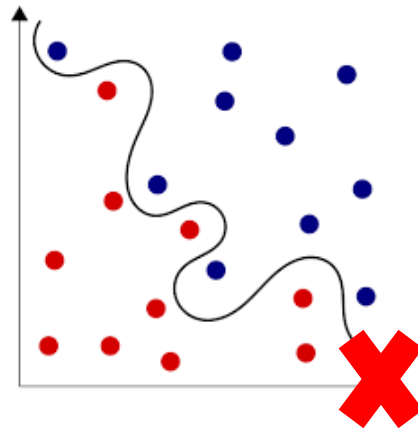# Basics of Deep Learning

Single Layer Perceptron
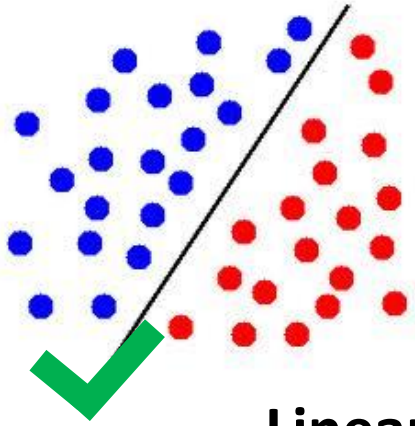
Linear Separable

Non-Linear Separable

Binary Class Perceptron

$$f\left(\sum_{i=1}^{d} X_i W_i + W_{bias}\right)$$

$f(x)$
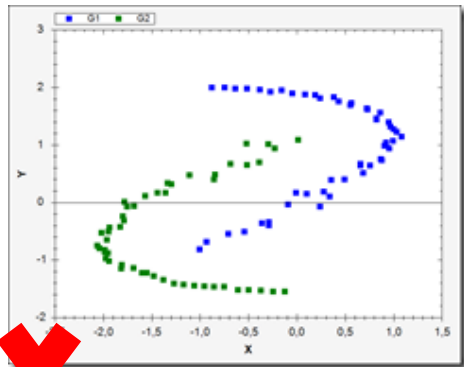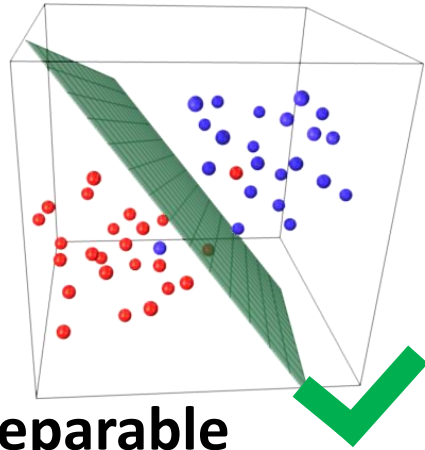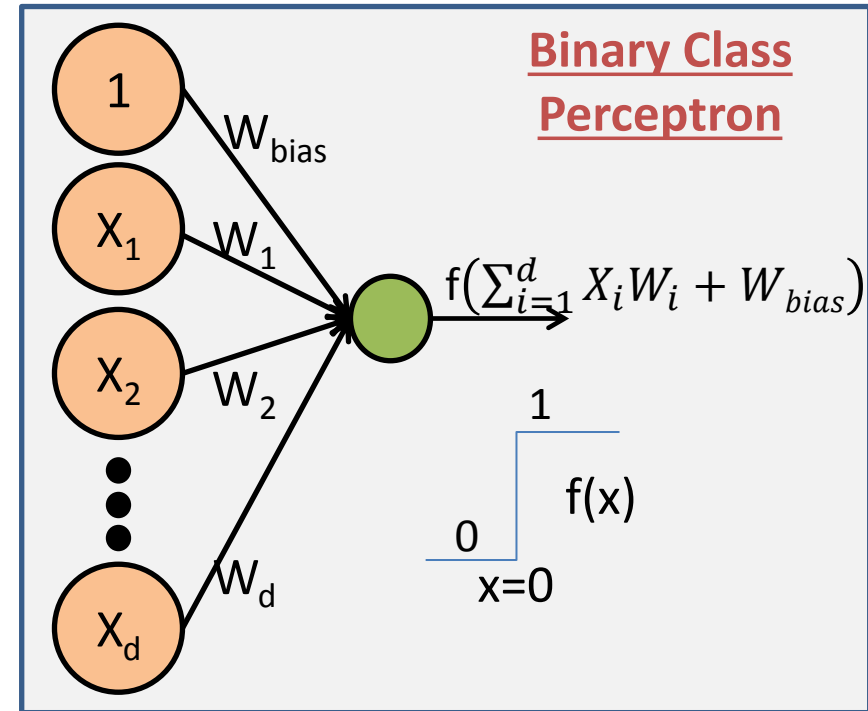
Single Layer Perceptron (Binary Class)

Learning Weights

$X(k) = [\ 1\ X^k_1\ X^k_2\ X^k_3\ ...\ X^k_d\ ]$

$W = [\ W_{bias}\ W_1\ W_2\ W_3\ ...\ W_d\ ]$

$Y(k) = 1$ if $\left(\sum_{i=1}^{d} X_i W_i + W_{bias} \right) > 0$

    $= 0$, otherwise

Or

$Y(k) = 1$ if $X(k)*W(k)^T > 0$

    $= 0$, otherwise

Intuitively,

$W(k+1) = W(k) + X(K)$ if $(X(k)*W(k)^T < 0)$ and $Y(k) = 1$

      $= W(k) - X(k)$ if $(X(k)*W(k)^T > 0)$ and $Y(k) = 0$

Single Layer Perceptron (Learning Weights)

