Neural Network Basics

FA690 Machine Learning in Finance

Dr. Zonghao Yang

2025 Spring

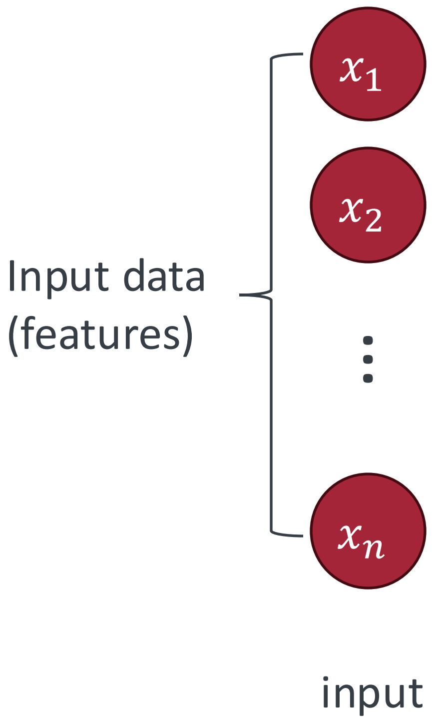
Learning Objectives

- Describe the architecture of a neural network, including input, hidden, and output layers, and explain the role of activation functions such as ReLU, sigmoid, and softmax in enabling nonlinear decision boundaries.
- Understand the principles of gradient descent, including backpropagation and optimization techniques such as momentum and adaptive learning rates.
- Identify key hyperparameters of a neural network, such as learning rate, batch size, and regularization techniques (dropout, L2 penalty), and develop intuition for tuning them based on learning curves and validation performance.

