Neural network

Ngô Minh Nhựt

2024

Model representation and forward propagation

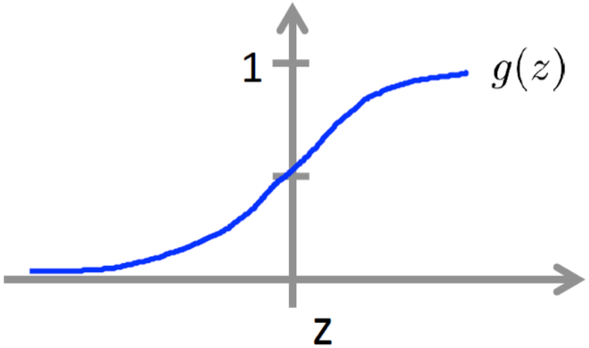
Part 1

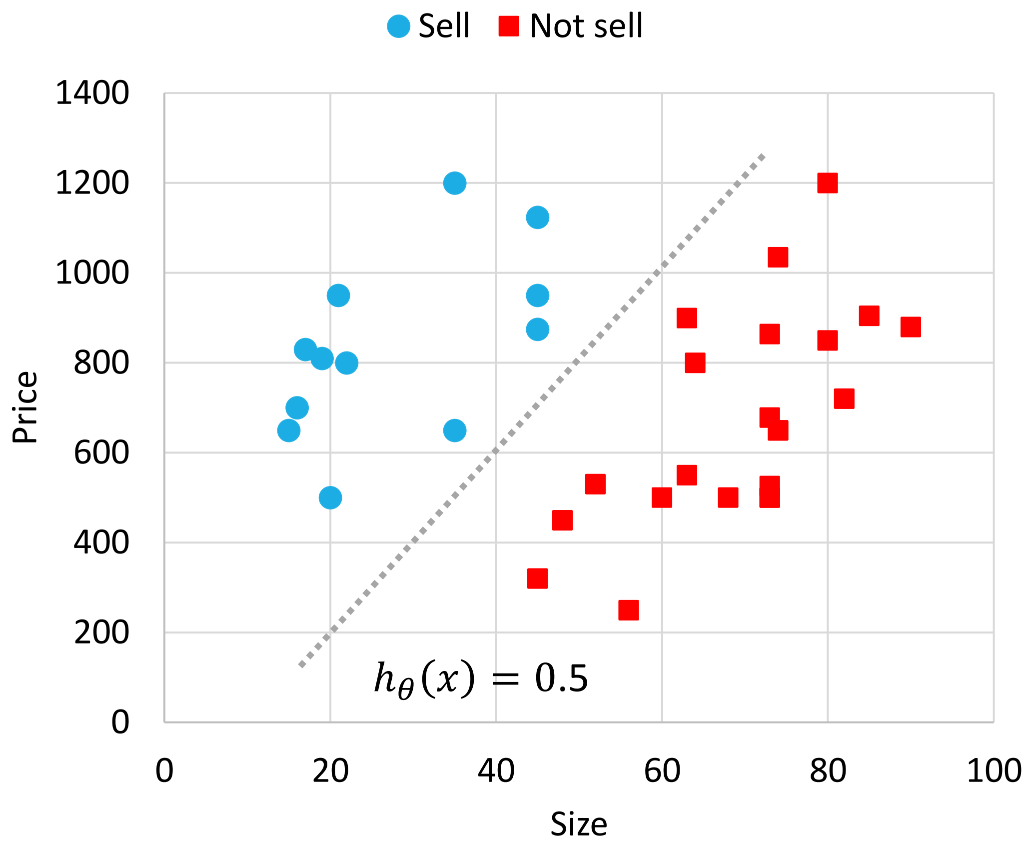
Logistic regression

- Linear classifier

$$h_\theta(x) = g(\theta^T x)$$

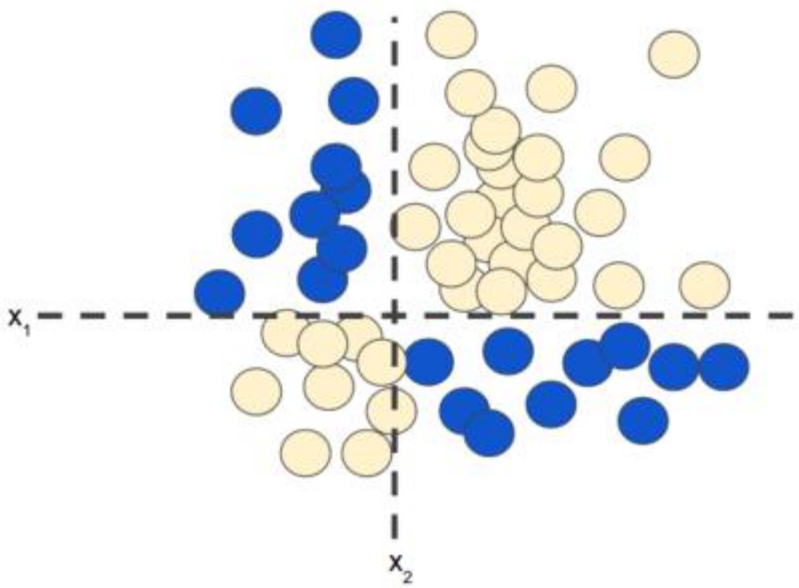
$$g(z) = \frac{1}{1+e^{-z}}$$





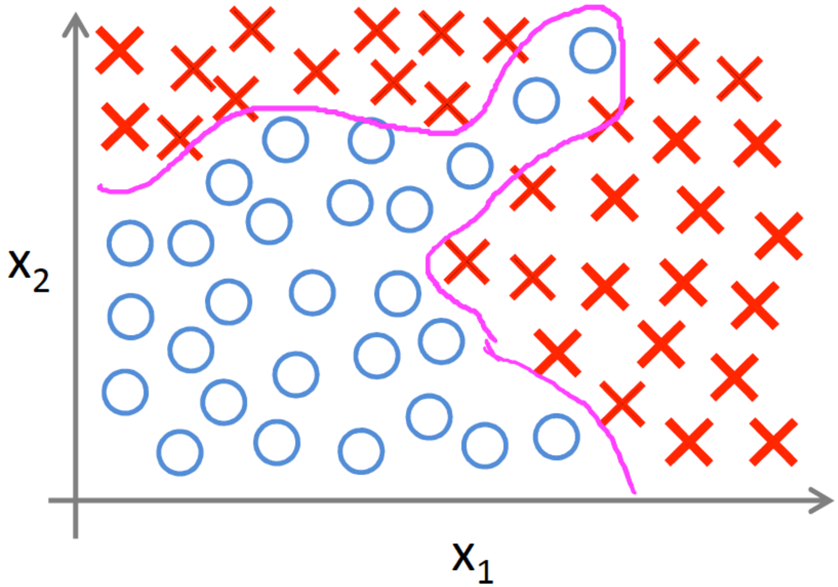
