Deep Learning

What is Deep Learning ?

- Artificial Intelligence
  - ✓ Any technique that enables computers to mimic human behavior
- Machine Learning
  - ✓ Ability to learn without explicitly being programmed
- Deep Learning
  - ✓ Extract patterns from data using neural networks

Why Deep Learning and Why Now ?

Why Deep Learning ?

- Hand engineered features are time consuming, brittle, and not scalable in practice

- Can we learn the underlying features directly data?
  - ✓ Data-driven approach

- Features in Face Images
  - ✓ Low level features: Lines, edges
  - ✓ Mid level features: Eyes, Nose, ears
  - ✓ High level features: Facial structure

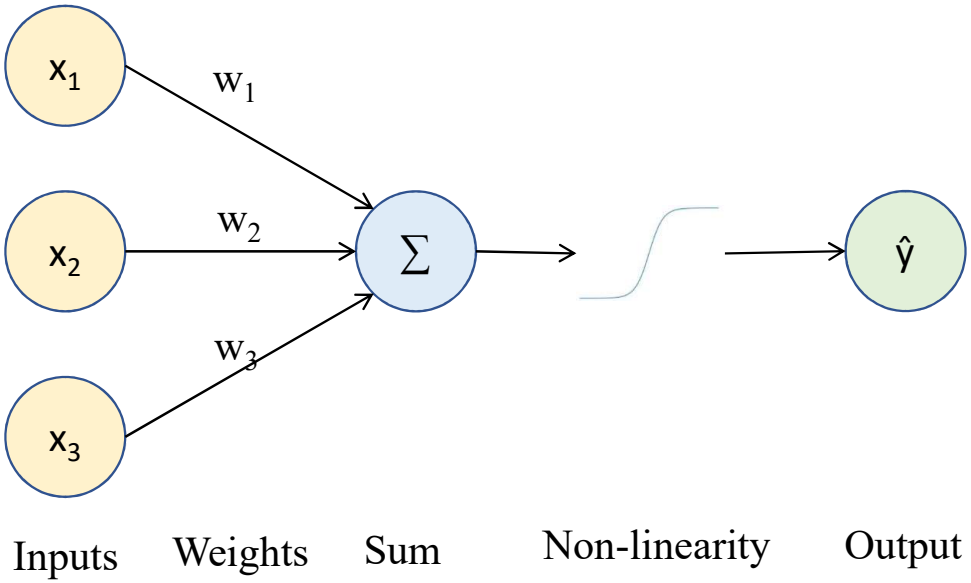
Why Now ?

- Neural networks study
  - ✓ 1952: Stochastic gradient descent
  - ✓ 1958: Perceptron
  - ✓ 1986: Backpropagation
  - ✓ 1995: Deep Convolutional NN

- Neural networks date back decades, so why the resurgence?
  - ✓ Big Data
    - ➢ Larger datasets, easier collection & storage
  - ✓ Hardware
    - ➢ Graphics processing units (GPUs), massively parallelizable
  - ✓ Software
    - ➢ Improved techniques, new models, toolboxes

The Perceptron
(The structural building block of deep learning)

