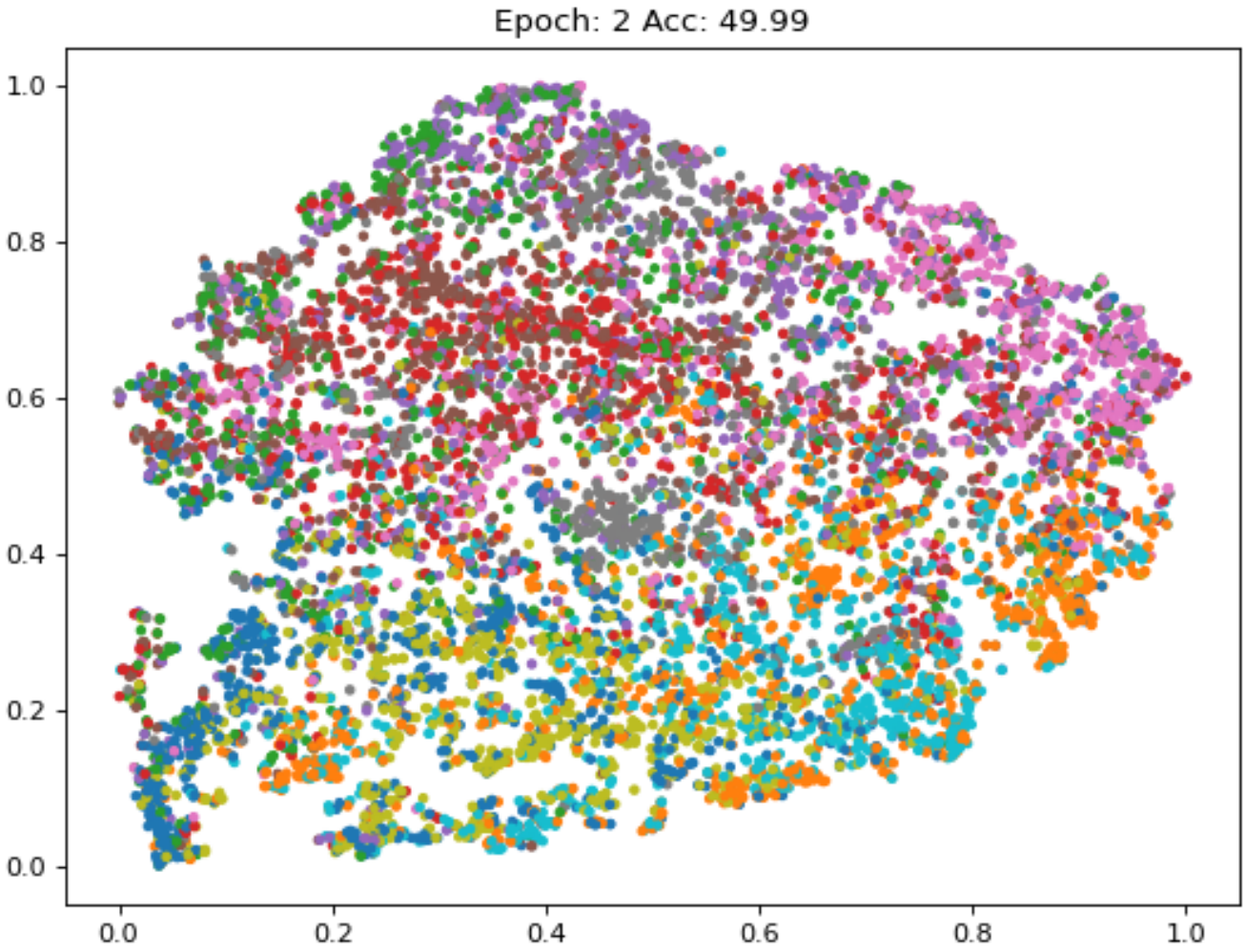
Visualization of the last hidden layer output for CIFAR10 samples



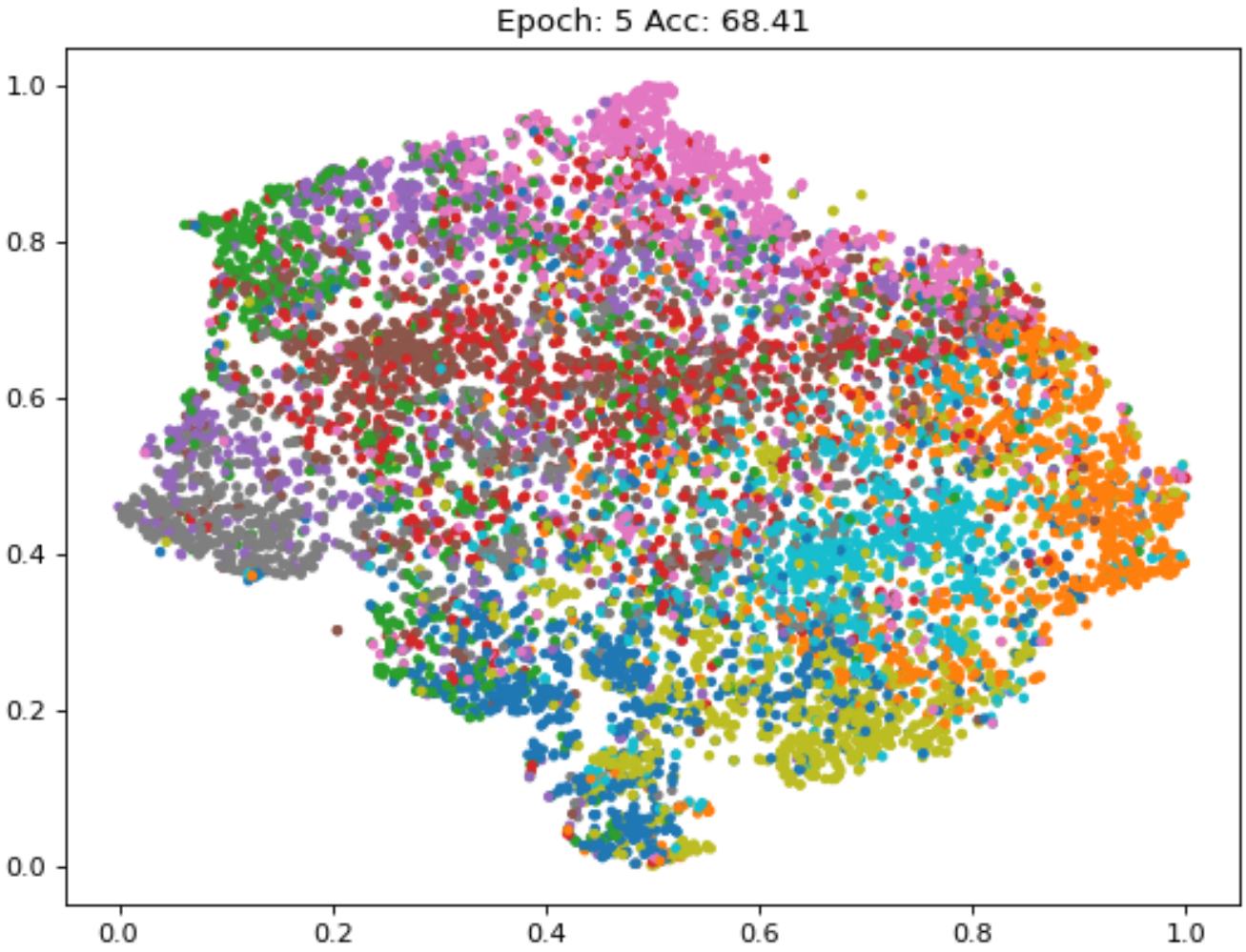
Visualization of the last hidden layer output for CIFAR10 samples



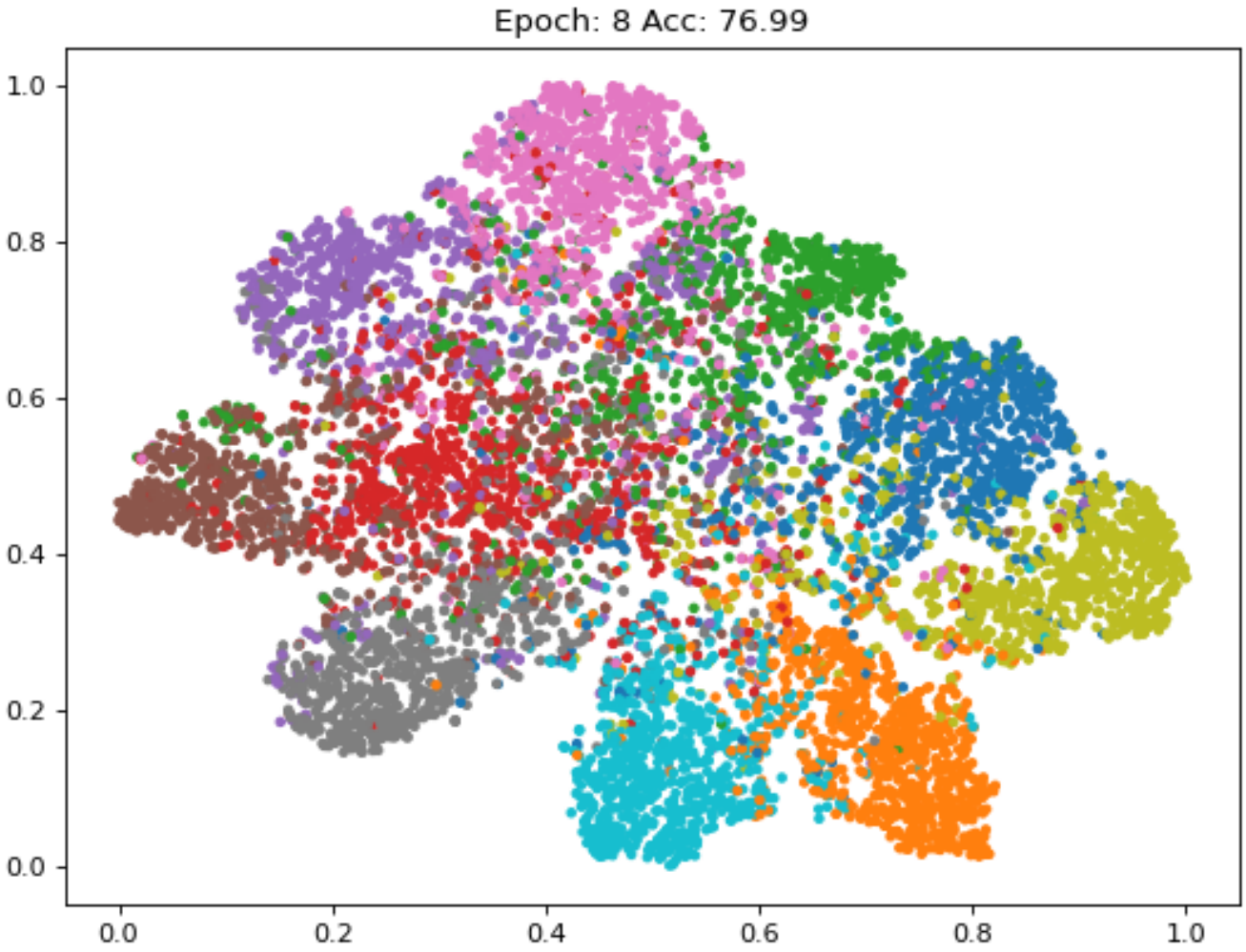
Visualization of the last hidden layer output for CIFAR10 samples



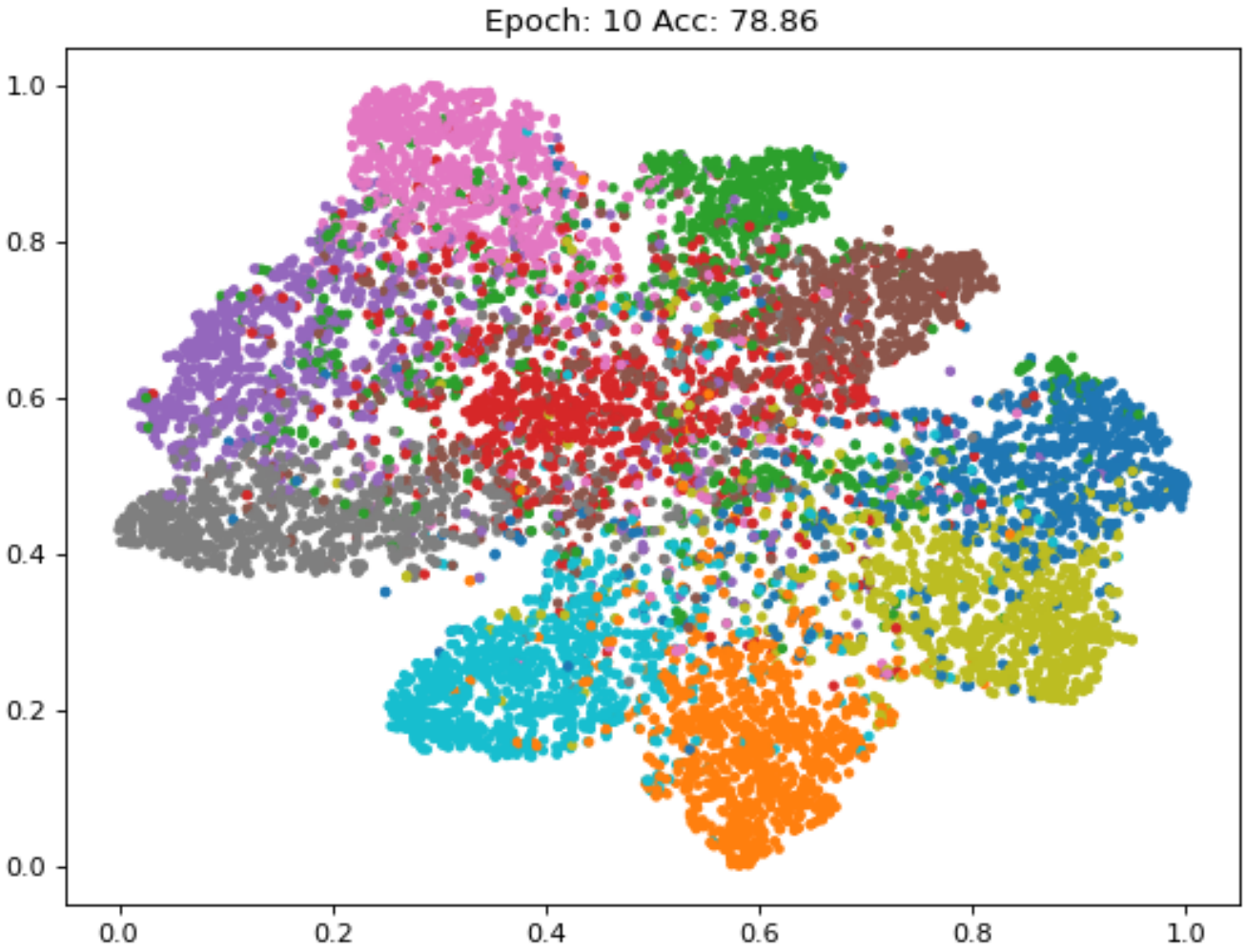
Visualization of the last hidden layer output for CIFAR10 samples



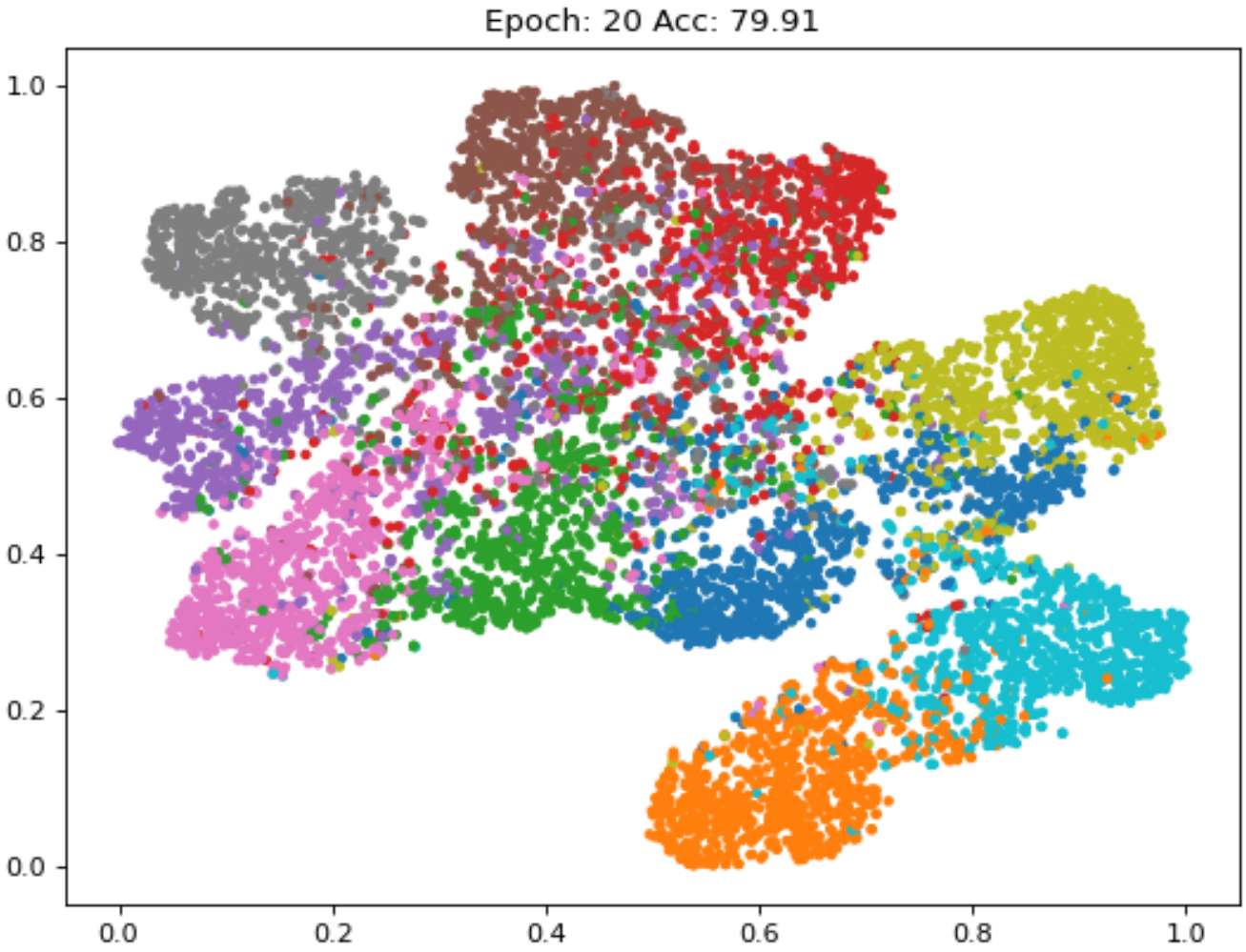
Visualization of the last hidden layer output for CIFAR10 samples



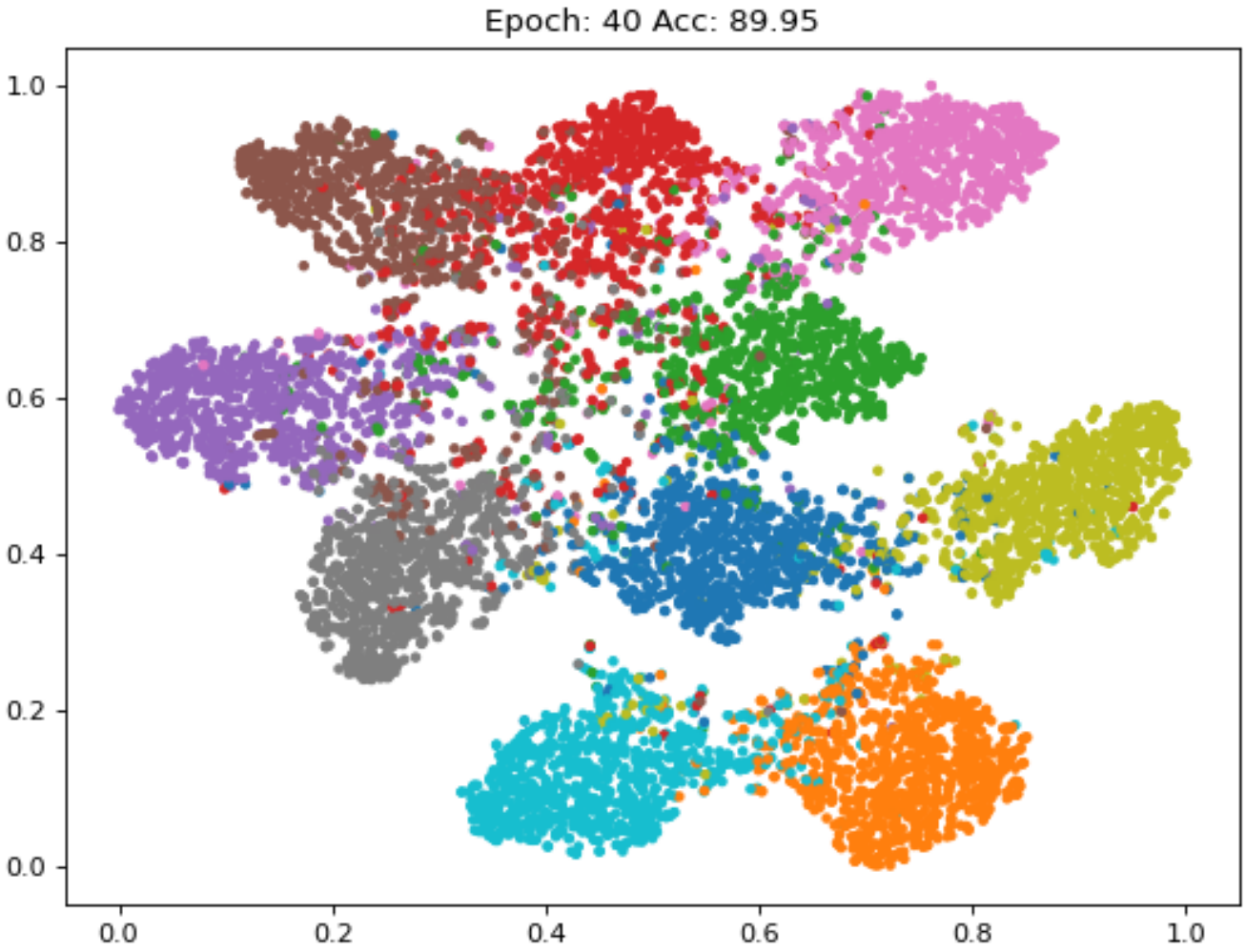
Visualization of the last hidden layer output for CIFAR10 samples



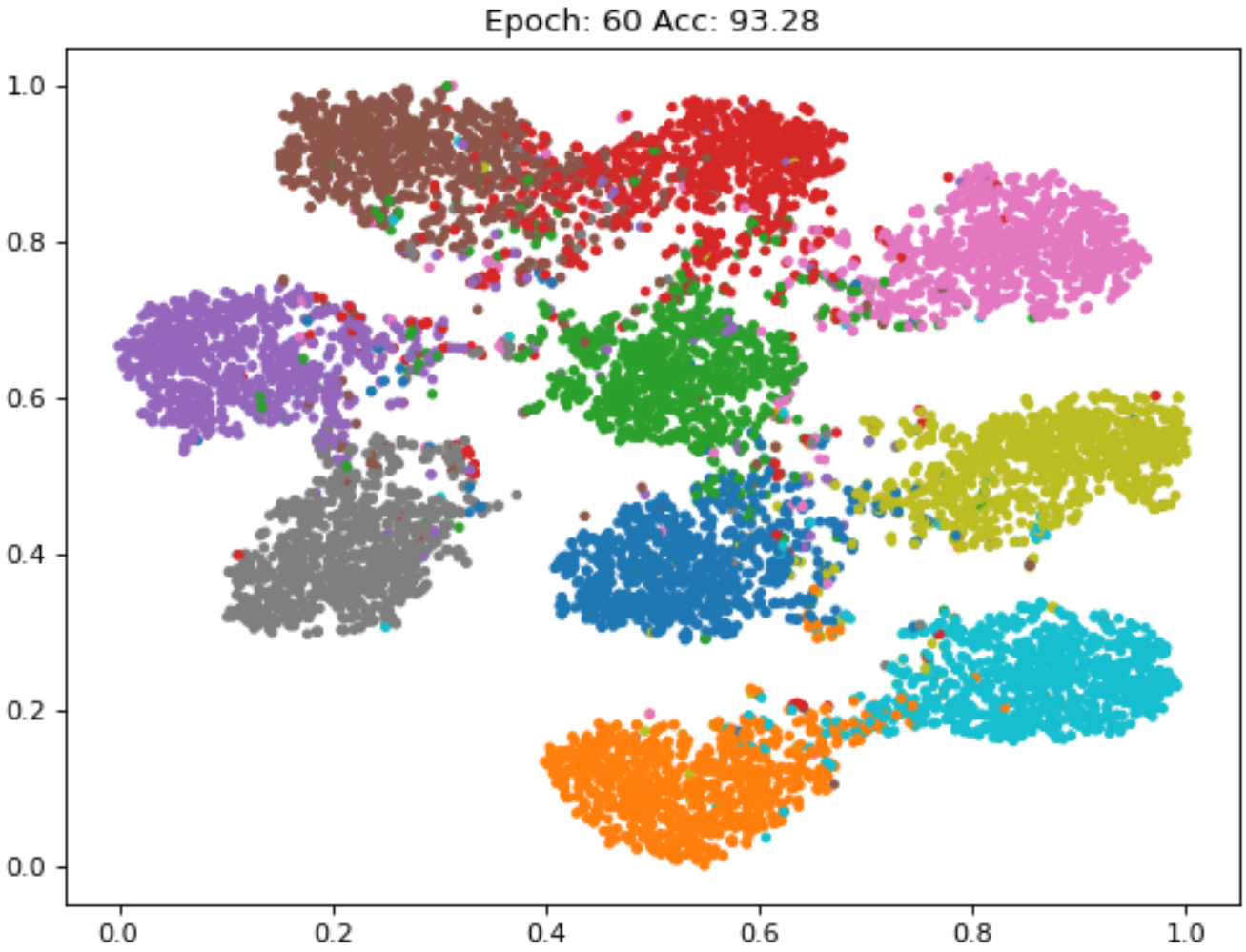
Visualization of the last hidden layer output for CIFAR10 samples



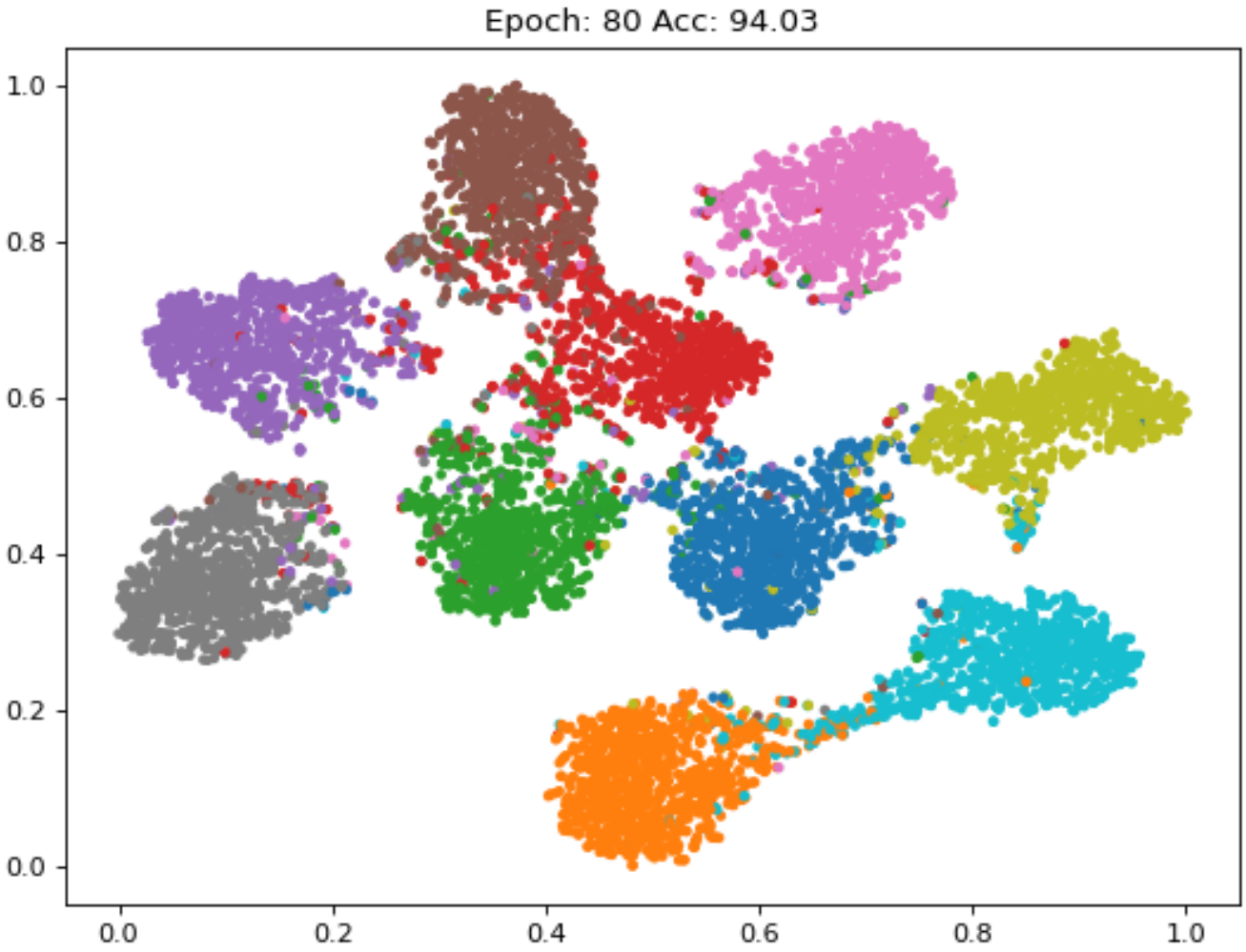
Visualization of the last hidden layer output for CIFAR10 samples