Fundamentals of Neural Networks

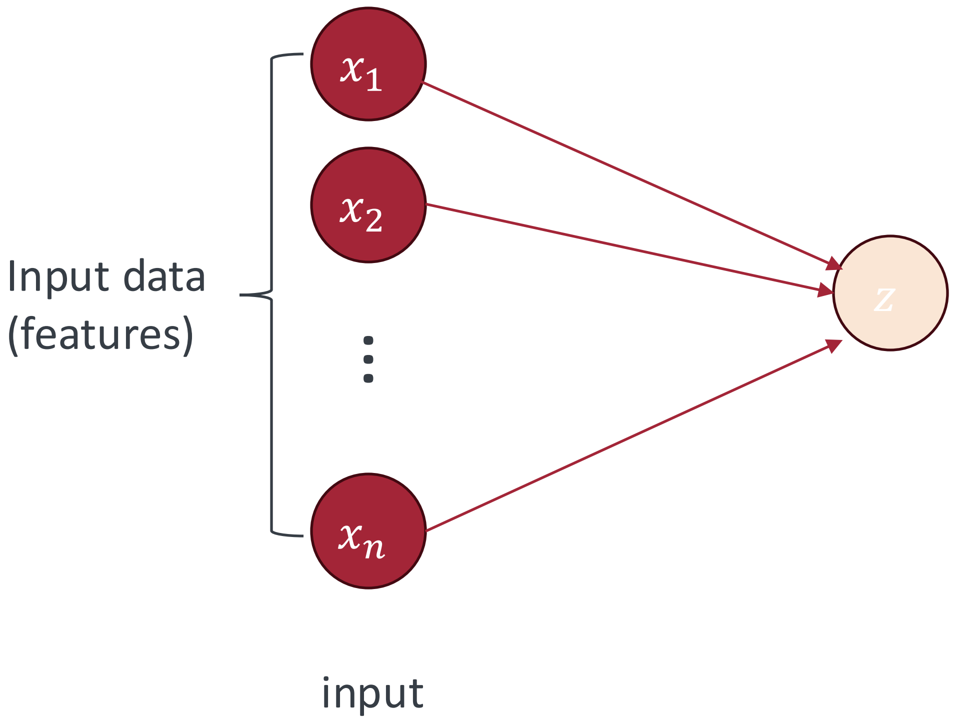
Introduction to the Perceptron

The base architecture



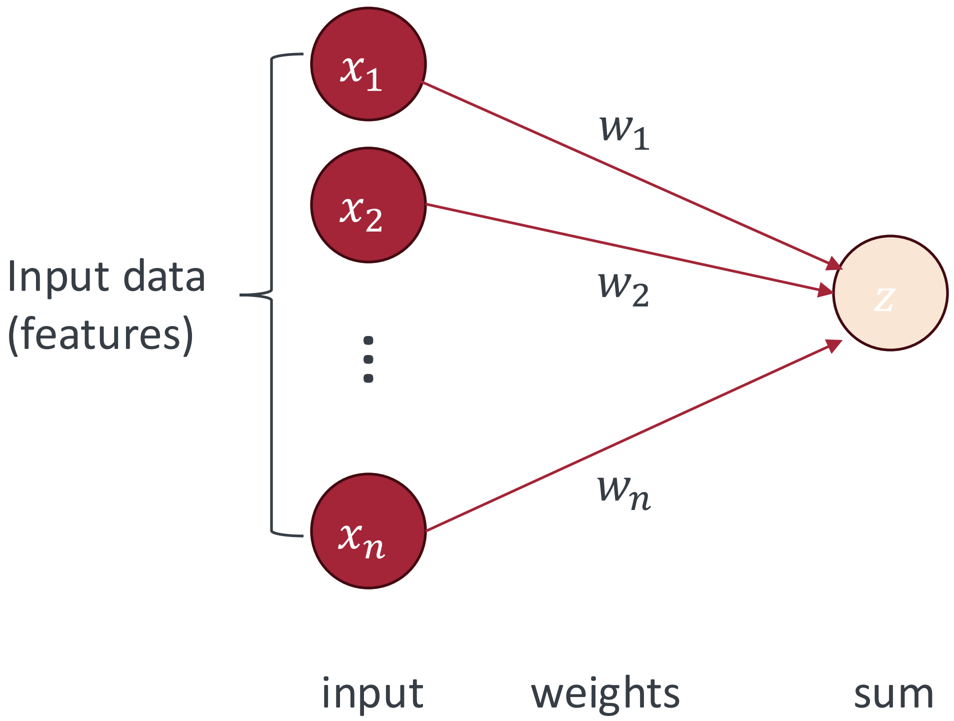
Introduction to the Perceptron

The base architecture



Introduction to the Perceptron

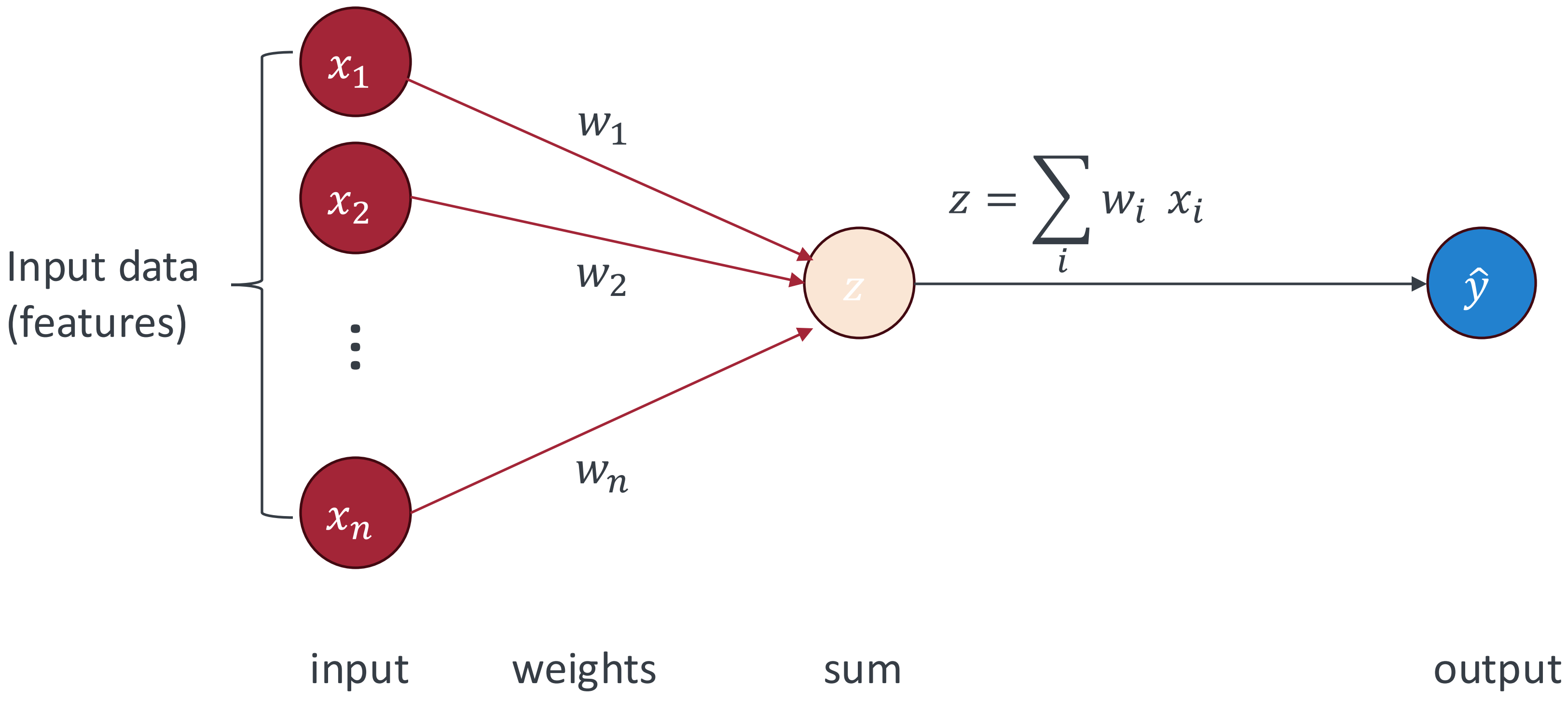
The base architecture



$$z = \sum_i w_i \; x_i$$

Introduction to the Perceptron

The base architecture



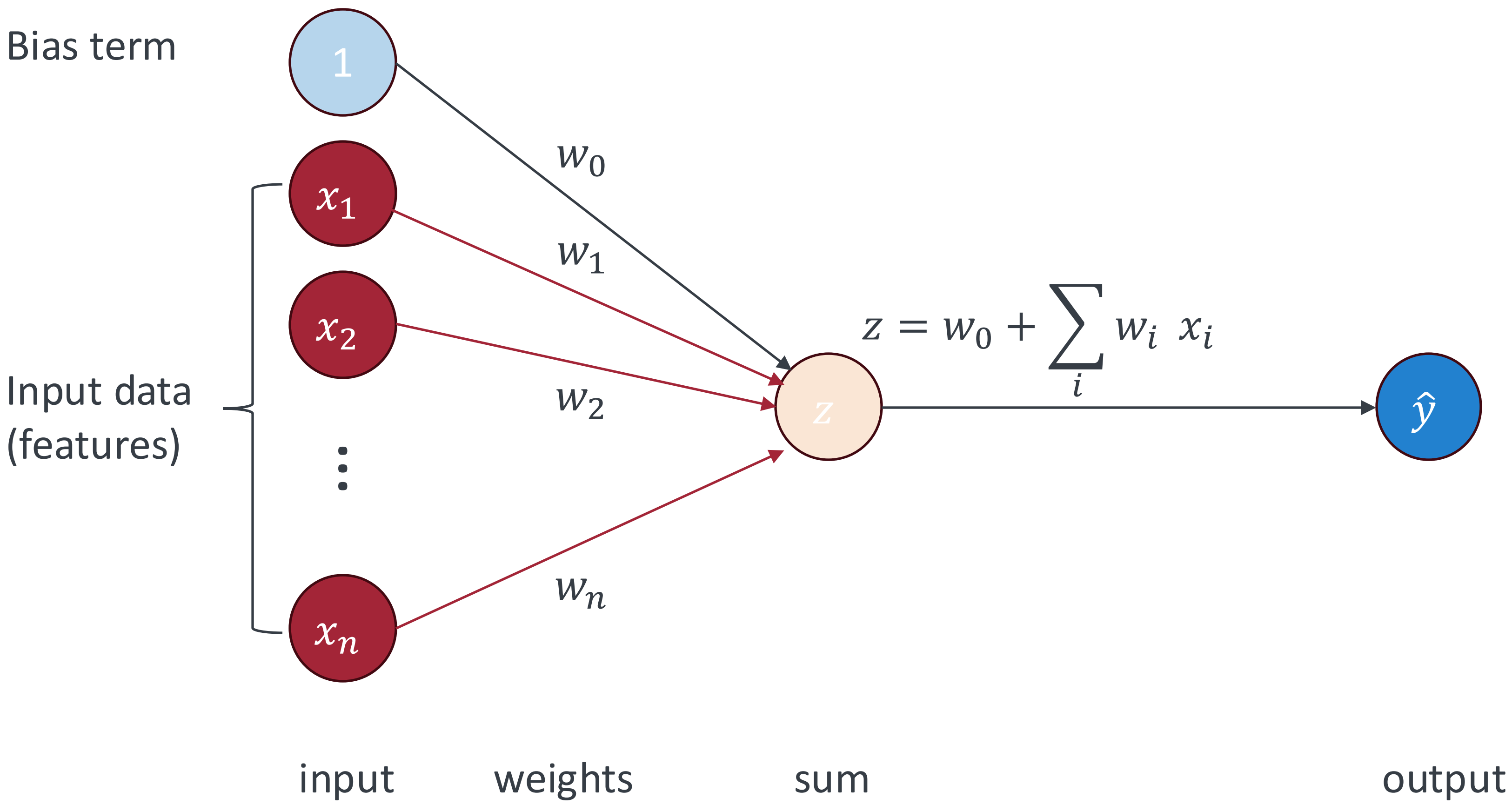
Linear combination of inputs

$$\hat{y} = \sum_i w_i \; x_i$$

Looks like a linear regression

Introduction to the Perceptron

The base architecture



Linear combination of inputs

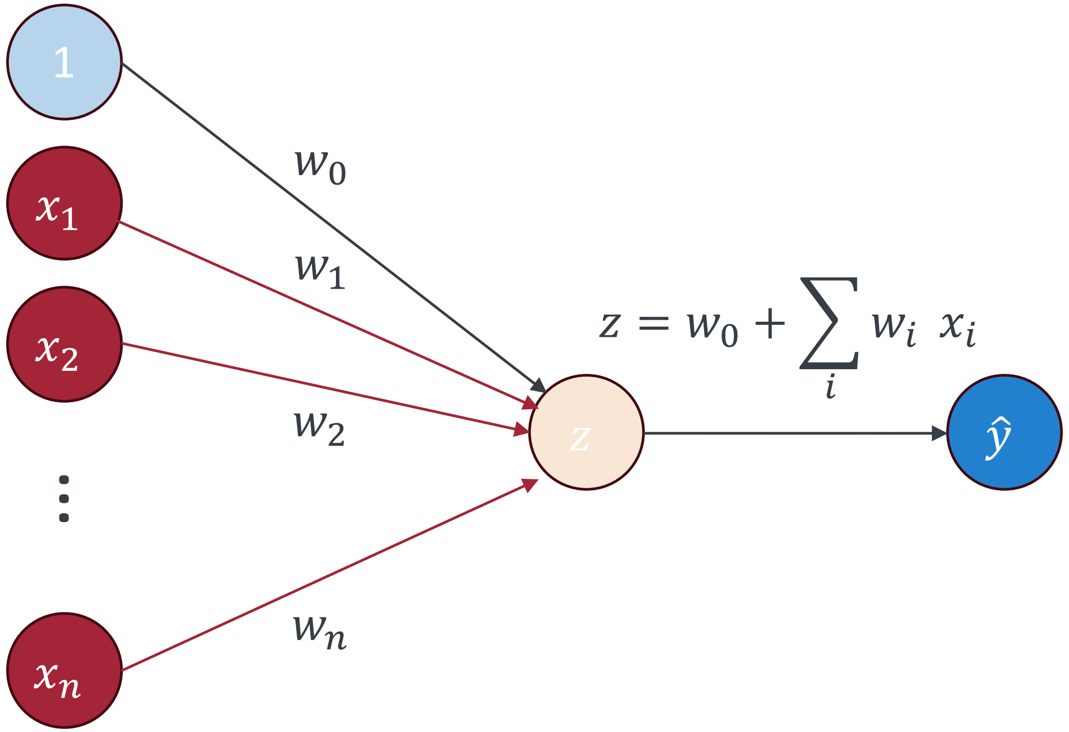
$$\hat{y} = w_0 + \sum_i w_i \ x_i$$

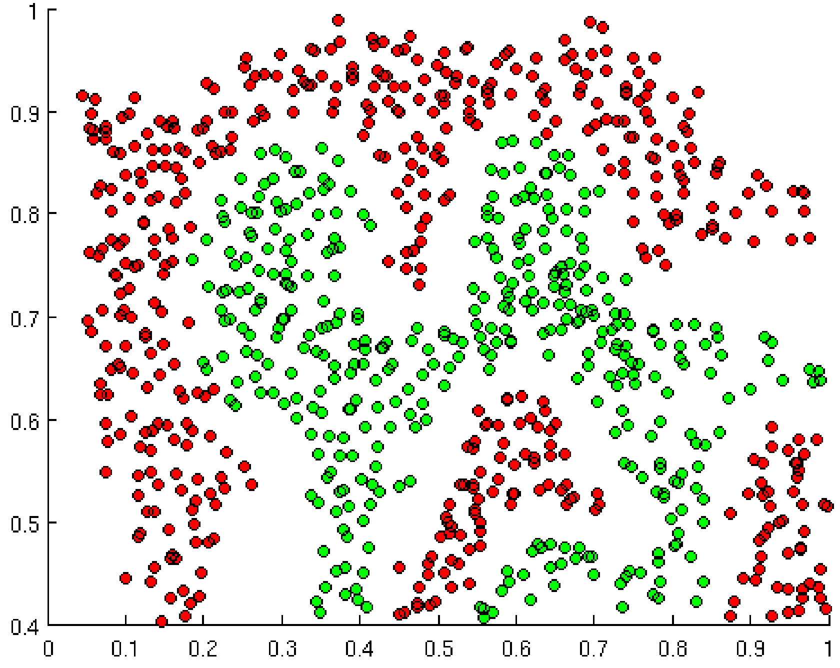
Bias (intercept)

So far, this resembles a linear regression

Introduction to the Perceptron

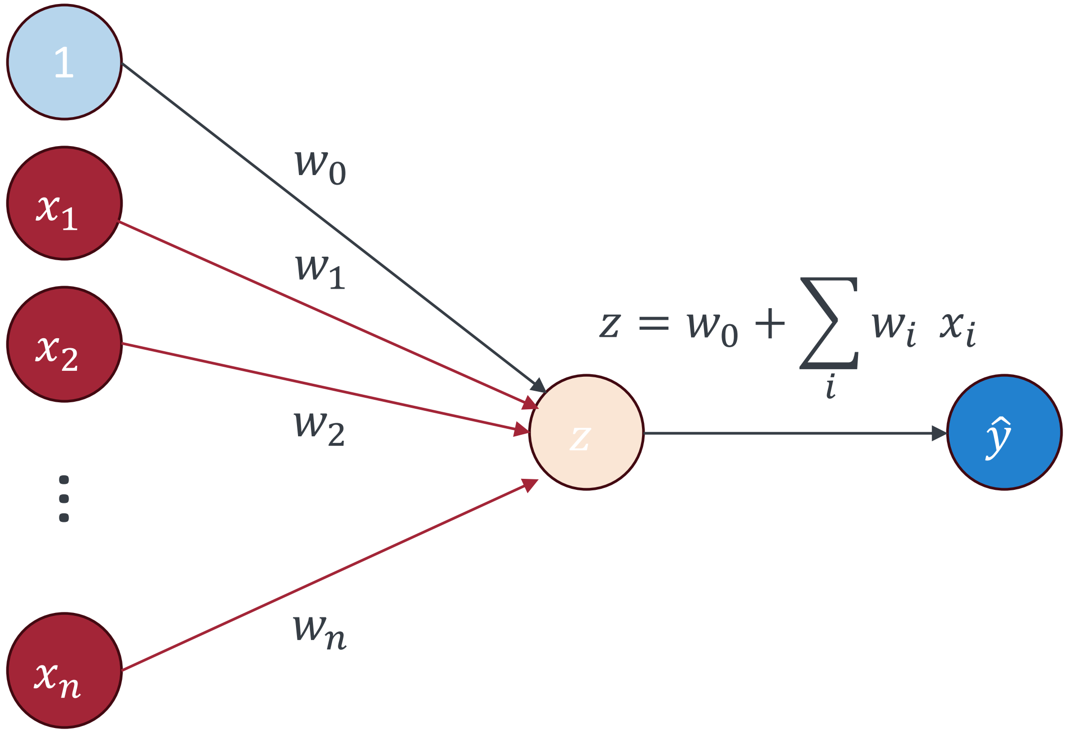
But how could we separate the red and green points?

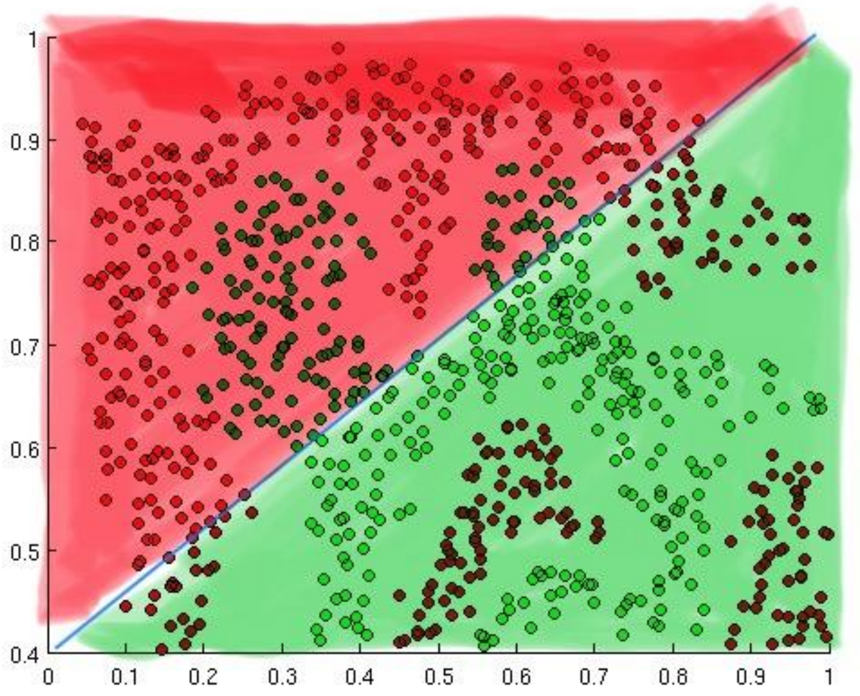




Introduction to the Perceptron

But how could we separate the red and green points?



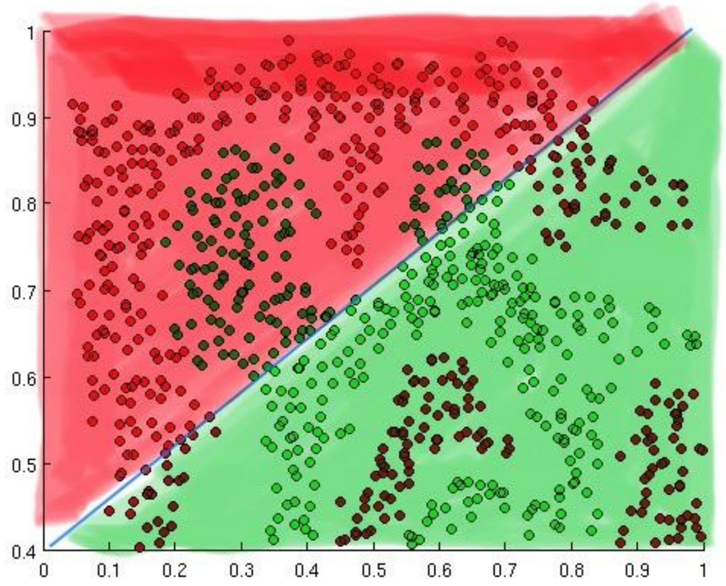


Linear functions result in linear decision boundaries, regardless of network size

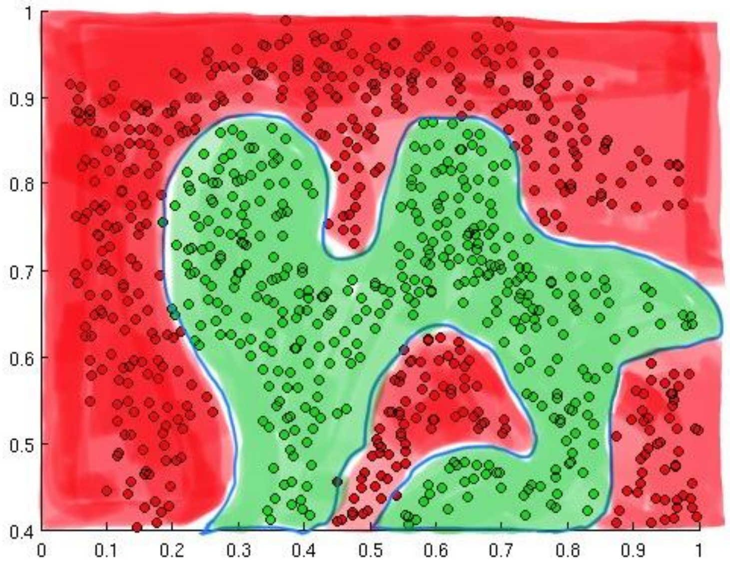
Introduction to the Perceptron

Why (nonlinear) activation functions?

$$z = w_0 + \sum_i w_i \ x_i \longrightarrow \hat{y} = g(z), \text{ where } g \text{ is a non-linear activation function.}$$



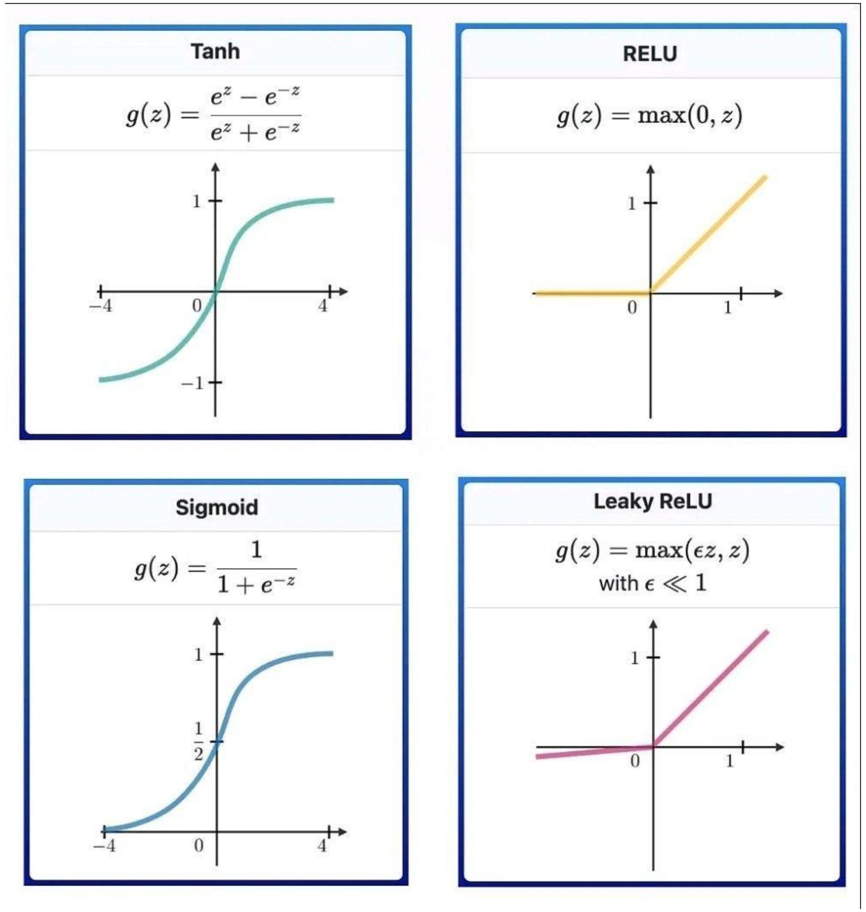
Linear functions result in linear decision boundaries, regardless of network size



