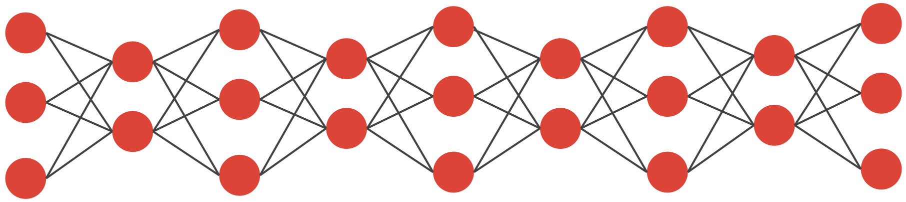
Deep Learning Fundamentals



Justin Chen
Boston University **M**achine **I**ntelligence **C**ommunity

Fluffy Stuff

- Motivation
- Application
- Statistical Learning

Motivation

Intelligent Machines

Patterns in Data

Understand the Mind

Application

SOUNDCLOUD

NOVARTIS

Google

Google Search

I'm Feeling Lucky

NETFLIX

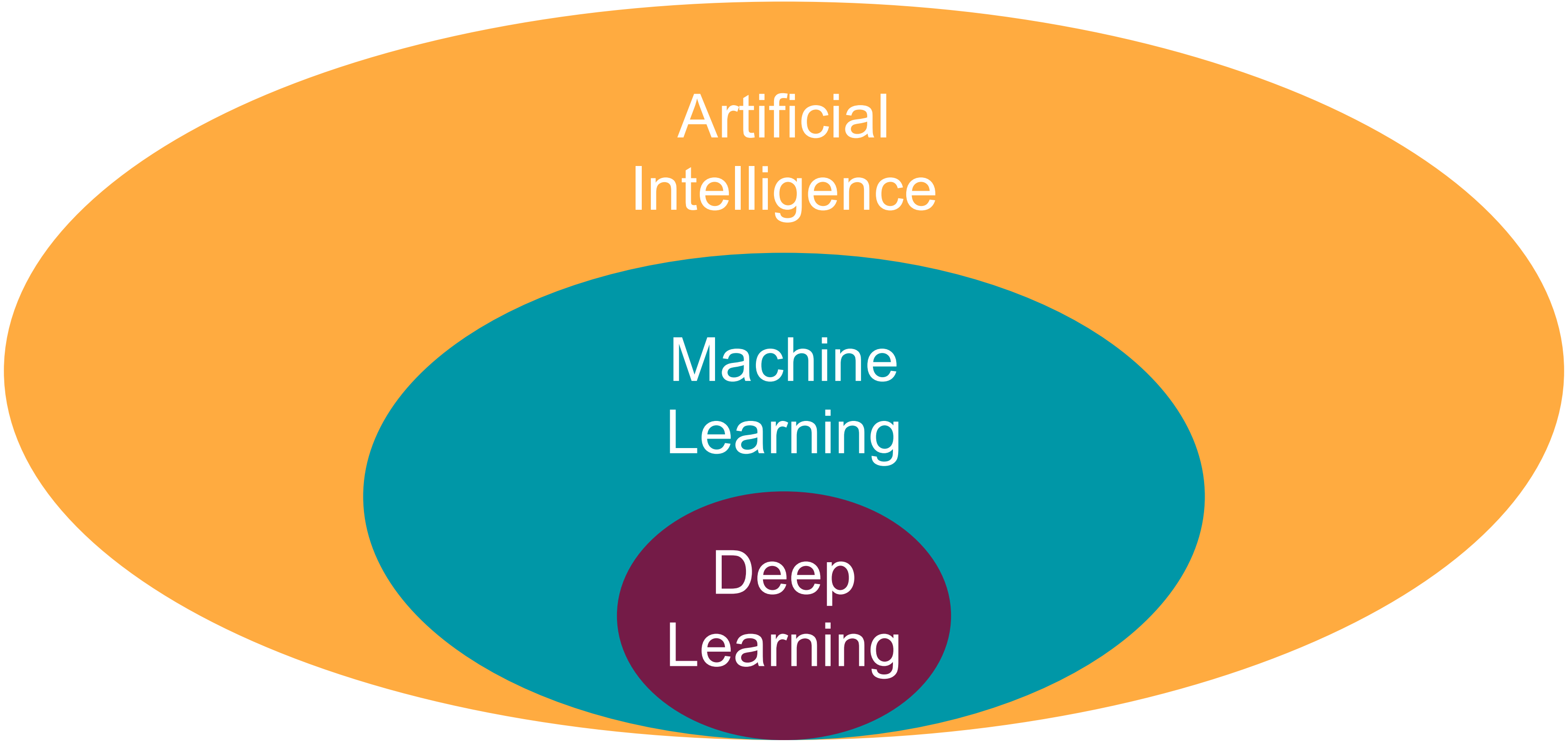
YouTube

WAYMO

amazon

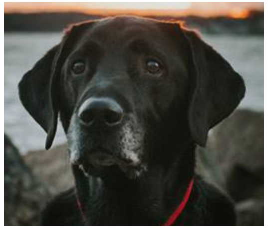
NASA

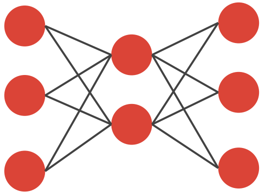
What is Deep Learning

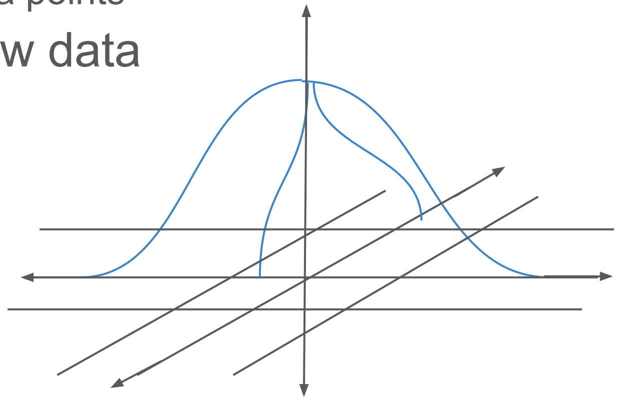


Fancy name aside

- Idealized model of biological neurons
- Learn patterns **directly from data**
- Calculus, Linear Algebra, Statistics, Probability, Optimization, Learning Theory, Computational Neuroscience, Systems Neuroscience, Computer Science,...