Non-linear classifier

- Non-linearity: data samples cannot be separated by a hyperplane



Source: Google Crash Course

Non-linear classifier



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_2 x_2^2 + \ldots)$$

Feature mapping

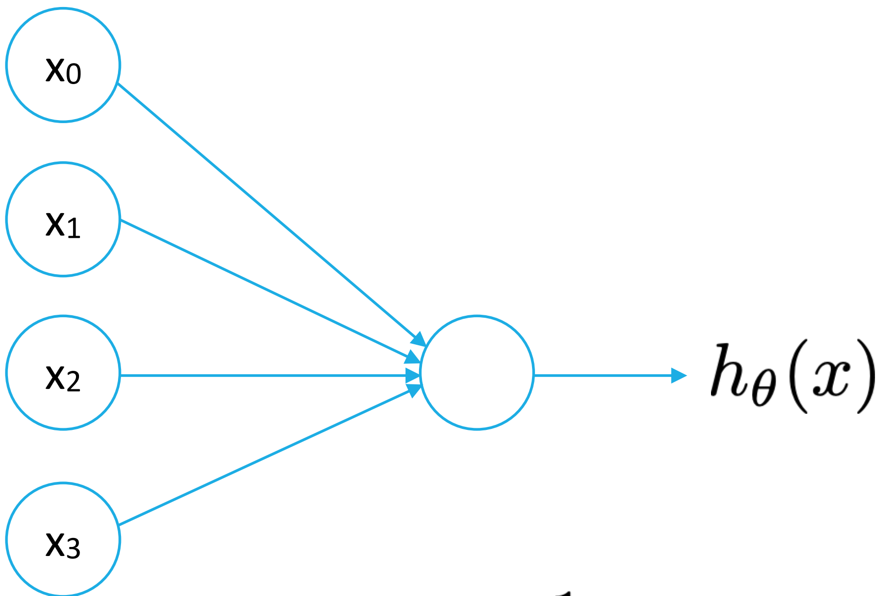
x_1 = size
x_2 = number of bedrooms
x_3 = number of floors
x_4 = age
…
x_{100}

