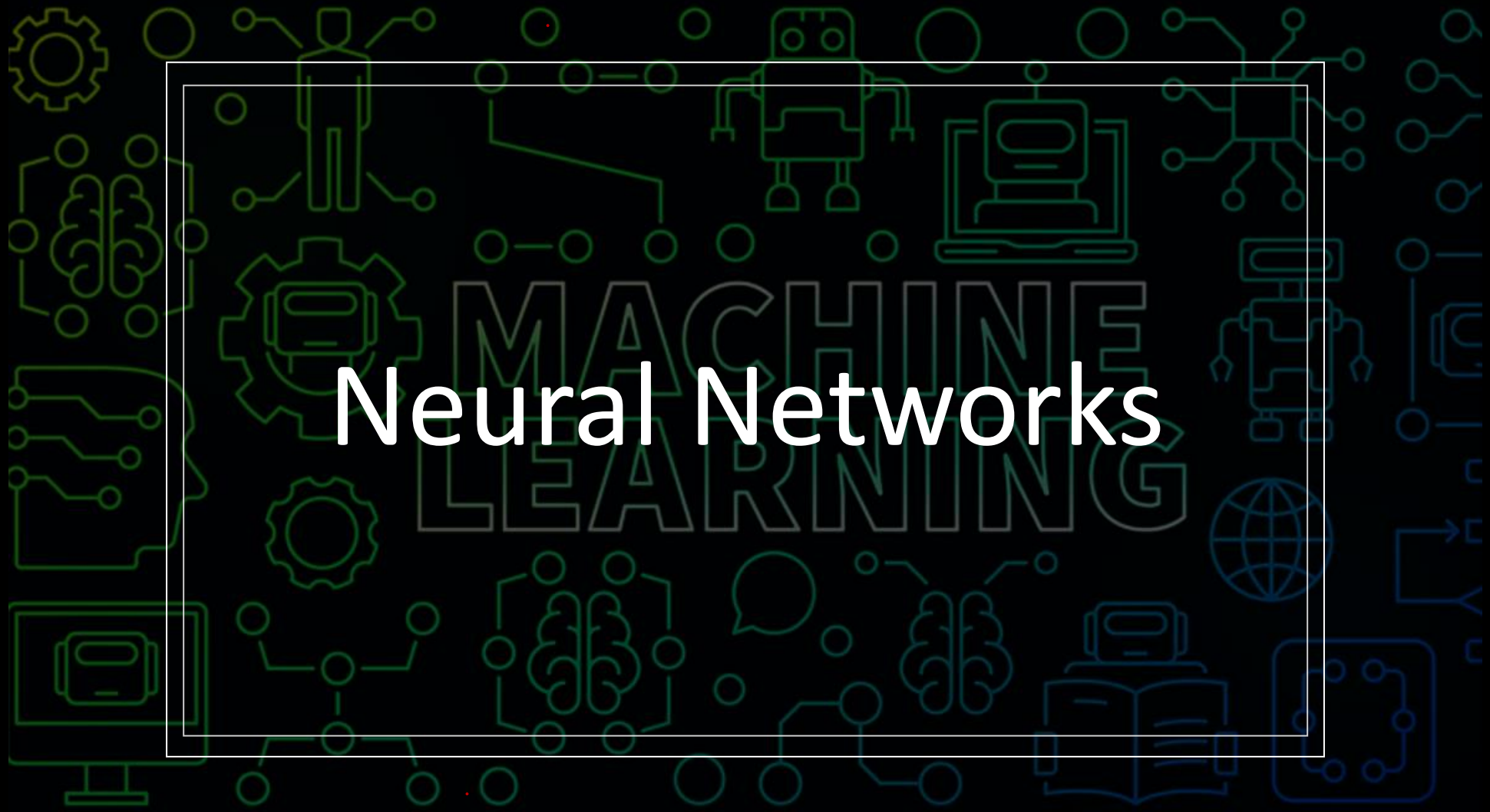
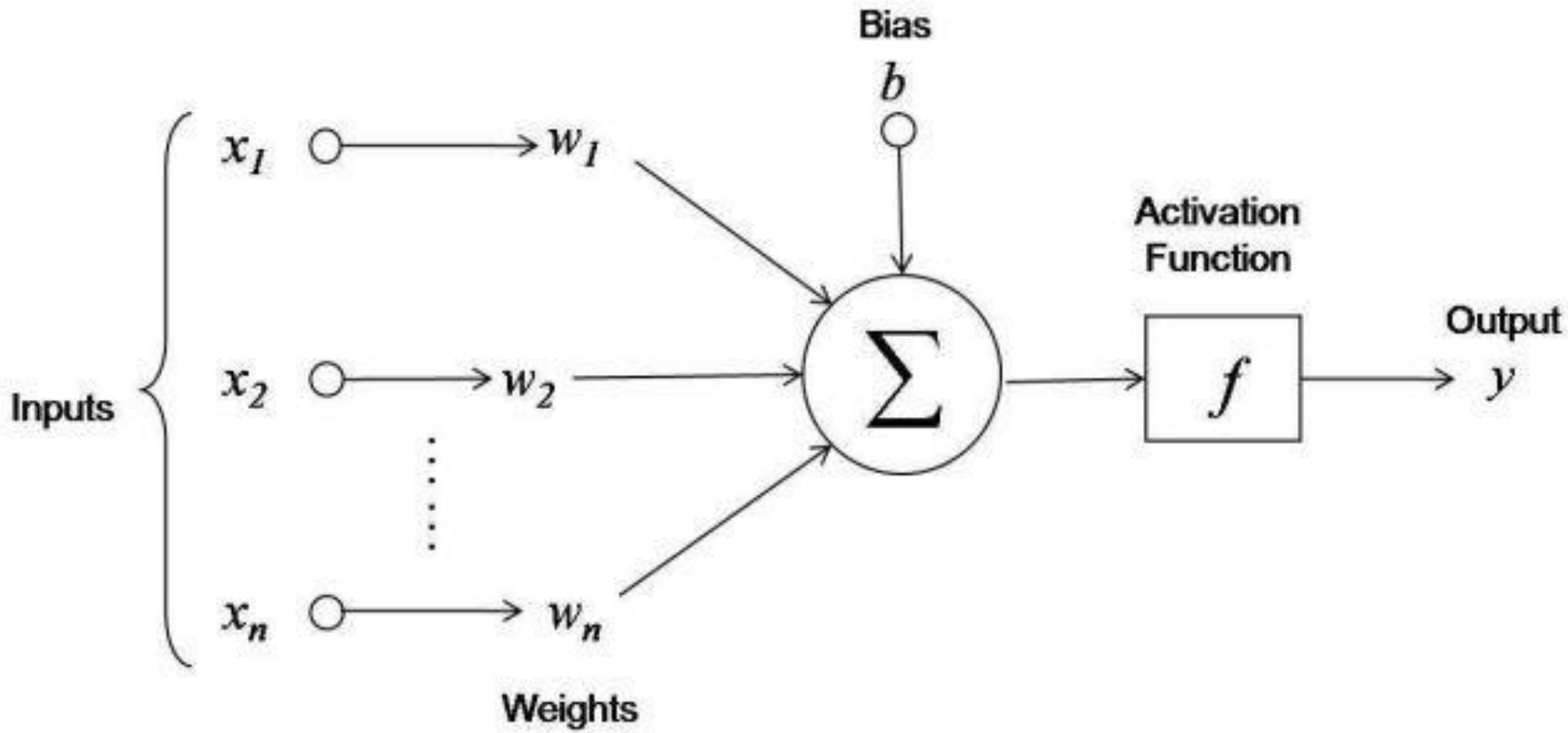
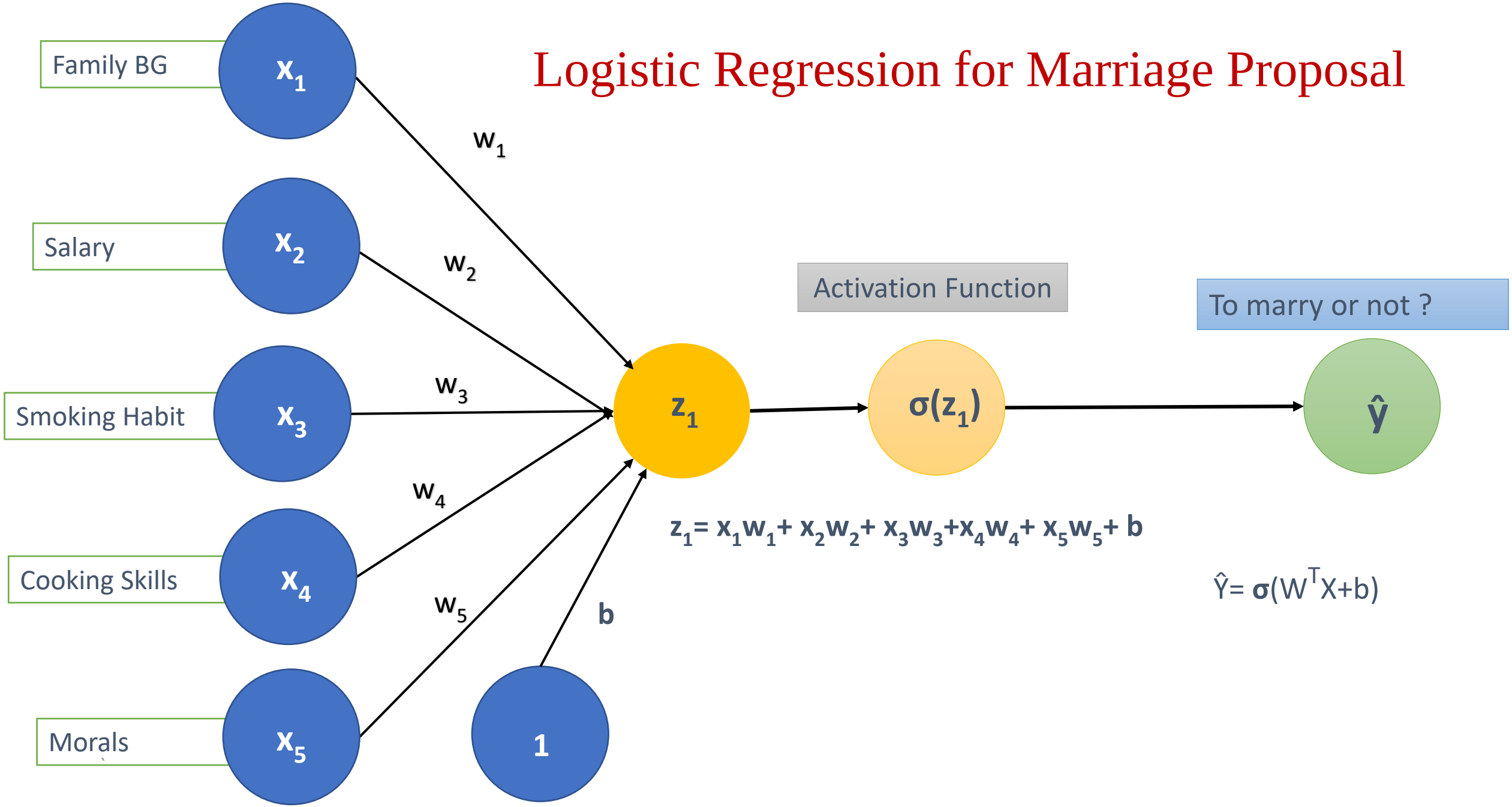