Multi-Class Perceptron

$f\left(\sum_{i=1}^{d} X_i \textcolor{red}{W^1_i} + W^1_{bias}\right)$

$f\left(\sum_{i=1}^{d} X_i \textcolor{blue}{W^2_i} + W^2_{bias}\right)$

$f\left(\sum_{i=1}^{d} X_i \textcolor{orange}{W^3_i} + W^3_{bias}\right)$

1

0

f(x)

x=0
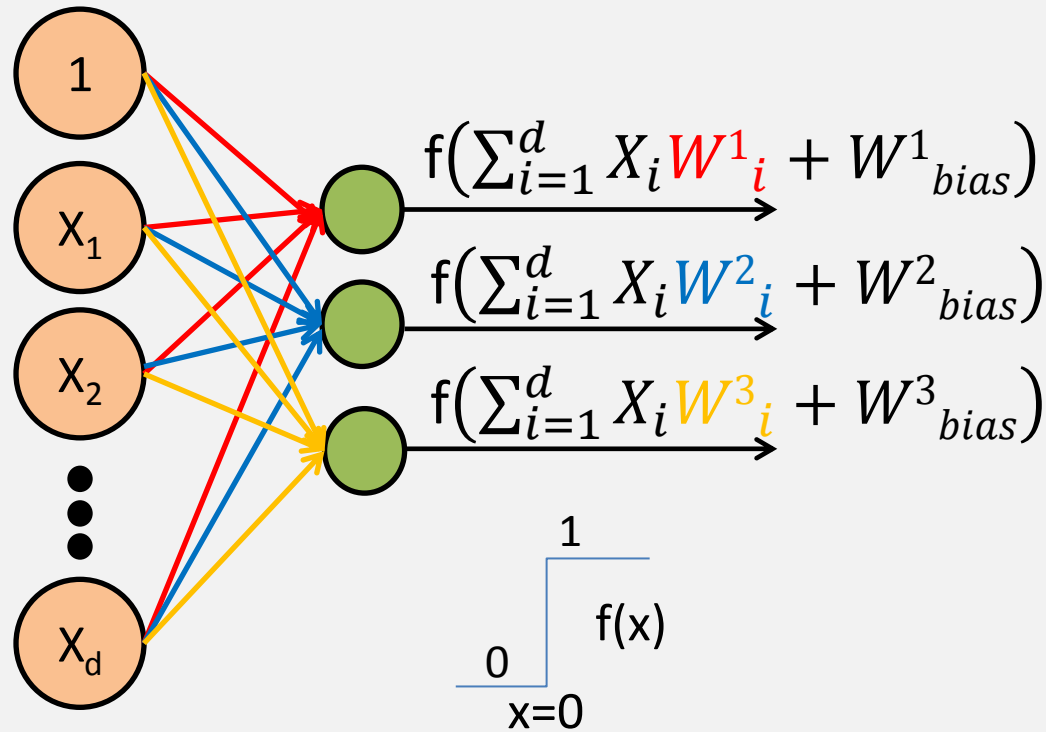
Single Layer Perceptron (Multi Class)

Linear Least Square Method

$$J(w) = \frac{1}{2} \sum_{i=1}^{n} (x_i^T w - y_i)^2$$

$$x_i = \begin{bmatrix} 1 \\ x_1^i \\ x_2^i \\ \vdots \\ x_d^i \end{bmatrix}_{(d+1)\times 1} \qquad w = \begin{bmatrix} w_{bias} \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}_{(d+1)\times 1}$$

