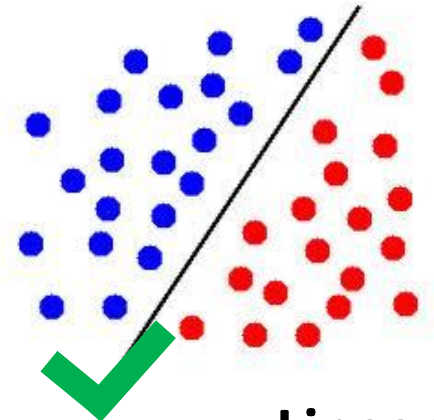
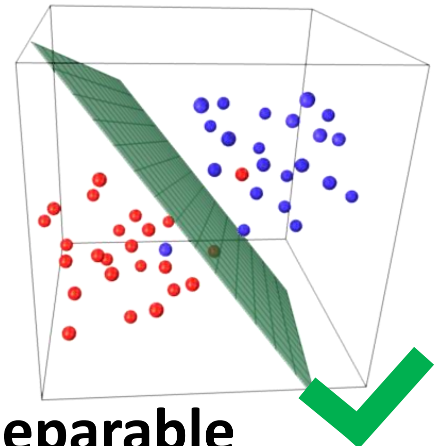
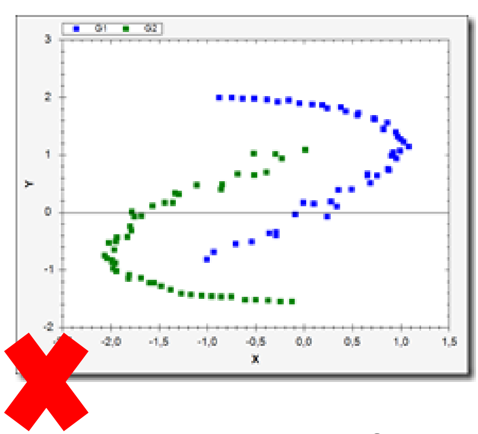
Basics of Deep Learning

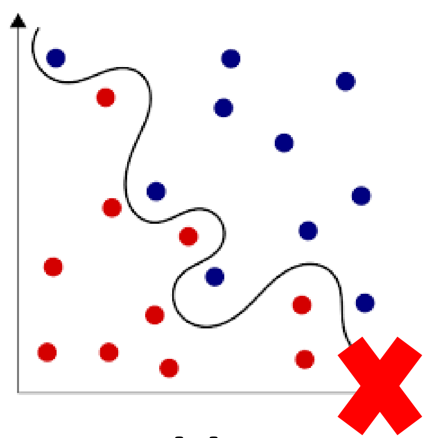
Single Layer Perceptron



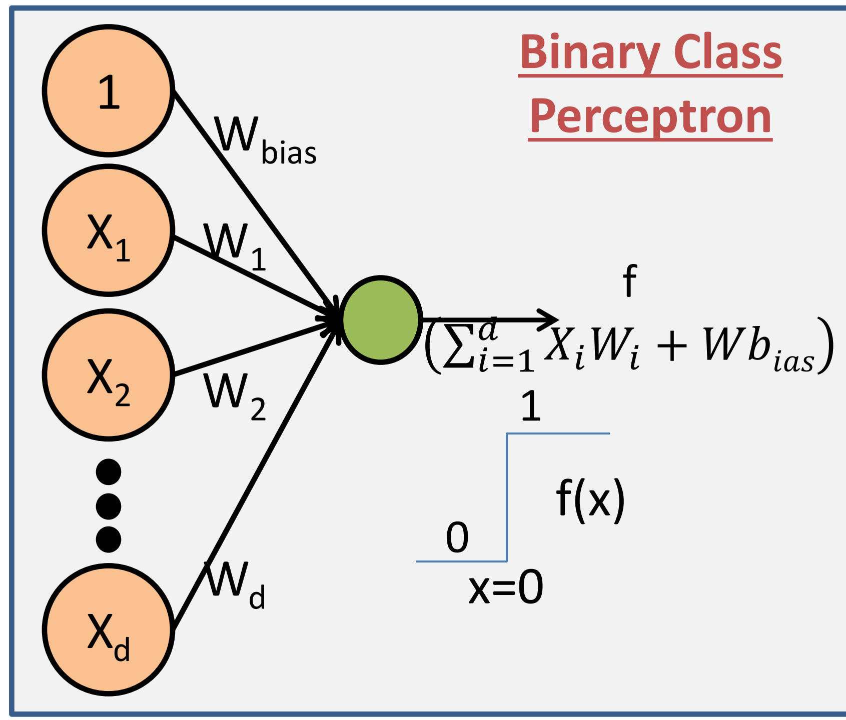


Linear Separable





Non-Linear Separable



Single Layer Perceptron (Binary Class)

