CS5120 VLSI System Design, Spring 2025

Deep Neural Networks (DNNs) Part 1

黃稚存

Chih-Tsun Huang

cthuang@cs.nthu.edu.tw

國立清華大學
NATIONAL TSING HUA UNIVERSITY

資訊工程學系
Computer Science

Lecture 02

聲明

- 本課程之內容 (包括但不限於教材、影片、圖片、檔案資料等)，僅供修課學生個人合理使用，非經授課教師同意，不得以任何形式轉載、重製、散布、公開播送、出版或發行本影片內容(例如將課程內容放置公開平台上，如 Facebook, Instagram, YouTube, Twitter, Google Drive, Dropbox 等等)。如有侵權行為，需自負法律責任。

Outline

- Machine learning basics
- Deep learning basics
- On training deep neural networks
  - Review of DNN
  - Empirical loss functions
  - Backpropagation
  - Learning rate
  - Gradient decent
  - Overfitting and regularization
- Tinker with a Neural Network
- Deep learning frameworks
- Training vs. inference

Machine Learning Basics

- Machine Learning: Infrastructure for Everything
- Reference: https://www.coursera.org/learn/machine-learning
  Prof. Andrew Ng, Stanford

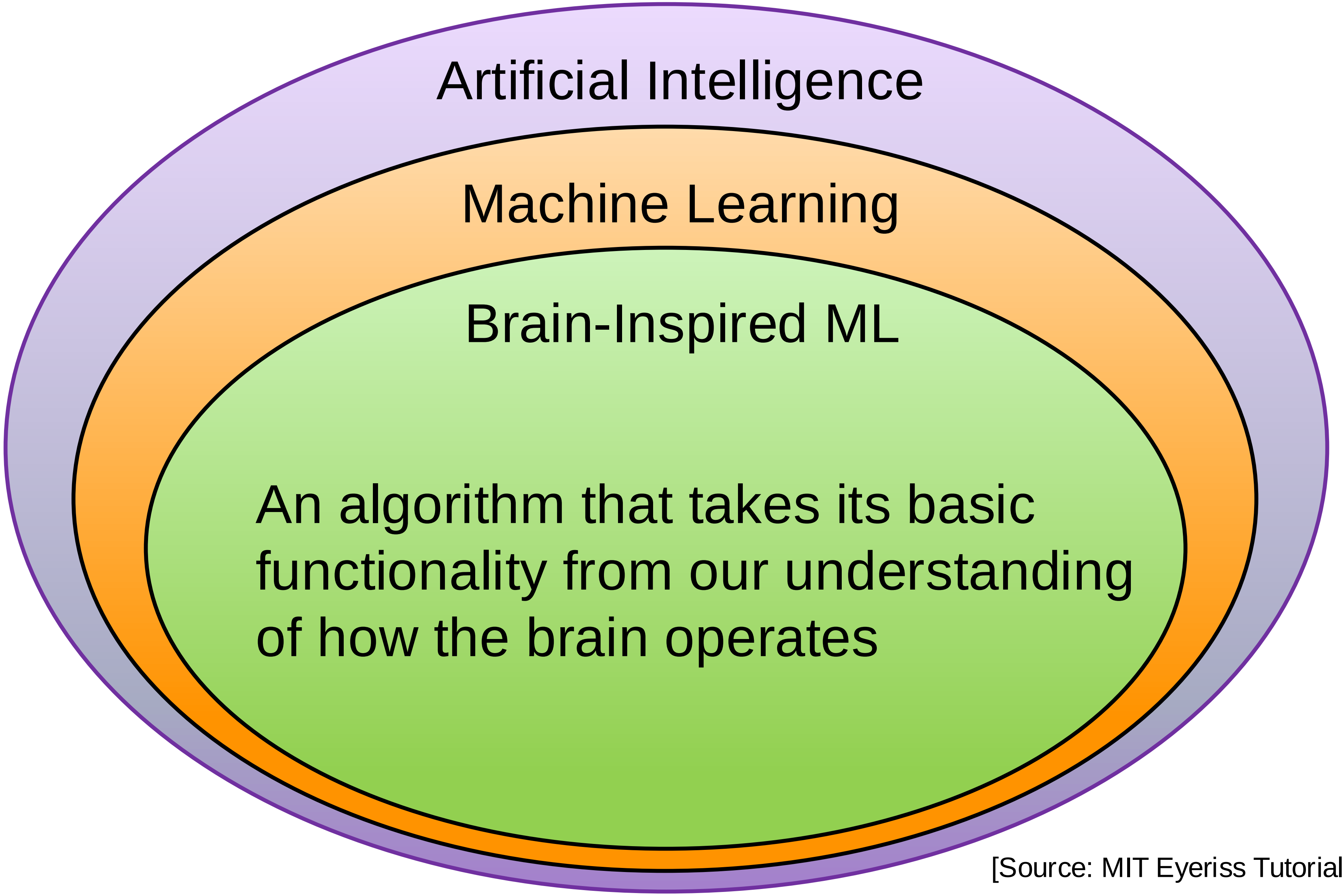
What is Machine Learning (ML)

- “The field of study that gives computers the ability to learn without being explicitly programmed.”
  Arthur Samuel (1959)

- Artificial Intelligence (AI)?
- Deep Learning (DL)

- Convolutional Neural Network (CNN)?
- Deep Neural Network (DNN)?
- Neuromorphic Computing (NC)?

General Definitions

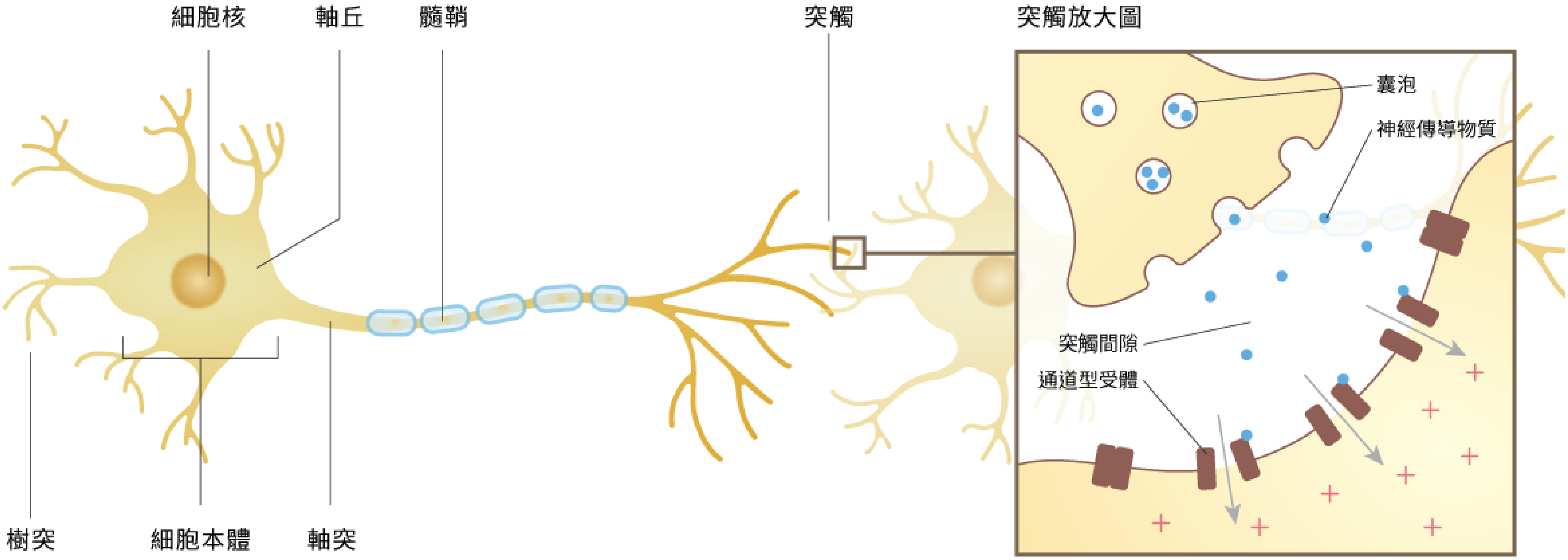


[Source: MIT Eyeriss Tutorial]

General Definitions (Cont)

- Artificial Intelligence
  - “The science and engineering of creating intelligent machines” - John McCarthy, 1956
- Machine Learning (ML)
  - “Field of study that gives computers the ability to learn without being explicitly programmed” - Arthur Samuel, 1959
- Brain-Inspired ML
  - An algorithm that takes its basic functionality from our understanding of how the brain operates

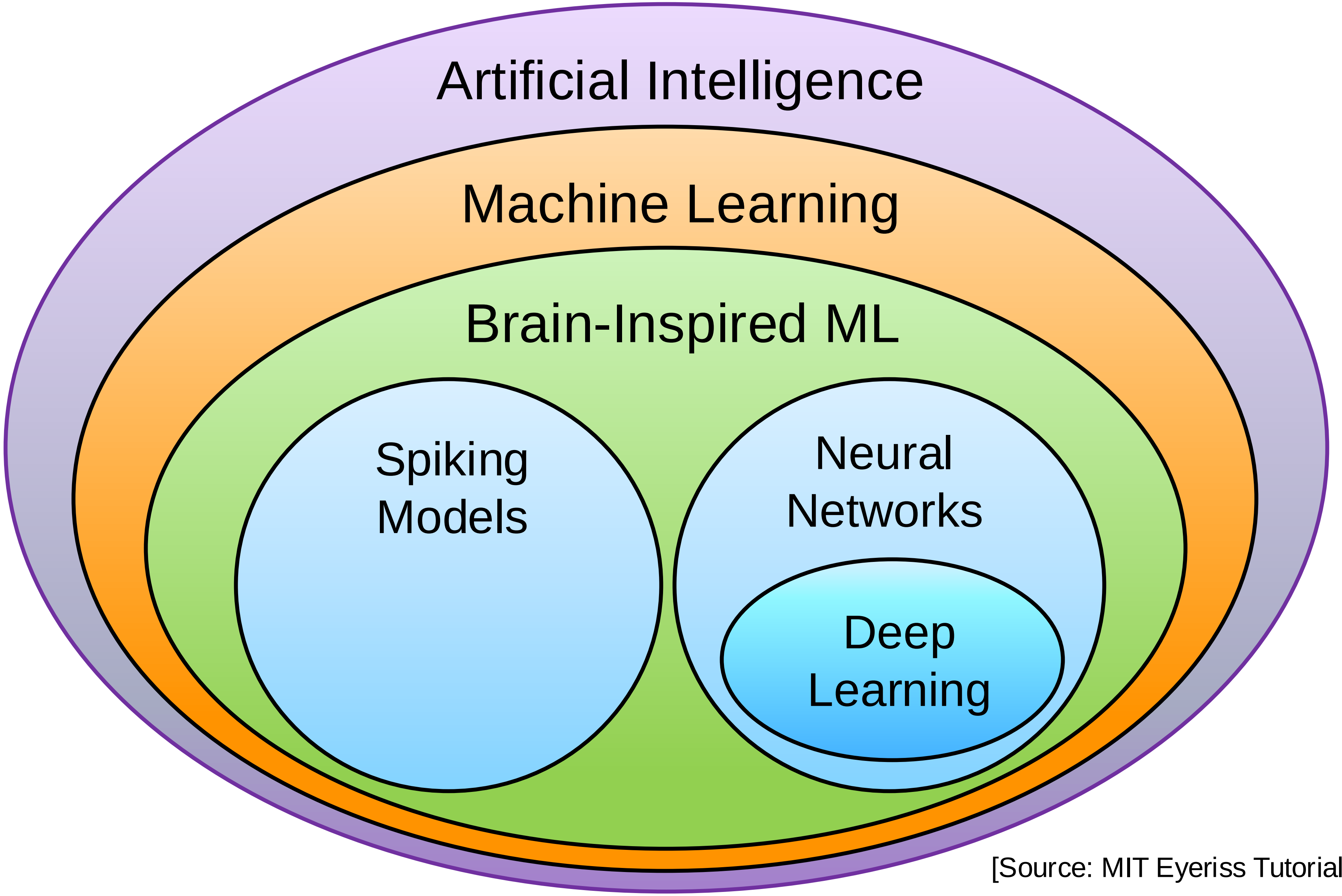
How Does the Brain Work?



<https://www.narlabs.org.tw/>

- **Neuron**: basic computational unit of the brain
  - 86B neurons in the brain
  - Connected with 10^{14} – 10^{15} **synapses (突觸)**
- Neurons receive input signal from **dendrites (樹突)** and produce output signal along **axon (軸突)**, which interact with the dendrites of other neuros via **synaptic weights**
  - Synaptic weights: learnable & control influence strength

General Definitions



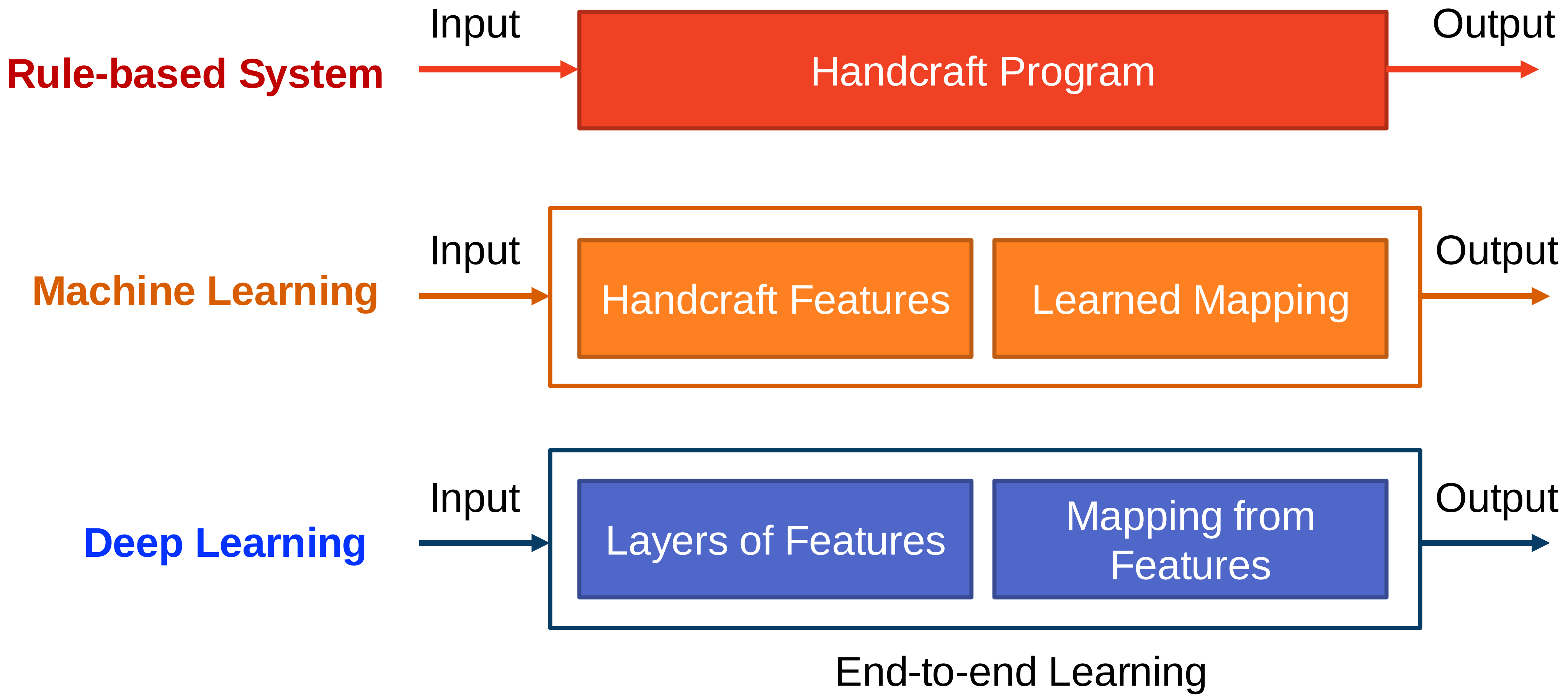
[Source: MIT Eyeriss Tutorial]

Deep Learning (DL)

- "Seek to exploit the unknown structure in the input distribution in order to discover good representations, often at multiple levels."

  - Yoshua Bengio, 2012

Variations of Artificial Intelligence



Is The Winter Over?

- Two major **AI Winters** (1974-80, 1987-93) and some smaller periods
  - https://en.wikipedia.org/wiki/AI_winter
- It is!
  - AI has surpassed human performance
    - Chess (1997)
    - Trivia (2011)
    - Atari games (2013)
    - Image recognition (2015)
    - Speech recognition (2015)
    - Go (2016)
- Maybe?!
  - We need explainable AI (or general AI)

©. Artistic illustration of nuclear winter. (Elena Naylor/Shutterstock)

Machine Learning Algorithms

- A computer program is said to ***learn*** from ***experience (E)*** with respect to some ***task (T)*** and some ***performance measure (P),*** if its performance on *T*, as measured by *P*, improves with experience *E*.
  - Tom Mitchell, 1998

- E.g., spam classification
  - Task (*T*): Predict emails as spam or not spam
  - Experience (*E*): Observe users label emails as spam or not
  - Performance (*P*): # of emails that are correctly predicted

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Machine Learning Algorithms

- Suppose a tumor classification program watches which tumor is being marked as “benign” or “malignant” by doctors, and it learns how to better predict benign/malignant tumors.

- What is the task *T* in this setting?
  A. The number (or fraction) of tumors correctly classified as benign/malignant
  B. Watching doctors label tumors as benign or malignant
  C. Classifying tumors as benign or malignant
  D. Non of the above. This is not a machine learning problem.