Let $A = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{bmatrix}$

$$n \times (d+1)$$

$$\therefore J(w) = \frac{1}{2} (Aw - y)^T (Aw - y)$$

$$\nabla J(w) = A^T (Aw - y) \Big|_{w = w^*} = 0$$

$$\Rightarrow A^T A w^* = A^T y$$

$$\Rightarrow w^* = (A^T A)^{-1} A^T y$$

$$J(w) = \frac{1}{2} (Aw - y)^T (Aw - y) + \frac{\lambda}{2} w^T w$$

$$\nabla J(w) = A^T (Aw - y) + \lambda w \Big|_{w = w^*} = 0$$

$$\Rightarrow w^* = (A^T A + \lambda I)^{-1} A^T y$$

Gradient Descent

Cost Function, $J(W) = \frac{1}{2} \sum_{K=1}^{n} (X^k * W(k)^T - y^k)^2$

Gradient, $\nabla J(W) = \sum_{k=1}^{n} X^k (X^k W(k)^T - y^k)$
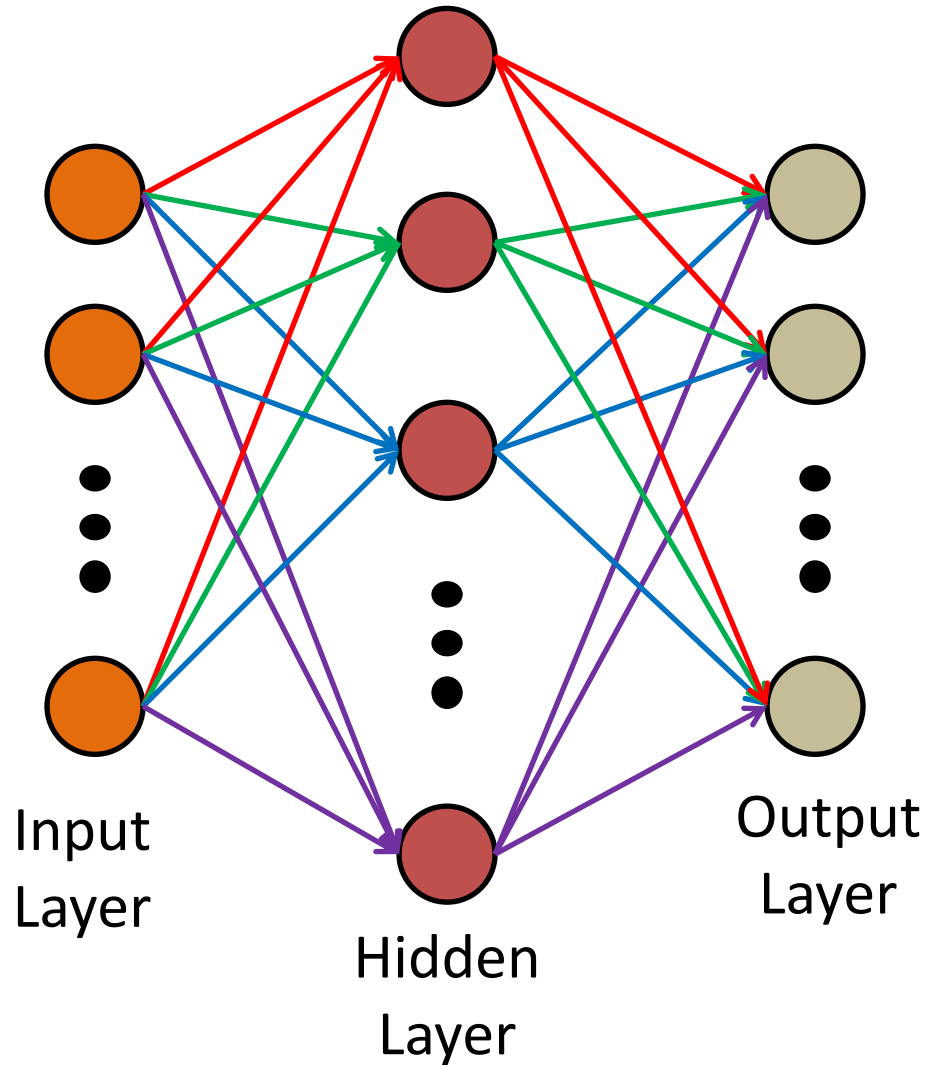
$W(k+1) = W(k) - \eta \, \nabla J(W)$

Batch Learning:
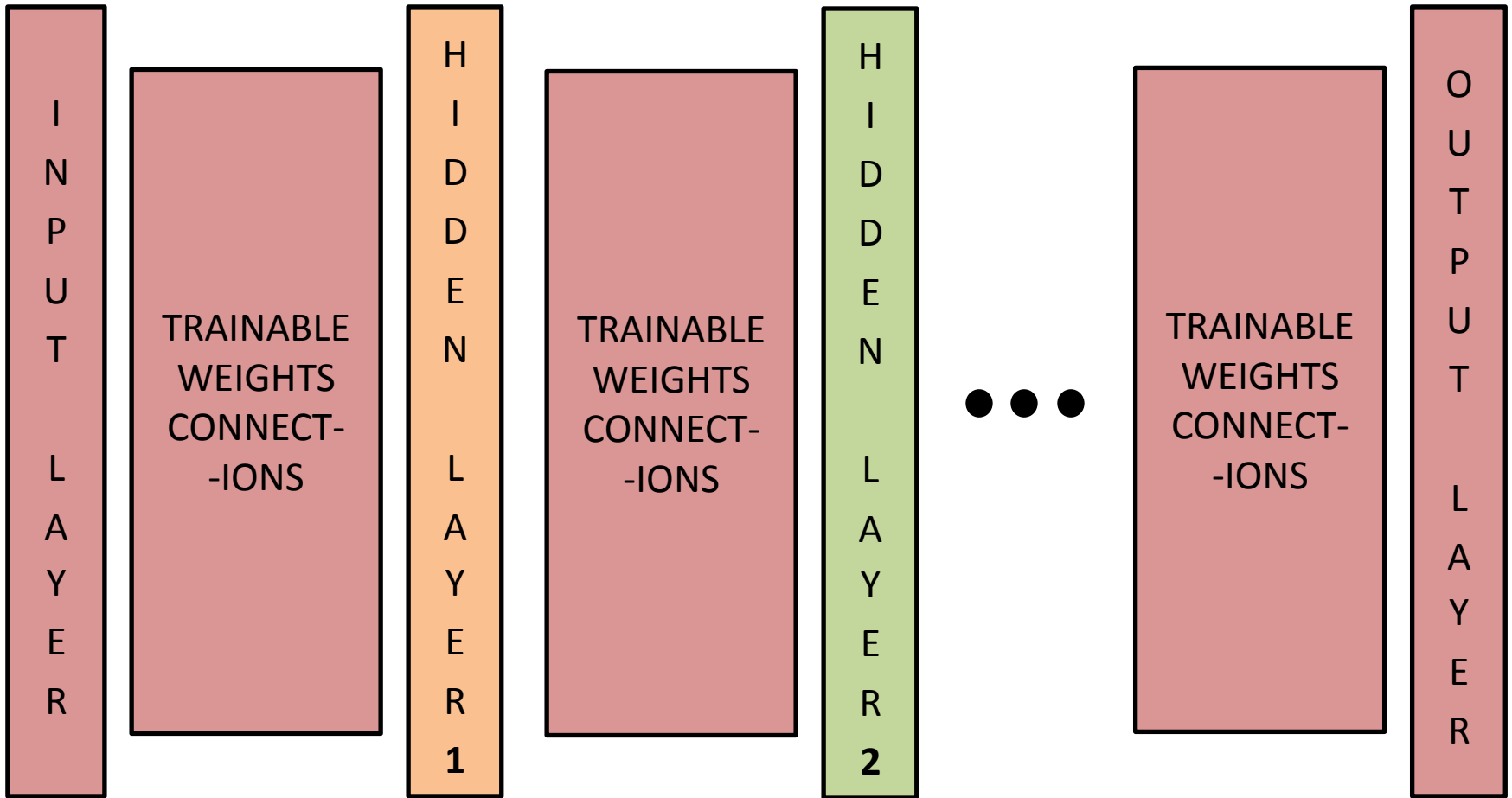$W(k+1) = W(k) - \eta \sum_{k=1}^{n} X^k (X^k W(k)^T - y^k)$