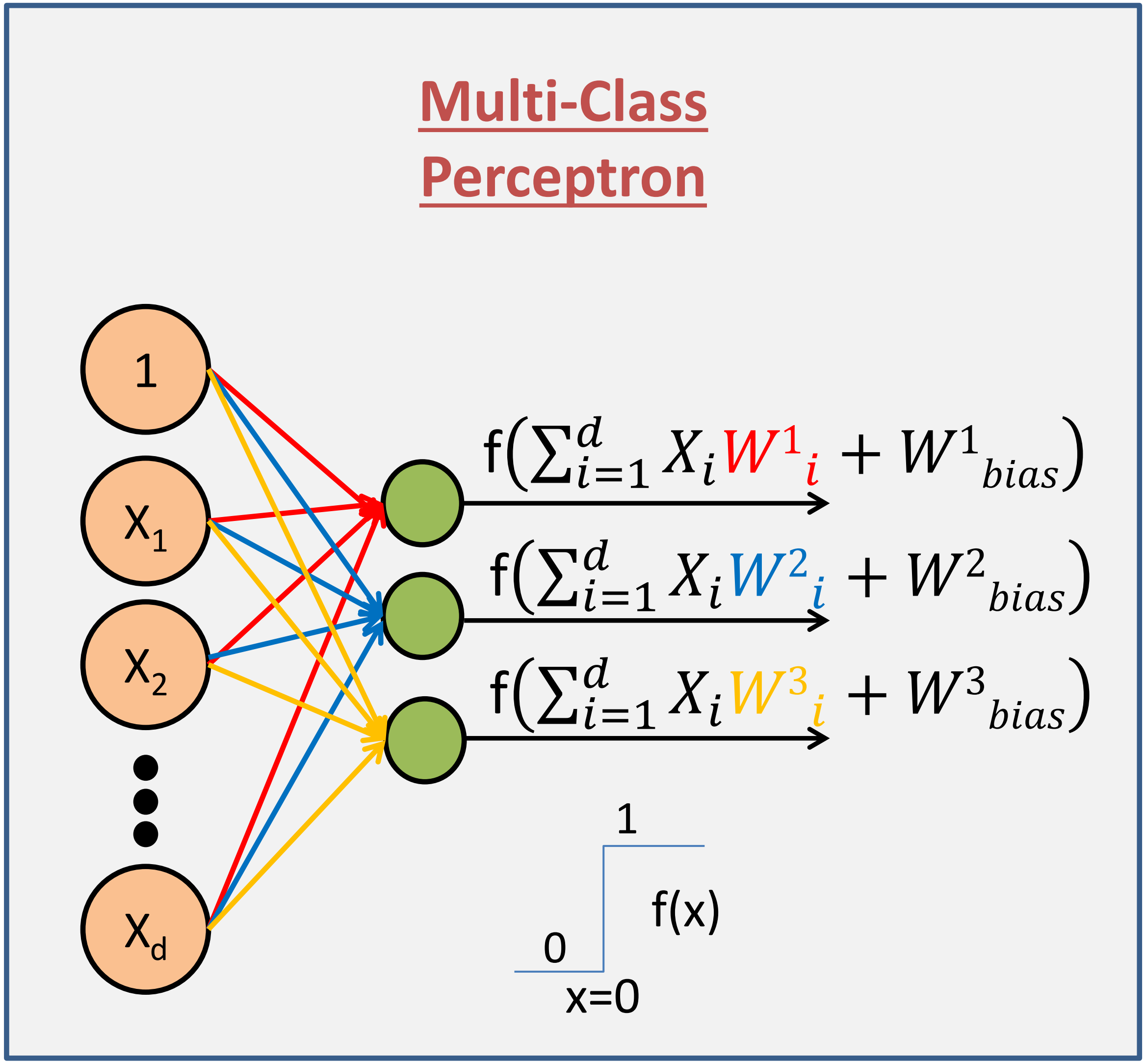
Learning Weights

$X(k) = [\ 1\ X^k{}_1\ X^k{}_2\ X^k{}_3\ \ldots\ X^k{}_d\ ]$
$W = [\ W_{bias}\ W_1\ W_2\ W_3\ \ldots\ W_d\ ]$

$Y(k) = 1$ if $\left(\sum_{i=1}^{d} X_i W_i + W b_{ias}\right) > 0$
$= 0$, 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)



Single Layer Perceptron (Multi Class)

Linear Least Square Method

$$J(w) = \frac{1}{2}\sum_{i=1}^{n} (x_i^T w - y_i)^2, \quad x_i = \begin{bmatrix}1\\ x_1^i\\ x_2^i\\ \vdots\\ x_d^i\end{bmatrix}_{(d+1)\times 1} \quad w = \begin{bmatrix}w_{bias}\\ w_1\\ w_2\\ \vdots\\ w_d\end{bmatrix}_{(d+1)\times 1}$$

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, ∇J(W) = $\sum_{k=1}^{n} X^k(X^k W(k)^T - y^k)$

W(k+1) = W(k) - η ∇J(W)