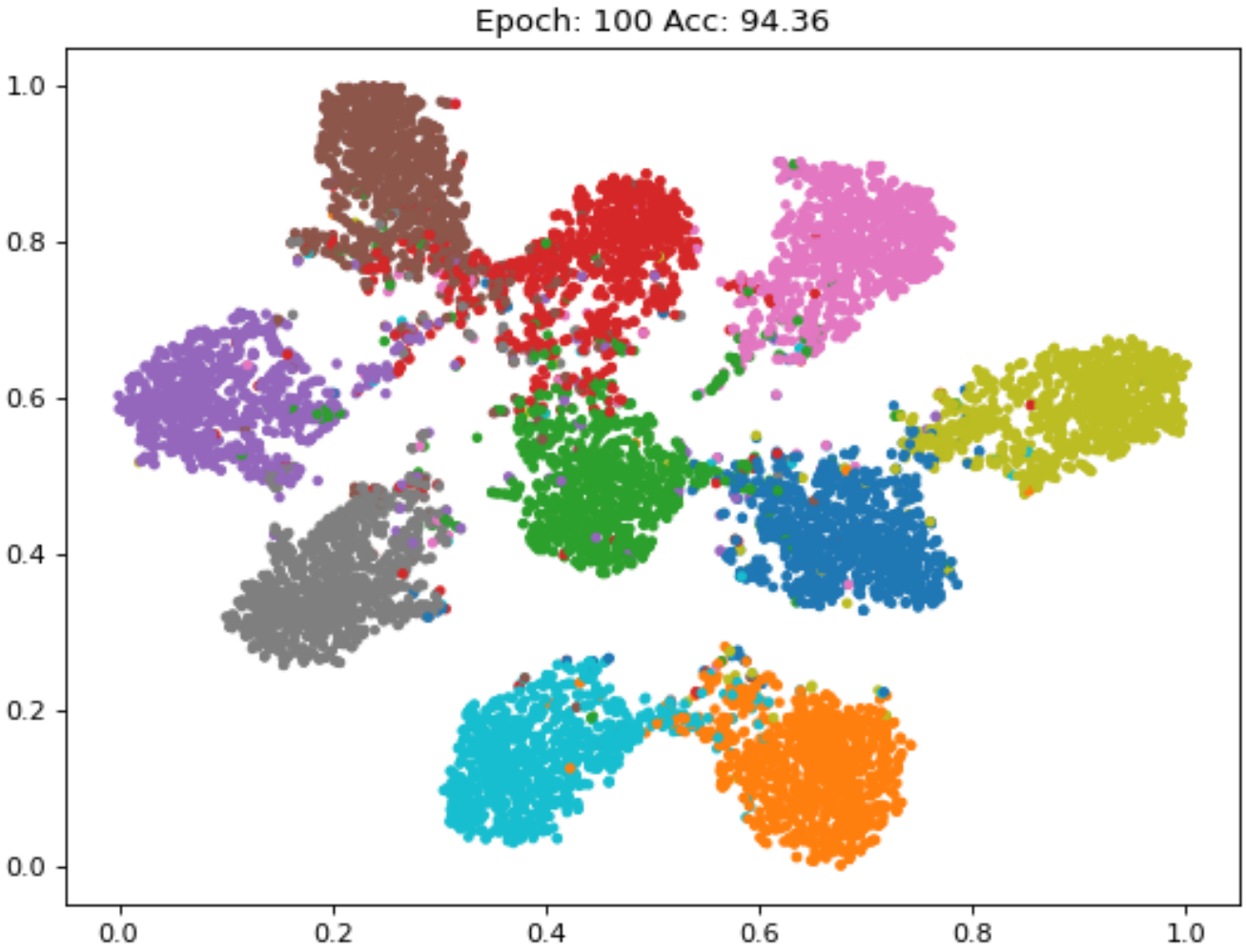
Visualization of the last hidden layer output for CIFAR10 samples



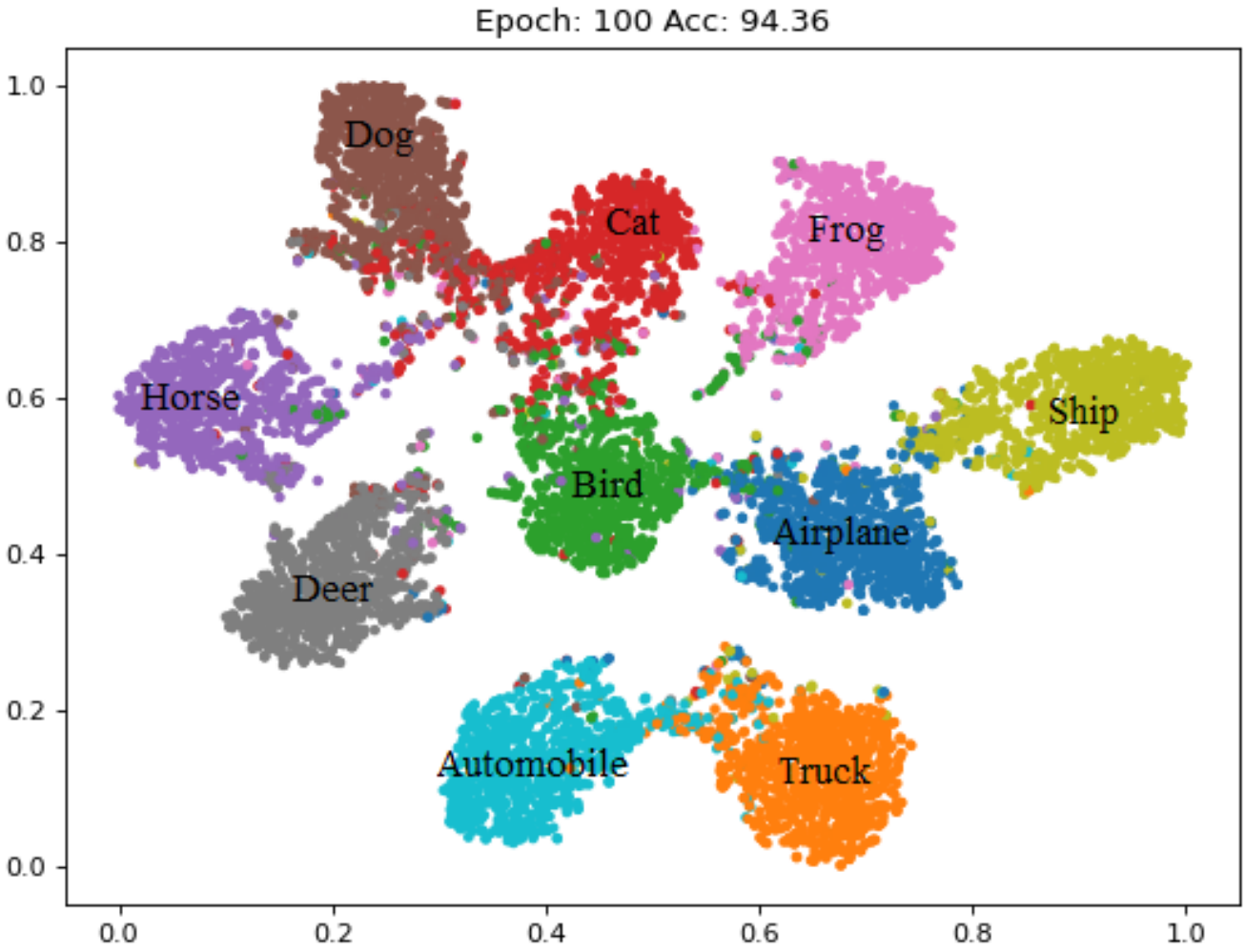
Visualization of the last hidden layer output for CIFAR10 samples



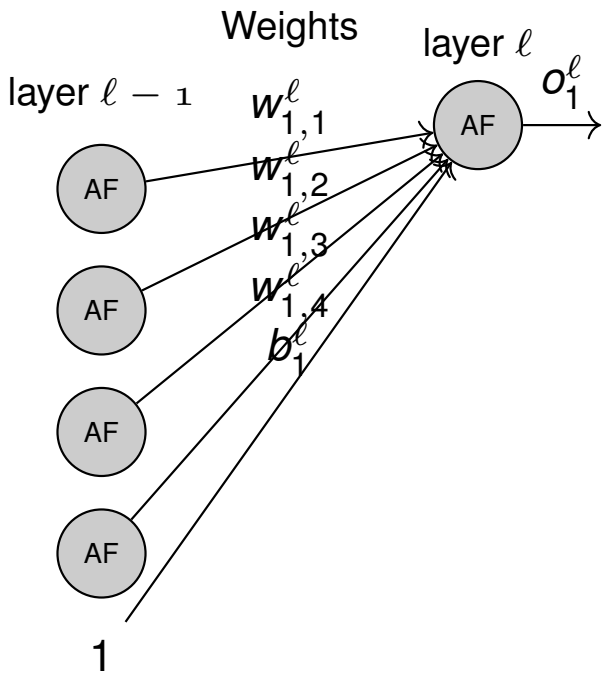
Visualization of the last hidden layer output for CIFAR10 samples



Visualization of the last hidden layer output for CIFAR10 samples



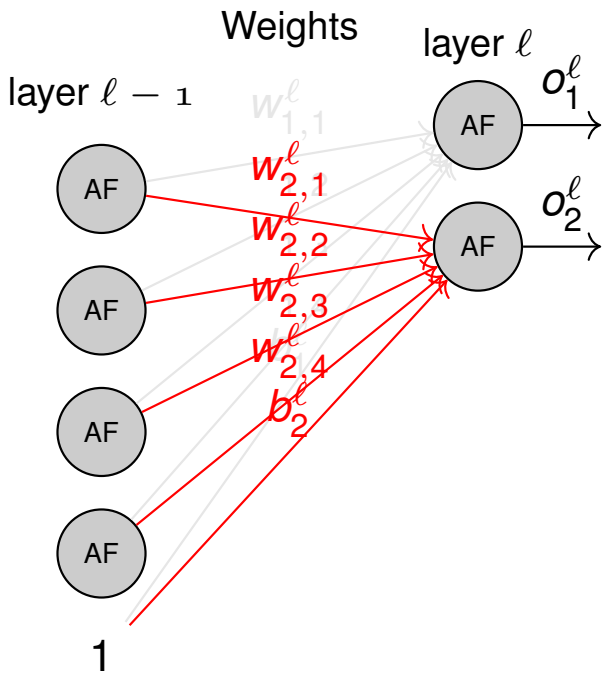
Fully connected neural networks



$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \boldsymbol{w_1^\ell}^T \boldsymbol{o}^{\ell-1} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

$$z_1^\ell = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + b_1^\ell$$

Fully connected neural networks

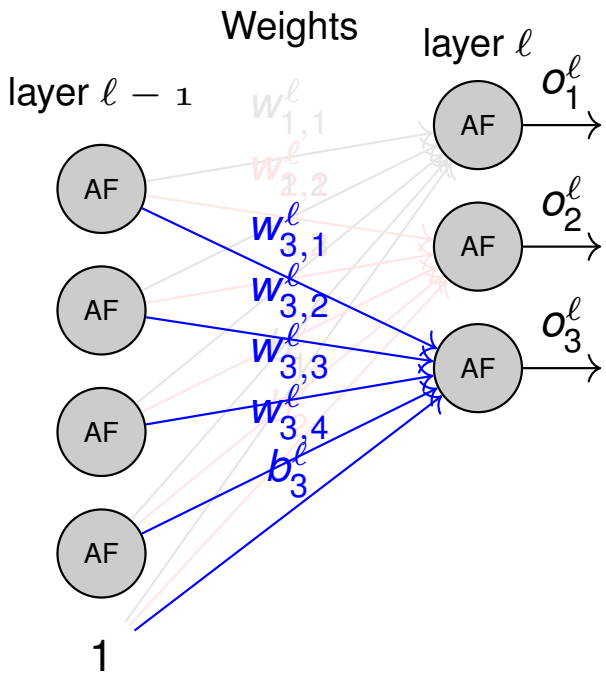


$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \boldsymbol{w_1^\ell}^T \boldsymbol{o^{\ell-1}} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

$$z_2^\ell = \sum_i w_{2,i}^\ell o_i^{\ell-1} + b_2^\ell = \boldsymbol{w_2^\ell}^T \boldsymbol{o^{\ell-1}} + b_2^\ell, \quad o_2^\ell = AF(z_2^\ell)$$

$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^\ell \\ b_2^\ell \end{bmatrix}$$

Fully connected neural networks



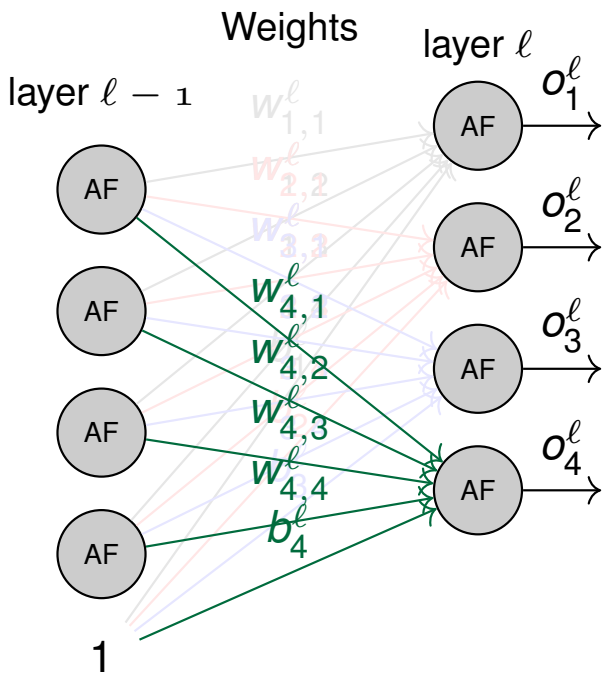
$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \boldsymbol{w_1^\ell}^T \boldsymbol{o^{\ell-1}} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

$$z_2^\ell = \sum_i w_{2,i}^\ell o_i^{\ell-1} + b_2^\ell = \boldsymbol{w_2^\ell}^T \boldsymbol{o^{\ell-1}} + b_2^\ell, \quad o_2^\ell = AF(z_2^\ell)$$

$$z_3^\ell = \sum_i w_{3,i}^\ell o_i^{\ell-1} + b_3^\ell = \boldsymbol{w_3^\ell}^T \boldsymbol{o^{\ell-1}} + b_3^\ell, \quad o_3^\ell = AF(z_3^\ell)$$

$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \\ w_{3,1}^\ell & w_{3,2}^\ell & w_{3,3}^\ell & w_{3,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^\ell \\ b_2^\ell \\ b_3^\ell \end{bmatrix}$$

Fully connected neural networks



$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \boldsymbol{w_1^\ell}^T \boldsymbol{o^{\ell-1}} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

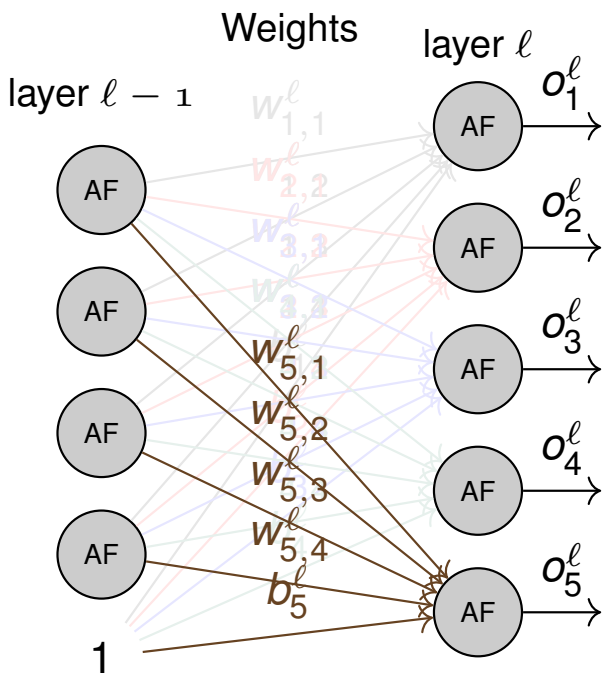
$$z_2^\ell = \sum_i w_{2,i}^\ell o_i^{\ell-1} + b_2^\ell = \boldsymbol{w_2^\ell}^T \boldsymbol{o^{\ell-1}} + b_2^\ell, \quad o_2^\ell = AF(z_2^\ell)$$

$$z_3^\ell = \sum_i w_{3,i}^\ell o_i^{\ell-1} + b_3^\ell = \boldsymbol{w_3^\ell}^T \boldsymbol{o^{\ell-1}} + b_3^\ell, \quad o_3^\ell = AF(z_3^\ell)$$

$$z_4^\ell = \sum_i w_{4,i}^\ell o_i^{\ell-1} + b_4^\ell = \boldsymbol{w_4^\ell}^T \boldsymbol{o^{\ell-1}} + b_4^\ell, \quad o_4^\ell = AF(z_4^\ell)$$

$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \\ z_4^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \\ w_{3,1}^\ell & w_{3,2}^\ell & w_{3,3}^\ell & w_{3,4}^\ell \\ w_{4,1}^\ell & w_{4,2}^\ell & w_{4,3}^\ell & w_{4,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^\ell \\ b_2^\ell \\ b_3^\ell \\ b_4^\ell \end{bmatrix}$$

Fully connected neural networks



$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \boldsymbol{w_1^\ell}^T \boldsymbol{o^{\ell-1}} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

$$z_2^\ell = \sum_i w_{2,i}^\ell o_i^{\ell-1} + b_2^\ell = \boldsymbol{w_2^\ell}^T \boldsymbol{o^{\ell-1}} + b_2^\ell, \quad o_2^\ell = AF(z_2^\ell)$$

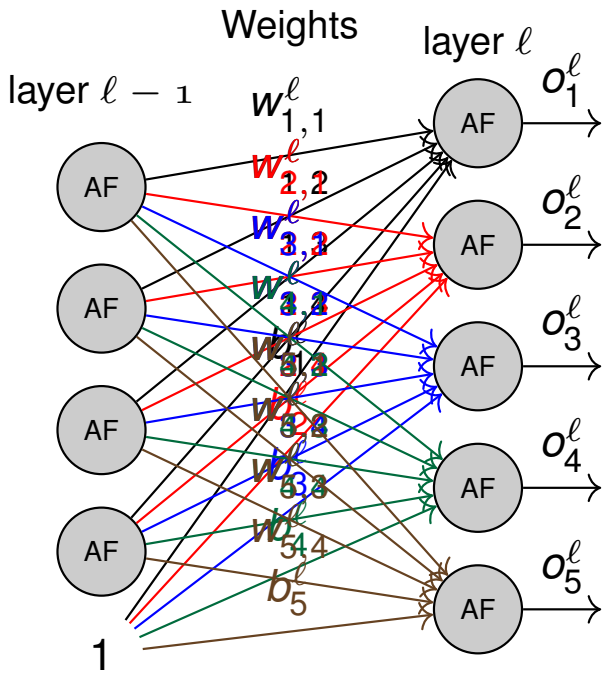
$$z_3^\ell = \sum_i w_{3,i}^\ell o_i^{\ell-1} + b_3^\ell = \boldsymbol{w_3^\ell}^T \boldsymbol{o^{\ell-1}} + b_3^\ell, \quad o_3^\ell = AF(z_3^\ell)$$

$$z_4^\ell = \sum_i w_{4,i}^\ell o_i^{\ell-1} + b_4^\ell = \boldsymbol{w_4^\ell}^T \boldsymbol{o^{\ell-1}} + b_4^\ell, \quad o_4^\ell = AF(z_4^\ell)$$

$$z_5^\ell = \sum_i w_{5,i}^\ell o_i^{\ell-1} + b_5^\ell = \boldsymbol{w_5^\ell}^T \boldsymbol{o^{\ell-1}} + b_5^\ell, \quad o_5^\ell = AF(z_5^\ell)$$

$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \\ z_4^\ell \\ z_5^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \\ w_{3,1}^\ell & w_{3,2}^\ell & w_{3,3}^\ell & w_{3,4}^\ell \\ w_{4,1}^\ell & w_{4,2}^\ell & w_{4,3}^\ell & w_{4,4}^\ell \\ w_{5,1}^\ell & w_{5,2}^\ell & w_{5,3}^\ell & w_{5,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^\ell \\ b_2^\ell \\ b_3^\ell \\ b_4^\ell \\ b_5^\ell \end{bmatrix}$$

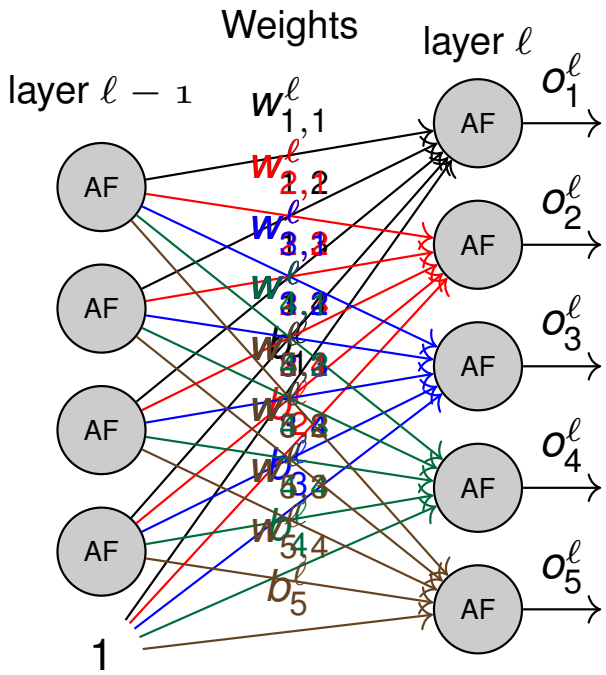
Vector activation



$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \\ z_4^\ell \\ z_5^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \\ w_{3,1}^\ell & w_{3,2}^\ell & w_{3,3}^\ell & w_{3,4}^\ell \\ w_{4,1}^\ell & w_{4,2}^\ell & w_{4,3}^\ell & w_{4,4}^\ell \\ w_{5,1}^\ell & w_{5,2}^\ell & w_{5,3}^\ell & w_{5,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^\ell \\ b_2^\ell \\ b_3^\ell \\ b_4^\ell \\ b_5^\ell \end{bmatrix}$$

$$\begin{bmatrix} o_1^\ell \\ o_2^\ell \\ o_3^\ell \\ o_4^\ell \\ o_5^\ell \end{bmatrix} = \begin{bmatrix} AF(z_1^\ell) \\ AF(z_2^\ell) \\ AF(z_3^\ell) \\ AF(z_4^\ell) \\ AF(z_5^\ell) \end{bmatrix}$$

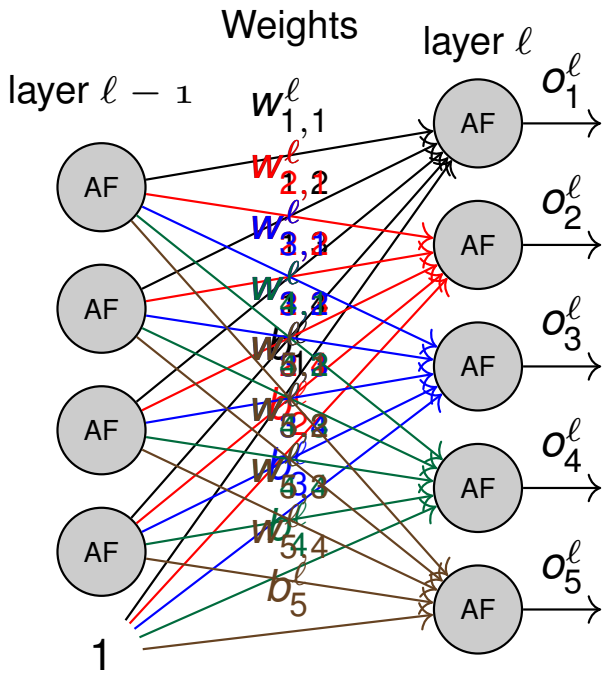
Vector activation



$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \\ z_4^\ell \\ z_5^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \\ w_{3,1}^\ell & w_{3,2}^\ell & w_{3,3}^\ell & w_{3,4}^\ell \\ w_{4,1}^\ell & w_{4,2}^\ell & w_{4,3}^\ell & w_{4,4}^\ell \\ w_{5,1}^\ell & w_{5,2}^\ell & w_{5,3}^\ell & w_{5,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^\ell \\ b_2^\ell \\ b_3^\ell \\ b_4^\ell \\ b_5^\ell \end{bmatrix}$$

$$\begin{bmatrix} o_1^\ell \\ o_2^\ell \\ o_3^\ell \\ o_4^\ell \\ o_5^\ell \end{bmatrix} = \begin{bmatrix} AF(z_1^\ell) \\ AF(z_2^\ell) \\ AF(z_3^\ell) \\ AF(z_4^\ell) \\ AF(z_5^\ell) \end{bmatrix} = AF\left(\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \\ z_4^\ell \\ z_5^\ell \end{bmatrix} \right)$$

Vector activation

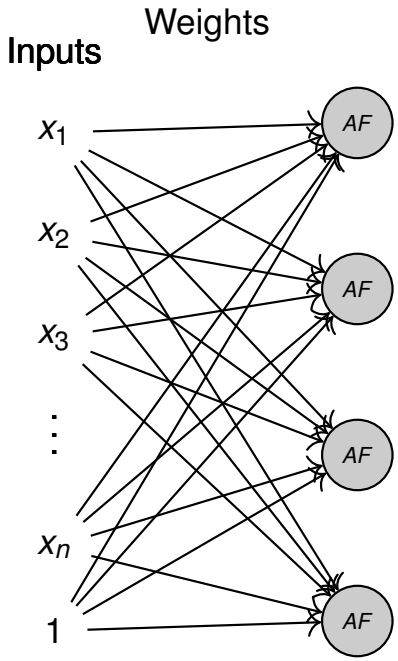


$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \\ z_4^\ell \\ z_5^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \\ w_{3,1}^\ell & w_{3,2}^\ell & w_{3,3}^\ell & w_{3,4}^\ell \\ w_{4,1}^\ell & w_{4,2}^\ell & w_{4,3}^\ell & w_{4,4}^\ell \\ w_{5,1}^\ell & w_{5,2}^\ell & w_{5,3}^\ell & w_{5,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^\ell \\ b_2^\ell \\ b_3^\ell \\ b_4^\ell \\ b_5^\ell \end{bmatrix}$$

$$\begin{bmatrix} o_1^\ell \\ o_2^\ell \\ o_3^\ell \\ o_4^\ell \\ o_5^\ell \end{bmatrix} = \begin{bmatrix} AF(z_1^\ell) \\ AF(z_2^\ell) \\ AF(z_3^\ell) \\ AF(z_4^\ell) \\ AF(z_5^\ell) \end{bmatrix} = AF\left(\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \\ z_4^\ell \\ z_5^\ell \end{bmatrix} \right)$$

$$\boldsymbol{o}^\ell = AF(W^\ell \boldsymbol{o}^{\ell-1} + \boldsymbol{b}^\ell)$$

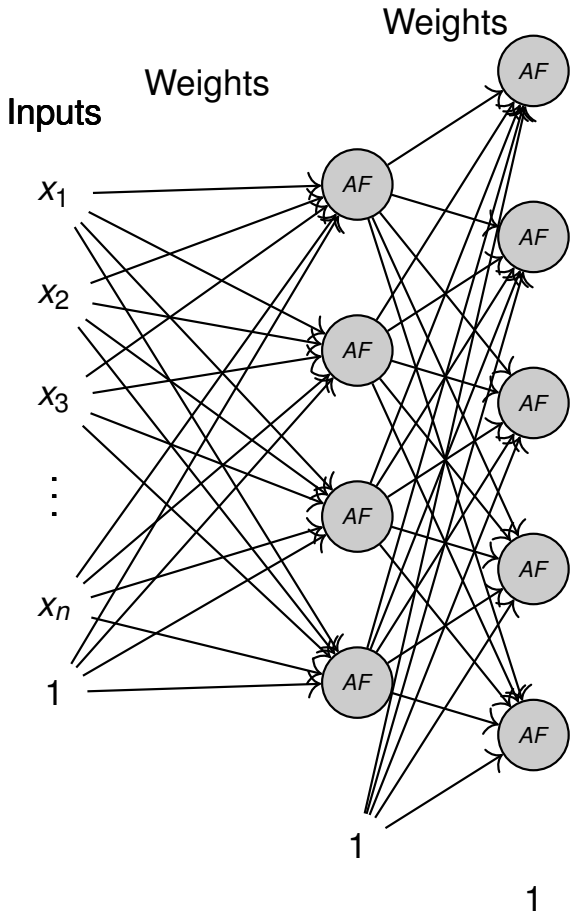
Fully connected neural networks



1

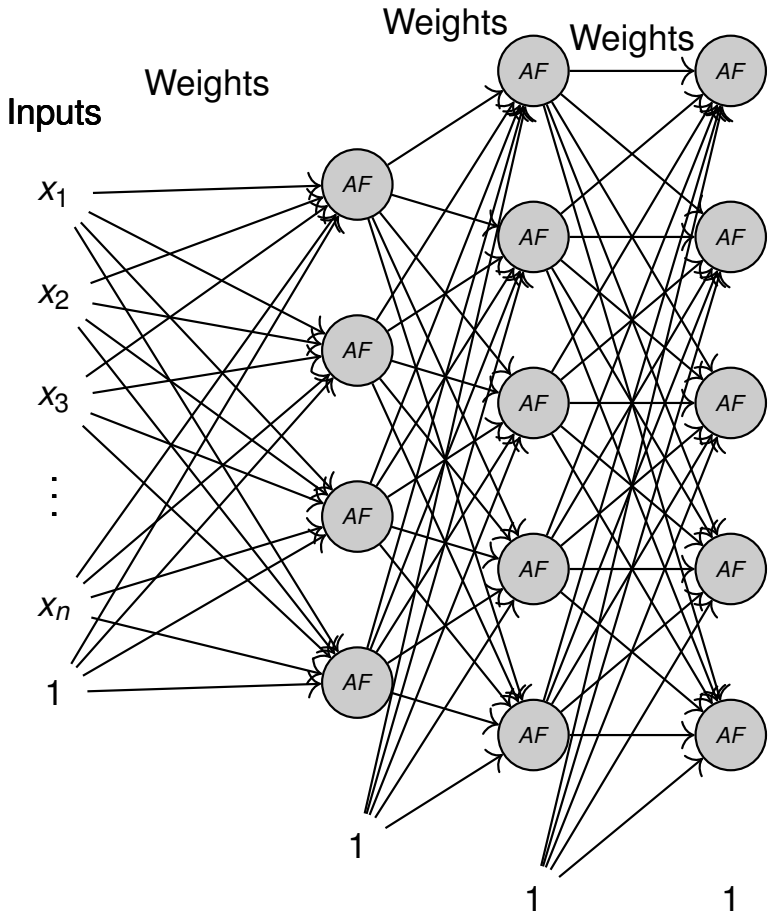
$$\boldsymbol{o}^1 = AF(W^1\boldsymbol{x} + \boldsymbol{b}^1), \quad W^1 \in \mathbb{R}^{4\times n},\ \boldsymbol{x} \in \mathbb{R}^n,\ \boldsymbol{o}^1 \in \mathbb{R}^4,\ \boldsymbol{b} \in \mathbb{R}^4.$$