The Perceptron: Forward Propagation



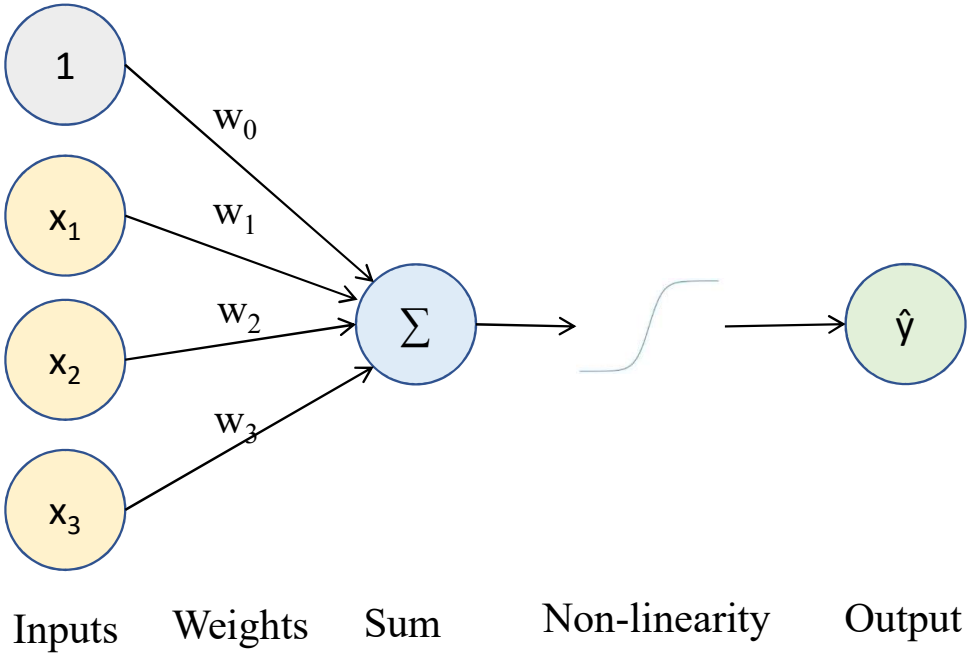
Linear combination of inputs

Output

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

Non-linear
Activation function

The Perceptron: Forward Propagation



Linear combination of inputs

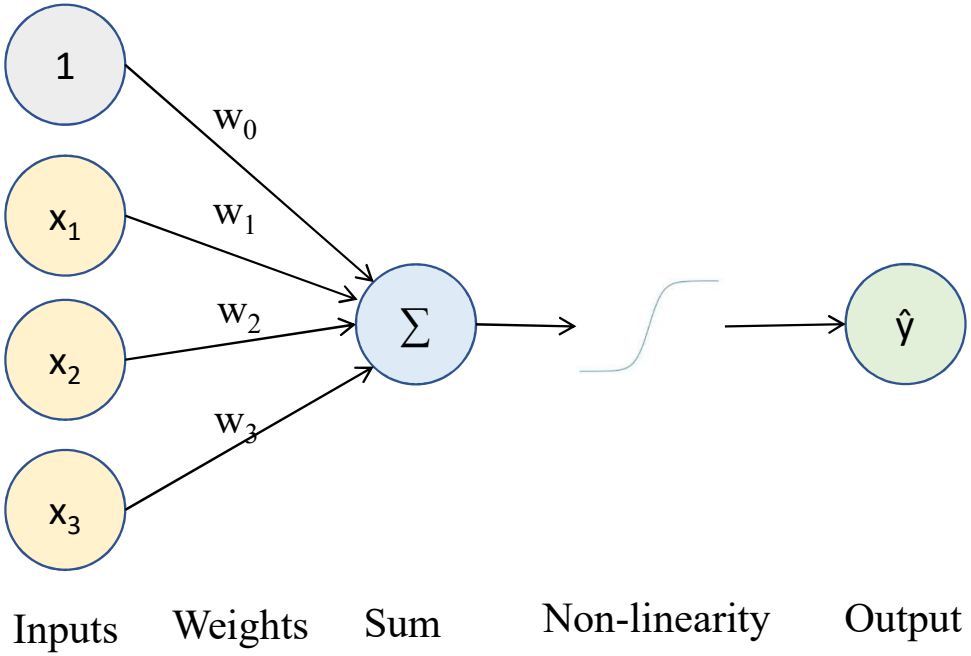
Bias

Output

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

Non-linear
Activation function

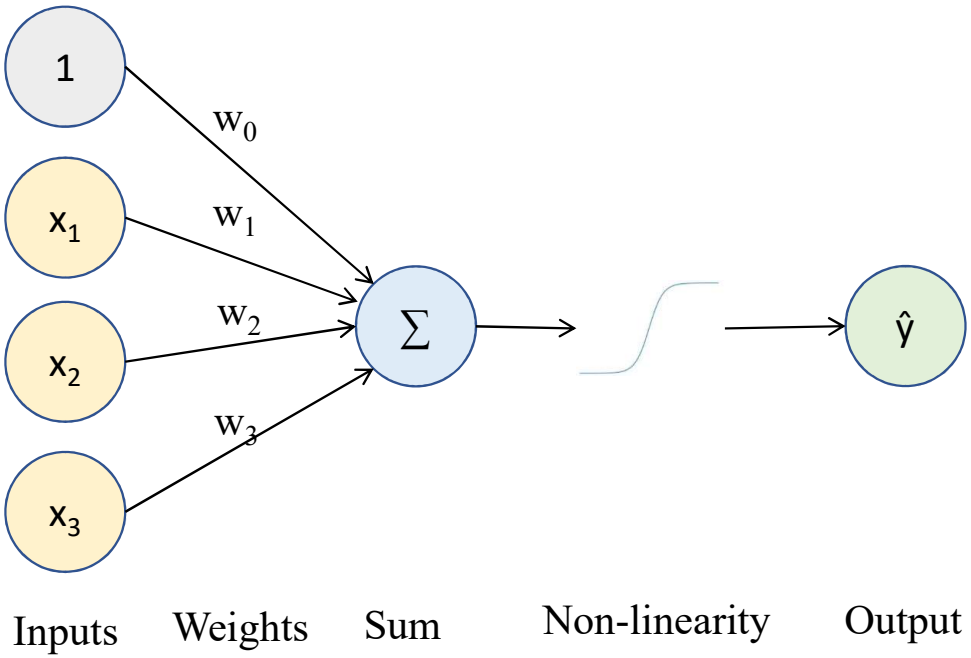
The Perceptron: Forward Propagation



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

$$\hat{y} = g\left(w_0 + X^T W\right), \quad X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}, W = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

The Perceptron: Forward Propagation

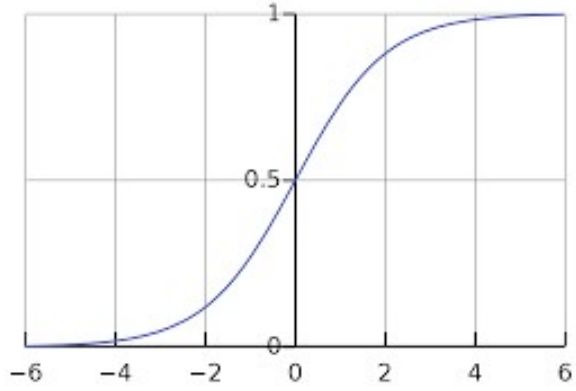


Activation Function

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

Sigmoid Function

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

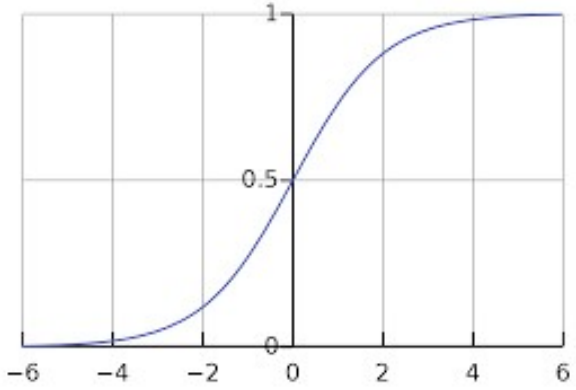


Common Activation Functions

Sigmoid Function

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

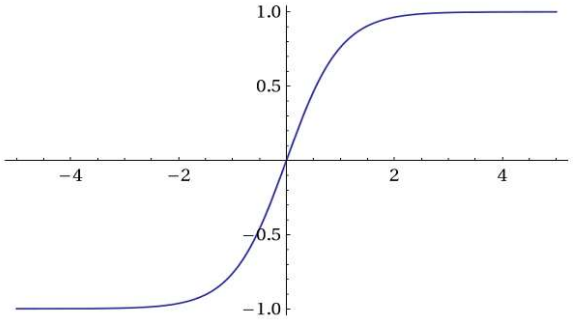
$$g'(z) = g(z)(1-g(z))$$



Hyperbolic Tangent (tanh)

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

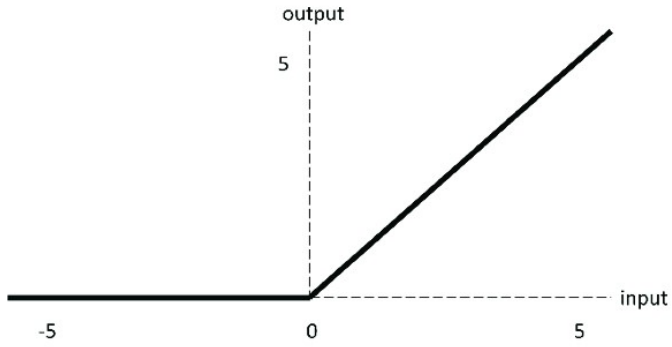
$$g'(z) = 1 - g(z)^2$$



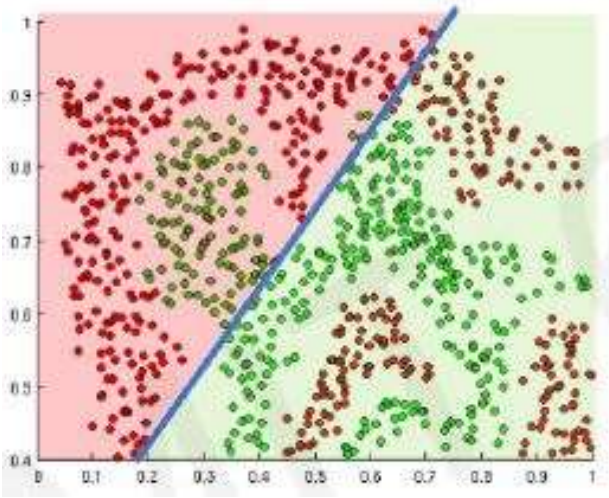
Rectified Linear Unit (ReLU)