Stochastic/Online/Incremental Learning:
$W(k+1) = W(k) - \eta \sum_{k=1}^{n} X^k (X^k W(k)^T - y^k)$

Gradient Descent

Single Layer NN

Input
Layer

Hidden
Layer

Output
Layer

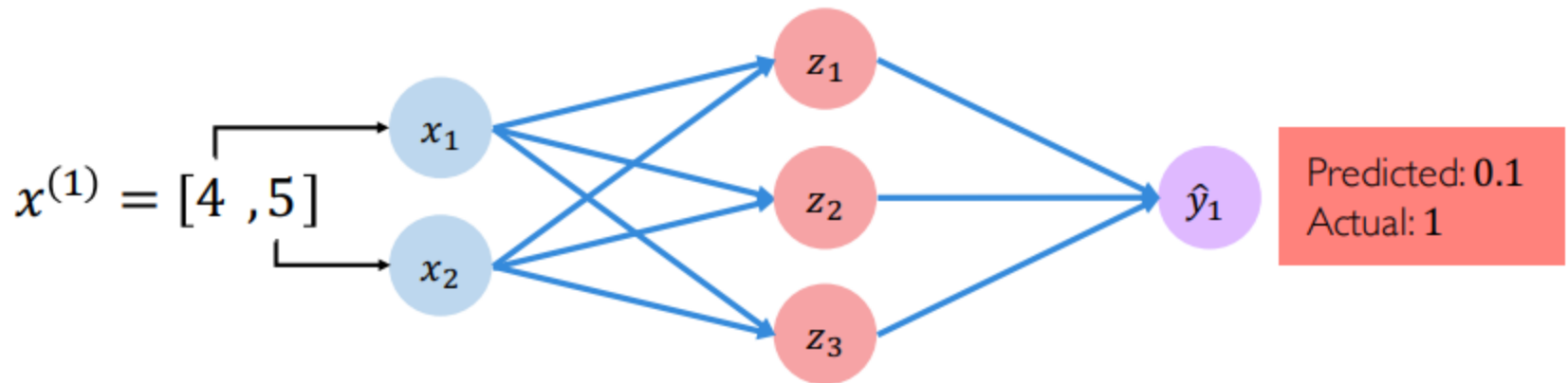Neural Networks (Single Layer)

Deep Neural Network
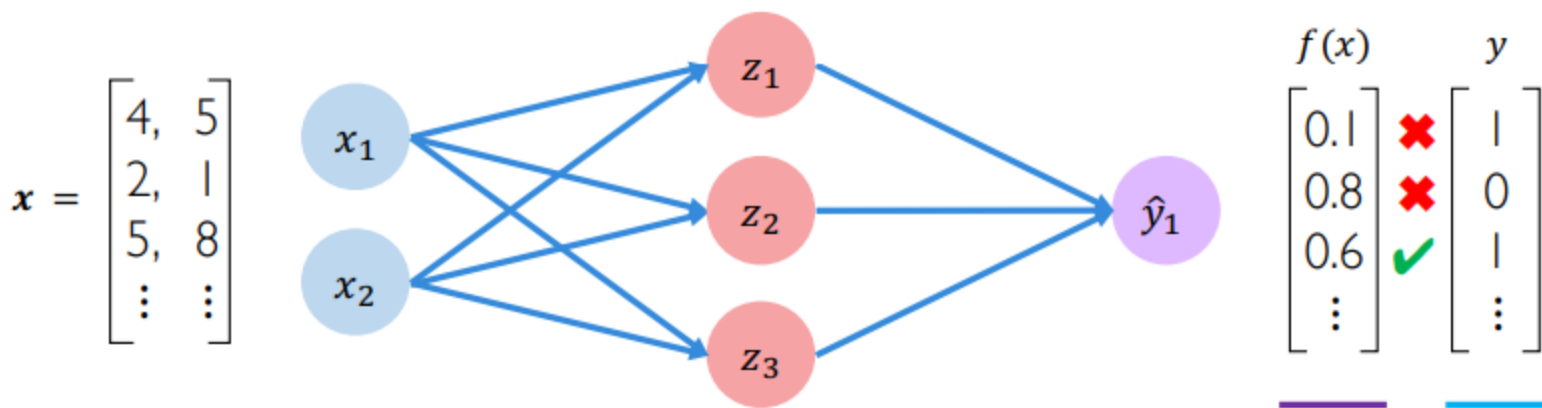

Neural Networks (DNN)