Fully connected neural networks



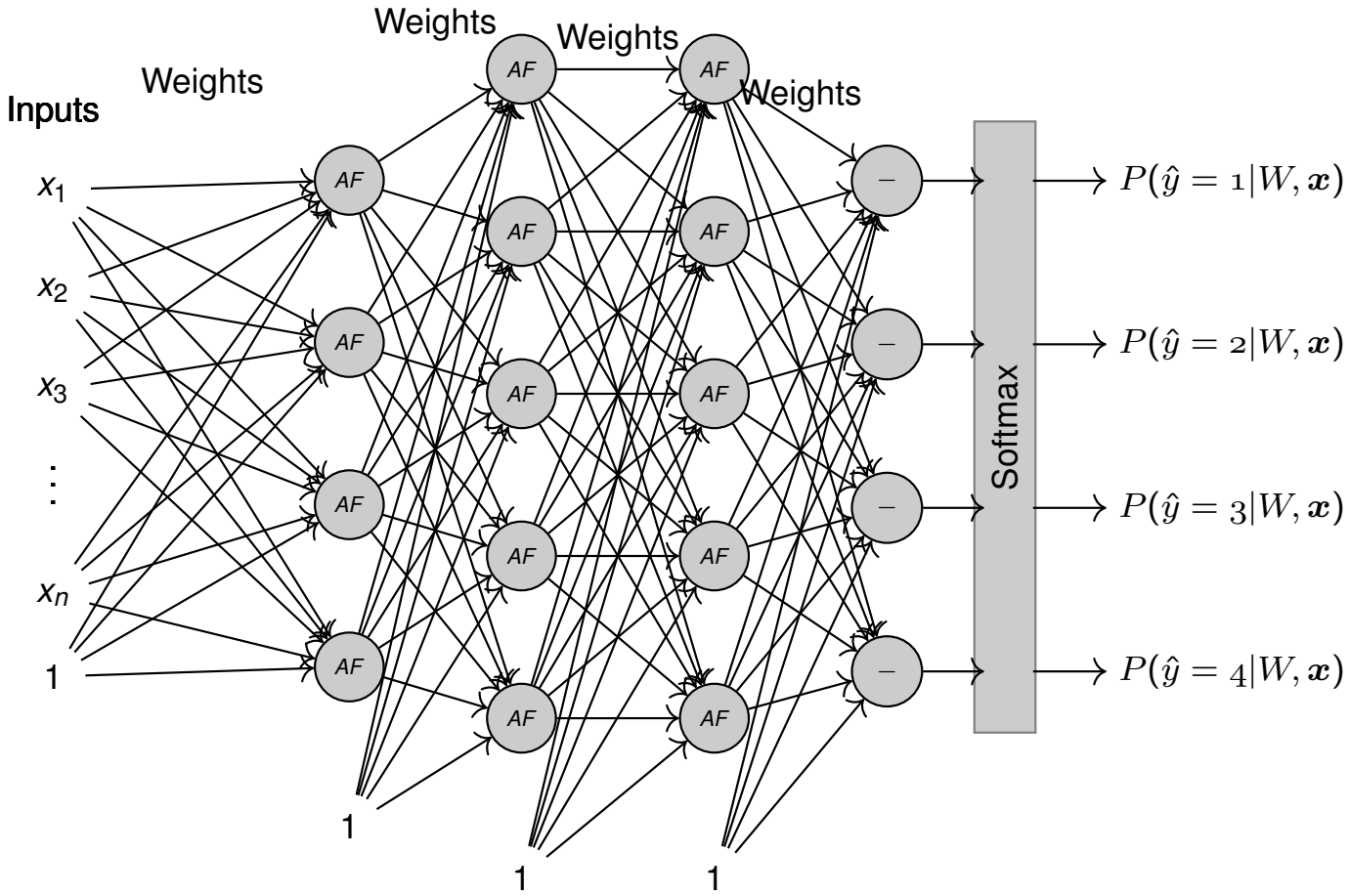
$$\boldsymbol{o^2} = AF(W^2\boldsymbol{o^1} + \boldsymbol{b^2}), \quad W^2 \in \mathbb{R}^{5\times 4},\ \boldsymbol{o^1} \in \mathbb{R}^4,\ \boldsymbol{o^2} \in \mathbb{R}^5,\ \boldsymbol{b^2} \in \mathbb{R}^5.$$

Fully connected neural networks



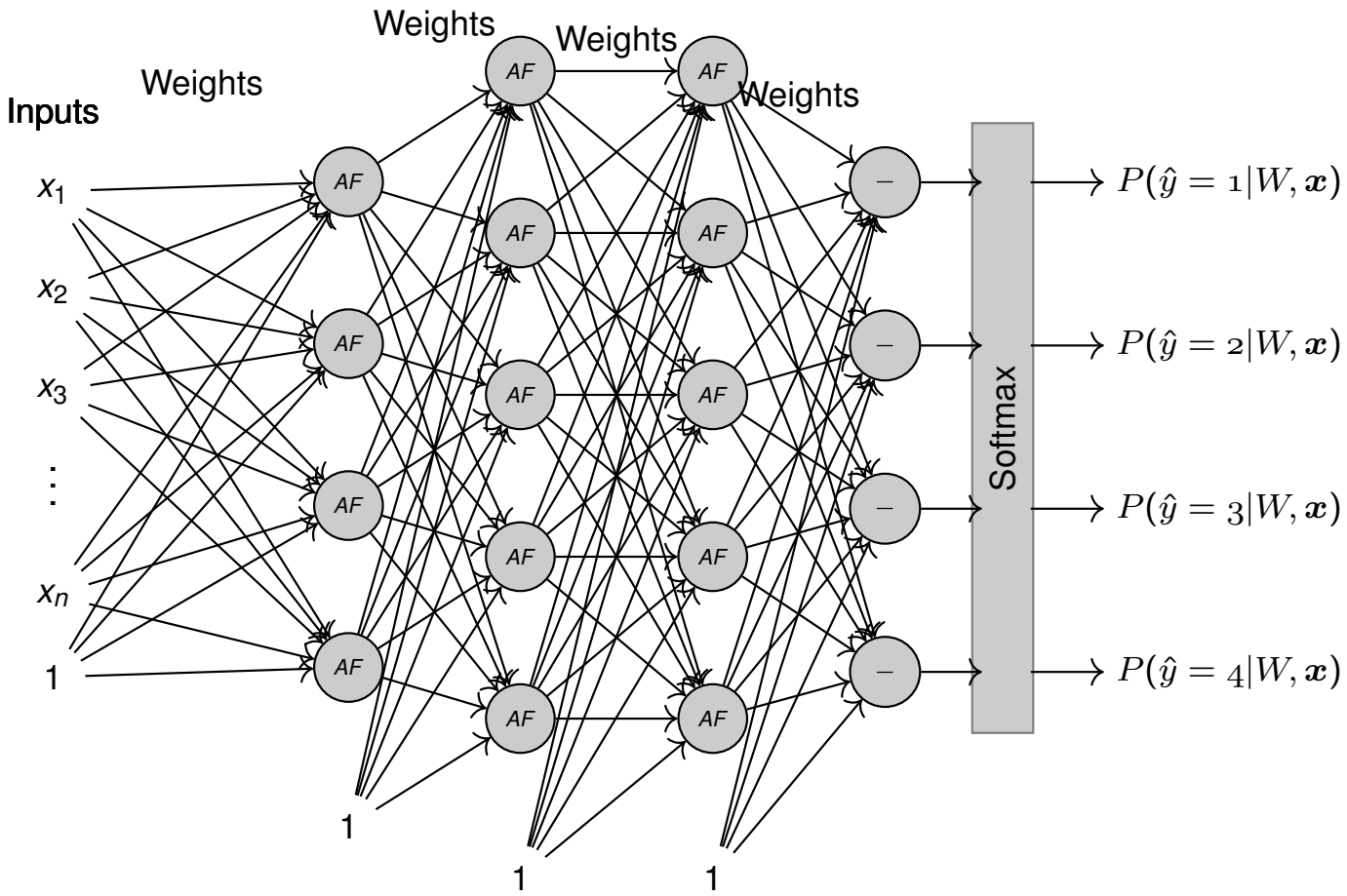
$$\boldsymbol{o}^3 = AF(W^3\boldsymbol{o}^2 + \boldsymbol{b}^3), \quad W^3 \in \mathbb{R}^{5\times 5},\ \boldsymbol{o}^2 \in \mathbb{R}^5,\ \boldsymbol{o}^3 \in \mathbb{R}^5,\ \boldsymbol{b}^3 \in \mathbb{R}^5.$$

Fully connected neural networks



$$\hat{\boldsymbol{y}} = Softmax(W^4 \boldsymbol{o}^3 + \boldsymbol{b}^4), \quad W^4 \in \mathbb{R}^{4\times 5},\ \boldsymbol{o}^3 \in \mathbb{R}^5,\ \hat{\boldsymbol{y}} \in \mathbb{R}^4,\ \boldsymbol{b}^4 \in \mathbb{R}^4.$$

Fully connected neural networks



$$\hat{\boldsymbol{y}} = Softmax(W^4(AF(W^3(AF(W^2(AF(W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

Batch of inputs

- The output of the first hidden layer for a batch of inputs

$$\begin{bmatrix} z_1^{1,(1)} & z_1^{1,(2)} & z_1^{1,(3)} \\ z_2^{1,(1)} & z_2^{1,(2)} & z_2^{1,(3)} \\ z_3^{1,(1)} & z_3^{1,(2)} & z_3^{1,(3)} \\ z_4^{1,(1)} & z_4^{1,(2)} & z_4^{1,(3)} \end{bmatrix} = \begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & w_{1,3}^1 & \dots & w_{1,n}^1 \\ w_{2,1}^1 & w_{2,2}^1 & w_{2,3}^1 & \dots & w_{2,n}^1 \\ w_{3,1}^1 & w_{3,2}^1 & w_{3,3}^1 & \dots & w_{3,n}^1 \\ w_{4,1}^1 & w_{4,2}^1 & w_{4,3}^1 & \dots & w_{4,n}^1 \end{bmatrix} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} \\ x_3^{(1)} & x_3^{(2)} & x_3^{(3)} \\ \vdots & \vdots & \vdots \\ x_n^{(1)} & x_n^{(2)} & x_n^{(3)} \end{bmatrix}$$

$$+ \begin{bmatrix} b_1^1 & b_1^1 & b_1^1 \\ b_2^1 & b_2^1 & b_2^1 \\ b_3^1 & b_3^1 & b_3^1 \\ b_4^1 & b_4^1 & b_4^1 \end{bmatrix}$$

$$\begin{bmatrix} o_1^{1,(1)} & o_1^{1,(2)} & o_1^{1,(3)} \\ o_2^{1,(1)} & o_2^{1,(2)} & o_2^{1,(3)} \\ o_3^{1,(1)} & o_3^{1,(2)} & o_3^{1,(3)} \\ o_4^{1,(1)} & o_4^{1,(2)} & o_4^{1,(3)} \end{bmatrix} = AF\left(\begin{bmatrix} z_1^{1,(1)} & z_1^{1,(2)} & z_1^{1,(3)} \\ z_2^{1,(1)} & z_2^{1,(2)} & z_2^{1,(3)} \\ z_3^{1,(1)} & z_3^{1,(2)} & z_3^{1,(3)} \\ z_4^{1,(1)} & z_4^{1,(2)} & z_4^{1,(3)} \end{bmatrix} \right)$$

Batch of inputs

- The output of the first hidden layer for a batch of inputs

$$\begin{bmatrix} z_1^{1,(1)} & z_1^{1,(2)} & z_1^{1,(3)} \\ z_2^{1,(1)} & z_2^{1,(2)} & z_2^{1,(3)} \\ z_3^{1,(1)} & z_3^{1,(2)} & z_3^{1,(3)} \\ z_4^{1,(1)} & z_4^{1,(2)} & z_4^{1,(3)} \end{bmatrix} = \begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & w_{1,3}^1 & \dots & w_{1,n}^1 \\ w_{2,1}^1 & w_{2,2}^1 & w_{2,3}^1 & \dots & w_{2,n}^1 \\ w_{3,1}^1 & w_{3,2}^1 & w_{3,3}^1 & \dots & w_{3,n}^1 \\ w_{4,1}^1 & w_{4,2}^1 & w_{4,3}^1 & \dots & w_{4,n}^1 \end{bmatrix} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} \\ x_3^{(1)} & x_3^{(2)} & x_3^{(3)} \\ \vdots & \vdots & \vdots \\ x_n^{(1)} & x_n^{(2)} & x_n^{(3)} \end{bmatrix}$$

$$+ \begin{bmatrix} b_1^1 & b_1^1 & b_1^1 \\ b_2^1 & b_2^1 & b_2^1 \\ b_3^1 & b_3^1 & b_3^1 \\ b_4^1 & b_4^1 & b_4^1 \end{bmatrix}$$

$$\begin{bmatrix} \begin{bmatrix} o_1^{1,(1)} \\ o_2^{1,(1)} \\ o_3^{1,(1)} \\ o_4^{1,(1)} \end{bmatrix} & \begin{bmatrix} o_1^{1,(2)} \\ o_2^{1,(2)} \\ o_3^{1,(2)} \\ o_4^{1,(2)} \end{bmatrix} & \begin{bmatrix} o_1^{1,(3)} \\ o_2^{1,(3)} \\ o_3^{1,(3)} \\ o_4^{1,(3)} \end{bmatrix} \end{bmatrix} = AF\left(\begin{bmatrix} z_1^{1,(1)} & z_1^{1,(2)} & z_1^{1,(3)} \\ z_2^{1,(1)} & z_2^{1,(2)} & z_2^{1,(3)} \\ z_3^{1,(1)} & z_3^{1,(2)} & z_3^{1,(3)} \\ z_4^{1,(1)} & z_4^{1,(2)} & z_4^{1,(3)} \end{bmatrix} \right)$$

$$O^1 = AF(W^1 X + B^1), \quad W^1 \in \mathbb{R}^{4\times n},\ X \in \mathbb{R}^{n\times 3},\ O^1 \in \mathbb{R}^{4\times 3},\ B \in \mathbb{R}^{4\times 3}.$$

Activation functions

What happens if we build a neural network with no activation function?

$$\hat{\boldsymbol{y}} = Softmax(W^4(AF(W^3(AF(W^2(AF(W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

Activation functions

What happens if we build a neural network with no activation function?

$$\hat{\boldsymbol{y}} = Softmax(W^4(AF(W^3(AF(W^2(AF(W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

$$\hat{\boldsymbol{y}} = Softmax(W^4(W^3(W^2(W^1\boldsymbol{x} + \boldsymbol{b}^1) + \boldsymbol{b}^2) + \boldsymbol{b}^3) + \boldsymbol{b}^4)$$

Activation functions

What happens if we build a neural network with no activation function?

$$\hat{\boldsymbol{y}} = Softmax(W^4(AF(W^3(AF(W^2(AF(W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

$$\hat{\boldsymbol{y}} = Softmax(W^4(W^3(W^2(W^1\boldsymbol{x} + \boldsymbol{b}^1) + \boldsymbol{b}^2) + \boldsymbol{b}^3) + \boldsymbol{b}^4)$$

$$\hat{\boldsymbol{y}} = Softmax(W^4W^3W^2W^1\boldsymbol{x} + b')$$

Activation functions

What happens if we build a neural network with no activation function?

$$\hat{\boldsymbol{y}} = Softmax(W^4(AF(W^3(AF(W^2(AF(W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

$$\hat{\boldsymbol{y}} = Softmax(W^4(W^3(W^2(W^1\boldsymbol{x} + \boldsymbol{b}^1) + \boldsymbol{b}^2) + \boldsymbol{b}^3) + \boldsymbol{b}^4)$$

$$\hat{\boldsymbol{y}} = Softmax(W^4W^3W^2W^1\boldsymbol{x} + b')$$

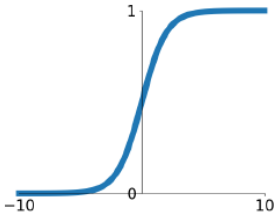
$$\hat{\boldsymbol{y}} = Softmax(W'\boldsymbol{x} + b')$$

We end up with a softmax classifier!

Activation functions

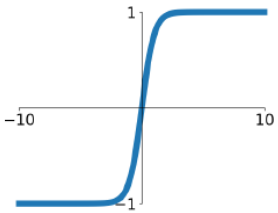
Sigmoid

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



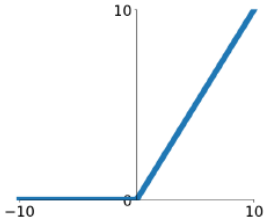
tanh

$$\tanh(x)$$



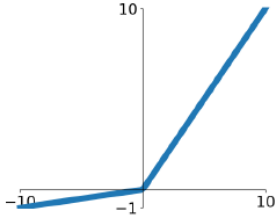
ReLU

$$\max(0, x)$$



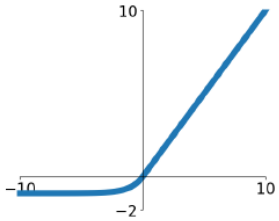
Leaky ReLU

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



ELU

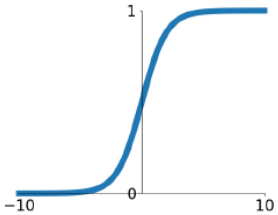
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation functions

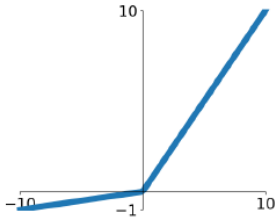
Sigmoid

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



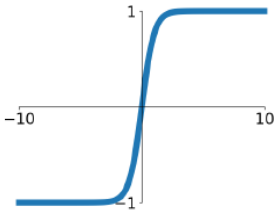
Leaky ReLU

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



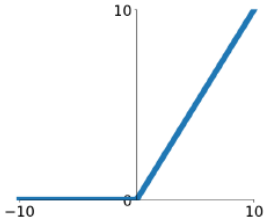
tanh

$$\tanh(x)$$



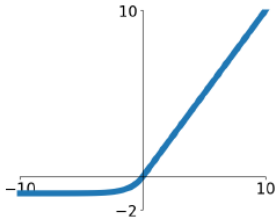
ReLU

$$\max(0, x)$$



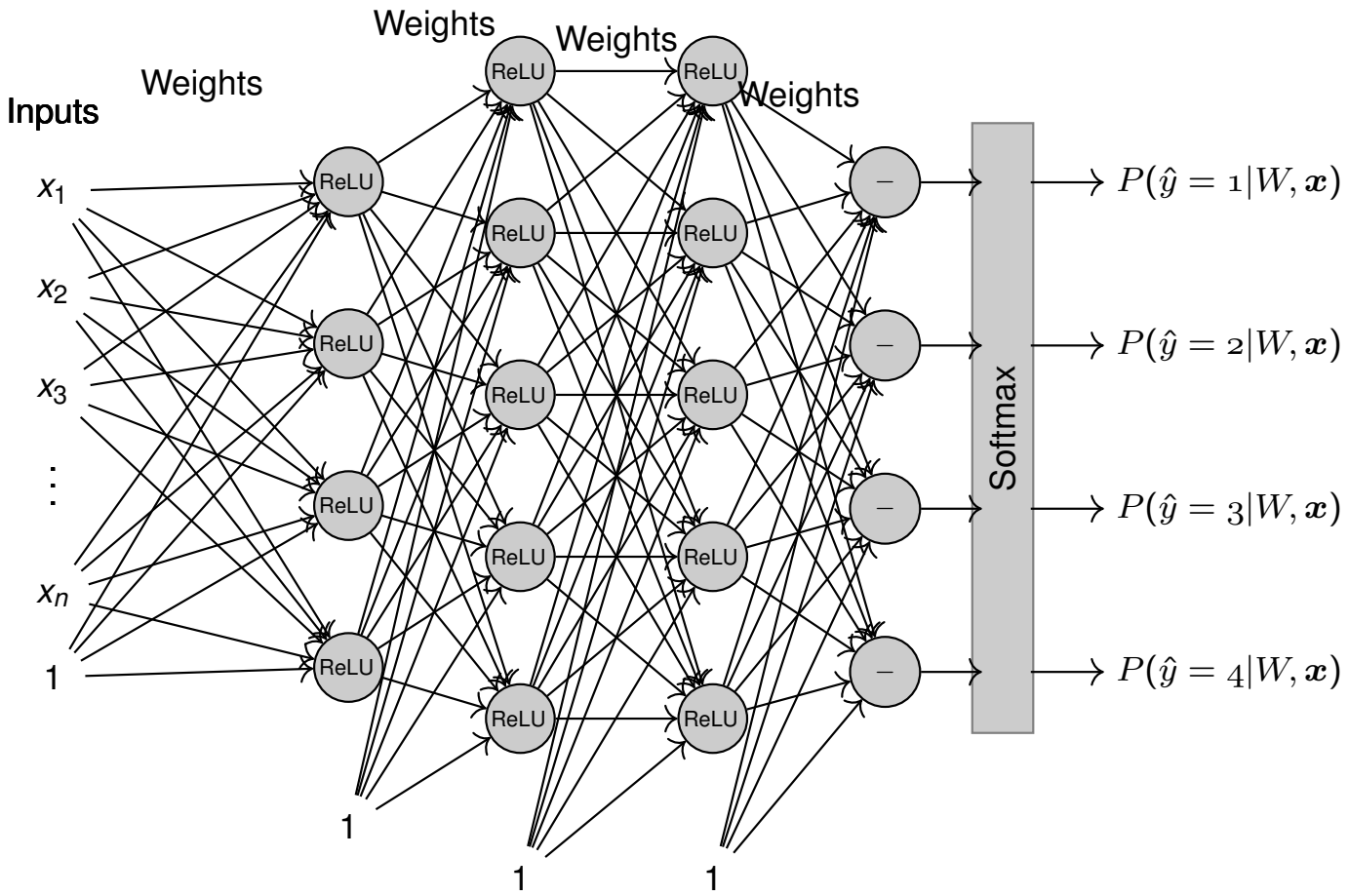
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



ReLU is a good default choice for most problems

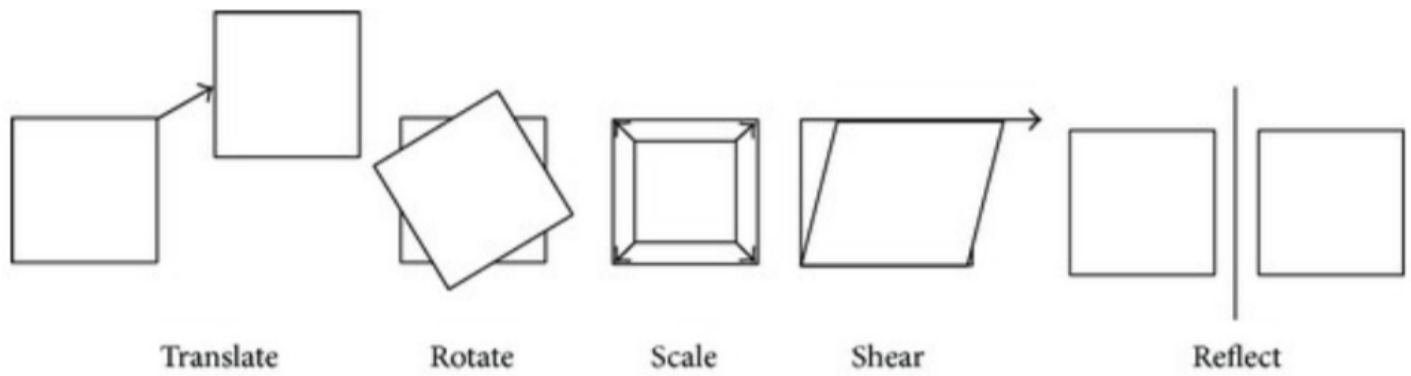
ReLU activation function



$$\hat{\boldsymbol{y}} = Softmax(W^4(max(0, W^3(max(0, W^2(max(0, W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

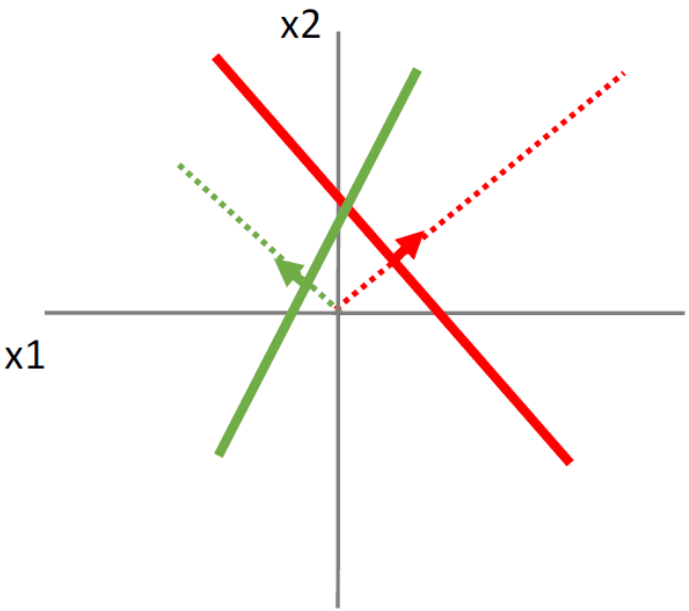
Affine transformation

- Affine transformation
  - Translate
  - Rotate
  - Scale
  - Shear
  - Reflect
- Preserve parallel lines



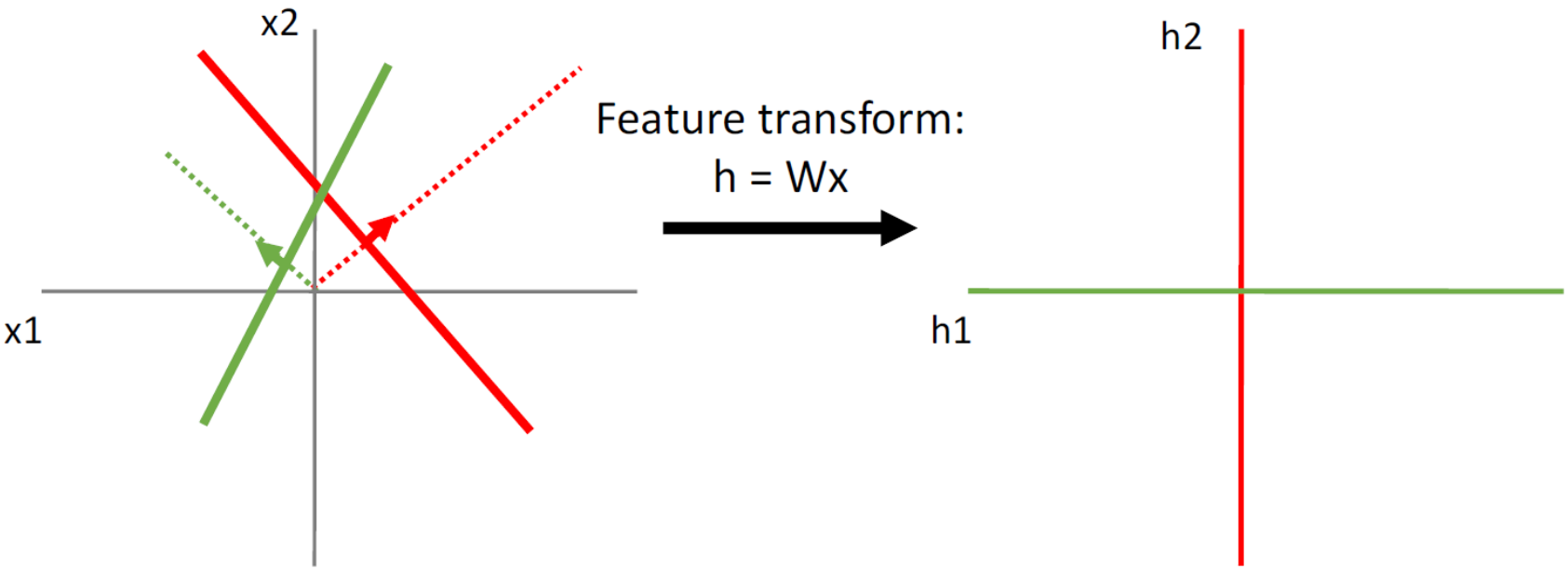
Space warping

- Consider a linear transform: $\boldsymbol{h} = W^T \boldsymbol{x}$ Where $\boldsymbol{x}, \boldsymbol{h}$ are both 2-dimensional



