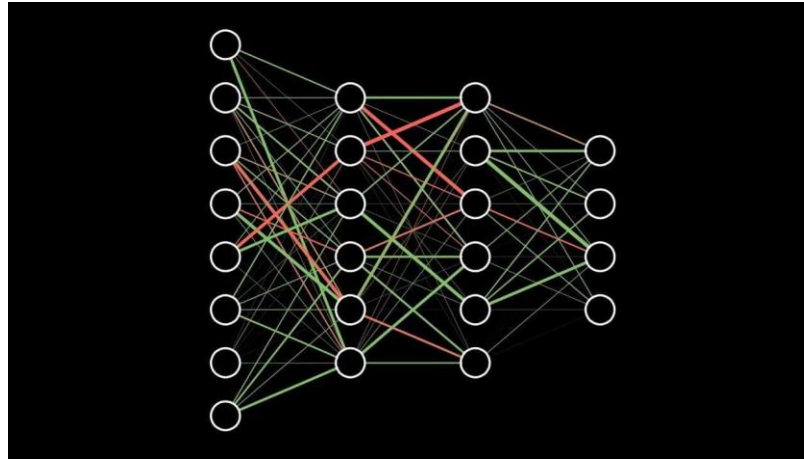


Deep Learning Workshop

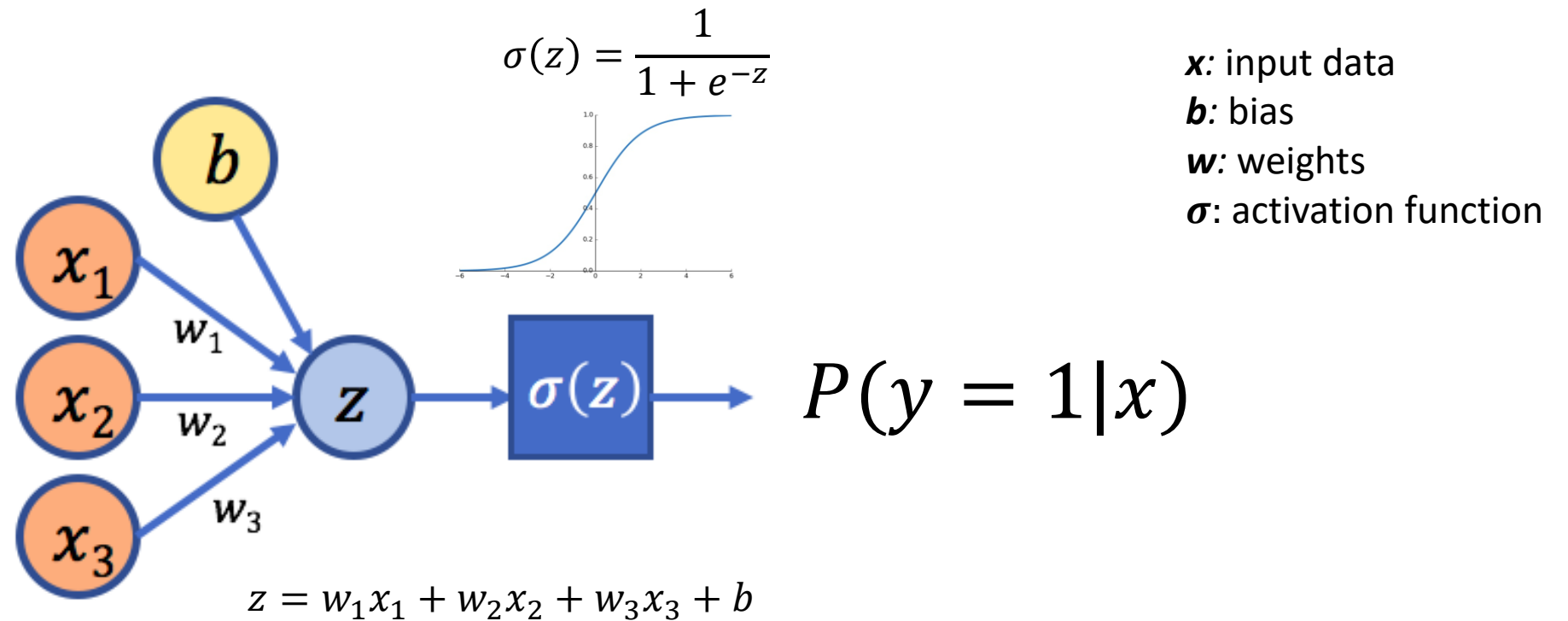
Neural Networks and Friends



Instructor: Aaron Low

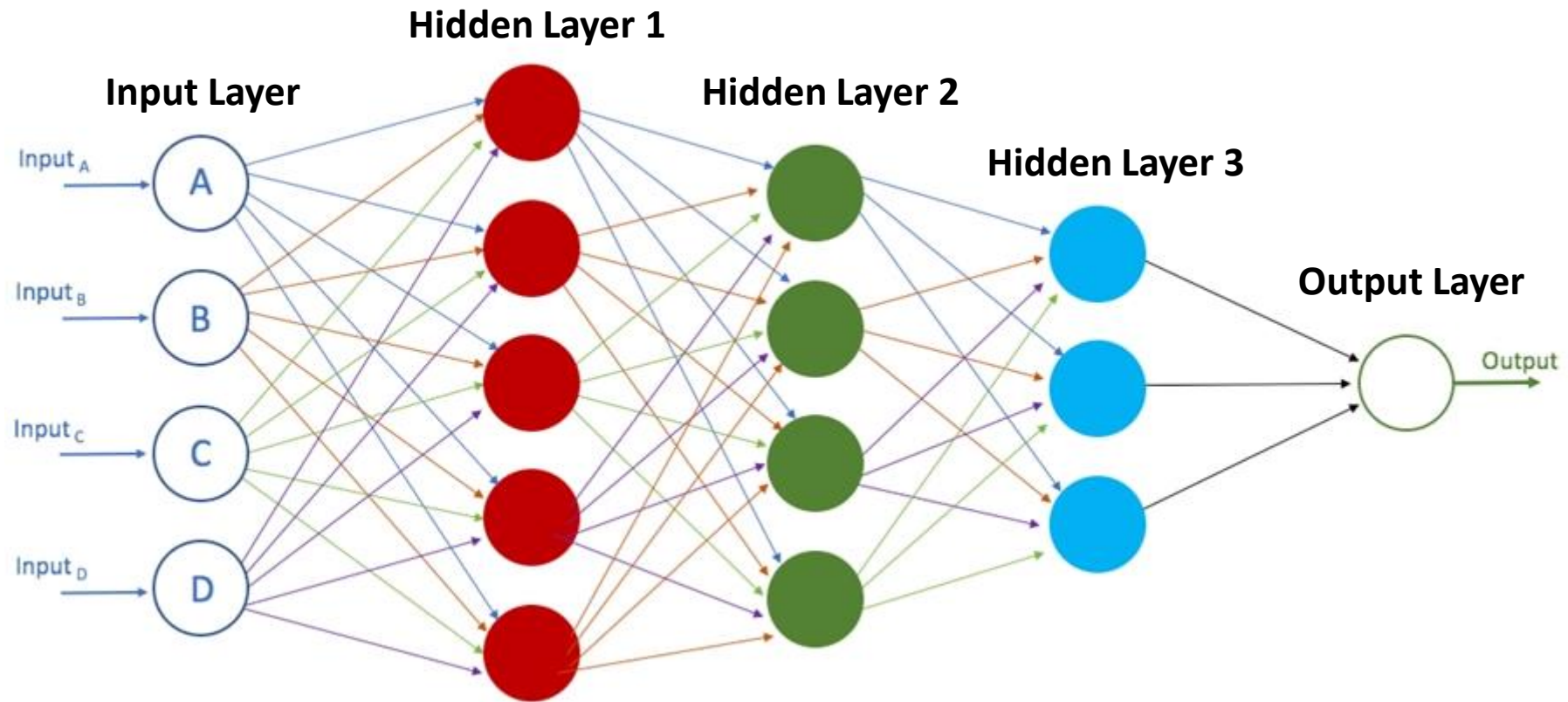
HELP University, Faculty of Computing and Digital Technology

Logistic Regression



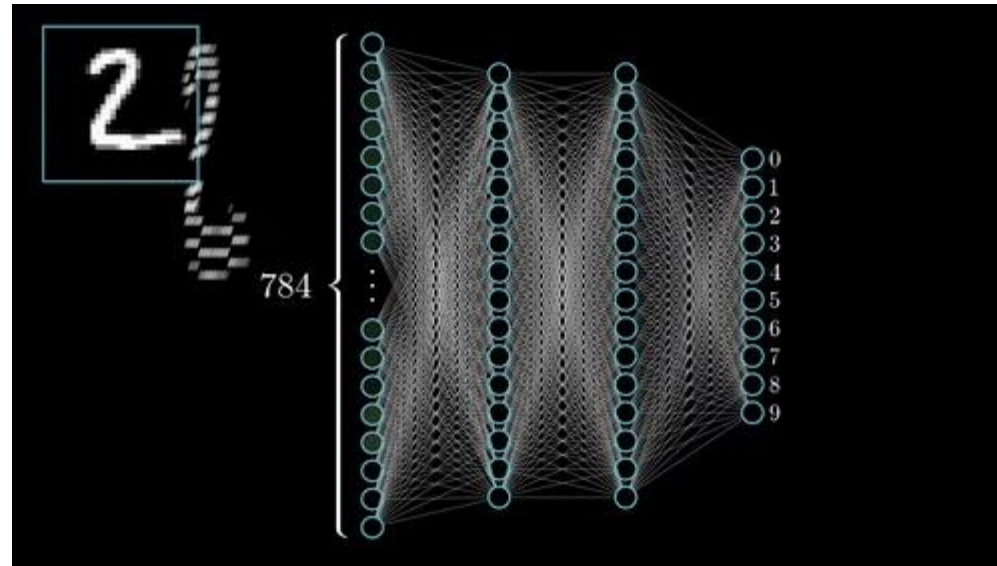
Logistic Regression from <https://medium.com/@melodious/understanding-deep-neural-networks-from-first-principles-logistic-regression-bd2f01c9e263>