Quantifying Loss

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

$$\mathcal{L}\left(\underline{f\left(x^{(i)}; \boldsymbol{\theta}\right)}, \underline{y^{(i)}}\right)$$

Predicted   Actual

http://introtodeeplearning.com/index.html

Empirical Loss

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

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

$f(x)$ y

$$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \begin{matrix} ✖ \\ ✖ \\ ✔ \end{matrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

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

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \theta\right), y^{(i)} \right)$$

Predicted   Actual

http://introtodeeplearning.com/index.html

Loss Functions

Binary Cross Entropy Loss:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log\left(\underbrace{f(x^{(i)};\theta)}_{\text{Predicted}}\right) + \underbrace{(1-y^{(i)})}_{\text{Actual}} \log\left(\underbrace{1-f(x^{(i)};\theta)}_{\text{Predicted}}\right)$$
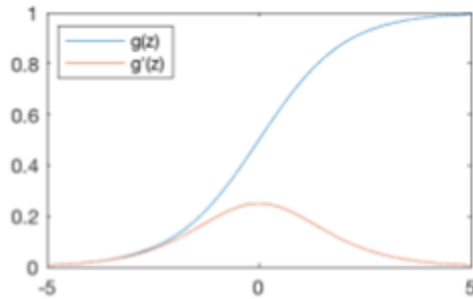
Mean Square Error Loss:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f(x^{(i)};\theta)}_{\text{Predicted}}\right)^2$$

http://introtodeeplearning.com/index.html

Common Activation Functions
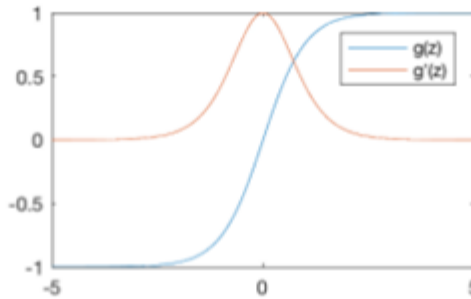
http://introtodeeplearning.com/index.html

Neural Networks (Activation Functions)
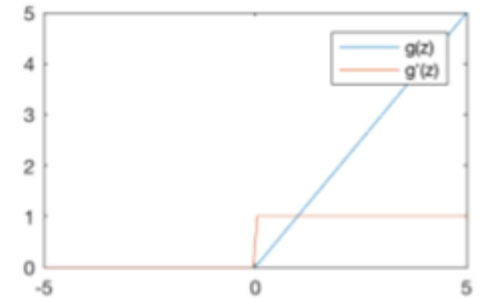
Forward Propagation
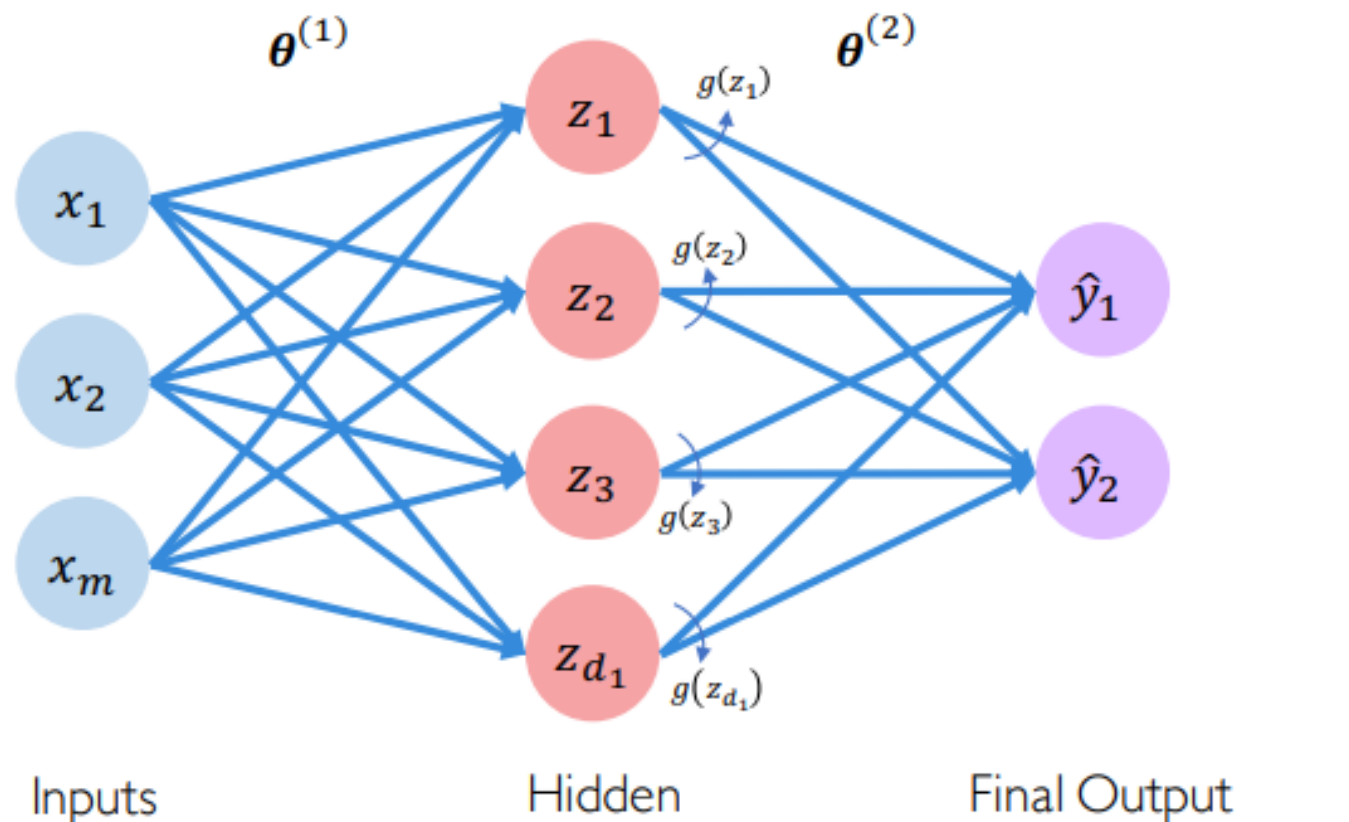
$$z_i = \theta_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, \theta_{j,i}^{(1)} \qquad \hat{y}_i = \theta_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, \theta_{j,i}^{(2)}$$

