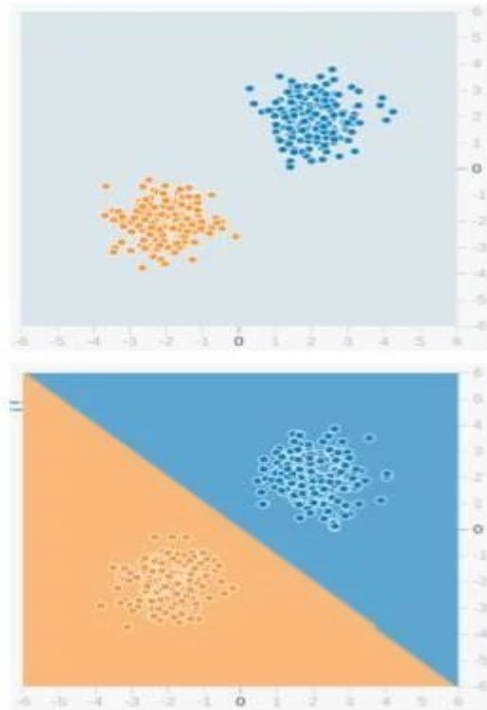


# K6312 Information Mining & Analysis

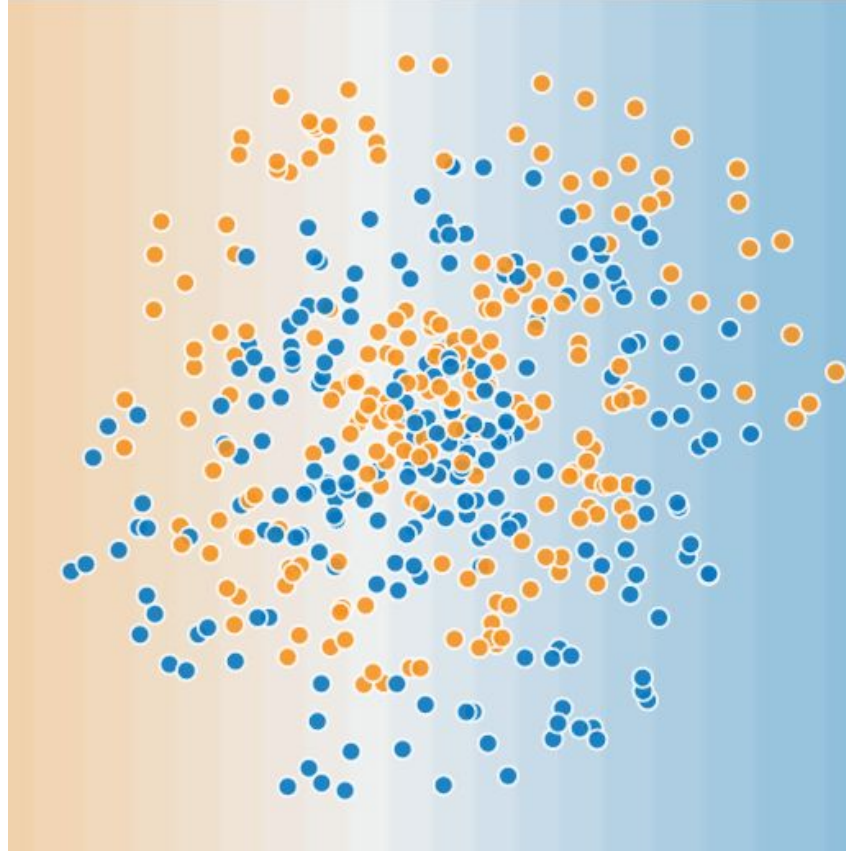
Chen Zhenghua & Zhao Rui

# Neural Networks

# A “Simple” Classification Problem

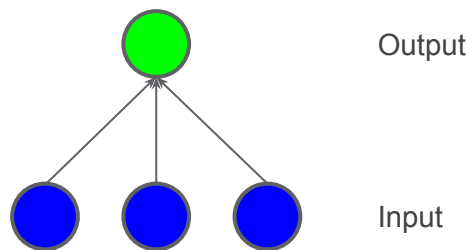


How about this classification problem?