What mappings?
How many mappings?
How many features?

Non-linear classification with logistic regression requires a lot of features

Source: Andrew Ng

Logistic regression

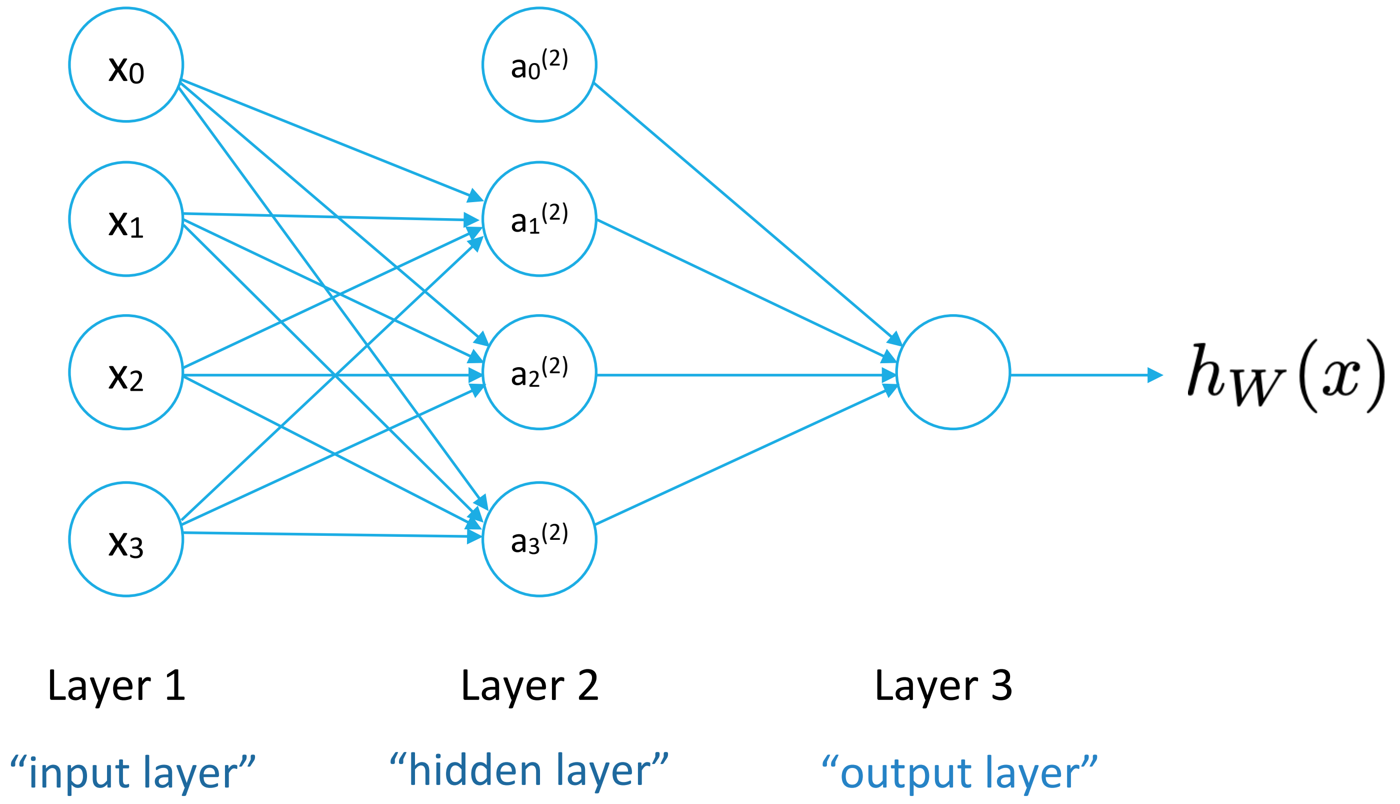


$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

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

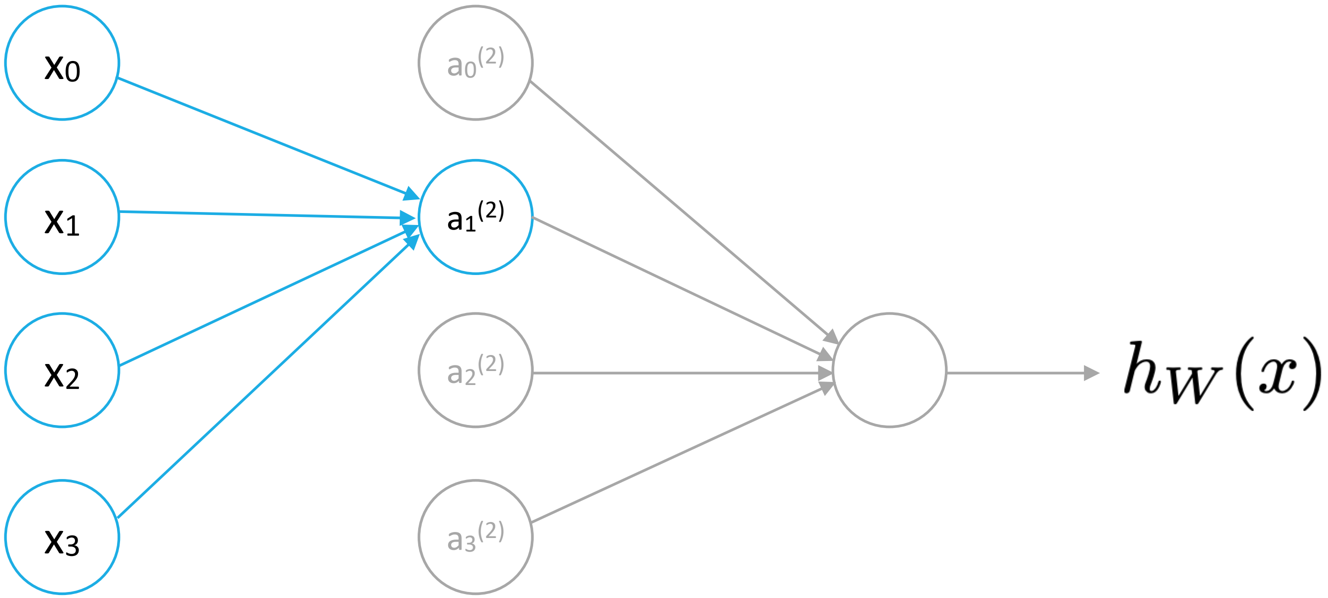