- Tasks:
  - **Classification** e.g. Given picture a picture, classify what object is in it
  - **Regression** e.g. Given a description of a house, predict the price
  - **Generation** e.g. Learn a distribution and generate data points
- Goal is to learn a function that **generalizes** to new data





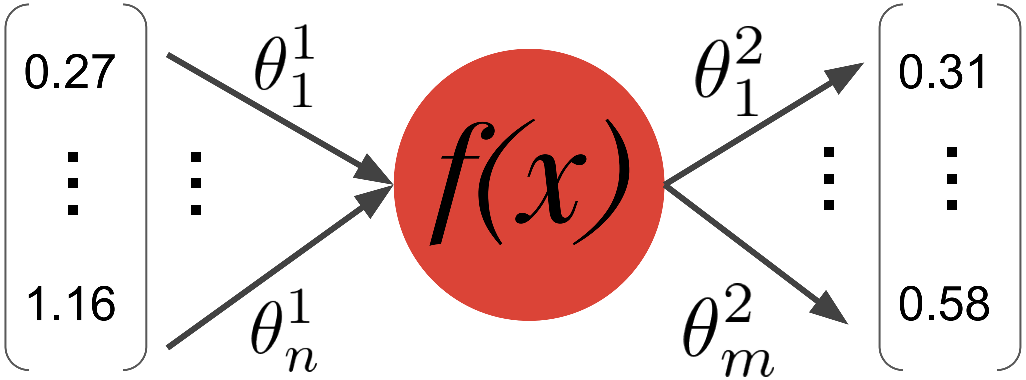


Artificial Neuron

- Nonlinear data
- Classification

Artificial Neuron

- **Linear Classifier**
- **Linear combination** of input values
- <u>**Nonlinear**</u> **activation function** of total input
- Each connection has an associated **weight** θ_i^l



Nonlinearity

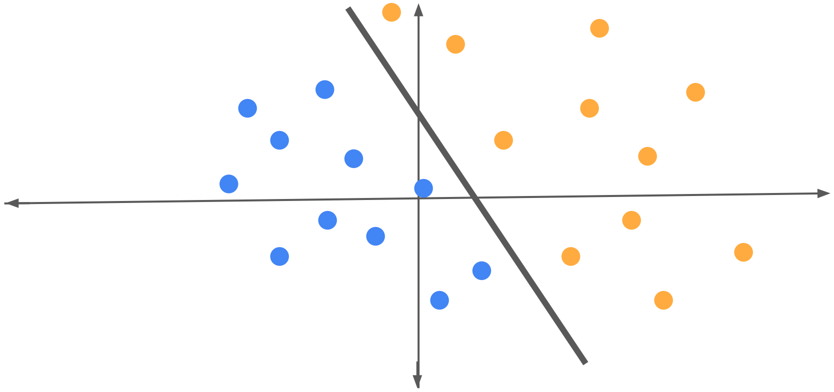
- Pass sum of all inputs (x in equations below) into a activation function
- Draw nonlinear **decision boundary**
- Handle **nonlinear data**