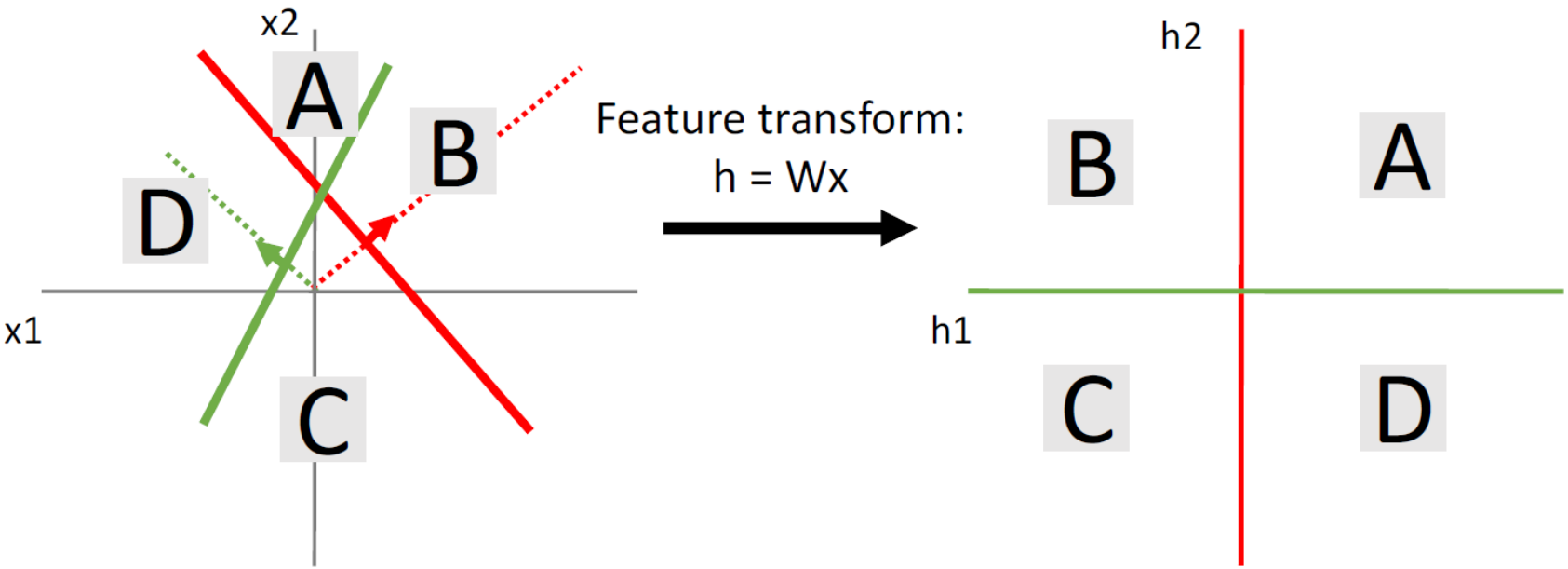
Space warping

- Consider a linear transform: $\boldsymbol{h} = W^T \boldsymbol{x}$ Where $\boldsymbol{x}, \boldsymbol{h}$ are both 2-dimensional



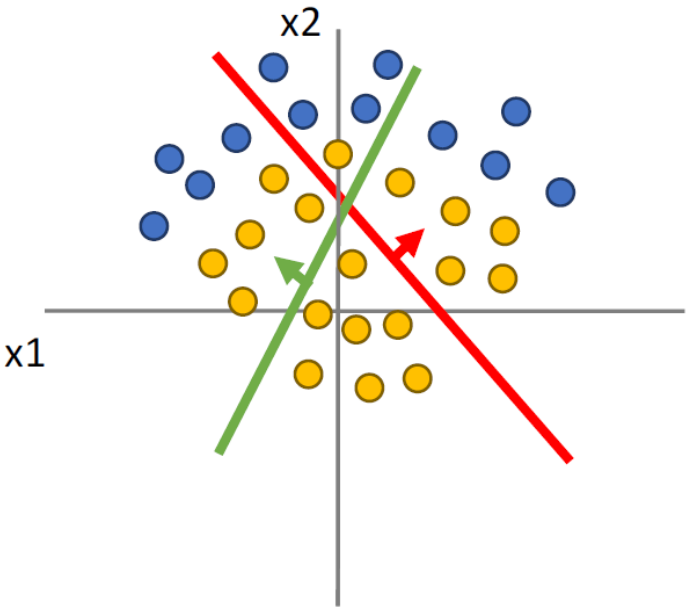
Space warping

- Consider a linear transform: $\boldsymbol{h} = W^T \boldsymbol{x}$ Where $\boldsymbol{x}, \boldsymbol{h}$ are both 2-dimensional



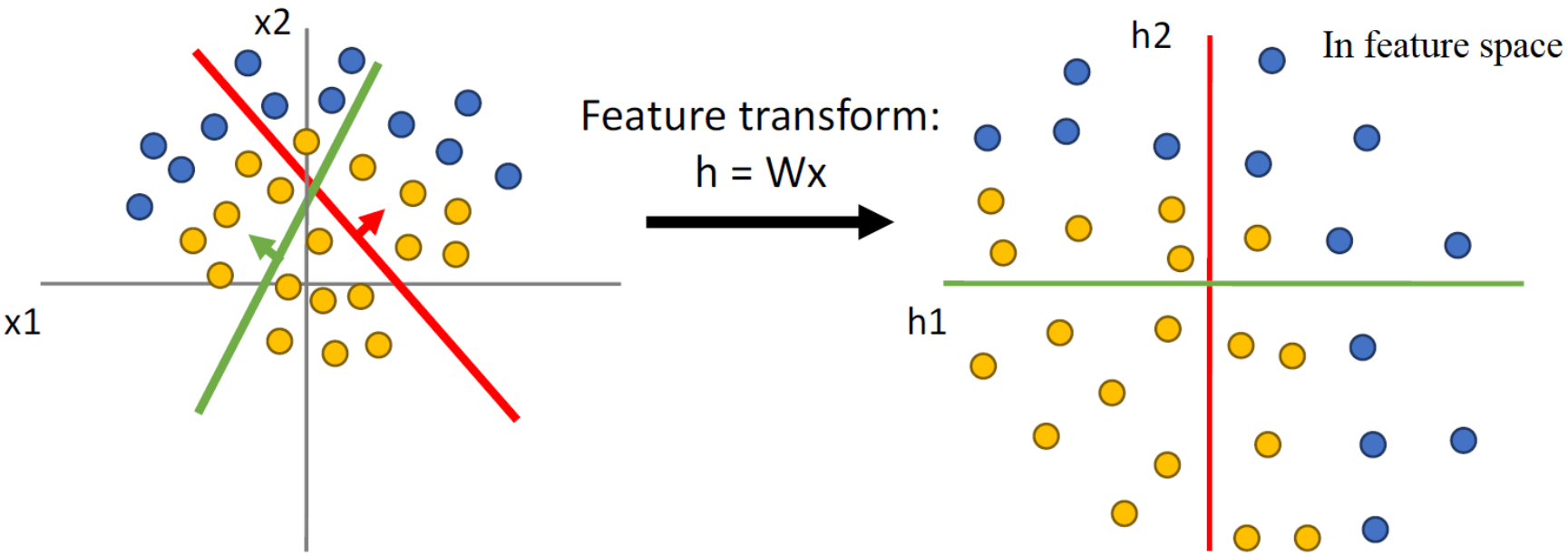
Space warping

- Consider a linear transform: $\boldsymbol{h} = W^T \boldsymbol{x}$ Where $\boldsymbol{x}, \boldsymbol{h}$ are both 2-dimensional
- Points not linearly separable in original space



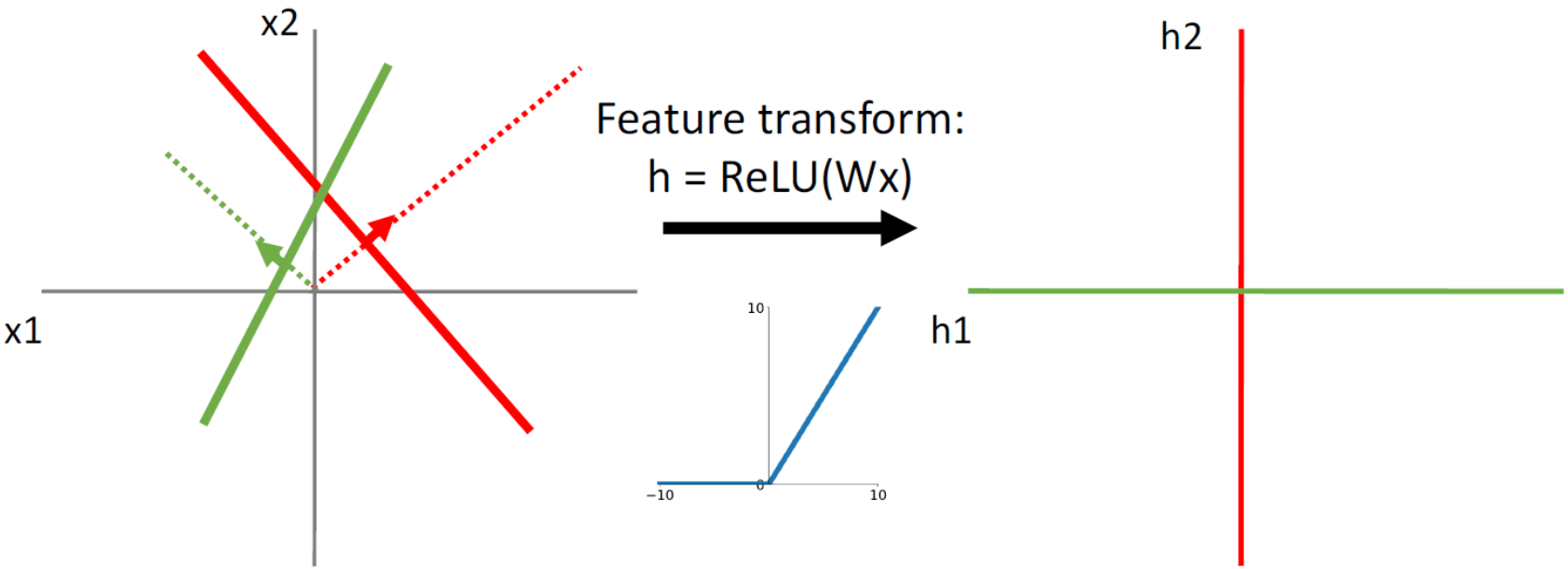
Space warping

- Consider a linear transform: $\boldsymbol{h} = W^T \boldsymbol{x}$ Where $\boldsymbol{x}, \boldsymbol{h}$ are both 2-dimensional
- Points not linearly separable in original space
  - Not linearly separable in feature space



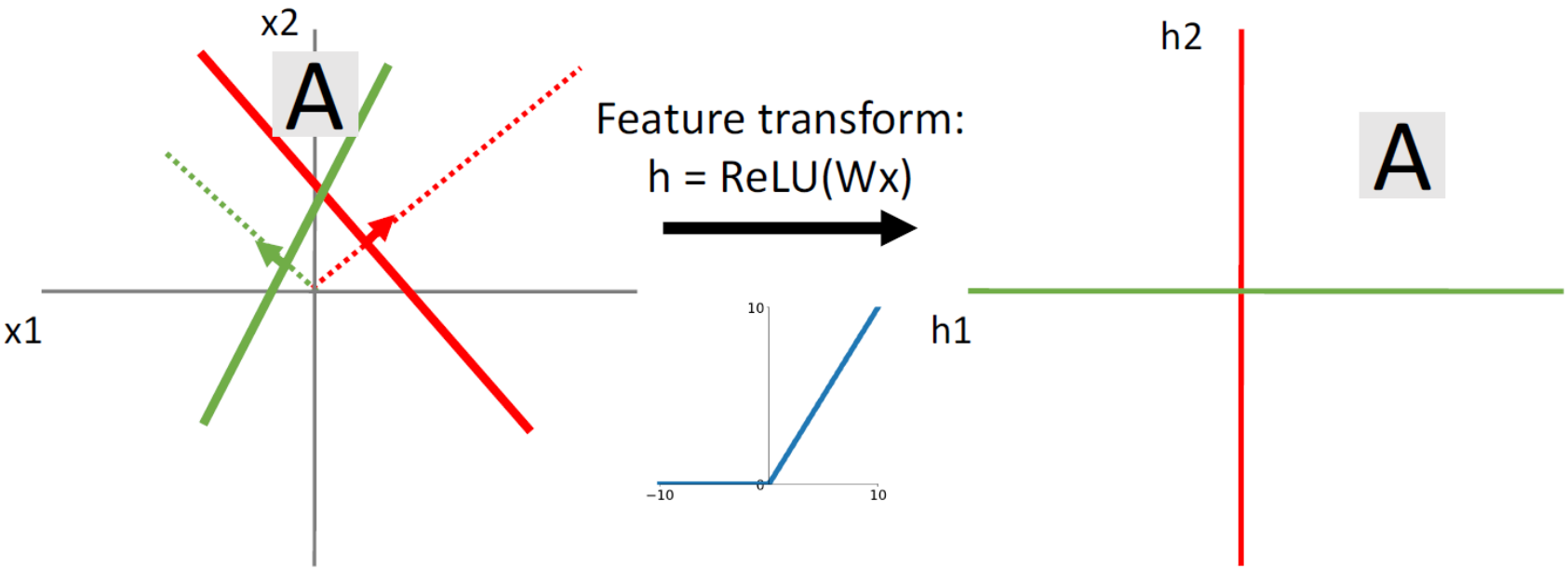
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W^T \boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.



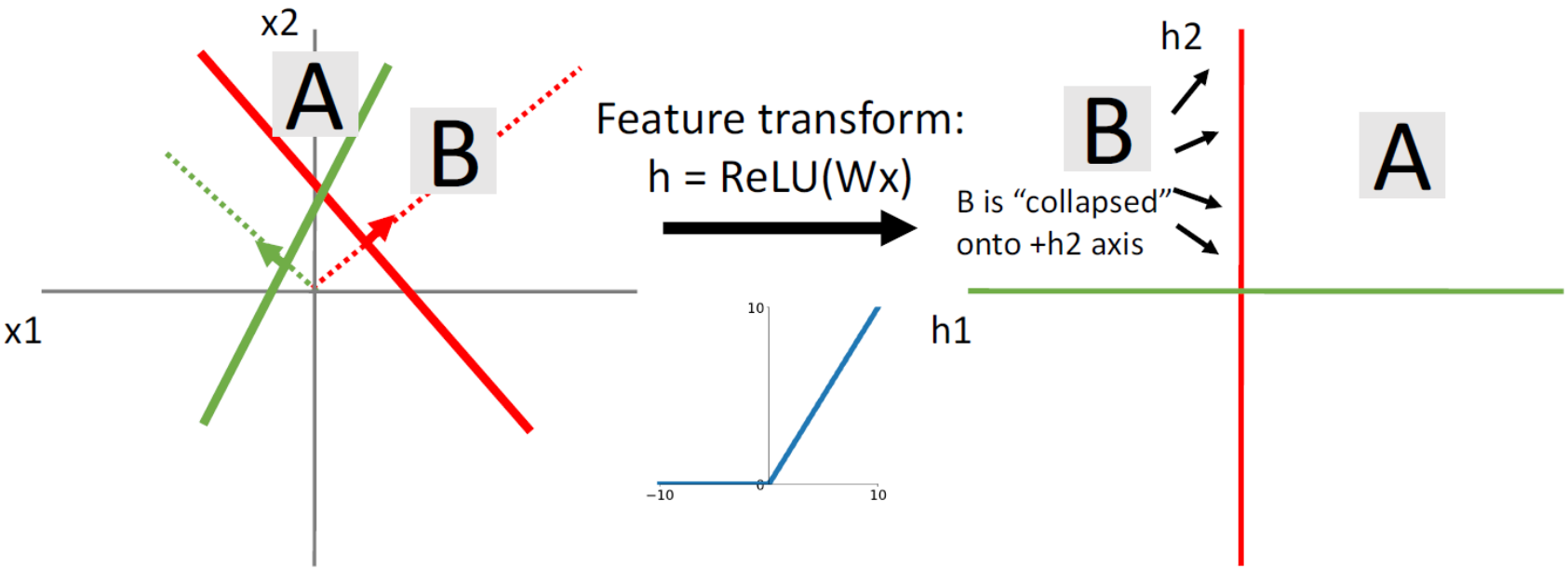
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W^T \boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.



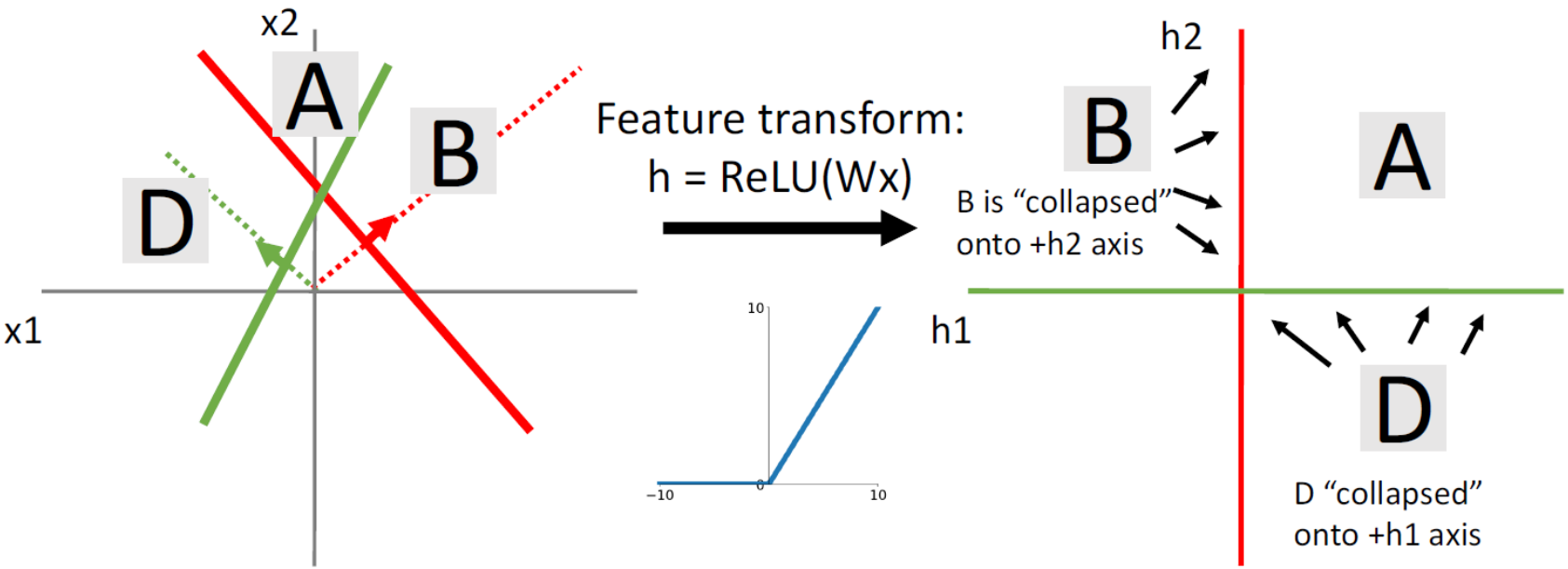
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W^T \boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.



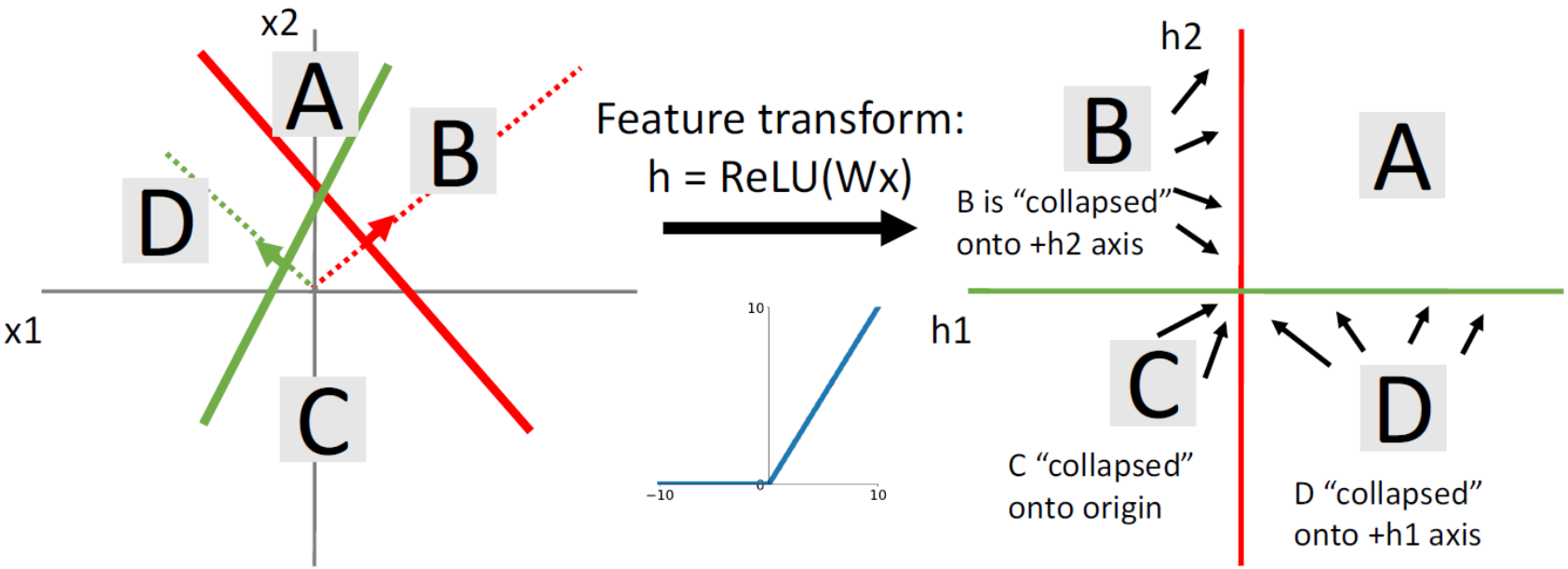
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W^T \boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.



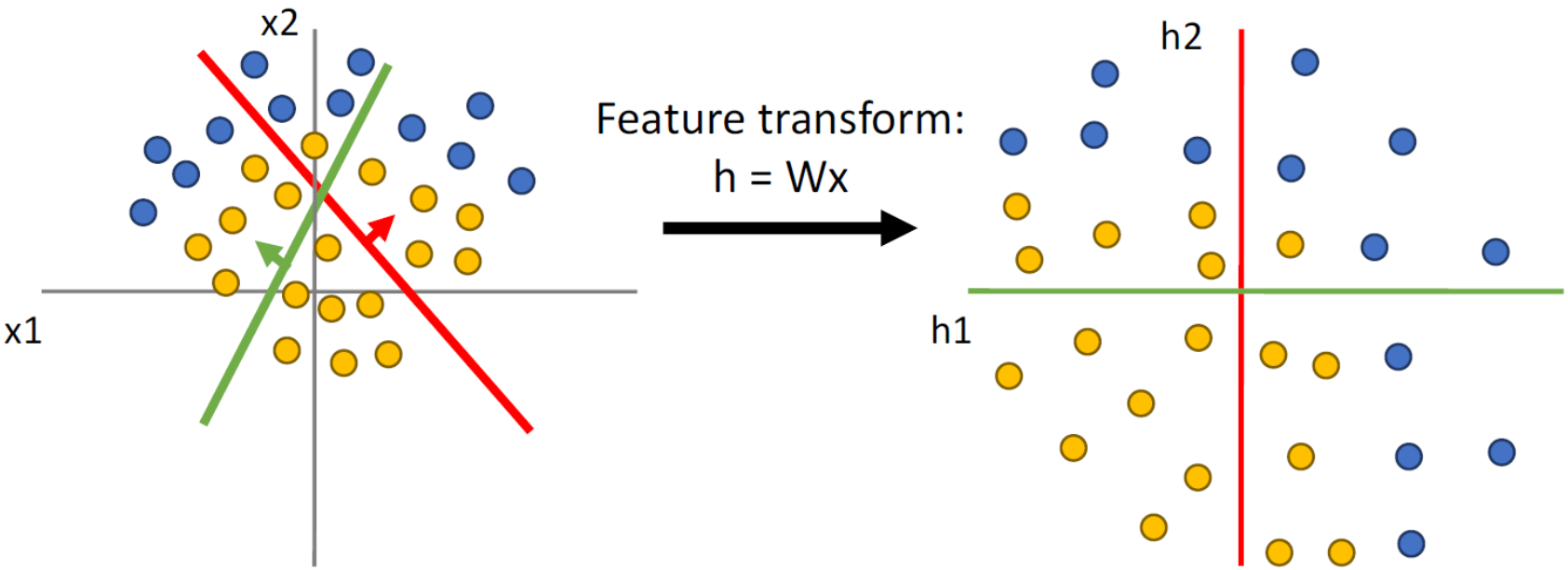
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W^T \boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.



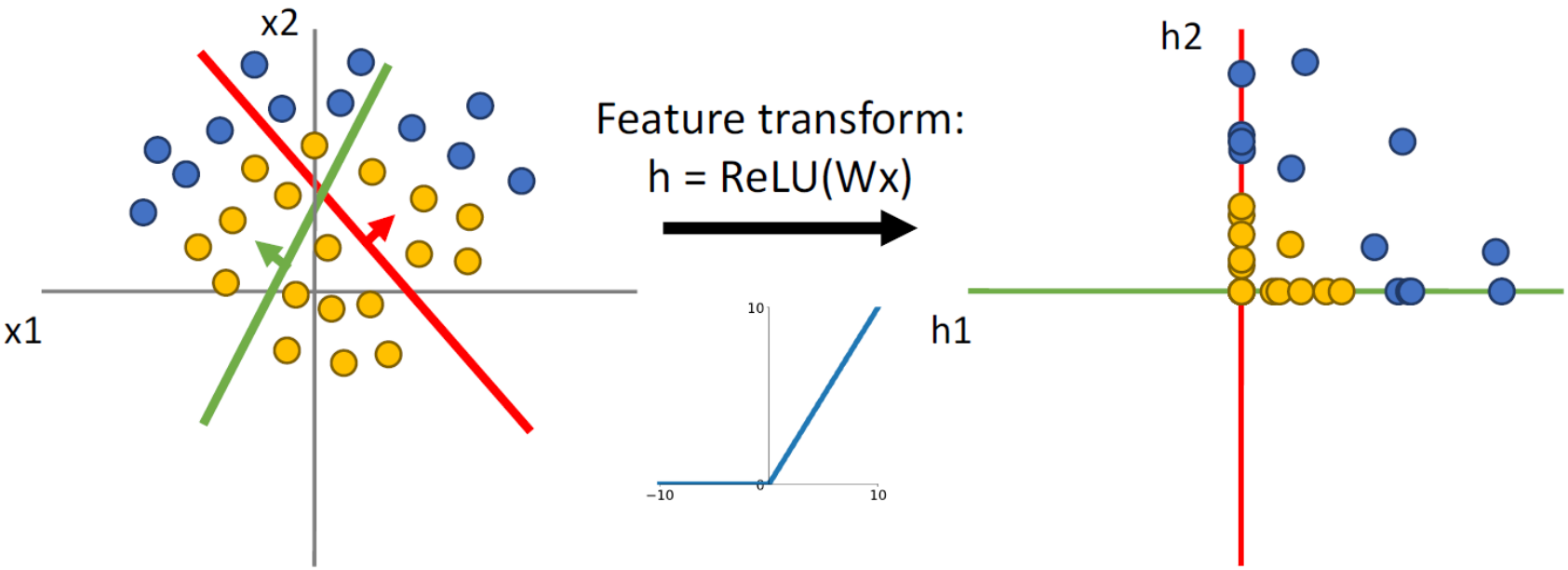
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W^T \boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.
- Points not linearly separable in original space



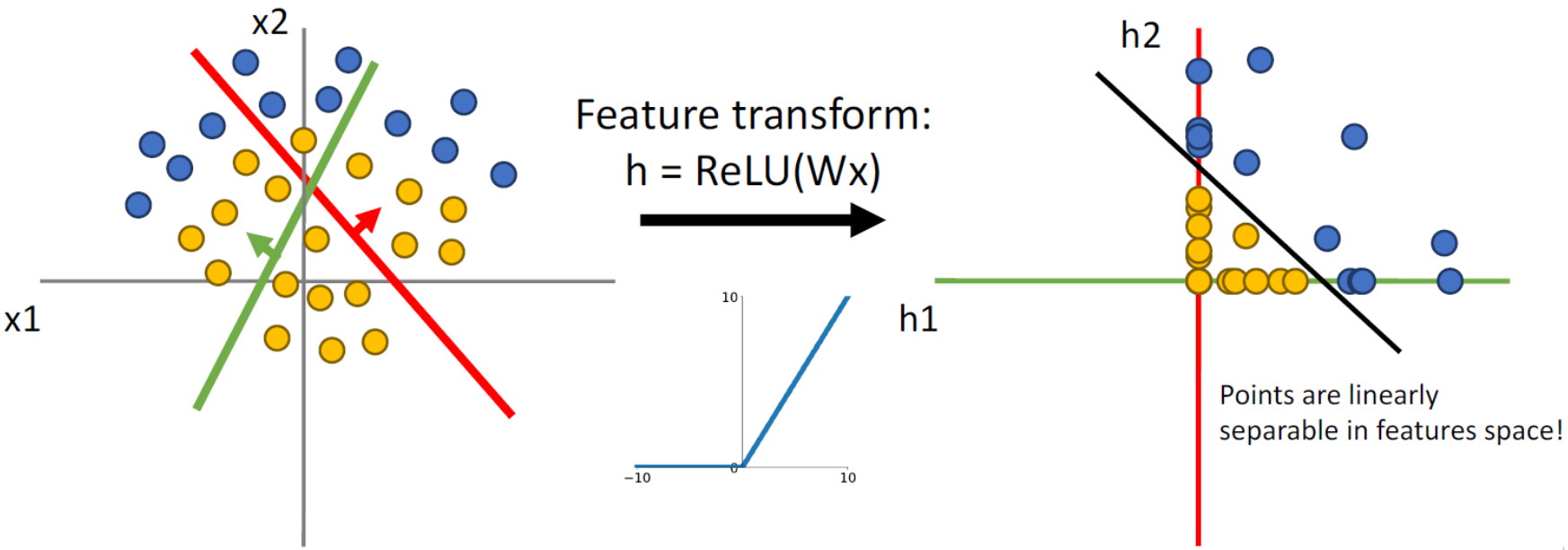
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W^T \boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.
- Points not linearly separable in original space