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

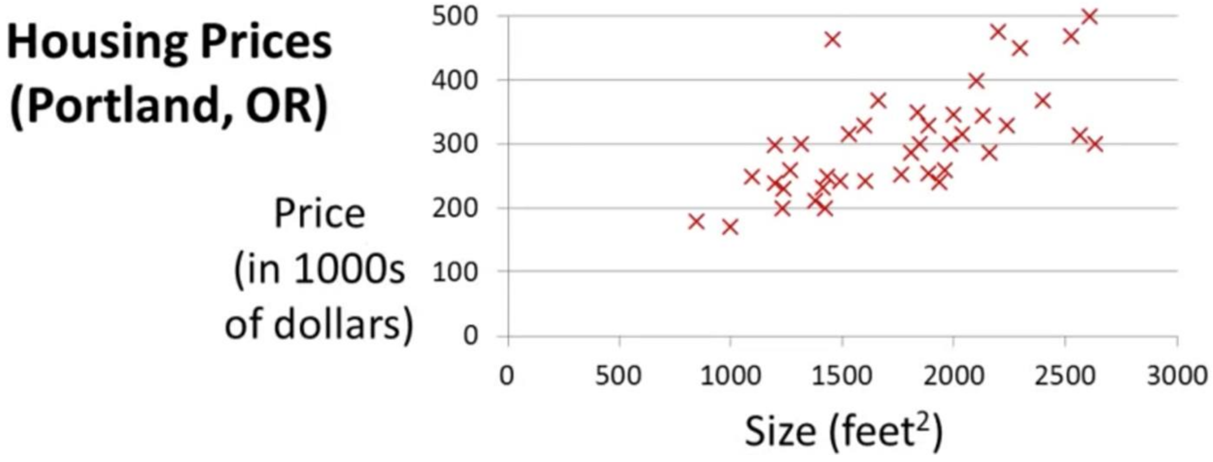
Building an ML Algorithm

- Nearly all ML algorithms can be described as particular instances of a simple recipe:
  - Experience (E) → Dataset with ground truth (labels)
  - Performance Measure (P) → Cost (loss) function
  - Task (T) → Model + optimization method
- Use this recipe to see the different algorithms:
  - As part of a taxonomy of methods for doing related tasks that work for similar reasons
  - Rather than as a long list of algorithms that each have separate justifications

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Example: Linear Regression

- **Dataset:**
  - *m* training examples
  - (x, y) = (size, price)
- **Model:**
  - $h(x) = w_0 + w_1 * x$
- **Cost function:**
  - Measures the performance of an ML model for given data
  - Mean Squared Error:
    $$MSE = \frac{1}{m}\sum_{i=0}^{m}(h(x_i) - y_i)^2$$
- **Optimization method:**
  - Solve for where its gradient is 0.
  - Gradient descent



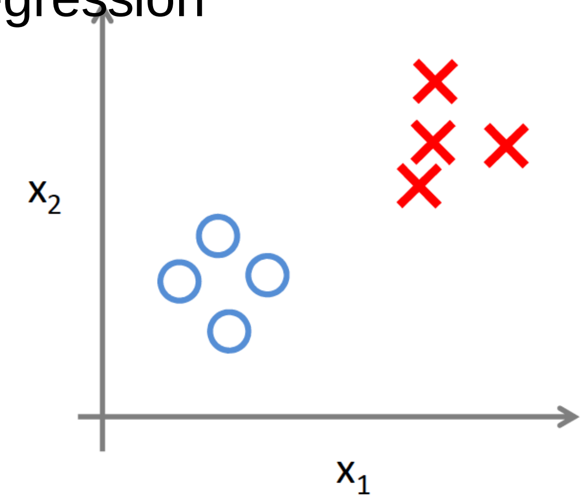
Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

[Source: Machine Learning, Andrew Ng]

Supervised vs. Unsupervised Learning

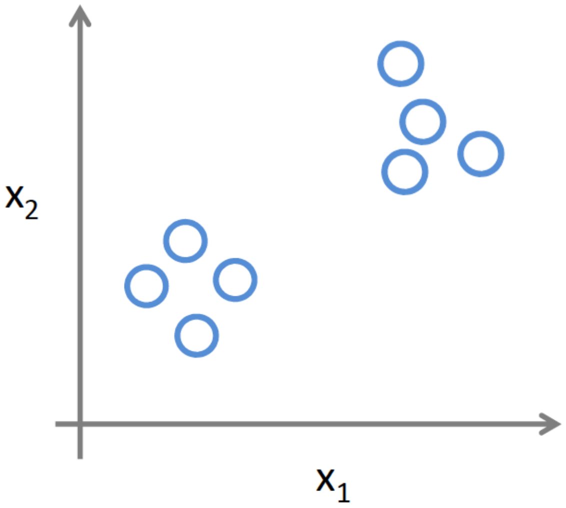
Supervised Learning

- Given right answers to learning
  - Dataset: a collection of many examples with labels (or ground truth)
- Predict proper output value
- Applications
  - Classification
  - Regression



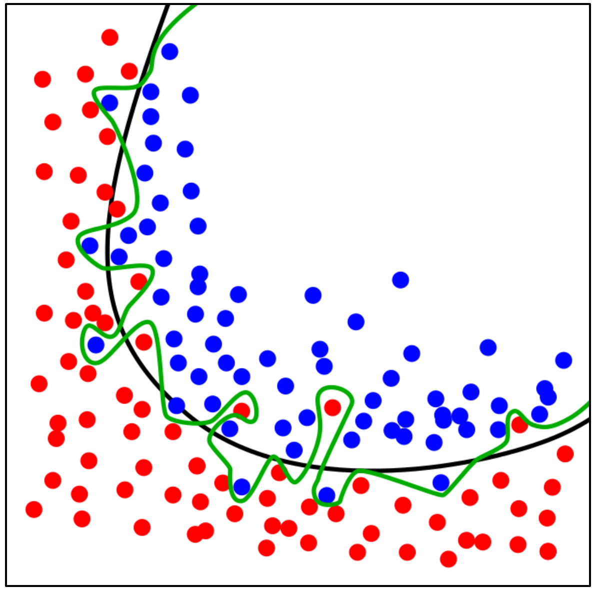
Unsupervised Learning

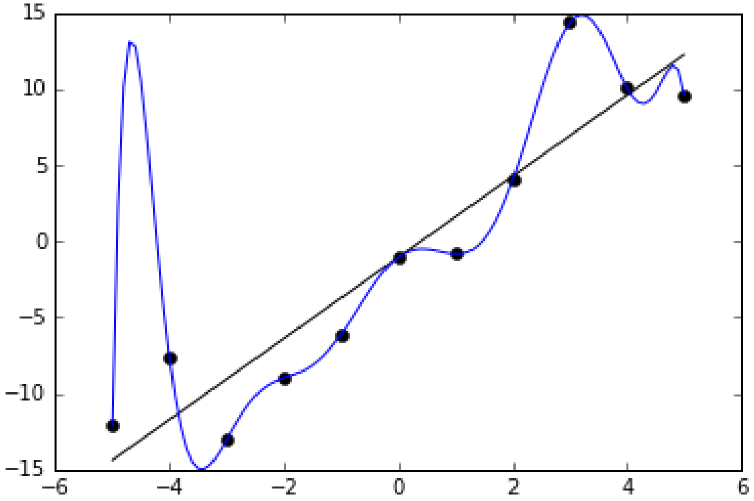
- No right answers with the data
  - Without label (ground truth)
- Applications
  - Clustering
  - Anomaly detection



Subtlety of Supervised Learning

- Hyper parameters
  - Model type
  - Architecture
  - Training parameters (learning rate, regularization, etc.)
  - Model parameters
- Data selection, data size
- Overfitting/underfitting

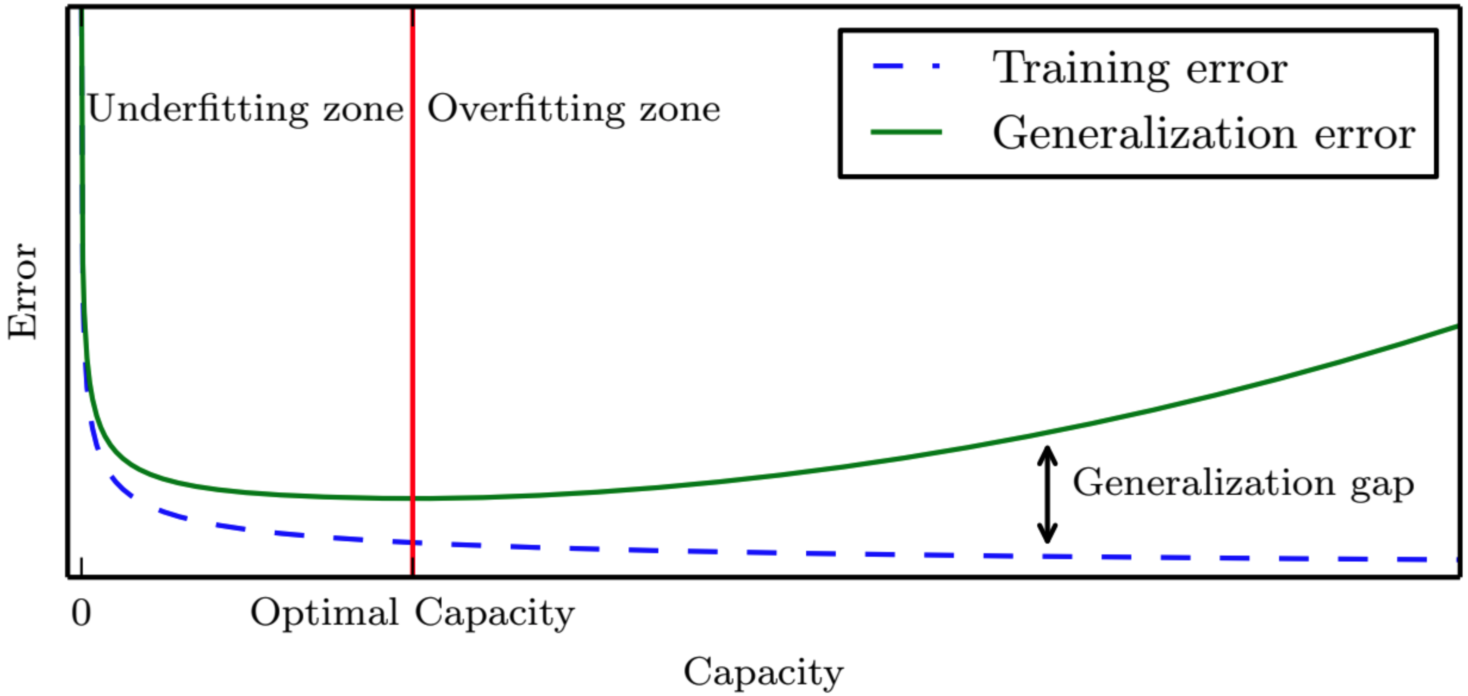




https://en.wikipedia.org/wiki/Overfitting

Generalization of Supervised Learning

- Training with dataset
- Generalization
  - Algorithm performs well on new, unseen inputs
  - Instead of those in the dataset
- Partition the dataset into
  - Training set: where the model was trained
  - Test set: where the trained model was tested
- Overfitting
  - Large gap between training and test errors

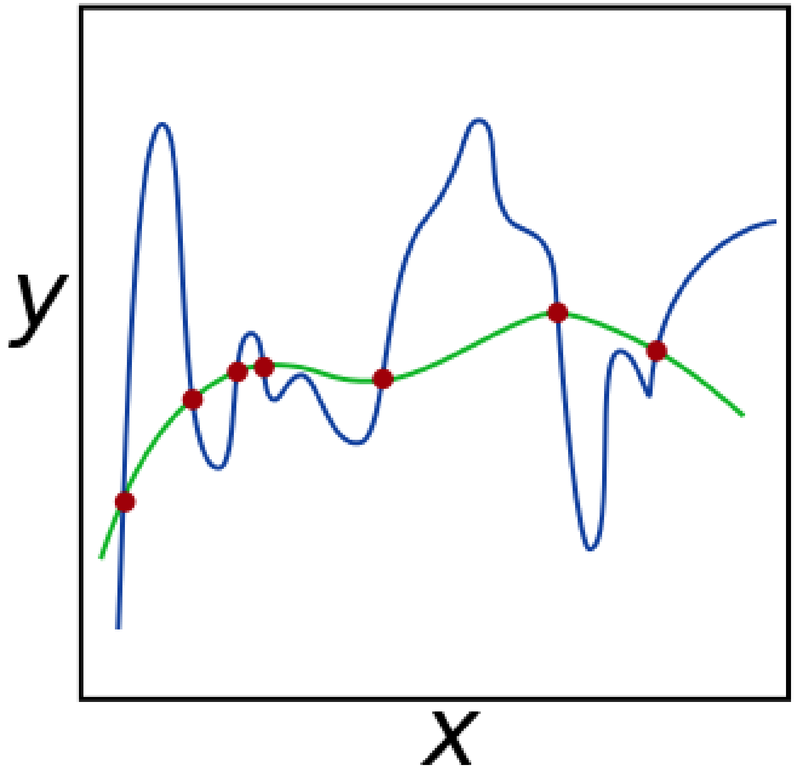


[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Use Regularization to Prevent Overfitting

- Regularization: modifications to reduce the generalization error but not the training errors
  - Weight decay (L2/L1 regularization)
    - Expressing preferences of smaller weights
    - $CF = MSE_{train} + \lambda \sum_i w_i^2 \ (L2)$
      (*CF*: cost function)
    - $CF = MSE_{train} + \lambda \sum_i |w_i| \ (L1)$