Neural Network



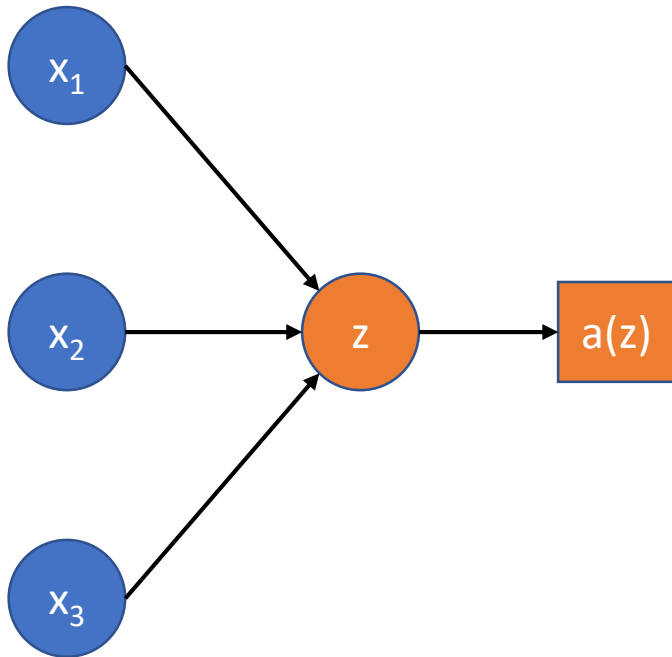
Deep Neural Network from <https://developer.oracle.com/databases/neural-network-machine-learning.html>

Neural Network

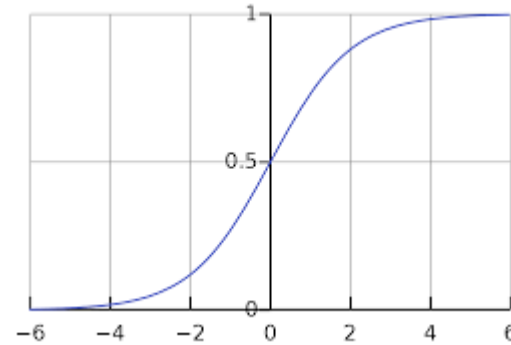


Neural Network GIF from https://www.youtube.com/channel/UCYO_jab_esuFRV4b17AJtAw

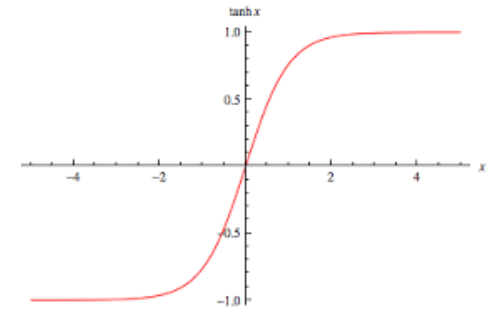
Activation Functions



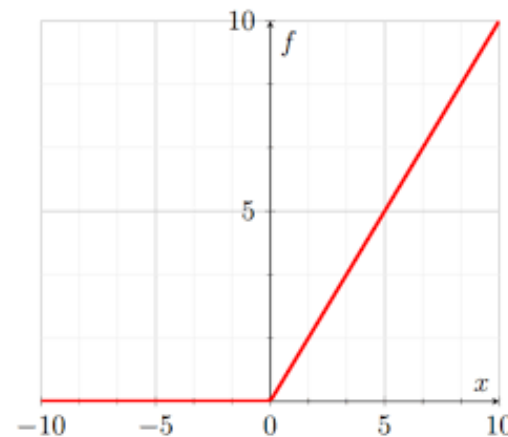
$$\text{Sigmoid}(z) = \frac{1}{1+e^{-z}}$$



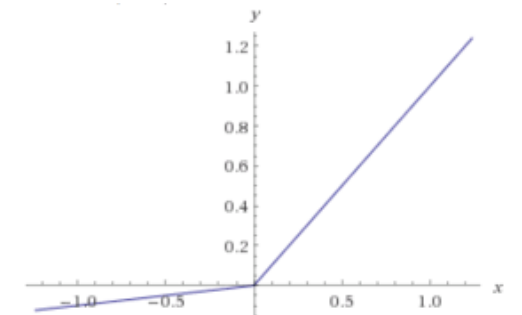
$$\text{Tanh}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



$$\text{ReLU}(z) = \max(0, z)$$

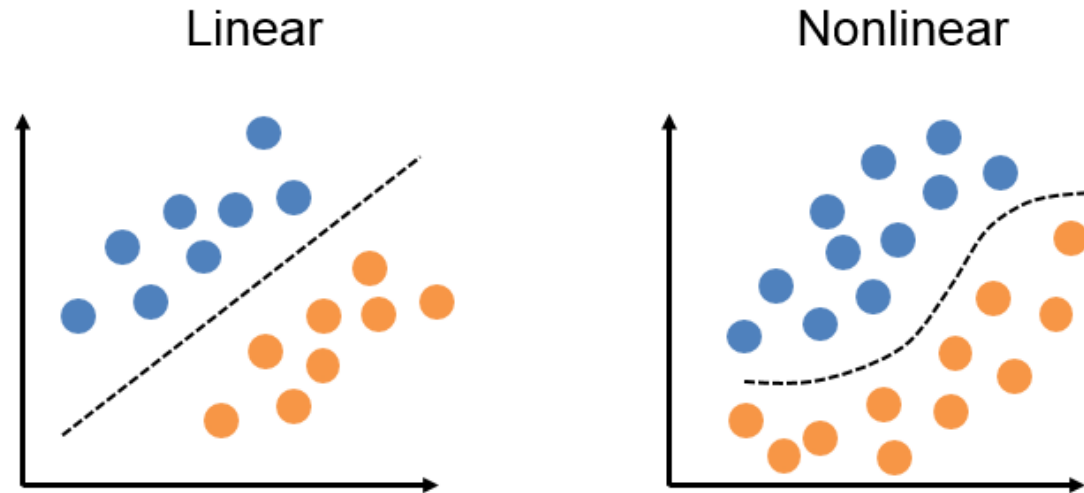


$$\text{Leaky_ReLU}(z) = \max(0.1z, z)$$



Why Activation Functions?

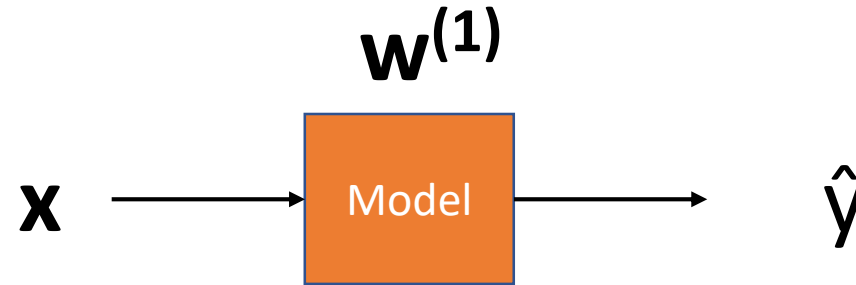
- Specifically we want **non-linear** activation functions
- To allow our model to learn **non-linear** mappings
- Most input-output mappings we would like to learn are **non-linear**



* The activation function should also be **differentiable**

How do we train the model?