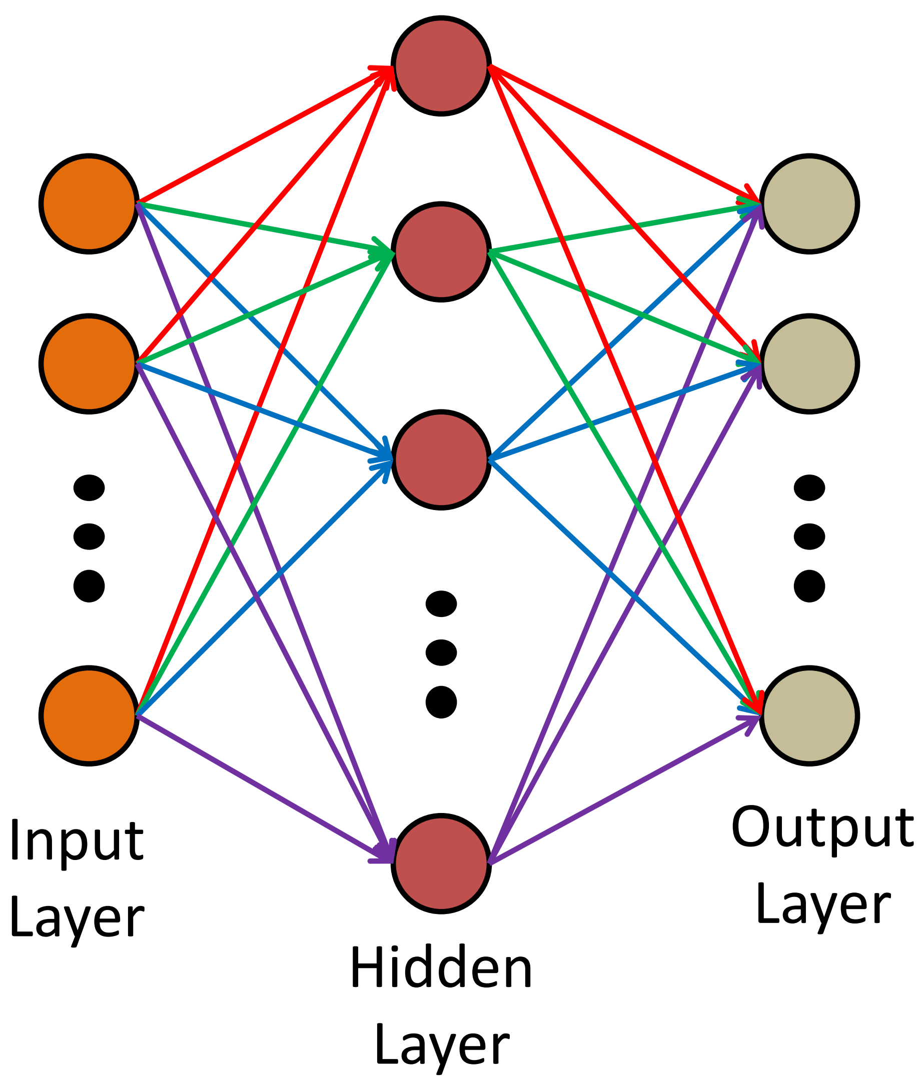
Batch Learning:

W(k+1) = W(k) - η $\sum_{k=1}^{n} X^k(X^k W(k)^T - y^k)$

Stochastic/Online/Incremental Learning:

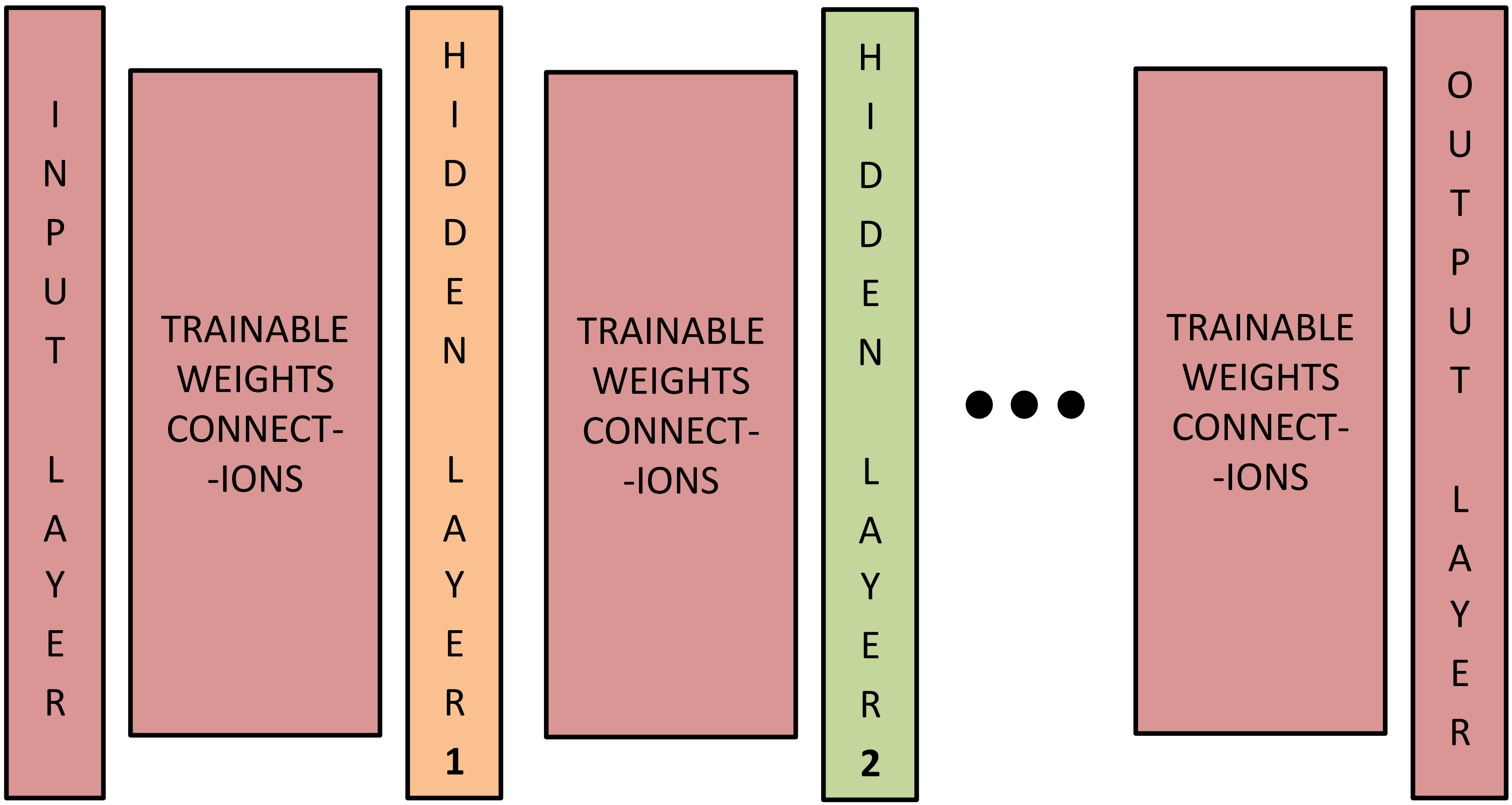
W(k+1) = W(k) - η $\sum_{k=1}^{n} X^k(X^k W(k)^T - y^k)$