Neural Networks

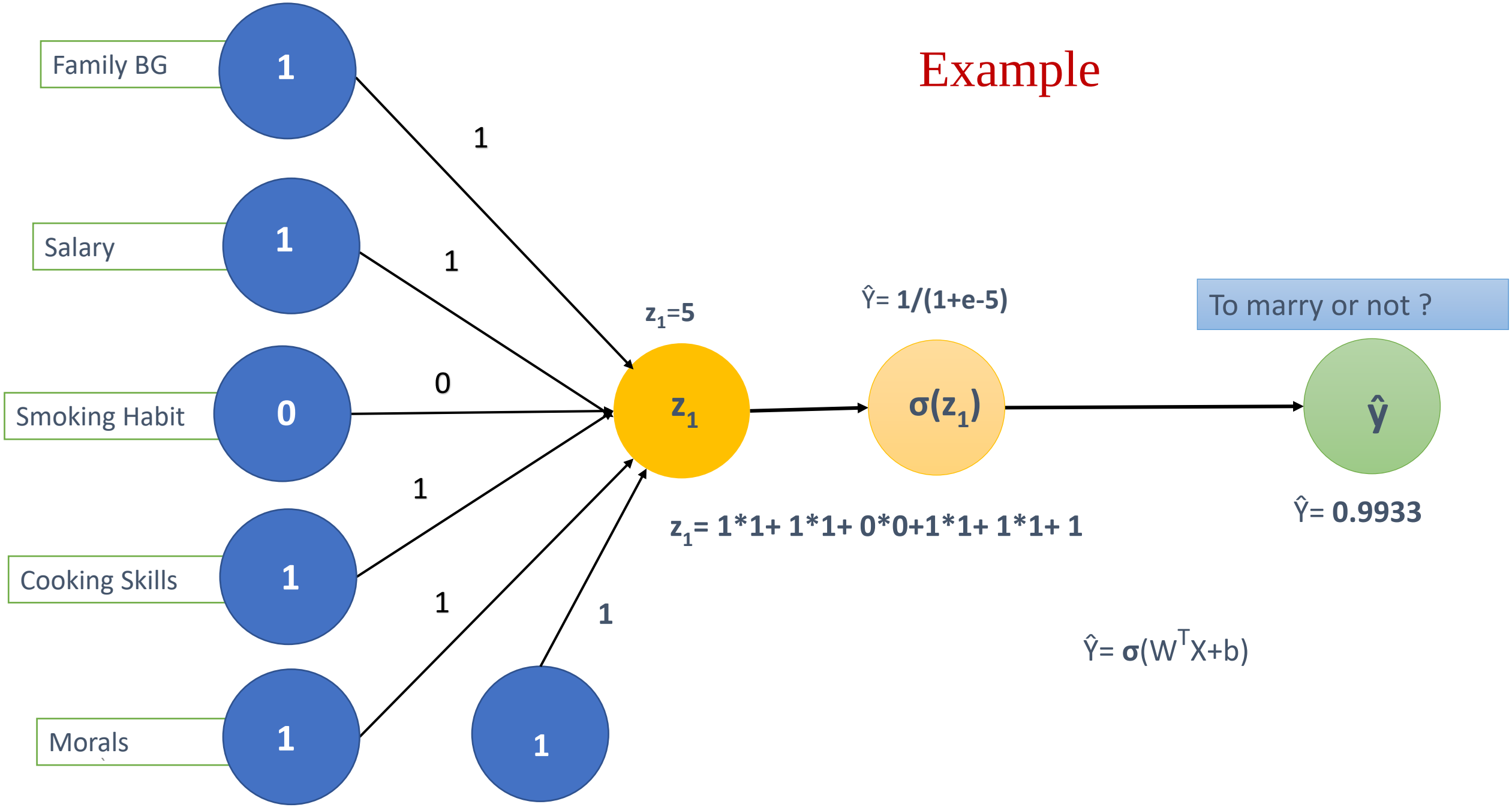
Linear Classification



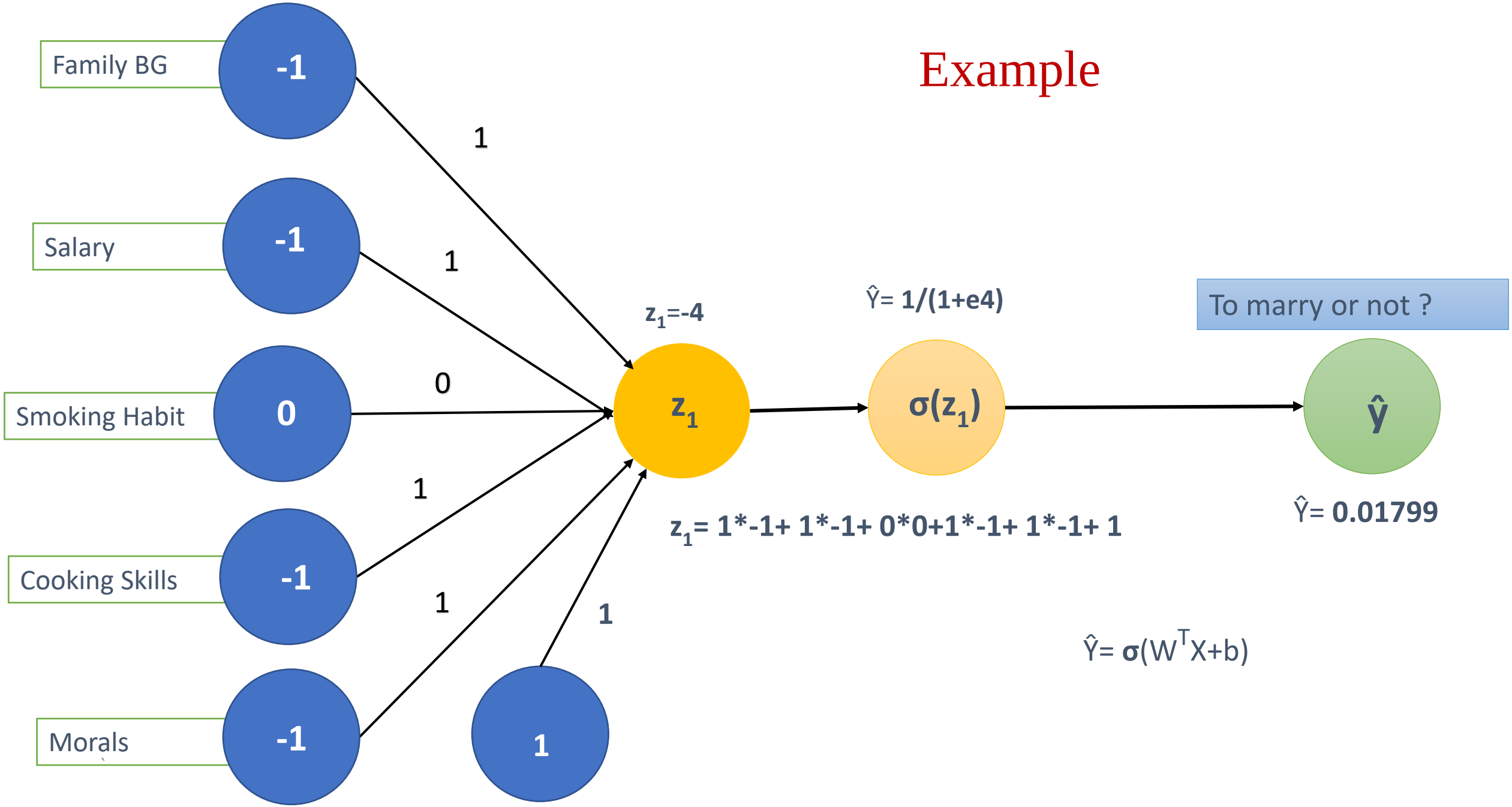
Logistic Regression for Marriage Proposal

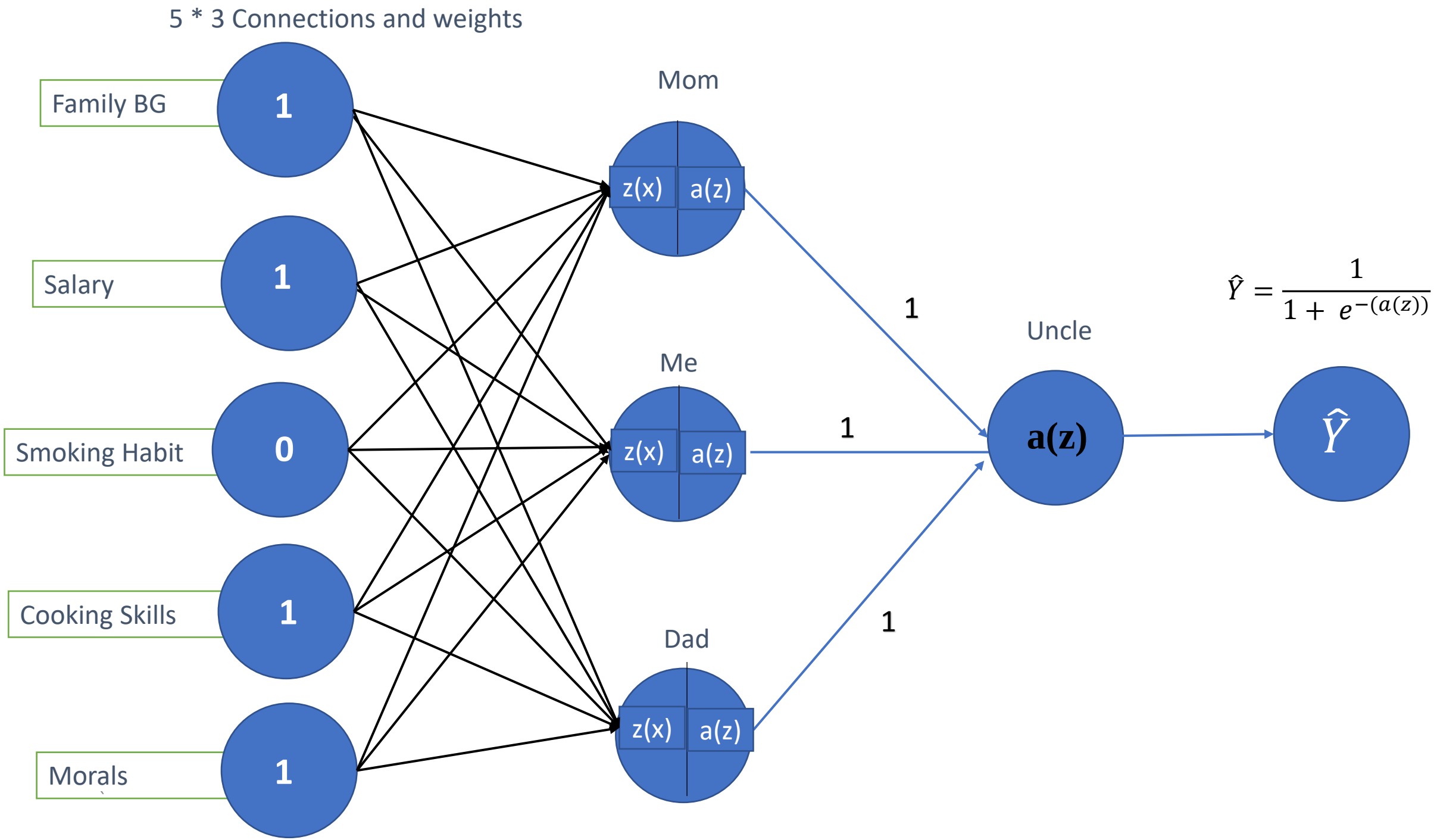


Example


Family BG
1
Salary
1
Smoking Habit
0
Cooking Skills
1
Morals
1
1
1
0
1
1
1
z_1=5
z_1
$\hat{Y}$= 1/(1+e-5)
$\sigma(z_1)$
To marry or not ?
$\hat{y}$
$\hat{Y}$= 0.9933
z_1= 1*1+ 1*1+ 0*0+1*1+ 1*1+ 1
$\hat{Y}= \sigma(W^T X+b)$

Example

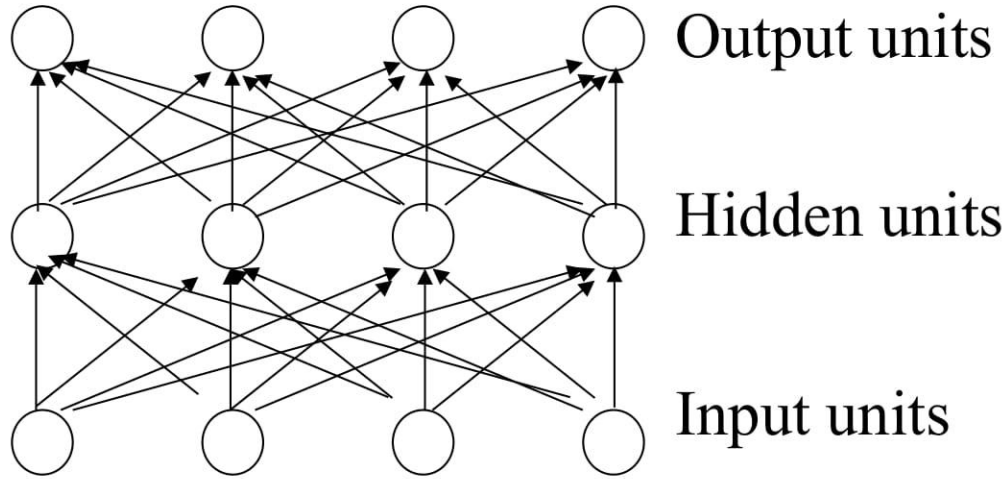

Family BG
-1
Salary
-1
Smoking Habit
0
Cooking Skills
-1
Morals
-1
1
1
0
1
1
1
z_1=-4
z_1
$\hat{Y}$= 1/(1+e4)
$\sigma(z_1)$
To marry or not ?
$\hat{y}$
$\hat{Y}$= 0.01799
z_1= 1*-1+ 1*-1+ 0*0+1*-1+ 1*-1+ 1
$\hat{Y}= \sigma(W^T X+b)$