1. Forward pass to obtain prediction



2. Calculate error using cost function, J

$$e = J(\mathbf{w})$$

3. Use backpropagation to calculate gradient

$$\frac{\partial e}{\partial \mathbf{w}^{(1)}}$$

4. Update weights using an optimization algorithm (typically a variant of gradient descent)

$$\mathbf{w}^{(2)} = \mathbf{w}^{(1)} - \alpha \frac{\partial e}{\partial \mathbf{w}^{(1)}}$$

Types of Cost Function

- **Mean Absolute Error / L1 Loss**

- $J = \frac{1}{N} \sum_i^N |y_i - \hat{y}_i|$

Typically used in regression

- **Mean Squared Error / L2 Loss / Euclidean distance**

- $J = \frac{1}{N} \sum_i^N (y_i - \hat{y}_i)^2$

- **Binary cross-entropy Loss (C = 2)**

- $J = \frac{1}{N} \sum_i^N (-y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i))$

Typically used in classification

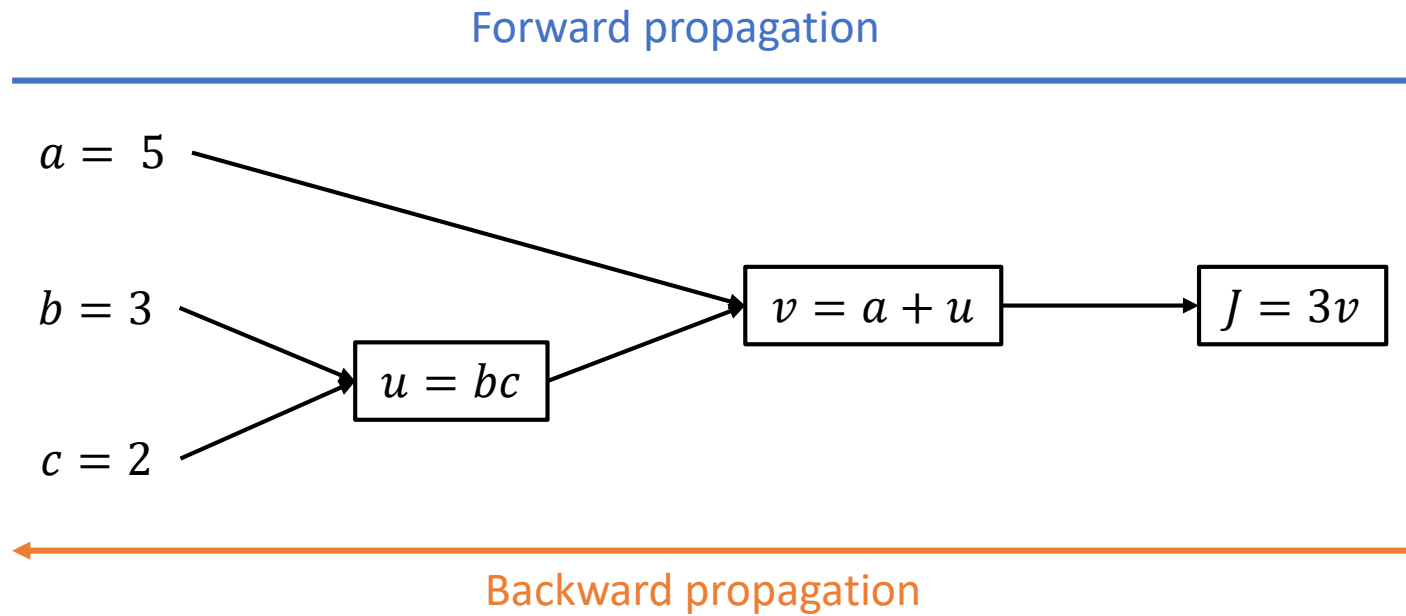
- **Cross-entropy Loss**

- $J = \frac{1}{N} \sum_i^N [-\sum_j^C y_j \log \hat{y}_j]$

* There are many types of cost functions that are possible depending on what you want your model to learn

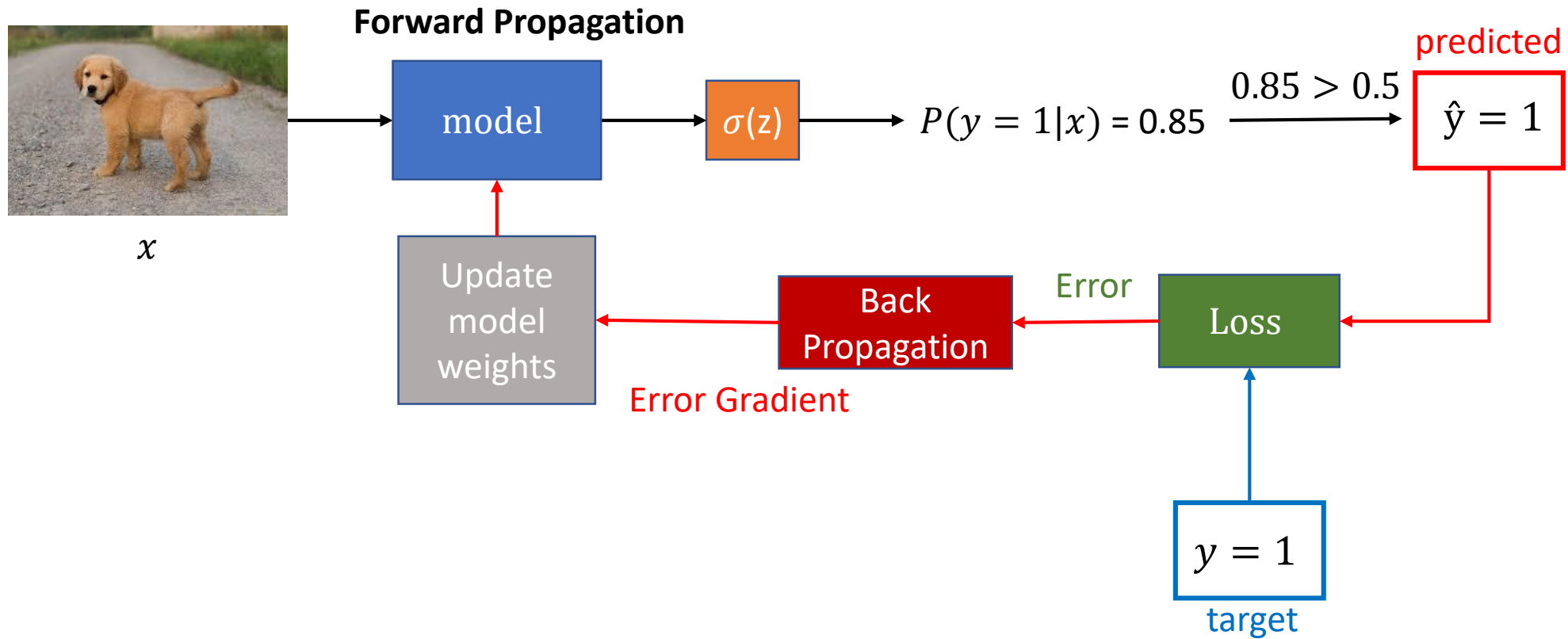
Backpropagation

- Used to calculate error gradient with respect to the model weights
- Calculate using chain rule



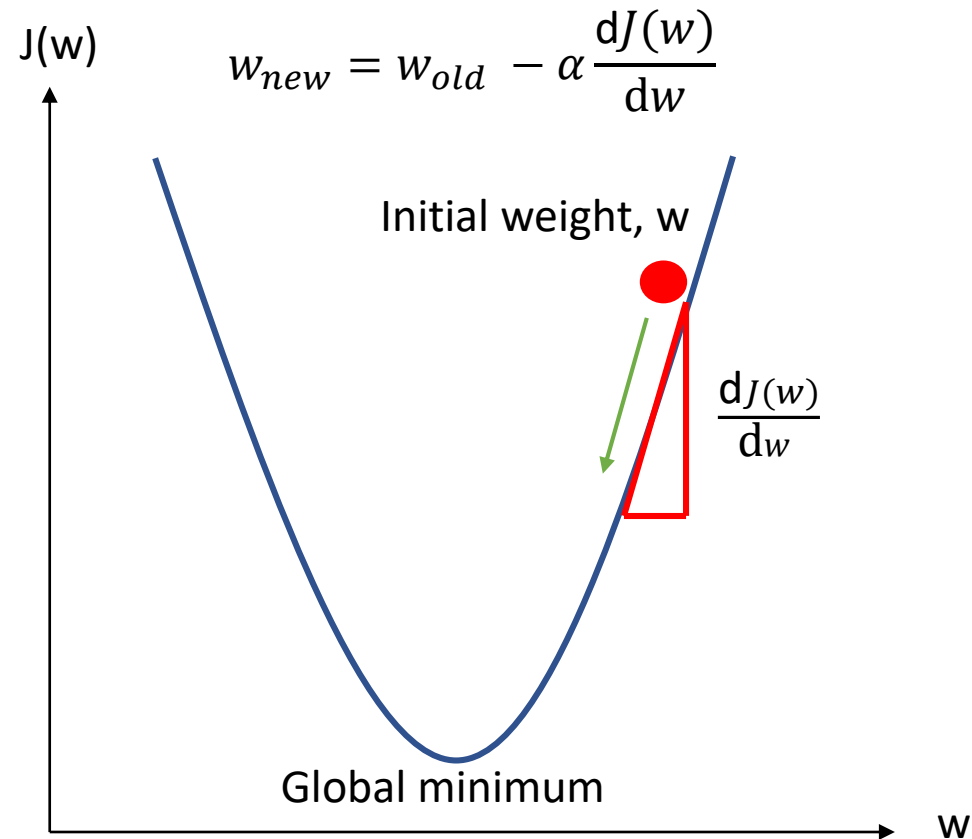
Computation graph as shown in by [Andrew Ng in his deeplearning.ai course](#)

Loss Calculation and Back Propagation



Gradient Descent

- Method to optimize our model and find our optimal weights
- We want to find the weights, w that minimize our cost function, $J(w)$
- Currently, there are many variants to improve standard gradient descent