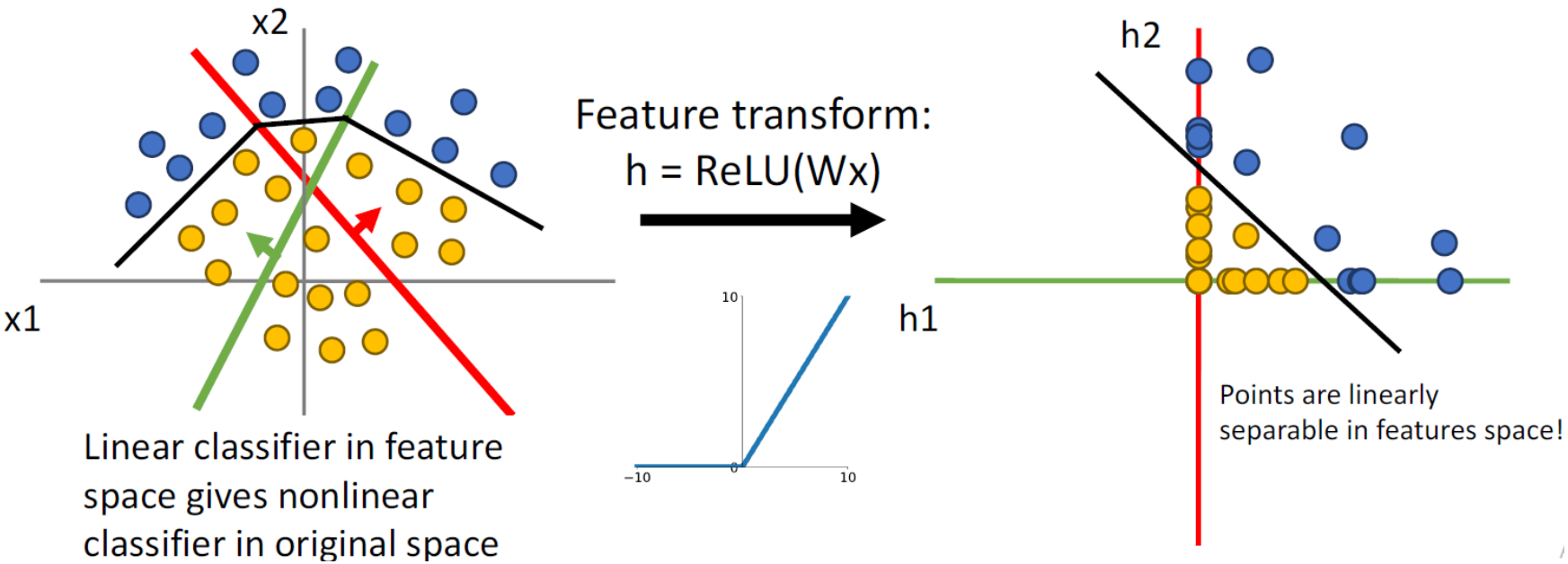
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W^T \boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.
- Points not linearly separable in original space

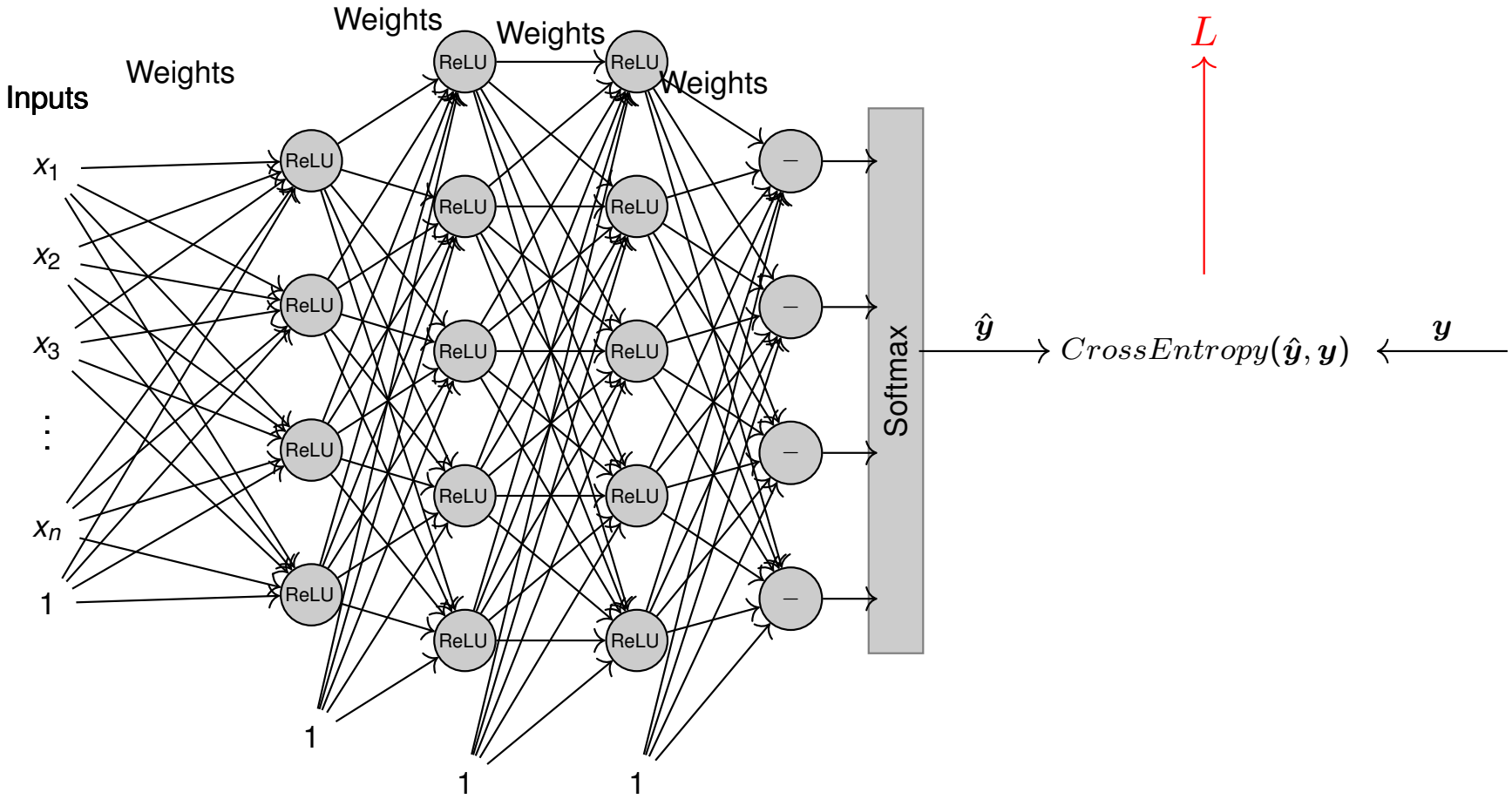


Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W^T \boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.
- Points not linearly separable in original space



Fully connected neural network loss



$$\hat{y} = Softmax(W^4(max(0, W^3(max(0, W^2(max(0, W^1 x + b^1)) + b^2)) + b^3)) + b^4)$$

Gradient descent

- The gradient descent method to find the minimum of a function iteratively

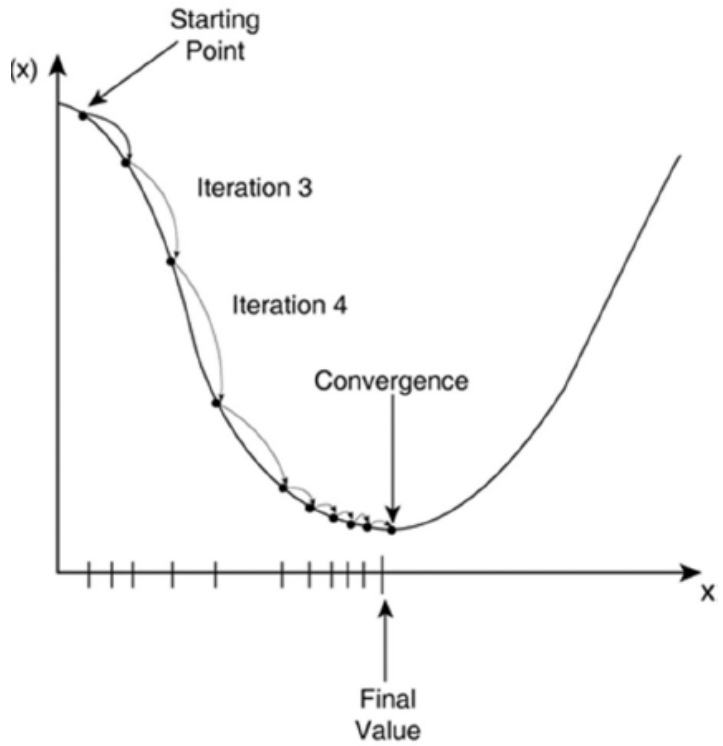
$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - \eta \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^t)$$

  - η is the "step size" (Also called "learning rate")

- The gradient descent algorithm converges when the following criterion is satisfied.

$$|f(\boldsymbol{x}^{t+k}) - f(\boldsymbol{x}^t)| < \epsilon$$

  - k is a hyperparameter.



Training neural nets through gradient descent

- Total training loss

$$L(f(X, W), \boldsymbol{y}) = \frac{1}{N} \sum_i L_i(f(\boldsymbol{x}^{(i)}, W), y^{(i)})$$

- Gradient descent algorithm:
  - Initialize all weights and biases $w_{ij}^{(\ell)}$
    - Using the extended notation: the bias is also a weight
  - Do:
    - For every layer ℓ for all i, j update:

$$w_{ij}^{(\ell)} = w_{ij}^{(\ell)} - \eta \frac{\partial L}{\partial w_{ij}^{(\ell)}}$$

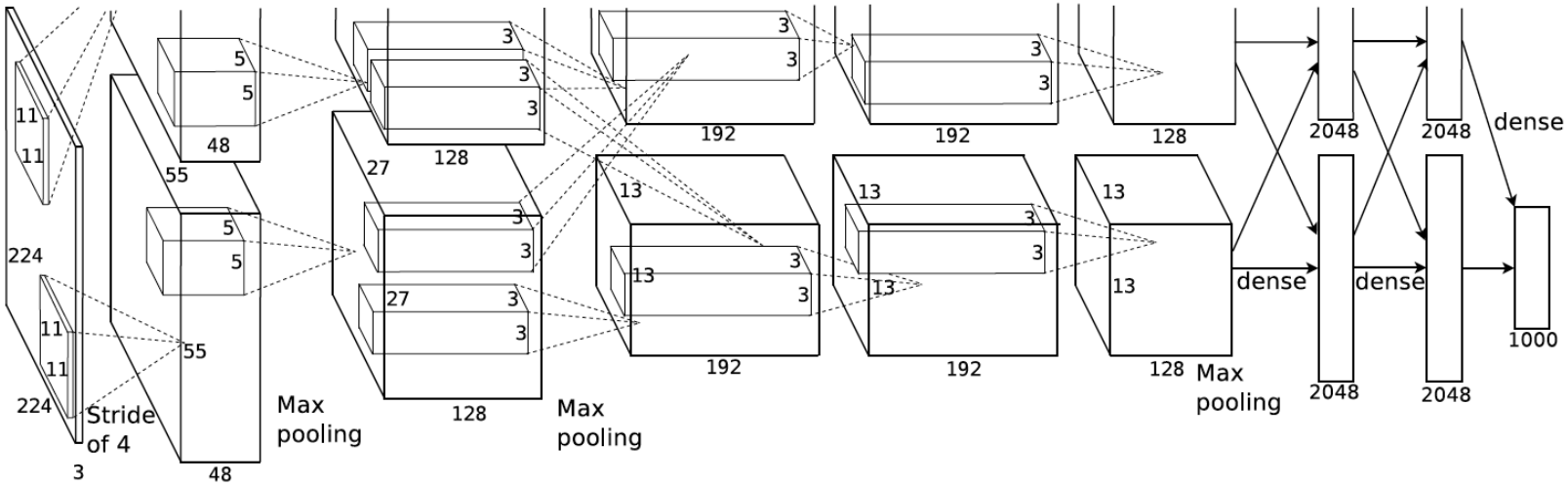
  - Until loss has converged

Idea: derive $\frac{\partial L}{\partial W}$ on paper

- Problems
  - Very tedious: Lots of matrix calculus, need lots of paper
  - What if we want to change loss? Need to re-derive from scratch. Not modular!
  - Not feasible for very complex models!

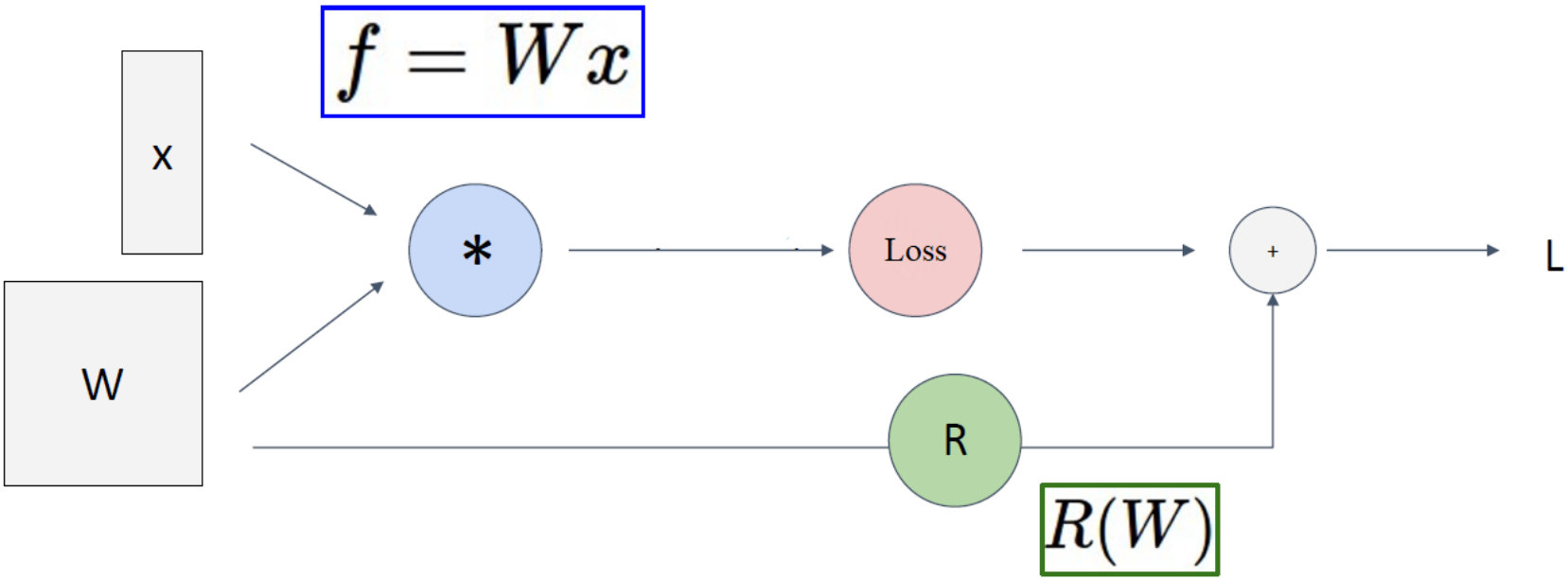
Idea: derive $\frac{\partial L}{\partial W}$ on paper

- Problems
  - Very tedious: Lots of matrix calculus, need lots of paper
  - What if we want to change loss? Need to re-derive from scratch. Not modular!
  - Not feasible for very complex models!



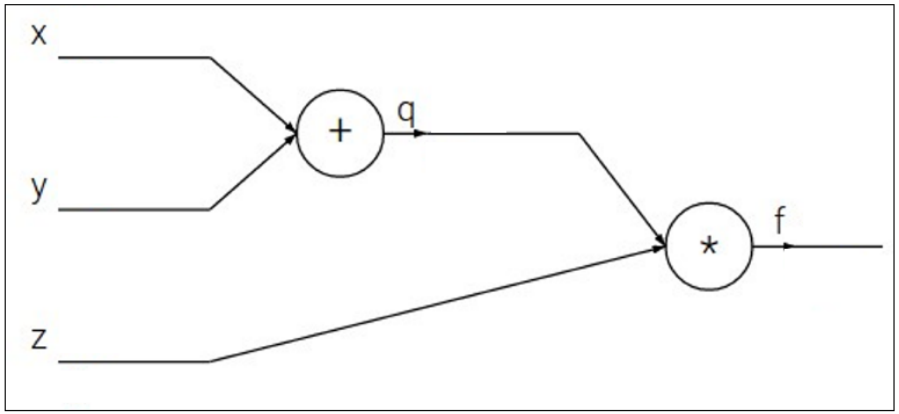
AlexNet has about 660K units, 61M parameters,

Better Idea: Computational Graphs



Backpropagation: simple example

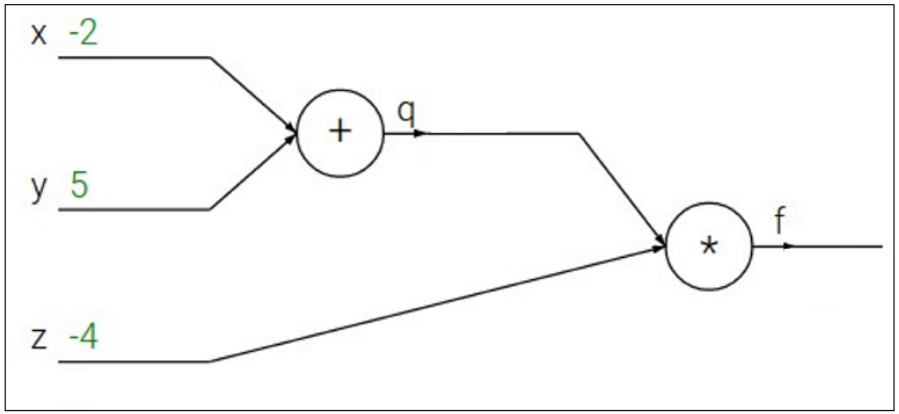
$$f(x, y, z) = (x + y)z$$



Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

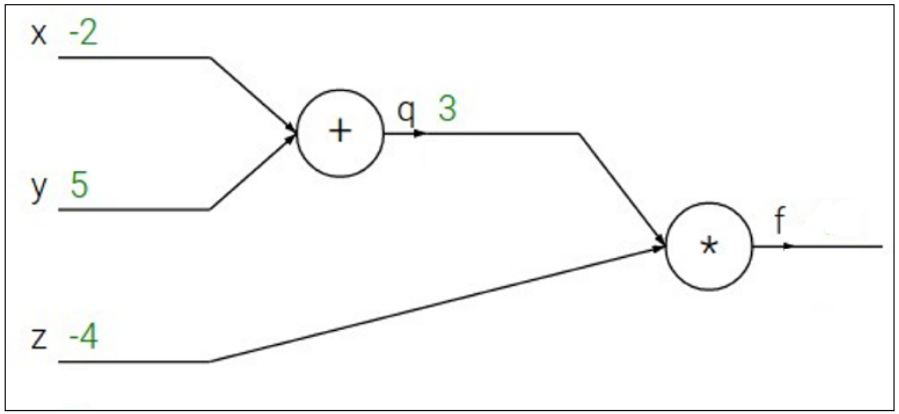
e.g. x = -2, y = 5, z = -4



Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



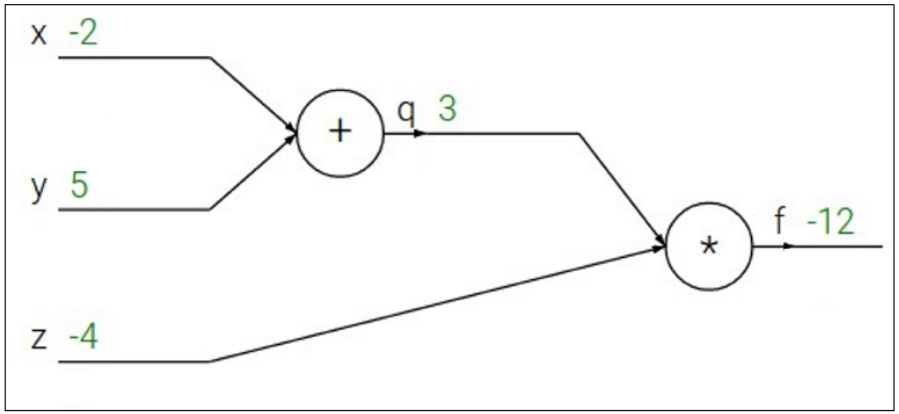
1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



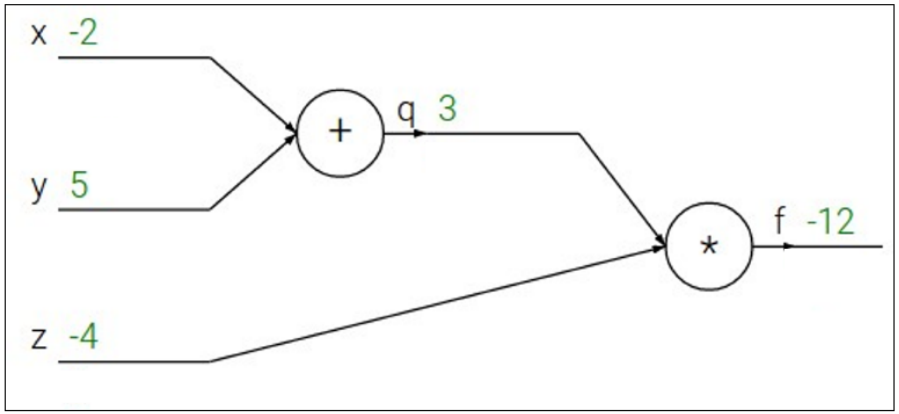
1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

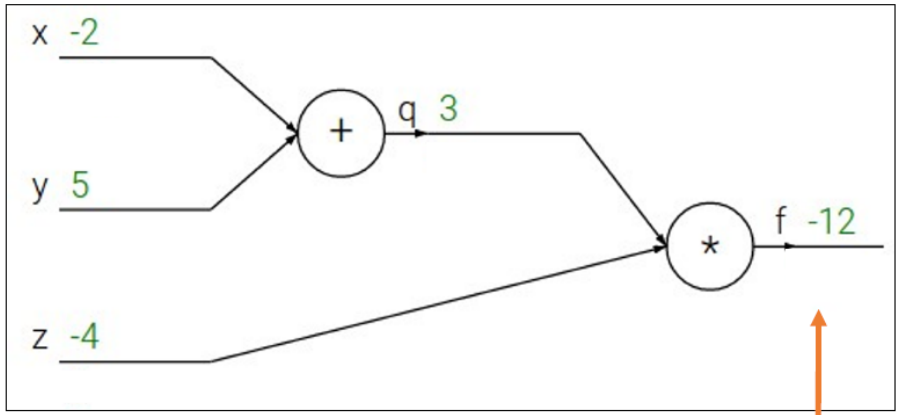
2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

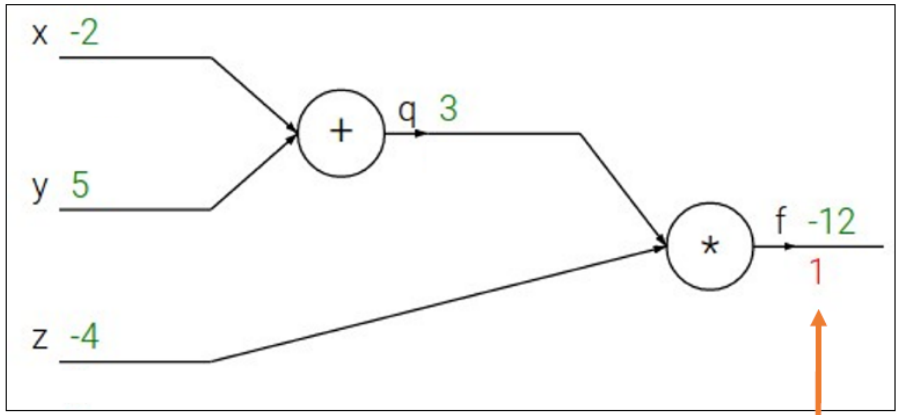
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\frac{\partial f}{\partial f}$$

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

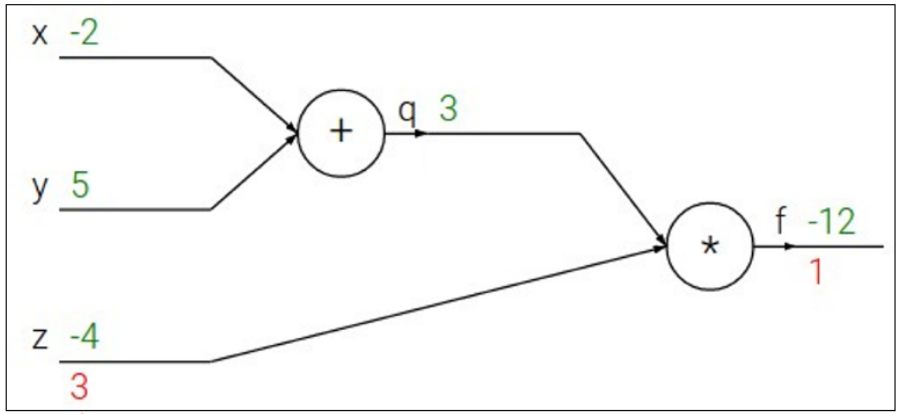
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\frac{\partial f}{\partial f}$$

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

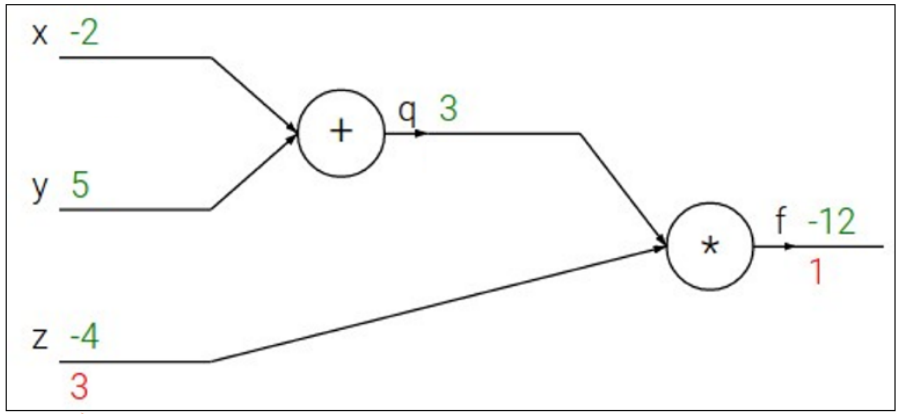
$$\frac{\partial f}{\partial z}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

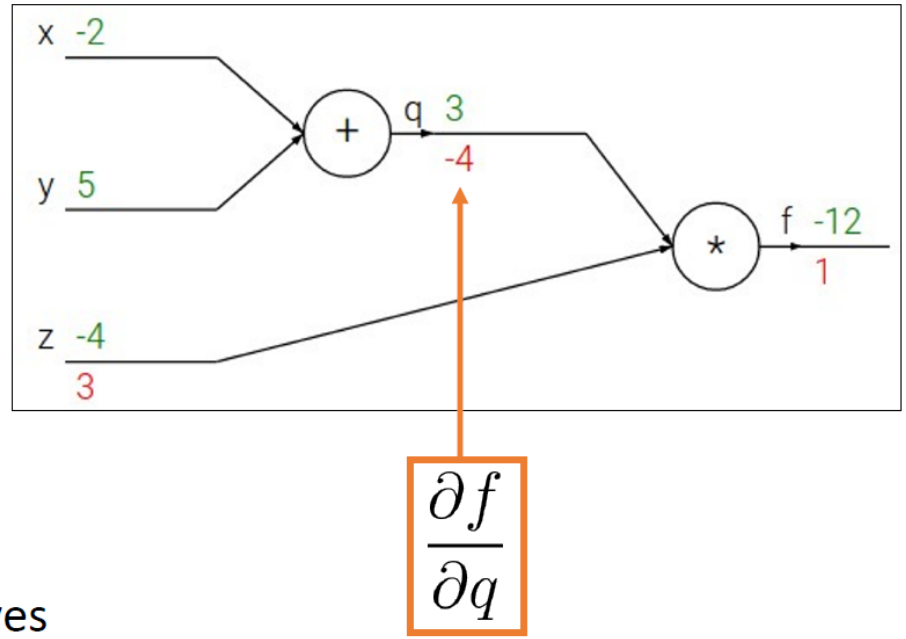
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\frac{\partial f}{\partial z} = q$$

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

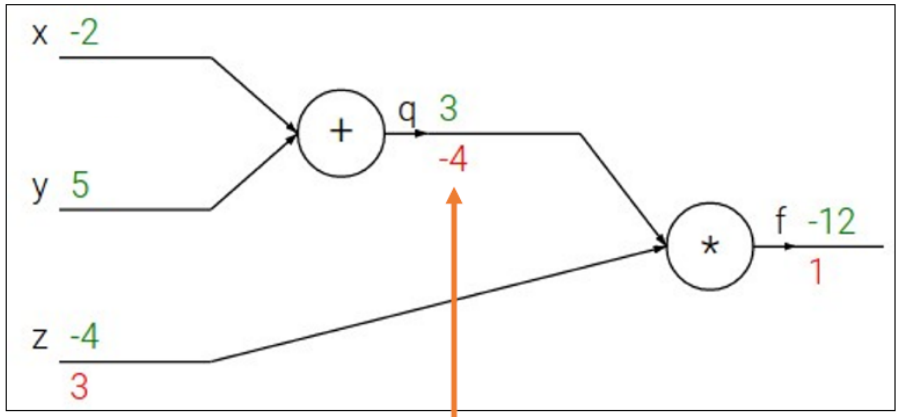
2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

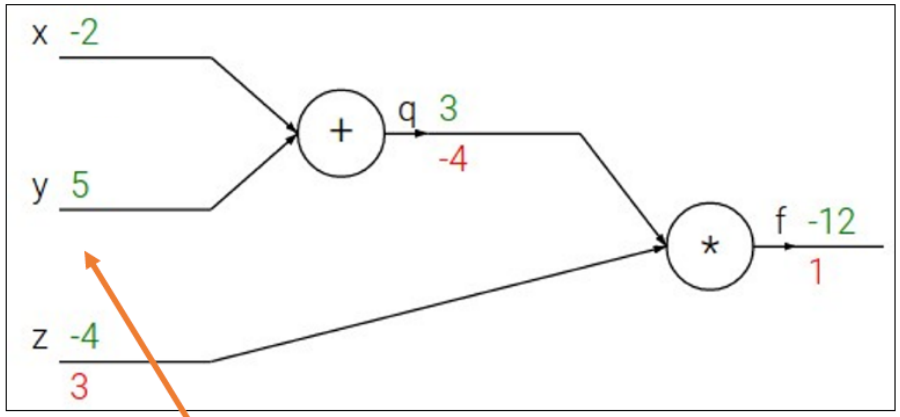
$$\frac{\partial f}{\partial q} = z$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