http://introtodeeplearning.com/index.html

Single Layer Neural Networks (Forward Propagation)
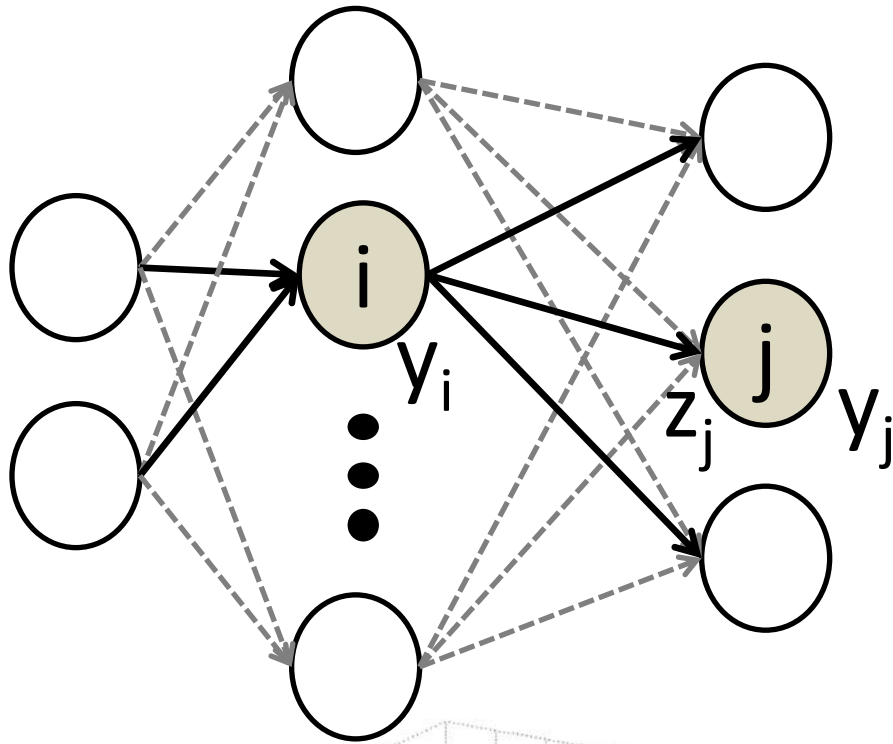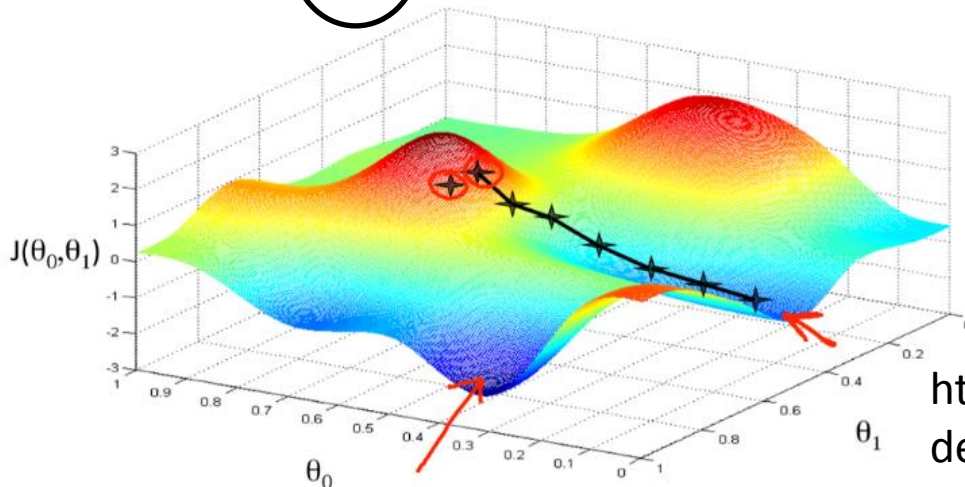
Back Propagation