$$y = Mx + b \rightarrow h(x) = \theta x$$

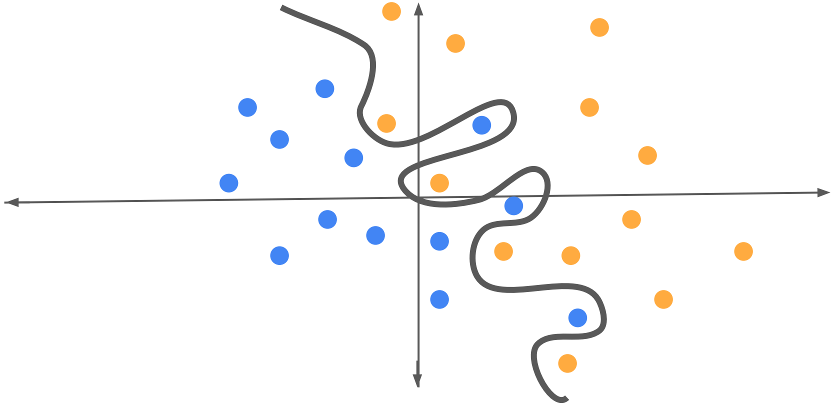
$$\downarrow$$

$$f(h(x)) = f(\theta x)$$

Linear Boundary



Nonlinear Boundary

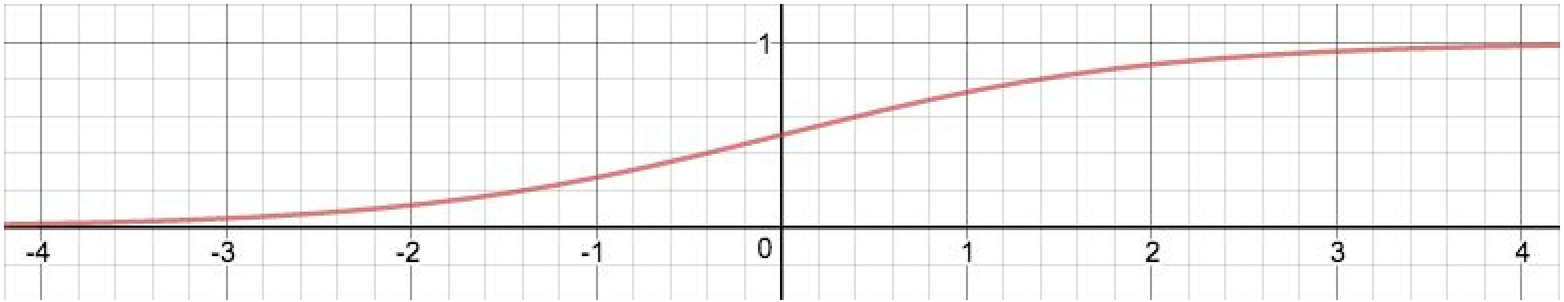


Logistic Sigmoid

- Used for binary classification
- Outputs value in (-1, 1)
- f(0) = 0.5
- Class 1 if output <   0.5
- Class 2 if output >= 0.5

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

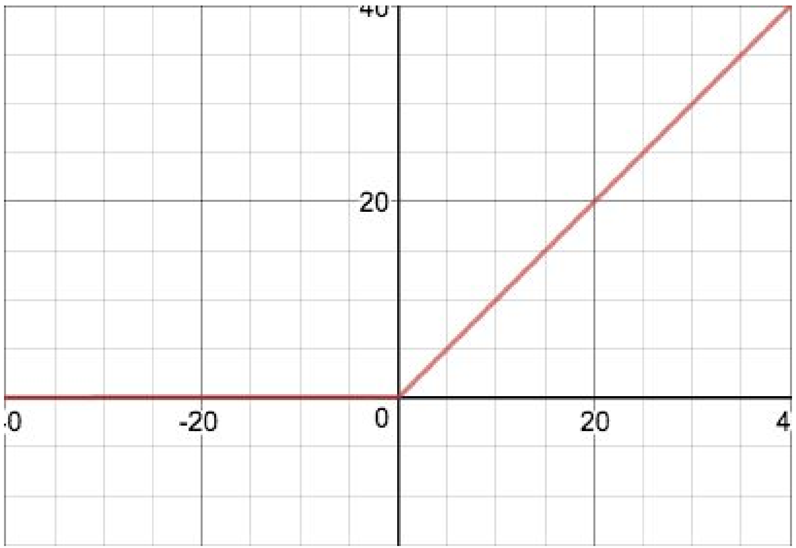
$$f'(x) = \left(\frac{1}{1 + e^{-x}}\right)\left(\frac{e^{-x}}{1 + e^{-x}}\right)$$