# A Linear Model: Simplest Neural Network

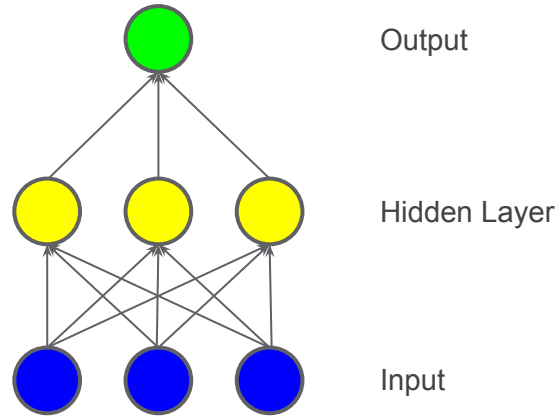
- Linear Regression if output is continuous
- Logistic Regression if output is discrete



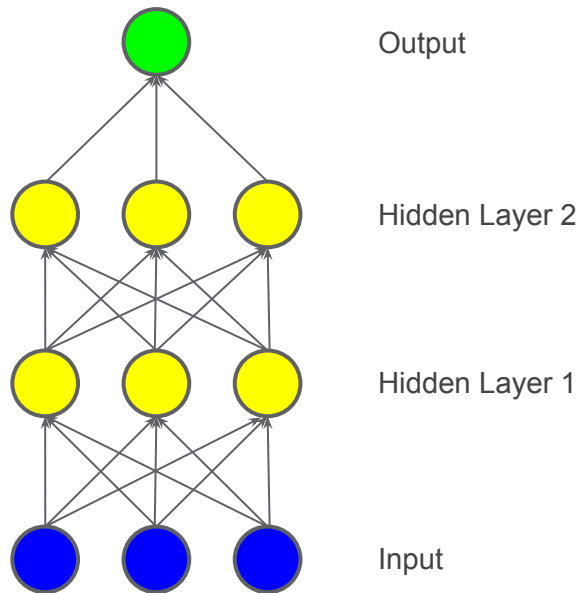
$$f(x) = w * x + b$$

$$f(x) = \sigma(w * x + b) = \frac{1}{1 + e^{-(w * x + b)}}$$

# Add Complexity: Is it not linear now?

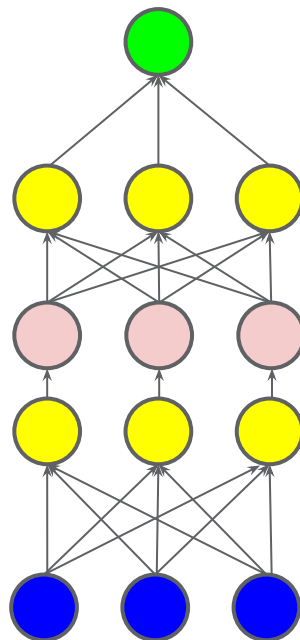


# How about now?



# Make it non-linear

We Usually Don't  
Draw Non-Linear  
Transforms



Output

Hidden Layer 2

Non-Linear Transformation Layer  
(a.k.a. Activation Function)