Nonlinear activation functions enable the approximation of arbitrarily complex functions (nonlinear decision boundaries)

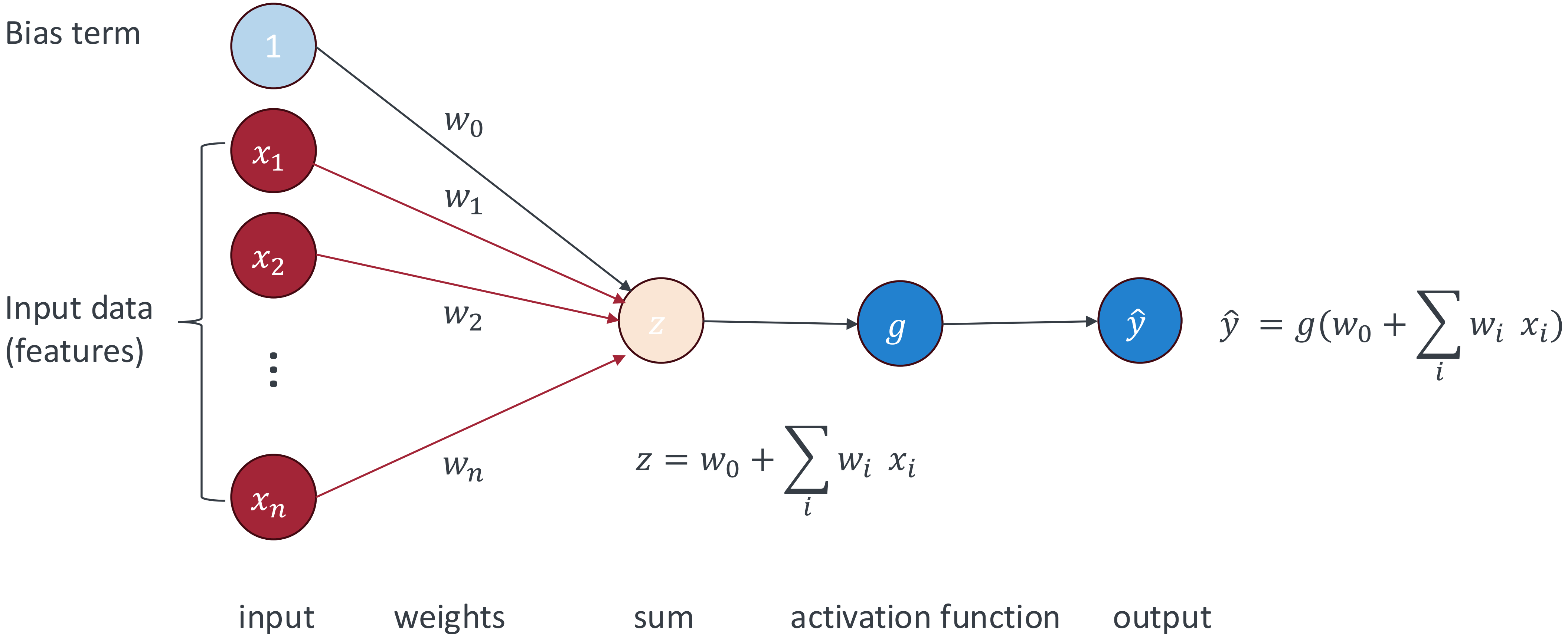
Introduction to the Perceptron

Classic activation functions



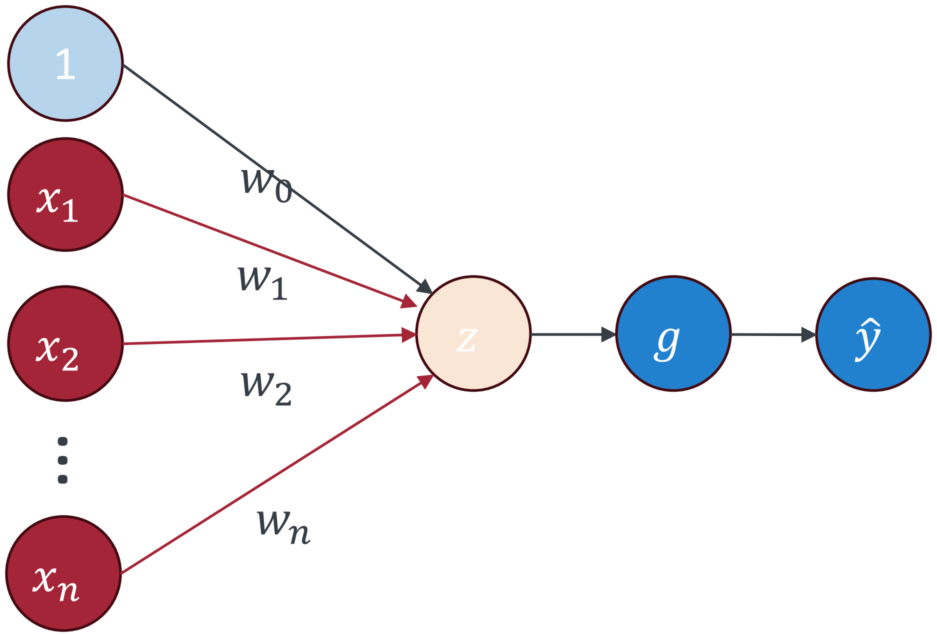
Introduction to the Perceptron

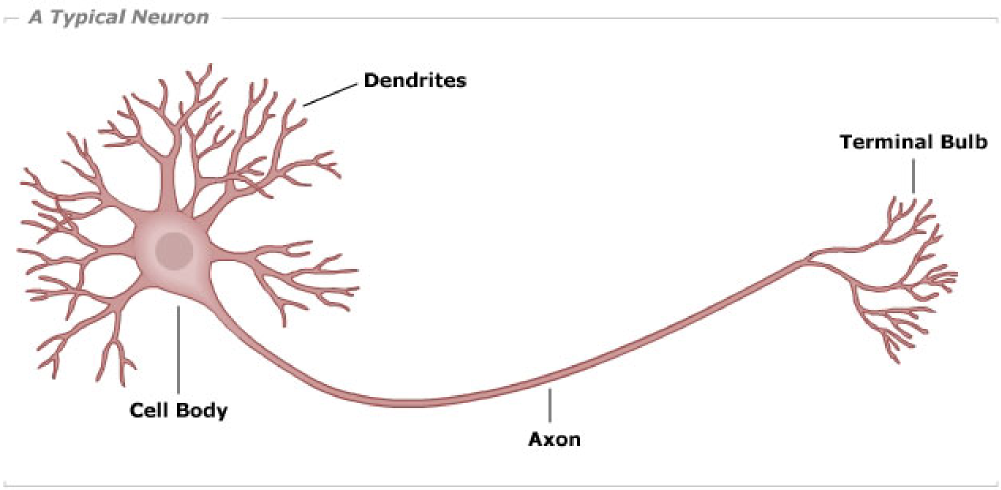
The full perceptron: forward propagation



Introduction to the Perceptron

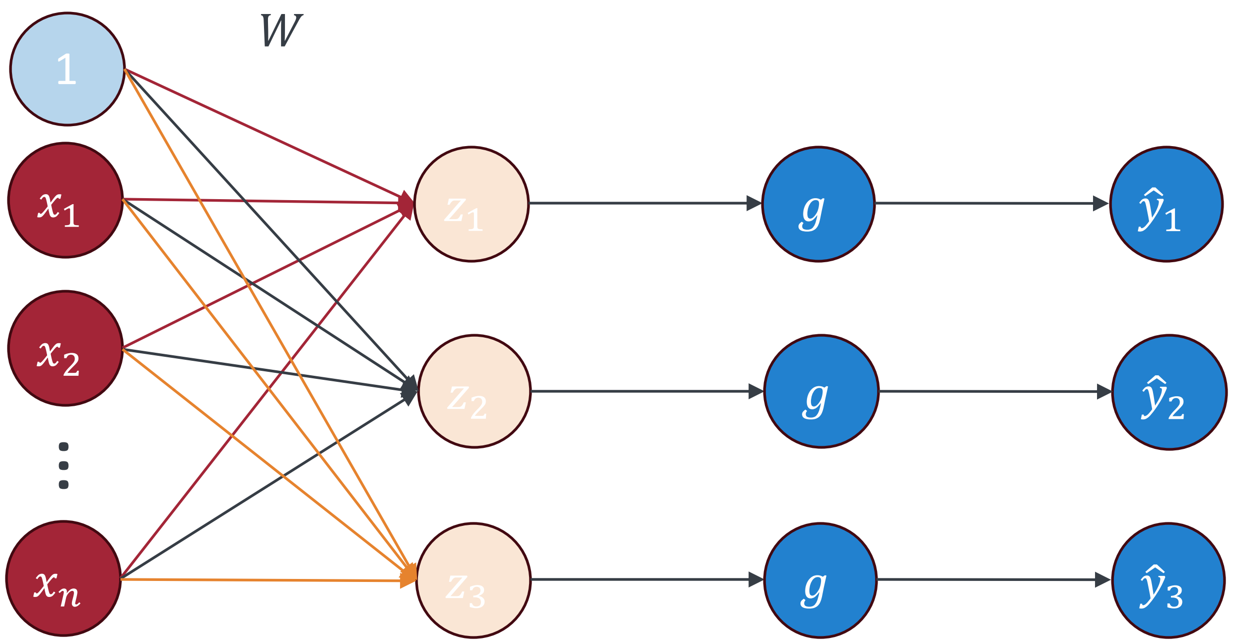
Artificial Neuron vs Brain Neuron





Neural Networks

Multi-output perceptron

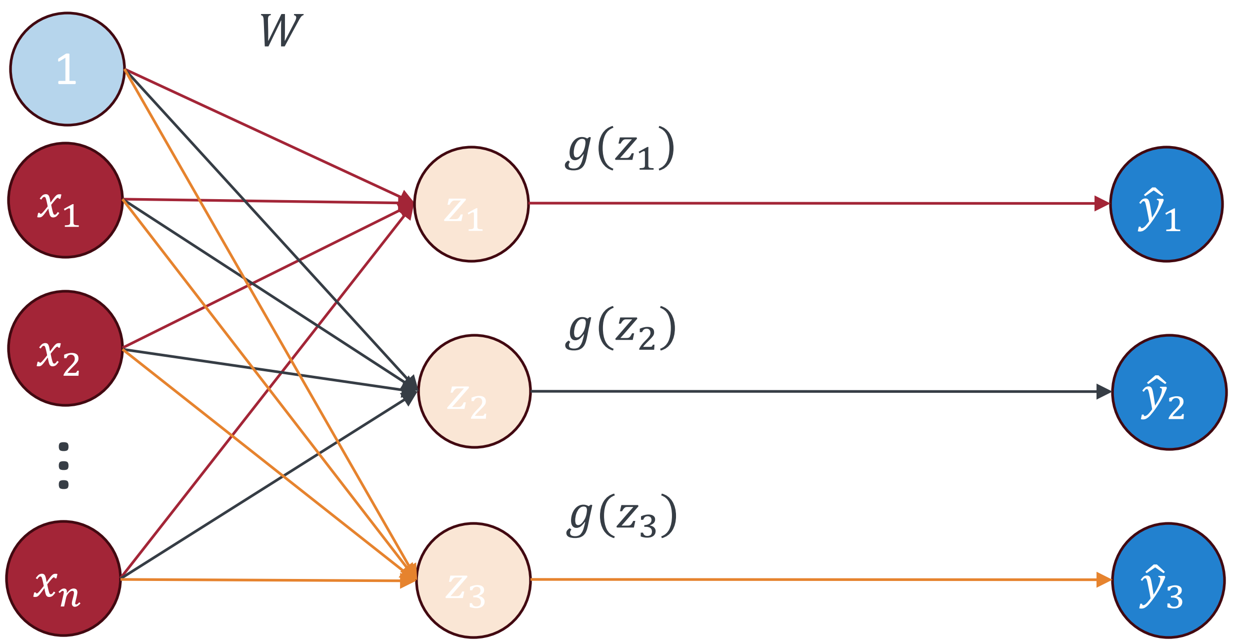


$$z_j = w_{0,j} + \sum_i w_{i,j} \; x_i$$

All inputs are densely connected to the outputs. We call this neural network structure a “dense layer.”

Neural Networks

Multi-output perceptron

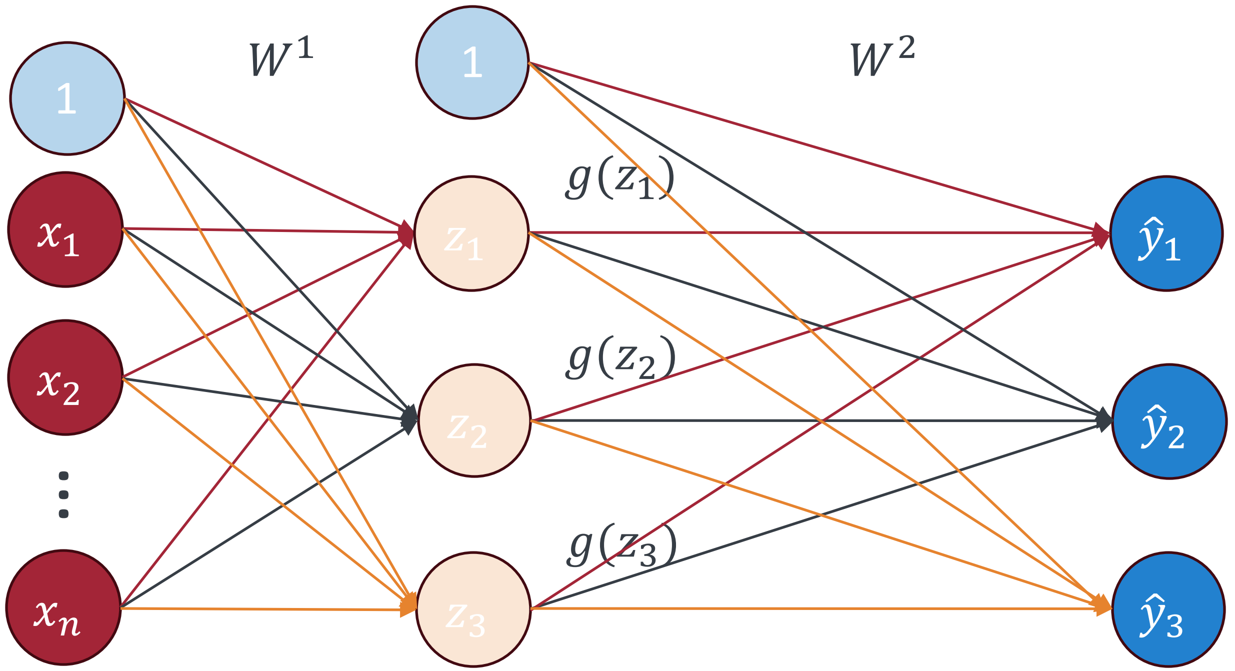


$$z_j = w_{0,j} + \sum_i w_{i,j} \; x_i$$

All inputs are densely connected to the outputs. We call this neural network structure a “dense layer.”