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

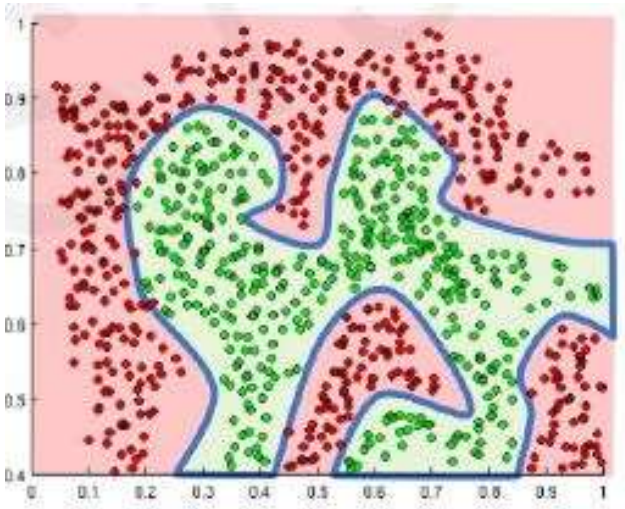
$$g'(z) = \begin{cases} 1, & z>0 \\ 0, & otherwise \end{cases}$$



Importance of Activation Functions

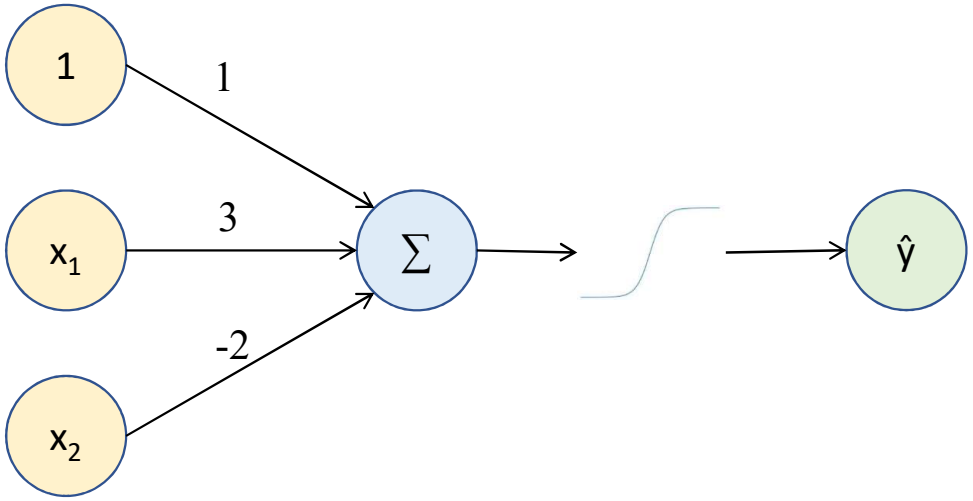


Linear activation functions produce linear decisions no matter the network size



Non-linearity allows us to approximate arbitrarily complex functions

The Perceptron: Example



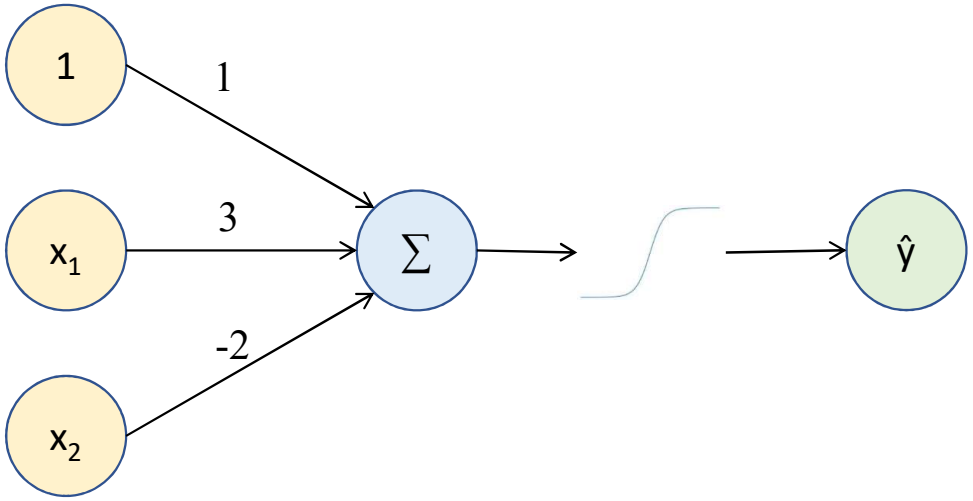
$$w_0 = 1, W = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$$

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

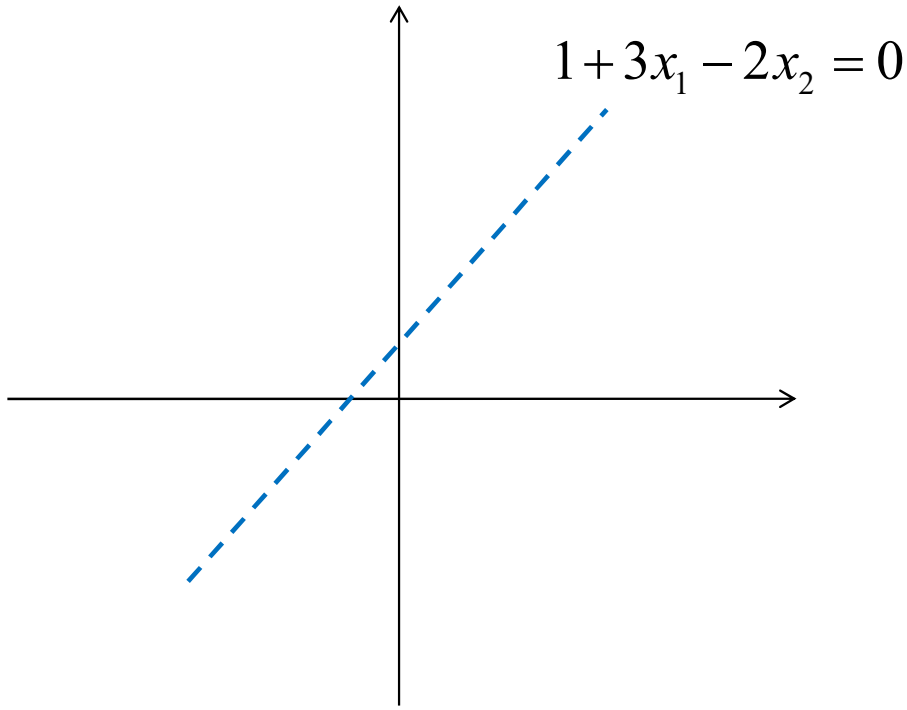
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