Sigmoid activation function

Neural network



Activation at node

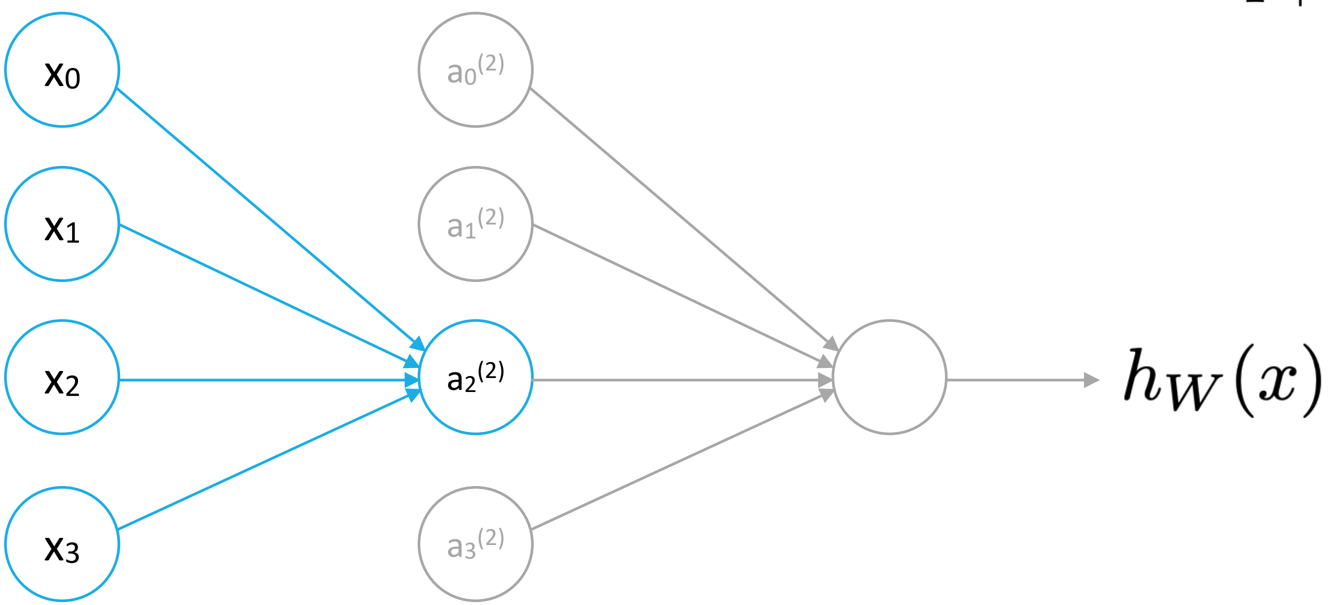
$$g(z) = \frac{1}{1 + e^{-z}}$$



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

Activation at node

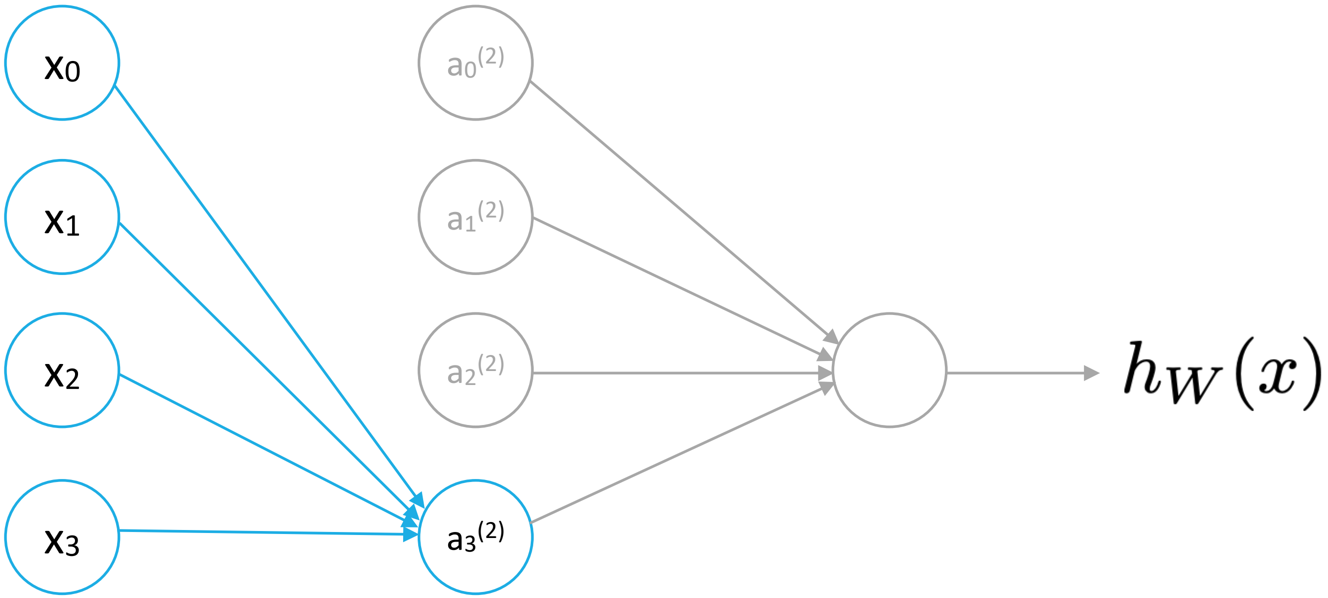
$$g(z) = \frac{1}{1 + e^{-z}}$$



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

Activation at node