Rectified Linear Unit

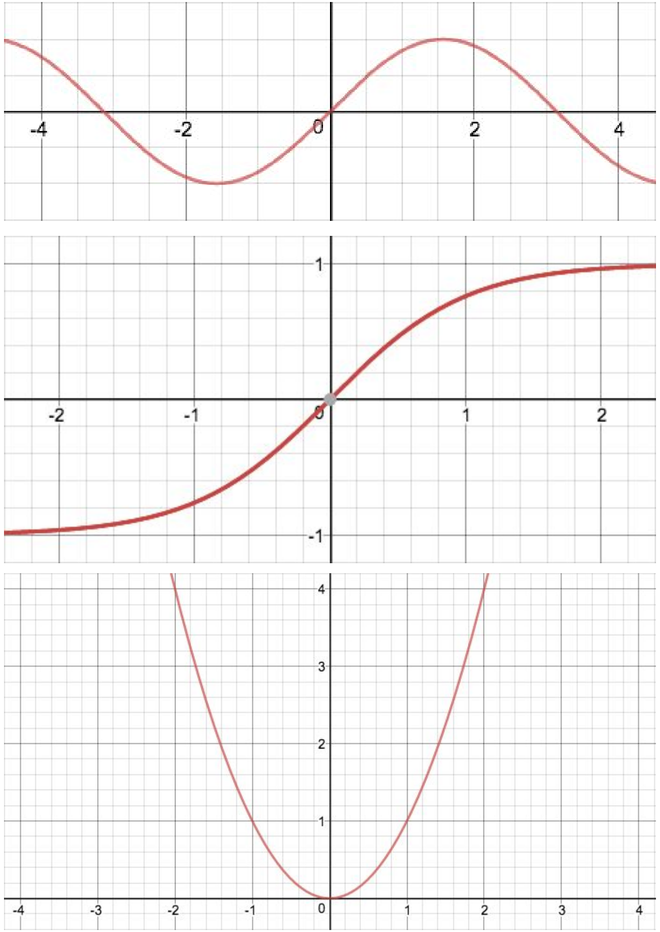
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

- Currently most popular activation function
- Derivative of 0 is 0 and derivative of x is a constant
- **Inexpensive** to compute
- **Faster** to train [1]



Expressivity

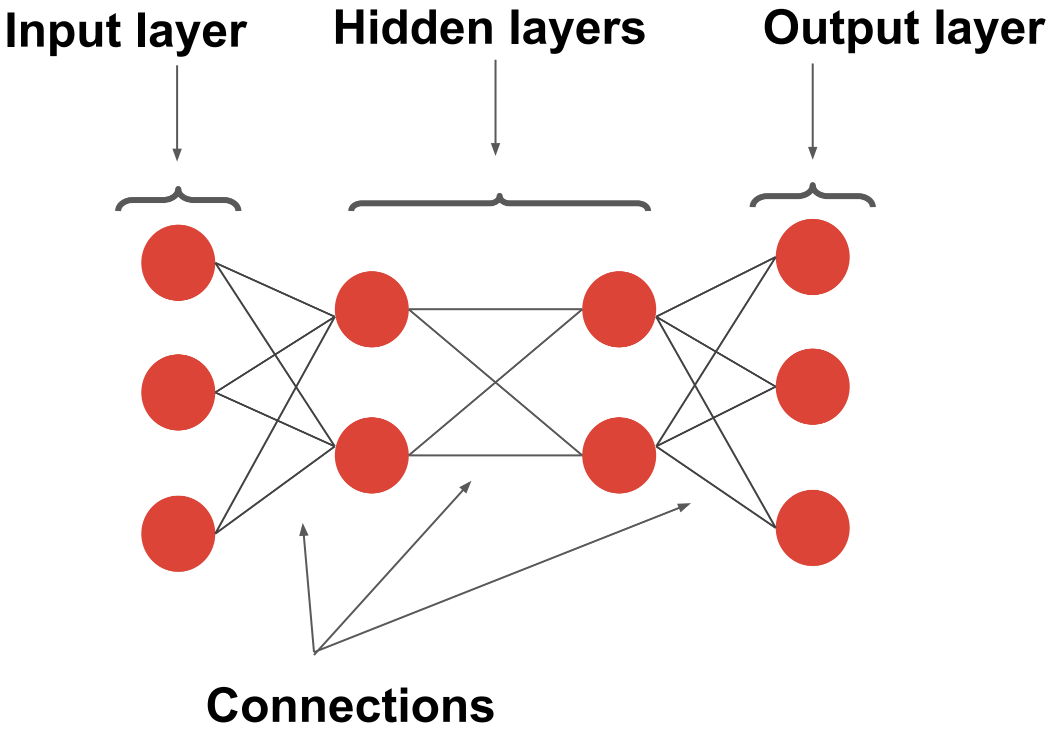
- Expressivity - ability/power to **express solution**
- Possible to use any activation function as long as it's nonlinear, but depends on how you train the network
- Typically will only use **differentiable** functions
- More nonlinearity = more **expressive power**
- Think of activation function as **behavior of a neuron**
    - Biological brain contains various types of neurons with different behaviors