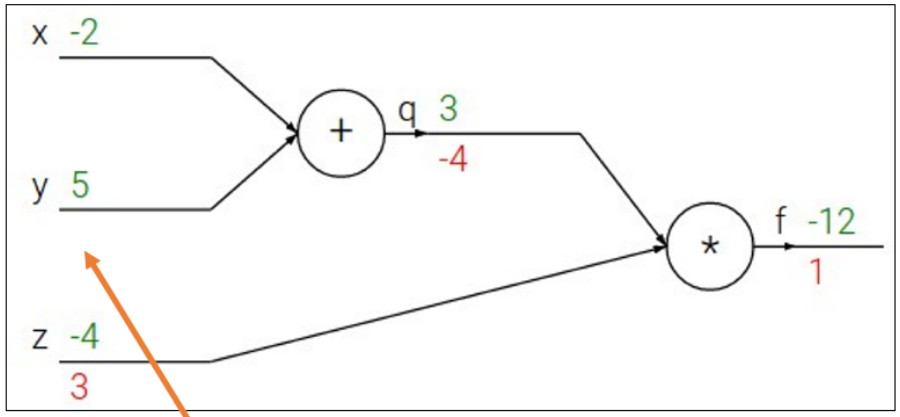
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$\frac{\partial f}{\partial y}$

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

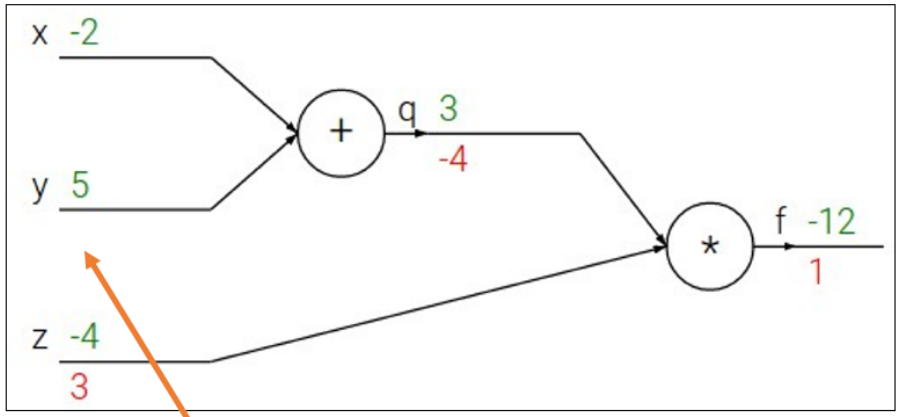
Chain Rule

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$$

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Chain Rule

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$$

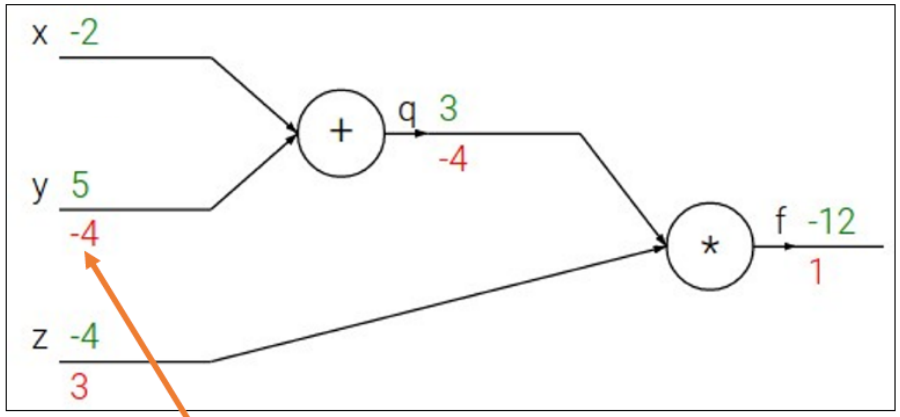
$$\frac{\partial q}{\partial y} = 1$$

Downstream Gradient **Local Gradient** **Upstream Gradient**

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Chain Rule

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$$

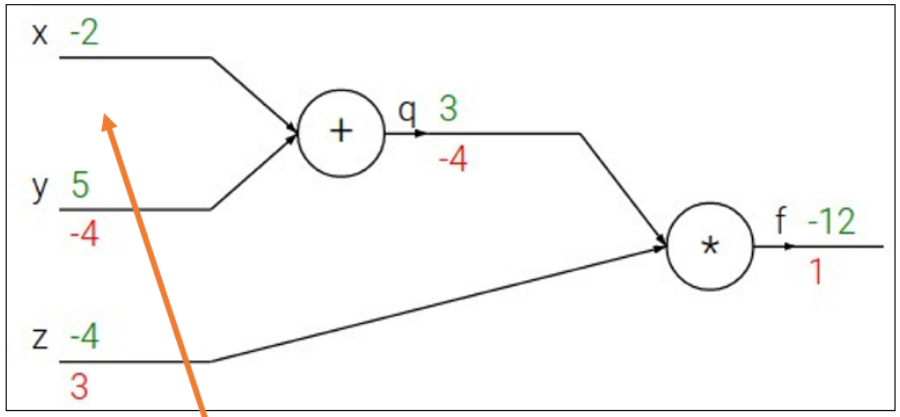
$$\frac{\partial q}{\partial y} = 1$$

Downstream Gradient **Local Gradient** **Upstream Gradient**

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}$$

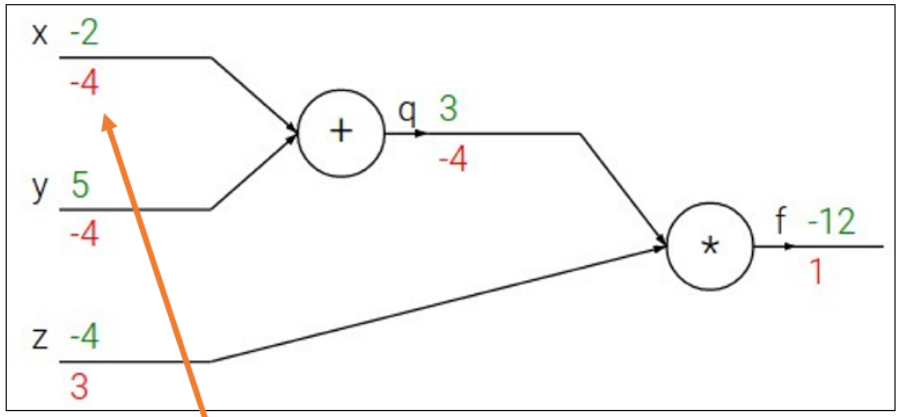
$$\frac{\partial q}{\partial x} = 1$$

Downstream Gradient **Local Gradient** **Upstream Gradient**

Backpropagation: simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



1. Forward pass: Compute outputs

$$q = x + y \qquad f = qz$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

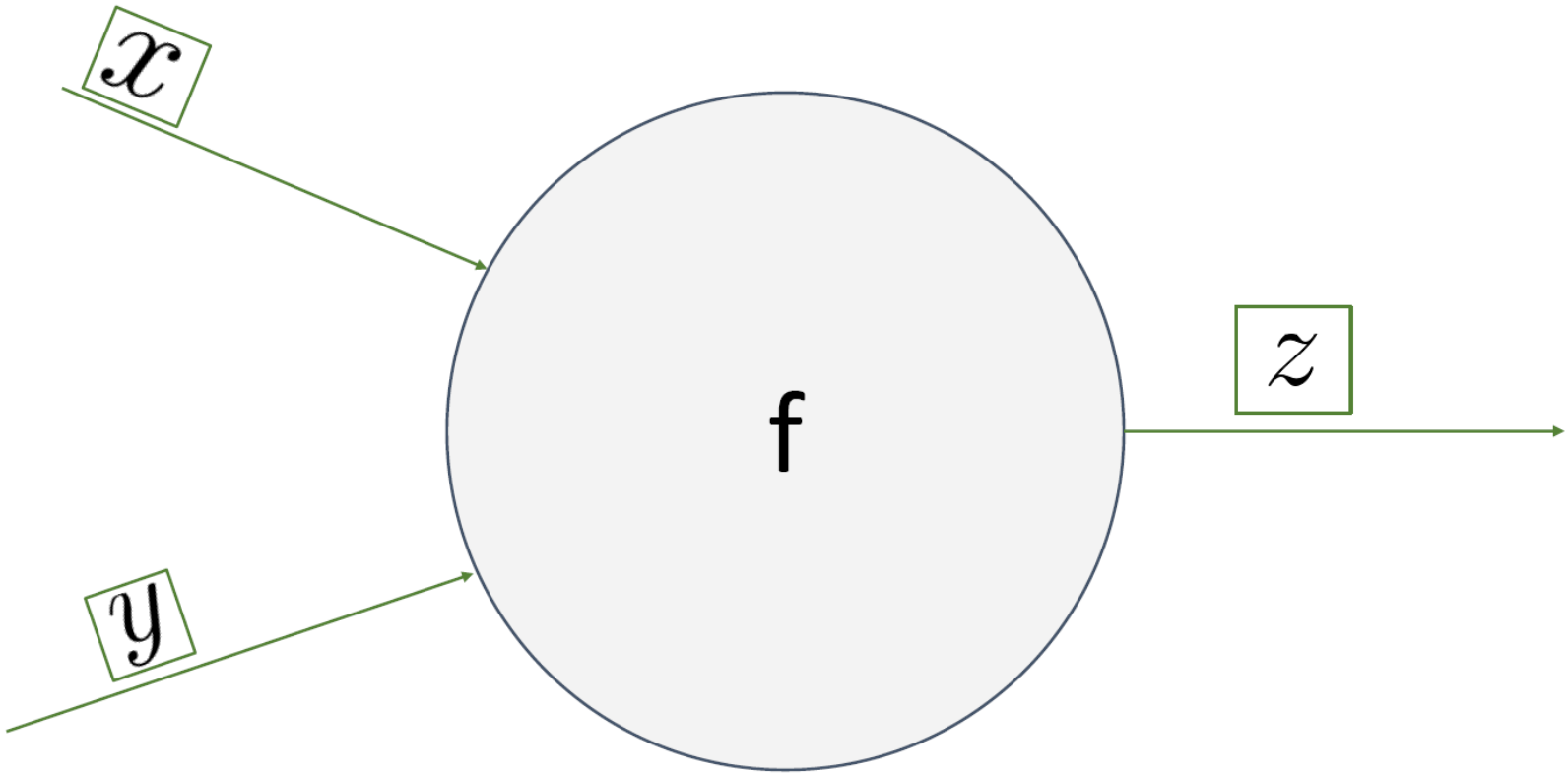
Chain Rule

$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}$$

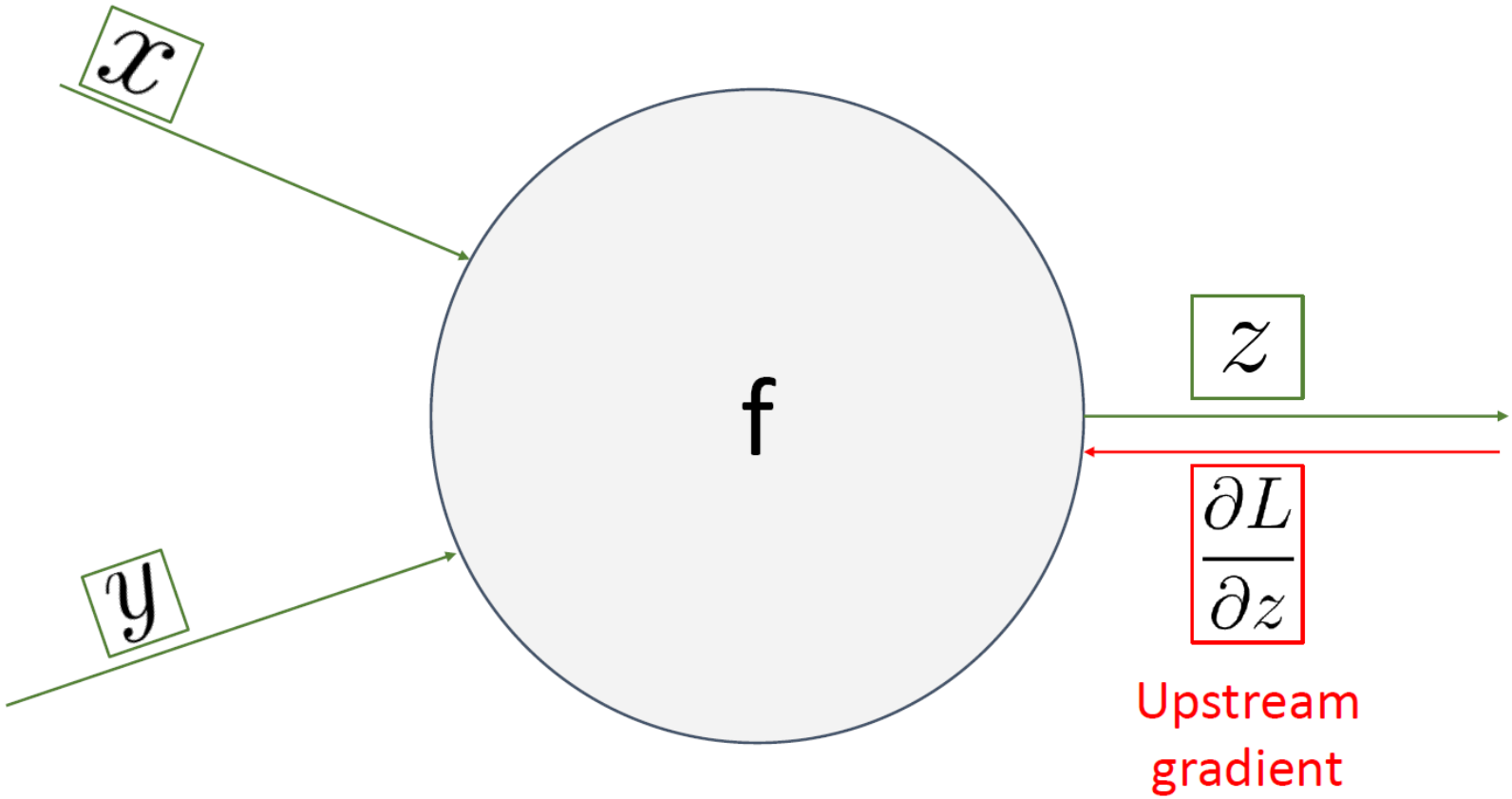
$$\frac{\partial q}{\partial x} = 1$$

Downstream Gradient **Local Gradient** **Upstream Gradient**

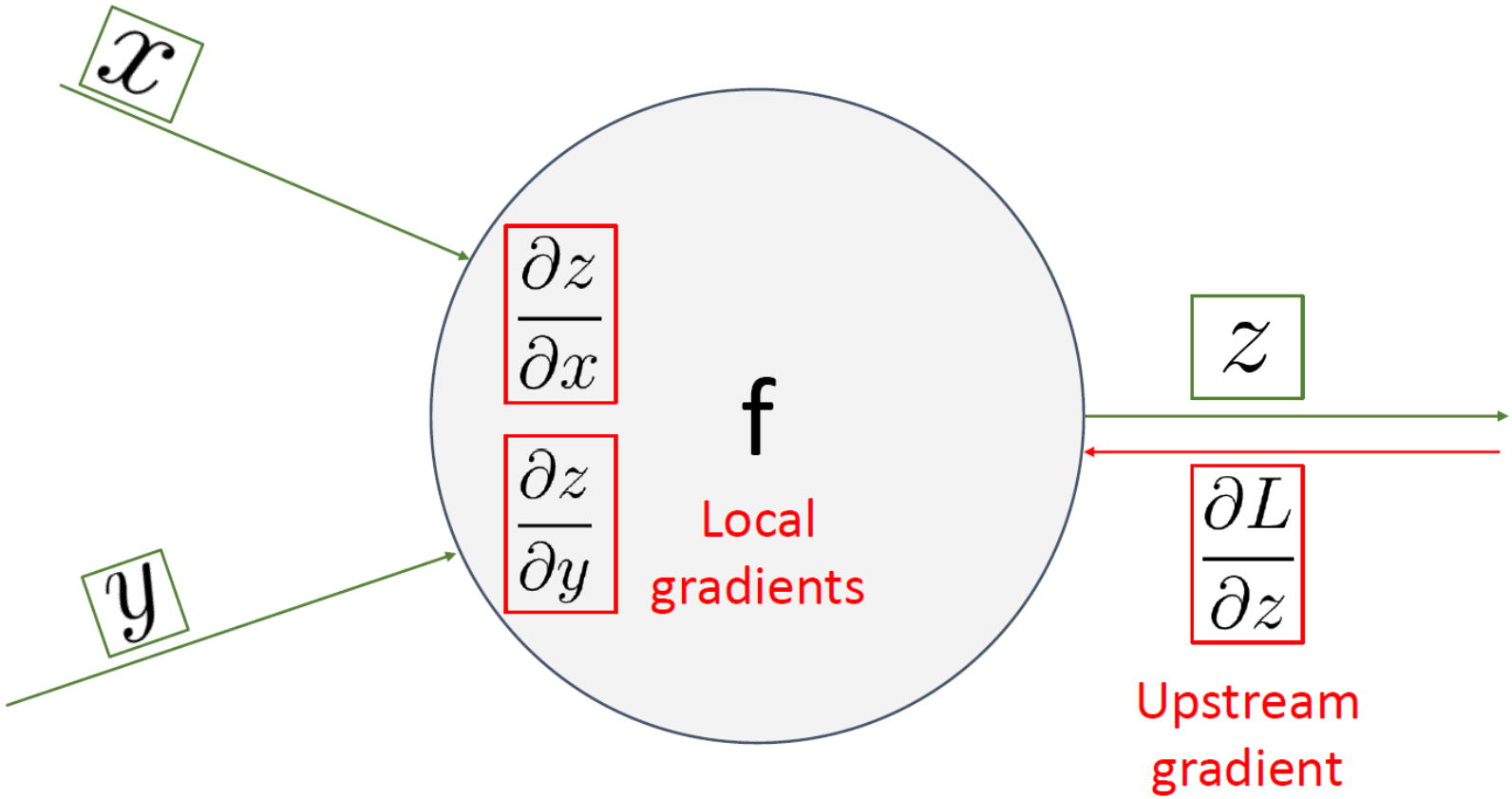
Up/Down stream gradients



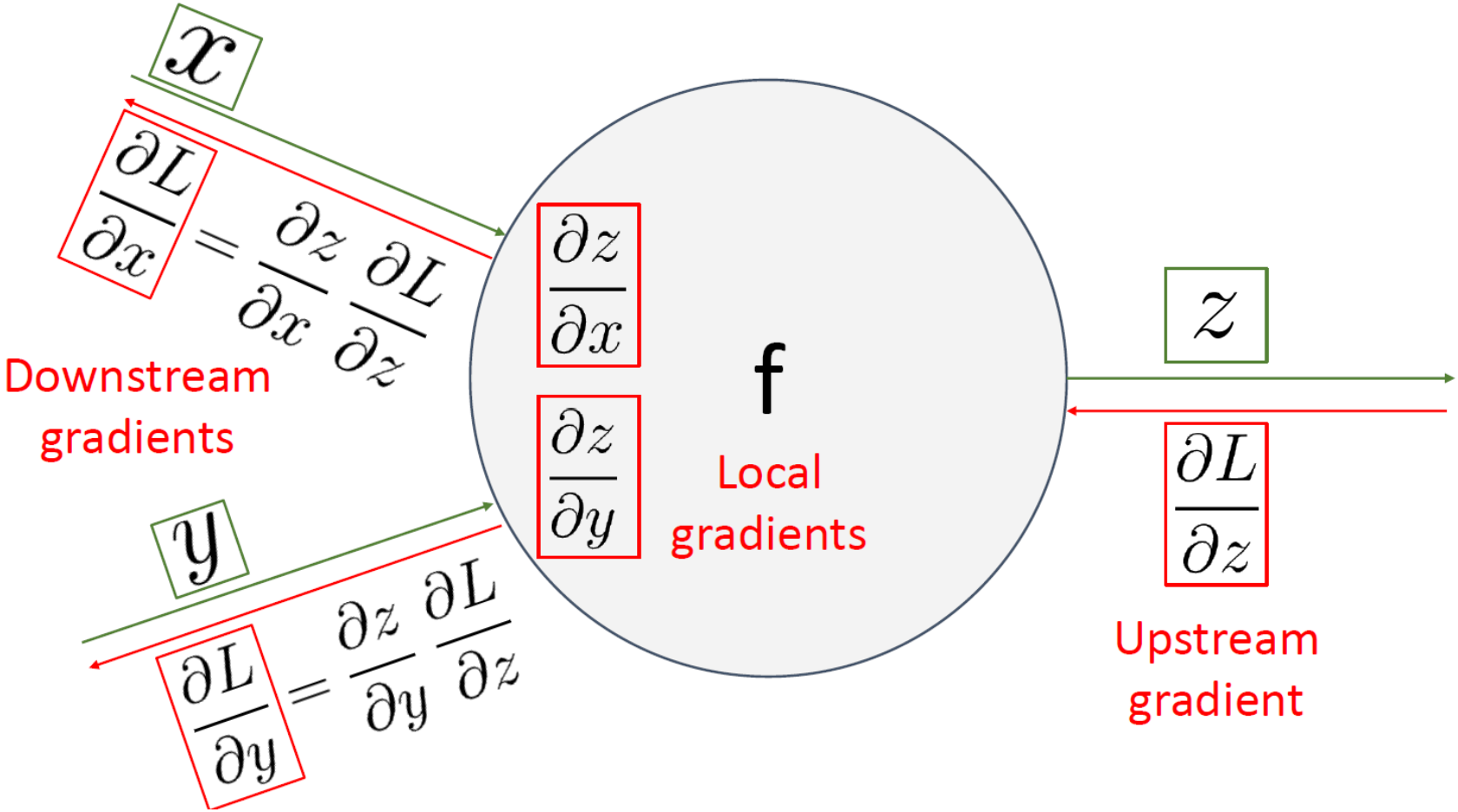
Up/Down stream gradients



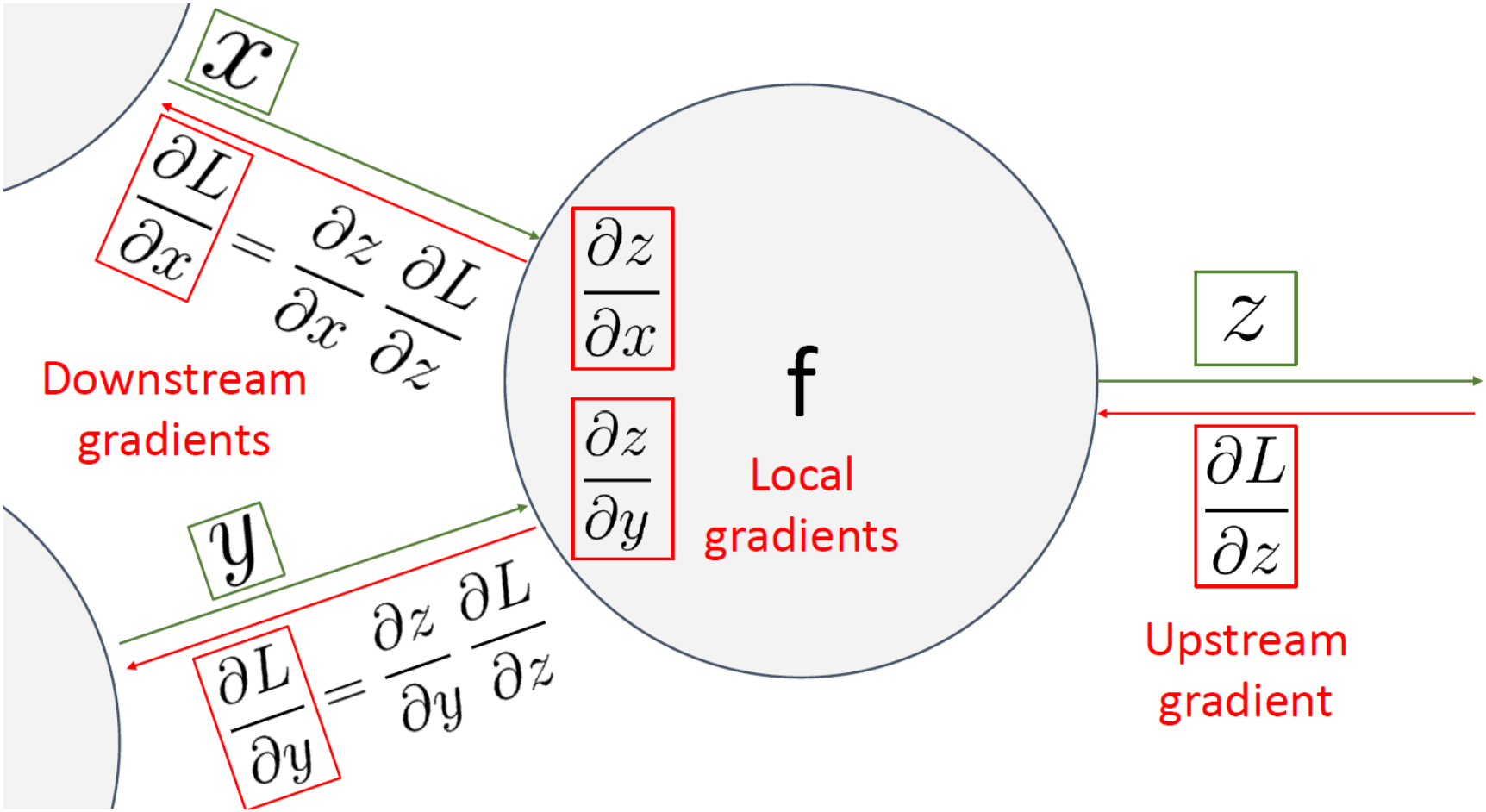
Up/Down stream gradients



Up/Down stream gradients

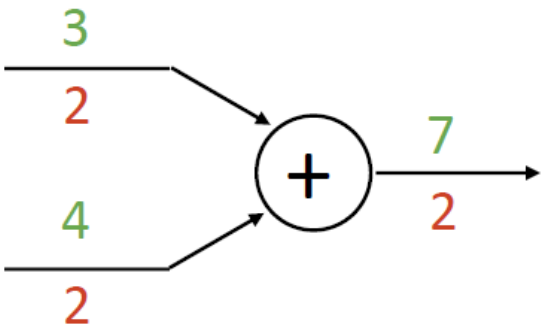


Up/Down stream gradients



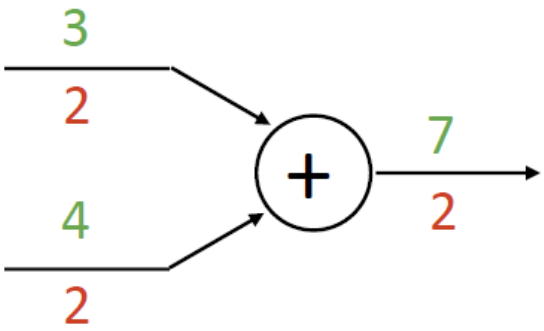
Patterns in Gradient Flow

add gate: gradient distributor

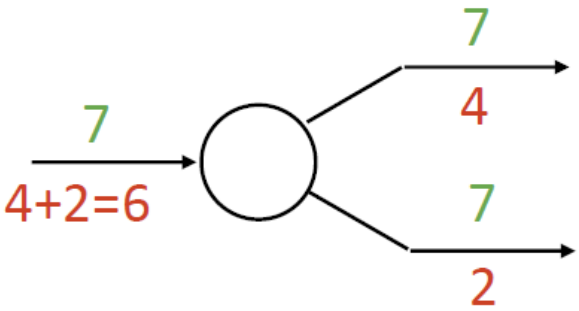


Patterns in Gradient Flow

add gate: gradient distributor

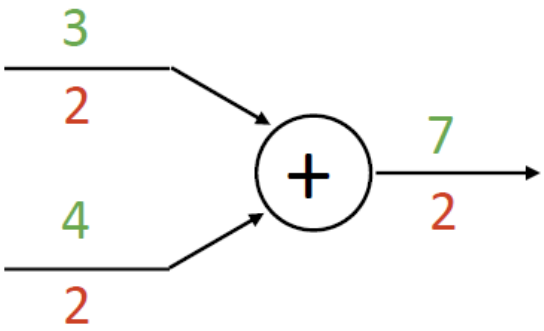


copy gate: gradient adder

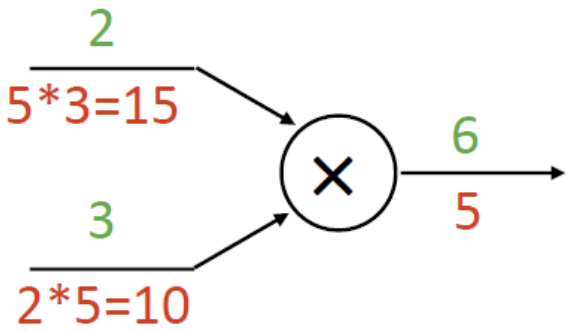


Patterns in Gradient Flow

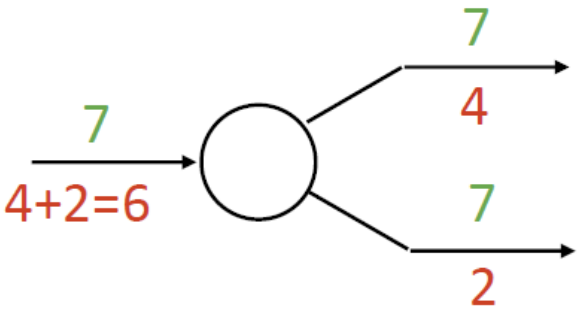
add gate: gradient distributor



mul gate: "swap multiplier"