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

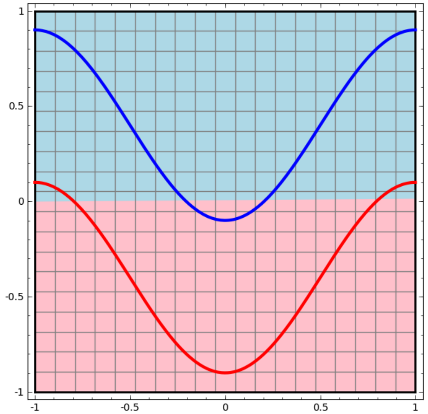


$$a_3^{(2)} = g\left(W_{30}^{(1)}x_0 + W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3\right)$$

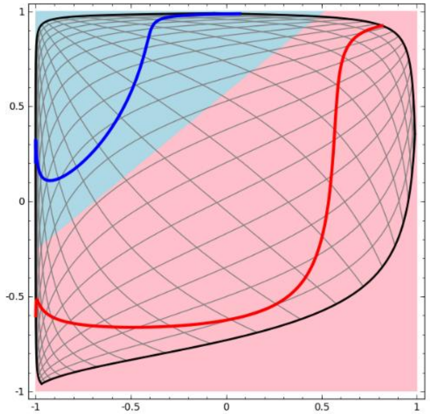
Activation at node

Why does neural network need activation?

- Non-linearity of activation functions help transform data into a new representation