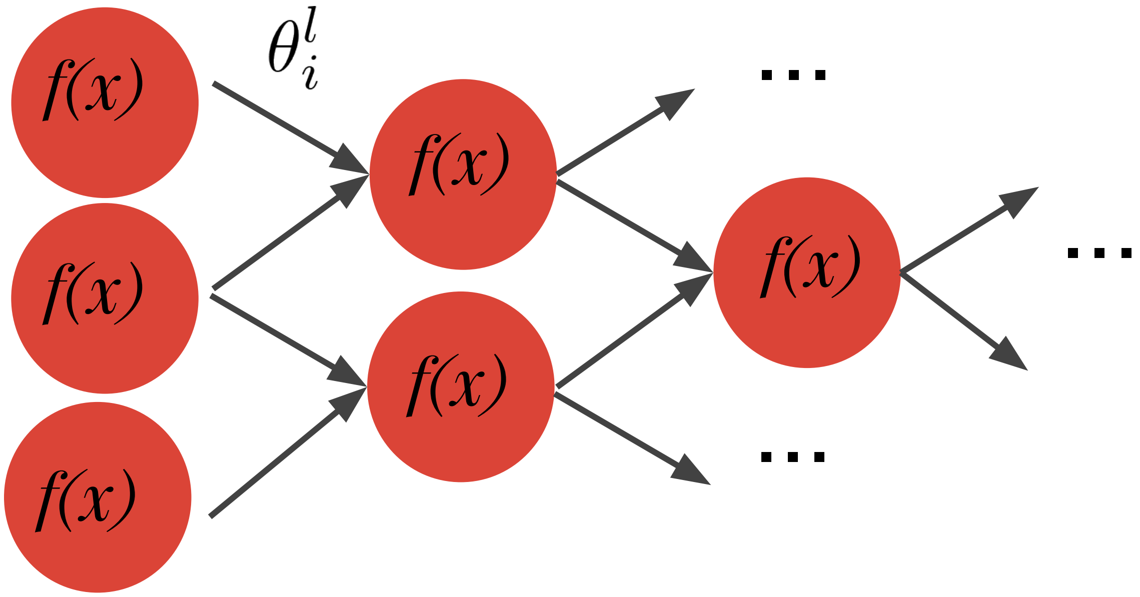
Artificial Neural Network

- Composition
- Computation Graph
- Deep Neural Networks

Artificial Neural Network



Computation Graph



- **Directed computation graph**
- Typically use the same activation function for each node
- Each connection has an associated weight
- **Weight** θ_i^l represents how much the receiving neuron should consider it during its own computation

Ugly Truth

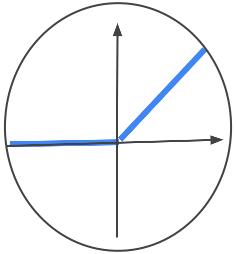
N-dimensional feature vector (Input)

Possibly more hidden layers

Output layer weights

Activations from previous layer

$$\begin{bmatrix} a_{00} & \cdots & a_{0n} \\ \vdots & \ddots & \vdots \\ a_{m0} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix}$$



$$\cdots \begin{bmatrix} z_{00} & \cdots & z_{0q} \\ \vdots & & \vdots \\ z_{p0} & \cdots & z_{pq} \end{bmatrix} \begin{bmatrix} h_0 \\ \vdots \\ h_q \end{bmatrix}_d \rightarrow \begin{bmatrix} y_0 \\ \vdots \\ y_p \end{bmatrix}$$

Weights between input layer and first hidden layer

ReLU

Potentially squash last hidden activations through softmax for classification

Output vector

Universal Approximation Theorem

- Any feed-forward neural network with **at least one hidden layer** using **logistic sigmoid** activation function can **approximate** any **continuous function** over a **compact set** to an arbitrary accuracy given sufficient parameters

  (Cybenko, 1989) [2]

- Any feed-forward neural network with **at least one hidden layer** using **any activation function** can **approximate** any **continuous function** over a **compact set** to an arbitrary accuracy given sufficient parameters