$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial E}{\partial y_j} = y_j\,(1-y_j)\frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_i\frac{\partial E}{\partial z_j}$$

$$w_{ij}^l(t+1) = w_{ij}^l(t) - \lambda\frac{\partial E(W(t))}{\partial w_{ij}^l}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial z_j}$$

https://mc.ai/a-dummies-guide-to-gradient-descent-and-backpropagation/

Single Layer Neural Networks (Back Propagation)

Regularization

Overfitting Problem

Underfitting
Model does not have capacity
to fully learn the data

Ideal fit

Overfitting
Too complex, extra parameters,
does not generalize well

http://introtodeeplearning.com/index.html
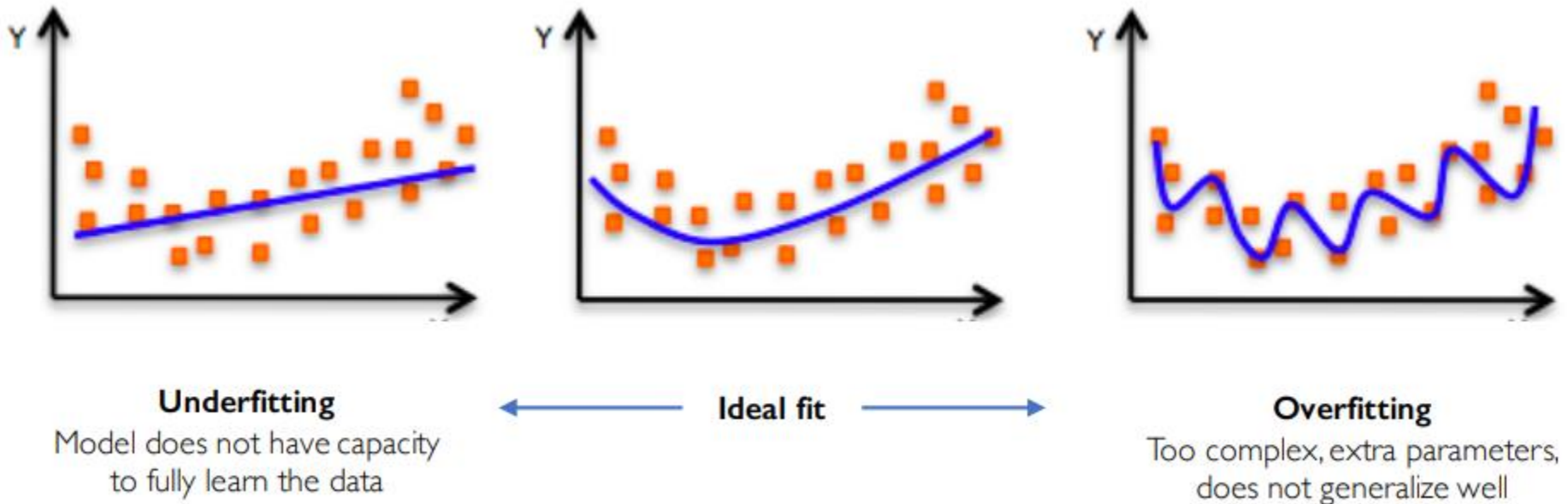
Solution ⟶ Regularization