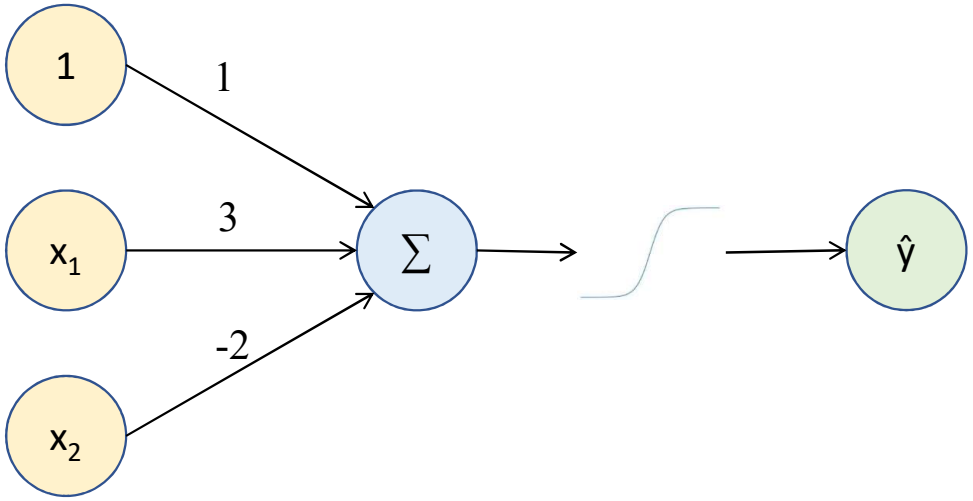
The Perceptron: Example



$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



The Perceptron: Example



$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

$$1 + 3x_1 - 2x_2 = 0$$

$z < 0$

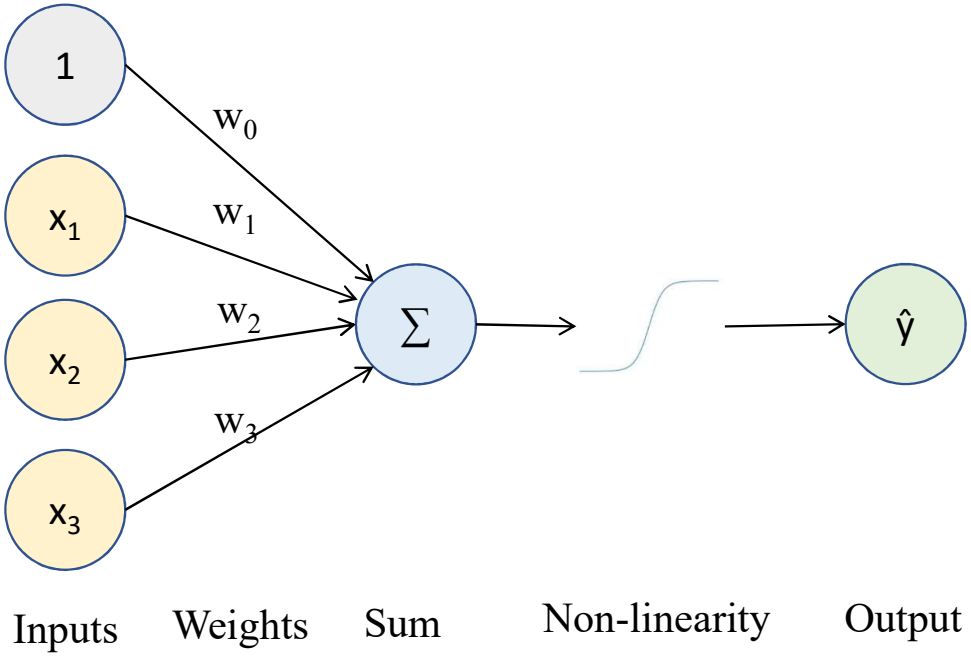
$y < 0.5$

$z > 0$

$y > 0.5$

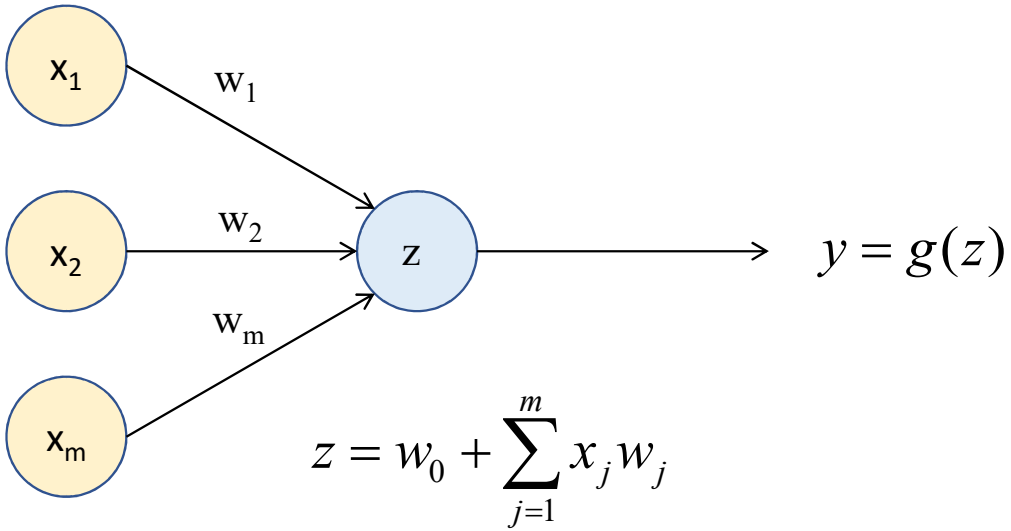
Building Neural Networks with Perceptrons