Single Layer NN



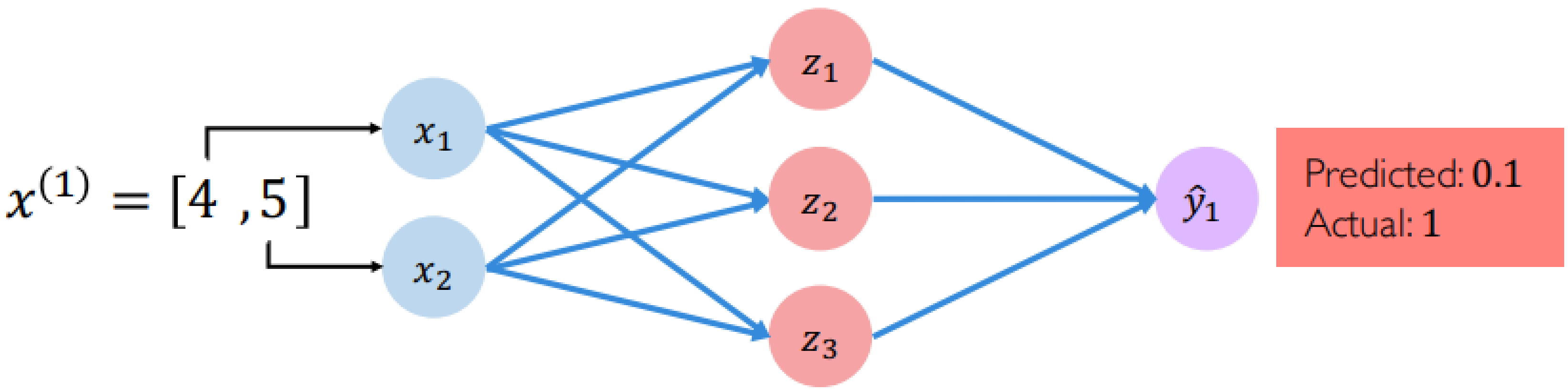
Neural Networks (Single Layer)

Deep Neural Network



Quantifying Loss

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

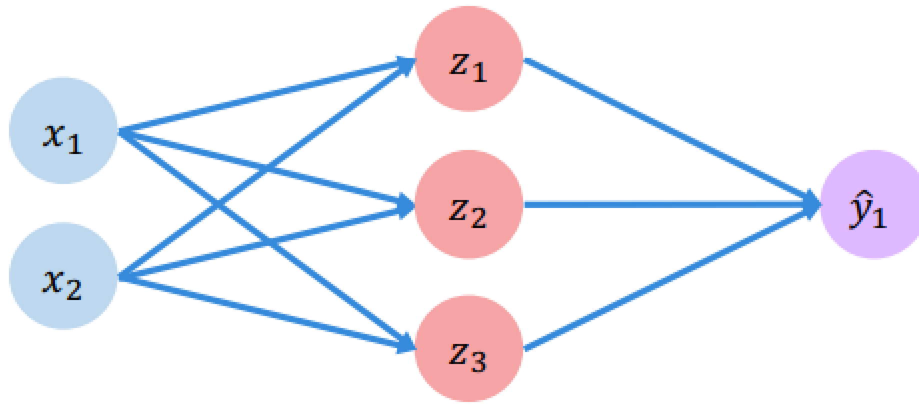


$$\mathcal{L}\Big(\underbrace{f\big(x^{(i)};\boldsymbol{\theta}\big)}_{\text{Predicted}},\ \underbrace{y^{(i)}}_{\text{Actual}}\Big)$$

http://introtodeeplearning.com/index.html

Empirical Loss

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



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

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

Also known as:

- Objective function
- Cost function
- Empirical Risk

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

http://introtodeeplearning.com/index.html

Loss Functions

Binary Cross Entropy Loss:

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

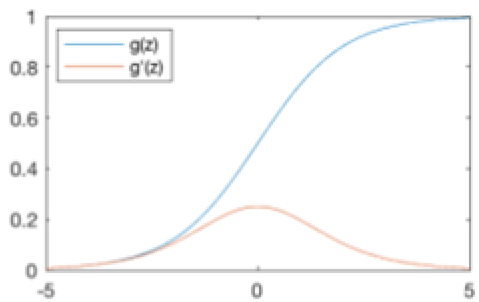
Mean Square Error Loss:

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

http://introtodeeplearning.com/index.html

Common Activation Functions

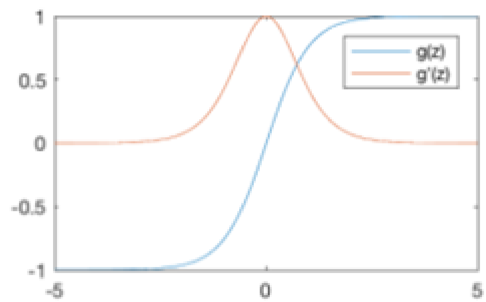
Sigmoid Function



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

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

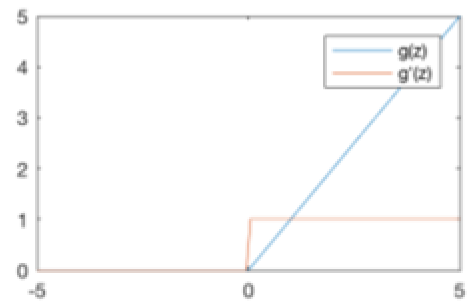
Hyperbolic Tangent



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

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

Rectified Linear Unit (ReLU)

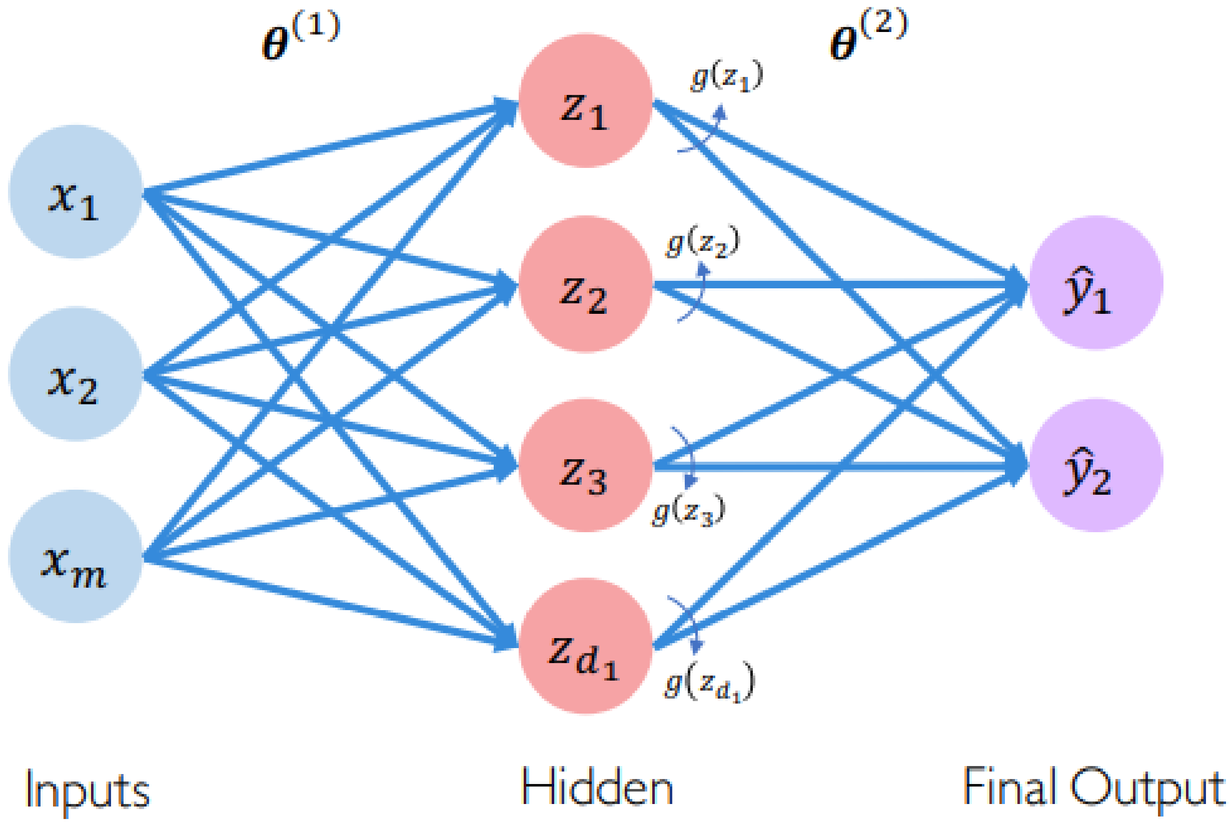


$$g(z) = \max(0, z)$$