Helps to generalize
our network

Neural Networks: Regularization

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
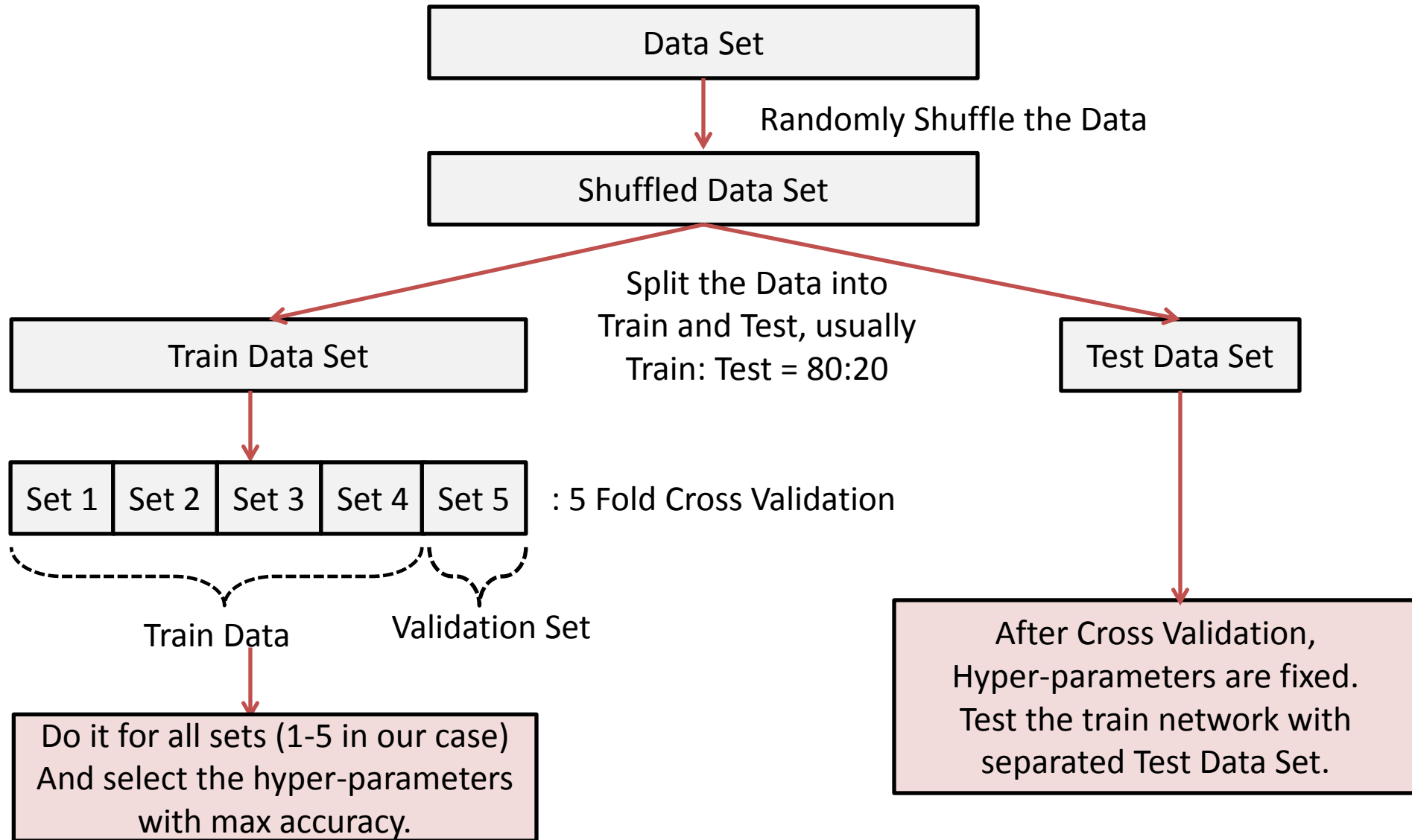

http://introtodeeplearning.com/index.html

Regularization: Dropout
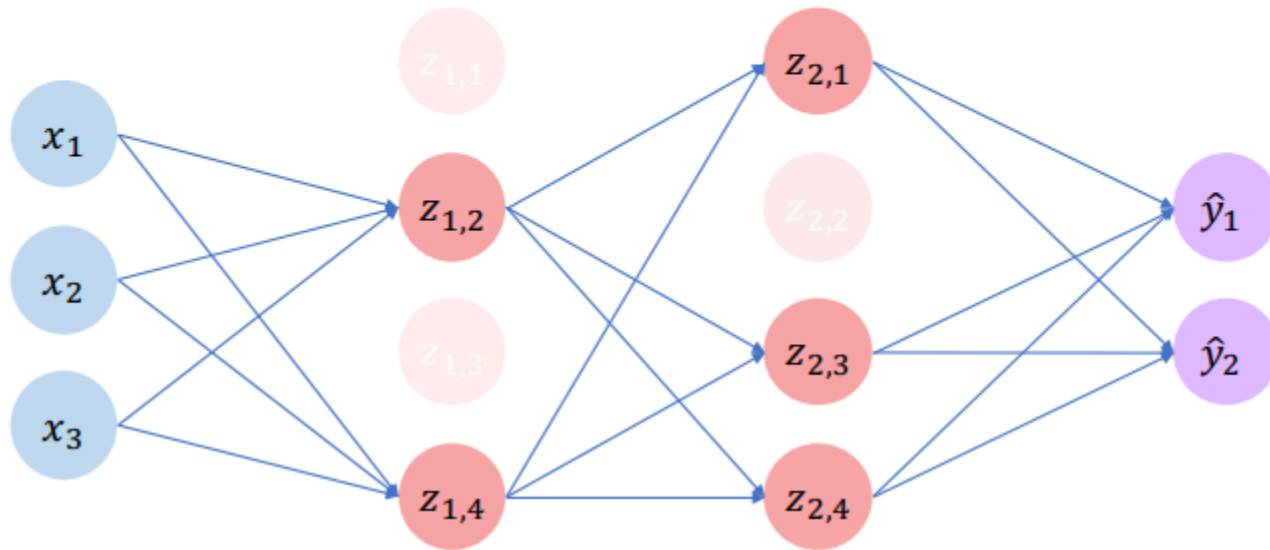
Regularization: Early Stopping

- Stop training before we have a chance to overfit

http://introtodeeplearning.com/index.html

Regularization: Early Stopping

Training and Validation

Data Set

Randomly Shuffle the Data

Shuffled Data Set

Split the Data into
Train and Test, usually
Train: Test = 80:20

Train Data Set

Test Data Set

Set 1 | Set 2 | Set 3 | Set 4 | Set 5 : 5 Fold Cross Validation

Train Data

Validation Set

Do it for all sets (1-5 in our case)
And select the hyper-parameters
with max accuracy.

After Cross Validation,
Hyper-parameters are fixed.
Test the train network with
separated Test Data Set.

Cross Validation