Neural Networks

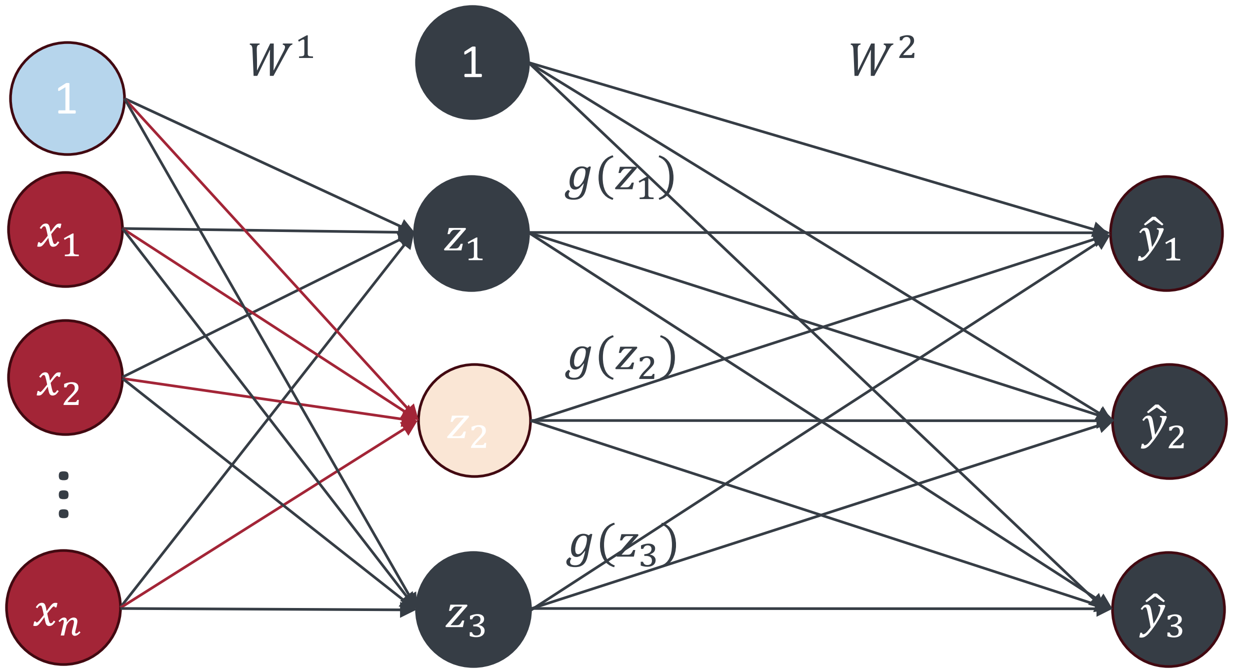
Single hidden layer network



$$z_j = w^1_{0,j} + \sum_i w^1_{i,j}\, x_i \qquad \hat{y}_j = g(w^2_{0,j} + \sum_i w^2_{i,j}\, g(z_i))$$

Neural Networks

Single hidden layer network

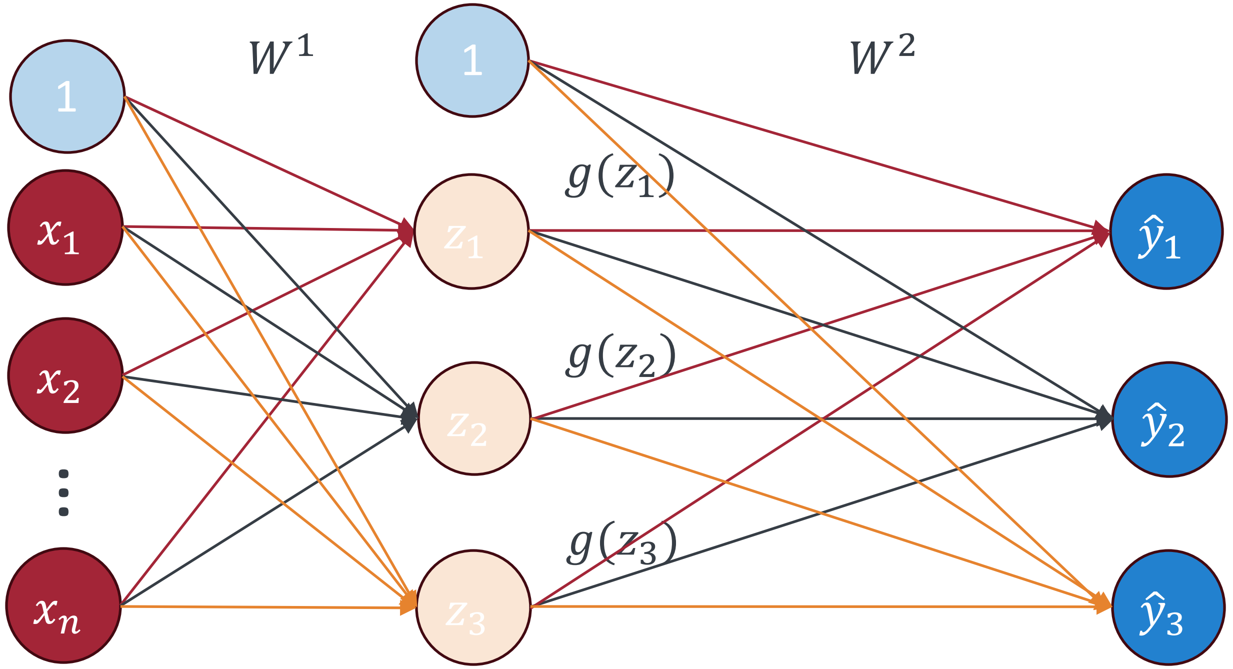


$$z_2 = w_{0,2}^1 + \sum_i w_{i,2}^1 x_i$$

$$z_2 = w_{0,2}^1 + w_{1,2}^1 x_1 + w_{2,2}^1 x_2 + \ldots + w_{n,2}^1 x_n$$

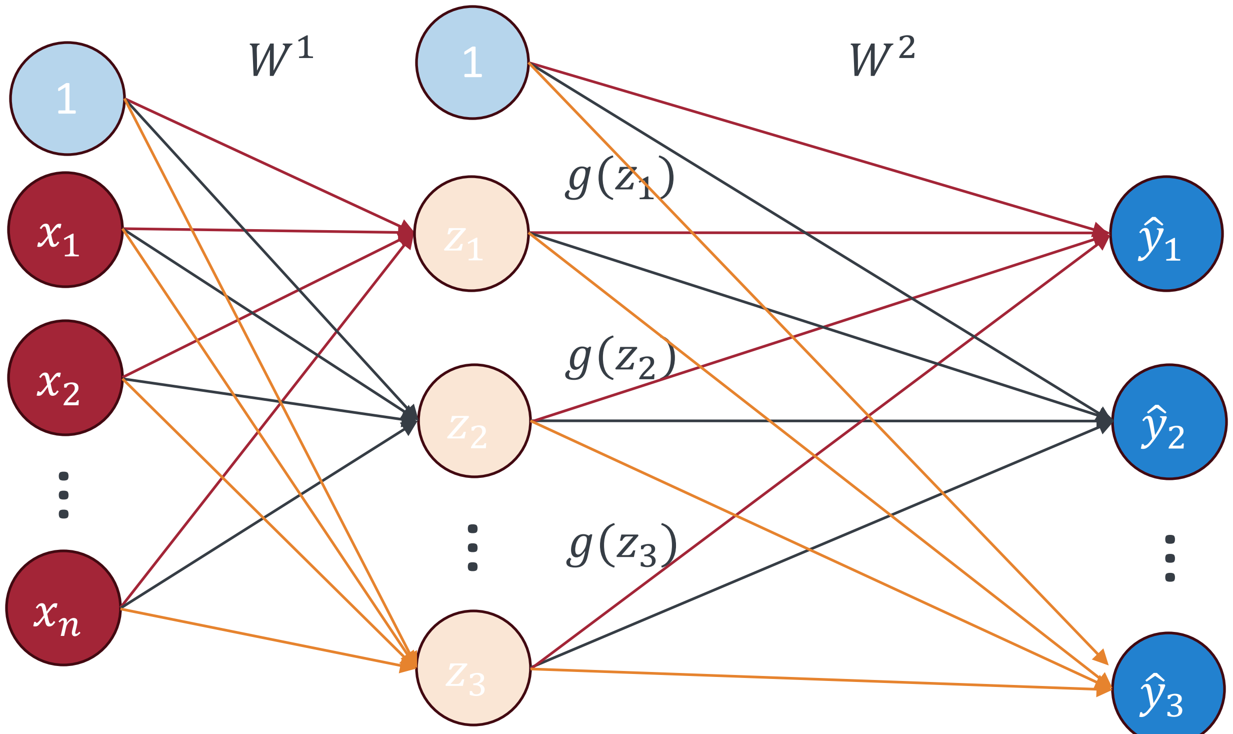
Neural Networks

Single hidden layer network



Neural Networks

Single hidden layer network



Activation Functions for the Output Layer

- Regression task: The final layer is just a single neuron with no activation

$$y = g(z) = z$$

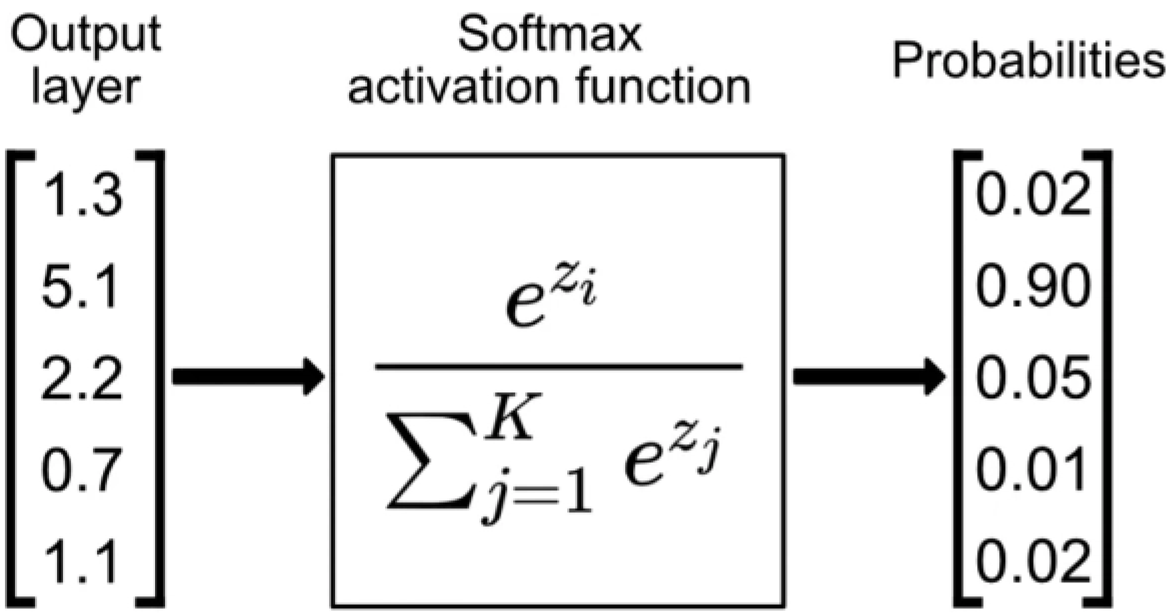
- Classification task with 2 classes: The final layer is a single neuron with the sigmoid activation function

$$y = g(z) = \mathrm{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

- Classification task with K classes: The final layer is composed of K neurons and a softmax function that combines their outputs through normalization

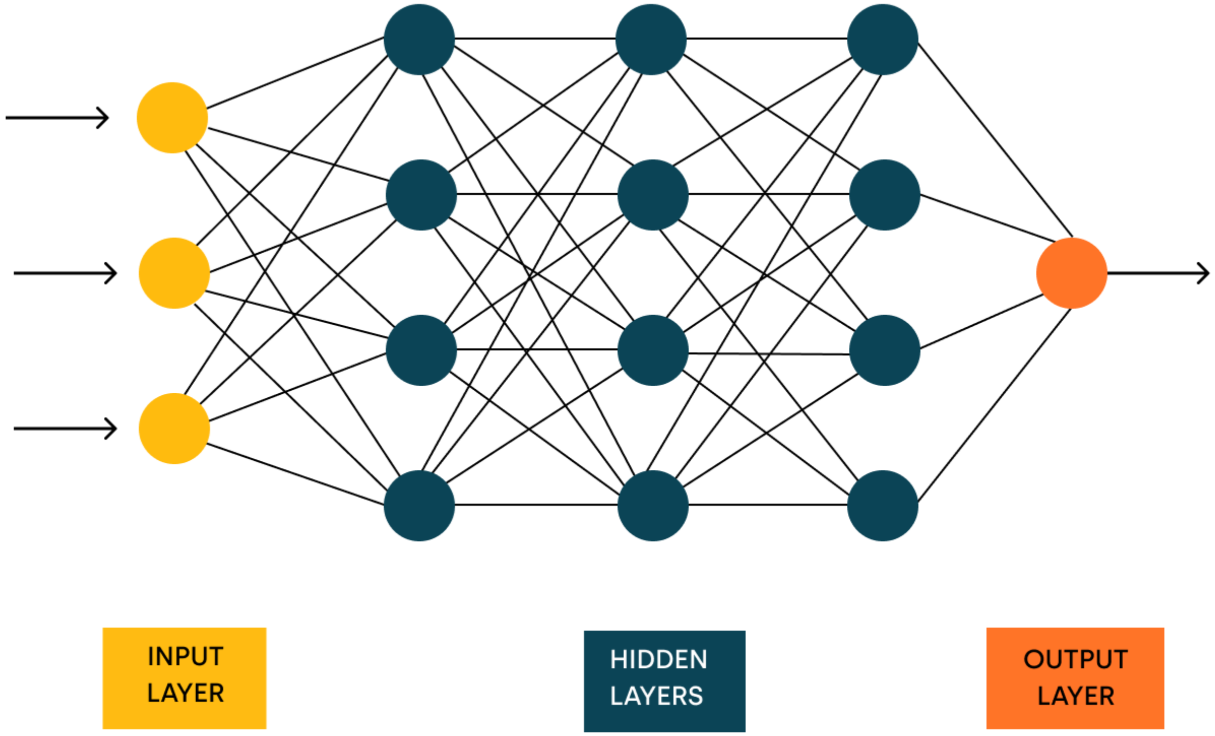
$$y_i = g(z_i) = \mathrm{softmax}(z_i) = \frac{1 + e^{-z_i}}{\sum_{j=1}^{K} 1 + e^{-z_j}}, i = \{1, \ldots, K\}$$