Hidden Layer 1

Input

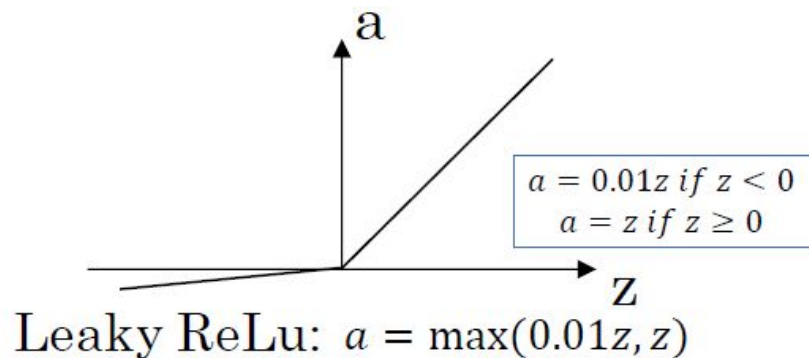
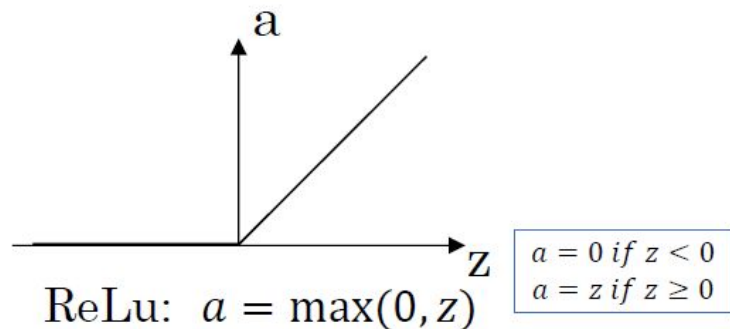
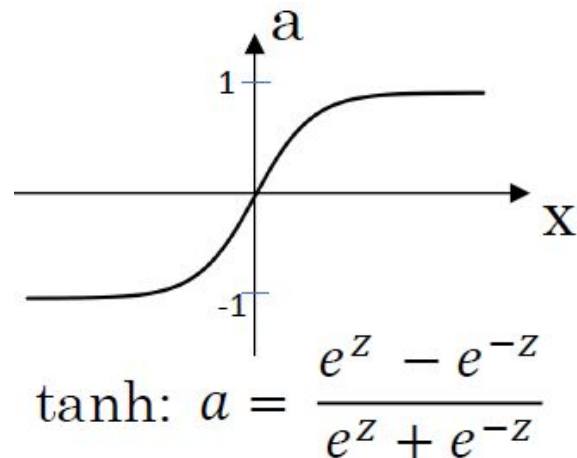
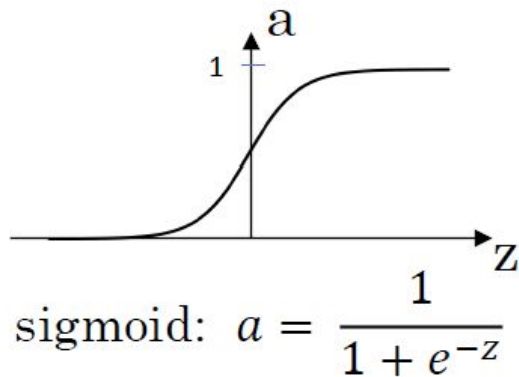


# Why Non-linear Activation

- The non-linearities activation function increases the capacity of model
- Without non-linearities, neural networks is meaningless: each extra layer is just one linear transform.
- How to select activation functions?

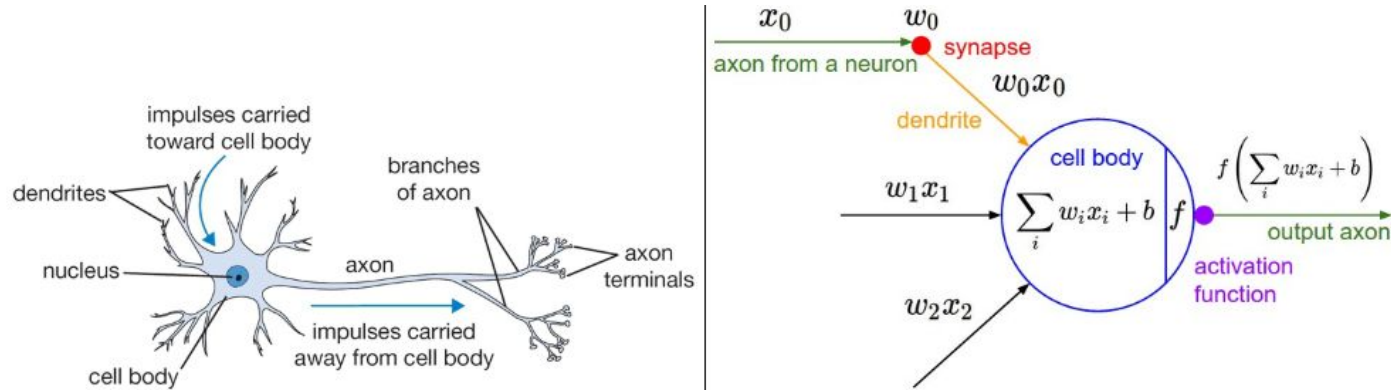
*You can select an activation function which will approximate the distribution faster leading to faster training process.*