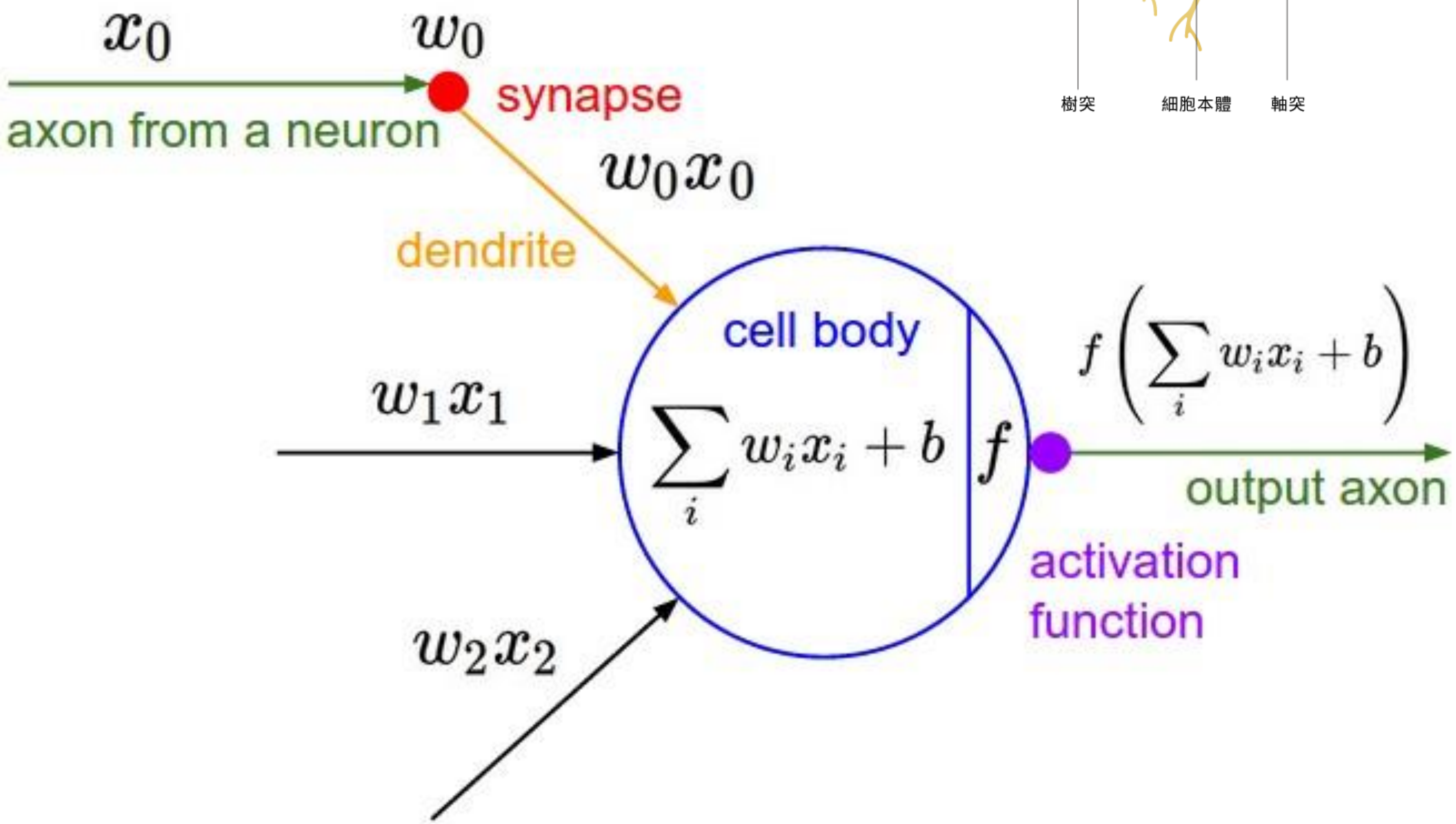
Hyperparameter

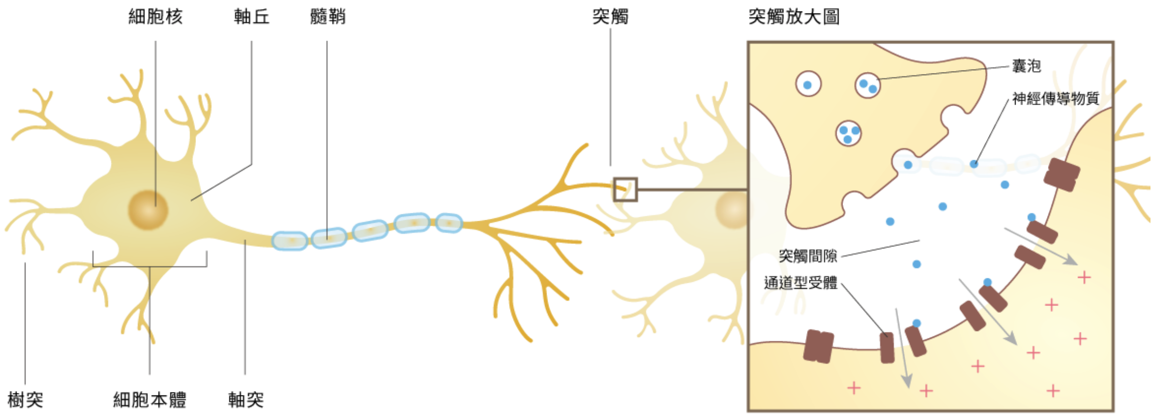


[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Deep Learning Basics

Neural Networks: Weighted Sum

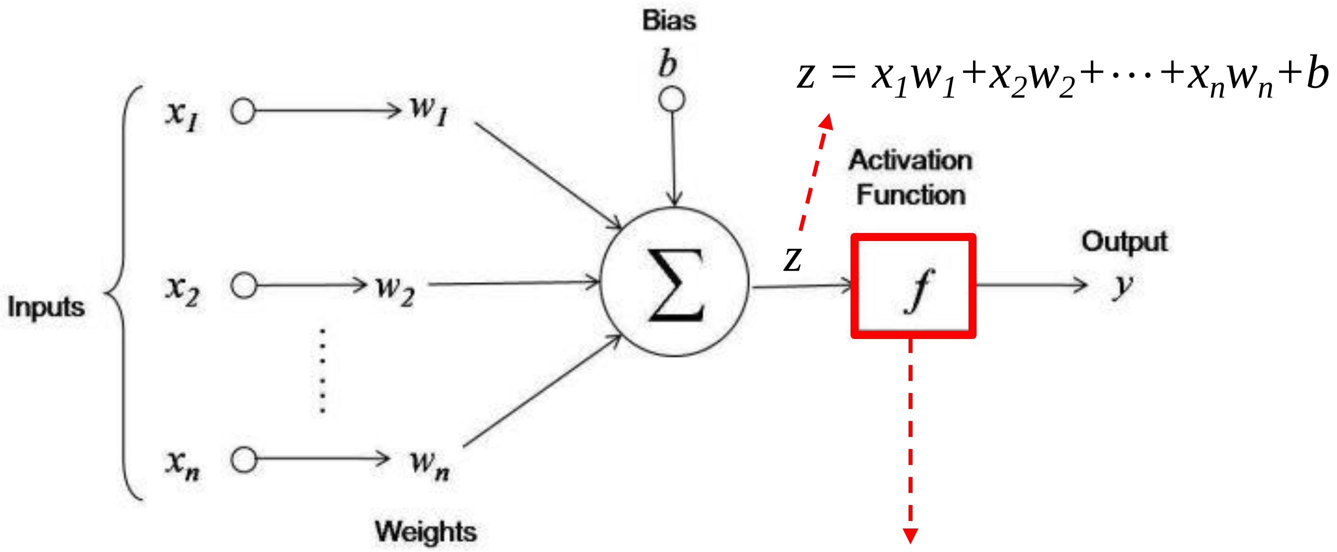


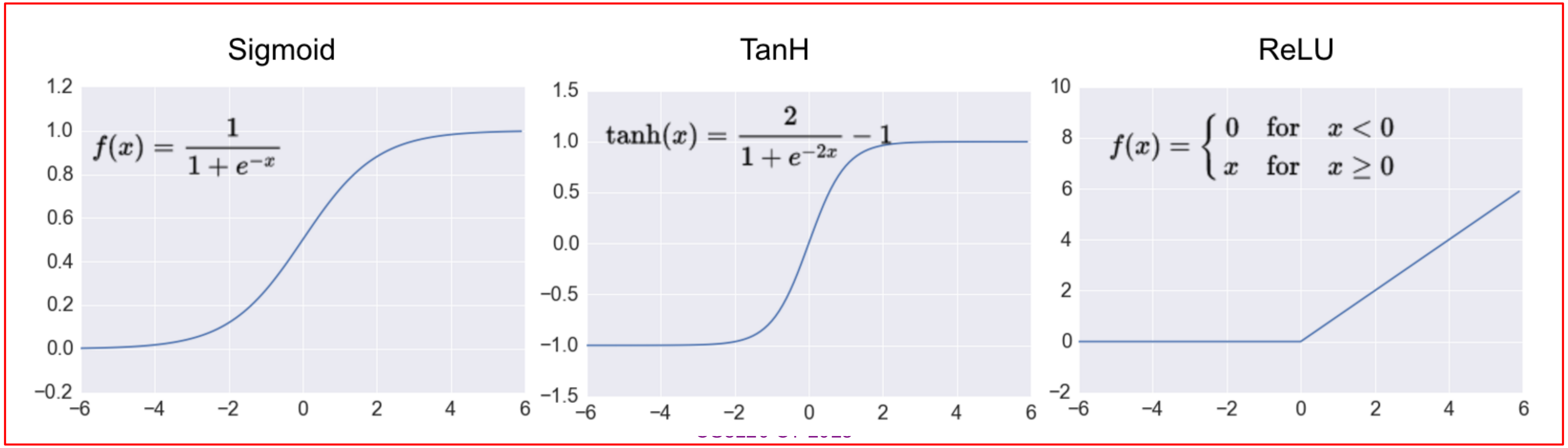


<https://www.narlabs.org.tw/>

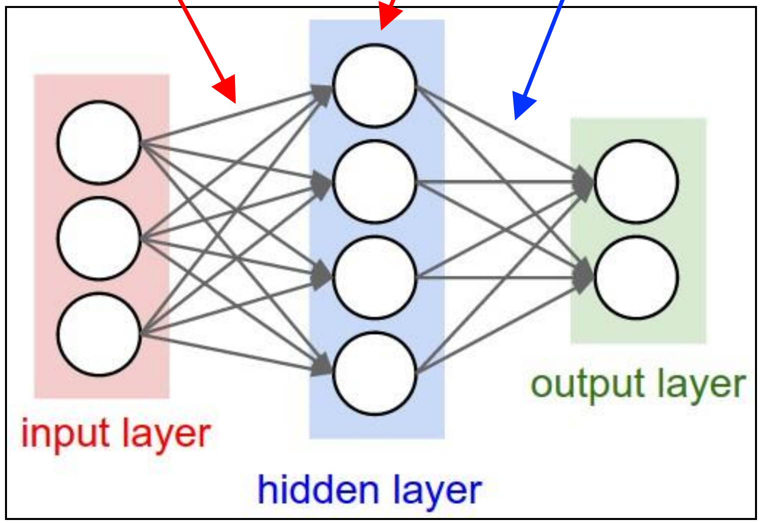
[Source: Stanford CS231n]

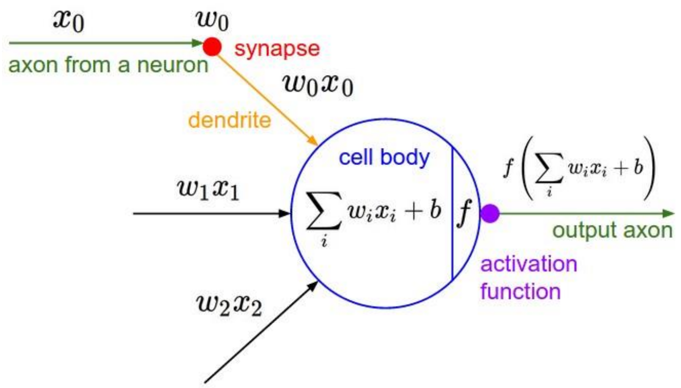
Neural Networks: Non-Linear Activation Function





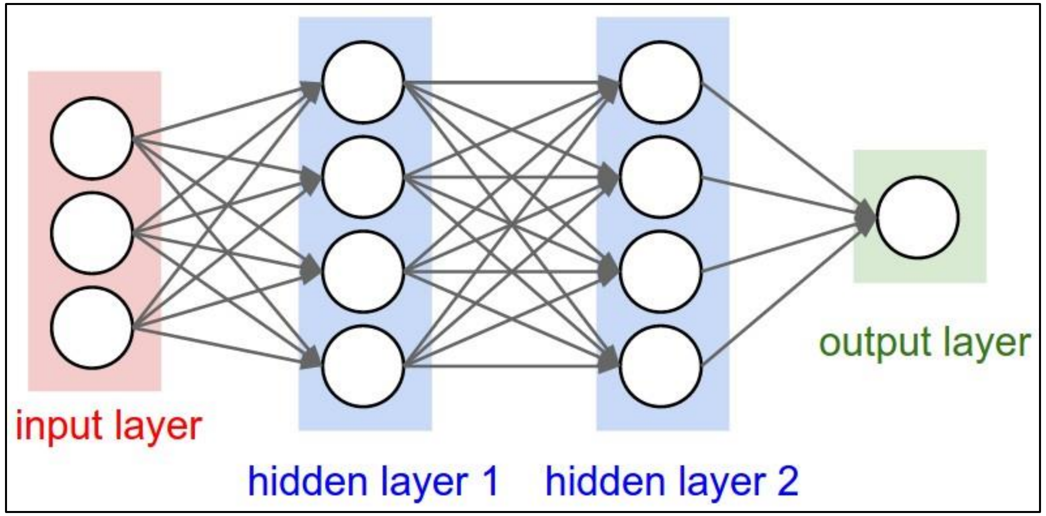
Neural Networks: Layer Architecture





Deep Learning

"Deep" Neural Network (DNN)



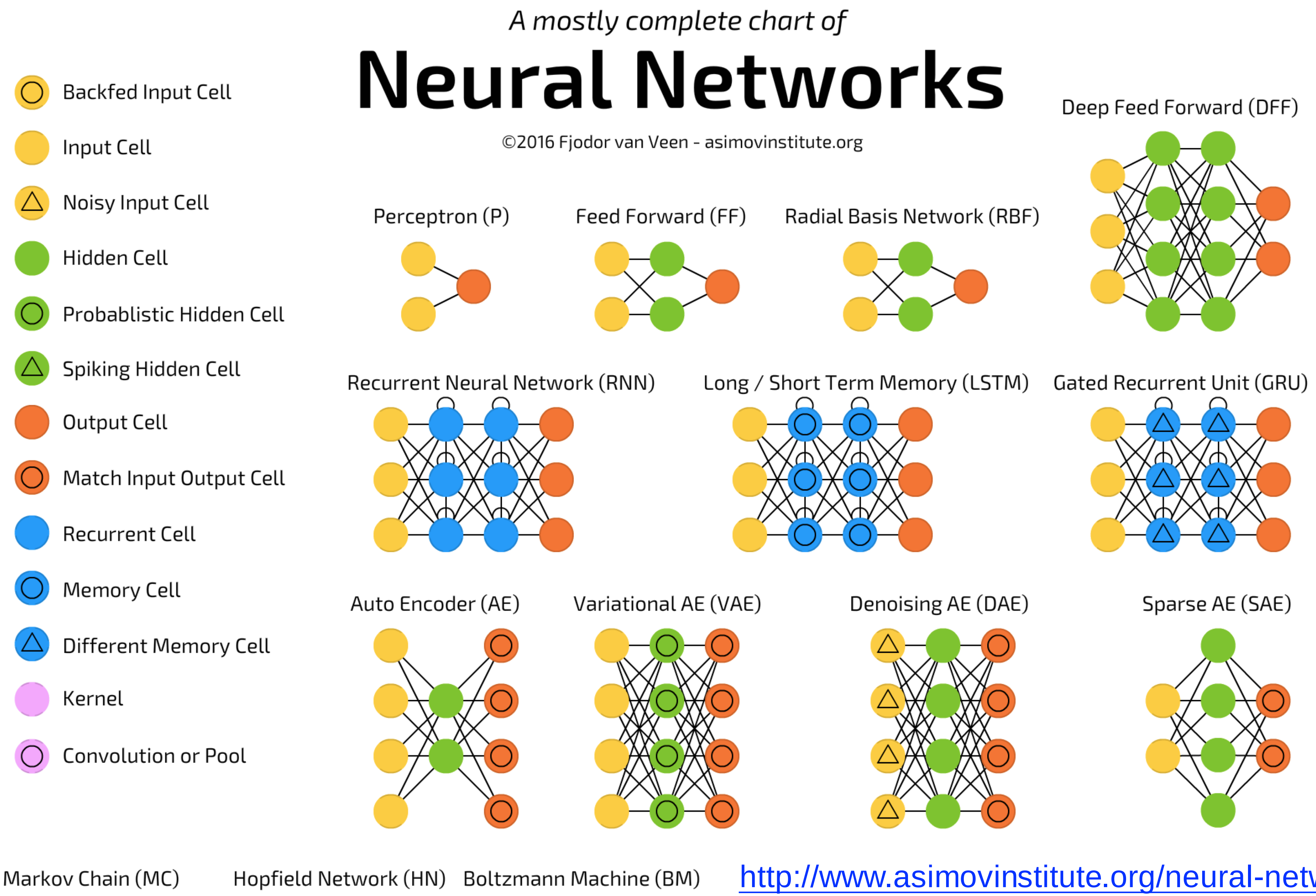
(Multilayer Perceptron, MLP)

[Source: Stanford CS231n]

Deep Learning

- A subset of machine learning algorithms
- Artificial neural networks with **more than one hidden layers**
- Using a cascade of many layers of nonlinear processing units for feature extraction and transformation
- Each successive layer uses the output from the previous layer as input
- Using the **learned weights** instead of hand-coded features

The Neural Network Zoo



http://www.asimovinstitute.org/neural-network-zoo/

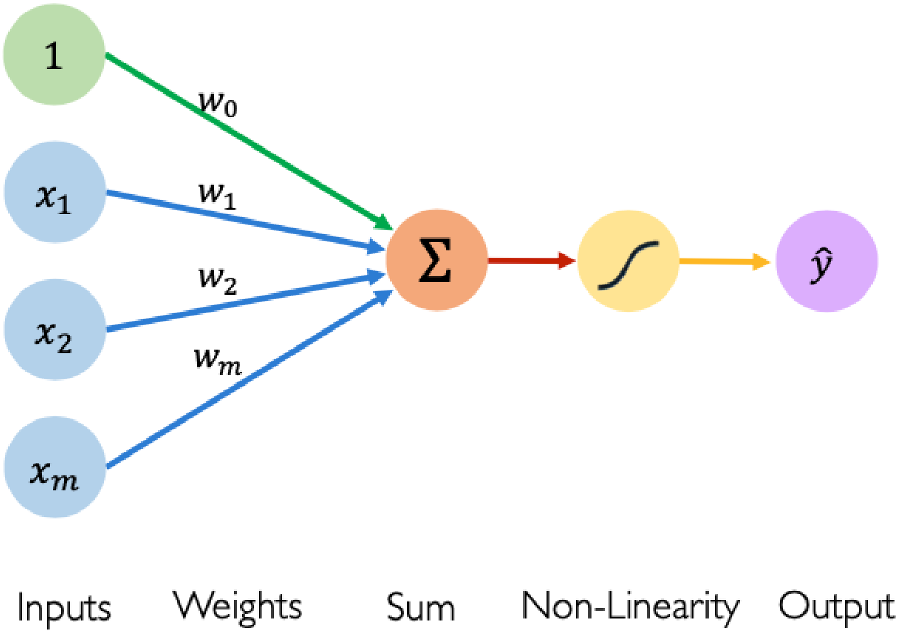
On Training Deep Neural Networks

- Prof. Alexander Amini,
  Lecture 1: Introduction to Deep Learning, MIT 6.S191
- http://introtodeeplearning.com

  https://youtu.be/ErnWZxJovaM?si=_qZNfceTdMLInBkI

The Perceptron: Forward Propagation

(Single neuron)



Output

Linear combination of inputs

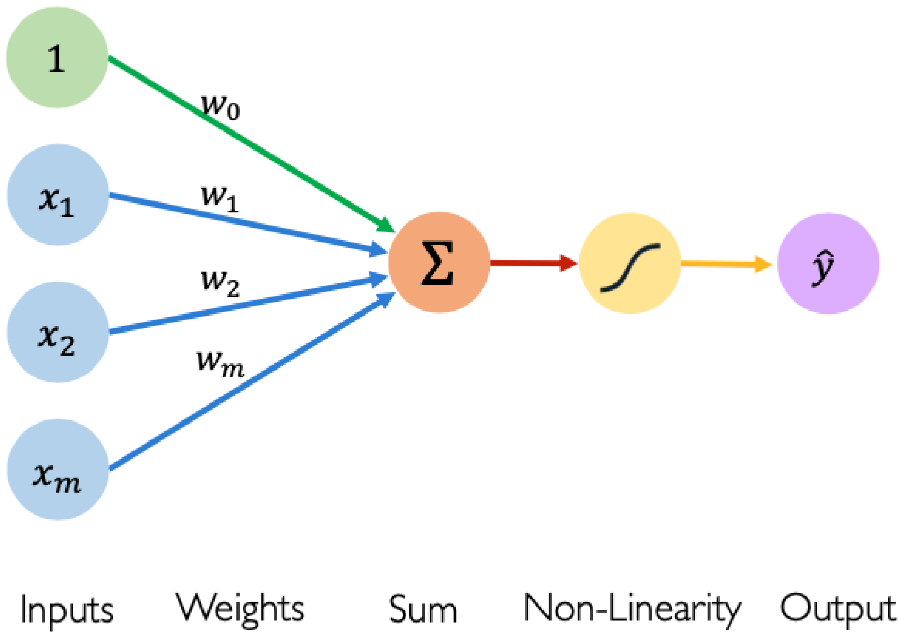
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

Non-linear activation function

Bias

[Source: Prof. Alexander Amini, MIT 6.S191]

The Perceptron: Forward Propagation



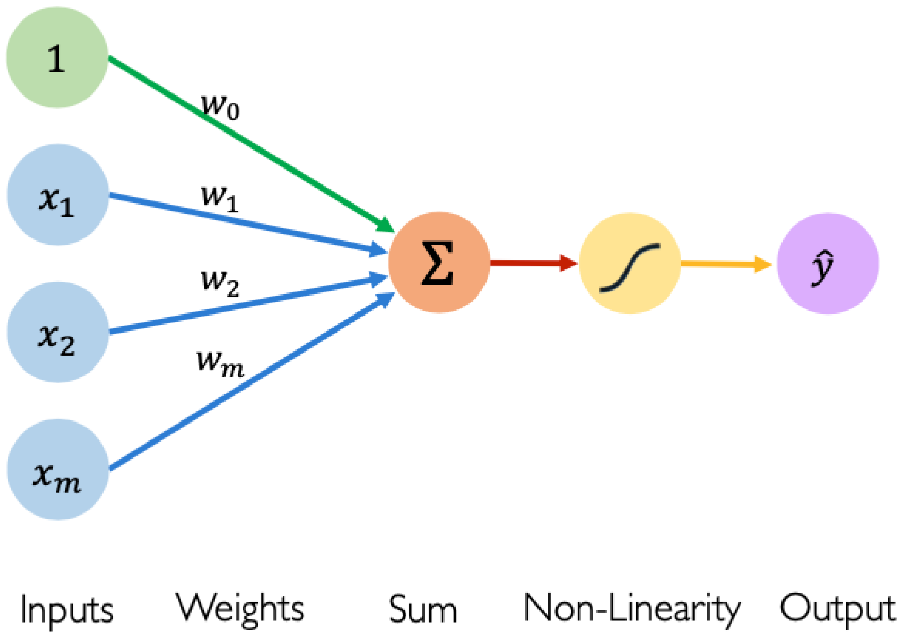
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

[Source: Prof. Alexander Amini, MIT 6.S191]

The Perceptron: Forward Propagation

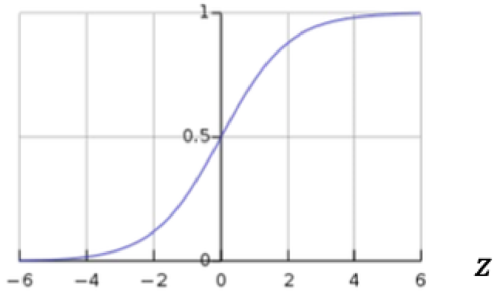


Activation Functions

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

- Example: sigmoid function

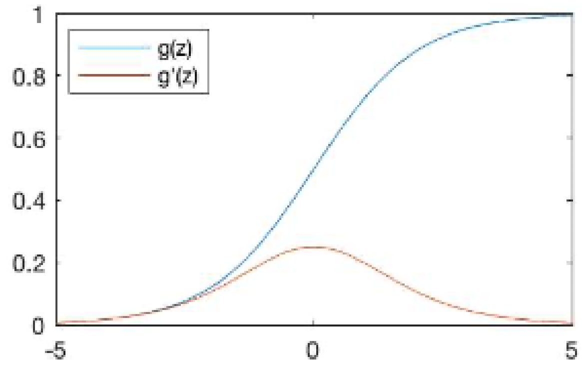
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