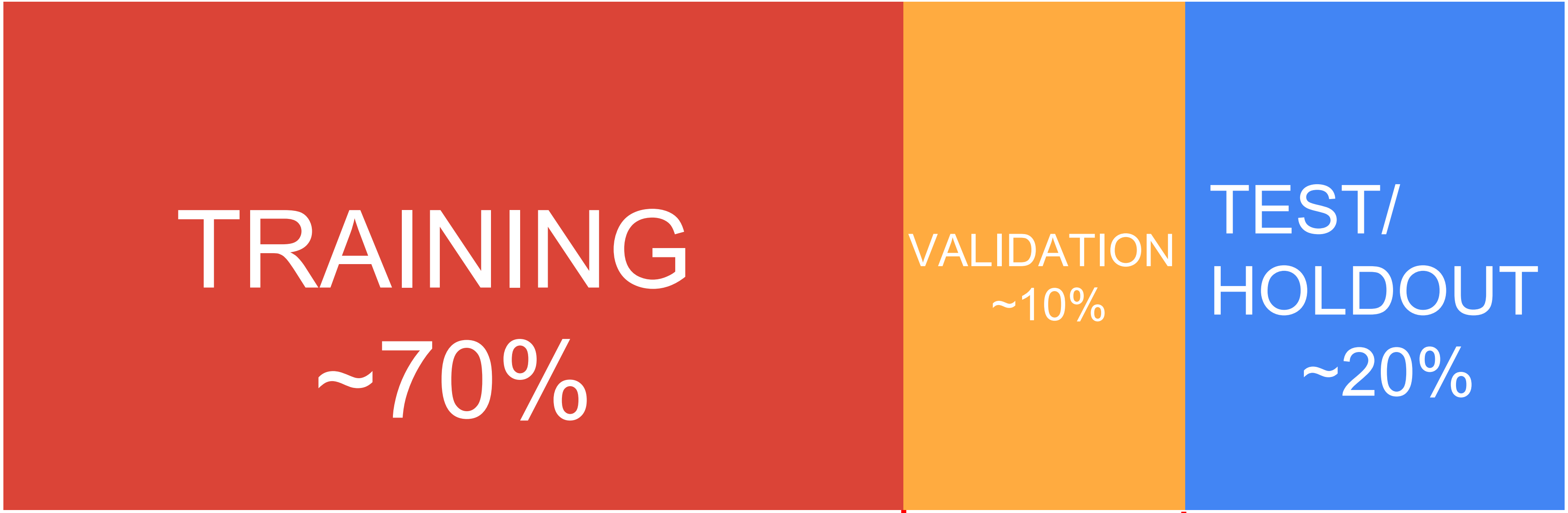
  (Hornik, 1991) [3]

How do they actually learn?