5 * 3 Connections and weights
Family BG
Salary
Smoking Habit
Cooking Skills
Morals
1
1
0
1
1
Mom
Me
Dad
z(x)
a(z)
z(x)
a(z)
z(x)
a(z)
1
1
1
Uncle
a(z)
$\hat{Y} = \frac{1}{1 + e^{-(a(z))}}$
$\hat{Y}$

Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

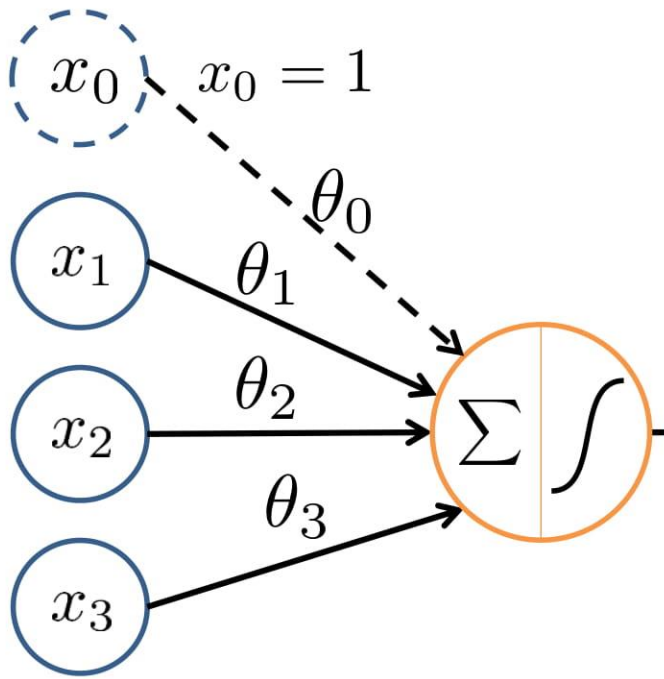
Neural networks



Layered feed-forward network

- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Neuron Model: Logistic Unit



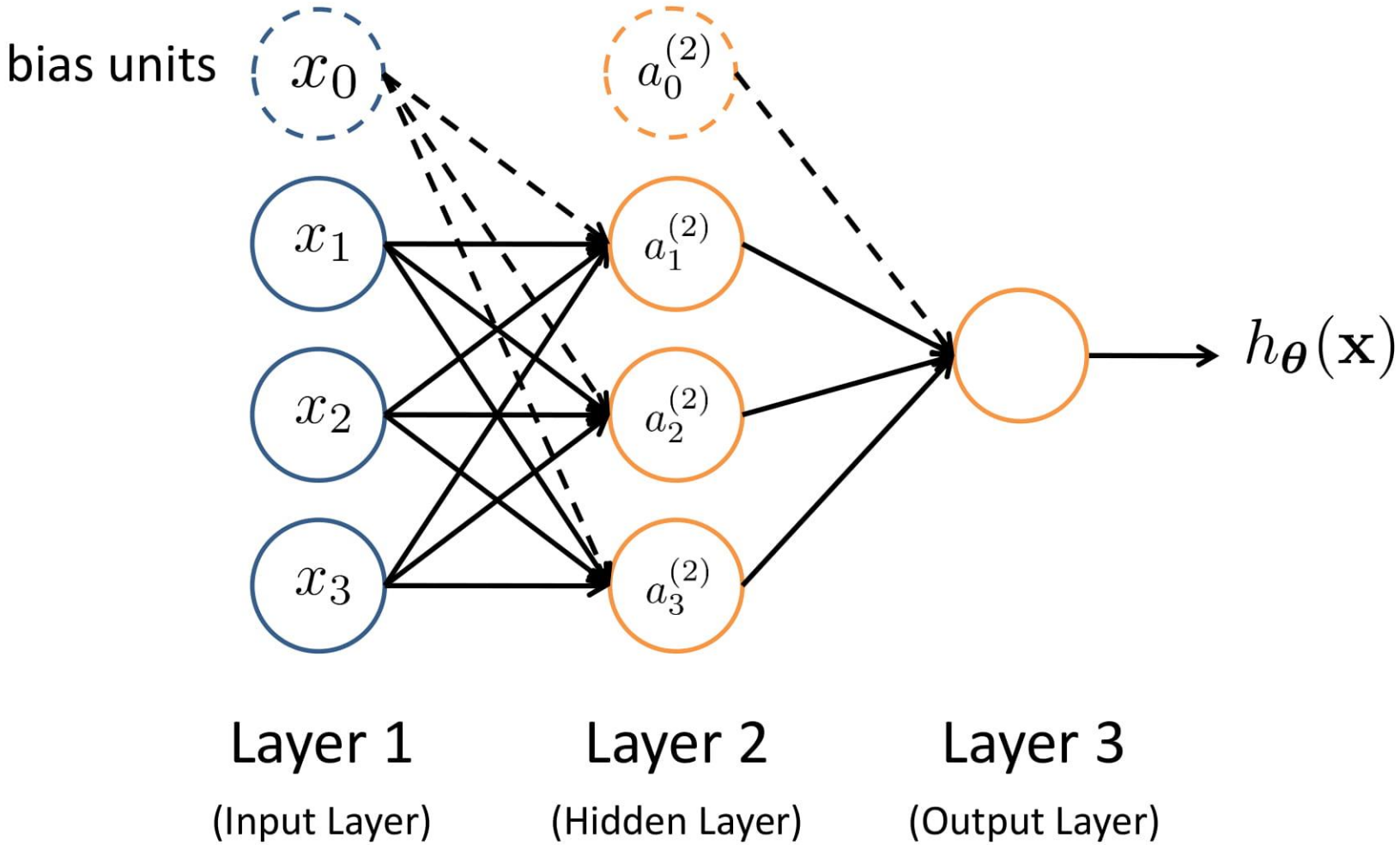
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

Based on slide by Andrew Ng

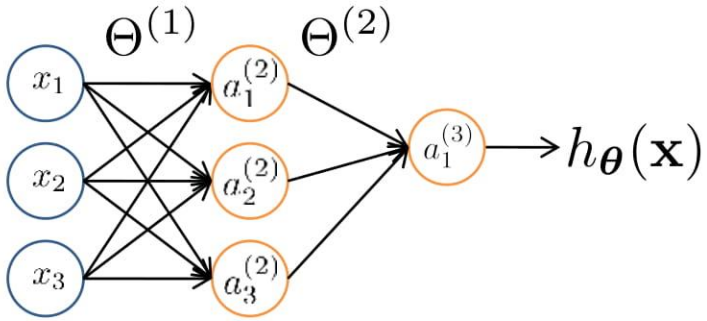
Neural Network



Feed-Forward Process

- Input layer units are set by some exterior function (think of these as **sensors**), which causes their output links to be **activated** at the specified level
- Working forward through the network, the **input function** of each unit is applied to compute the input value
  - Usually this is just the weighted sum of the activation on the links feeding into this node
- The **activation function** transforms this input function into a final value
  - Typically this is a **nonlinear** function, often a **sigmoid** function corresponding to the "threshold" of that node

Neural Network



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = weight matrix controlling function mapping from layer j to layer $j + 1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

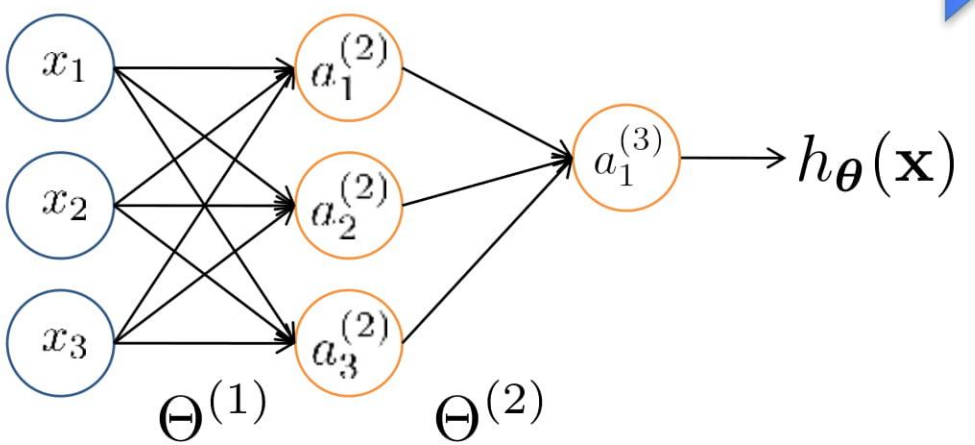
Vectorization

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g\left(z_3^{(2)}\right)$$

$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

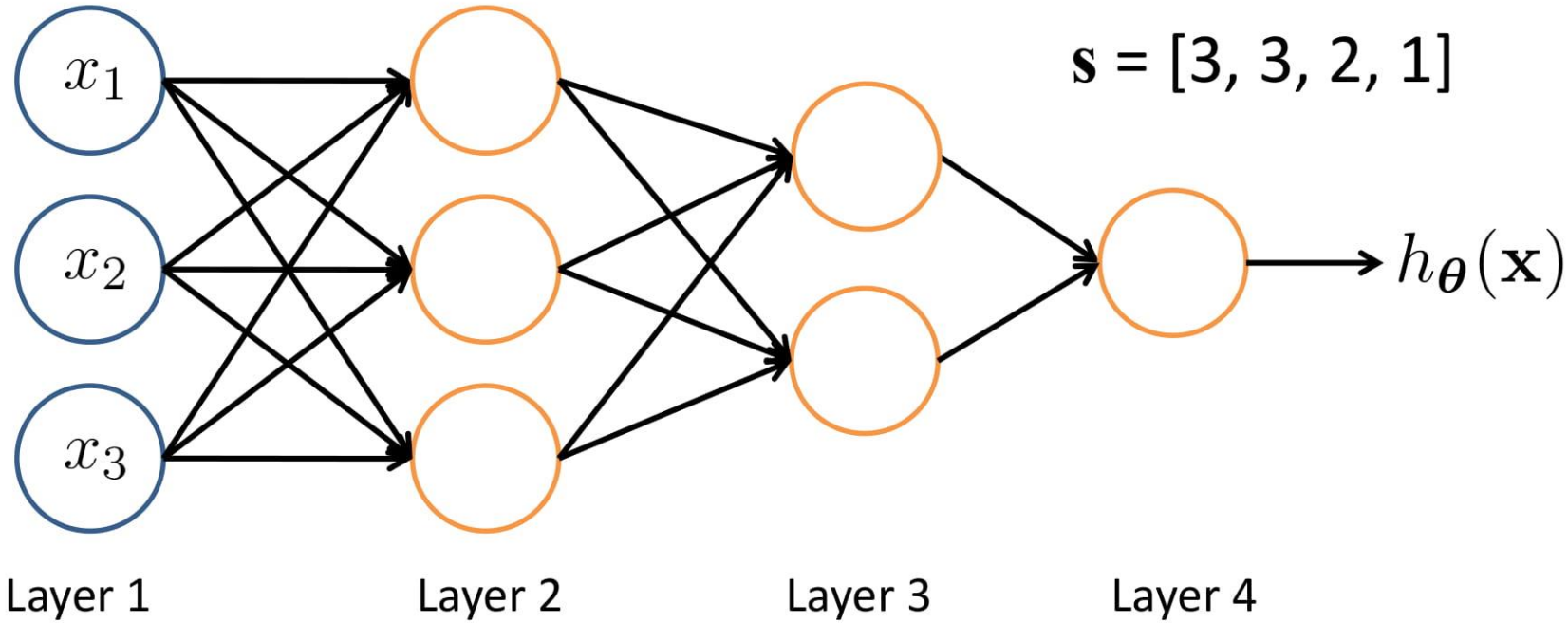
Add $a_0^{(2)} = 1$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Based on slide by Andrew Ng