[Source: Prof. Alexander Amini, MIT 6.S191]

Common Activation Functions

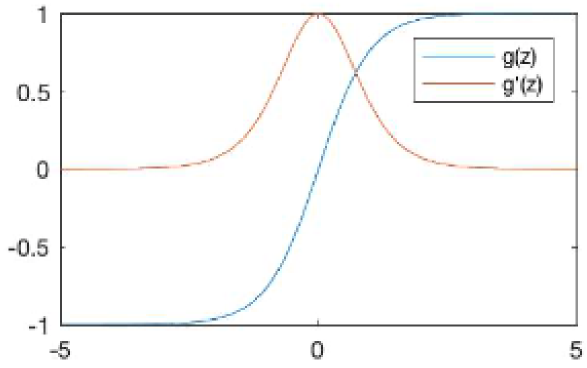
Sigmoid Function



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

$$g'(z) = g(z)(1 - g(z))$$

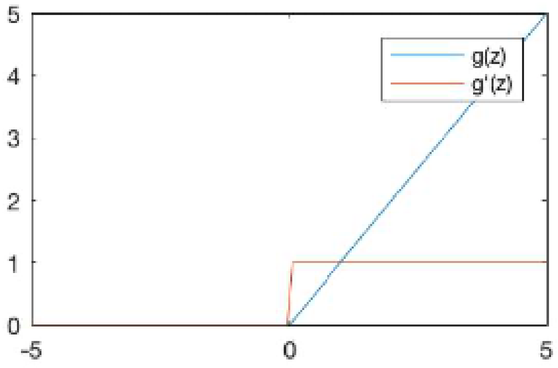
Hyperbolic Tangent



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

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



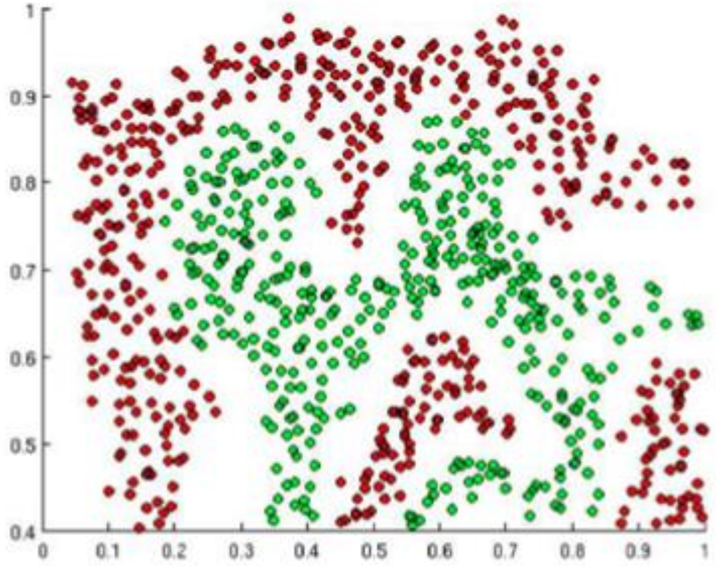
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

[Source: Prof. Alexander Amini, MIT 6.S191]

Importance of Activation Functions

The purpose of activation functions is to ***introduce non-linearities*** *into the network*

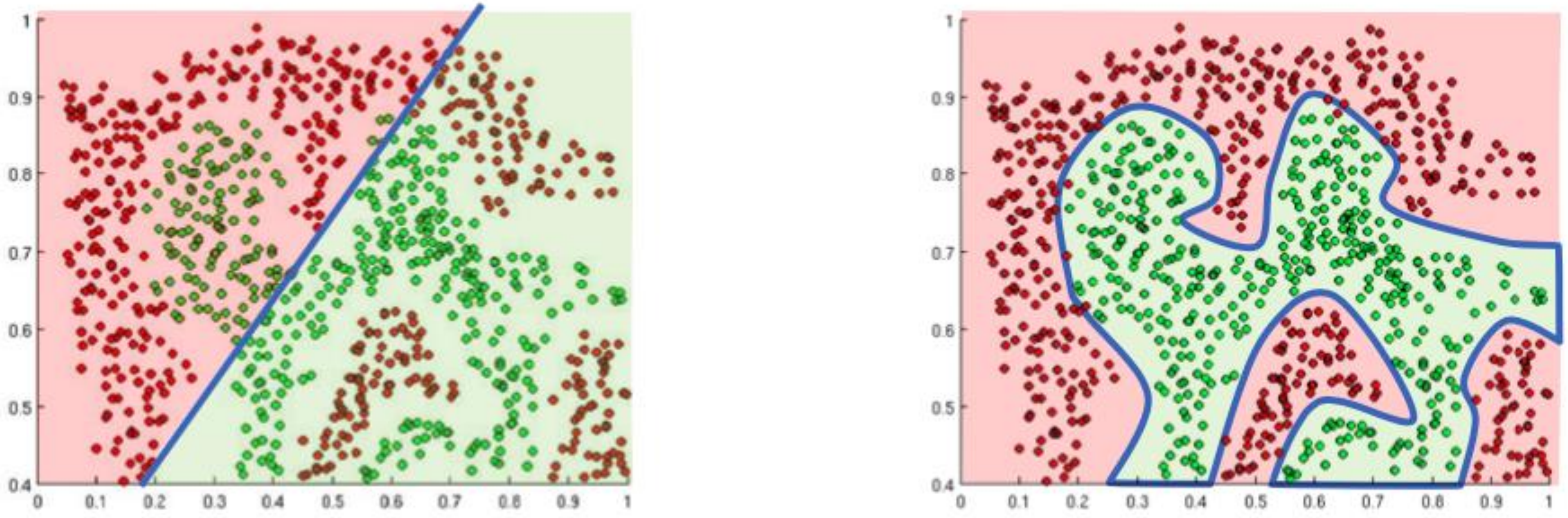


What if we wanted to build a Neural Network to distinguish green vs red points?

[Source: Prof. Alexander Amini, MIT 6.S191]

Importance of Activation Functions

The purpose of activation functions is to ***introduce non-linearities*** *into the network*



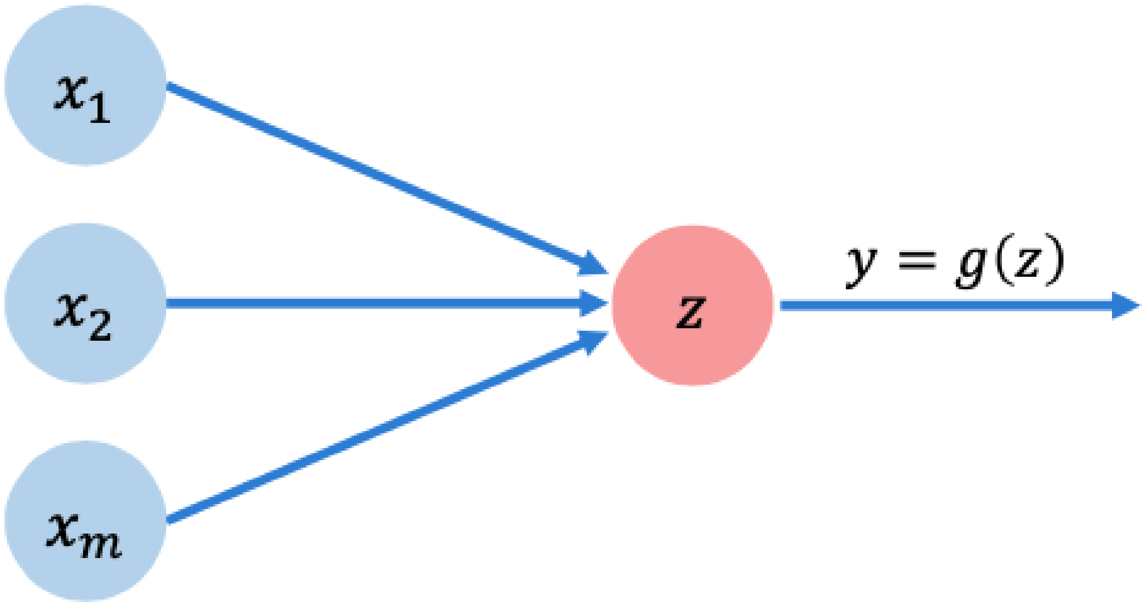
Linear Activation functions produce linear decisions no matter the network size

Non-linearities allow us to approximate arbitrarily complex functions

[Source: Prof. Alexander Amini, MIT 6.S191]

The Perceptron: Simplified

(Bias removed for simplicity)

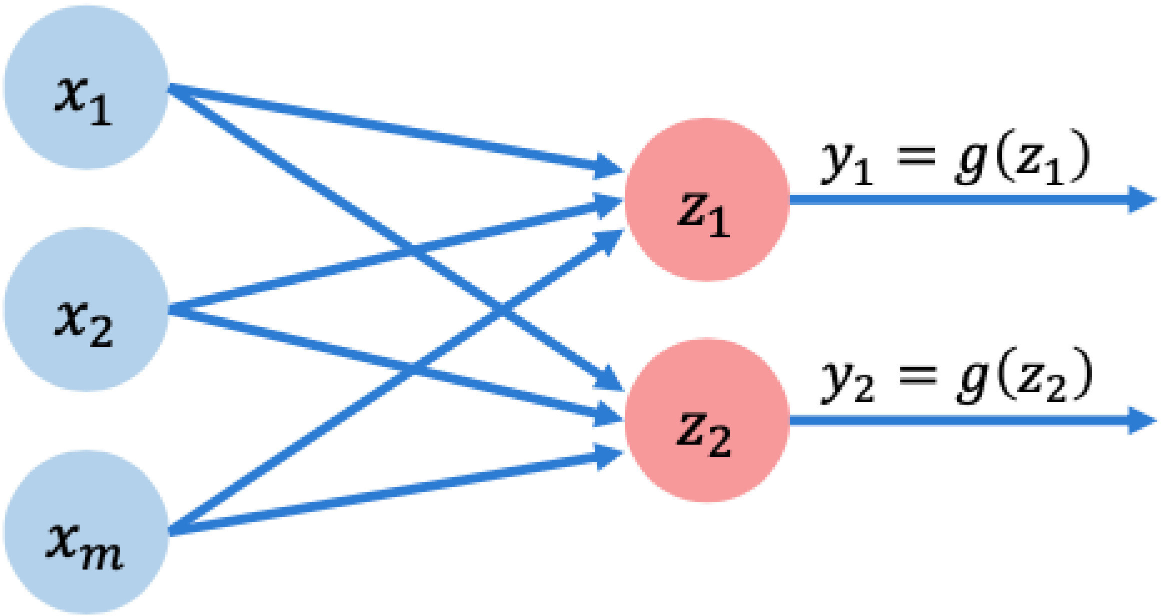


$$z = w_0 + \sum_{j=1}^{m} x_j \, w_j$$

[Source: Prof. Alexander Amini, MIT 6.S191]

Multi-Output Perceptron

All inputs are densely connected to all outputs: **dense layer** or **fully connected layer**

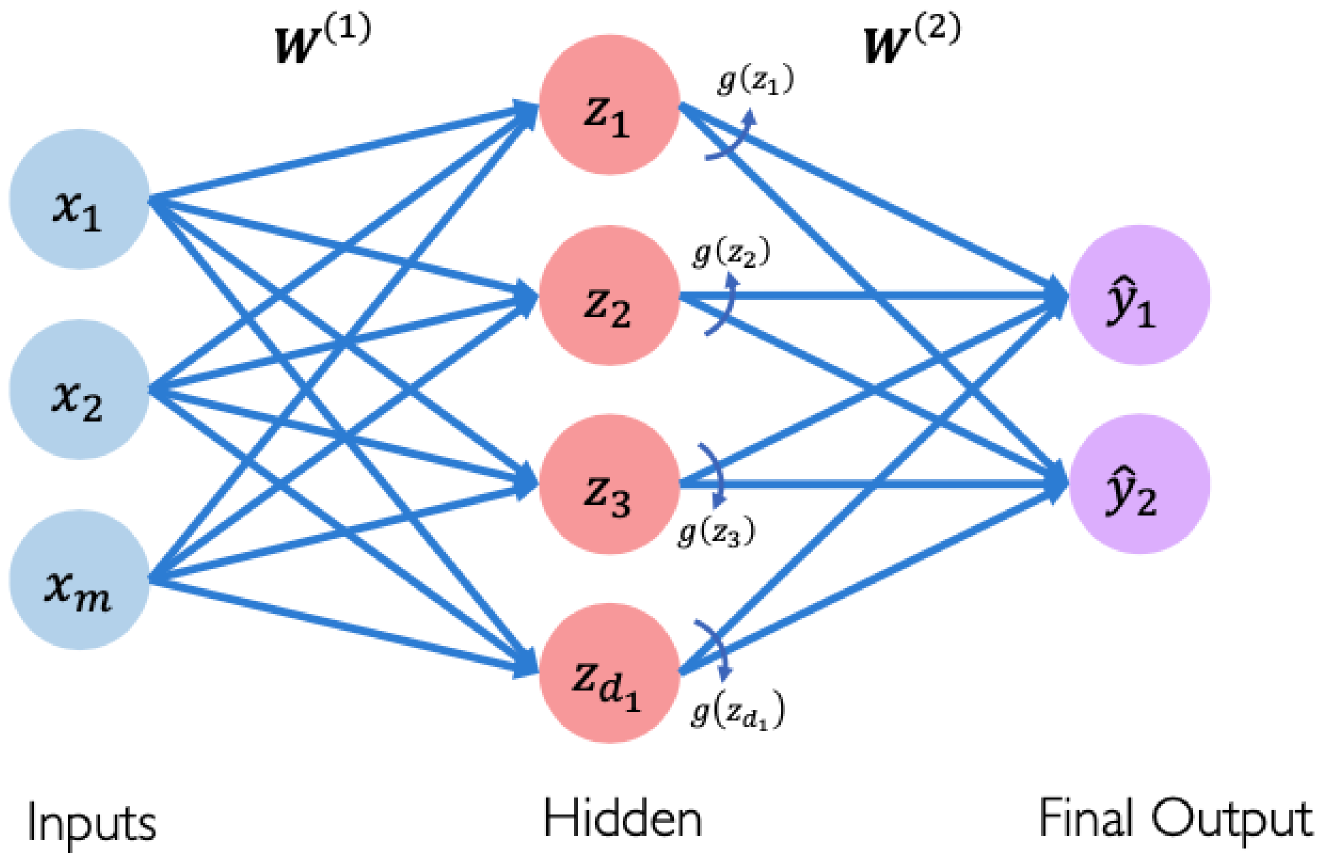


$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

[Source: Prof. Alexander Amini, MIT 6.S191]

Single-Layer Neural Network

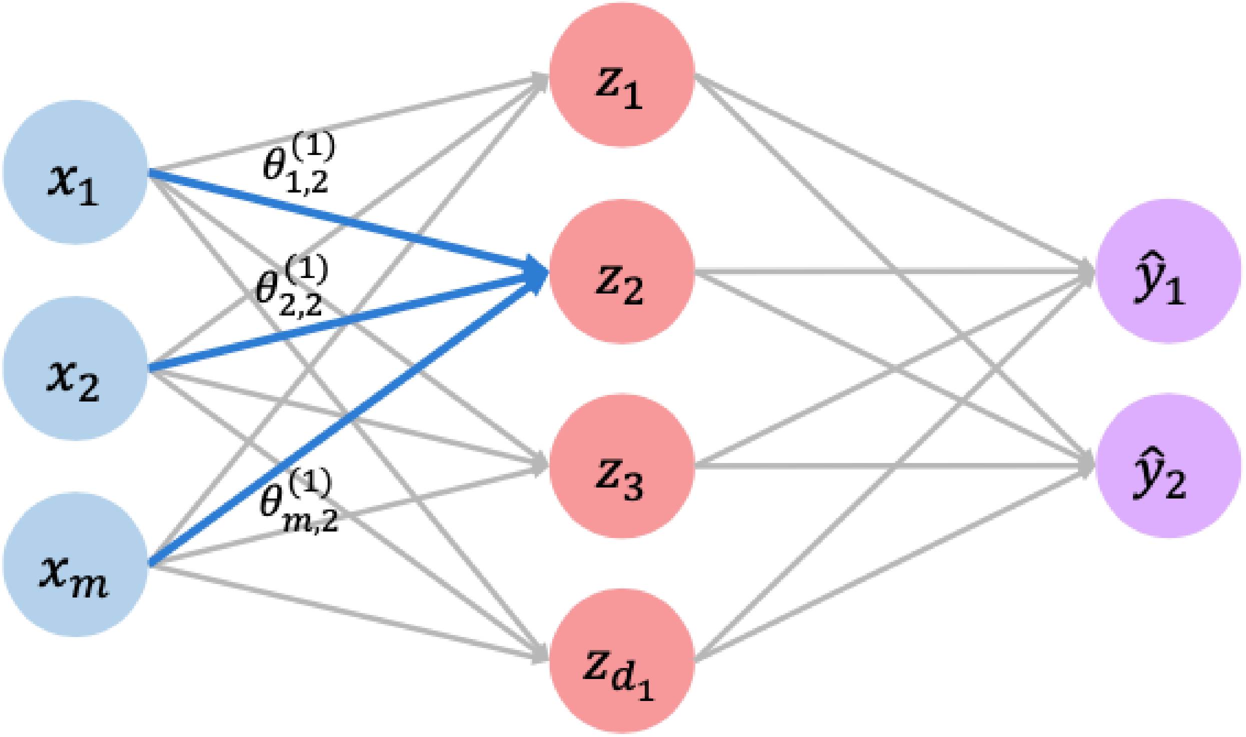
(Single hidden layer)



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, w_{j,i}^{(2)}\right)$$

[Source: Prof. Alexander Amini, MIT 6.S191]

Single-Layer Neural Network



$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

$$= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}$$

[Source: Prof. Alexander Amini, MIT 6.S191]

Example

Will I pass this class?