- Costs
- Gradients
- Error corrections

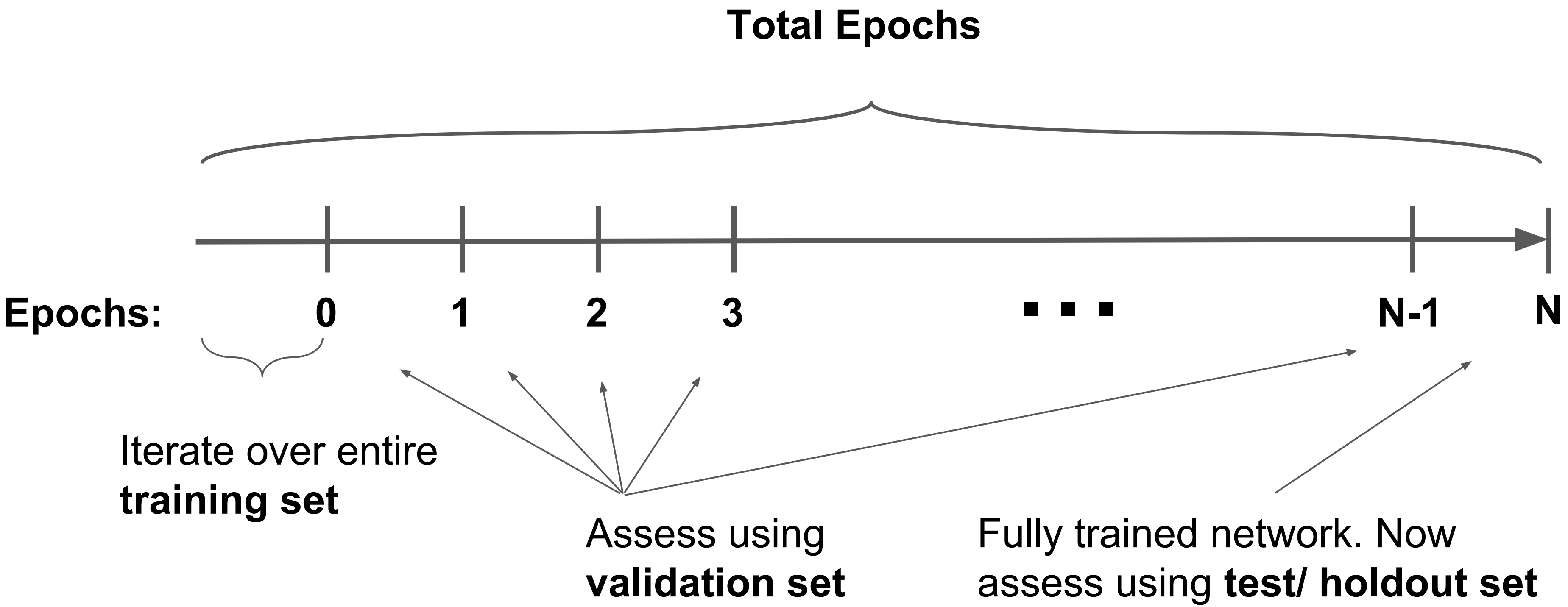
Dataset

Split - percentages depend on available data



Balance - data distributed close to uniformly

Training


Total Epochs
Epochs:
0
1
2
3
...
N-1
N
Iterate over entire **training set**
Assess using **validation set**
Fully trained network. Now assess using **test/ holdout set**

Cost Function

- Also known as **objective** function, **loss** function
- Tells learning algorithm how well/poorly it's doing
- Task-dependent hyperparameter

Least Square Error

for regression

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Cross-Entropy

for classification

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)})))$$

Gradient Descent

- Minimize cost function w.r.t. Weights
- Each update is also known as a **Gradient Step**

Scalar learning rate

$$\theta_j := \theta_j - \alpha \overbrace{\frac{\partial}{\partial \theta_j} J(\theta)}^{\text{Also written } \nabla J(\theta)}$$

Individual weights

Vector of weights

Cost/objective/loss function

Gradient Descent cont.

Batch/Vanilla Gradient Descent

- Feed in entire dataset and then make one update to weights
- Slow and must store dataset in memory

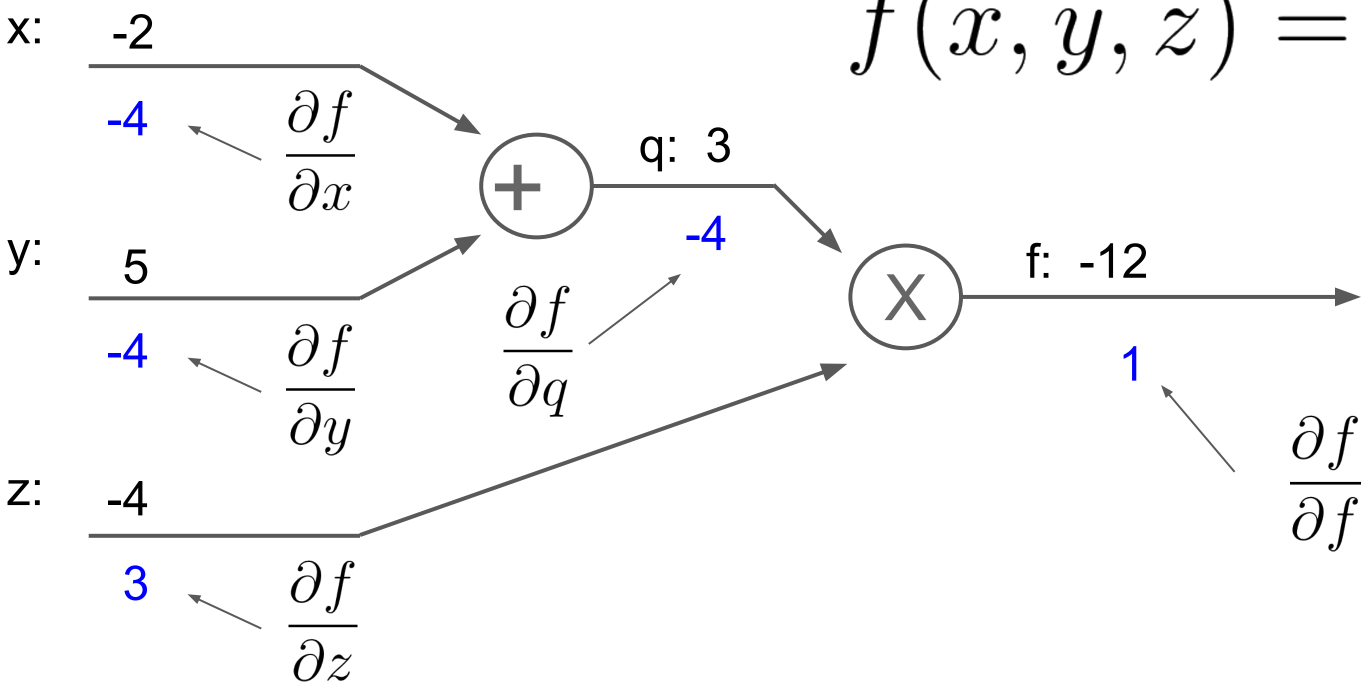
Stochastic Gradient Descent

- Feed in one example and then update weights
- Loop over dataset
- Fast and implicitly regularizes