# Non-linear activation function



# Neural Networks

- From Wiki: NN is based on a collection of connected units of nodes called artificial **neurons** which loosely model the neurons in a biological brain.
- The fact that a neuron is essentially a logistic regression unit:
  - performs a dot product with the input and its weights
  - adds the bias and apply the non-linearity

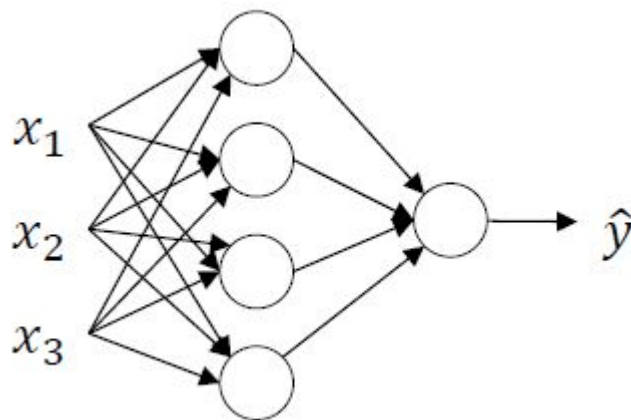


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

# Forward Operation

# Neural Network Representation

- Input layer: attributes or features
- Hidden layer: insert non-linearity
- Output layer: Target

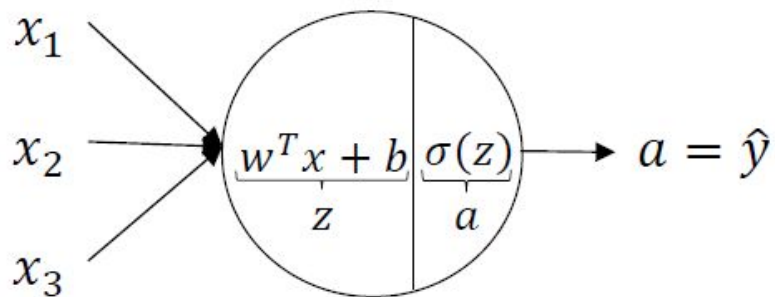


Input layer

Hidden layer

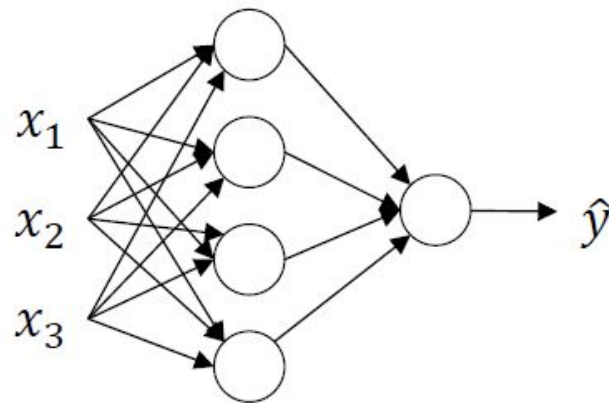
Output layer

# Neural Network Representation



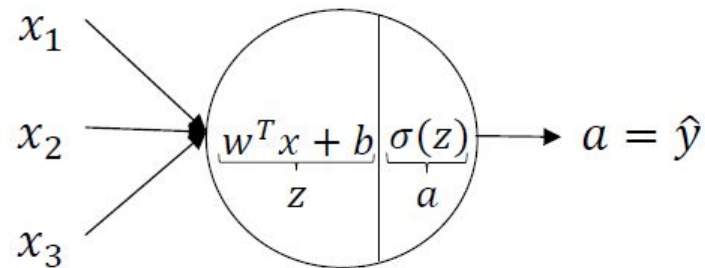
Computing One  
Neuron Network's  
Output