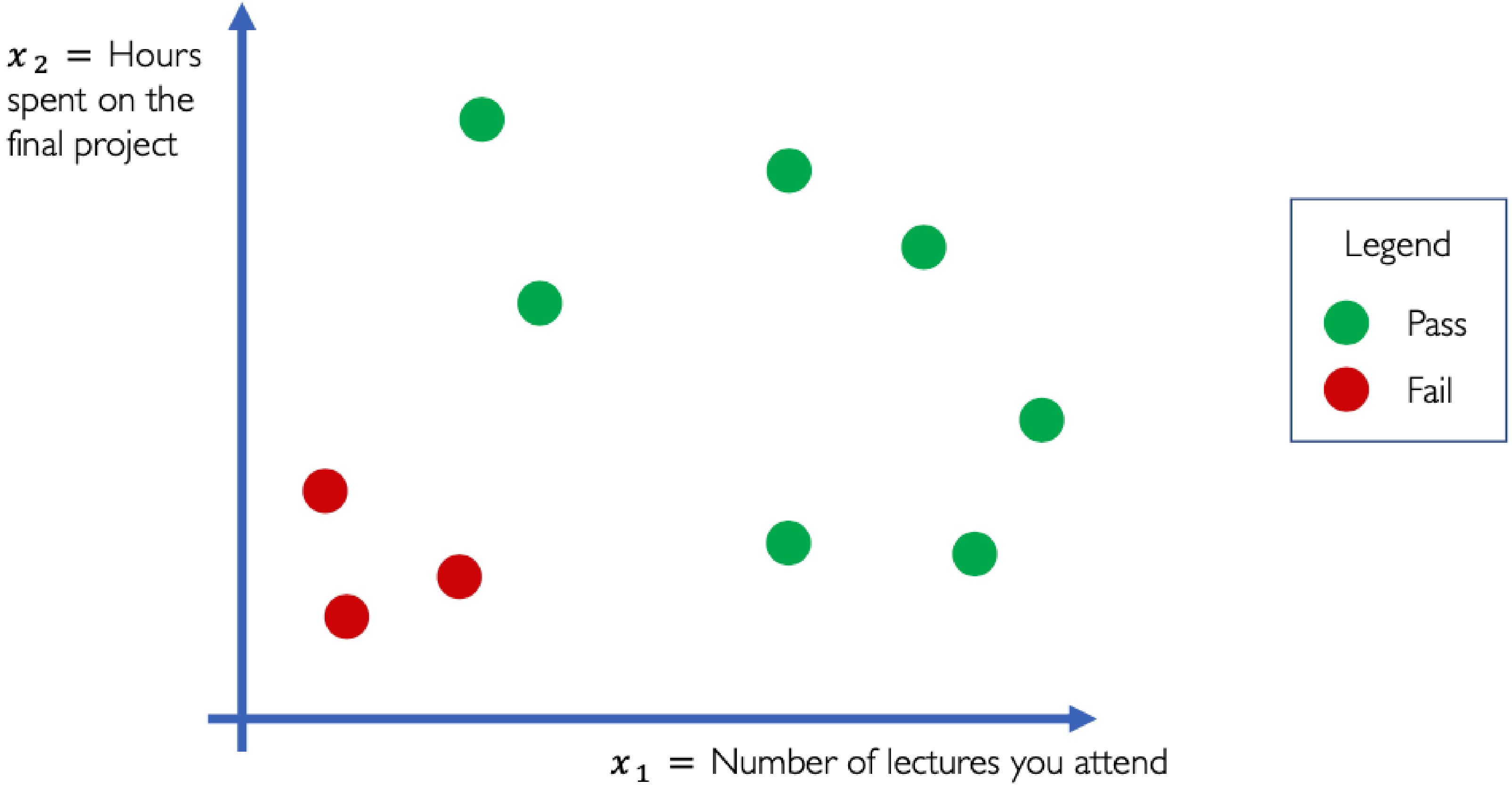
Let's start with a simple two feature model

x_1 = Number of lectures you attend

x_2 = Hours spent on the final project

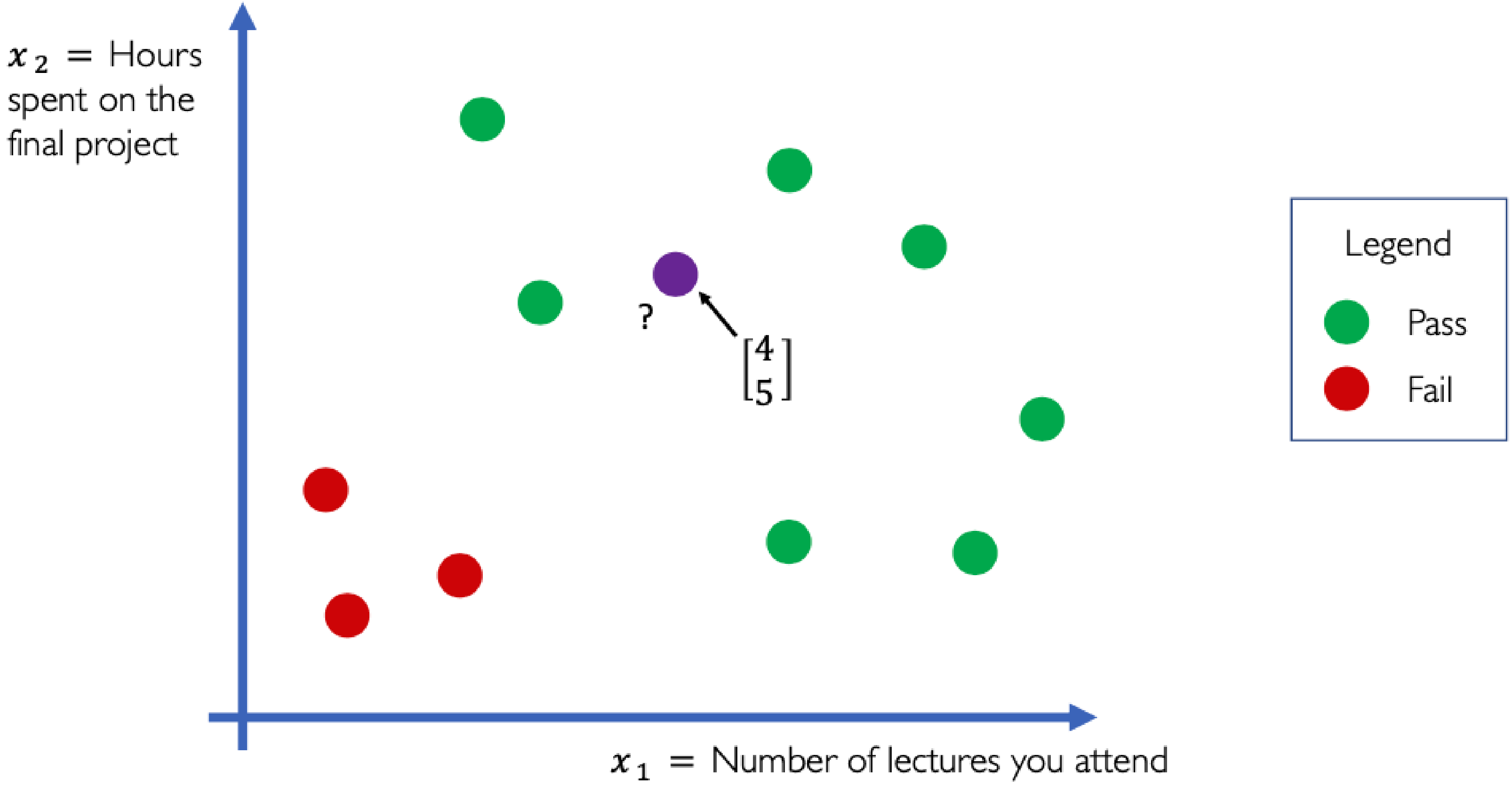
[Source: Prof. Alexander Amini, MIT 6.S191]

Example Problem: Will I pass this class?



[Source: Prof. Alexander Amini, MIT 6.S191]

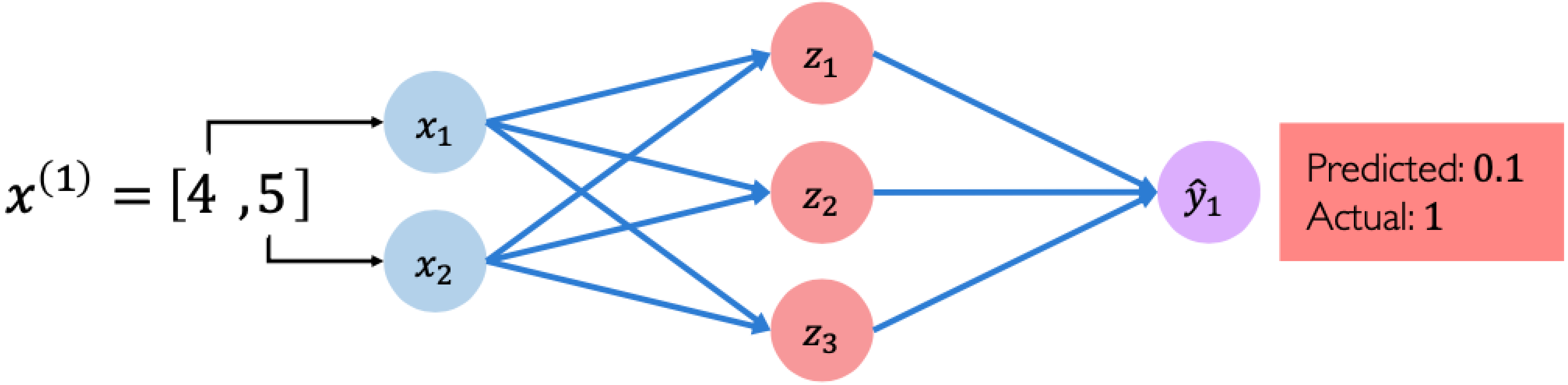
Example Problem: Will I pass this class?



[Source: Prof. Alexander Amini, MIT 6.S191]

Quantifying Loss

*The **loss** of our network measures the cost incurred from incorrect predictions*



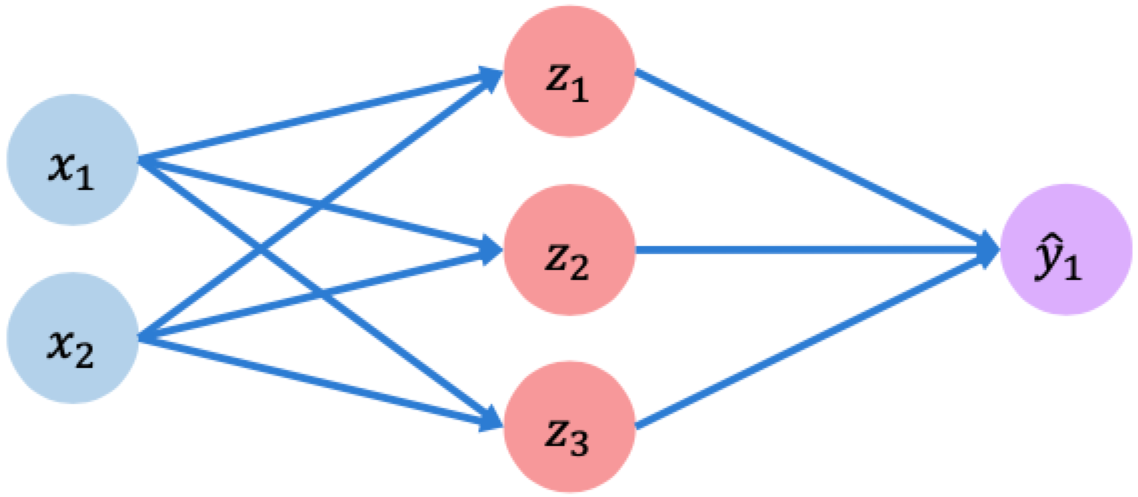
$$\mathcal{L}\left(\underbrace{f\left(x^{(i)}; \boldsymbol{W}\right)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\right)$$

[Source: Prof. Alexander Amini, MIT 6.S191]

Empirical Loss

*The **empirical loss** measures the total loss over our entire dataset*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{array}{cc} f(x) & y \\ \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

Also known as:
- Objective function
- Cost function
- Empirical Risk

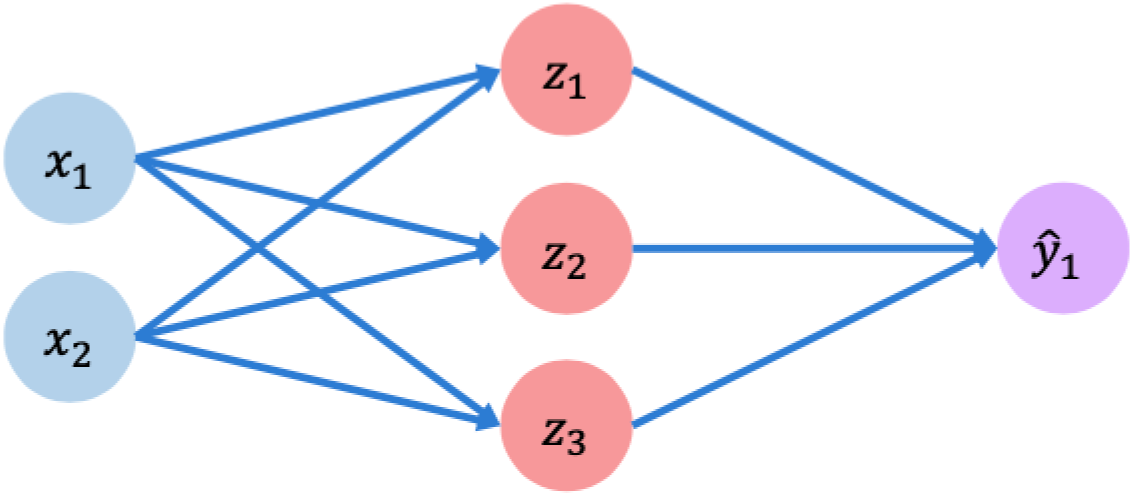
$$\boldsymbol{J}(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\Big(\underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\Big)$$

[Source: Prof. Alexander Amini, MIT 6.S191]

Binary Cross Entropy Loss

Cross entropy loss *can be used with models that output a probability between 0 and 1*

$$X = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



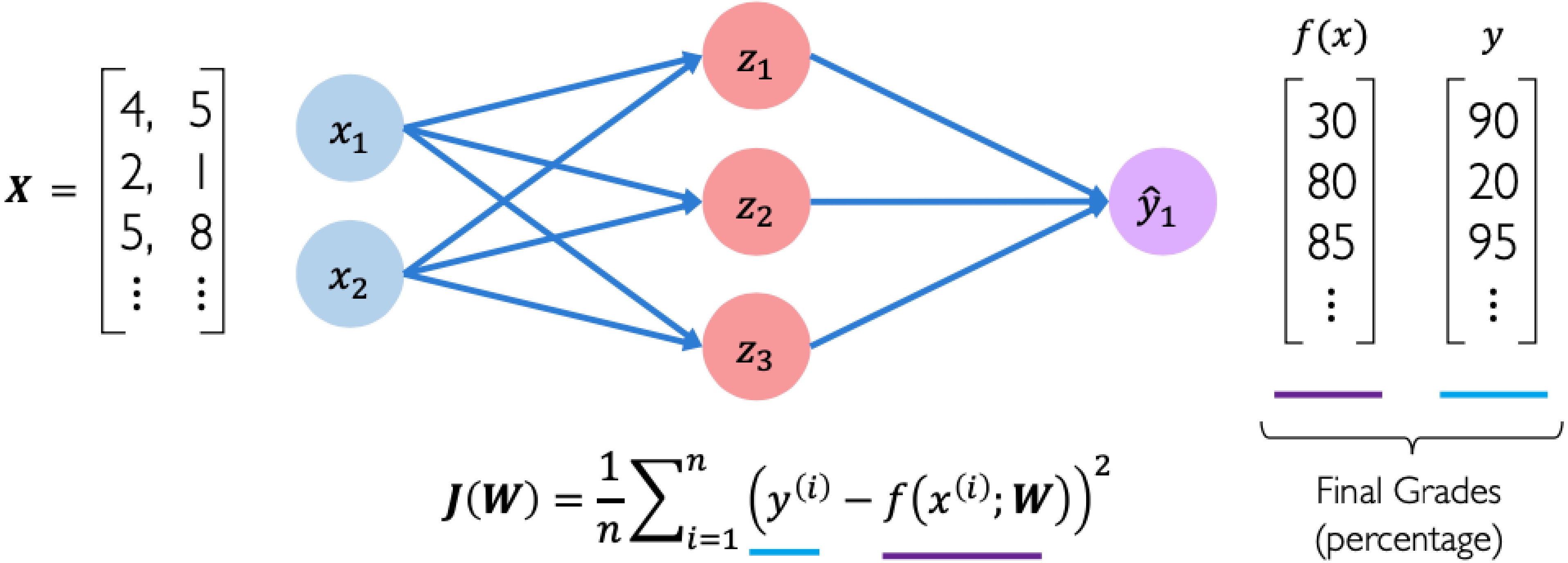
$$f(x) = \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \qquad y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

$$J(\boldsymbol{W}) = -\frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log\Big(\underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big) + (1-\underbrace{y^{(i)}}_{\text{Actual}}) \log\Big(1-\underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)$$

[Source: Prof. Alexander Amini, MIT 6.S191]

Mean Squared Error Loss

Mean squared error loss *can be used with regression models that output continuous real numbers*



$$\boldsymbol{J}(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)^2$$

[Source: Prof. Alexander Amini, MIT 6.S191]

Loss Optimization

We want to find the network weights that ***achieve the lowest loss***

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

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

Remember:

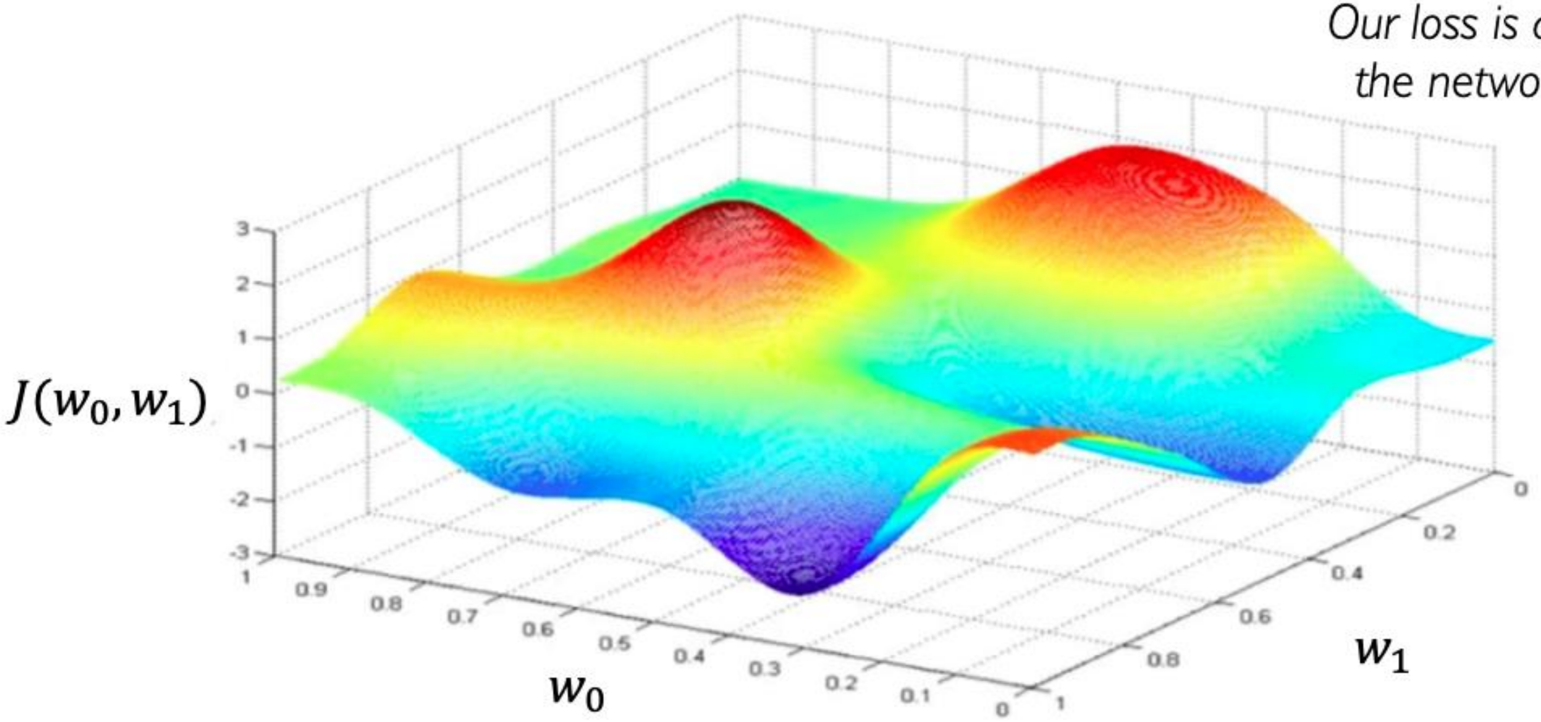
$$\boldsymbol{W} = \left\{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \cdots\right\}$$