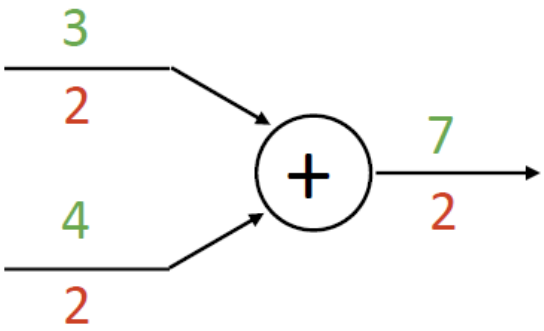


copy gate: gradient adder

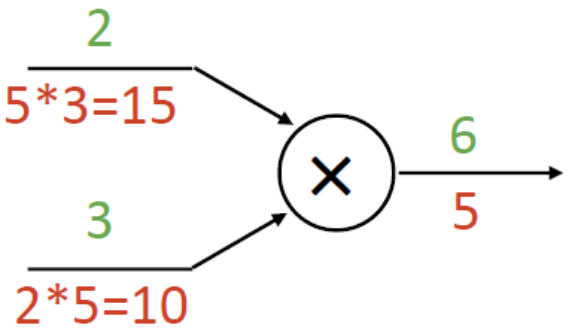


Patterns in Gradient Flow

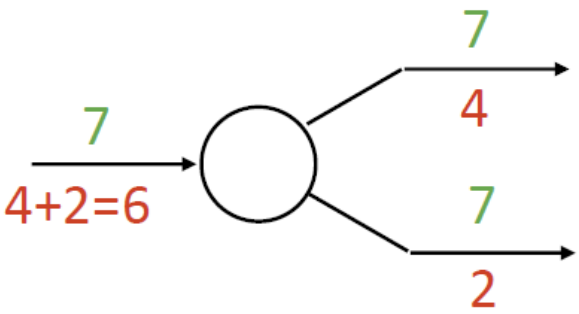
add gate: gradient distributor



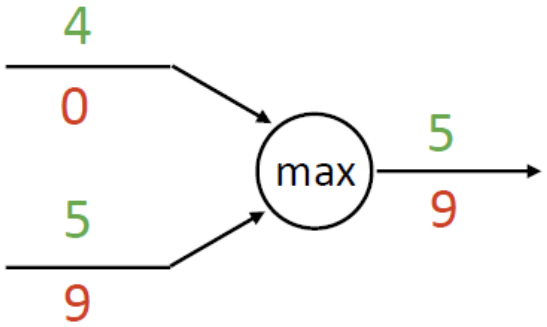
mul gate: "swap multiplier"



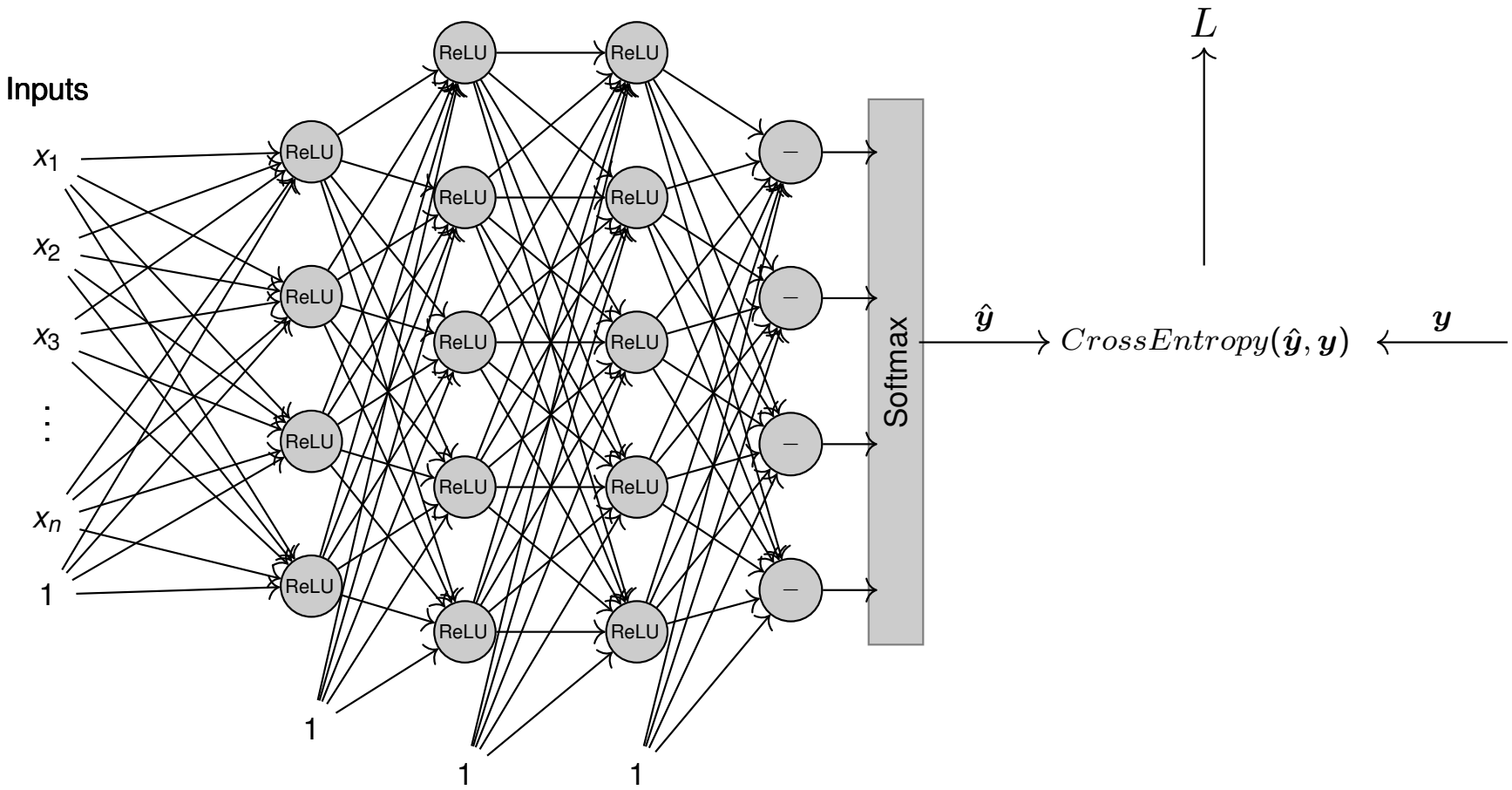
copy gate: gradient adder



max gate: gradient router

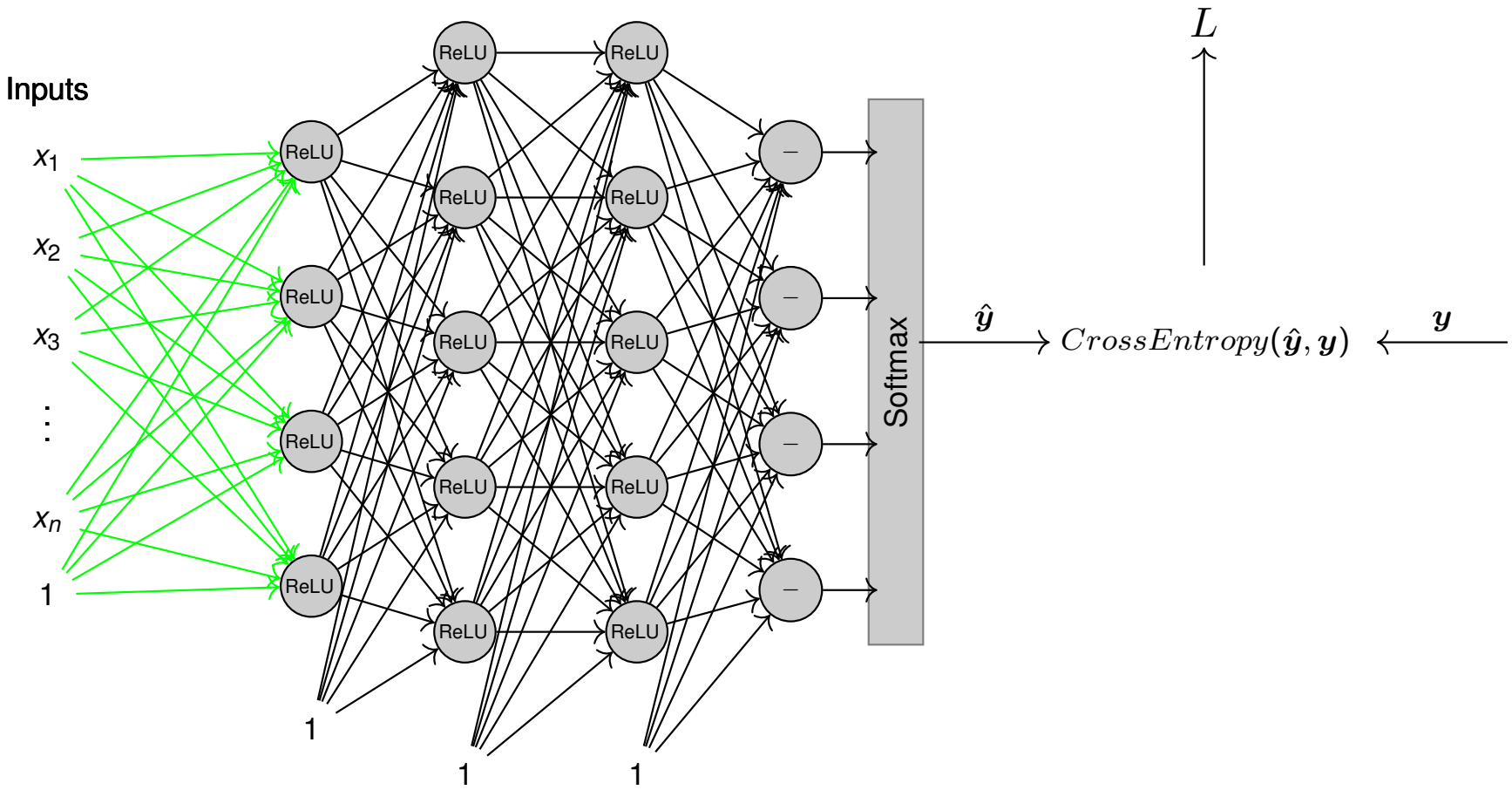


Fully connected neural network - Forward Pass



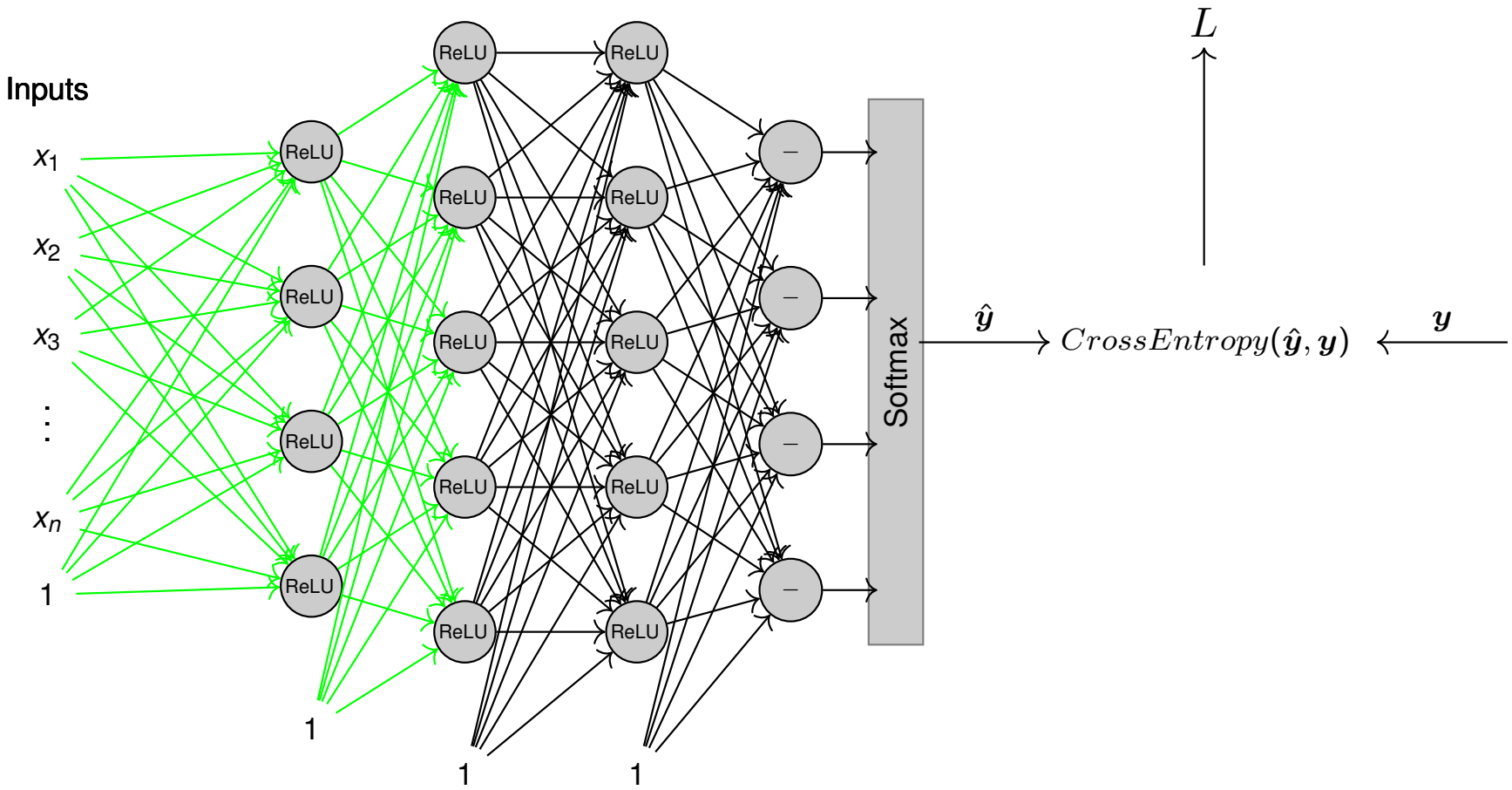
$$\hat{\boldsymbol{y}} = Softmax(W^4(max(0, W^3(max(0, W^2(max(0, W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

Fully connected neural network - Forward Pass



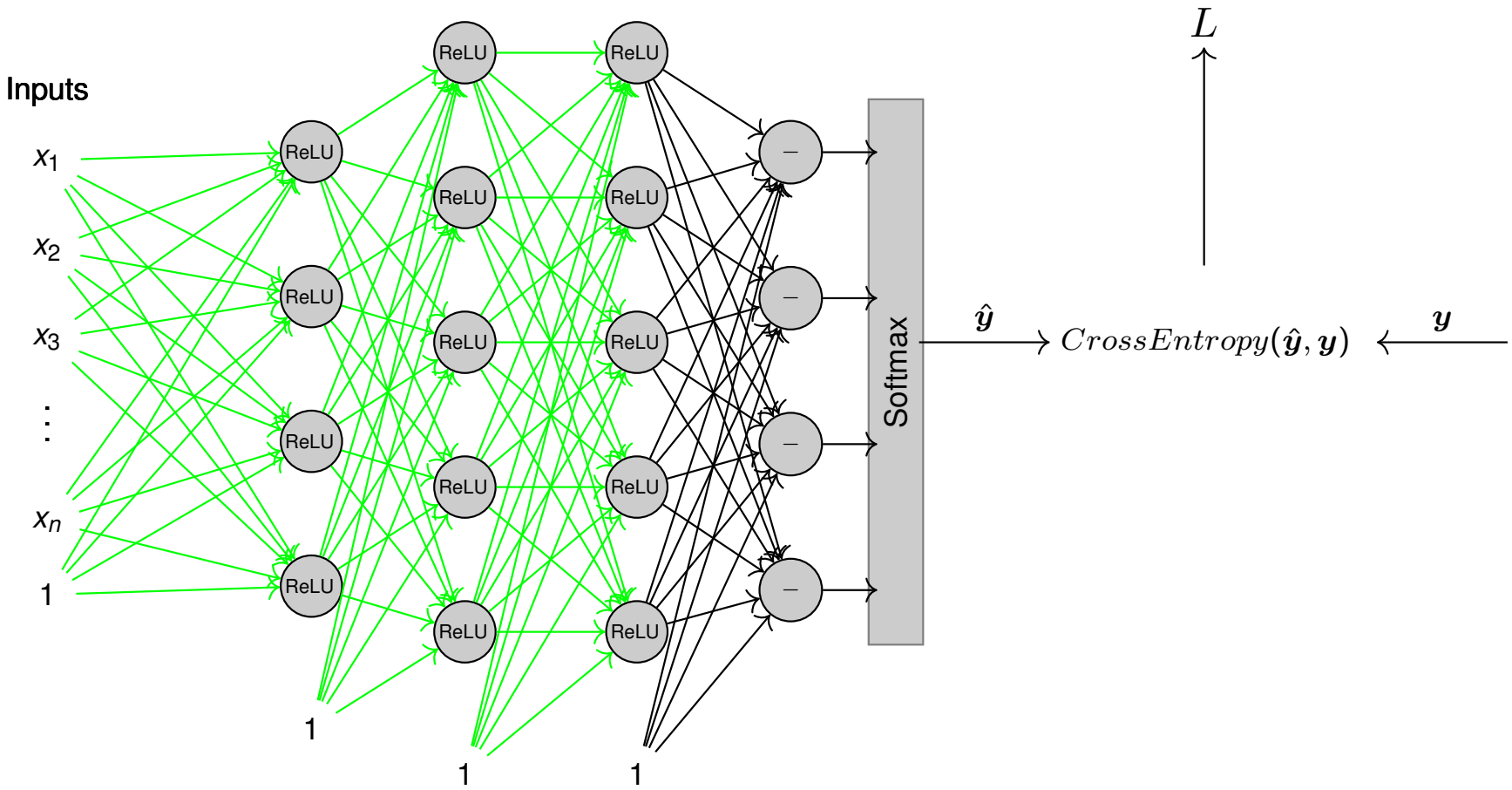
$$z^1 = W^1 x + b^1, \quad o^1 = ReLU(z^1)$$

Fully connected neural network - Forward Pass



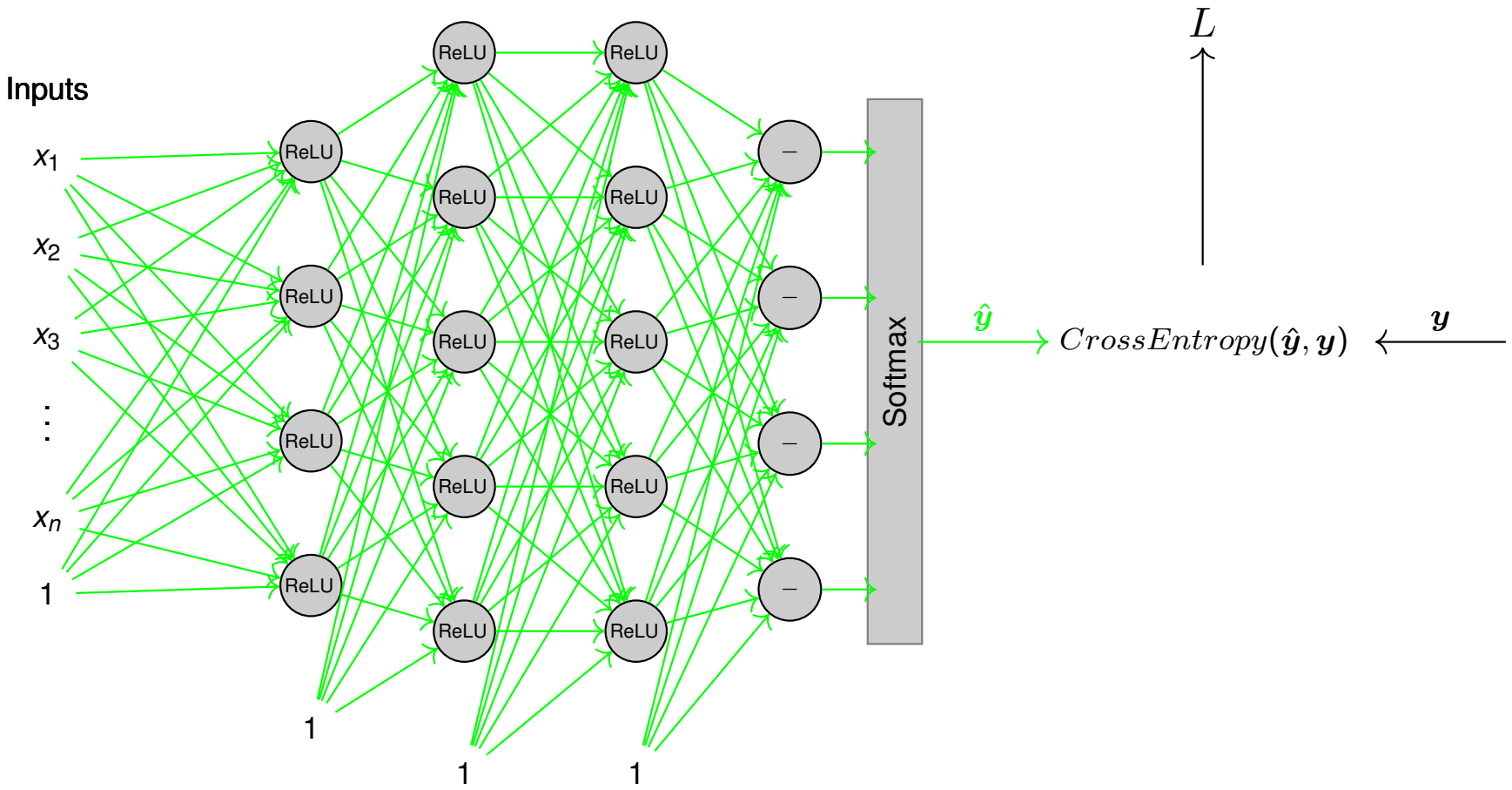
$$z^2 = W^2 o^1 + b^2, \quad o^2 = ReLU(z^2)$$

Fully connected neural network - Forward Pass



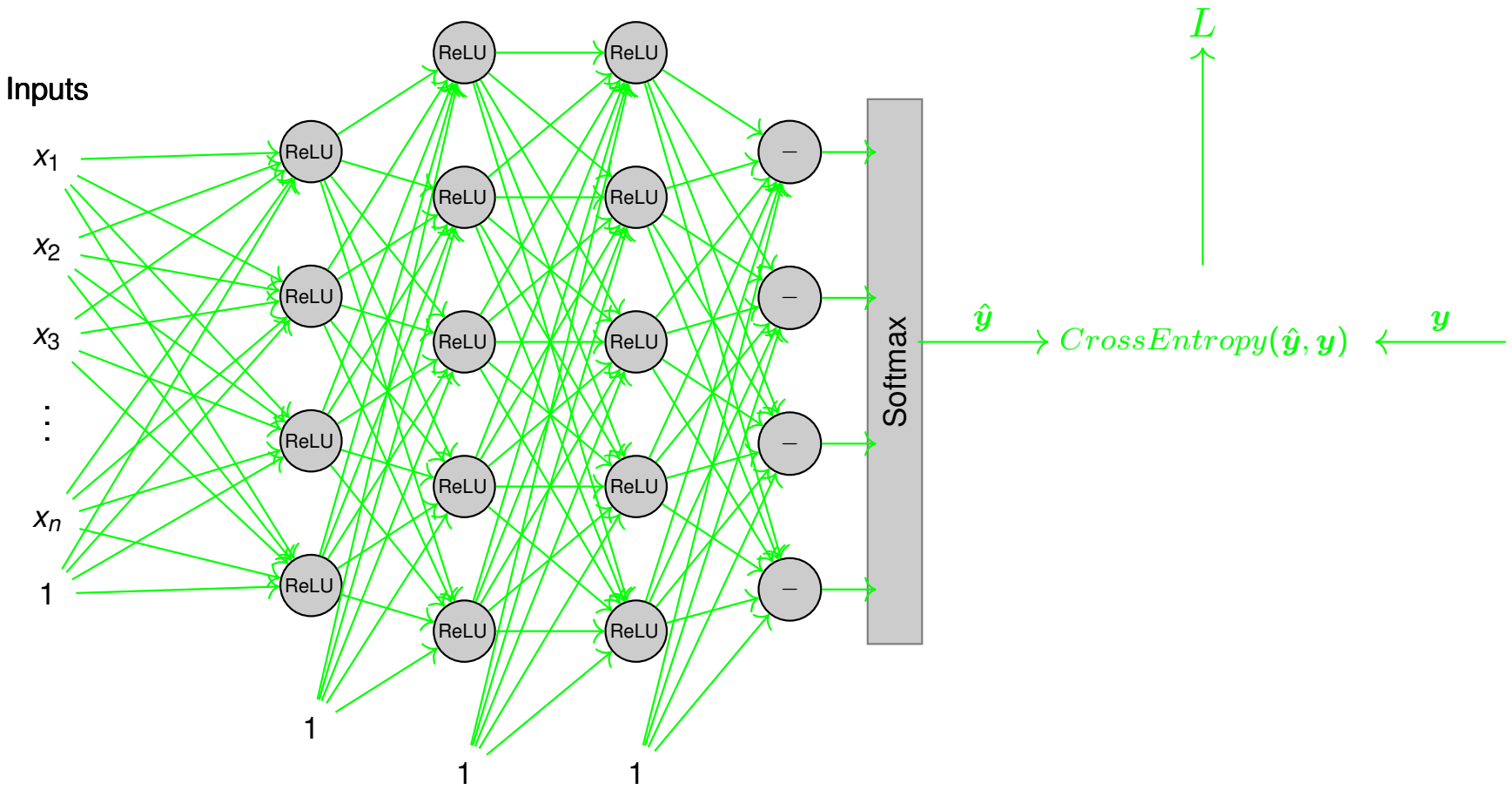
$$z^3 = W^3 o^2 + b^3, \quad o^3 = ReLU(z^3)$$

Fully connected neural network - Forward Pass



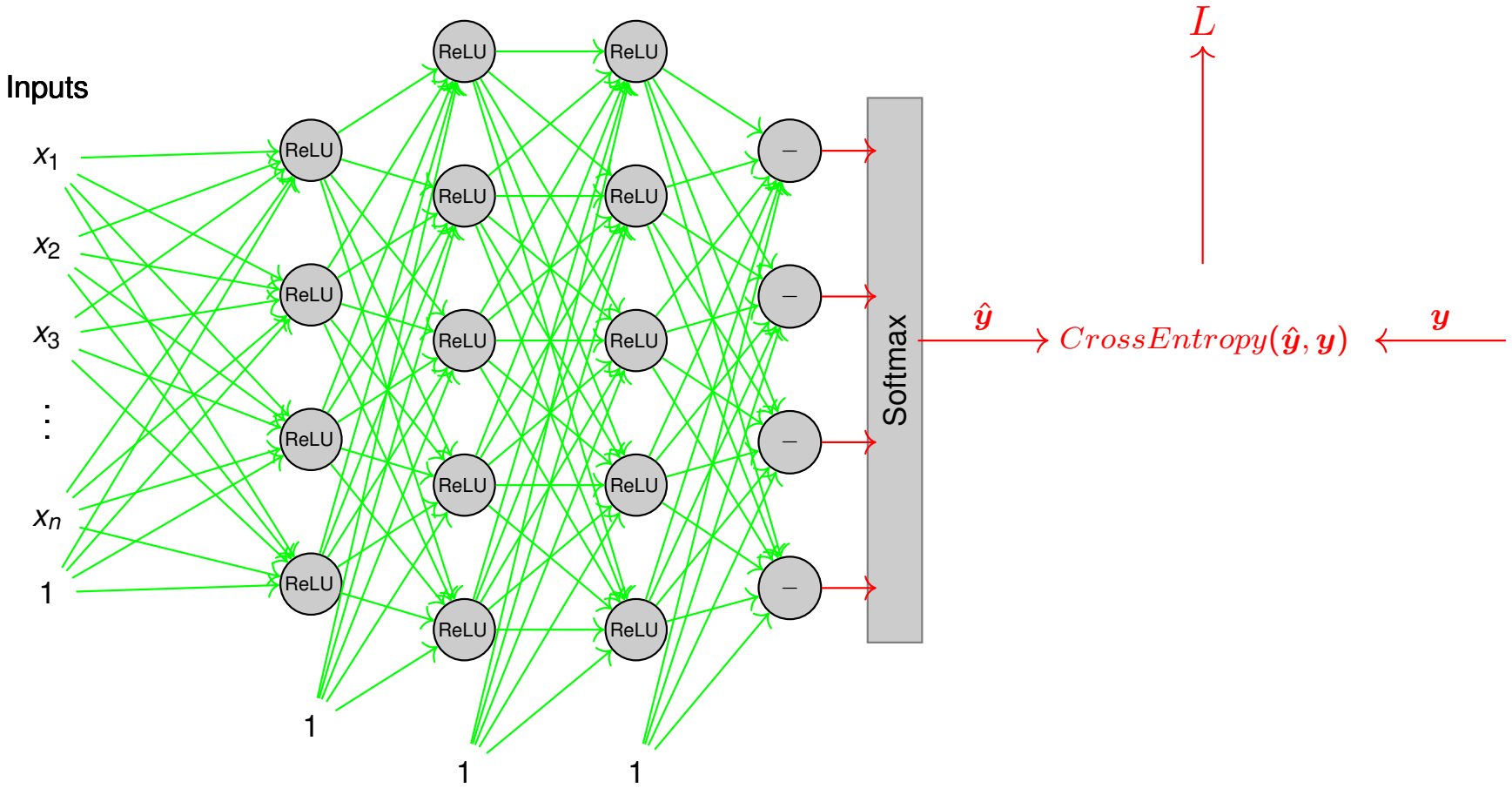
$$z^4 = W^4 o^3 + b^4, \quad \hat{y} = softmax(z^4)$$