Batch Gradient Descent

- Training is normally carried out in **batches** of training data
- **Stochastic Gradient Descent**
 - Update weights for each training data example
 - More generalization
- **Batch Gradient Descent**
 - Update weights after going through **every** training data example
 - Faster computation (vectorization)
- **Mini-batch Gradient Descent**
 - Update weights after going through **N** training data examples

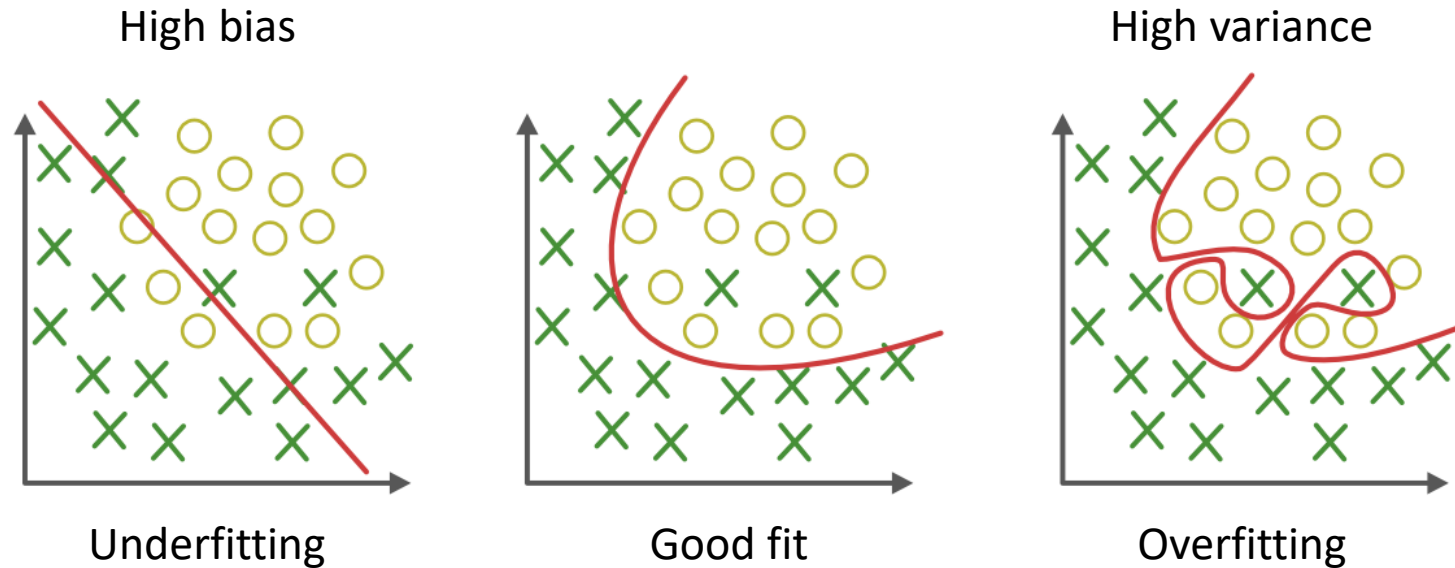
Evaluating your Model

- **Identify how well the model performs**
- **Identify how well the model generalizes to unseen data samples**
- **Quantitative analysis**
 - Use performance metrics
- **Qualitative analysis**
 - Useful for visual based output



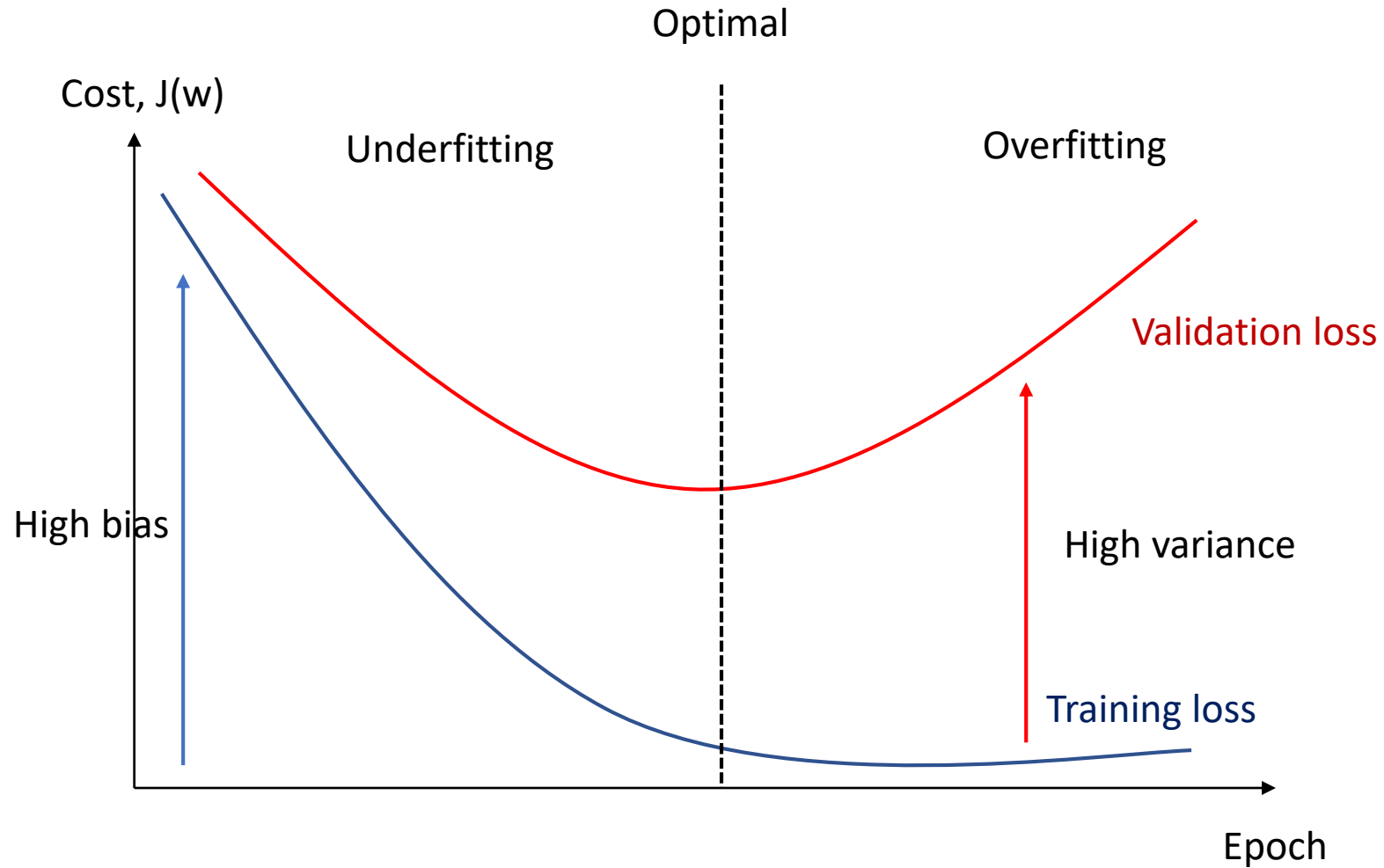
Underfitting and Overfitting

Understanding the model's ability to **generalize** to unseen data



Graphs from <https://towardsdatascience.com/underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6fe4a8a49dbf>