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

http://introtodeeplearning.com/index.html

Forward Propagation

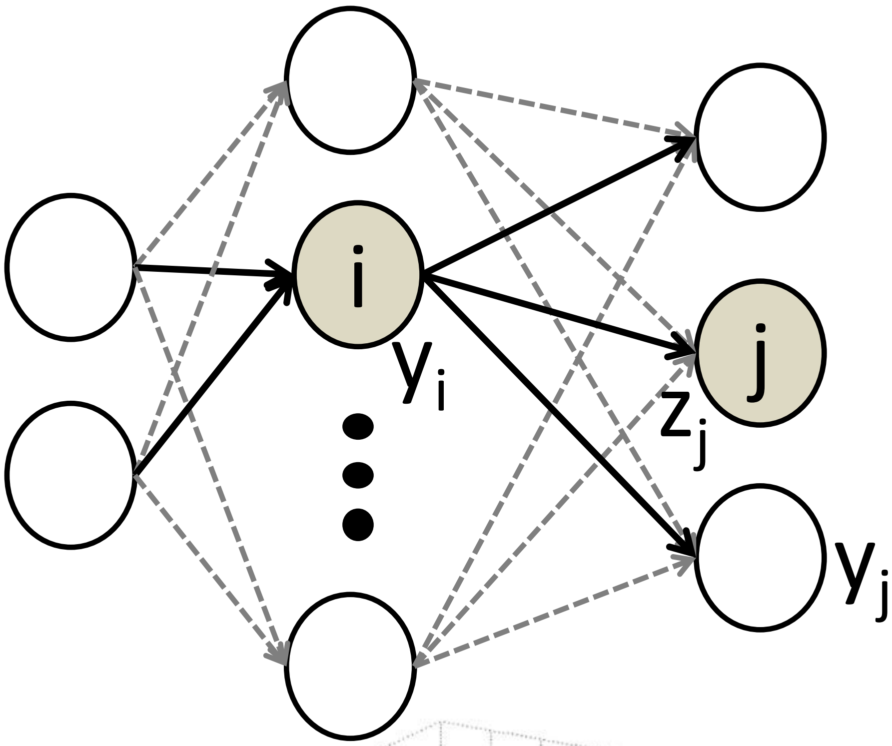


$$z_i = \theta_{0,i}^{(1)} + \sum_{j=1}^{m} x_j\, \theta_{j,i}^{(1)} \qquad \hat{y}_i = \theta_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j\, \theta_{j,i}^{(2)}$$

http://introtodeeplearning.com/index.html

Single Layer Neural Networks (Forward Propagation)

Back Propagation

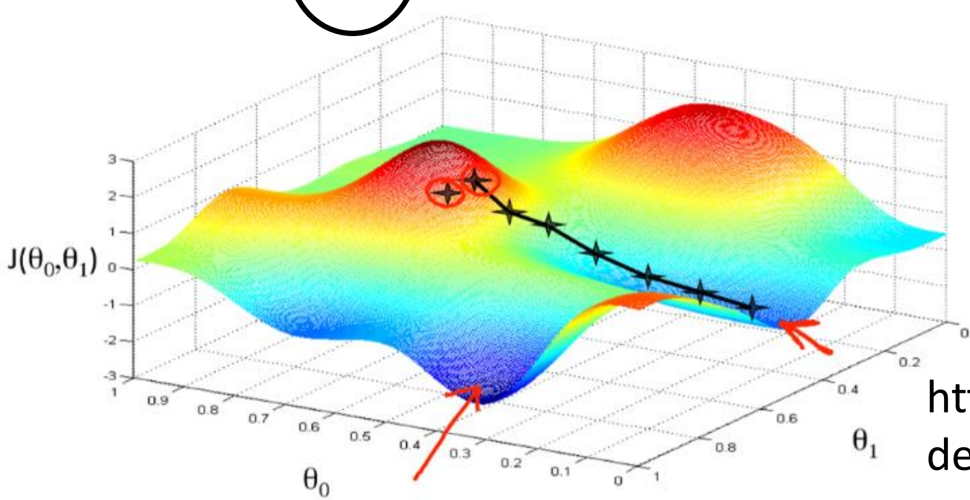


$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial E}{\partial y_j} = y_j\,(1-y_j)\,\frac{\partial E}{\partial y_j}$$