$$z = w^T x + b$$
$$a = \sigma(z)$$



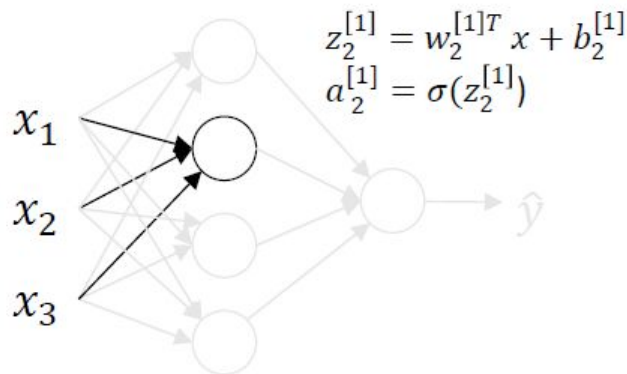
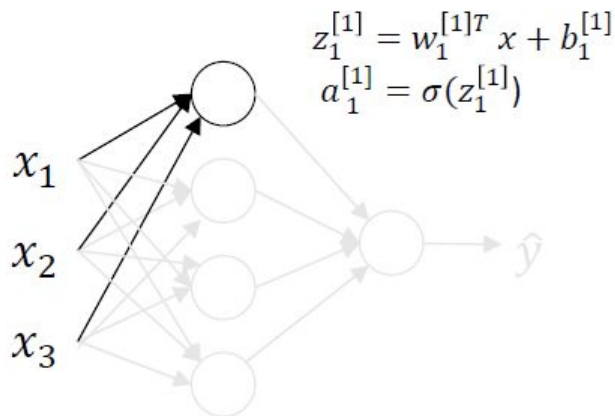
Computing a  
Neural Network's  
Output

# Neural Network Representation

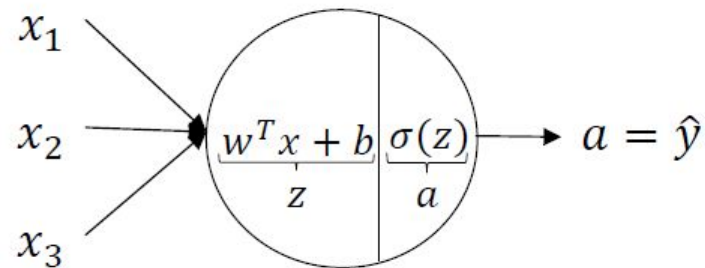


$$z = w^T x + b$$

$$a = \sigma(z)$$

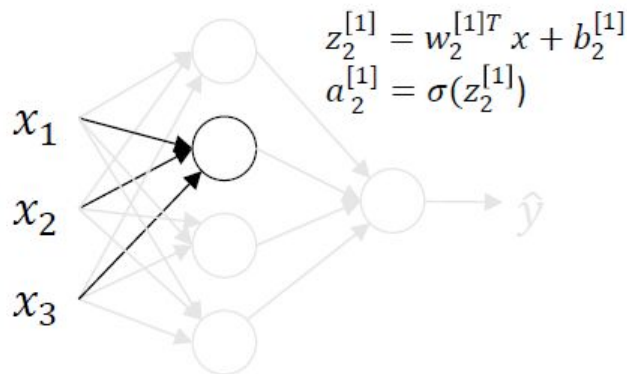
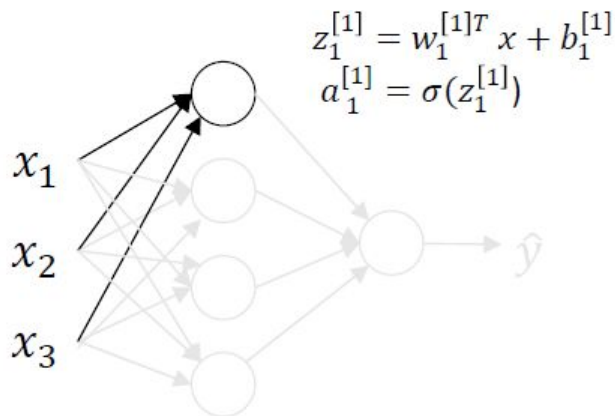