[Source: Prof. Alexander Amini, MIT 6.S191]

Loss Optimization

To minimize J(W)

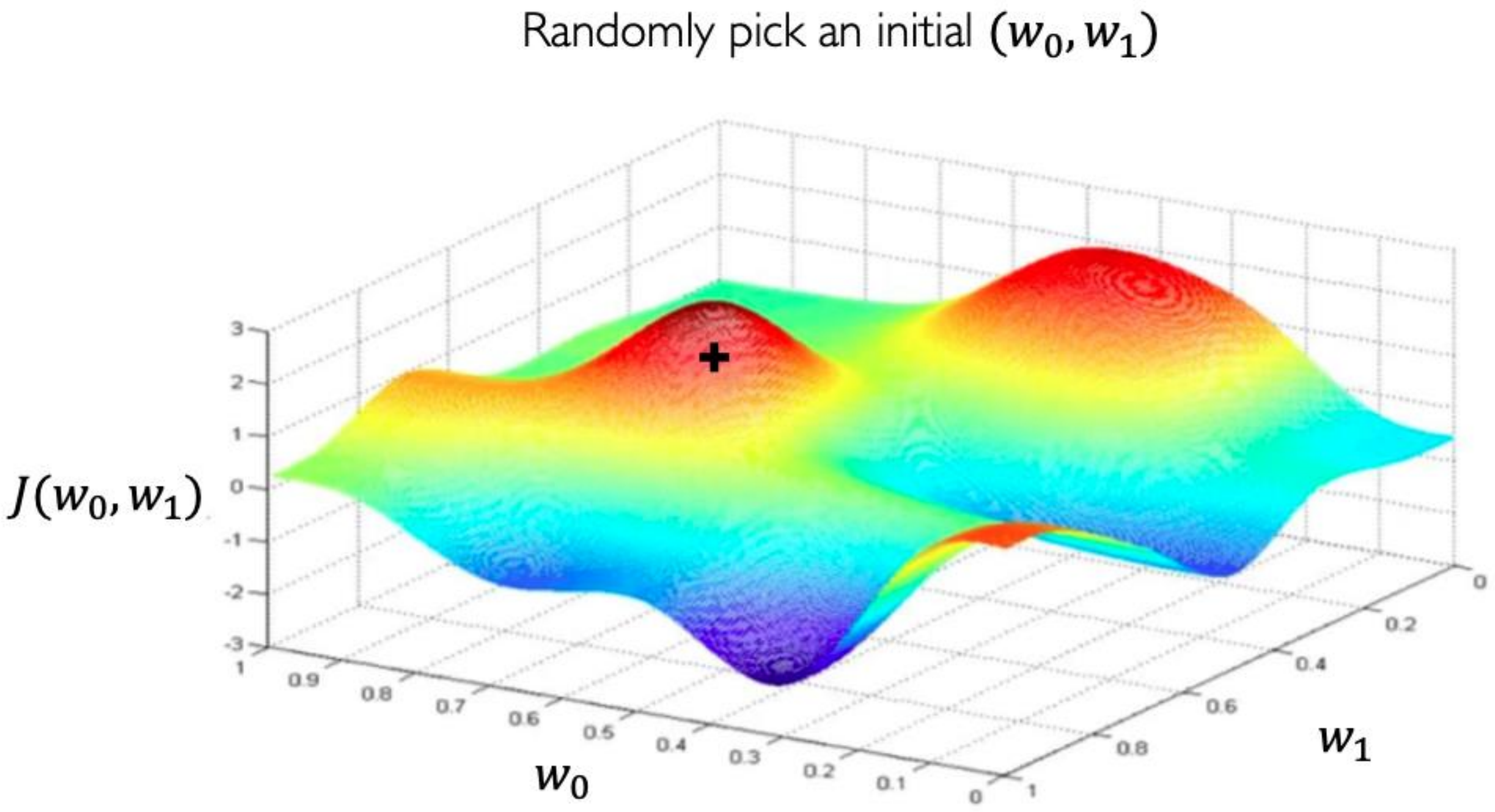
$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$



Remember:
Our loss is a function of the network weights!

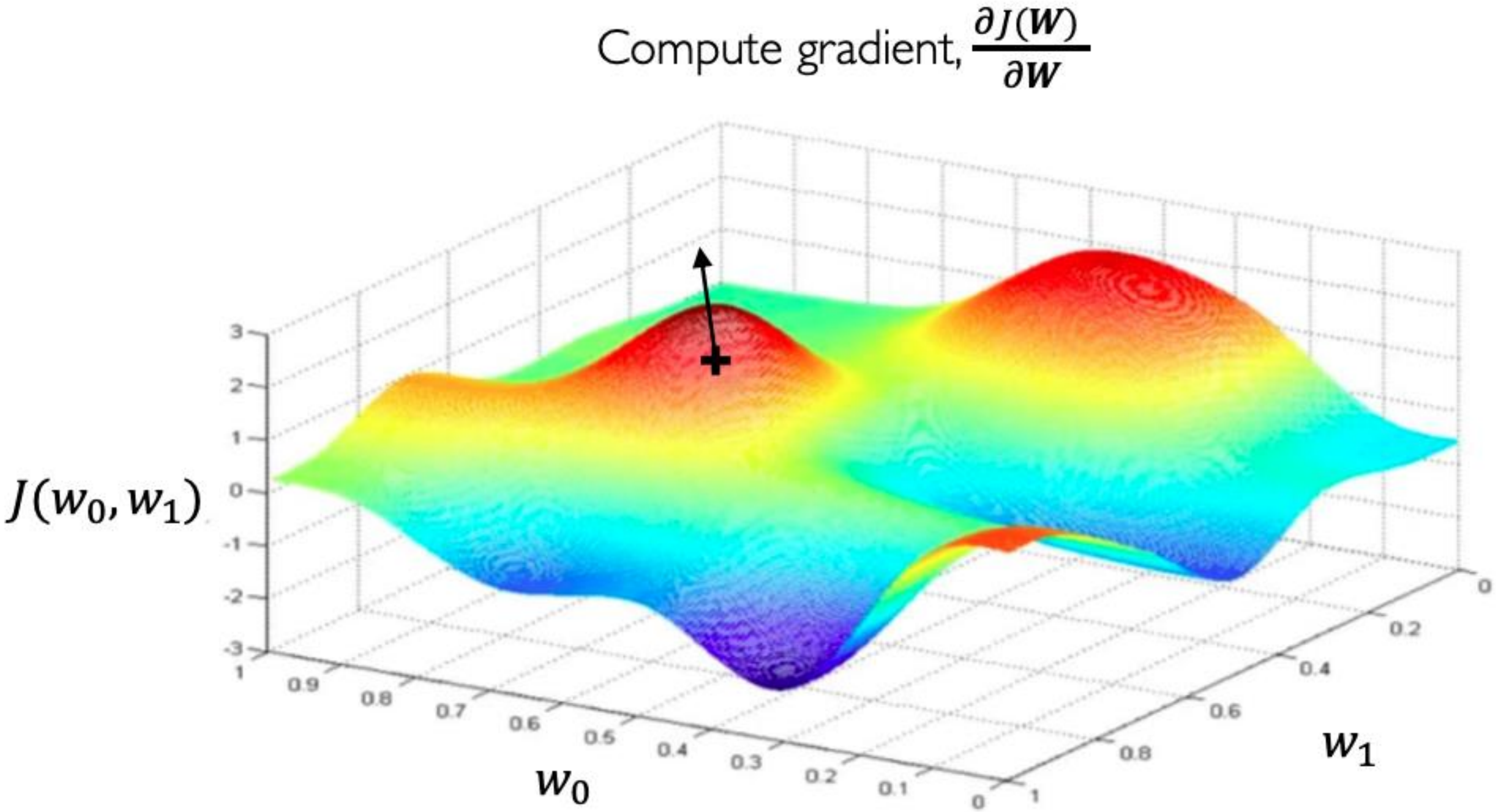
[Source: Prof. Alexander Amini, MIT 6.S191]

Loss Optimization



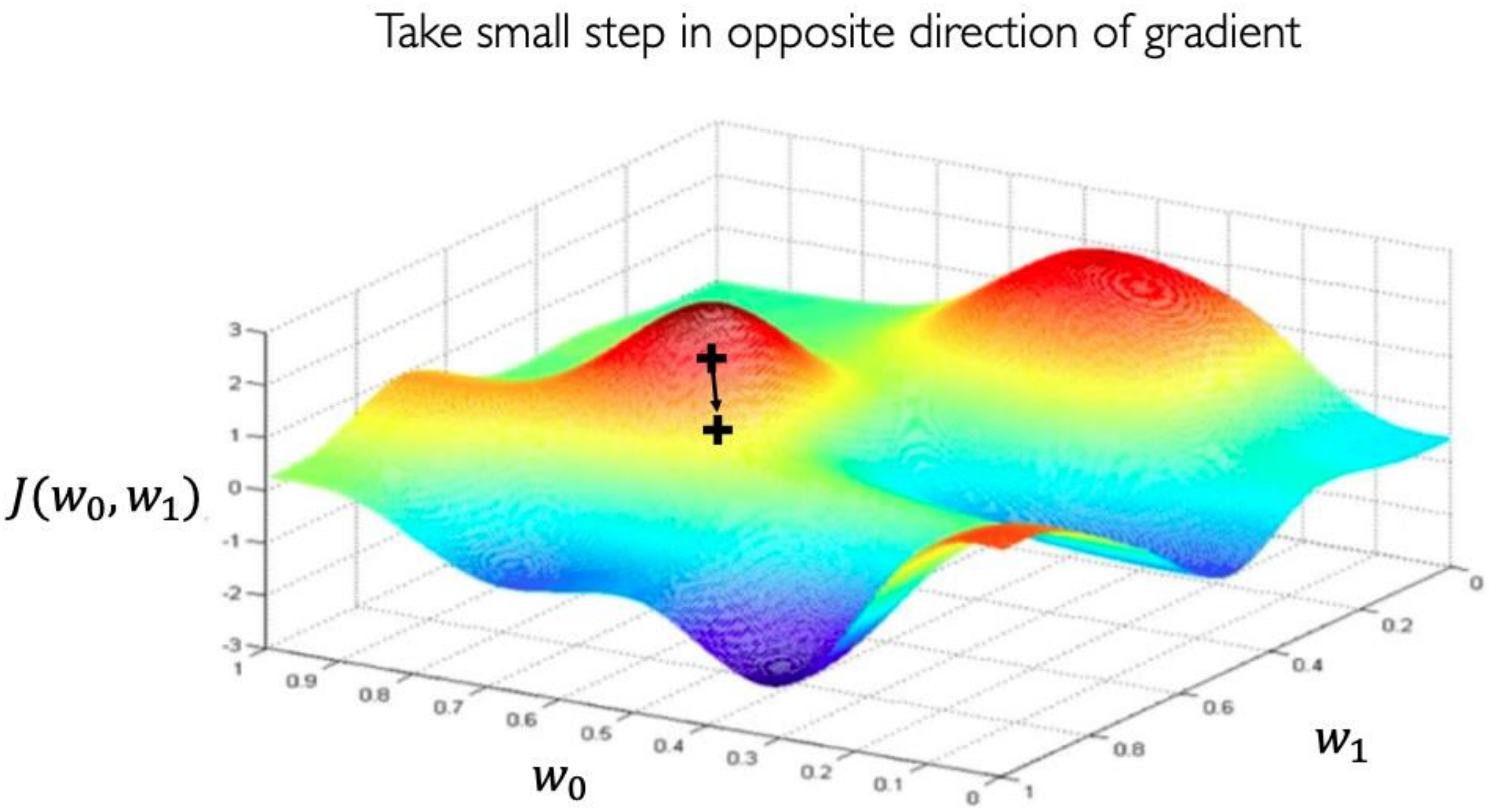
[Source: Prof. Alexander Amini, MIT 6.S191]

Loss Optimization



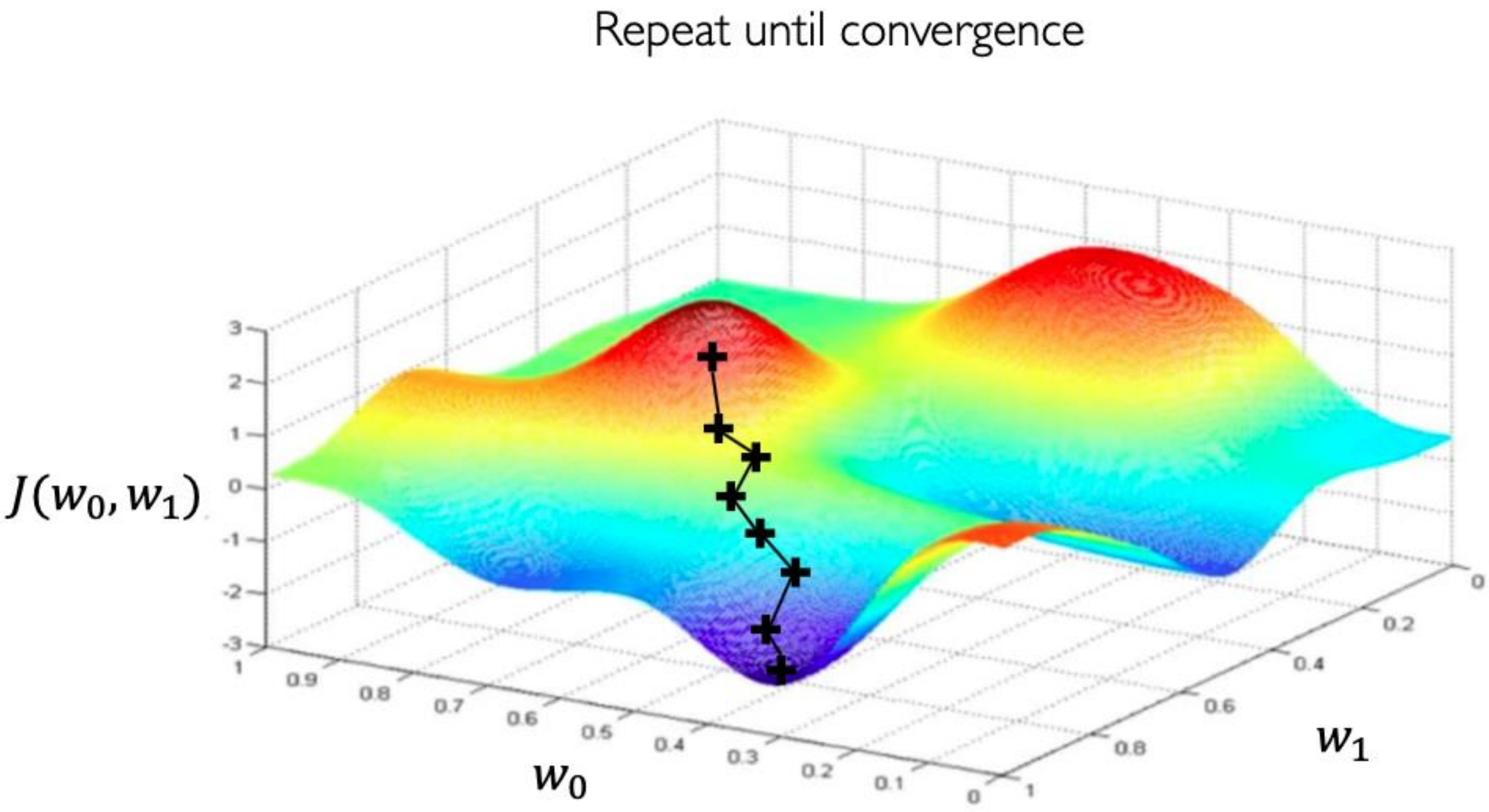
[Source: Prof. Alexander Amini, MIT 6.S191]

Loss Optimization



[Source: Prof. Alexander Amini, MIT 6.S191]

Loss Optimization



[Source: Prof. Alexander Amini, MIT 6.S191]

Gradient Descent

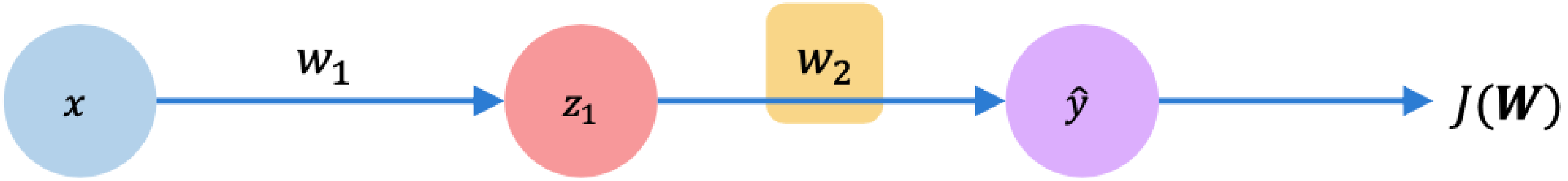
Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Learning rate

[Source: Prof. Alexander Amini, MIT 6.S191]

Computing Gradients: Backpropagation



How does a small change in one weight (ex. w_2) affect the final loss $J(\boldsymbol{W})$?