Other Network Architectures



L denotes the number of layers

$\mathbf{s} \in \mathbb{N}^{+L}$ contains the numbers of nodes at each layer

- Not counting bias units
- Typically, $s_0 = d$ (# input features) and $s_{L-1} = K$ (# classes)

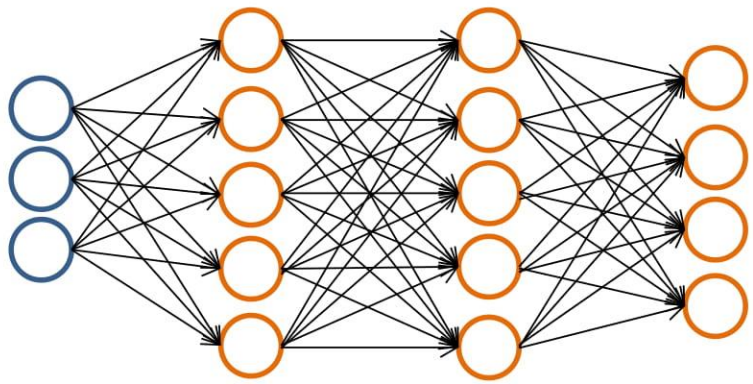
Multiple Output Units: One-vs-Rest

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

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

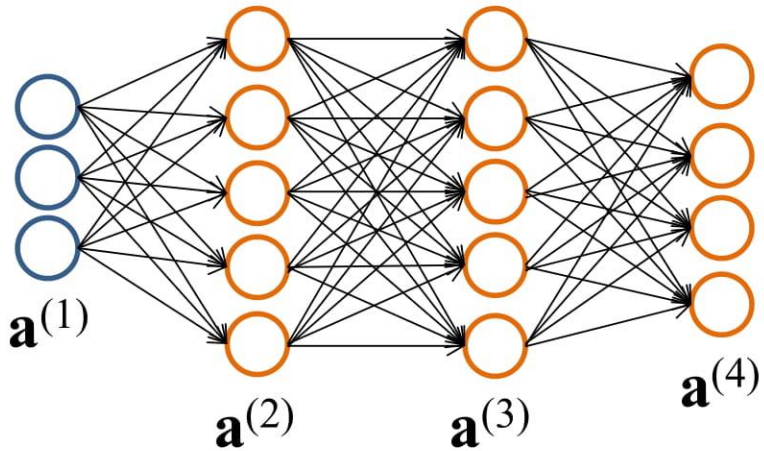
when motorcycle

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

when truck

Forward Propagation

- Given one labeled training instance $(\mathbf{x}, y)$:



Forward Propagation

- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $a_0^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $a_0^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = h_\Theta(\mathbf{x}) = g(\mathbf{z}^{(4)})$

Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

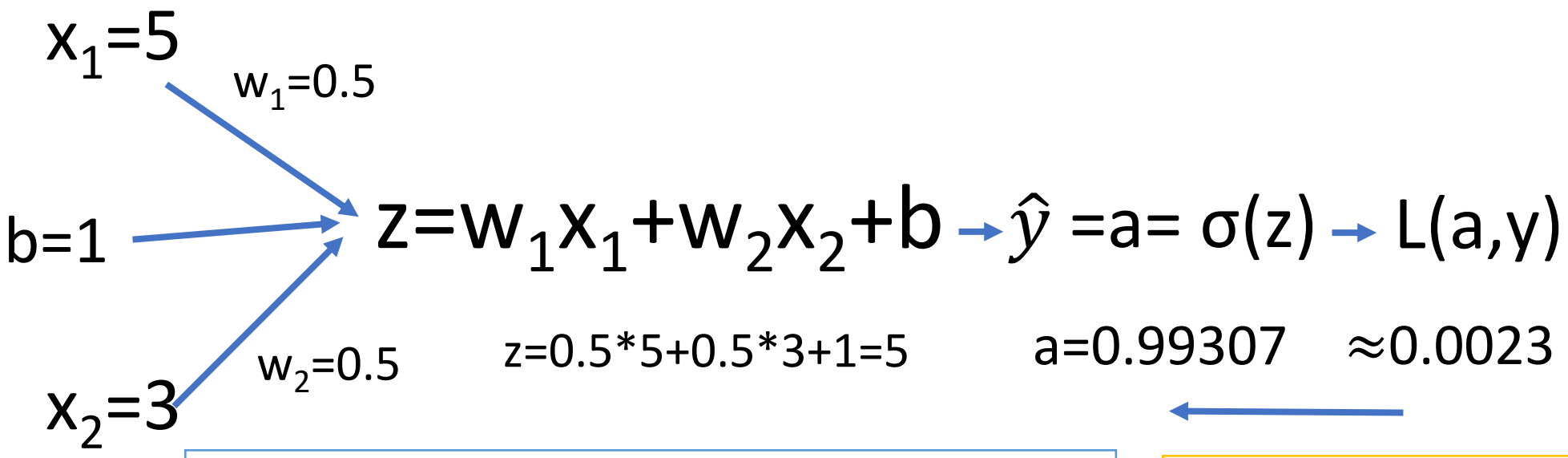
simultaneous update
for j = 0 ... d

For Linear Regression:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) &= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right)^2 \\
&= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) \times \frac{\partial}{\partial \theta_j} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) \\
&= \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) x_j^{(i)}
\end{aligned}$$

Derivatives in logistic regression

Original Value of y=1



dz=$\frac{dL(a,y)}{dz}$ = $\frac{dL}{da}\frac{da}{dz}$ = ($\frac{-y}{a}$ + $\frac{1-y}{1-a}$)*a(1-a)=a-y=-0.00693

da=$\frac{dL(a,y)}{da}$ = $\frac{-y}{a}$ + $\frac{1-y}{1-a}$ = -1.006978

dw_1= $\frac{dL(a,y)}{dw_1}$ =x_1.dz=5*-0.00695=-0.03475

dw_2= $\frac{dL(a,y)}{dw_2}$ =x2.dz=3*-0.00695=-0.02085

db=dz

$w_1=w_1-\alpha dw_1$ = 0.5-(0.01*-0.03475)=0.503475

α= 0.01

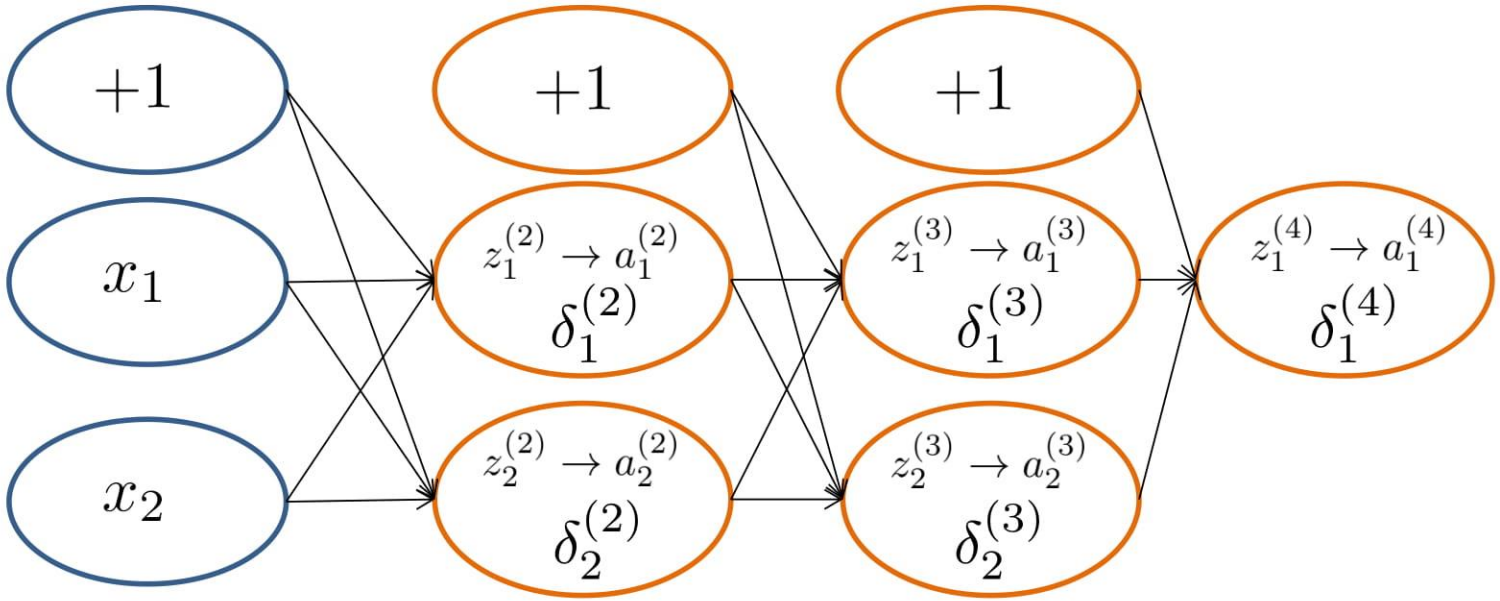
$w_2=w_2-\alpha dw_2$ = 0.5-(0.01*-0.02085)=0.502085

b=b- αdb =1-(0.01)*(-0.00693)=1.000693

Backpropagation Intuition

- Each hidden node j is "responsible" for some fraction of the error $\delta_j^{(l)}$ in each of the output nodes to which it connects
- $\delta_j^{(l)}$ is divided according to the strength of the connection between hidden node and the output node
- Then, the "blame" is propagated back to provide the error values for the hidden layer

Backpropagation Intuition



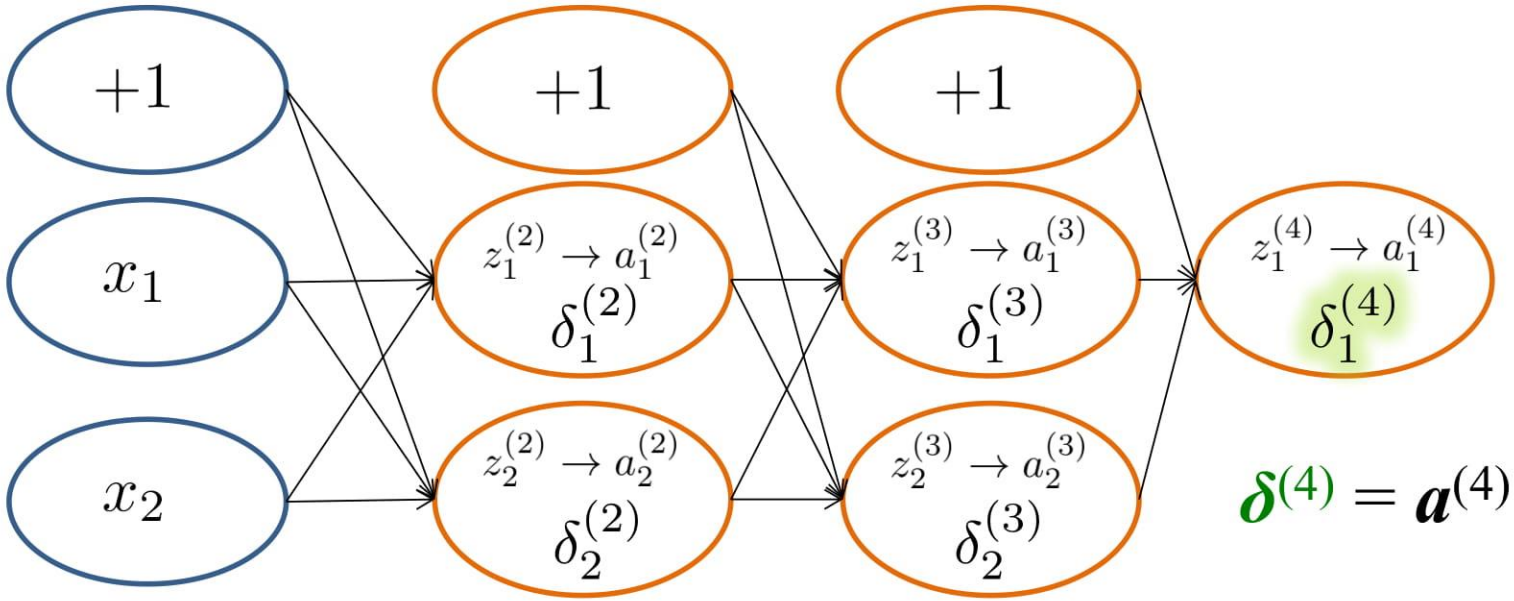
$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