The Perceptron: Simplified



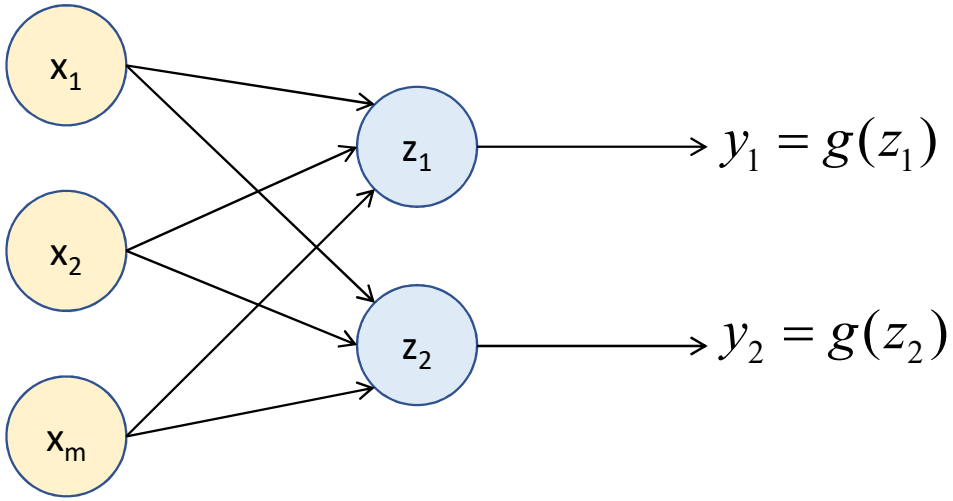
$$\hat{y} = g\left(w_0 + X^T W\right)$$

The Perceptron: Simplified



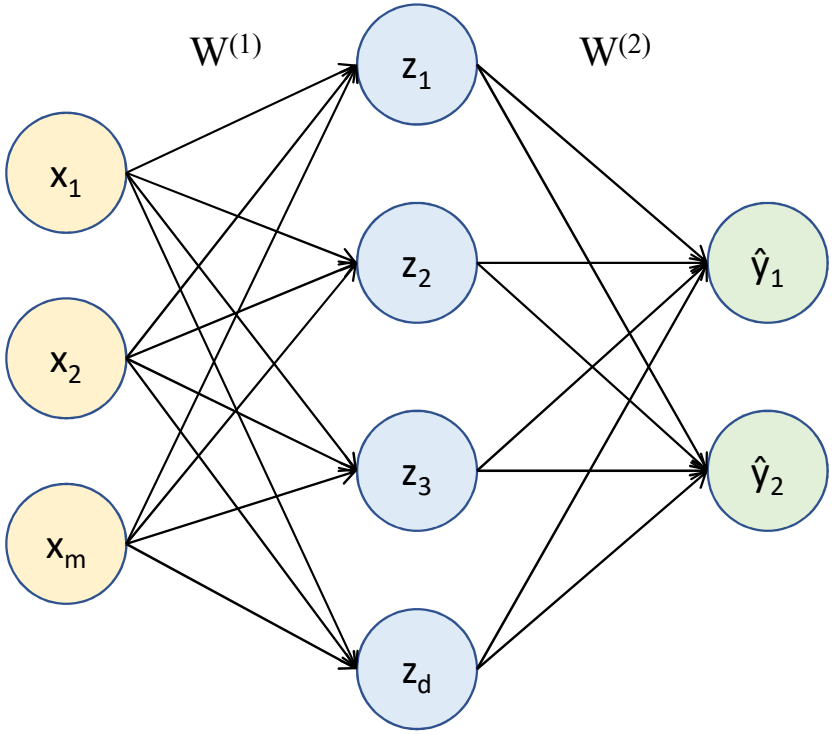
Multi Outputs

- Because all inputs are densely connected to all outputs, these layers are called Dense layers