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

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

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

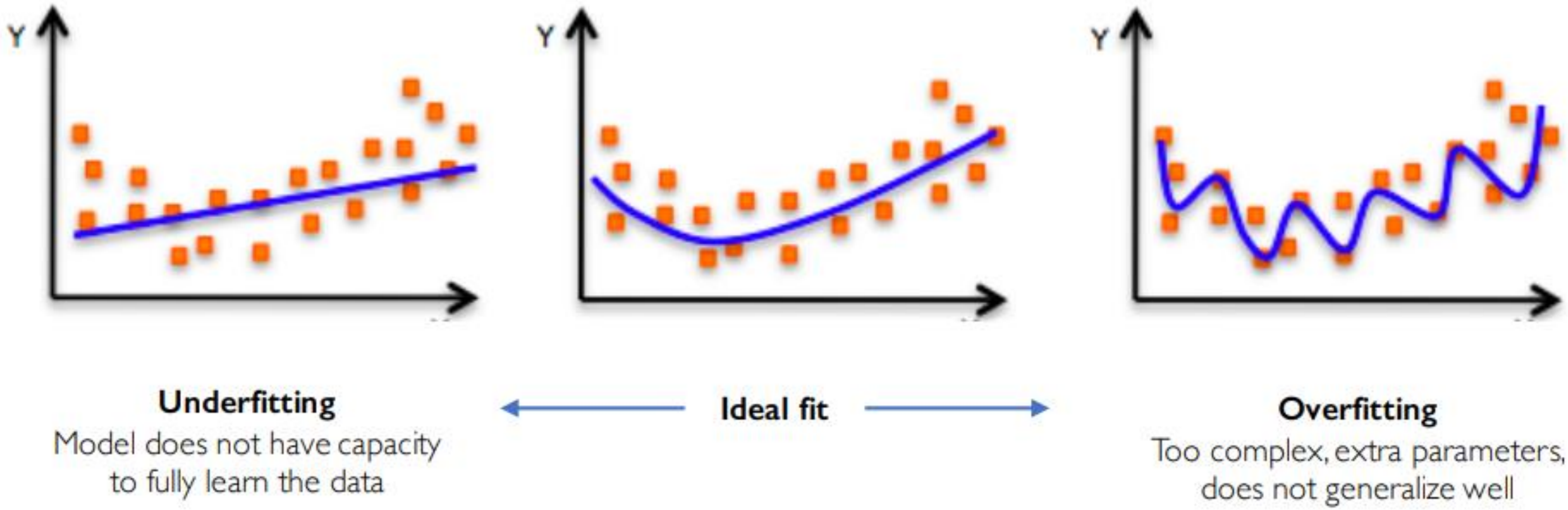


https://mc.ai/a-dummies-guide-to-gradient-descent-and-backpropagation/

Single Layer Neural Networks (Back Propagation)

Regularization

Overfitting Problem



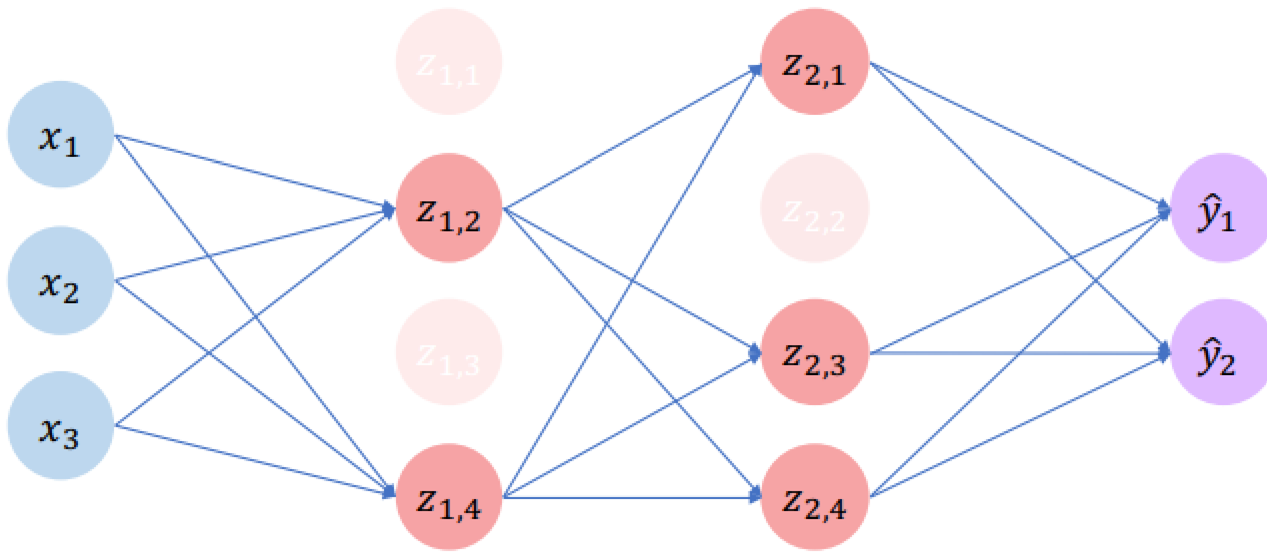
http://introtodeeplearning.com/index.html