$$\text{where } \text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$$

Based on slide by Andrew Ng

Backpropagation Intuition



$$\boldsymbol{\delta}^{(4)} = \boldsymbol{a}^{(4)} - \mathbf{y}$$

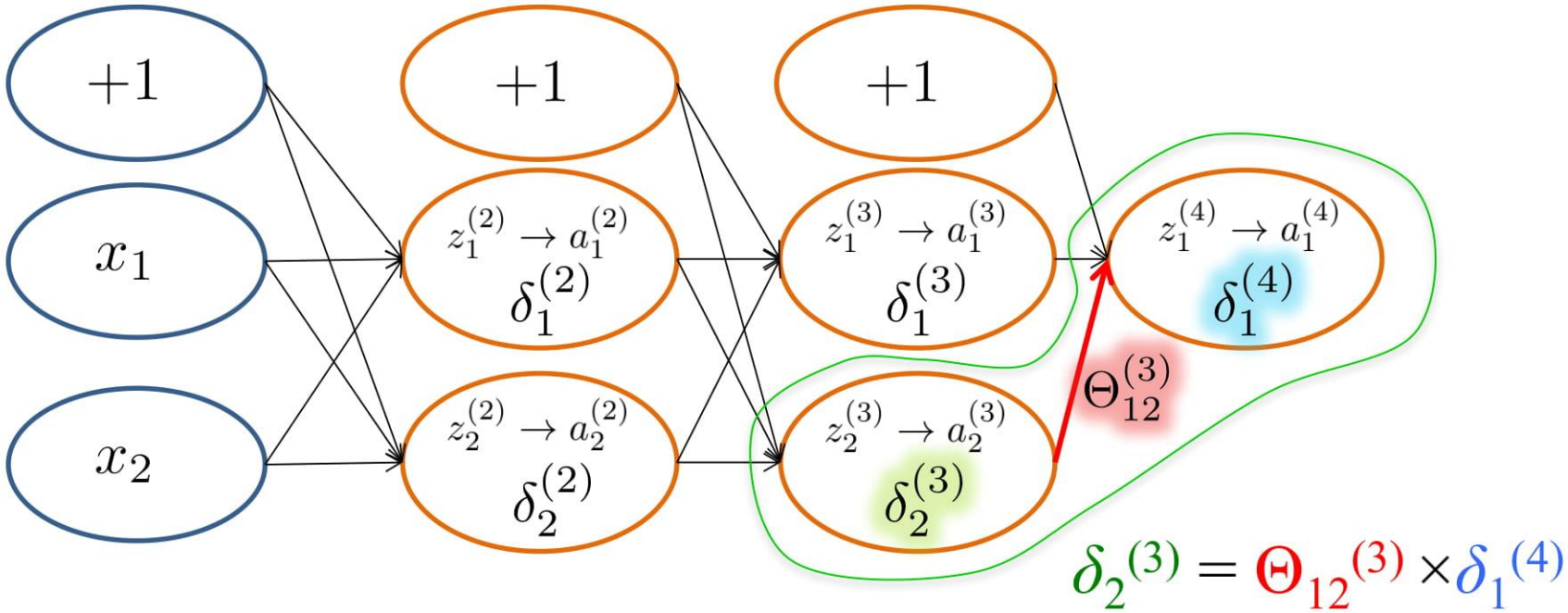
$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

$$\text{where } \text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$$

Based on slide by Andrew Ng

Backpropagation Intuition



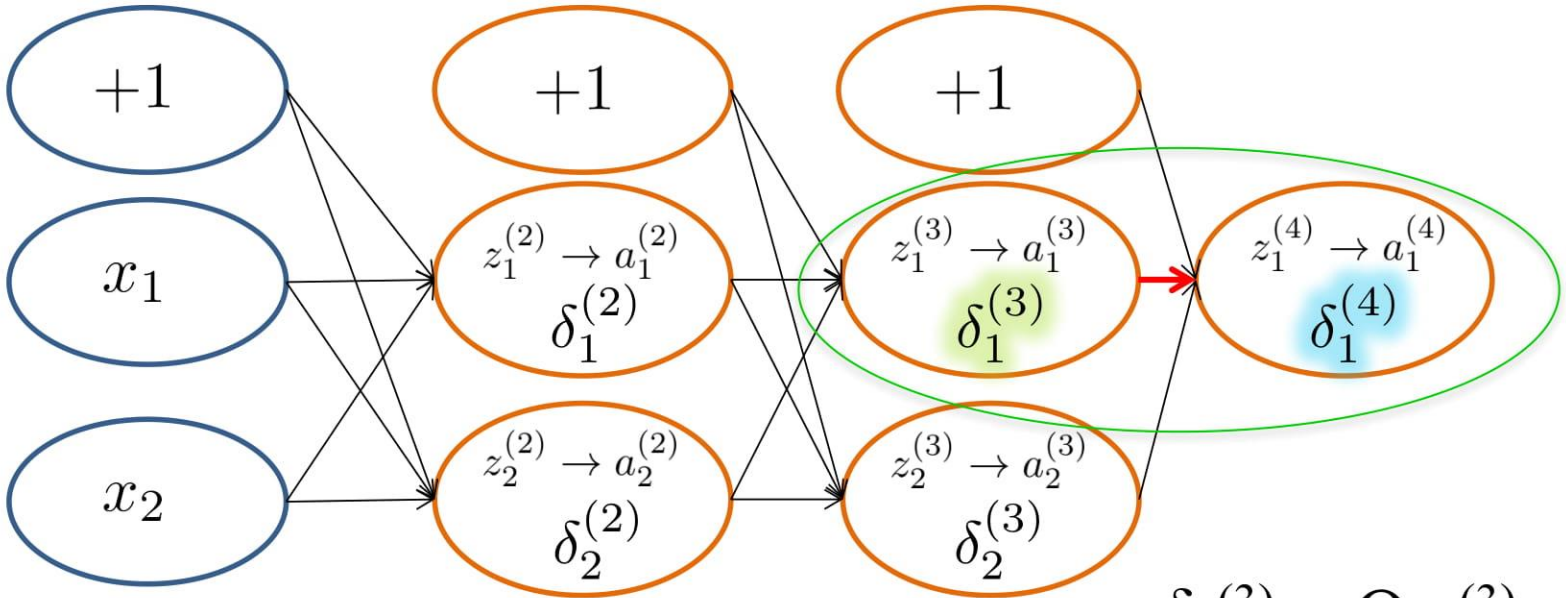
$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

$$\text{where } \text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$$

Based on slide by Andrew Ng

Backpropagation Intuition



$$\delta_2^{(3)} = \Theta_{12}^{(3)} \times \delta_1^{(4)}$$

$$\delta_1^{(3)} = \Theta_{11}^{(3)} \times \delta_1^{(4)}$$

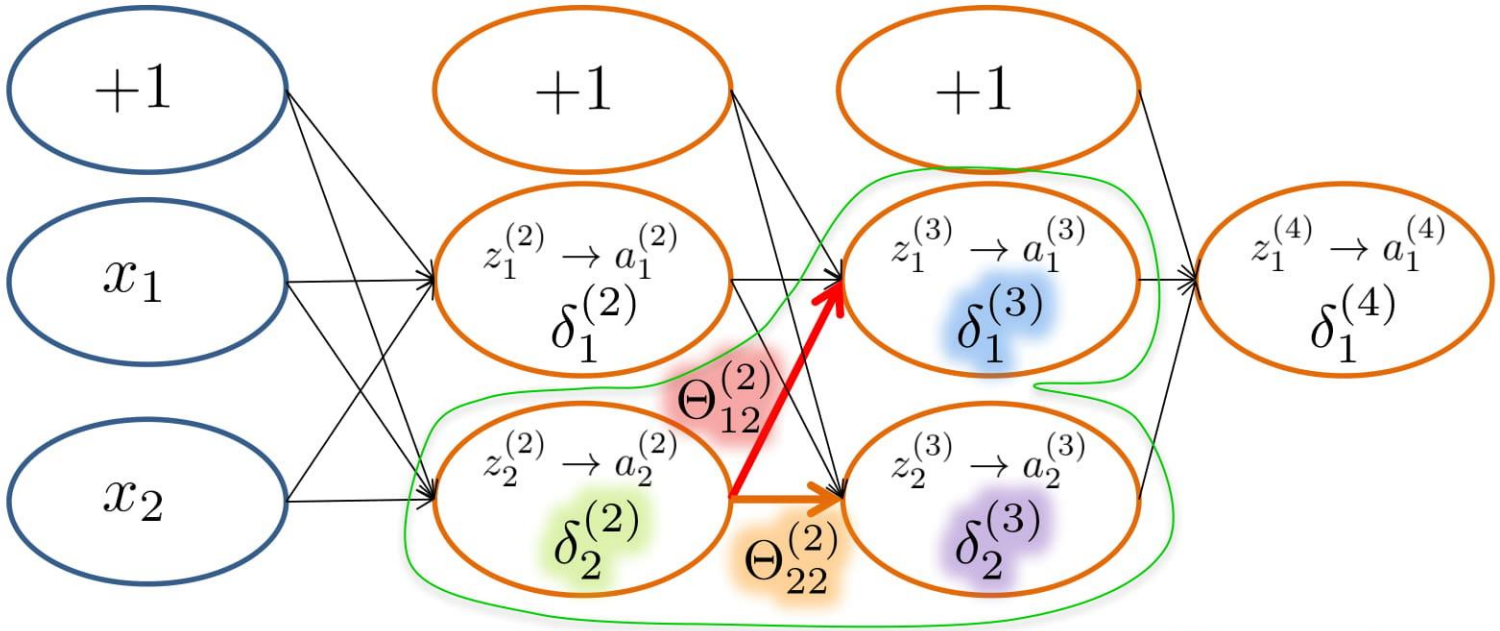
$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

$$\text{where } \text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$$

Based on slide by Andrew Ng

Backpropagation Intuition



$$\delta_2^{(2)} = \Theta_{12}^{(2)} \times \delta_1^{(3)} + \Theta_{22}^{(2)} \times \delta_2^{(3)}$$

$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

$$\text{where } \text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$$

Based on slide by Andrew Ng

Backprop Issues

“Backprop is the cockroach of machine learning. It’s ugly, and annoying, but you just can’t get rid of it.”

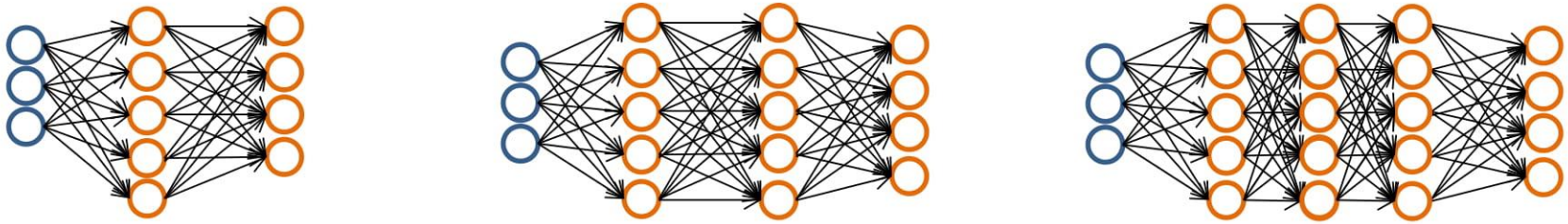
–Geoff Hinton

Problems:

- black box
- local minima

Training a Neural Network

Pick a network architecture (connectivity pattern between nodes)



- # input units = # of features in dataset
- # output units = # classes

Reasonable default: 1 hidden layer

- or if >1 hidden layer, have same # hidden units in every layer (usually the more the better)