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

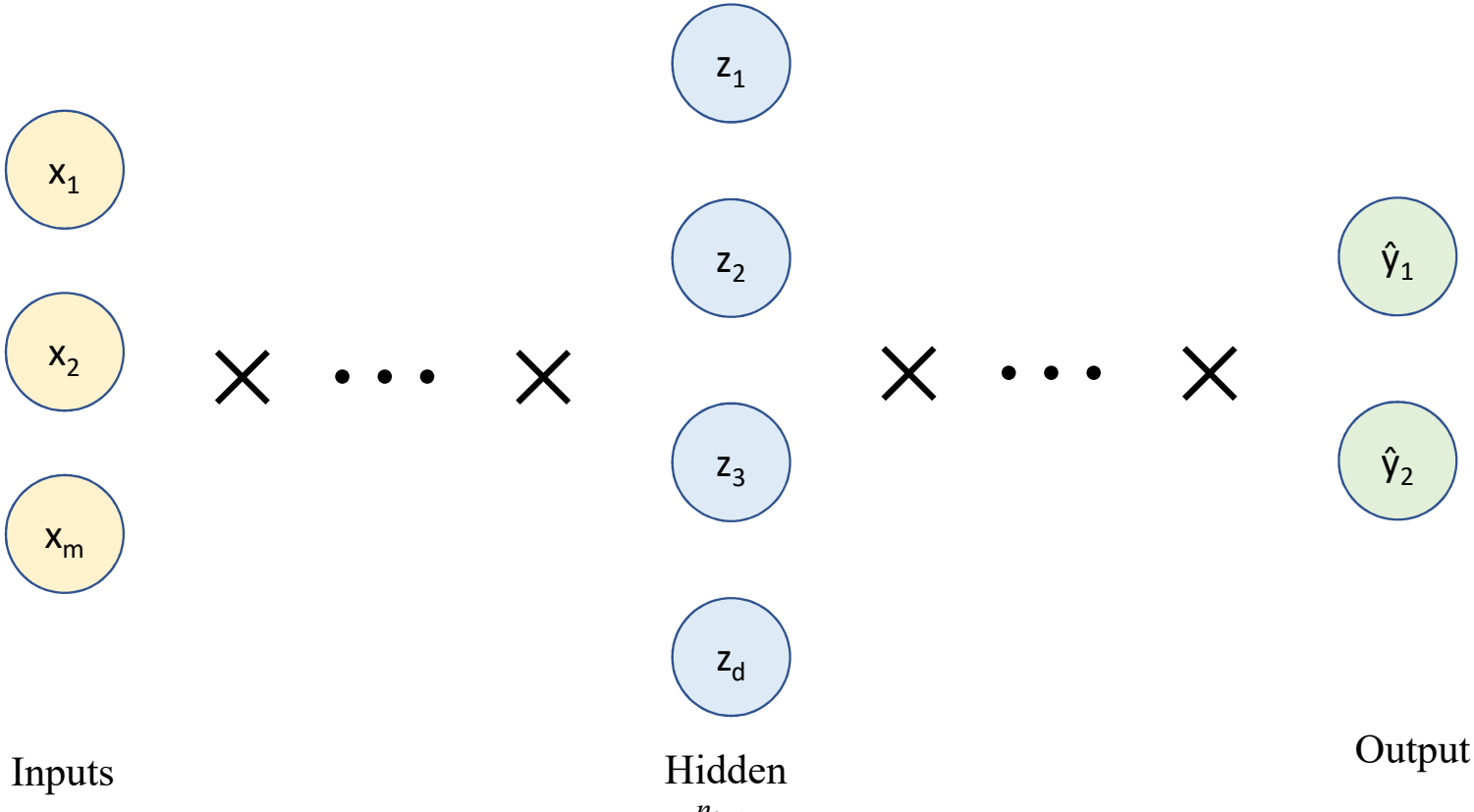
Single Layer Neural Network



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

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

Deep Neural Network



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

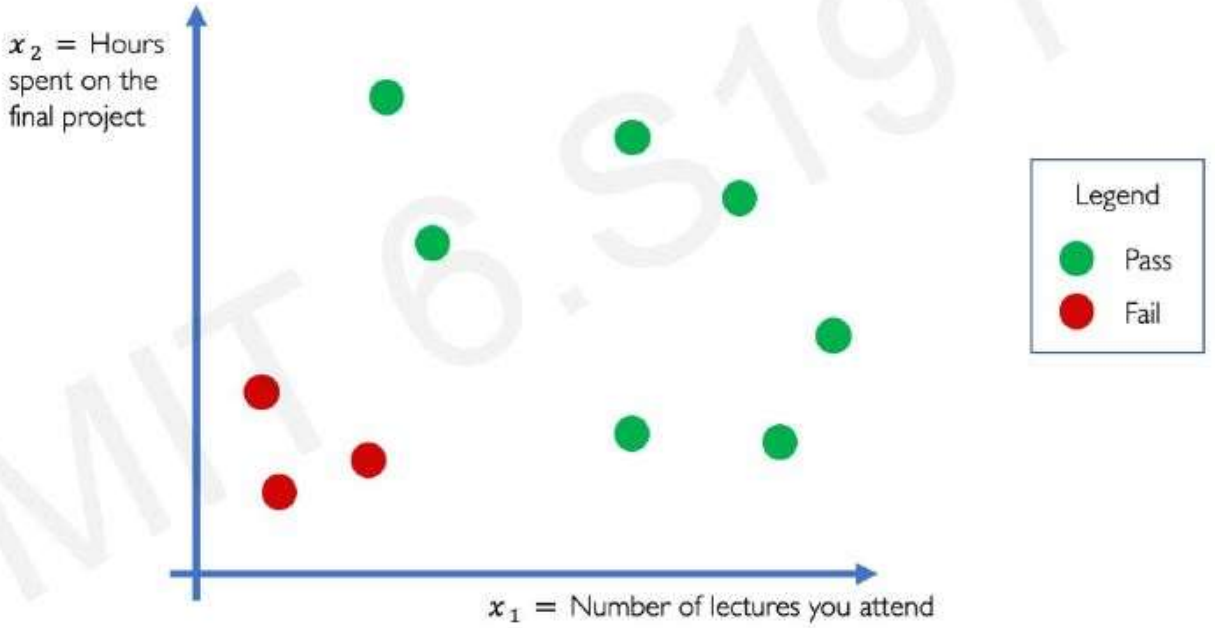
Applying Neural Networks

Example

Pass the class

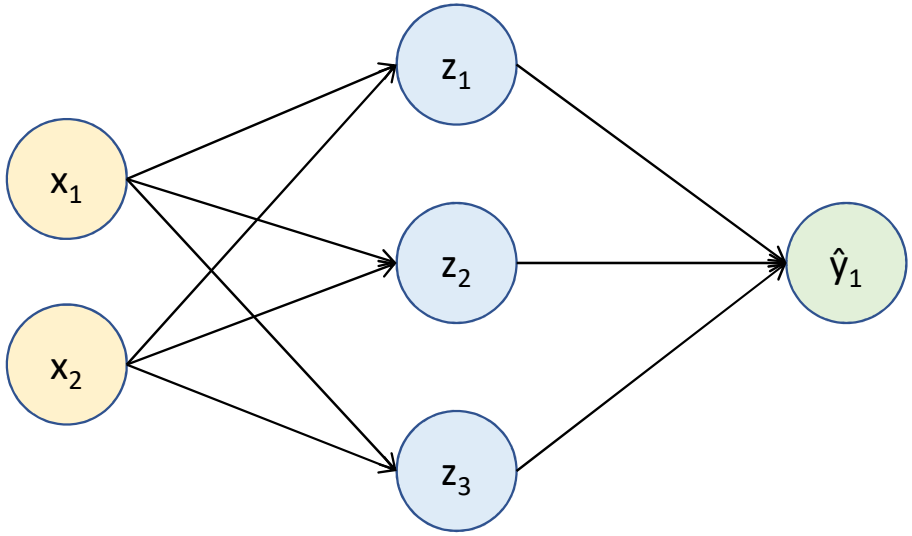
- Let's start with a simple two feature model
  - ✓ x_1 = # of lectures you attend
  - ✓ x_2 = Hours spent on the final project