# Neural Network Representation



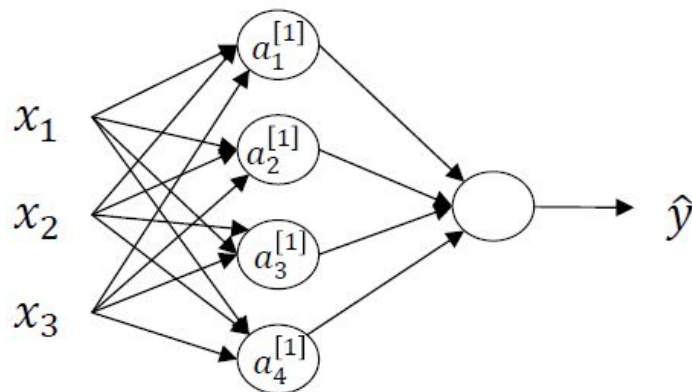
$$z = w^T x + b$$

$$a = \sigma(z)$$





# Neural Network Representation



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$

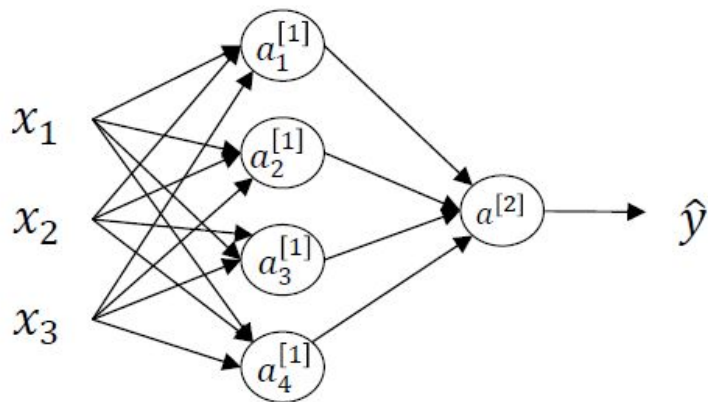
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$W^{[1]}$

$$z^{[1]} = \begin{bmatrix} -w_1^{[1]T} & - \\ -w_2^{[1]T} & - \\ -w_3^{[1]T} & - \\ -w_4^{[1]T} & - \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}, \quad \sigma(z^{[1]}) = a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

# Neural Network Representation



Given input  $x$ :

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$(4,1) \quad (4,3) \quad (3,1) \quad (4,1)$

$$a^{[1]} = \sigma(z^{[1]})$$

$(4,1) \quad (4,1)$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$(1,1) \quad (1,4) \quad (4,1) \quad (1,1)$