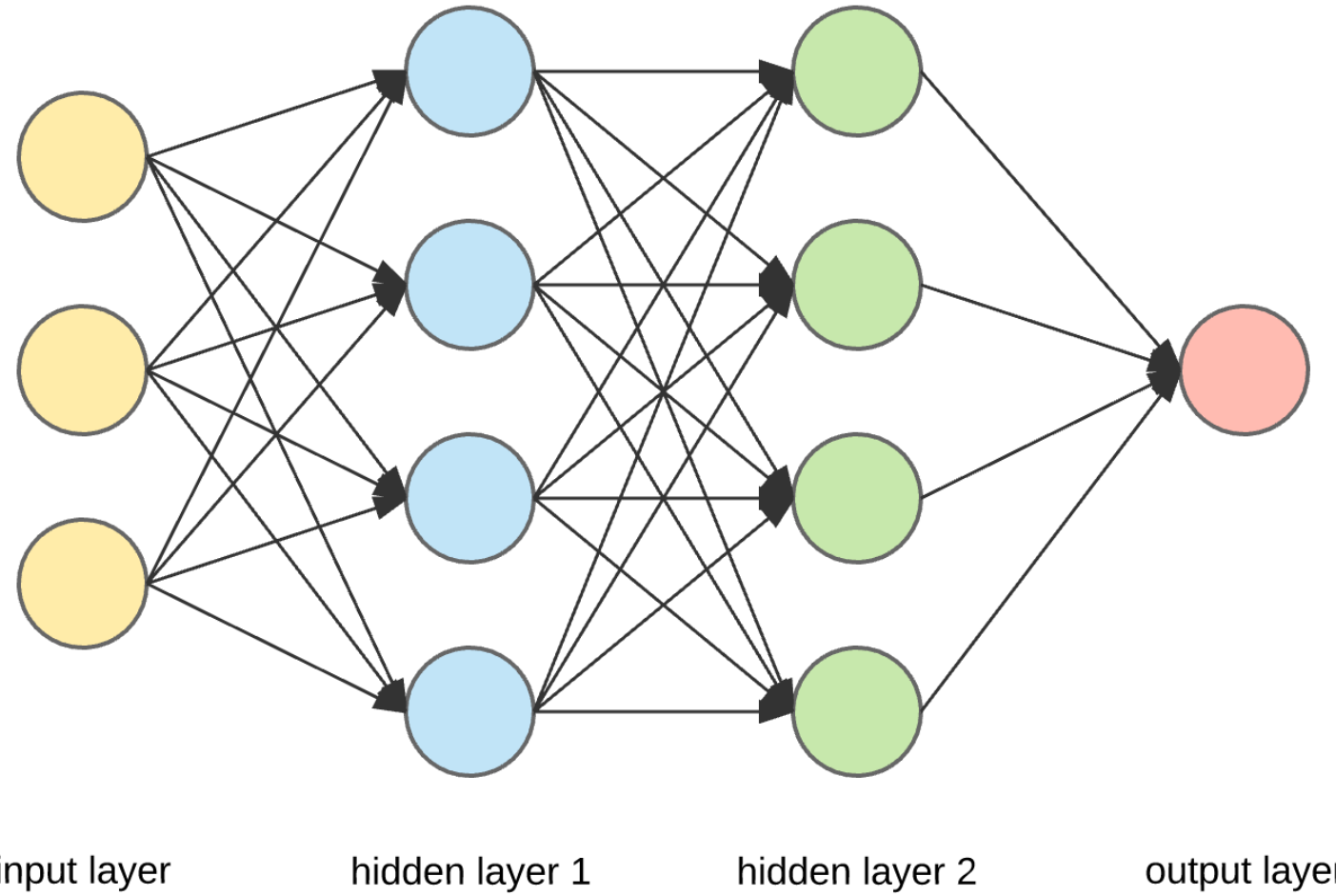
Based on slide by Andrew Ng

Training a Neural Network

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(\mathbf{x}_i)$ for any instance $\mathbf{x}_i$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$
5. Use gradient descent with backprop to fit the network

Based on slide by Andrew Ng

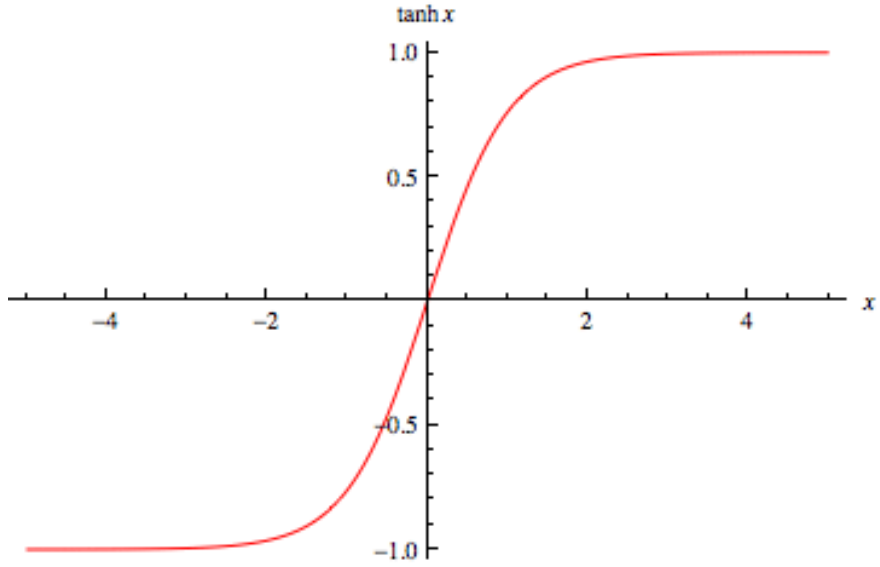
Hidden Activations



Activation Functions: tanh

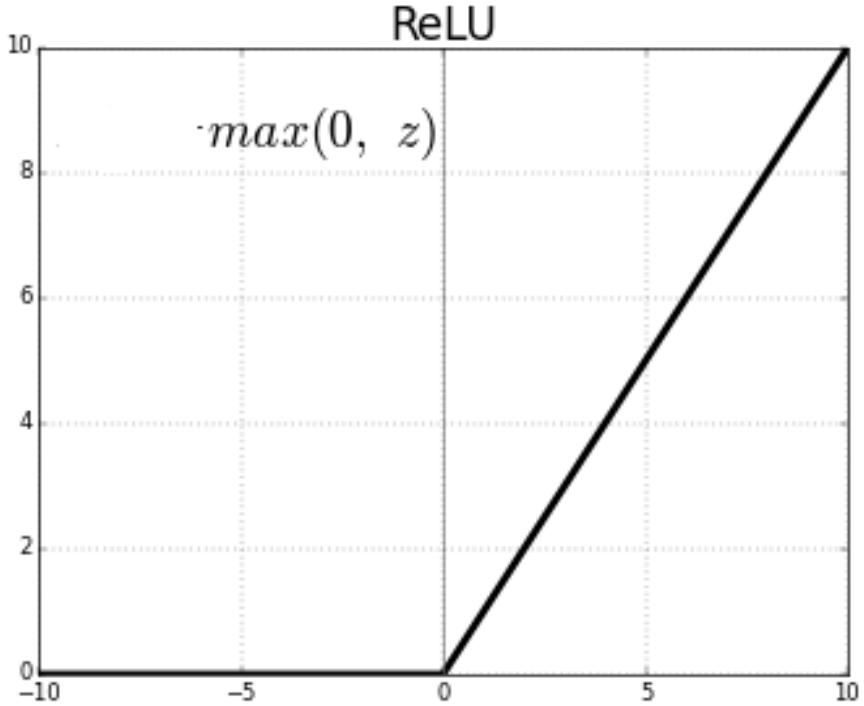
Tanh activation function is more preferred than the sigmoid function.
It is the shifted version of sigmoid function with a mean value of zero.
It has better centering effect for the activation function to be used on the hidden layer.
For binary classification problem at the output layer we use the Sigmoid function.

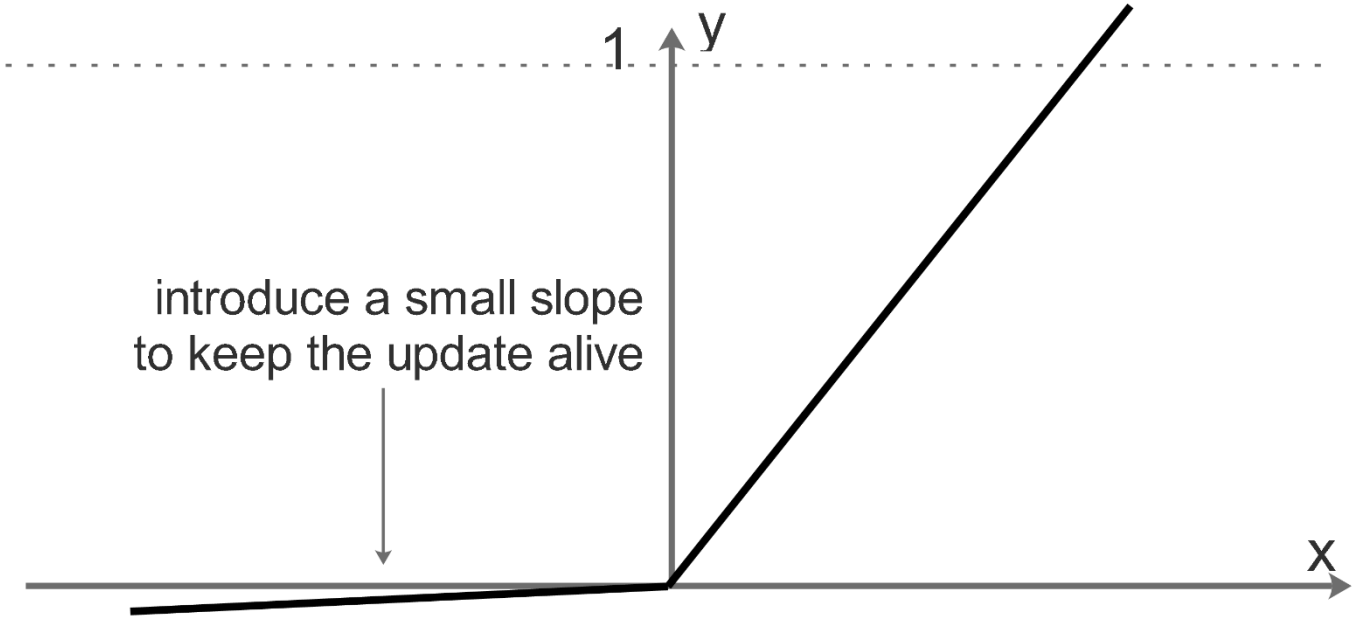
$$\tanh = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



Problem in both Sigmoid and tanh is that the slope of the curve except the middle region is too small and goes close to zero. This create a serious issue with gradient descent and learning becomes very slow.

ReLU and Leaky ReLU

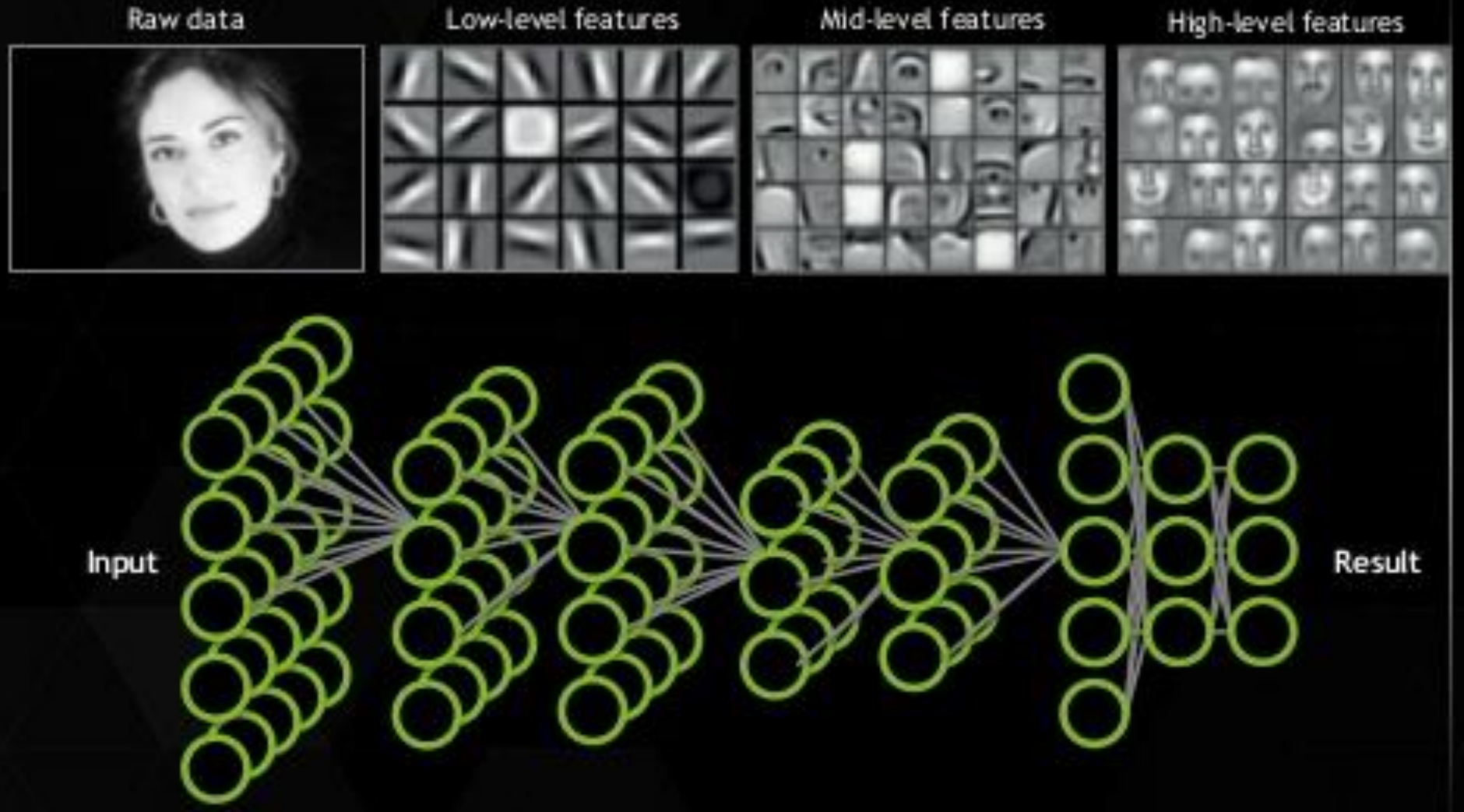




Leaky ReLU max(0.1z,z)