$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \underbrace{\frac{\partial J(\boldsymbol{W})}{\partial \hat{y}}} * \underbrace{\frac{\partial \hat{y}}{\partial w_2}}$$

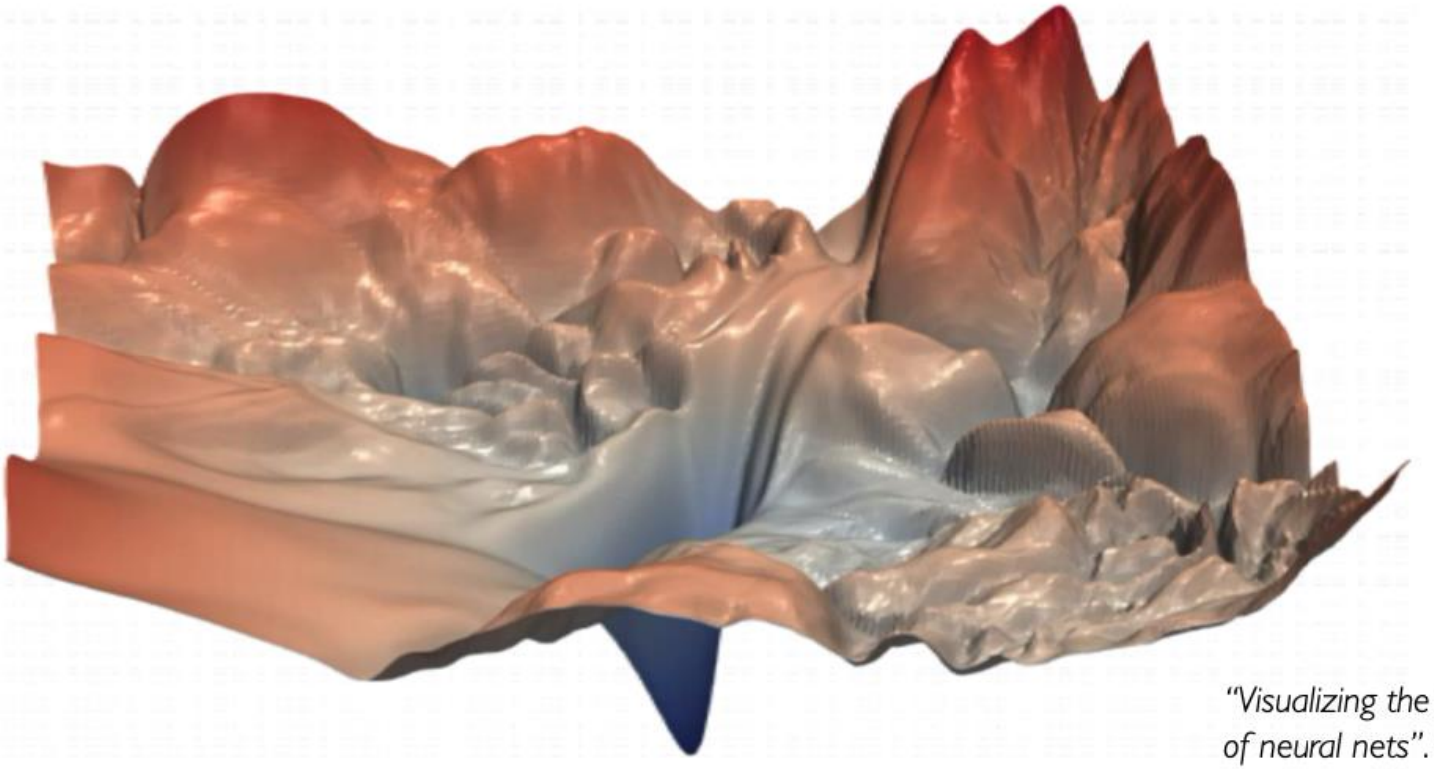
Chain rule!

$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \underbrace{\frac{\partial J(\boldsymbol{W})}{\partial \hat{y}}} * \underbrace{\frac{\partial \hat{y}}{\partial z_1}} * \underbrace{\frac{\partial z_1}{\partial w_1}}$$

*Repeat this for **every weight in the network** using gradients from later layers*

[Source: Prof. Alexander Amini, MIT 6.S191]

Training Neural Networks is Difficult



"Visualizing the loss landscape of neural nets". Dec 2017.

[Source: Prof. Alexander Amini, MIT 6.S191]

Loss Functions Can Be Difficult to Optimize

Remember:

Optimization through gradient descent

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \boxed{\eta} \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

How can we set the learning rate?

Hyperparameter!

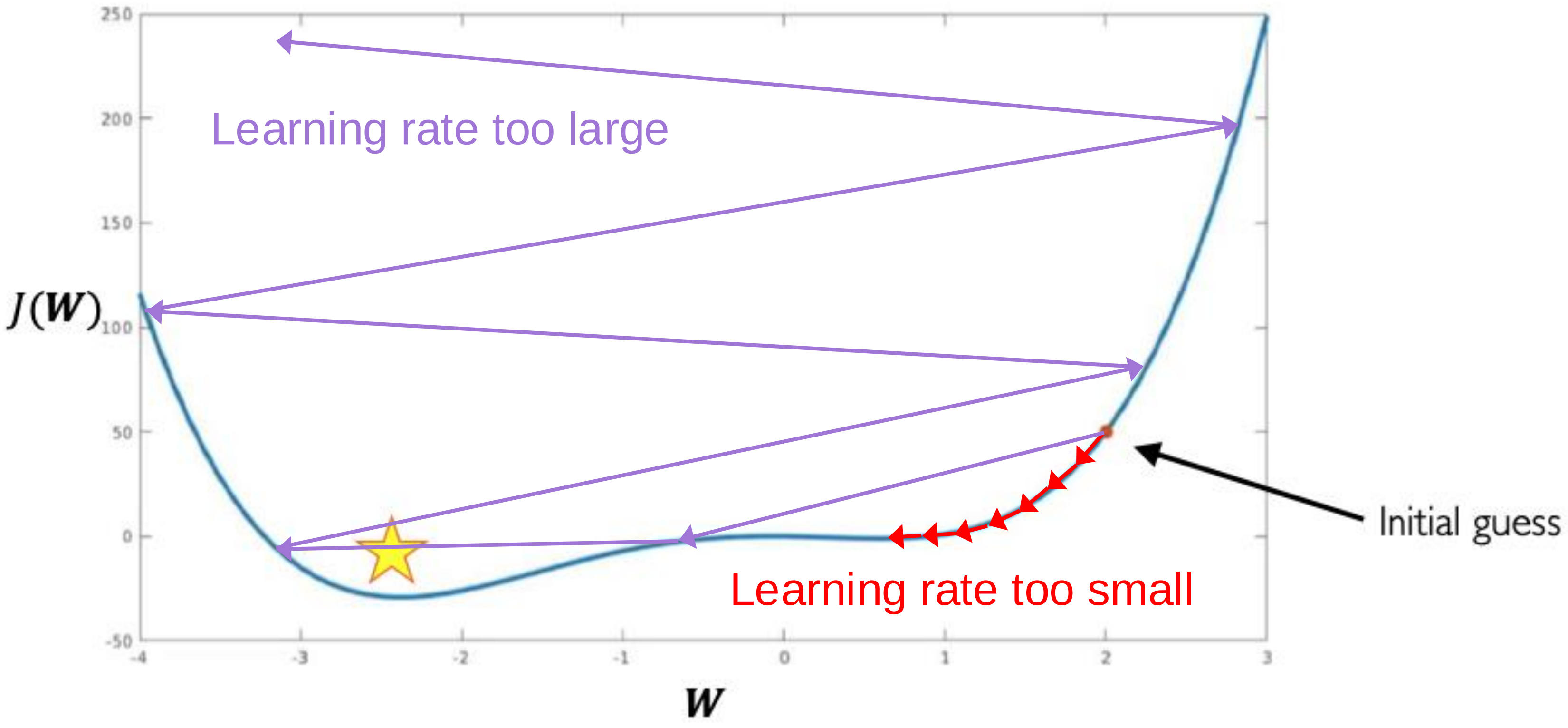
[Source: Prof. Alexander Amini, MIT 6.S191]

Setting the Learning Rate

Small learning rate *converges slowly and gets stuck in false local minima*

Large learning rates *overshoot, become unstable and diverge*

Stable learning rates *converge smoothly and avoid local minima*



[Source: Prof. Alexander Amini, MIT 6.S191]

How to Choose A Good Learning Rate?

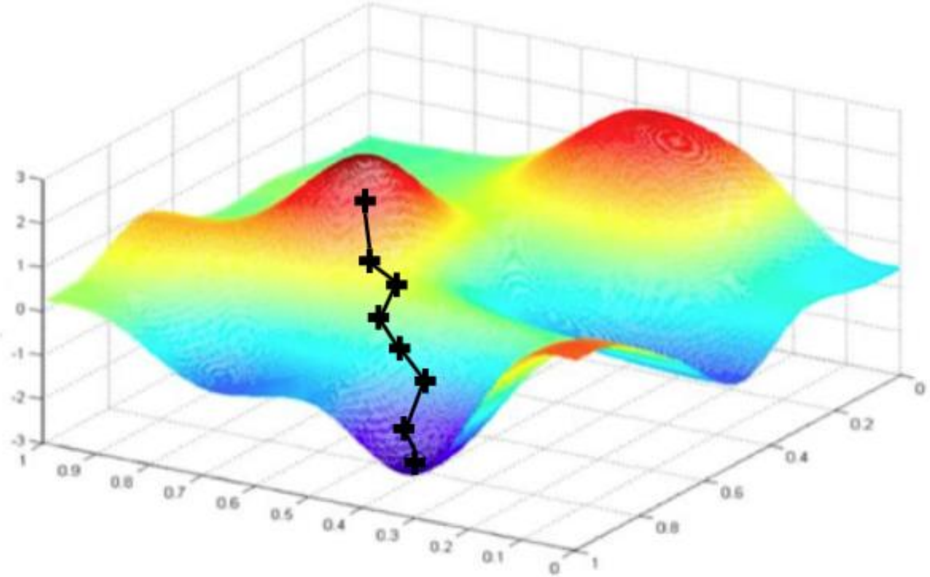
- Try lots of different learning rates and see what works "just right"
- Use an adaptive learning rate that "adapts" to the landscape
  - Learning rates can be larger or smaller dynamically, depending on
    - How large gradient is
    - How fast learning is happening
    - Size of particular weights
    - Etc.

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Can be very computational to compute!

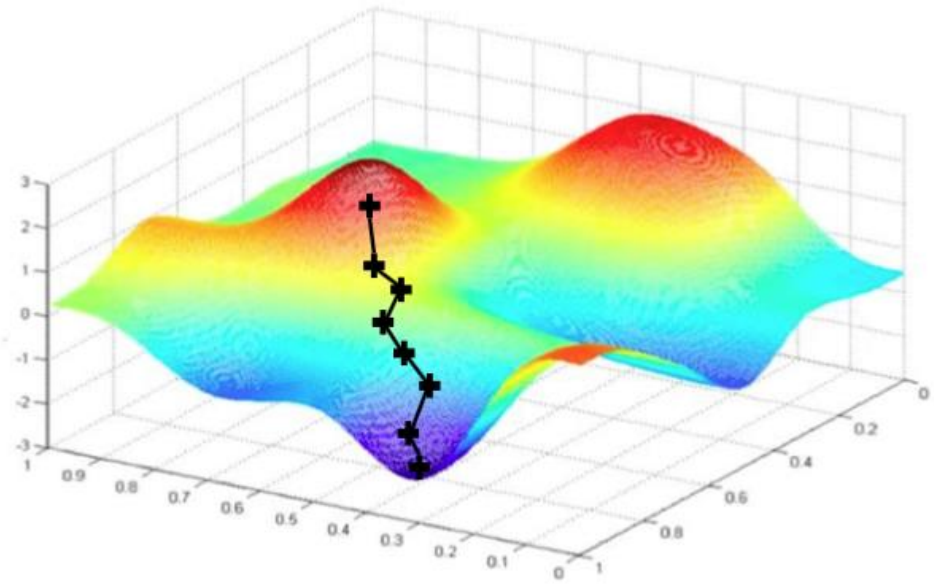


[Source: Prof. Alexander Amini, MIT 6.S191]

Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point $\boldsymbol{i}$
4. Compute gradient, $\boxed{\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



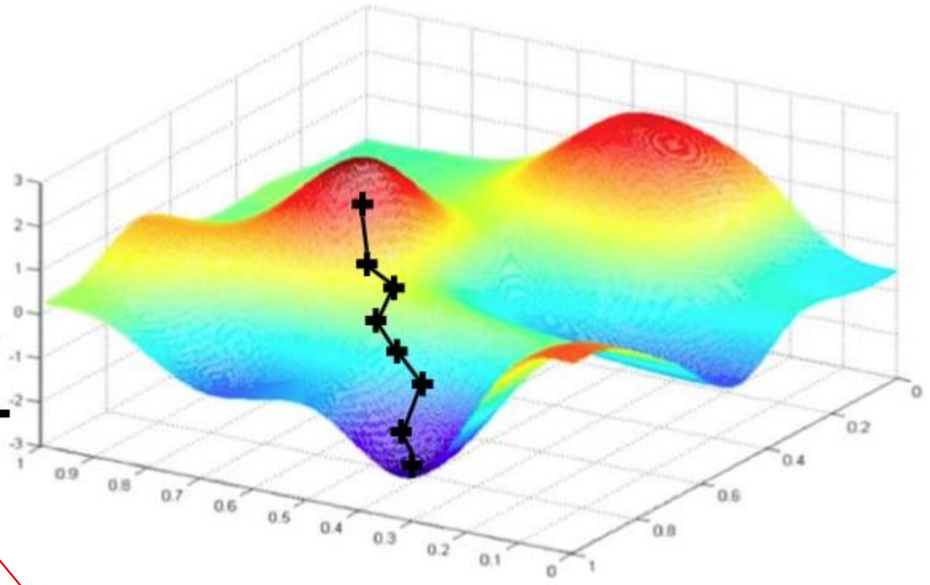
Easy to compute but
very noisy
(stochastic)!

[Source: Prof. Alexander Amini, MIT 6.S191]

Stochastic Gradient Descent (with Mini-Batches)

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



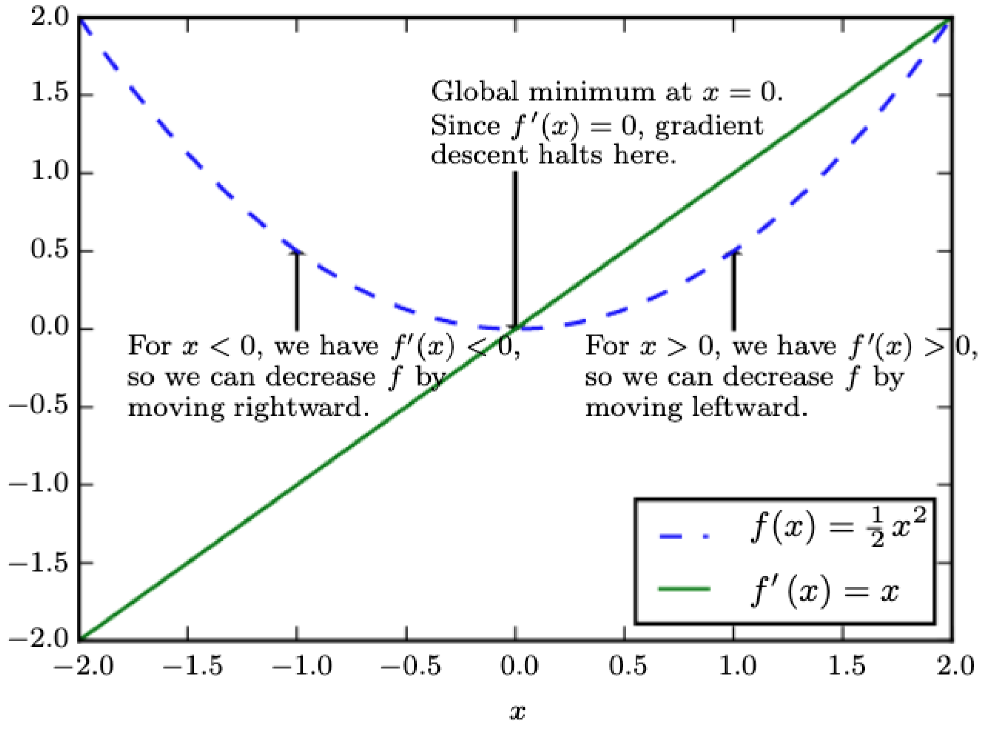
Fast to compute
A much better estimation of the true gradient!

Training with Mini-batches

- More accurate estimation of gradient
  - Smoother convergence
  - Allows for larger learning rates
- Mini-batches lead to fast training
  - Can parallelize computation
  - Achieve significant speed increases on GPU

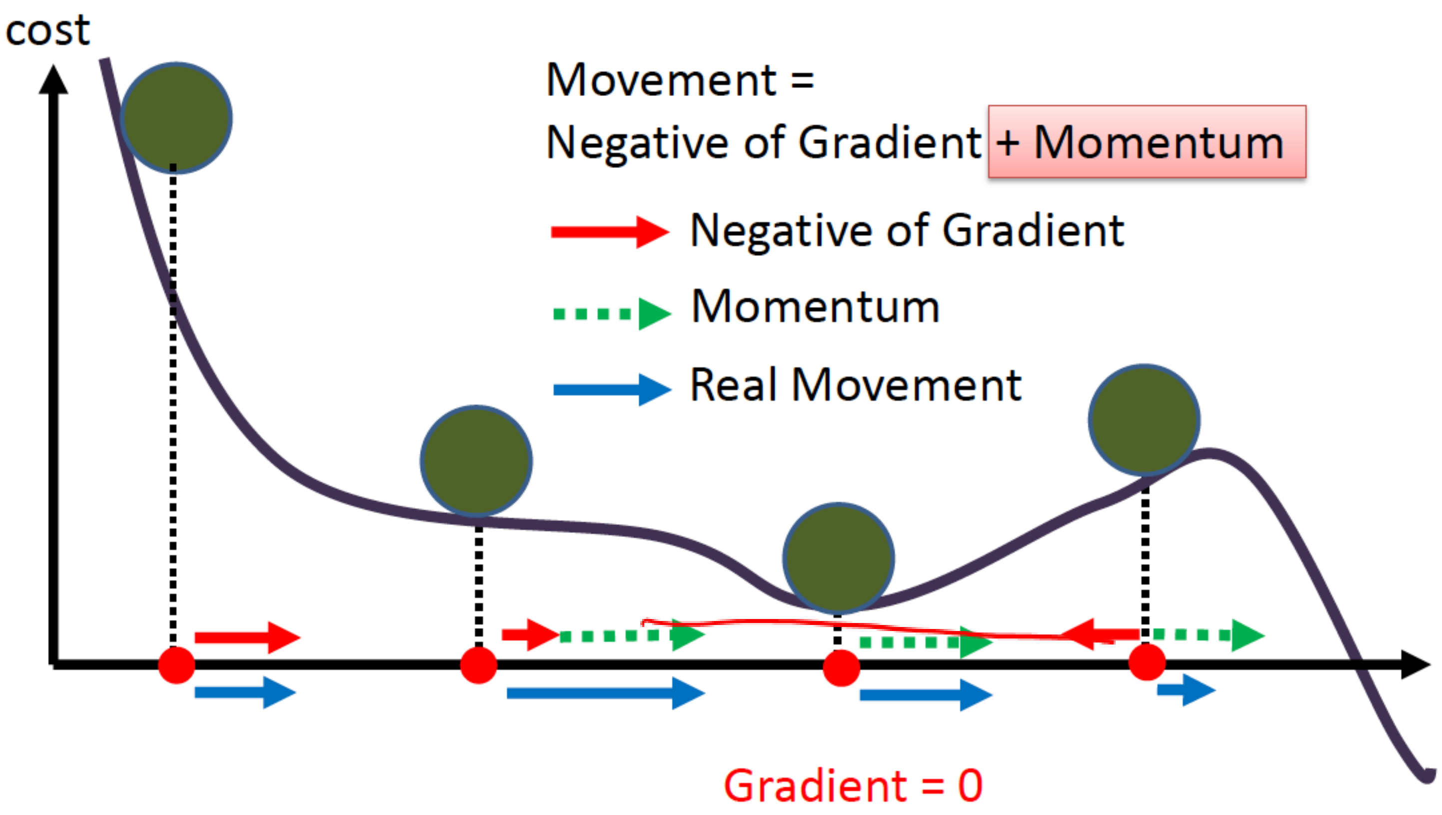
Optimization

- Method to minimize the cost function by updating weights
- Gradient descent
  - Iteratively moving in the direction of steepest descent as defined by the negative of the gradient
- Stochastic gradient descent (SGD)
  - To handle large training sets
  - Only run a subset of the training sets (i.e., batch/minibatch) for each update
  - Easier to converge