Underfitting and Overfitting



How do we deal with this? - **Regularization**

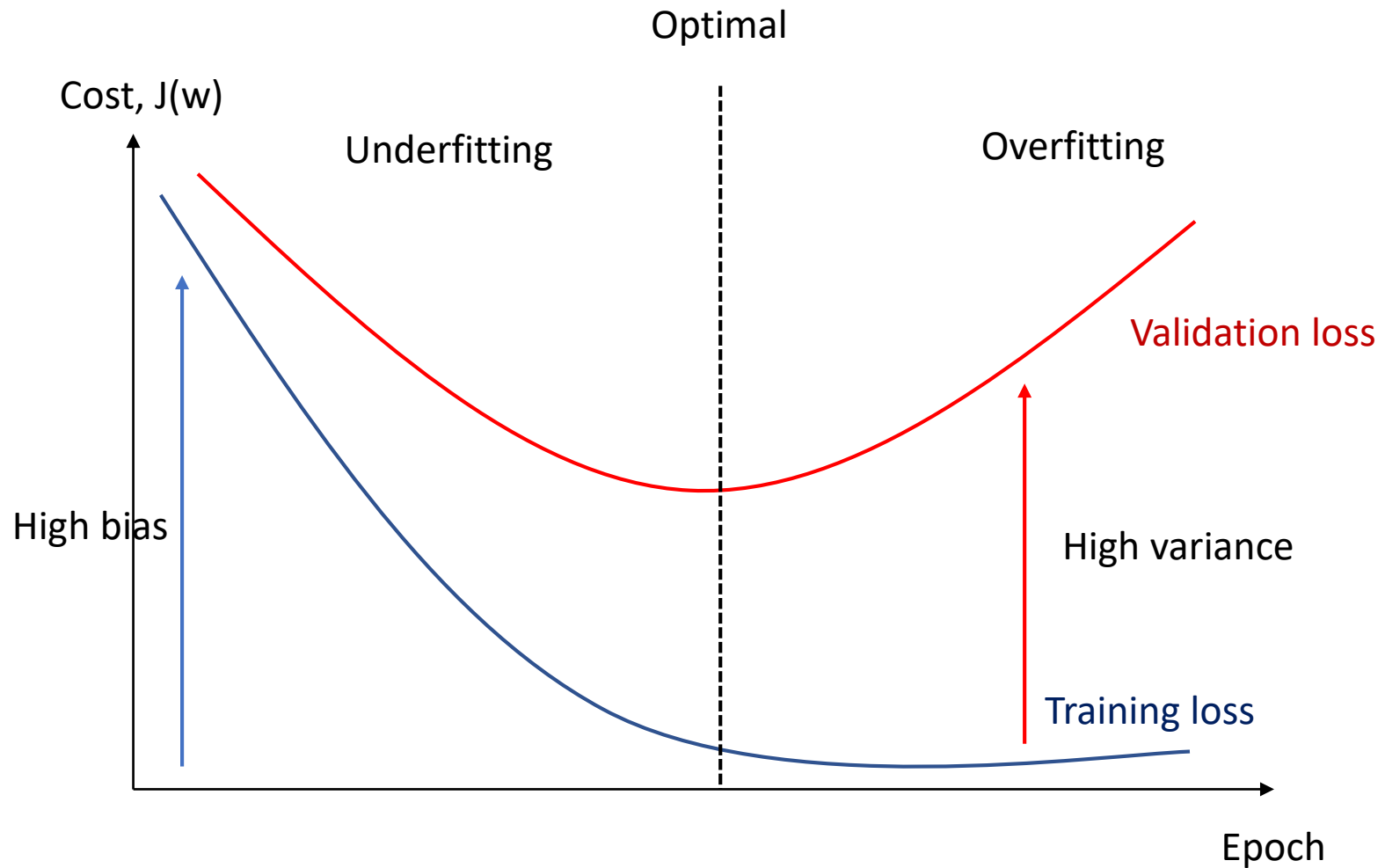
Underfitting and Overfitting

- **We would like to reduce bias and variance**
- **Reducing bias (Prevent underfitting)**
 - Increase size of network
 - Train longer
- **Reducing variance (Prevent overfitting)**
 - Add more training data
 - Regularization

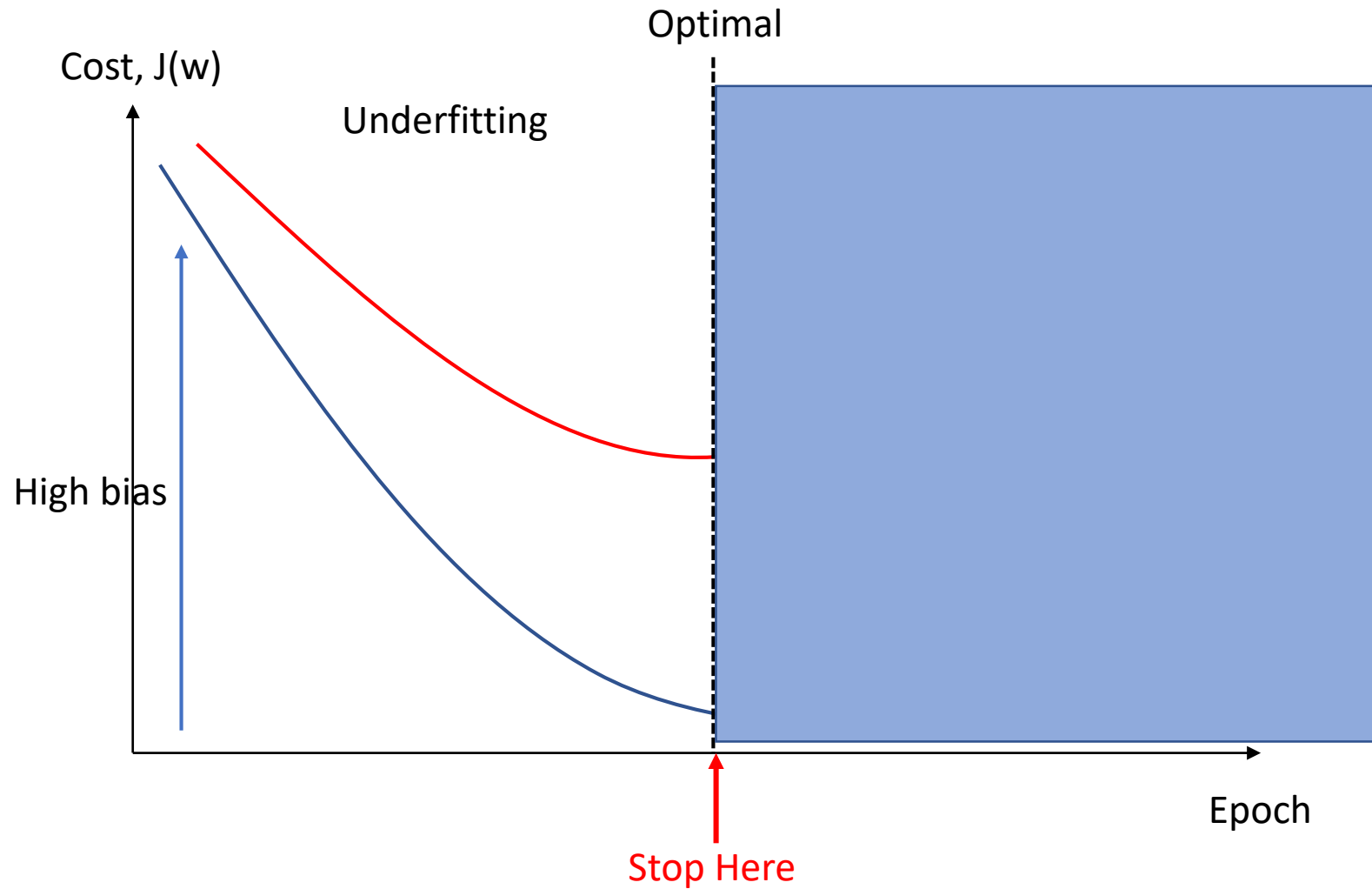
Regularization

- **Process of adding or constraining information in order to prevent overfitting**
- **Prevent our network from “memorizing” the correct output when training**
- **Types of regularization methods**
 - L1 regularization ← add to cost function
 - $\lambda \sum_{i=1}^N |w_i|$
 - L2 regularization ← add to cost function
 - $\lambda \sum_{i=1}^N w_i^2$
 - Dropout layer
 - Early stopping
 - Data augmentation
 - Batch Normalization