Rectified Linear Unit is the default activation function being used now. If you see the left part of the function, you can see that it is not zero, but almost zero. To resolve the issue of dead neurons people use Leaky ReLU

DEEP NEURAL NETWORK (DNN)



Application components:

Task objective
- e.g. Identify face

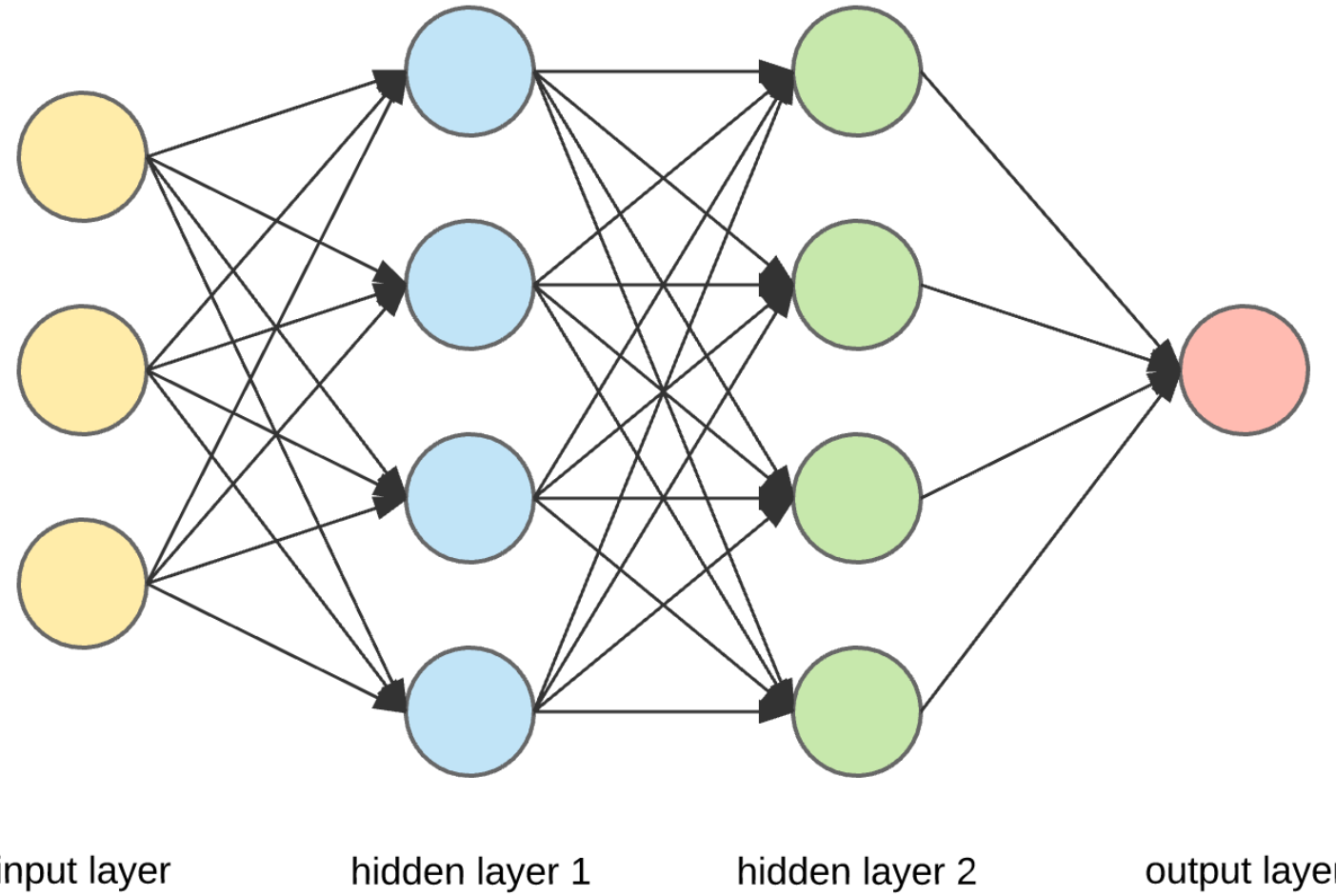
Training data
- 10-100M images

Network architecture
- ~10 layers
- 1B parameters

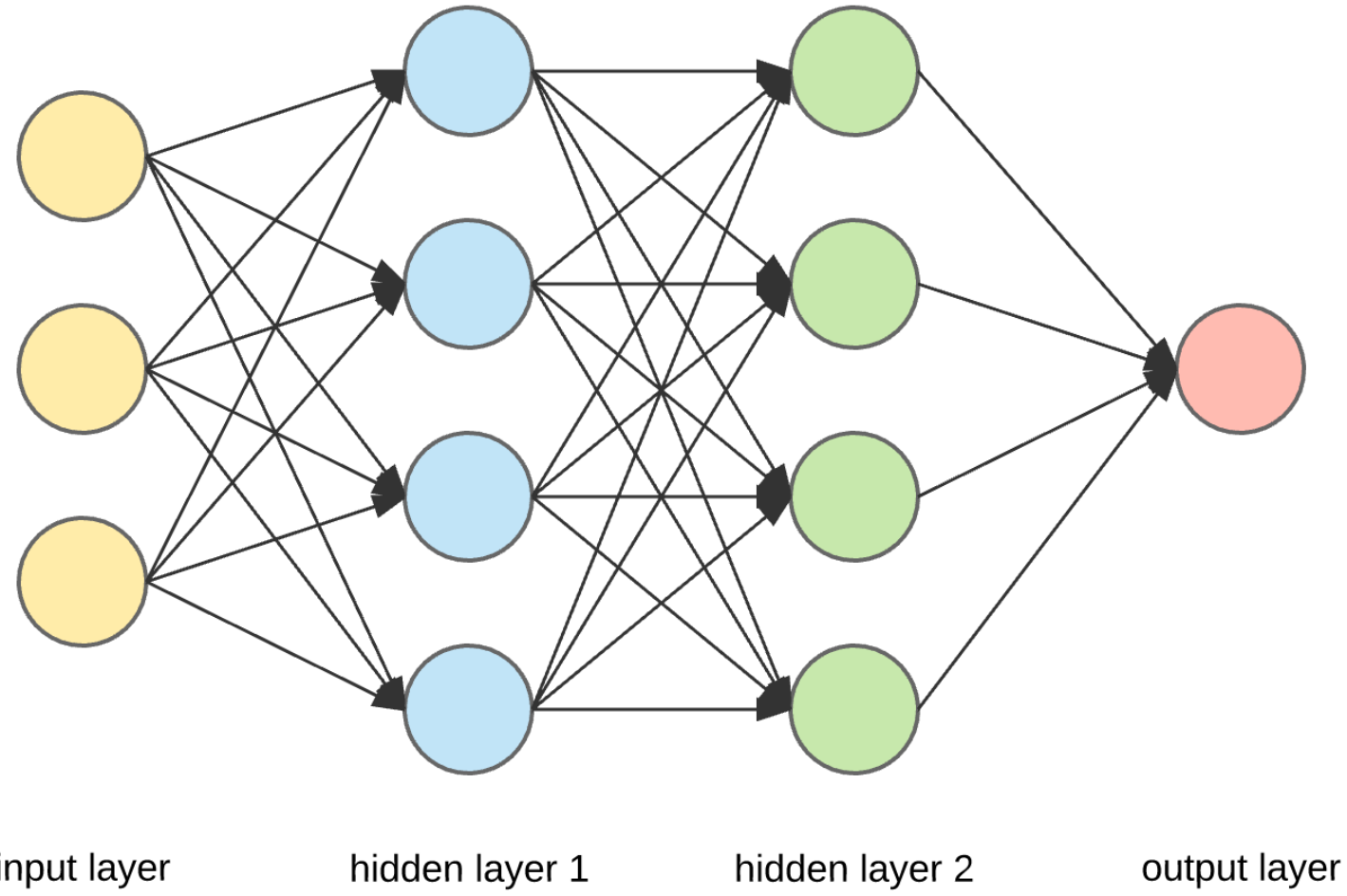
Learning algorithm
- ~30 Exaflops
- ~30 GPU days

Standing near elephant

Neural Network for Classification



Neural Network for Regression



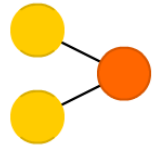
A mostly complete chart of

Neural Networks

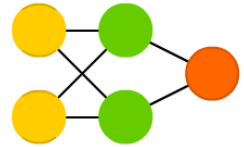
 asimovinstitute.org

- Input Cell
- Backfed Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Capsule Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Gated Memory Cell
- Kernel
- Convolution or Pool

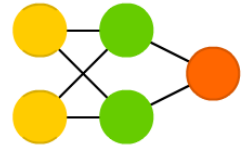
Perceptron (P)



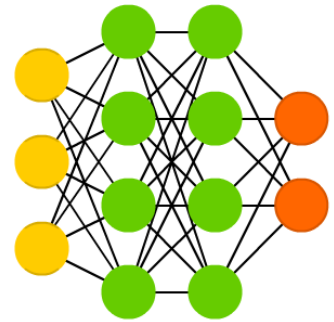
Feed Forward (FF)



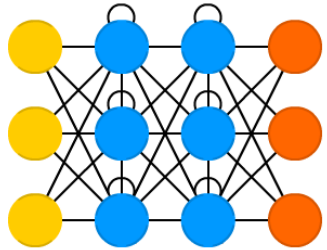
Radial Basis Network (RBF)



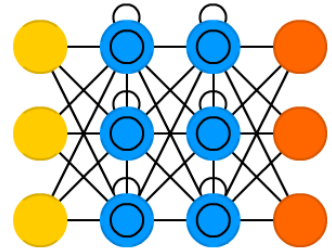
Deep Feed Forward (DFF)



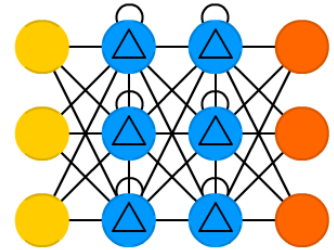
Recurrent Neural Network (RNN)



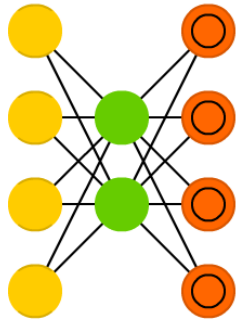
Long / Short Term Memory (LSTM)



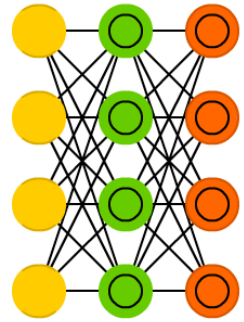
Gated Recurrent Unit (GRU)



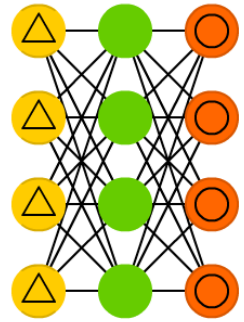
Auto Encoder (AE)