Softmax



Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

Deep Neural Network



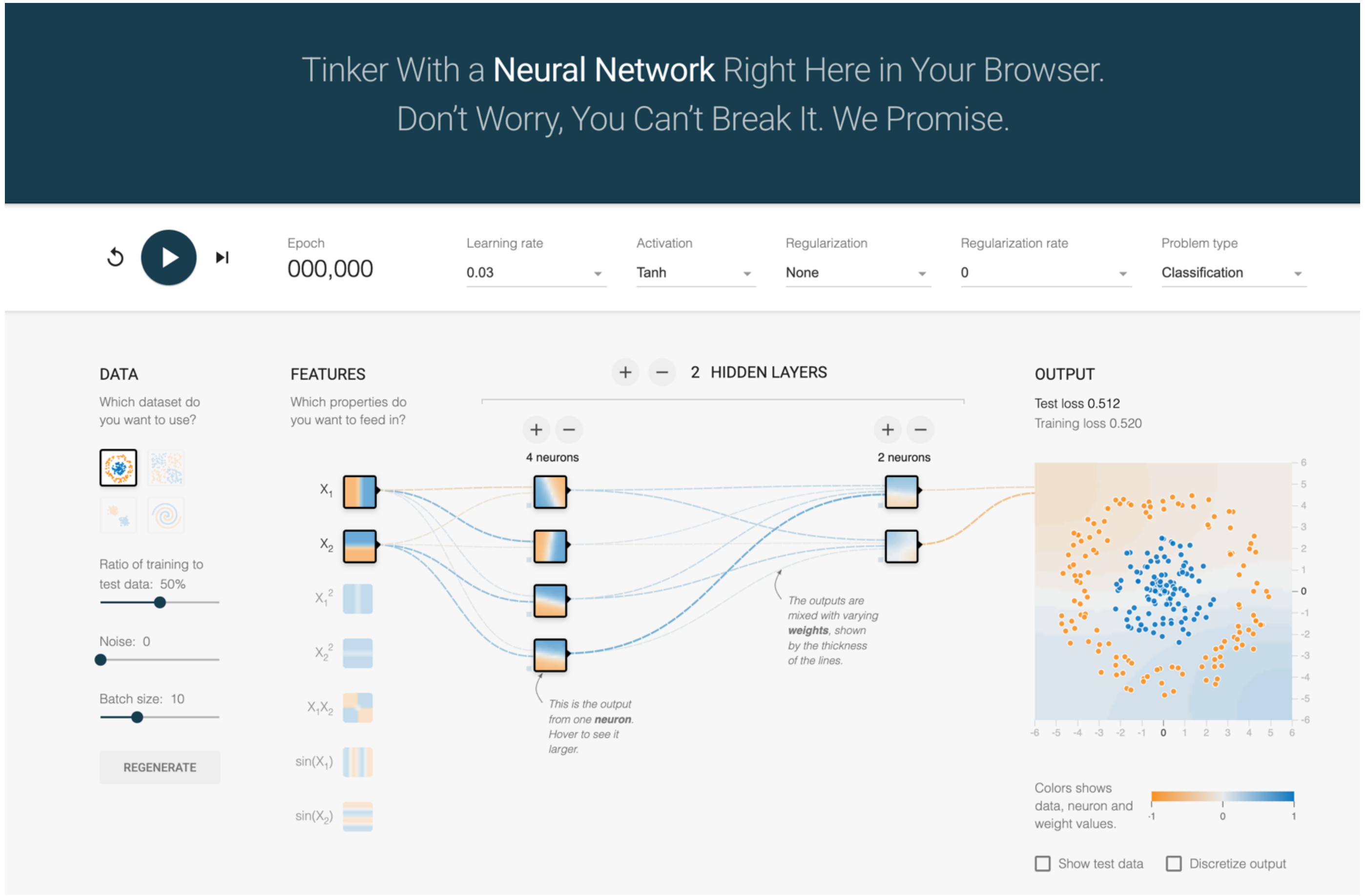
Layer dimension

$$z_{k,j} = w_{0,j}^{k} + \sum_{i=1}^{d_{k-1}} w_{i,j}^{k}\, g(z_{k-1,i})$$

Layer number

A Neural Network Playground

Link to the [Neural Network Playground]



Training a Neural Network

Loss Function

- Loss Function: $L(f(x^{(i)}; W), y^{(i)})$, where
  - $x^{(i)}$ are the inputs, $f(\cdot; W)$ is the neural network
  - $f(x^{(i)}; W)$ is the predicted value
  - $y^{(i)}$ is the actual value
  - The loss of a neural network measures the cost incurred from incorrect predictions
- Empirical Loss: $J(W) = \frac{1}{n}\sum_{i=1}^{n} L(f(x^{(i)}; W), y^{(i)})$
  - The empirical loss measures the total loss over the entire dataset
- Objective of training a neural network

$$W^* = \underset{W}{\operatorname{argmin}}\, J(W)$$

Common Loss Functions

- Regression ($f(x^{(i)}; W) \in \mathbb{R}$): Mean squared error (MSE)

$$J(W) = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - f(x^{(i)}; W))^2$$

- Classification: Cross-entropy loss
  - Binary classification ($p^{(i)} = f(x^{(i)}; W) \in (0,1)$): Binary cross-entropy loss

$$J(W) = -\frac{1}{n}\sum_{i=1}^{n}[y^{(i)}\log(p^{(i)}) + (1 - y^{(i)})\log(p^{(i)})]$$

  - Multi-class classification ($p_c^{(i)} = f(x^{(i)}; W)_c \in (0,1)$): Categorical cross-entropy loss

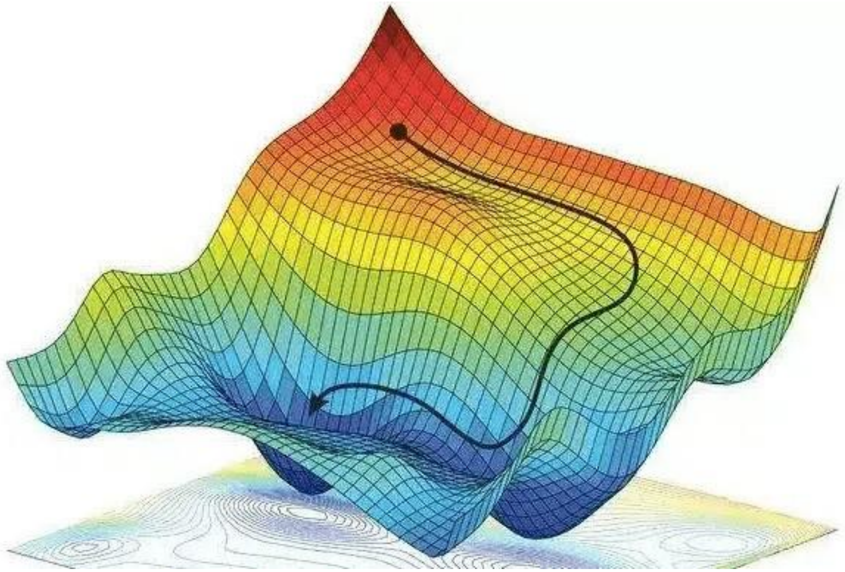
$$J(W) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_c^{(i)}\log p_c^{(i)}$$

  - Multi-label classification: Each input belong to multiple classes simultaneously (e.g., image classification in self-driving)

$$J(W) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C}[y_c^{(i)}\log p_c^{(i)} + (1 - y_c^{(i)})\log\left(1 - p_c^{(i)}\right)]$$

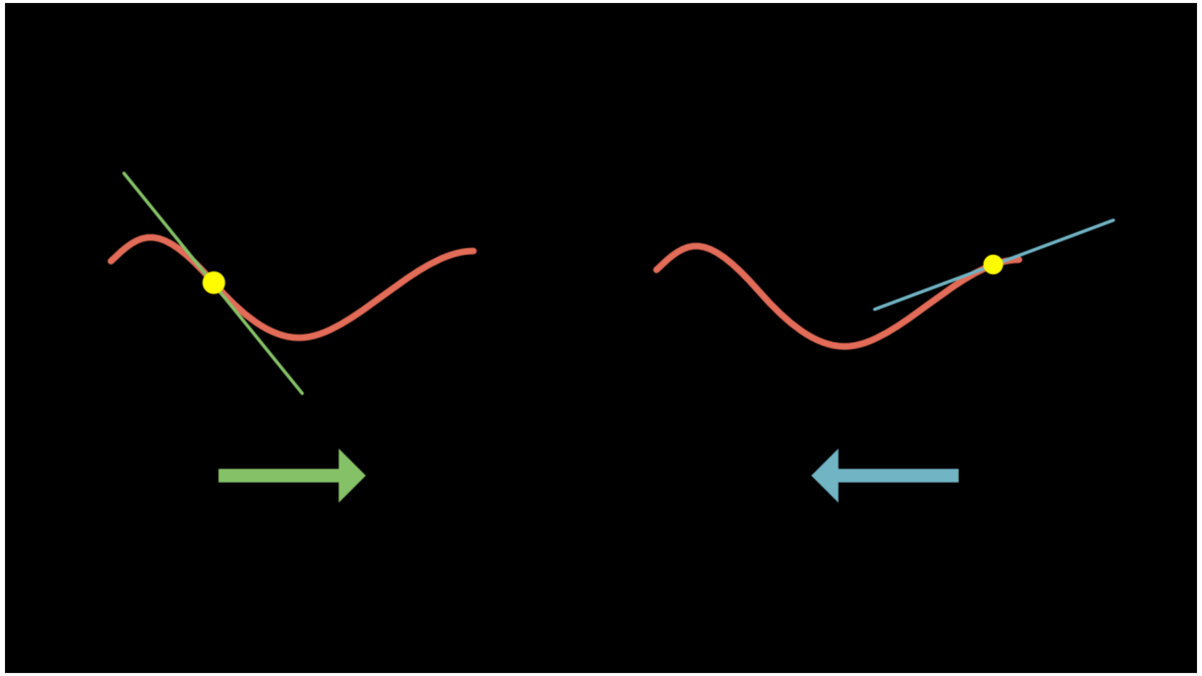
Optimization

- Objective of training a neural network
$$W^* = \underset{W}{\operatorname{argmin}}\, J(W)$$
- The loss function is a function of the network weights
- If we had only 2 weights, we could plot the loss function with respect to the different combinations of weights



Gradient

- The gradient of J at the point W_0, denoted as $\nabla_W J(W_0)$, is the direction to move in for the fastest increase in $J(W)$, when starting from W_0
- Thus, the opposite direction of the gradient points to the fastest decrease in $J(W)$



Gradient Descent

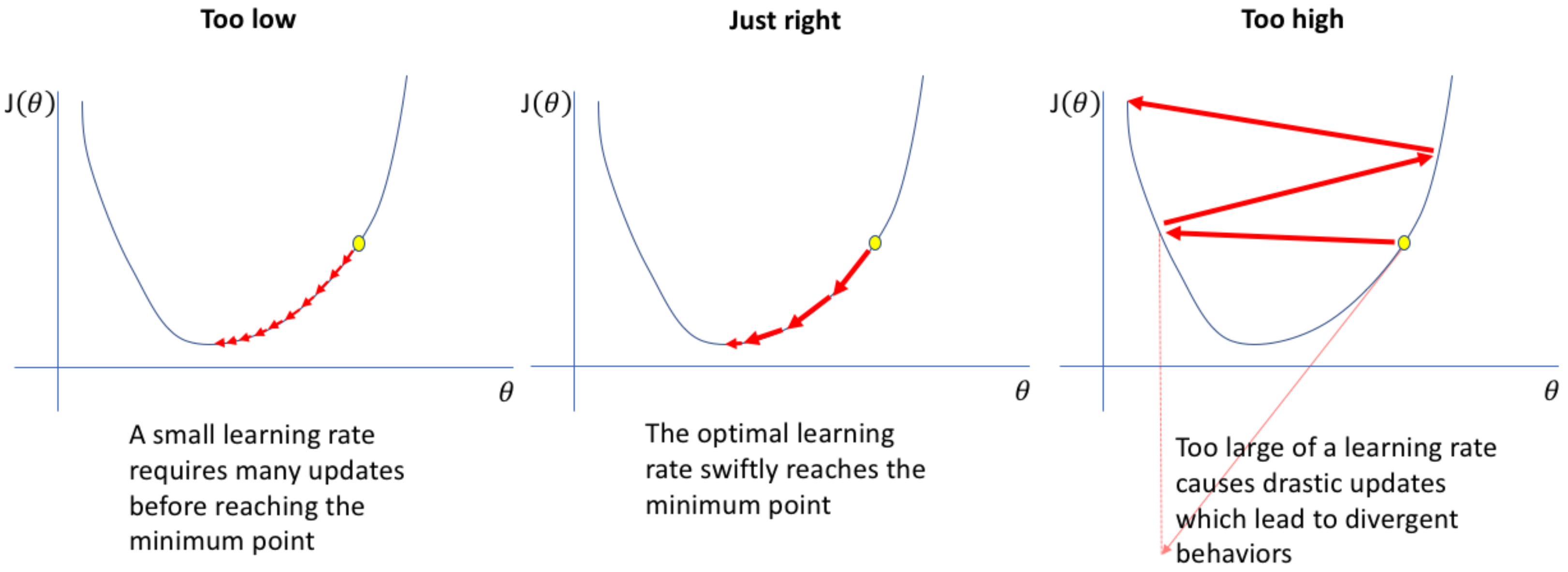
- Initialize W_0 randomly
- For i from 0 to M:

$$W_{i+1} = W_i - \eta \nabla_W J(W_i)$$

  - η is the step size, also called the learning rate
  - M is the maximum number of iterations
- The algorithm continues until the stopping condition
  - When $\| \nabla_W J(W) \|_2 \leq \varepsilon$ for some pre-set ε (Recall $\nabla_W J(W) = 0$ when function $J(W)$ is at minimum)
  - Evaluate the performance on validation data, and stop when not improving