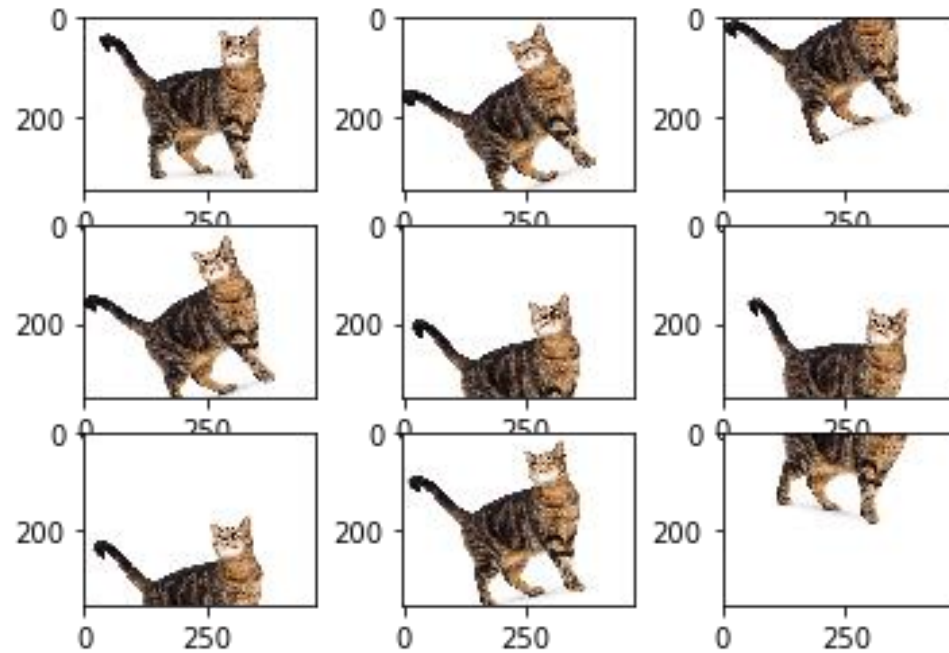
Regularization: Early Stopping



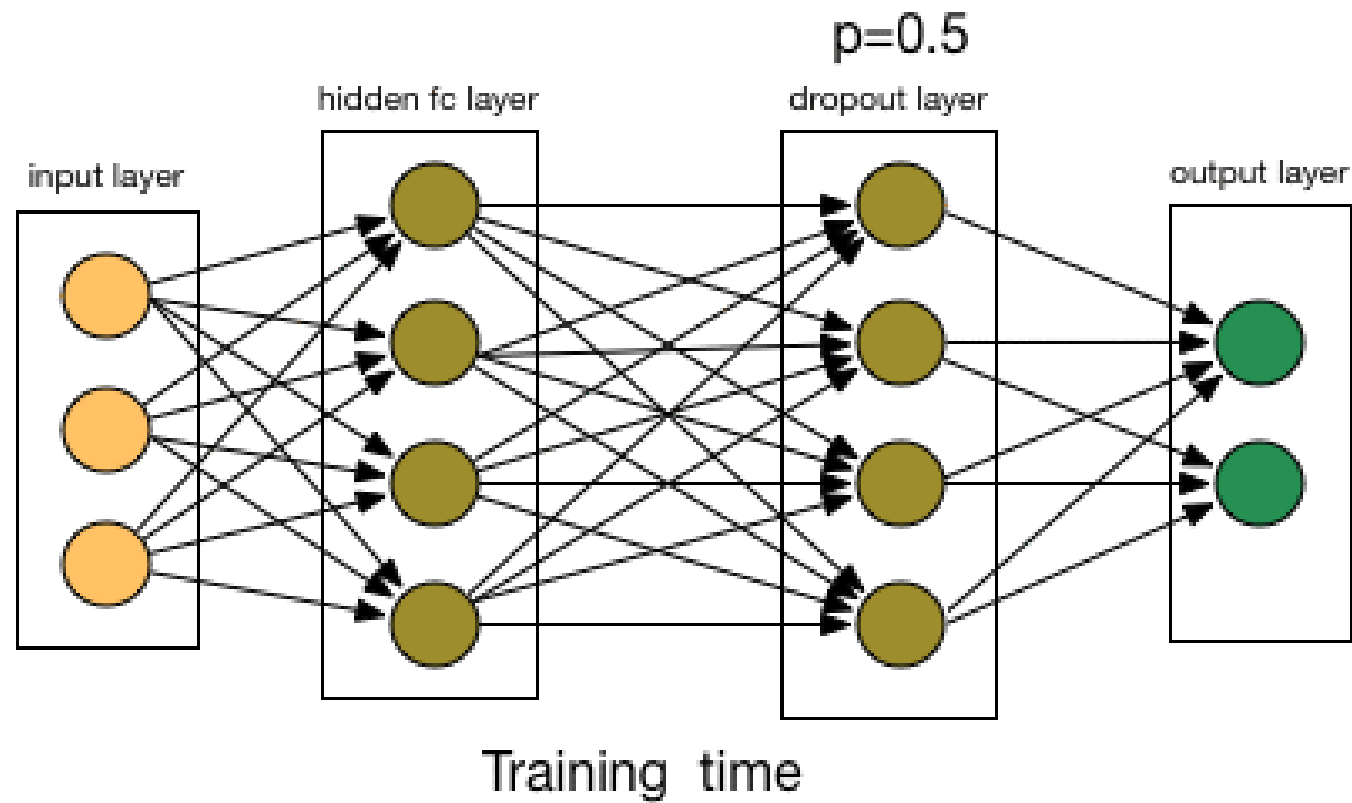
Regularization: Early Stopping



Regularization: Data Augmentation

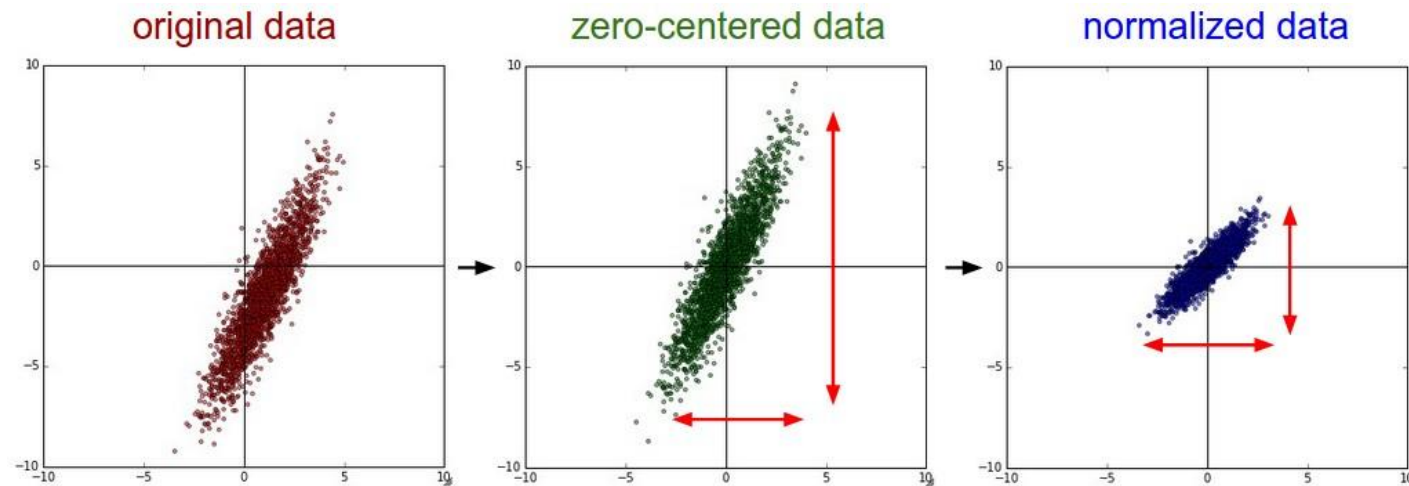


Regularization: Dropout



Normalization

- **Input normalization**
 - Scale input data to $[0, 1]$ or $[-1, 1]$ or according to mean and std
- **Batch normalization**
 - Normalization activations at hidden layer inputs

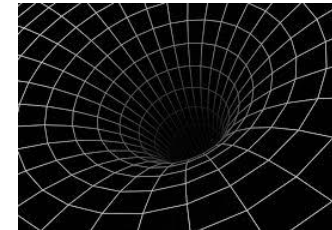
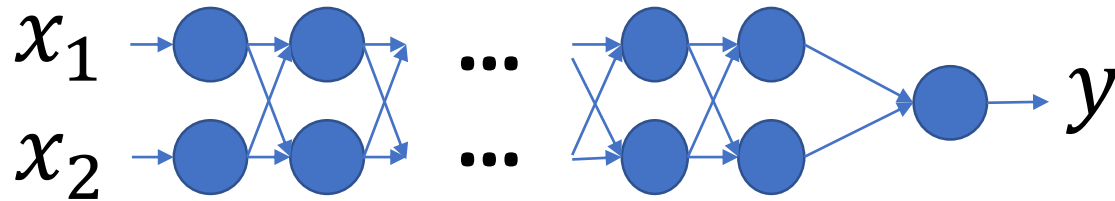


Why Normalization?

- **Input normalization**
 - Set features to similar scales (*imagine one set of features range $[0...1]$ and another $[0...1,000]$*)
 - Reduce outlier
 - Speed up training
- **Batch normalization**
 - Make weights deeper in networks more robust to changes in weights in earlier layers

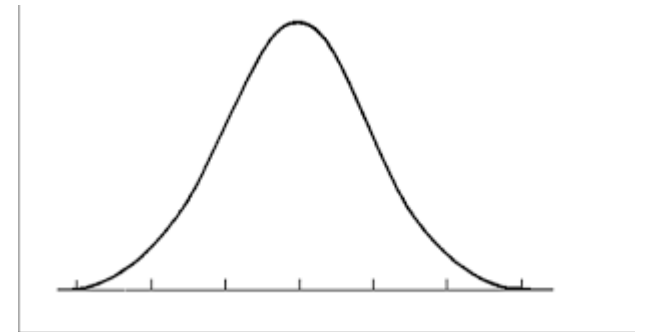
Vanishing/Exploding Gradients

- As neural network becomes deeper, gradient propagation can result in gradients becoming vanishingly small or explodingly large
- How to deal with this?
 - Gradient Clipping
 - Weight Initialization
 - Weight Regularization
 - Increase capacity of network
 - Change number of hidden layers