Example



Example

$x^{(1)}=[4\ 5]$

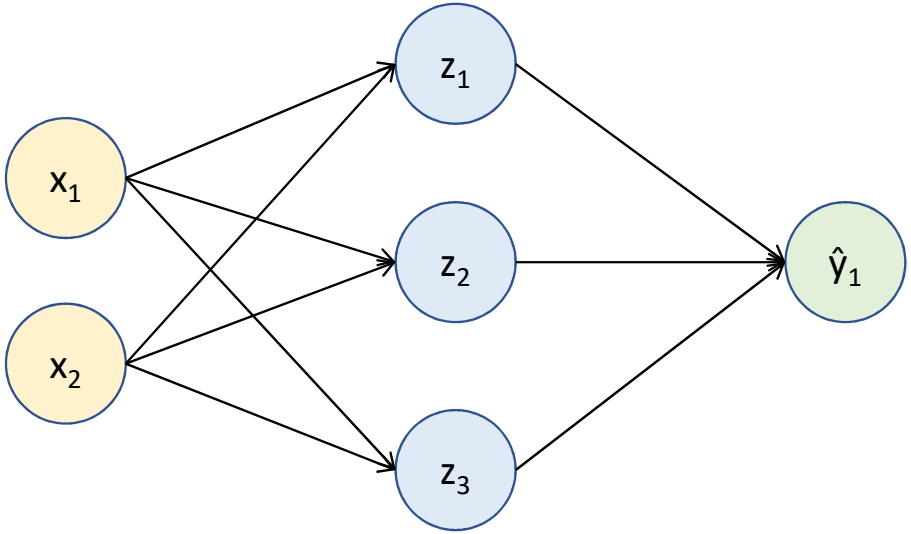


Prediction: 0.1
Actual:1

Quantifying Loss

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

$x^{(1)}=[4\ 5]$



Prediction: 0.1
Actual:1

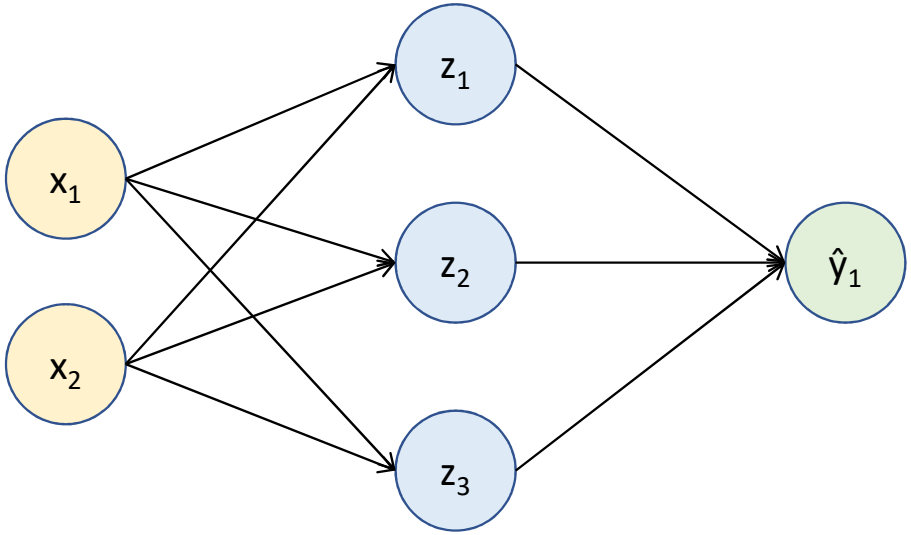
$$L\left(f(x^{(i)};W), y^{(i)}\right)$$

Predicted Actual

Empirical Loss

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

$\boldsymbol{x}=[x_1\ x_2]$



Prediction: $f(\boldsymbol{x})$
Actual: $\boldsymbol{y}$

Objective function
Cost function
Empirical Risk

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

Mean Squared Error Loss

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

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

Actual Predicted

Binary Cross Entropy Loss

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

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

Actual Predicted Actual Predicted

Loss Optimization

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

$$W^* = \arg\min_W \frac{1}{n} \sum_{i=1}^{n} L\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

$$= \arg\min_W J(W)$$

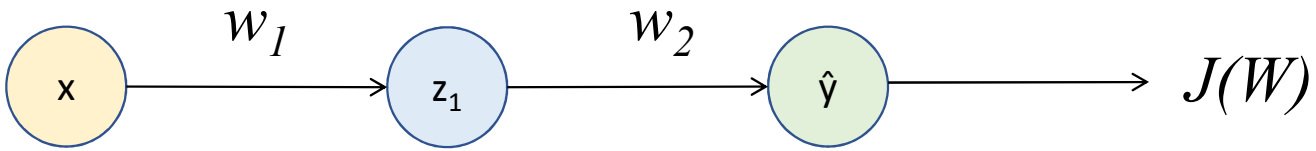
Gradient Descent Algorithm

- Loss is a function of the network weights

- Gradient Descent

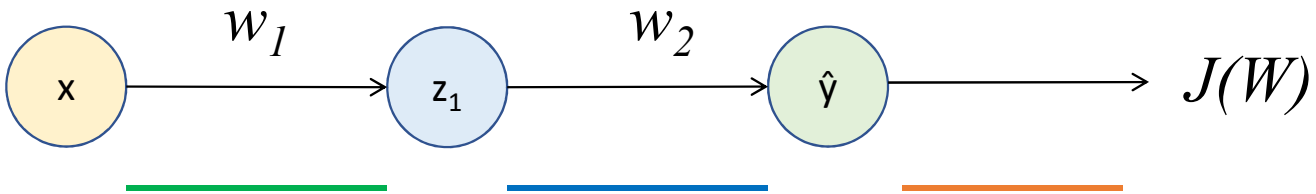
1. Initialize weights randomly $\sim N(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights

Computing Gradients: Backpropagation



Chain rule $$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_2}$$

Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_1} \times \frac{\partial z_1}{\partial w_1}$$

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

Neural Networks in Practice: Optimization

Loss functions can be difficult to optimize

- Optimization through gradient descent

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

Learning rate
- How can we set ?

✓Small learning rates converge 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

Adaptive Learning Rates

- Learning rates are no longer fixed

- Can be made larger or smaller depending on:
  - ✓How large gradient is
  - ✓How fast learning is happening
  - ✓Size of particular weights
  - ✓Etc.

Optimizer - Adaptive Learning Rates

- Algorithm
  - ✓ SGD
  - ✓ Adam
  - ✓ Adadelta
  - ✓ Adagrad
  - ✓ RMSProp

Neural Networks in Practice: Mini-batches

Gradient Descent

- Loss is a function of the network weights

- Gradient Descent

1. Initialize weights randomly $\sim N(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$ ***Can be very computationally intensive to compute***
4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights

Stochastic Gradient Descent

- Stochastic Gradient Descent

1. Initialize weights randomly $\sim N(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(W)}{\partial W}$ ***Easy to compute but very noisy (stochastic)***
5. Update weights, $W \leftarrow W - \eta \frac{\partial J_i(W)}{\partial W}$
6. Return weights

Stochastic Gradient Descent

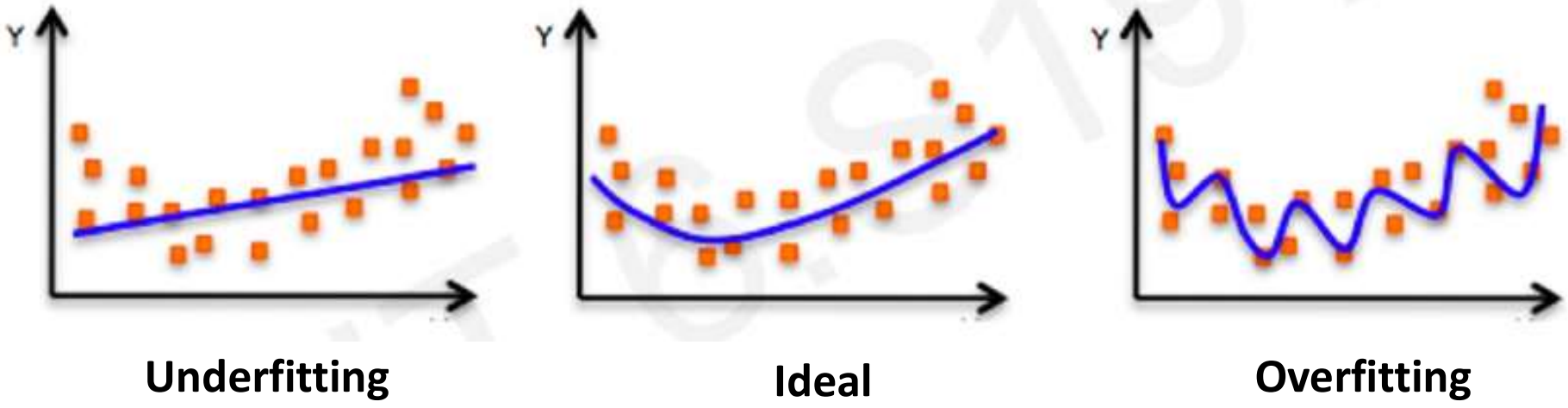
- Stochastic Gradient Descent

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

Fast to compute and a much Better estimate of the true gradient

Neural Networks in Practice: Overfitting

The problem of overfitting



Underfitting **Ideal** **Overfitting**

- Underfitting – Model does not have capacity to fully learn the data
- Overfitting – Too complex, extra parameters, does not generalize well