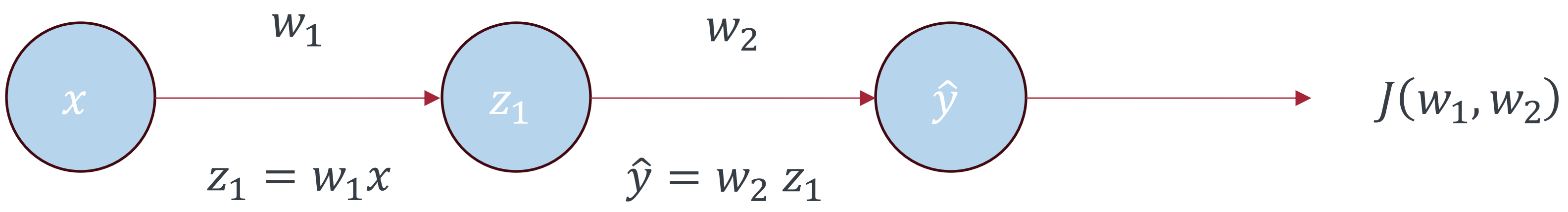
Learning Rate

- Step size or learning rate (η) is a hyperparameter
- Small η leads to slow convergence, but large η leads to divergence
- Different step sizes have to be tried



Backpropagation

Forward and backward pass: Chain rule



- Let's consider a simple case with two weight parameters, $W = (w_1, w_2)$
- Gradient for w_2

$$\frac{\partial J(w_1, w_2)}{\partial w_2} = \frac{\partial J(w_1, w_2)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

- Gradient for w_1

$$\frac{\partial J(w_1, w_2)}{\partial w_1} = \frac{\partial J(w_1, w_2)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

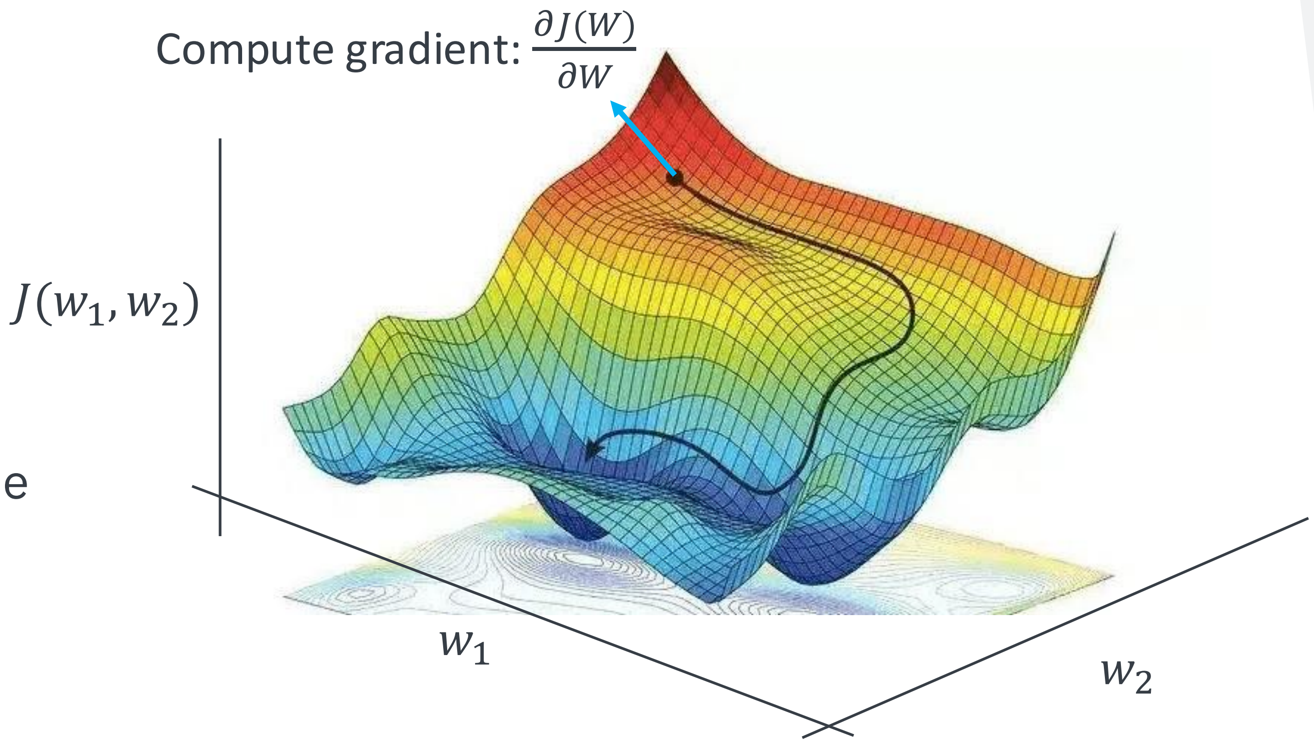
- PyTorch has the autograd function that computes the gradient automatically
- YouTube video on autograd from scratch by Andrej Karpathy [Link, Credit to Hitanshu]

Training Neural Networks with Gradient Descent

Objective: $W^* = \underset{W}{\operatorname{argmin}} J(W)$, where $J(W) = \frac{1}{n}\sum_{i=1}^{n} L\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$

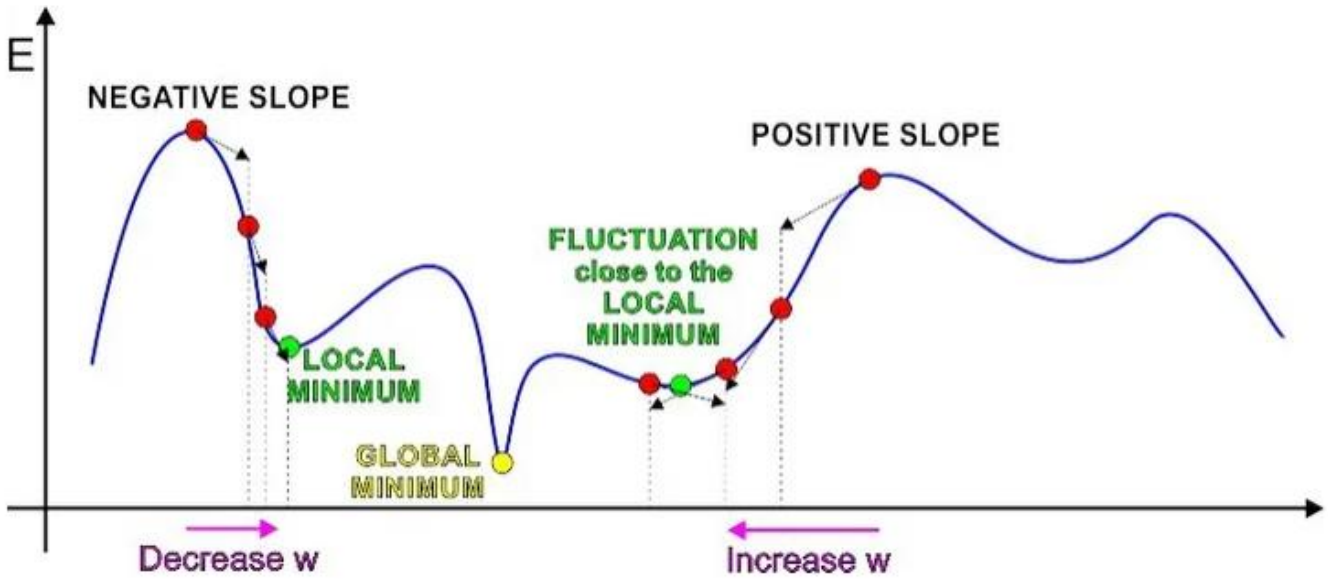
Algorithm

1. Initialize weight matrix W
2. Backpropagation: Loop until convergence
   a) Forward pass: Compute the predictions and the loss
   b) Backward pass: Compute the gradient with the chain rule
   c) Update the weights with the learning rate
3. Return the weight matrix W and the predictions



Local and Global Minimum

- Convergence is not guaranteed, and one may get stuck in a local minimum [Visualization]
- Smaller batch size and momentum help escape from critical points, which have zero gradients



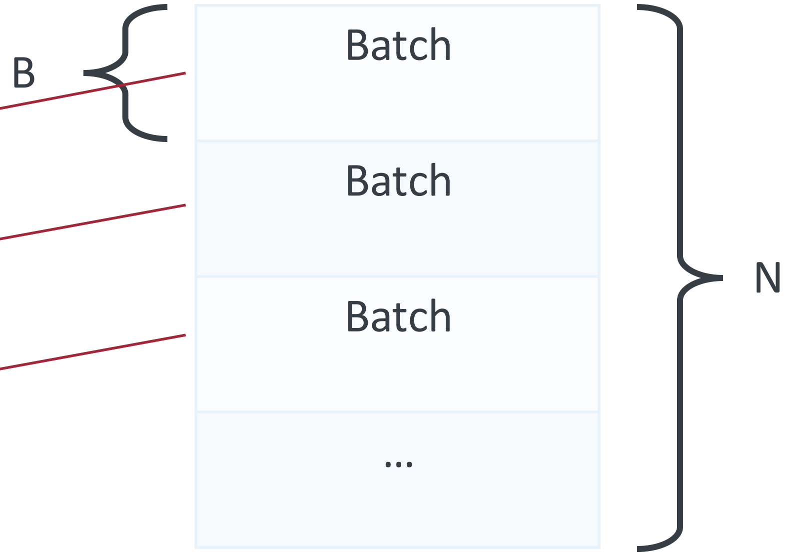
Optimization with Batch

- Batch: A batch is a small subset of the training data used to update the model weights during training
  - Typically, we sample the batches at random (without replacement)
- Optimization with batch
  - Initialize W_0 randomly
  - Compute gradient $\nabla_W J^1(W_0)$

    Update $W_1 = W_0 - \eta \nabla_W J^1(W_0)$
  - Compute gradient $\nabla_W J^2(W_1)$

    Update $W_2 = W_1 - \eta \nabla_W J^2(W_1)$
  - Compute gradient $\nabla_W J^3(W_2)$

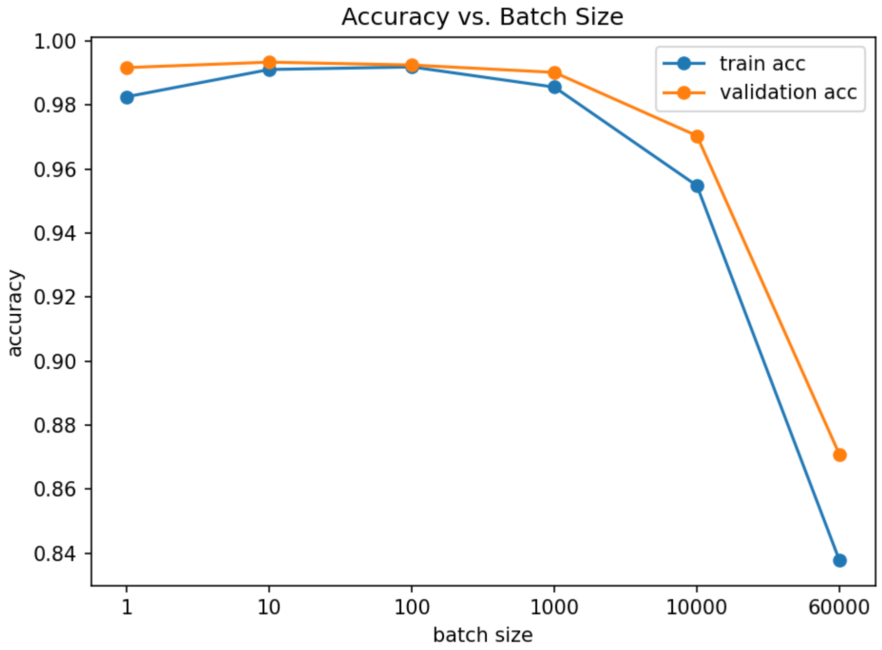
    Update $W_3 = W_2 - \eta \nabla_W J^3(W_2)$
  - …