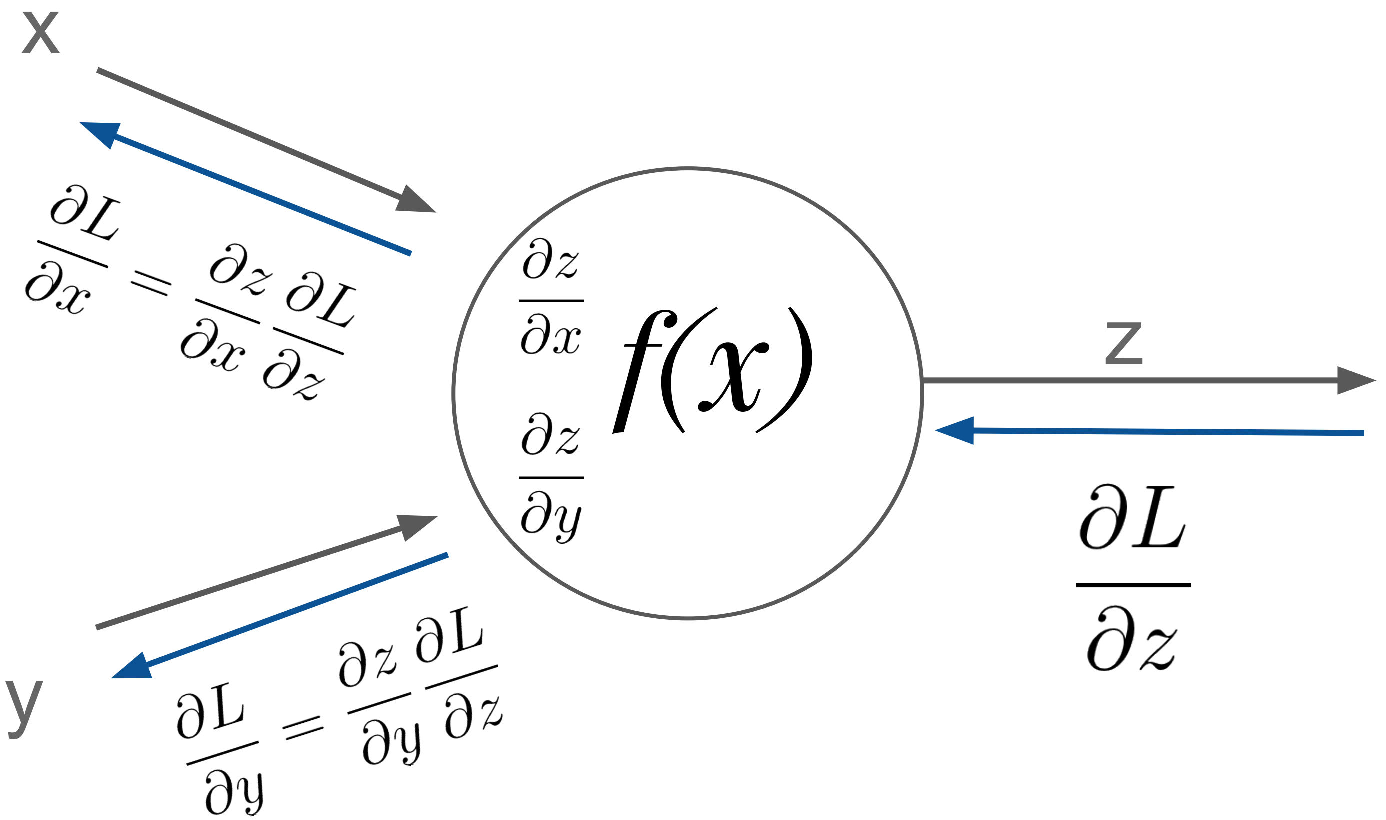
Backpropagation

- Compute gradient of loss w.r.t. to weights

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



Backpropagation



Reminders

1. **Stochastic Gradient Methods for Large Scale Machine Learning**

   Location: Harvard, Maxwell Dworkin G115

   Time: 4 PM, ice cream social at 3:30

2. **MIT MIC tomorrow at 5 PM on CycleGAN**

   Location: 56-154

   Time: 5 PM

   Food will be provided

References & Further Reading

[0] Le, Quoc V. "A Tutorial on Deep Learning Part 1: Nonlinear Classifiers and The Backpropagation Algorithm." (2015).

[1] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *Nature* 521.7553 (2015): 436-444.

[2] Gybenko, G. "Approximation by superposition of sigmoidal functions." *Mathematics of Control, Signals and Systems* 2.4 (1989): 303-314.

[3] Hornik, Kurt. "Approximation capabilities of multilayer feedforward networks." *Neural networks* 4.2 (1991): 251-257.

[4] Stanford CS231n: Convolutional Neural Networks for Visual Recognition

[5] https://github.com/ch3njust1n/bestofml