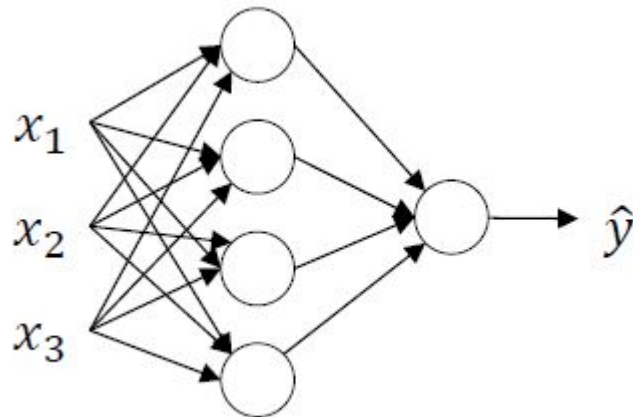
$$a^{[2]} = \sigma(z^{[2]})$$

$(1,1) \quad (1,1)$

# Neural Network for Regression

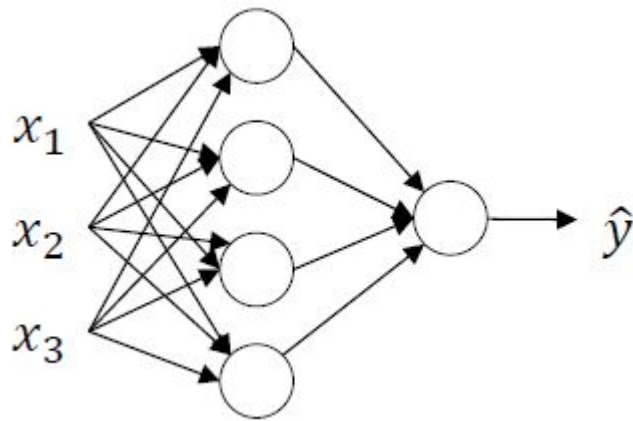
- One Output: Real value
- Loss Function: MSE

$$\text{MSE} = \frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2$$



# Neural Network for Classification

- One Output: Probability
- Loss Function: Cross Entropy



$$\text{Cross\_Entropy} = -\frac{1}{m} \left[ \sum_{i=1}^m y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \right]$$

# Neural Network for Classification (one-vs-all)



Pedestrian



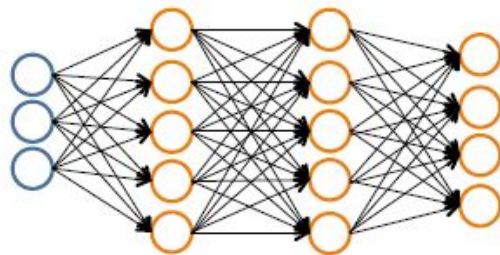
Car



Motorcycle



Truck



$$h_{\Theta}(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

when pedestrian

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

when car

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

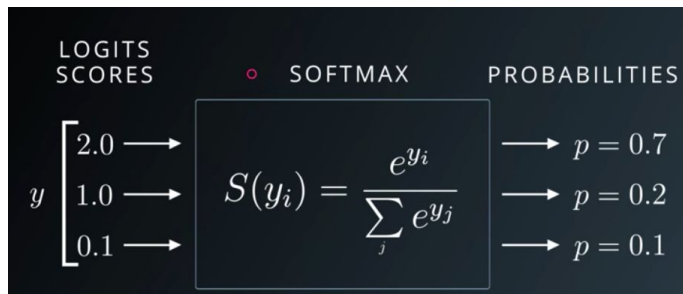
when truck

# Neural Network for Classification (one-vs-all)

- Given  $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$
- Must convert labels to 1-of- $K$  representation

– e.g.,  $y_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$  when motorcycle,  $y_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$  when car, etc.

Softmax  $S(x_j) = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}}, j = 1, 2, \dots, K$



$$\text{Cross\_Entropy} = -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_{ik} \log \hat{y}_{ik} + (1 - y_{ik}) \log(1 - \hat{y}_{ik}) \right]$$

# Sigmoid vs Softmax

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

$$S(x_j) = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}}, j = 1, 2, \dots, K$$

For binary:

$$p(y = 1|x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$p(y = 0|x) = 1 - p(y = 1|x) = \frac{e^{-\theta^T x}}{1 + e^{-\theta^T x}}$$

For Softmax when K=2

$$p(y = 1|x) = \frac{e^{\theta_1^T x}}{e^{\theta_0^T x} + e^{\theta_1^T x}} = \frac{1}{1 + e^{(\theta_0^T - \theta_1^T)x}} = \frac{1}{1 + e^{-\beta x}}$$