[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Optimization with Momentum



SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x -= learning_rate * dx
```

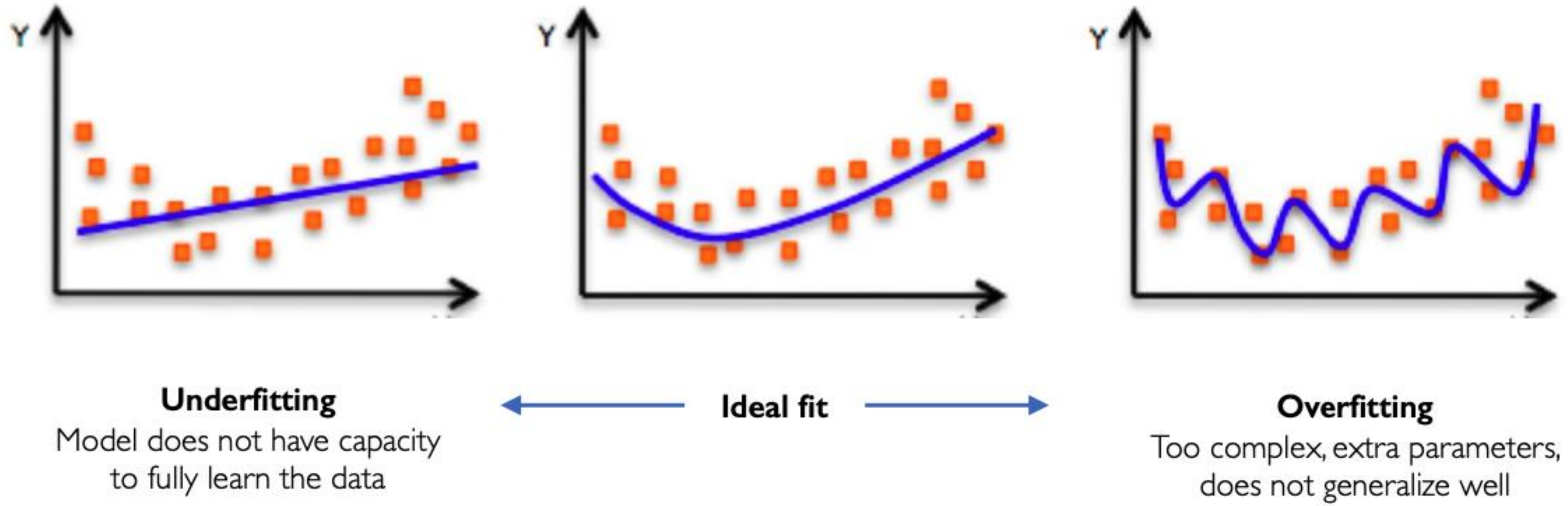
SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

The Problem of Overfitting



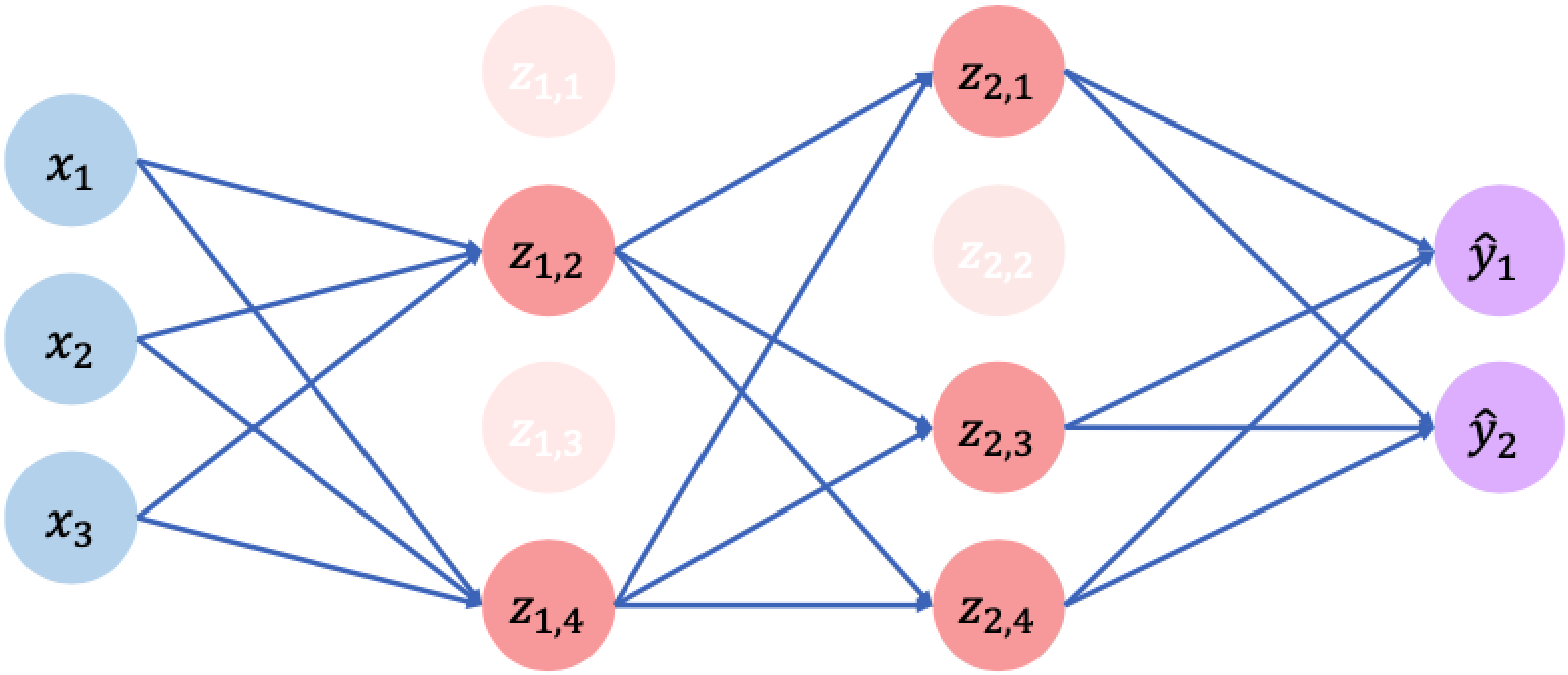
- Regularization:
  - Technique in the optimization loop to discourage complex models
  - Improve generalization of our model on unseen data

[Source: Prof. Alexander Amini, MIT 6.S191]

Regularization: Dropout

- During training, randomly set some activations to 0
  - Typically 'drop' 50% of activations in layer
  - Forces network to not rely on any 1 node

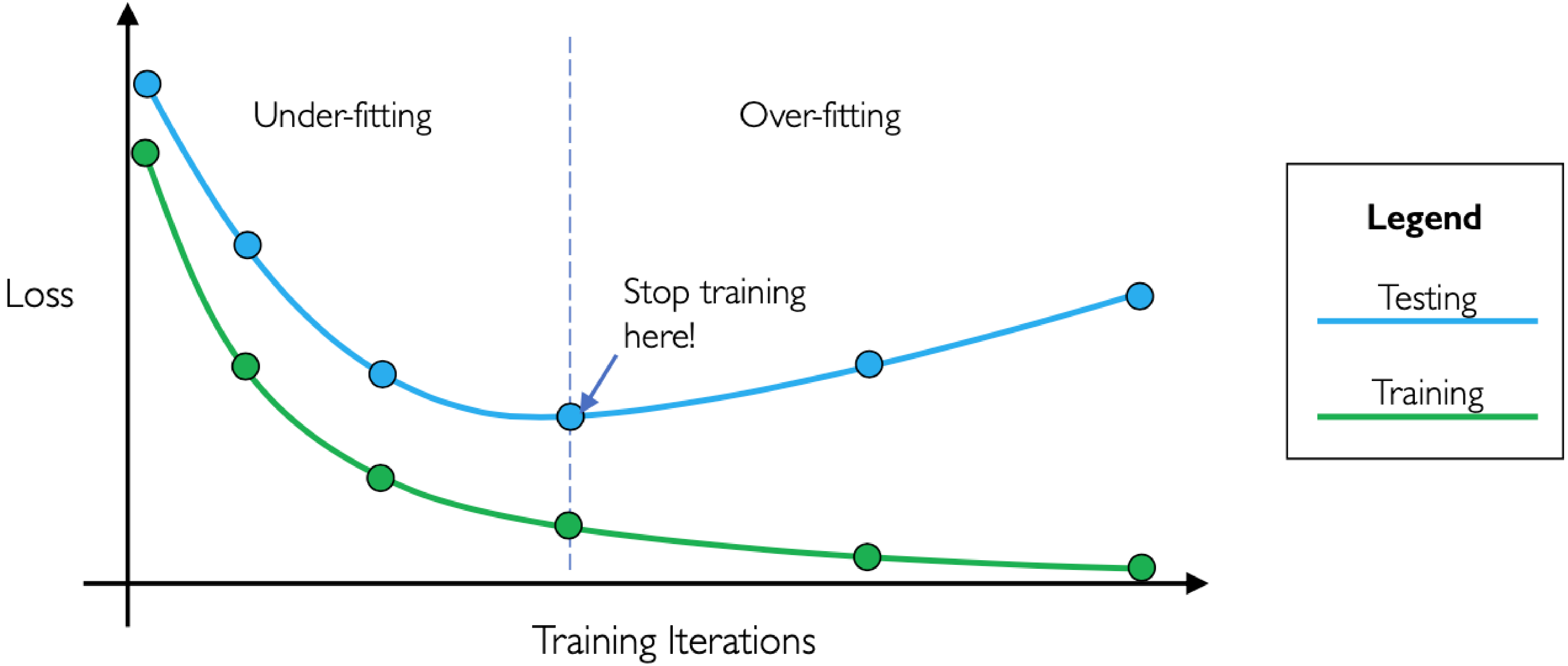
(Also faster to train!)



[Source: Prof. Alexander Amini, MIT 6.S191]

Regularization: Early Stopping

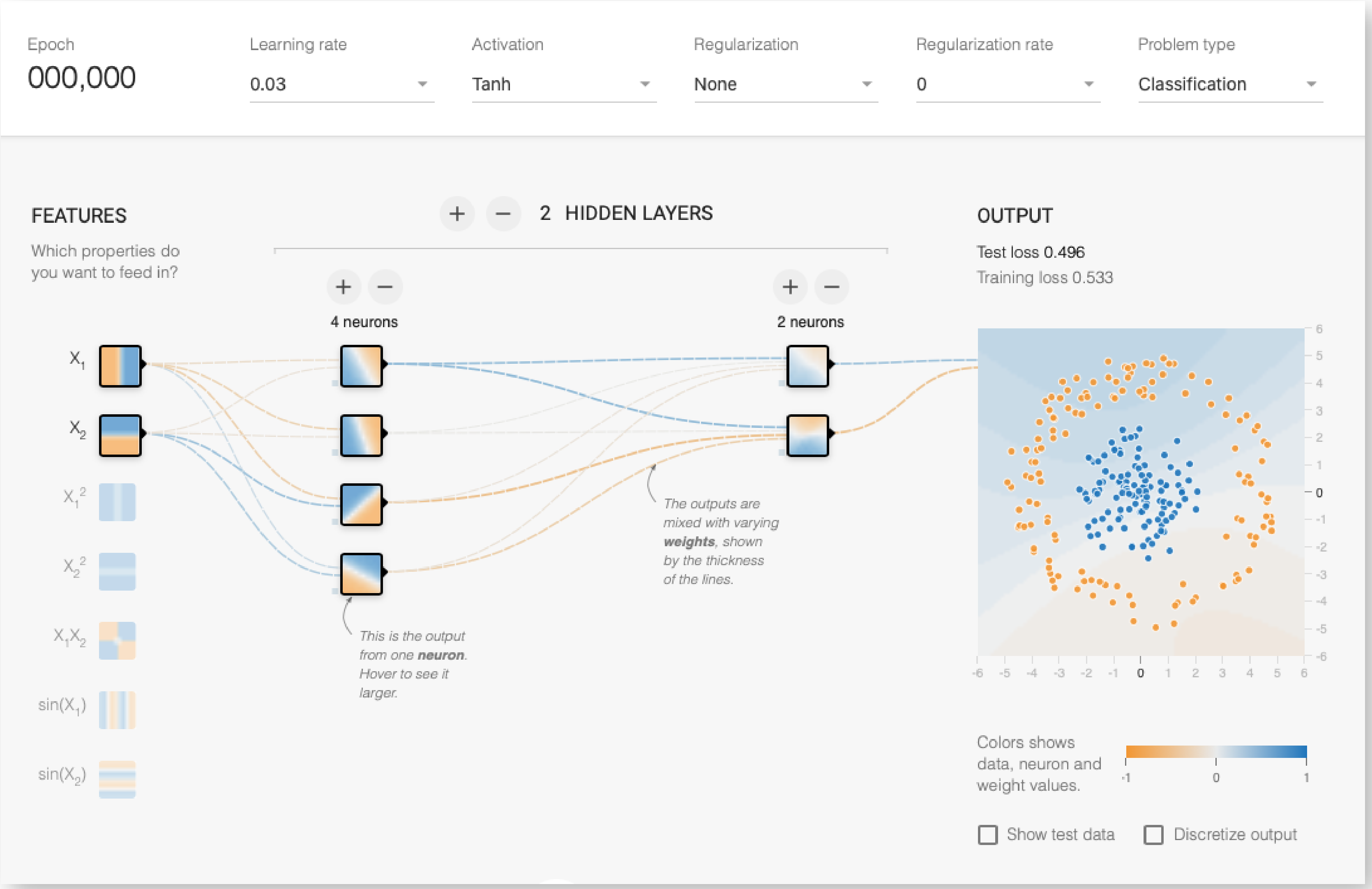
- Stop training before we have a chance to overfit



[Source: Prof. Alexander Amini, MIT 6.S191]

Tinker with a Neural Network

- http://playground.tensorflow.org



Deep Learning Frameworks

Caffe

Caffe2

TensorFlow

Microsoft Cognitive Toolkit

Microsoft

theano

PYTORCH

mxnet

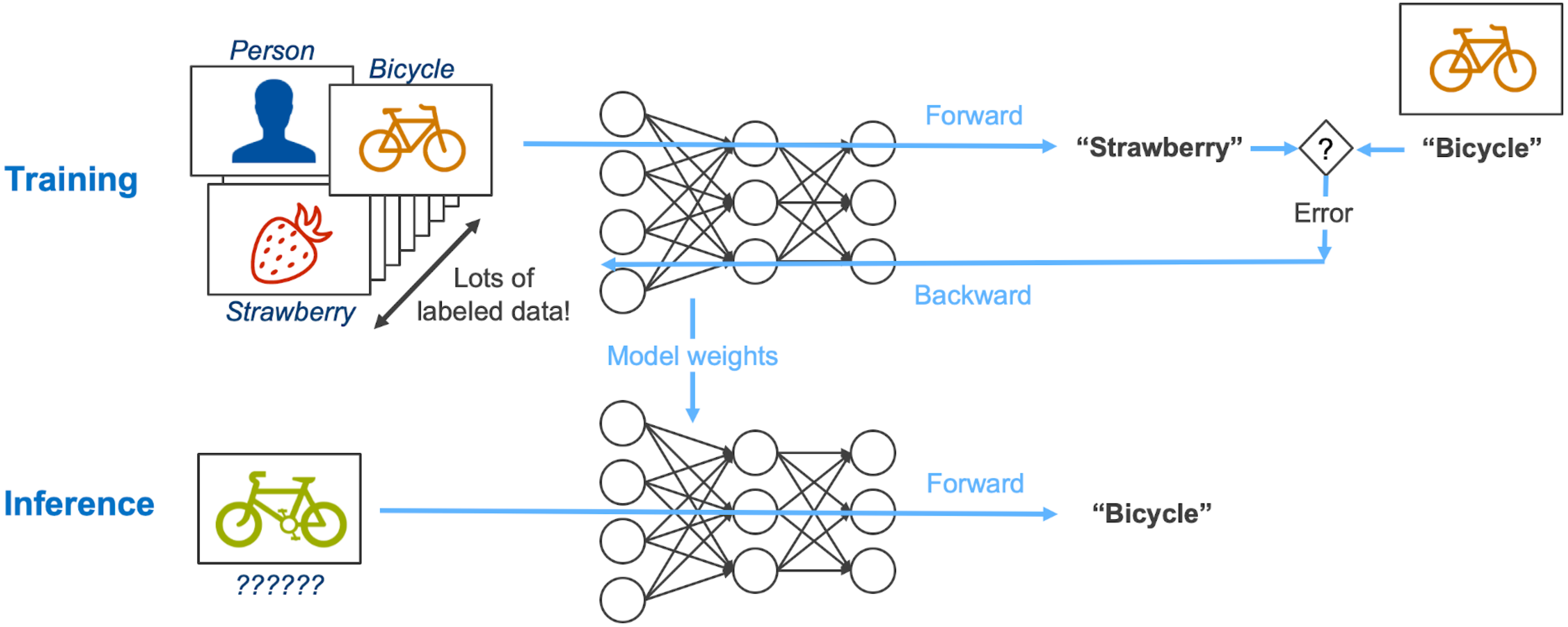
PaddlePaddle

Chainer

Ref: https://developer.nvidia.com/deep-learning-frameworks

Training vs. Inference

- Training
  - Dataset
  - Cost function
  - Optimization function
  - Model
- Inference
  - Dataset or realistic data
  - Model



Src: https://www.intel.com/content/www/us/en/artificial-intelligence/posts/deep-learning-training-and-inference.html