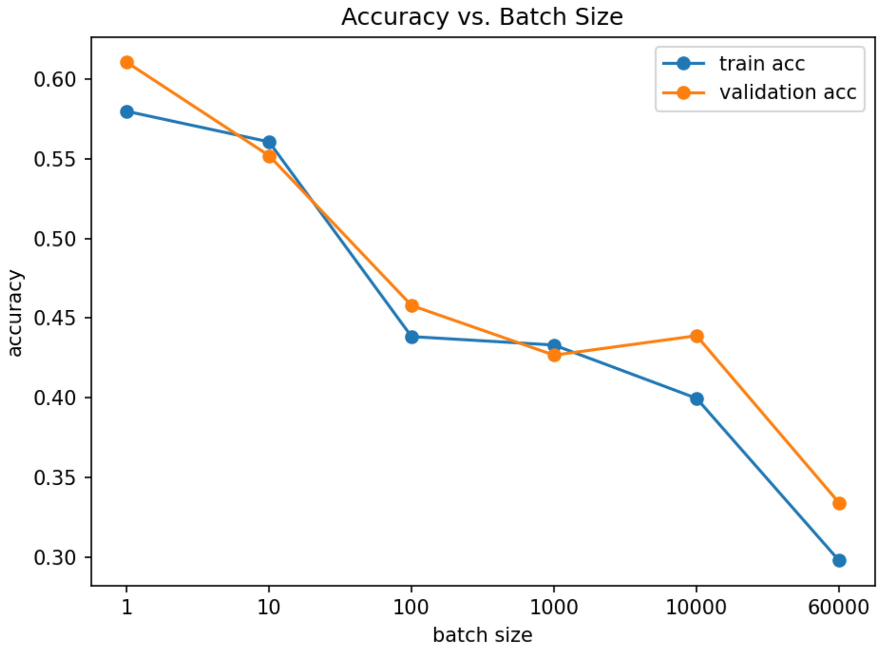
- Epoch: An epoch is one complete pass through the entire training dataset
- Shuffle the batches after each epoch

Small Batch vs. Large Batch

MNIST

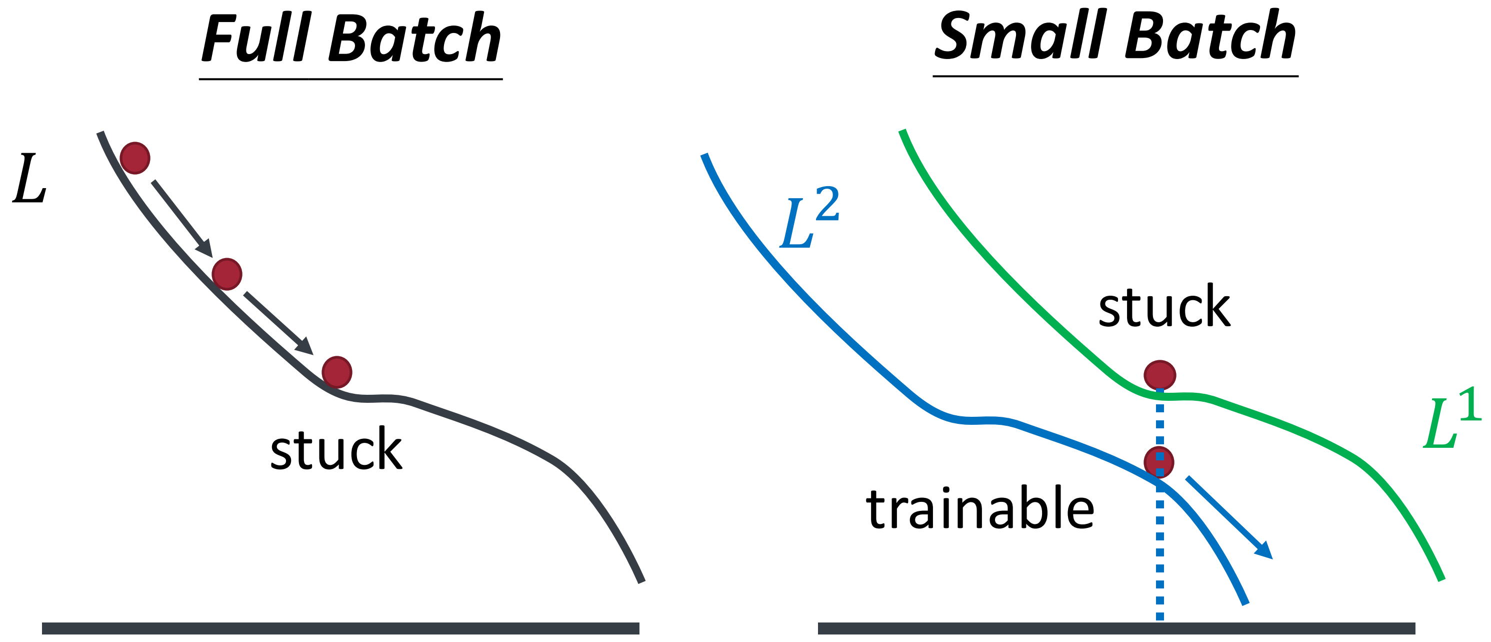


CIFAR-10



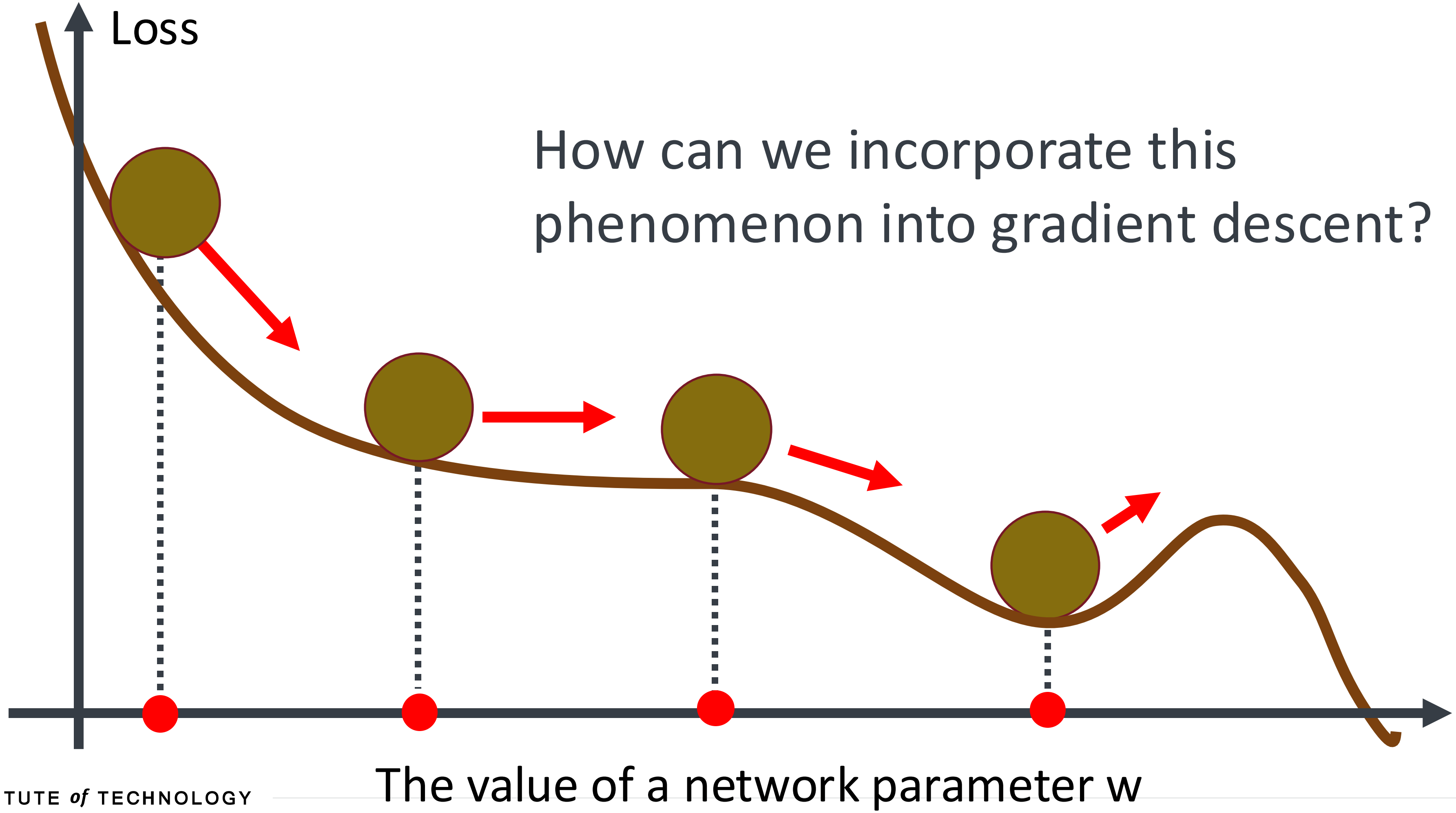
- **MNIST** refers to the hand-written digit classification; **CIFAR-10** refers to the 10-class image classification
- Smaller batch size has better performance
- What's wrong with large batch size?

Small Batch vs. Large Batch



Momentum: Adaptive Learning Rate

Physical world: Movement has momentum



How can we incorporate this phenomenon into gradient descent?

Momentum: Adaptive Learning Rate

Gradient Descent

Starting at $\boldsymbol{\theta}^0$

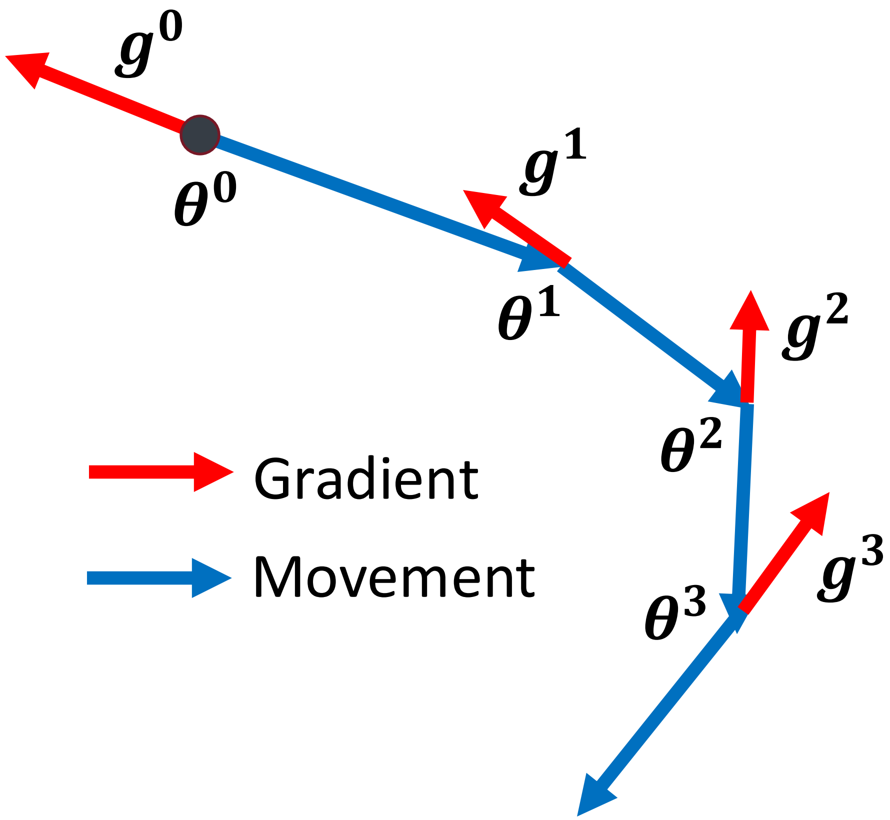
Compute gradient $\boldsymbol{g}^0$

Move to $\boldsymbol{\theta}^1 = \boldsymbol{\theta}^0 - \eta \boldsymbol{g}^0$

Compute gradient $\boldsymbol{g}^1$

Move to $\boldsymbol{\theta}^2 = \boldsymbol{\theta}^1 - \eta \boldsymbol{g}^1$

⋮



Momentum: Adaptive Learning Rate

Gradient Descent + Momentum

Movement: **movement of last step** minus **gradient** at present

Starting at $\boldsymbol{\theta}^0$

Movement $\boldsymbol{m}^0 = \mathbf{0}$

Compute gradient $\boldsymbol{g}^0$

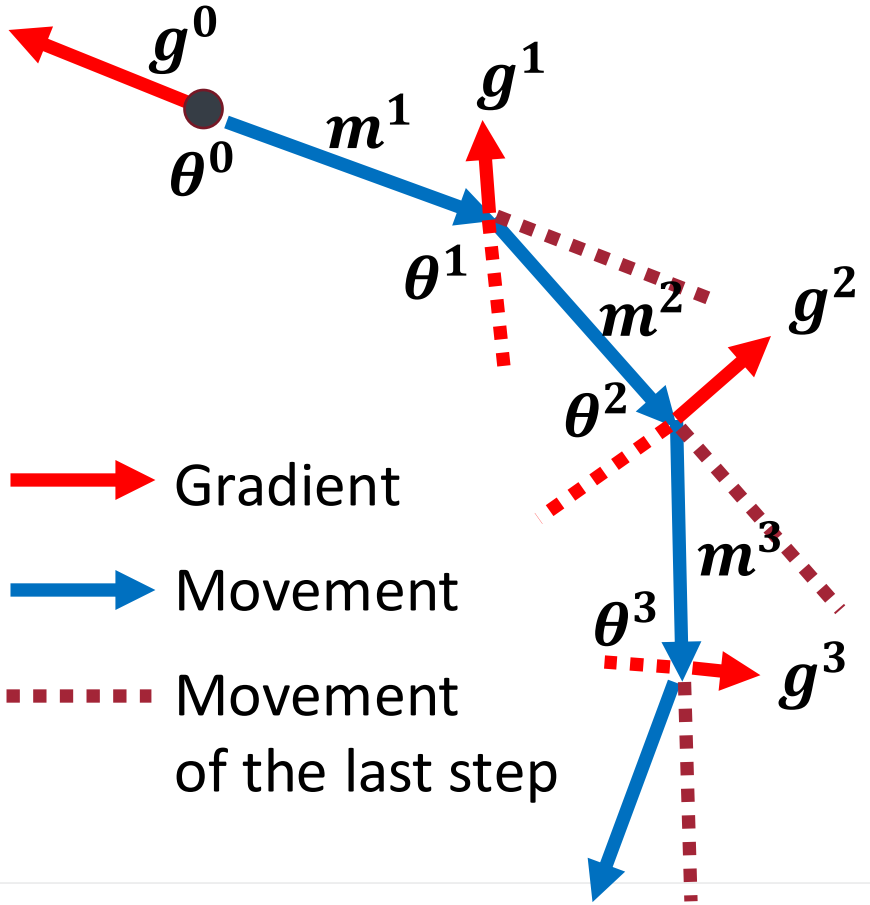
Movement $\boldsymbol{m}^1 = \lambda \boldsymbol{m}^0 - \eta \boldsymbol{g}^0$