Solution → Regularization

Helps to generalize our network

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

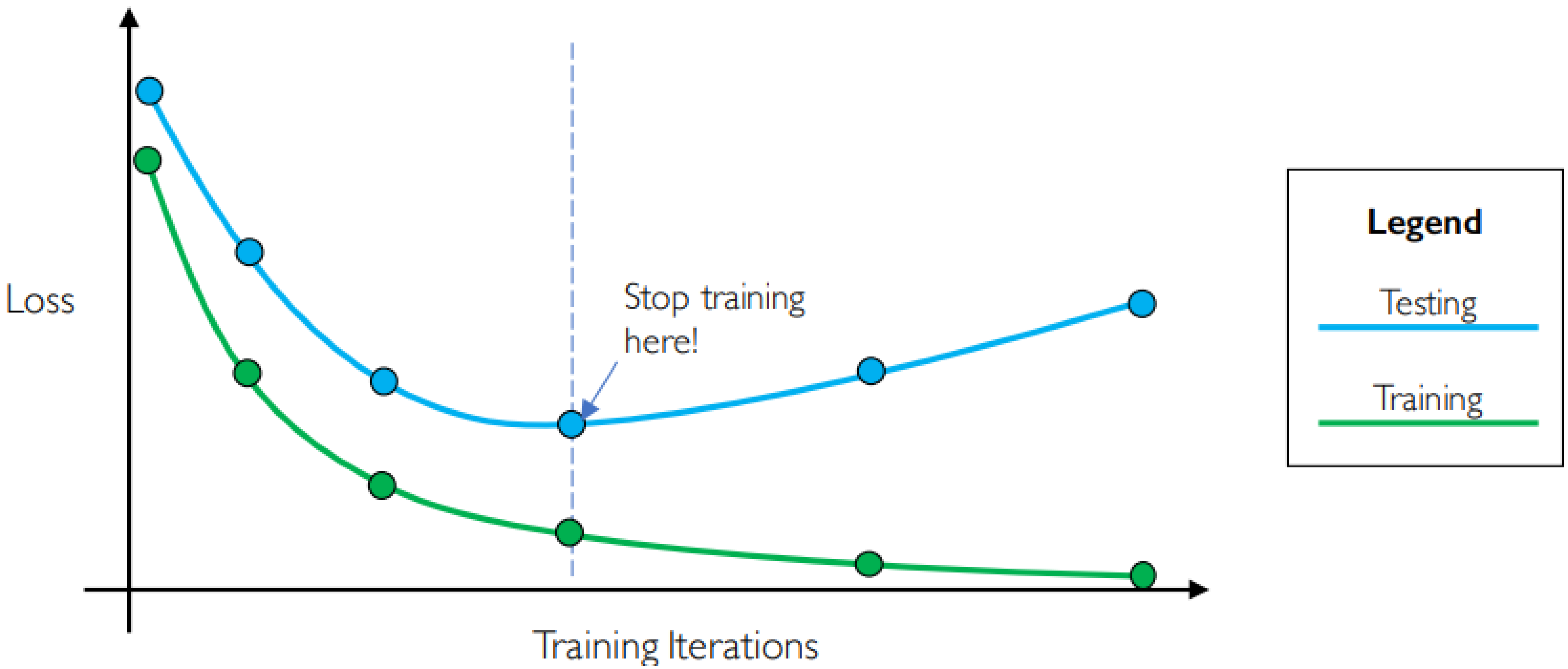


http://introtodeeplearning.com/index.html

Regularization: Dropout

Regularization: Early Stopping

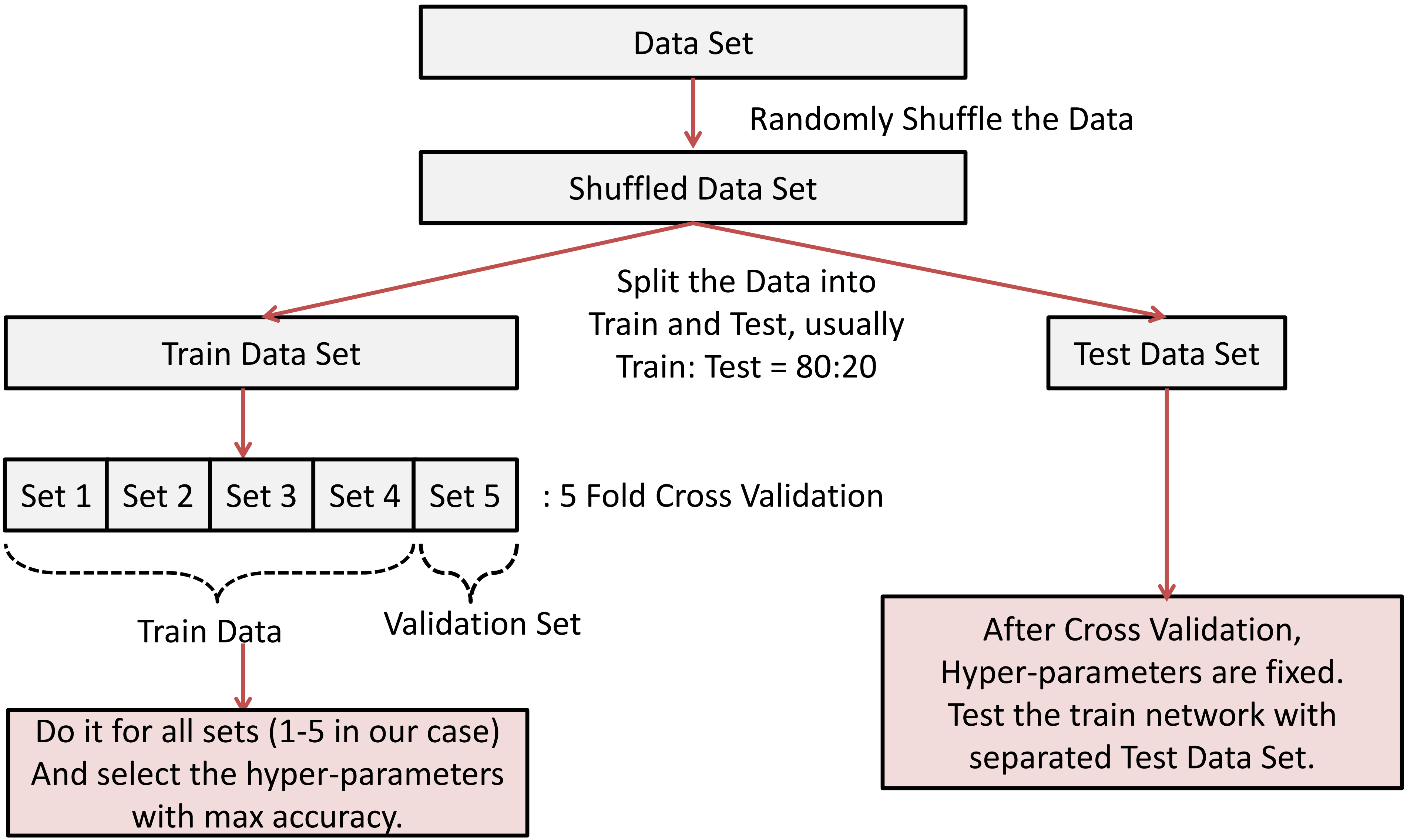
- Stop training before we have a chance to overfit



http://introtodeeplearning.com/index.html

Regularization: Early Stopping

Training and Validation



Cross Validation