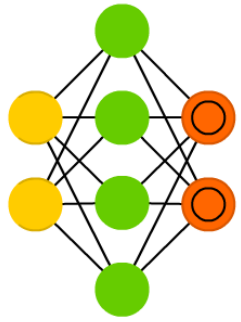
Variational AE (VAE)



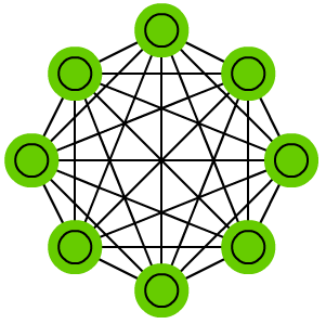
Denoising AE (DAE)



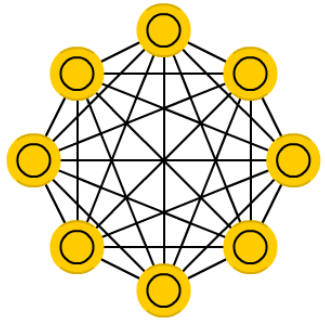
Sparse AE (SAE)



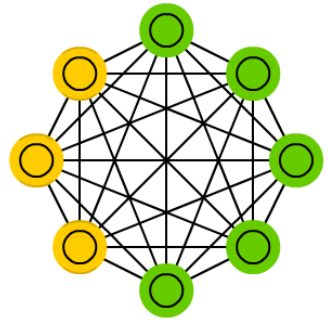
Markov Chain (MC)



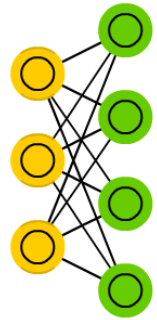
Hopfield Network (HN)



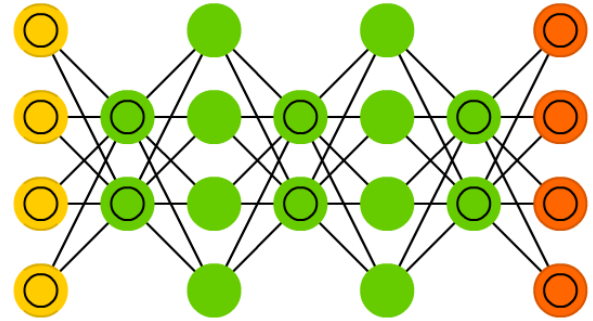
Boltzmann Machine (BM)



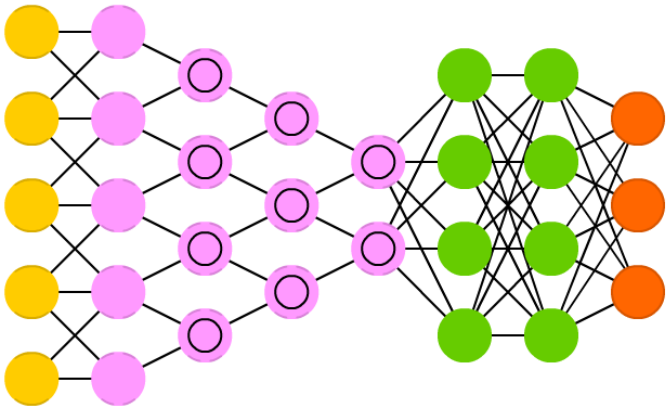
Restricted BM (RBM)



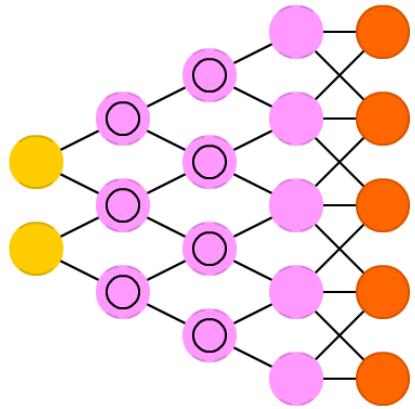
Deep Belief Network (DBN)



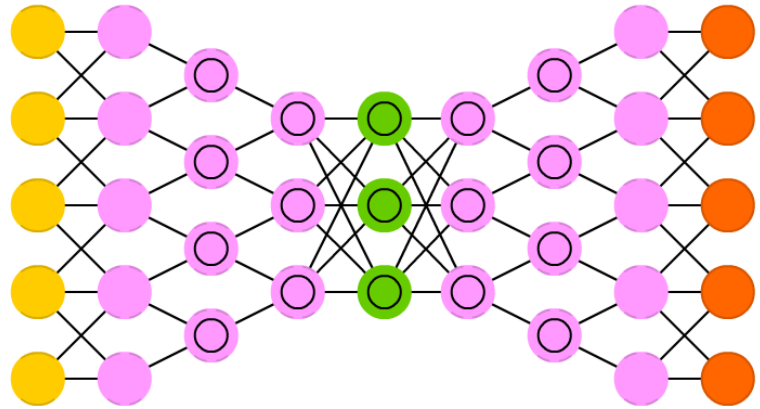
Deep Convolutional Network (DCN)



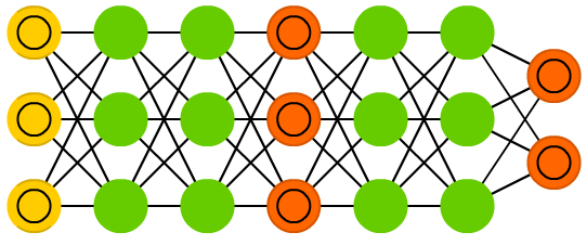
Deconvolutional Network (DN)



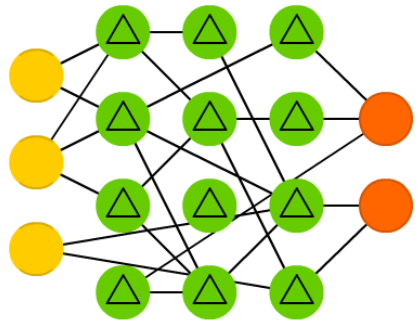
Deep Convolutional Inverse Graphics Network (DCIGN)



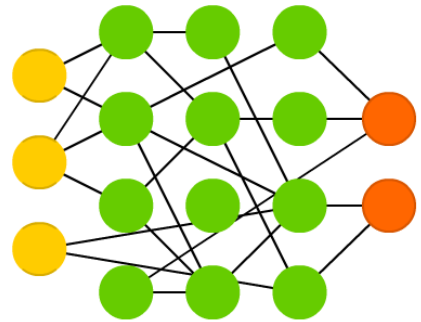
Generative Adversarial Network (GAN)



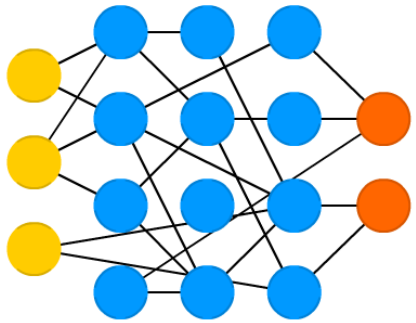
Liquid State Machine (LSM)



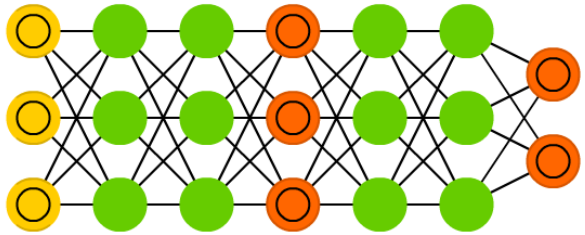
Extreme Learning Machine (ELM)



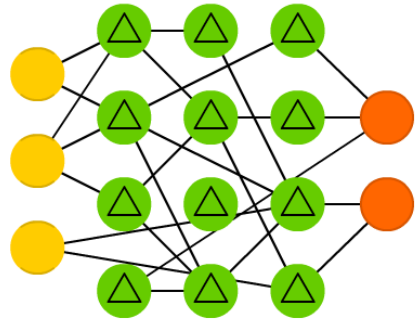
Echo State Network (ESN)



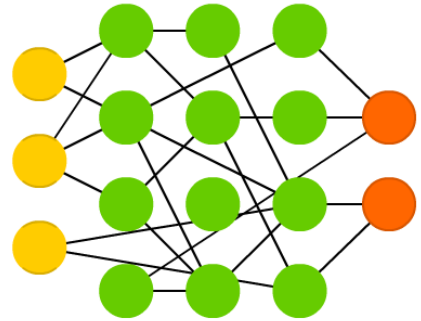
Generative Adversarial Network (GAN)



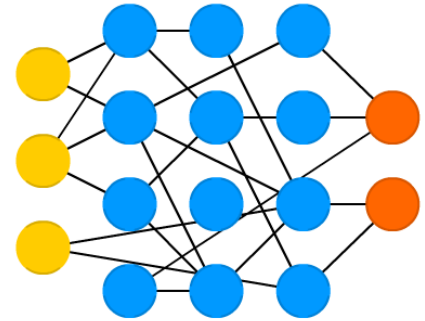
Liquid State Machine (LSM)



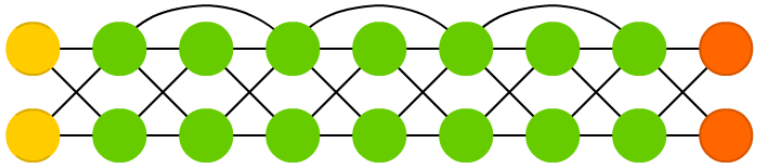
Extreme Learning Machine (ELM)



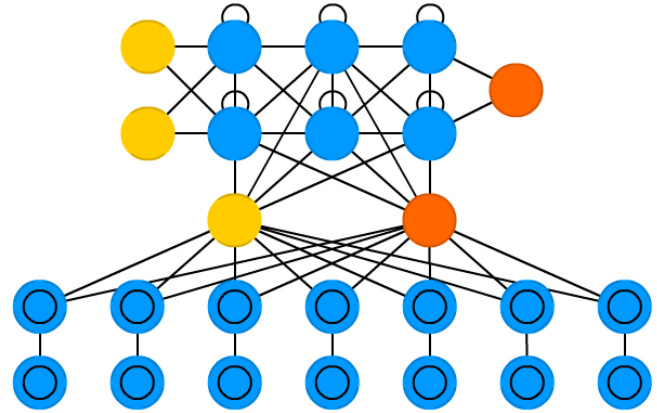
Echo State Network (ESN)



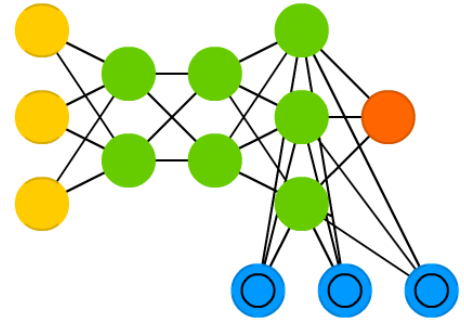
Deep Residual Network (DRN)



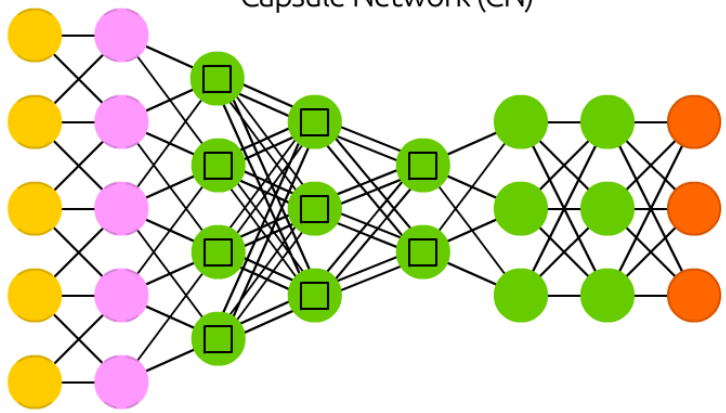
Differentiable Neural Computer (DNC)



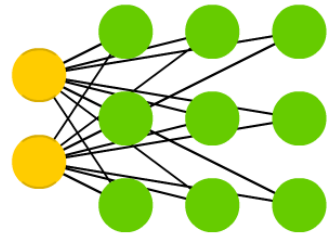
Neural Turing Machine (NTM)



Capsule Network (CN)



Kohonen Network (KN)



Attention Network (AN)