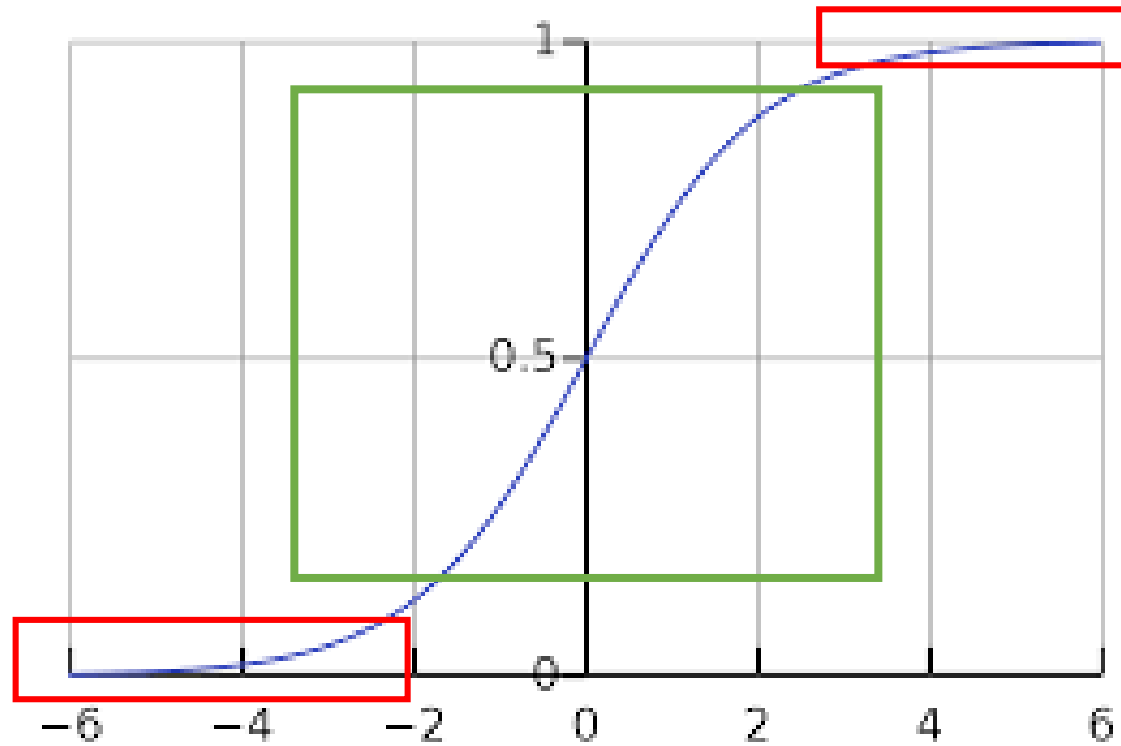
Weight Initialization

- **Weight initialization helps prevent the network from converging too slowly or “exploding”**
- **Zero initialization**
 - No matter how long you train, hidden unit weights will all be the same
 - **Not helpful**
- **Random initialization**
 - Typically Gaussian distribution
- **Xavier initialization**
 - Multiply randomly initialized weights with $\sqrt{\frac{1}{n}}$ where n is the number of features for the given layer

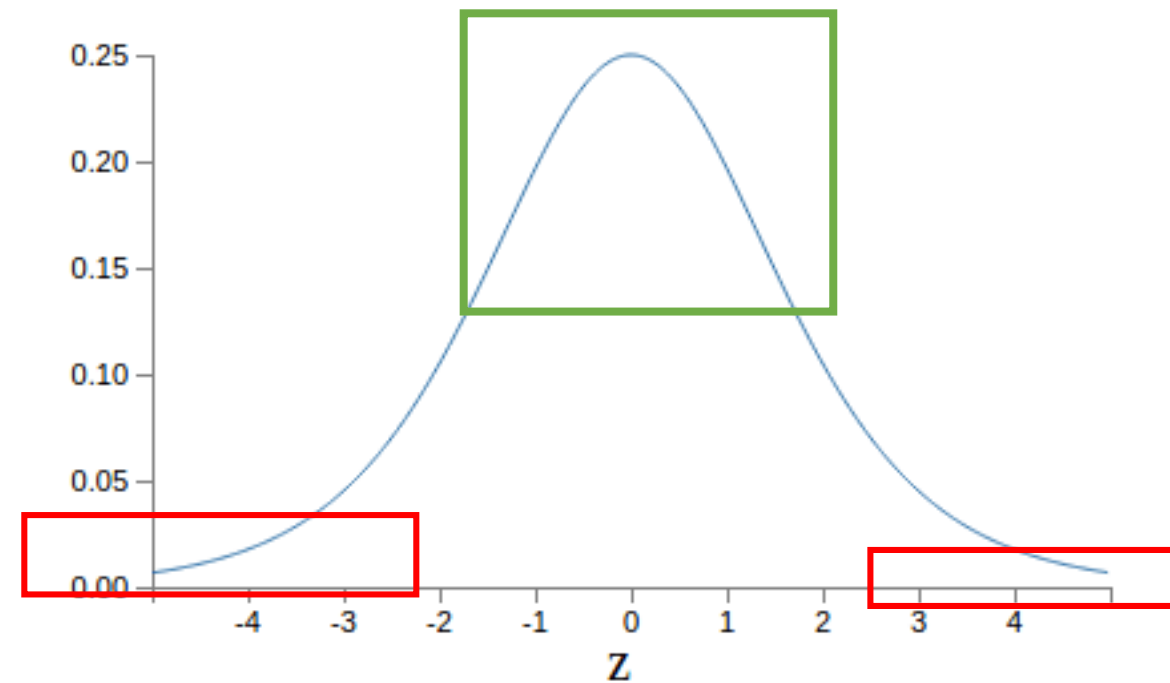


Weight Initialization

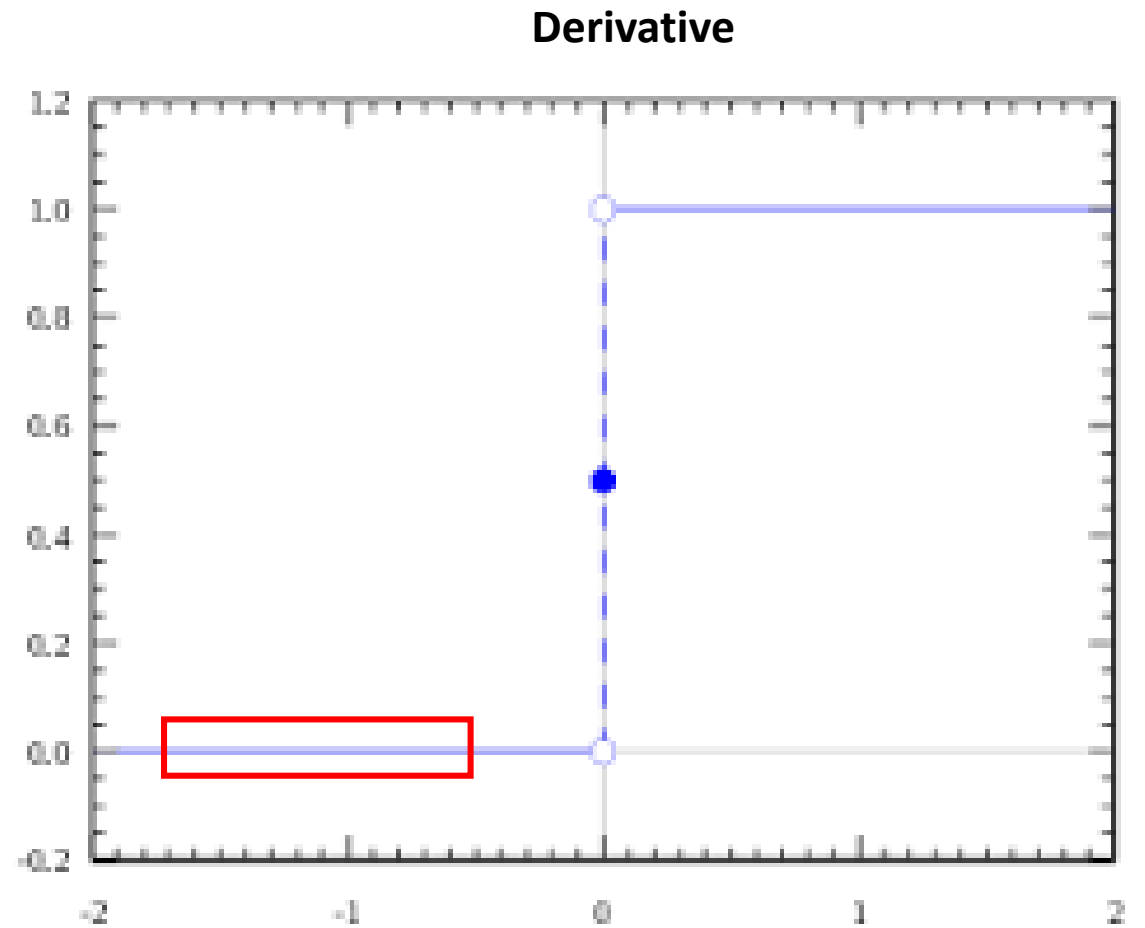
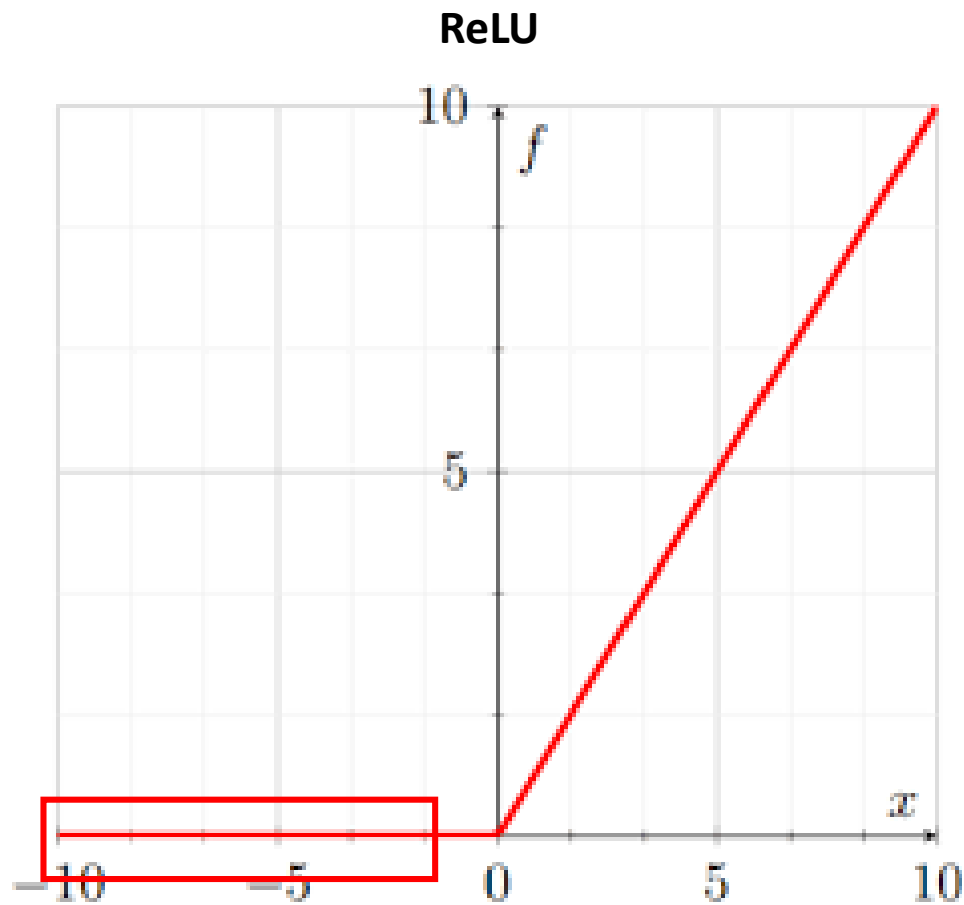
Sigmoid