Move to $\boldsymbol{\theta}^1 = \boldsymbol{\theta}^0 + \boldsymbol{m}^1$

Compute gradient $\boldsymbol{g}^1$

Movement $\boldsymbol{m}^2 = \lambda \boldsymbol{m}^1 - \eta \boldsymbol{g}^1$

Move to $\boldsymbol{\theta}^2 = \boldsymbol{\theta}^1 + \boldsymbol{m}^2$



Momentum: Adaptive Learning Rate

Gradient Descent + Momentum

Movement: **movement of last step** minus **gradient** at present

Starting at $\boldsymbol{\theta}^0$

Movement $\boldsymbol{m}^0 = \boldsymbol{0}$

Compute gradient $\boldsymbol{g}^0$

Movement $\boldsymbol{m}^1 = \lambda\boldsymbol{m}^0 - \eta\boldsymbol{g}^0$

Move to $\boldsymbol{\theta}^1 = \boldsymbol{\theta}^0 + \boldsymbol{m}^1$

Compute gradient $\boldsymbol{g}^1$

Movement $\boldsymbol{m}^2 = \lambda\boldsymbol{m}^1 - \eta\boldsymbol{g}^1$

Move to $\boldsymbol{\theta}^2 = \boldsymbol{\theta}^1 + \boldsymbol{m}^2$

$\boldsymbol{m}^i$ is the weighted sum of all the previous gradient: $\boldsymbol{g}^0, \boldsymbol{g}^1, \ldots, \boldsymbol{g}^{i-1}$

$$\boldsymbol{m}^0 = \boldsymbol{0}$$

$$\boldsymbol{m}^1 = -\eta\boldsymbol{g}^0$$

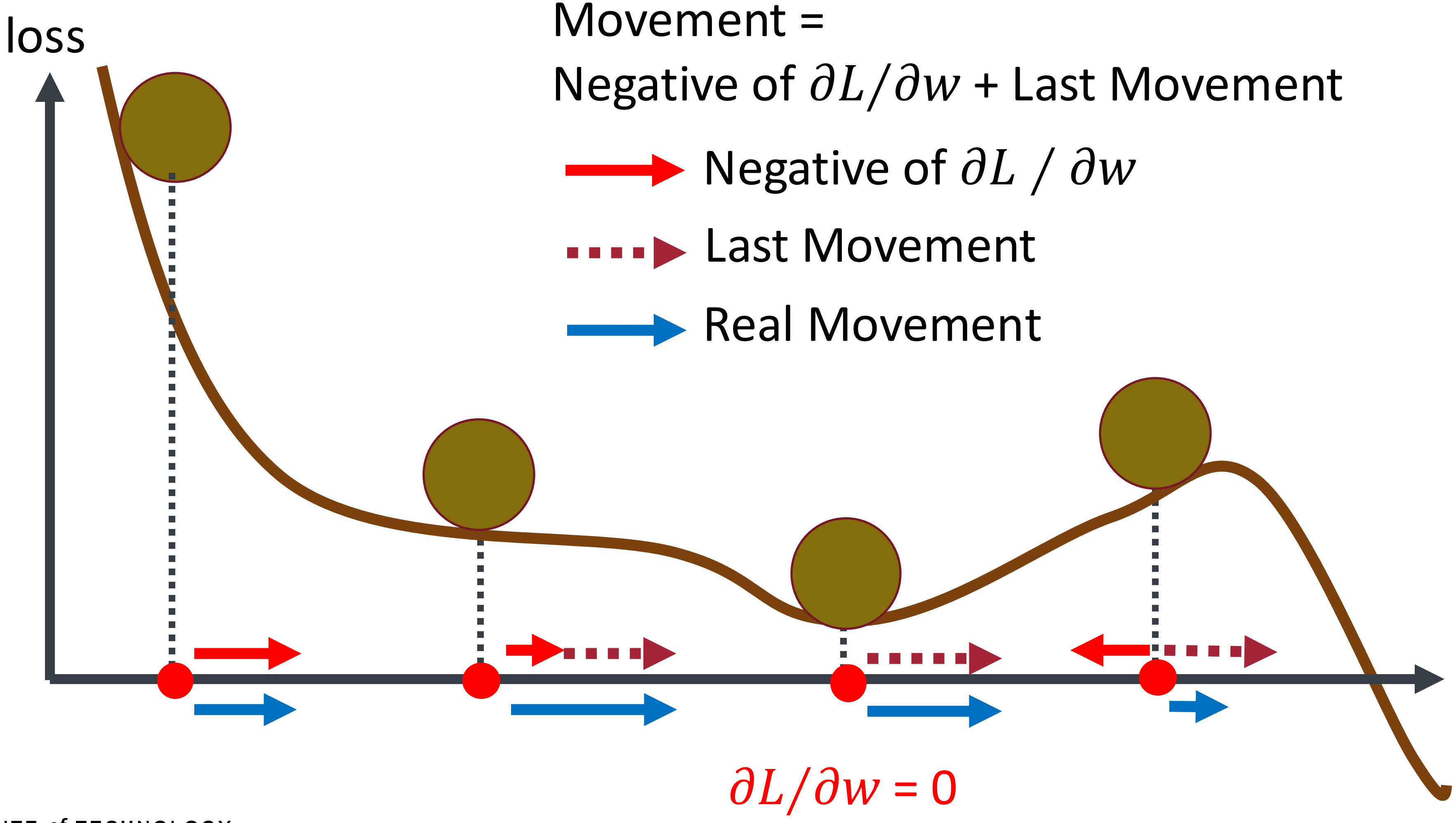
$$\boldsymbol{m}^2 = -\lambda\eta\boldsymbol{g}^0 - \eta\boldsymbol{g}^1$$

$$\vdots$$

Momentum: Adaptive Learning Rate

Gradient Descent + Momentum

Movement =
Negative of $\partial L/\partial w$ + Last Movement



Local and Global Minimum

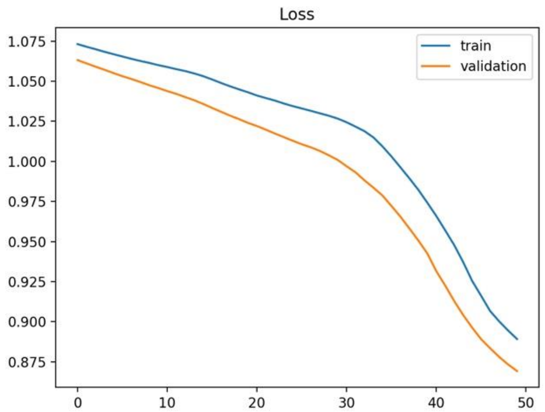
- Critical points have zero gradients
- Critical points can be either saddle points or local minima
- Smaller batch size and momentum help escape critical points
- Adam is a very popular optimizer that does all this automatically

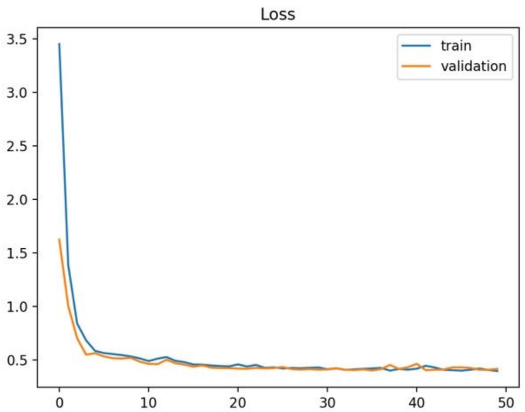
Adam: A method for stochastic optimization
DP Kingma, J Ba - arXiv preprint arXiv:1412.6980, 2014 - arxiv.org
… **Adam** works well in practice and compares favorably to other stochastic optimization methods. Finally, we discuss AdaMax, a variant of **Adam** … Overall, we show that **Adam** is a versatile …
Save Cite Cited by 202011 Related articles All 19 versions

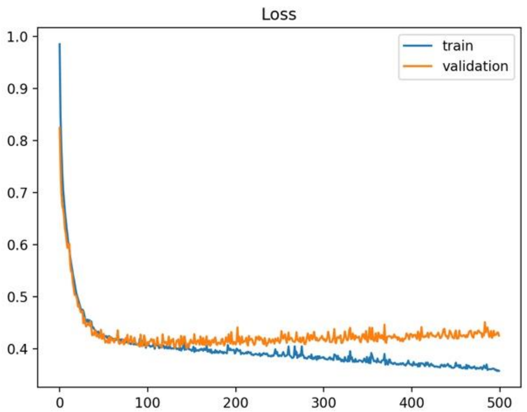
```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
optimizer = optim.Adam([var1, var2], lr=0.0001)
```

Learning Curve

- A learning curve is a plot that shows the number of epochs on the x-axis and the loss values on the y-axis
- Learning curves of model performance on the train and validation datasets can be used to diagnose an underfit (left), overfit (right), or well-fit (middle) model



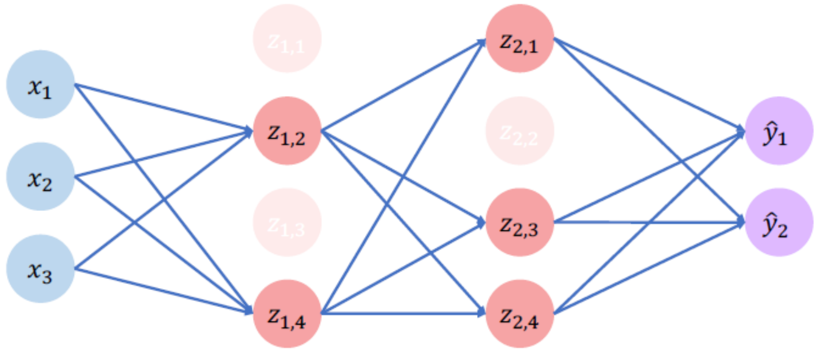




Regularization

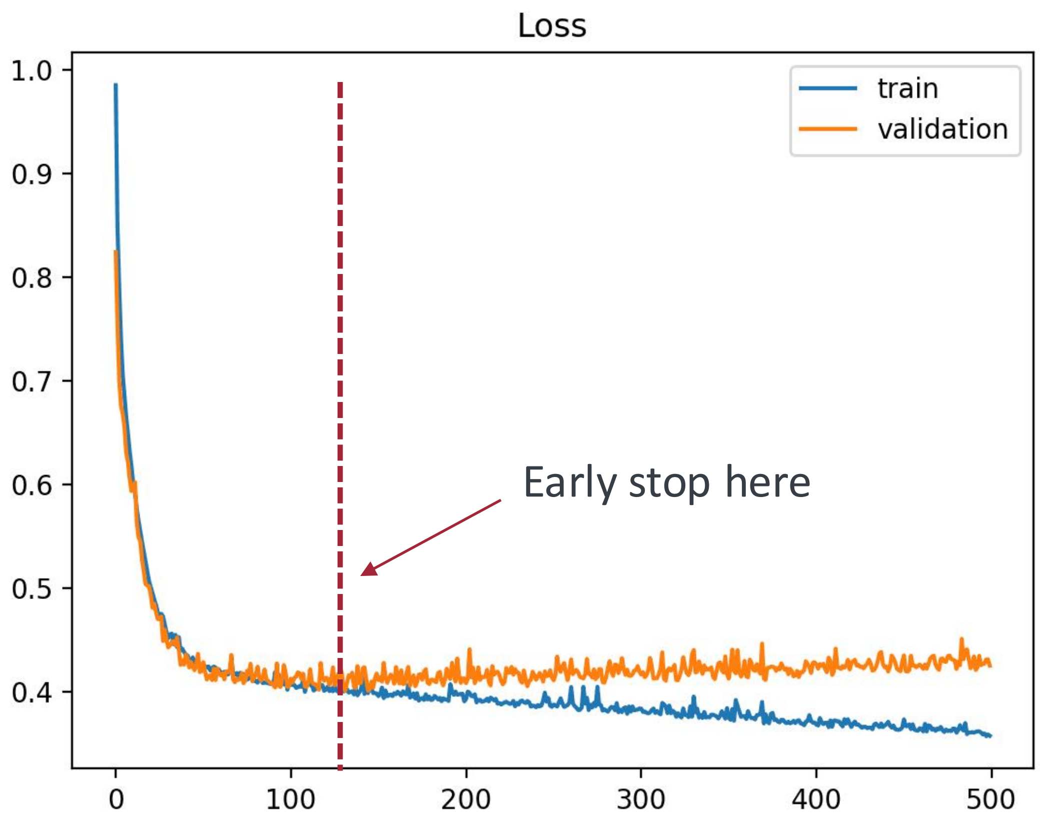
Dropout: Randomly dropping some nodes in the network during training to prevent the network from relying too much on a few nodes

- During training
  - In each forward pass during training, dropout randomly disables (or drops) a specified fraction of neurons in a layer by setting the neurons to zero
  - The remaining active neurons are scaled by a factor (often $\frac{1}{1-\text{dropout rate}}$) to ensure that the output has the correct expected value
- During inference: When making predictions, dropout is not applied
- Dropout helps the network generalize by preventing it from relying too heavily on specific neurons, reducing the likelihood of overfitting



Regularization

Early stopping: Stop training when performance on a validation set stops improving to prevent overfitting



Regularization

L2-Regularization: Add a penalty to the loss function to reduce model complexity and prevent overfitting

- Similar to ridge regression

$$\min J(W) \longrightarrow \min J(W) + \lambda \cdot \sum_{i \in \text{layers}} \sum_{k \in \text{nodes of } i} W_{ki}^2$$

Summary

Hyperparameters of a neural network that we need to tune

- The architecture of the neural network
  - The number of hidden layers
  - The number of hidden neurons in each hidden layer
  - The choice of the activation function

- Training a neural network
  - Batch size
  - Learning rate
  - Optimization

- Link to the Neural Network Playground

Acknowledgement

The lecture note has benefited from various resources, including those listed below. Please contact Zonghao Yang (zyang99@stevens.edu) with any questions or concerns about the use of these materials.

- Lecture Notes on Deep Learning Fundamentals by Léonard Boussioux at University of Washington
- Lecture Notes on Deep Learning from ML 2021 Spring by Hung-Yi Lee at National Taiwan University

THANK **YOU**

Stevens Institute of Technology
1 Castle Point Terrace, Hoboken, NJ 07030