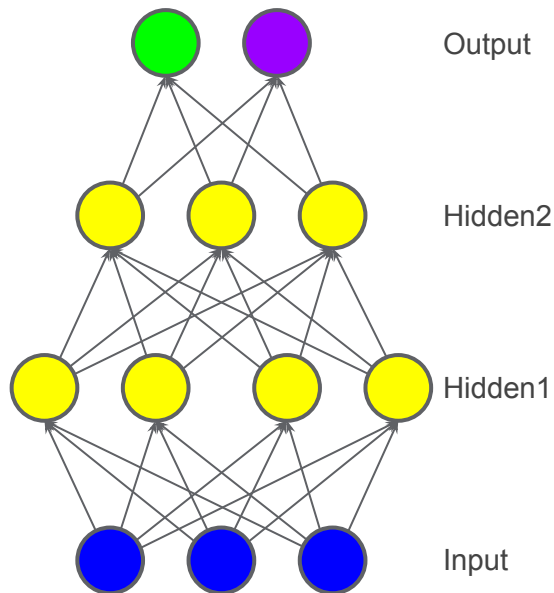
$$\beta = -(\theta_0^T - \theta_1^T)$$

$$p(y = 0|x) = \frac{e^{\theta_0^T x}}{e^{\theta_0^T x} + e^{\theta_1^T x}} = \frac{e^{(\theta_0^T - \theta_1^T)x}}{1 + e^{(\theta_0^T - \theta_1^T)x}} = \frac{e^{-\beta x}}{1 + e^{-\beta x}}$$

# Training



# Neural networks can be arbitrarily complex



Training done via  
BackProp algorithm:  
gradient descent in  
very non-convex  
space

# Gradient Descent Recap

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \nabla f(\mathbf{x}_n)$$

Diagram illustrating the Gradient Descent update formula:

- $\mathbf{x}_{n+1}$ : New Parameters Guess
- $\mathbf{x}_n$ : Current Parameters Guess
- $\alpha$ : Learning Rate
- $\nabla f(\mathbf{x}_n)$ : Gradient for loss function  $f$  for  $\mathbf{x}_n$ , which computed by BP algorithm

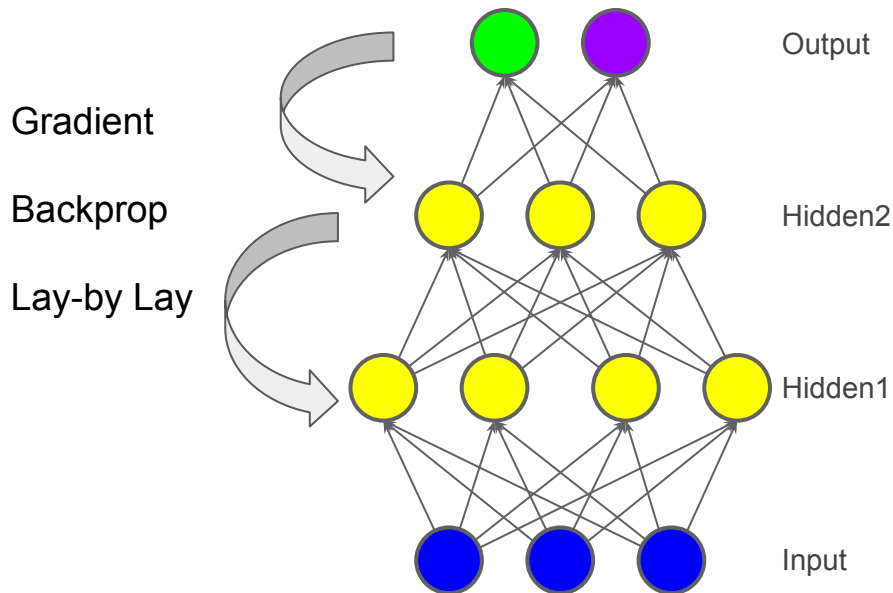


Like hiking down a mountain

# Training a Neural Network

1. Randomly initialize weights
2. Implement forward propagation to get output  $\hat{y}_i$  for each input data
3. Based on true output  $y_i$ , calculate loss function (e.g., MSE, Cross-Entropy)
4. Calculate gradient and update parameters layer-by-layer
5. Iterative for step 2

Loss: Calculated by model outputs and ground truth



# Neural Network Visualization

[Playground](#)