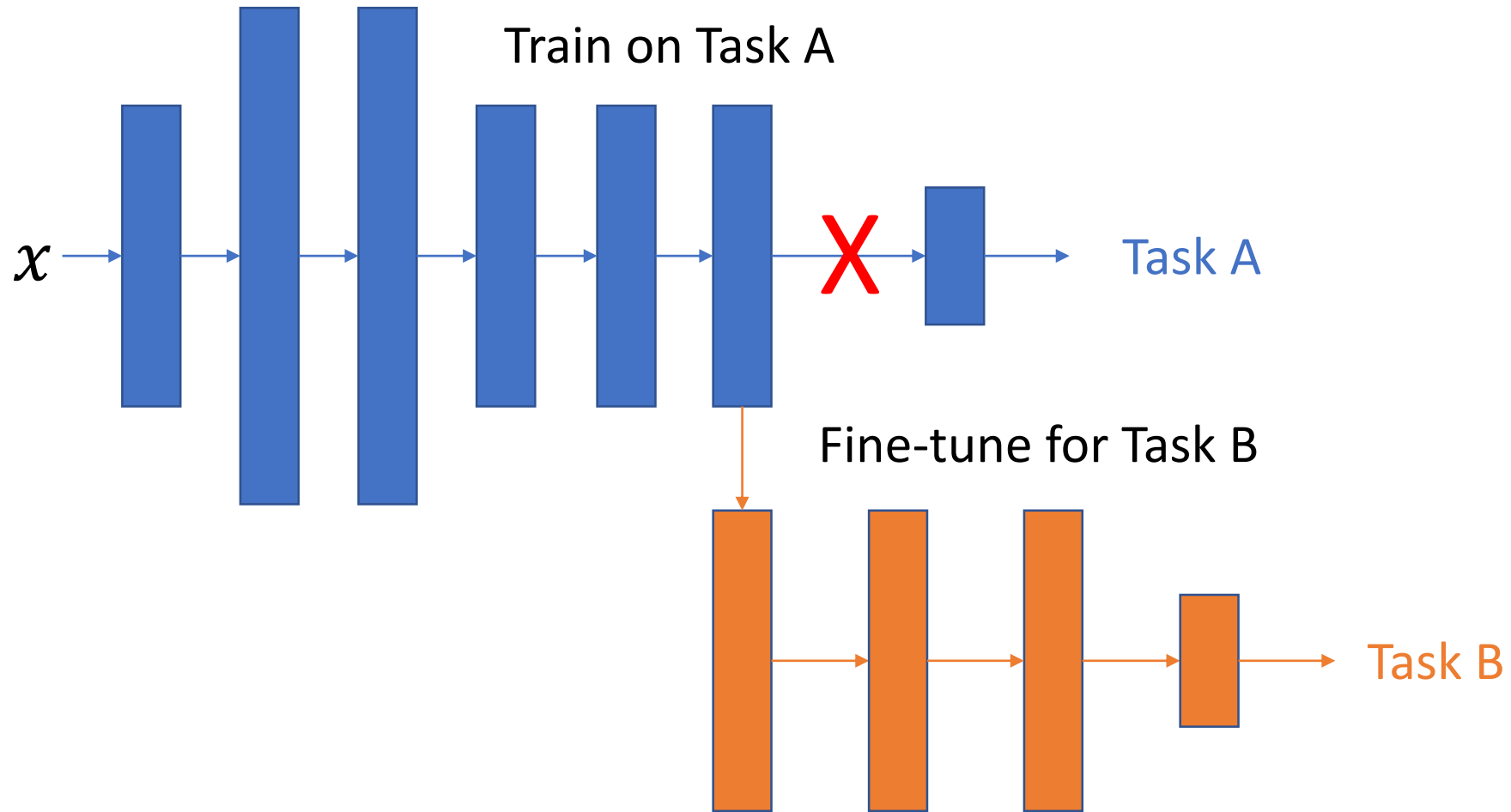
Derivative



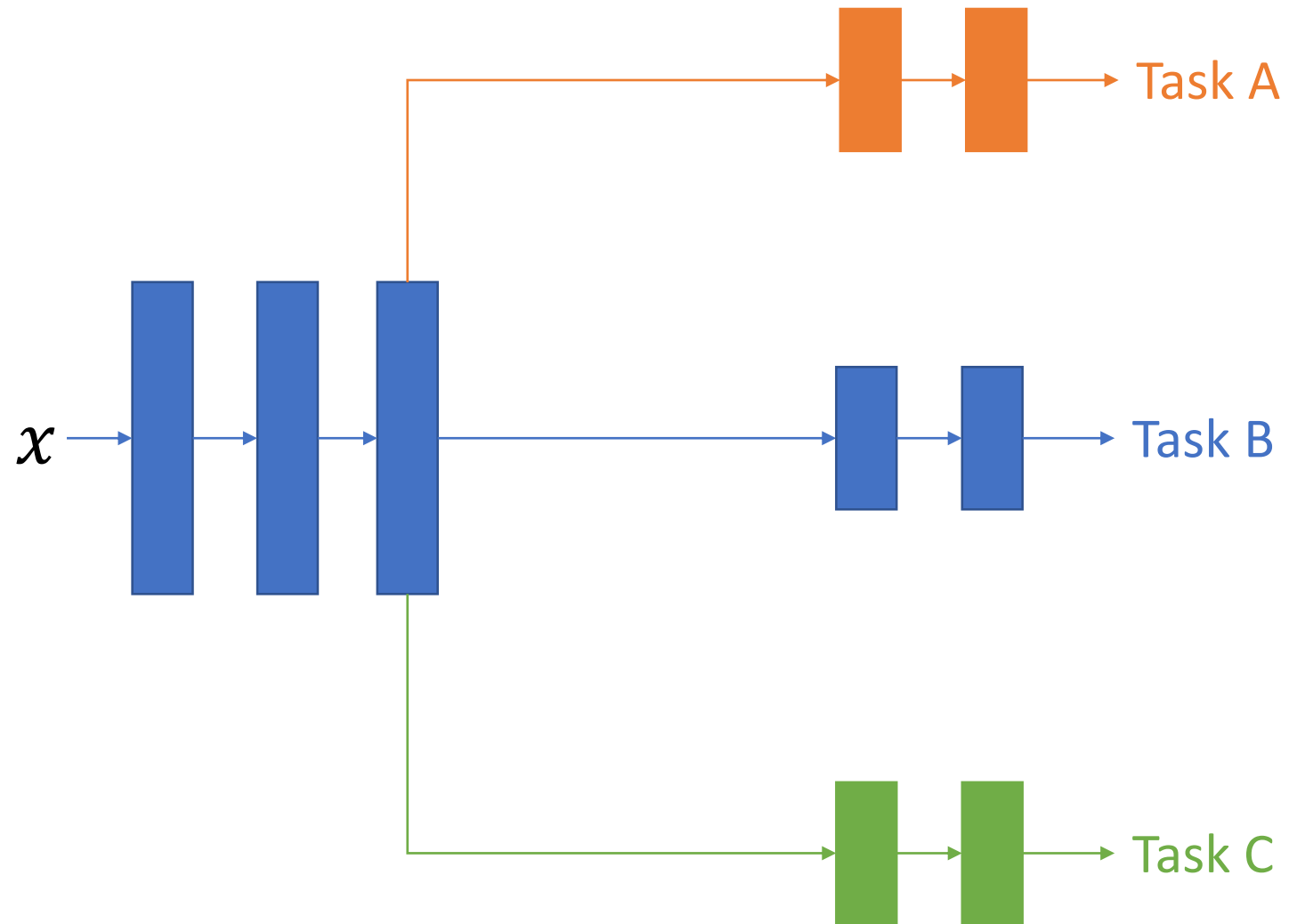
Weight Initialization



Transfer Learning



Multitask Learning



Questions?