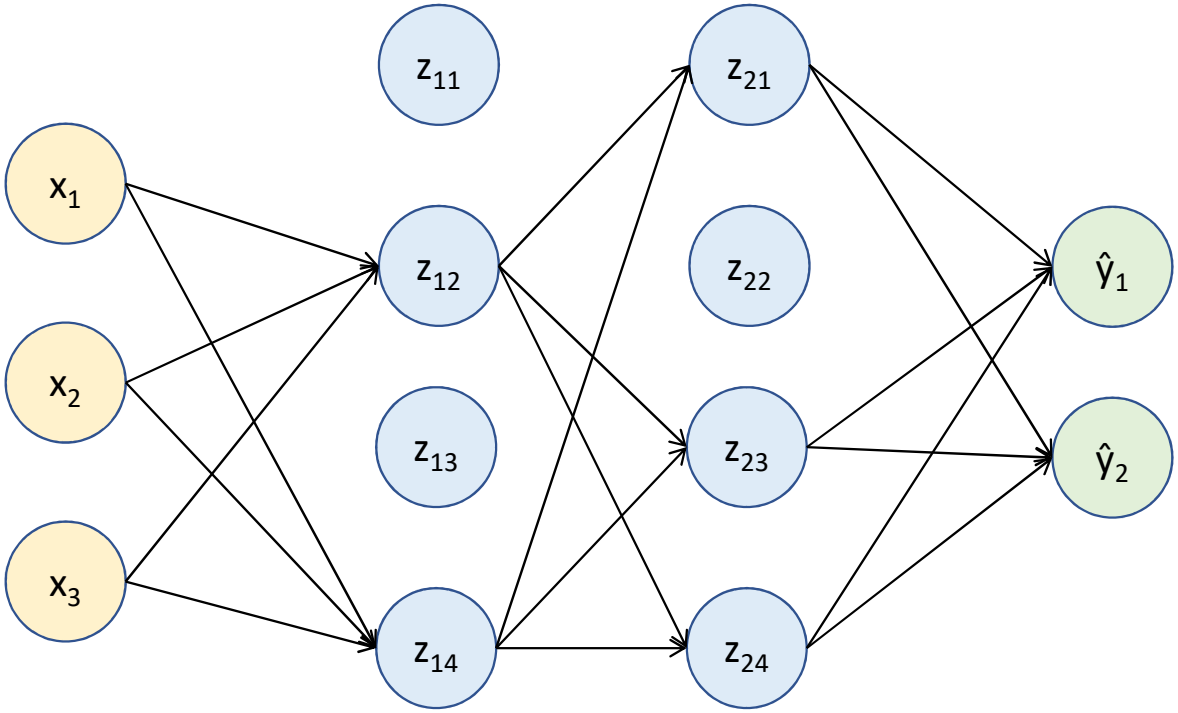
Regularization

- Technique that constrains our optimization problem to discourage complex models

- Improve generalization of our model on unseen data

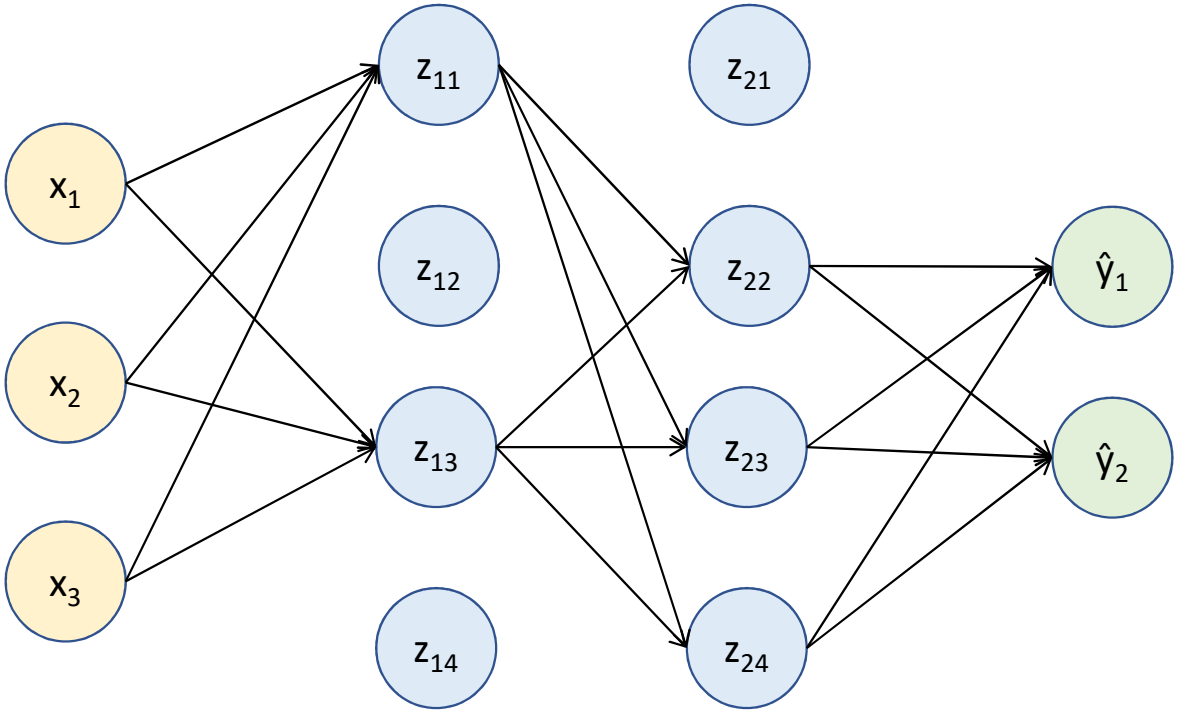
Regularization – Dropout

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



Regularization – Dropout

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



Regularization – Early Stopping

- Stop training before we have a chance to overfit

END