Fully connected neural network - Forward Pass



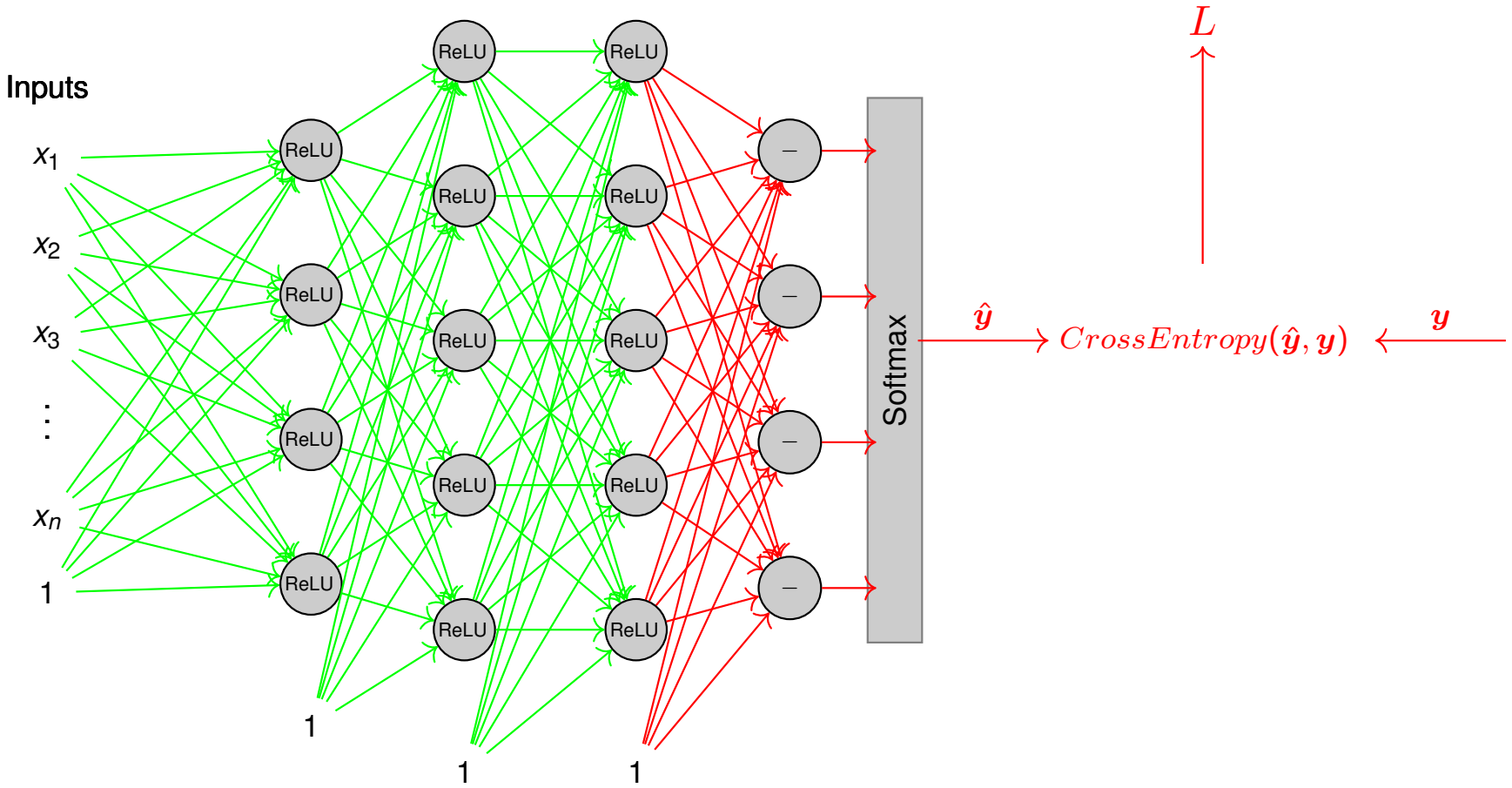
$$L = CrossEntropy(\hat{\boldsymbol{y}}, \boldsymbol{y})$$

Fully connected neural network - Backward Pass



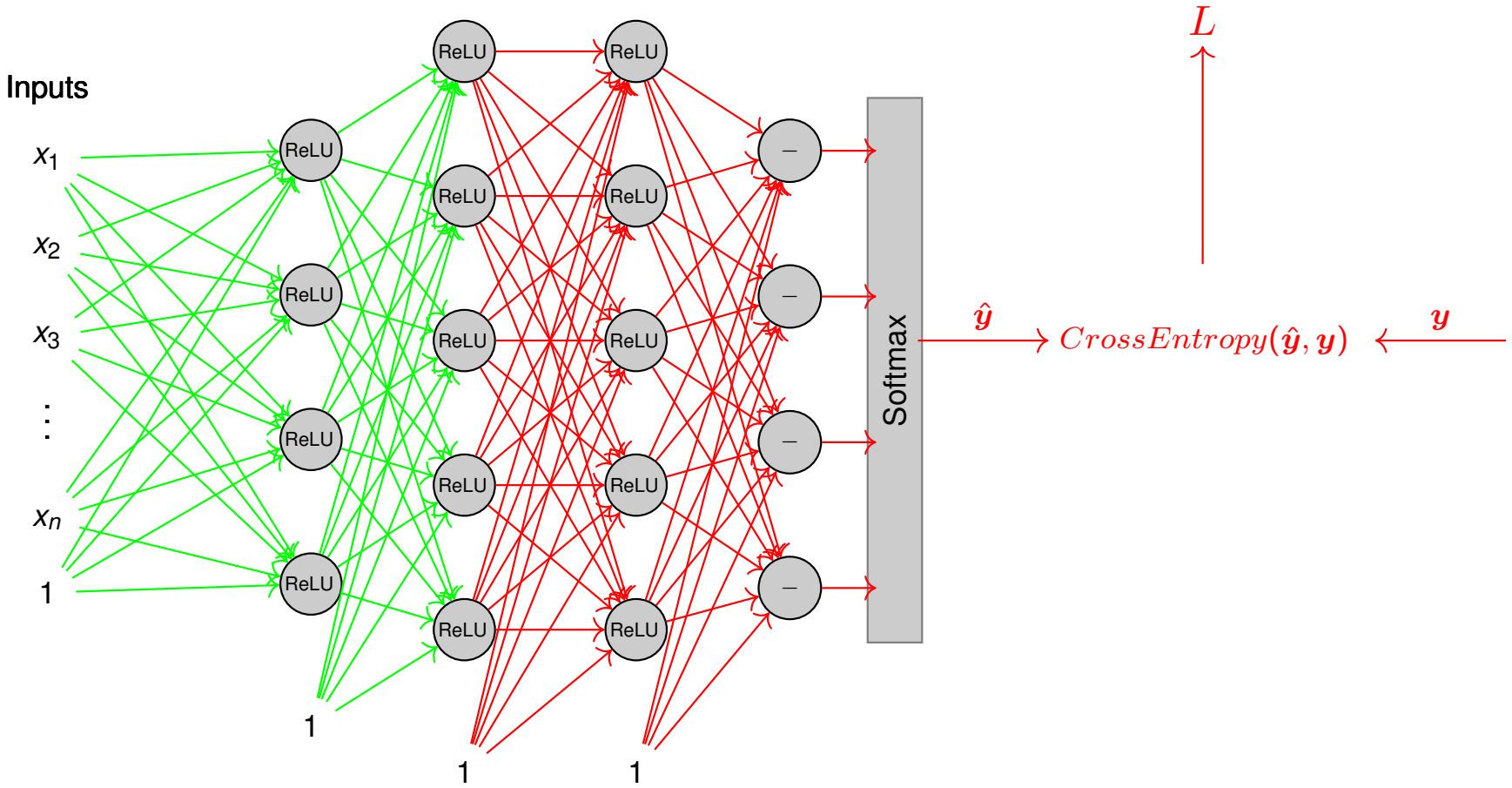
$$\frac{\partial L}{\partial z^4} = \hat{y} - y$$

Fully connected neural network - Backward Pass



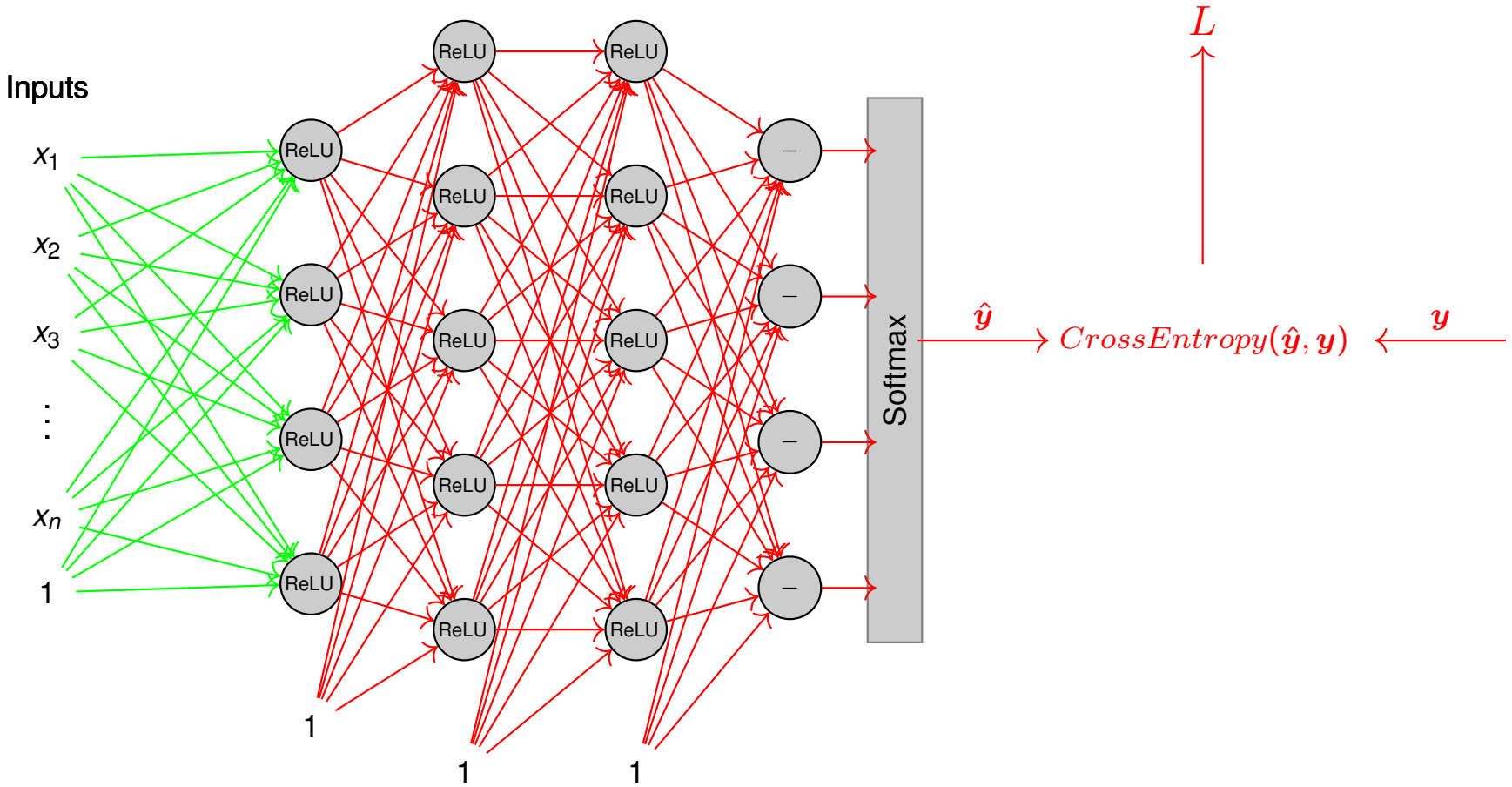
$$\frac{\partial L}{\partial W^4} = \frac{\partial L}{\partial z^4}\frac{\partial z^4}{\partial W^4} = \frac{\partial L}{\partial z^4}o^3, \quad \frac{\partial L}{\partial b^4} = \frac{\partial L}{\partial z^4}\frac{\partial z^4}{\partial b^4} = \frac{\partial L}{\partial z^4}, \quad \frac{\partial L}{\partial o^3} = \frac{\partial L}{\partial z^4}\frac{\partial z^4}{\partial o^3} = \frac{\partial L}{\partial z^4}W^4$$

Fully connected neural network - Backward Pass



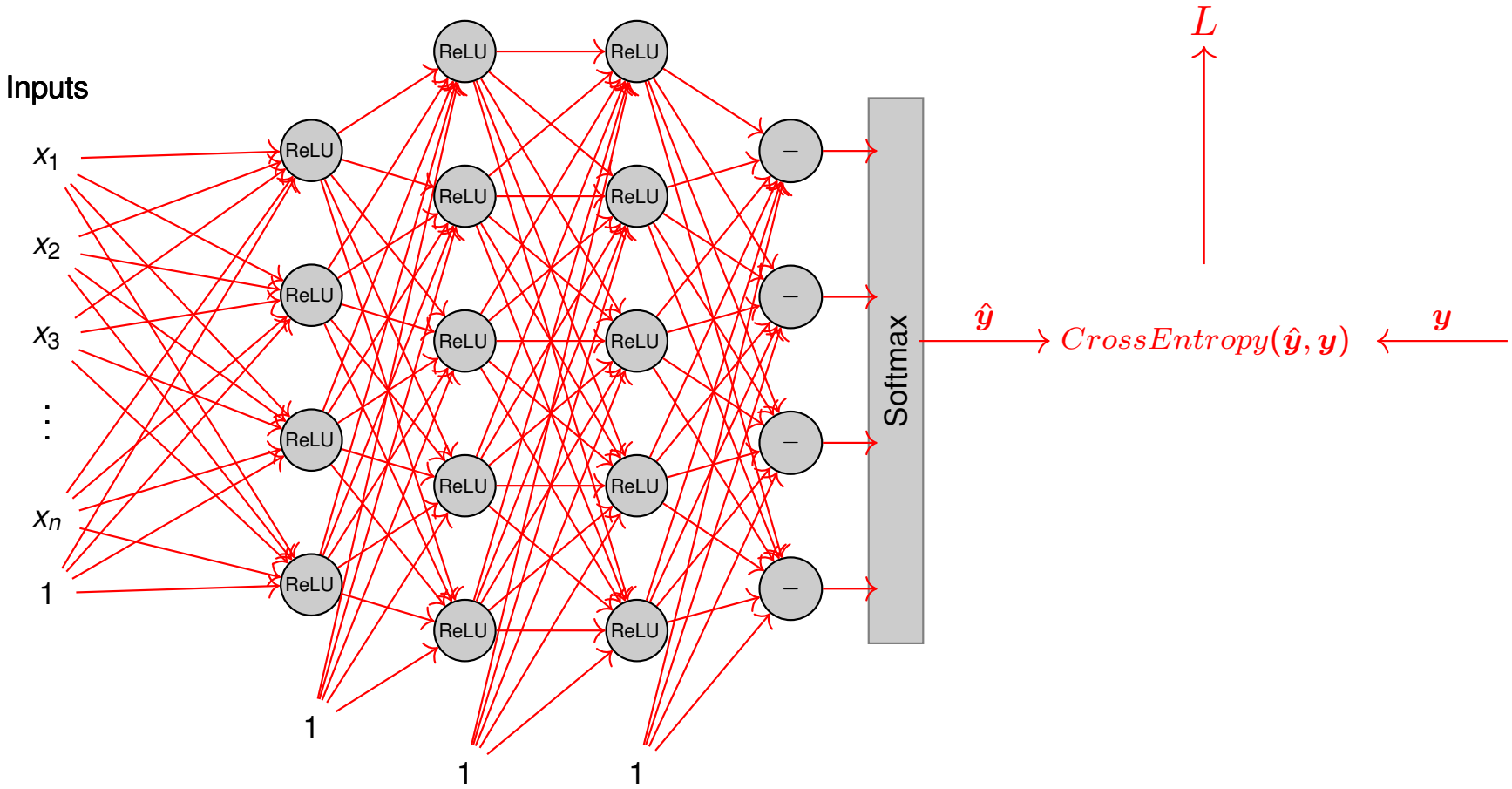
$$\frac{\partial L}{\partial z^3} = \frac{\partial L}{\partial o^3}\frac{\partial o^3}{\partial z^3} = \frac{\partial L}{\partial o^3} ReLU(sign(z^3)), \quad \frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial z^3} o^2, \quad \frac{\partial L}{\partial b^3} = \frac{\partial L}{\partial z^3}, \quad \frac{\partial L}{\partial o^2} = \frac{\partial L}{\partial z^3} W^3$$

Fully connected neural network - Backward Pass



$$\frac{\partial L}{\partial z^2} = \frac{\partial L}{\partial o^2} ReLU(sign(z^2)), \quad \frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial z^2} o^1, \quad \frac{\partial L}{\partial b^2} = \frac{\partial L}{\partial z^2}, \quad \frac{\partial L}{\partial o^1} = \frac{\partial L}{\partial z^2} W^2$$

Fully connected neural network - Backward Pass



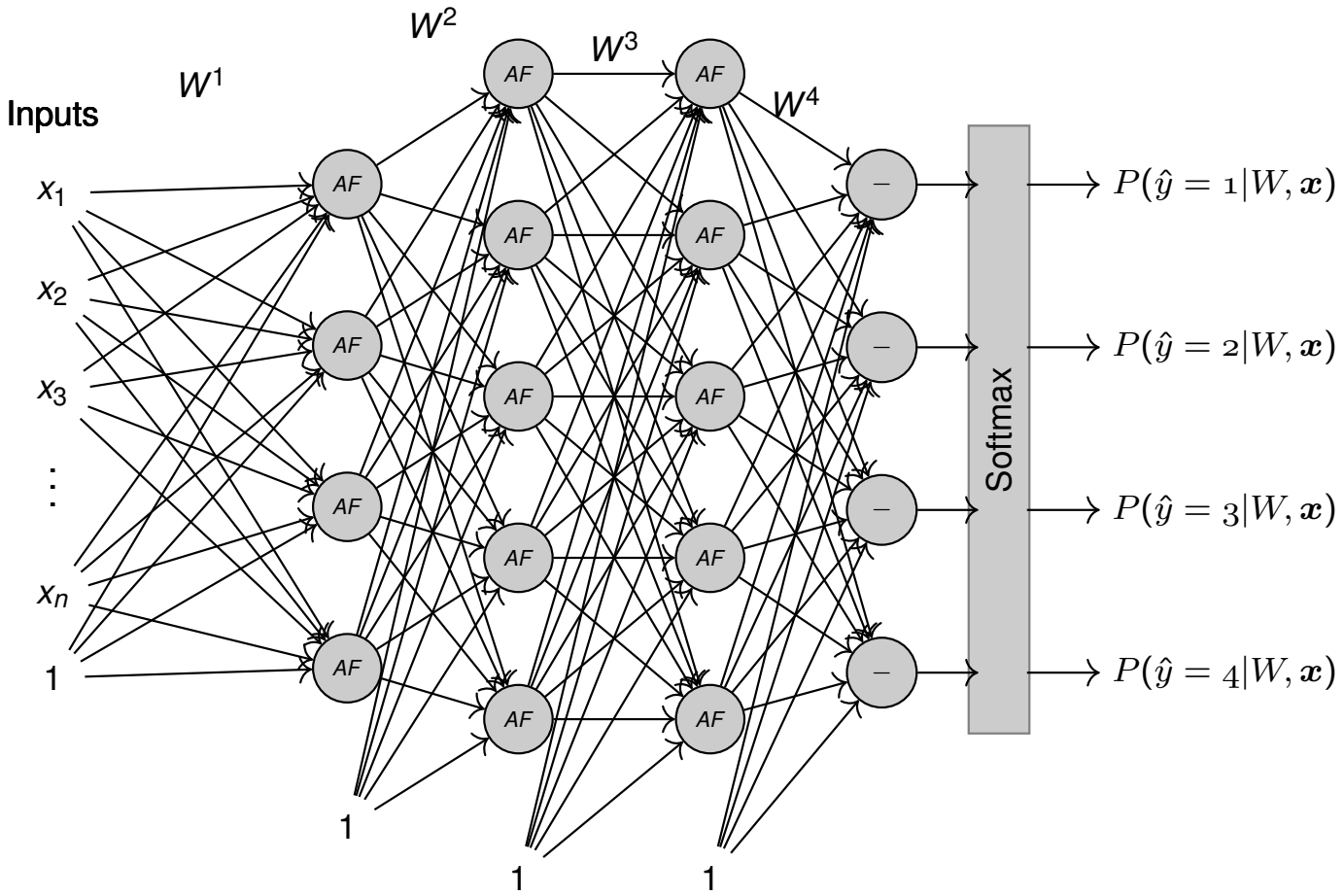
$$\frac{\partial L}{\partial z^1} = \frac{\partial L}{\partial o^1} ReLU(sign(z^1)), \quad \frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial z^1} x, \quad \frac{\partial L}{\partial b^1} = \frac{\partial L}{\partial z^1}, \quad \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z^1} W^1$$

Vanishing/Exploding Gradient

The vanishing gradient problem for deep networks

- A particular problem with training deep networks
  - The gradient of the error with respect to weights is unstable

Fully connected neural networks



$$\hat{y} = f^N(W^N f^{N-1}(W^{N-1} f^{N-2}(...f^2(W^2(f^1(W^1\boldsymbol{x}))))))$$

The problem with training deep networks

- A multilayer perceptron is a nested function

$$\hat{y} = f^N(W^N f^{N-1}(W^{N-1} f^{N-2}(...f^2(W^2(f^1(W^1\boldsymbol{x})))))$$

  - W^k is the weights matrix at the k^{th} layer
  - f^k is the activation function of the k^{th} layer
- The error for X can be written as

$$L = CE(f^N(W^N f^{N-1}(W^{N-1} f^{N-2}(...f^2(W^2(f^1(W^1\boldsymbol{x}))))), y)$$

  - CE is cross entropy

Training deep networks

- Let $\boldsymbol{z}^{\ell} = W^{\ell}\boldsymbol{o}^{\ell-1}$ and $\boldsymbol{o}^{\ell} = f^{\ell}(\boldsymbol{z}^{\ell})$

$$\frac{\partial L}{\partial \boldsymbol{o}^{\ell-1}} = ---$$

Training deep networks

- Let $\boldsymbol{z}^{\ell} = W^{\ell}\boldsymbol{o}^{\ell-1}$ and $\boldsymbol{o}^{\ell} = f^{\ell}(\boldsymbol{z}^{\ell})$

$$\frac{\partial L}{\partial \boldsymbol{o}^{\ell-1}} = --\frac{\partial L}{\partial \boldsymbol{o}^{\ell}}$$

Training deep networks

- Let $\boldsymbol{z}^{\ell} = W^{\ell}\boldsymbol{o}^{\ell-1}$ and $\boldsymbol{o}^{\ell} = f^{\ell}(\boldsymbol{z}^{\ell})$

$$\frac{\partial L}{\partial \boldsymbol{o}^{\ell-1}} = -\frac{\partial \boldsymbol{o}^{\ell}}{\partial \boldsymbol{z}^{\ell}}\frac{\partial L}{\partial \boldsymbol{o}^{\ell}}$$

Training deep networks

- Let $\boldsymbol{z}^{\ell} = W^{\ell}\boldsymbol{o}^{\ell-1}$ and $\boldsymbol{o}^{\ell} = f^{\ell}(\boldsymbol{z}^{\ell})$

$$\frac{\partial L}{\partial \boldsymbol{o}^{\ell-1}} = \frac{\partial \boldsymbol{z}^{\ell}}{\partial \boldsymbol{o}^{\ell-1}} \frac{\partial \boldsymbol{o}^{\ell}}{\partial \boldsymbol{z}^{\ell}} \frac{\partial L}{\partial \boldsymbol{o}^{\ell}}$$

Training deep networks

- Let $\boldsymbol{z}^{\ell} = W^{\ell}\boldsymbol{o}^{\ell-1}$ and $\boldsymbol{o}^{\ell} = f^{\ell}(\boldsymbol{z}^{\ell})$

$$\frac{\partial L}{\partial \boldsymbol{o}^{\ell-1}} = \frac{\partial \boldsymbol{z}^{\ell}}{\partial \boldsymbol{o}^{\ell-1}} \frac{\partial \boldsymbol{o}^{\ell}}{\partial \boldsymbol{z}^{\ell}} \frac{\partial L}{\partial \boldsymbol{o}^{\ell}}$$

$$\nabla_{\boldsymbol{o}^{\ell-1}} L = W^{\ell} . \nabla_{\boldsymbol{z}^{\ell}} f^{\ell} . \nabla_{\boldsymbol{o}^{\ell}} L$$

- Where
  - $\nabla_{\boldsymbol{x}} f$ is the jacobian matrix of $f(\boldsymbol{x})$ w.r.t $\boldsymbol{x}$

Training deep networks

- For

$$L = CE(f^N(W^N f^{N-1}(W^{N-1} f^{N-2}(...f^1(W^1 \boldsymbol{x})))), y)$$

Training deep networks

- For

$$L = CE(f^N(W^N f^{N-1}(W^{N-1} f^{N-2}(...f^1(W^1 \boldsymbol{x})))), y)$$

- We get

$$\nabla_{W^k} L = \boldsymbol{o}^{k-1}.\nabla_{\boldsymbol{z}^k} f^k.W^{k+1}.\nabla_{\boldsymbol{z}^{k+1}} f^{k+1}.....W^{N-1}.\nabla_{\boldsymbol{z}^{N-1}} f^{N-1}.W^N.\nabla_{\boldsymbol{z}^N} f^N.\nabla_{\hat{\boldsymbol{y}}} CE$$

$$\nabla_{W^k} L = \boldsymbol{o}^{k-1}.\nabla_{\boldsymbol{z}^k} f^k.\left(\prod_{i=k+1}^{N} W^i.\nabla_{\boldsymbol{z}^i} f^i\right).\nabla_{\hat{\boldsymbol{y}}} CE$$

- Where
  - W^N is the gradient of the affine transforation $\boldsymbol{z}^n$ w.r.t the output of the $n-1^{th}$ layer ($\boldsymbol{o}^{n-1}$) of the network
  - $\nabla_{\boldsymbol{z}^n} f^n$ is gradient of activation function $f^n(.)$ w.r.t to affine transforation $\boldsymbol{z}^n$ in the n^{th} layer of the network
  - $\boldsymbol{o}^{n-1}$ is the gradient of the affine transforation $\boldsymbol{z}^n$ w.r.t the weights of the n^{th} layer (W^n) of the network
  - $\nabla_{\hat{\boldsymbol{y}}} CE$ is the gradient of cross entropy w.r.t the output of the network ($\hat{\boldsymbol{y}}$)

Training deep networks

- For

$$L = CE(f^N(W^N f^{N-1}(W^{N-1} f^{N-2}(...f^1(W^1 \boldsymbol{x})))), y)$$

- We get

$$\nabla_{W^k} L = \boldsymbol{o}^{k-1} . \nabla_{\boldsymbol{z}^k} f^k . W^{k+1} . \nabla_{\boldsymbol{z}^{k+1}} f^{k+1} ..... W^{N-1} . \nabla_{\boldsymbol{z}^{N-1}} f^{N-1} . W^N . \nabla_{\boldsymbol{z}^N} f^N . \nabla_{\hat{\boldsymbol{y}}} CE$$

$$\nabla_{W^k} L = \boldsymbol{o}^{k-1} . \nabla_{\boldsymbol{z}^k} f^k . \left(\prod_{i=k+1}^{N} W^i . \nabla_{\boldsymbol{z}^i} f^i \right) . \nabla_{\hat{\boldsymbol{y}}} CE$$

- Where
  - W^N is the gradient of the affine transforation $\boldsymbol{z}^n$ w.r.t the output of the $n-1^{th}$ layer ($\boldsymbol{o}^{n-1}$) of the network
  - $\nabla_{\boldsymbol{z}^n} f^n$ is gradient of activation function $f^n(.)$ w.r.t to affine transforation $\boldsymbol{z}^n$ in the n^{th} layer of the network
  - $\boldsymbol{o}^{n-1}$ is the gradient of the affine transforation $\boldsymbol{z}^n$ w.r.t the weights of the n^{th} layer (W^n) of the network
  - $\nabla_{\hat{\boldsymbol{y}}} CE$ is the gradient of cross entropy w.r.t the output of the network ($\hat{\boldsymbol{y}}$)

Training deep networks

- For

$$L = CE(f^N(W^N f^{N-1}(W^{N-1} f^{N-2}(...f^1(W^1 \boldsymbol{x})))), y)$$

- We get

$$\nabla_{W^k} L = \boldsymbol{o}^{k-1} . \nabla_{\boldsymbol{z}^k} f^k . W^{k+1} . \nabla_{\boldsymbol{z}^{k+1}} f^{k+1} ..... W^{N-1} . \nabla_{\boldsymbol{z}^{N-1}} f^{N-1} . W^N . \nabla_{\boldsymbol{z}^N} f^N . \nabla_{\hat{\boldsymbol{y}}} CE$$

$$\nabla_{W^k} L = \boldsymbol{o}^{k-1} . \nabla_{\boldsymbol{z}^k} f^k . \left(\prod_{i=k+1}^{N} W^i . \nabla_{\boldsymbol{z}^i} f^i \right) . \nabla_{\hat{\boldsymbol{y}}} CE$$

- Where
  - W^N is the gradient of the affine transforation $\boldsymbol{z}^n$ w.r.t the output of the $n-1^{th}$ layer ($\boldsymbol{o}^{n-1}$) of the network
  - $\nabla_{\boldsymbol{z}^n} f^n$ is gradient of activation function $f^n(.)$ w.r.t to affine transforation $\boldsymbol{z}^n$ in the n^{th} layer of the network
  - $\boldsymbol{o}^{n-1}$ is the gradient of the affine transforation $\boldsymbol{z}^n$ w.r.t the weights of the n^{th} layer (W^n) of the network
  - $\nabla_{\hat{\boldsymbol{y}}} CE$ is the gradient of cross entropy w.r.t the output of the network ($\hat{\boldsymbol{y}}$)

Training deep networks

- For

$$L = CE(f^N(W^N f^{N-1}(W^{N-1} f^{N-2}(...f^1(W^1 \boldsymbol{x})))), y)$$

- We get

$$\nabla_{W^k} L = \boldsymbol{o}^{k-1} . \nabla_{\boldsymbol{z}^k} f^k . W^{k+1} . \nabla_{\boldsymbol{z}^{k+1}} f^{k+1} ..... W^{N-1} . \nabla_{\boldsymbol{z}^{N-1}} f^{N-1} . W^N . \nabla_{\boldsymbol{z}^N} f^N . \nabla_{\hat{\boldsymbol{y}}} CE$$

$$\nabla_{W^k} L = \boldsymbol{o}^{k-1} . \nabla_{\boldsymbol{z}^k} f^k . \left(\prod_{i=k+1}^{N} W^i . \nabla_{\boldsymbol{z}^i} f^i \right) . \nabla_{\hat{\boldsymbol{y}}} CE$$

- Repeated multiplication by the weights matrix and jacobian matrices of activation functions will result in exploding or vanishing gradients
  - Depends on the spectral norm (largest singular value) of jacobian matrices
- The gradients in the lower/earlier layers can explode or vanish
  - Resulting in insignificant or unstable gradient descent updates
  - Problem gets worse as network depth increases

References

- Pattern Recognition and Machine Learning, Ch.1 & Ch.4
- Deep Learning, Ian Goodfellow, MIT Press, Ch.5, 6, 7, 8 ,and 9