- Data samples in the new representation are expected to be linearly separated



Original



New representation

Without non-linear transformation, neural network works as a linear model.

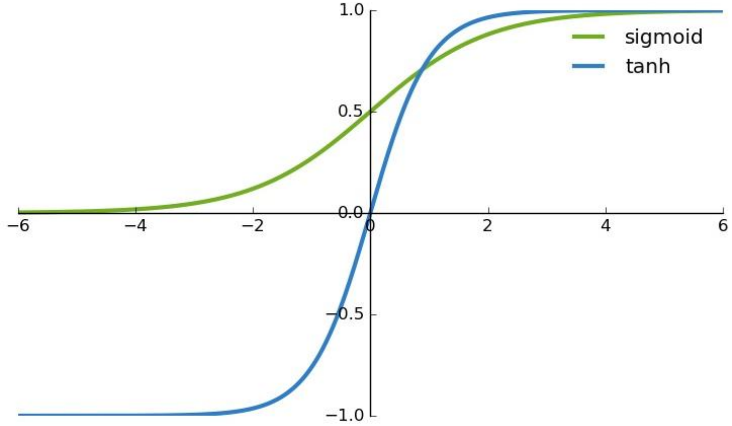
Source: https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

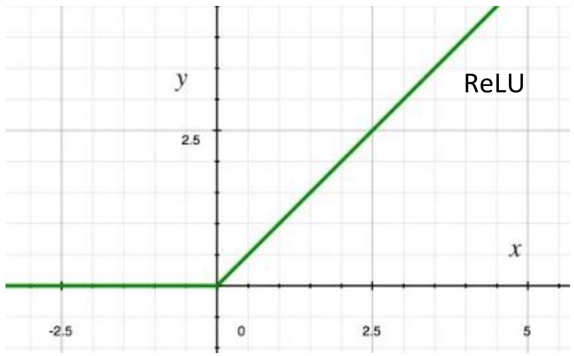
Common activation functions

- Sigmoid

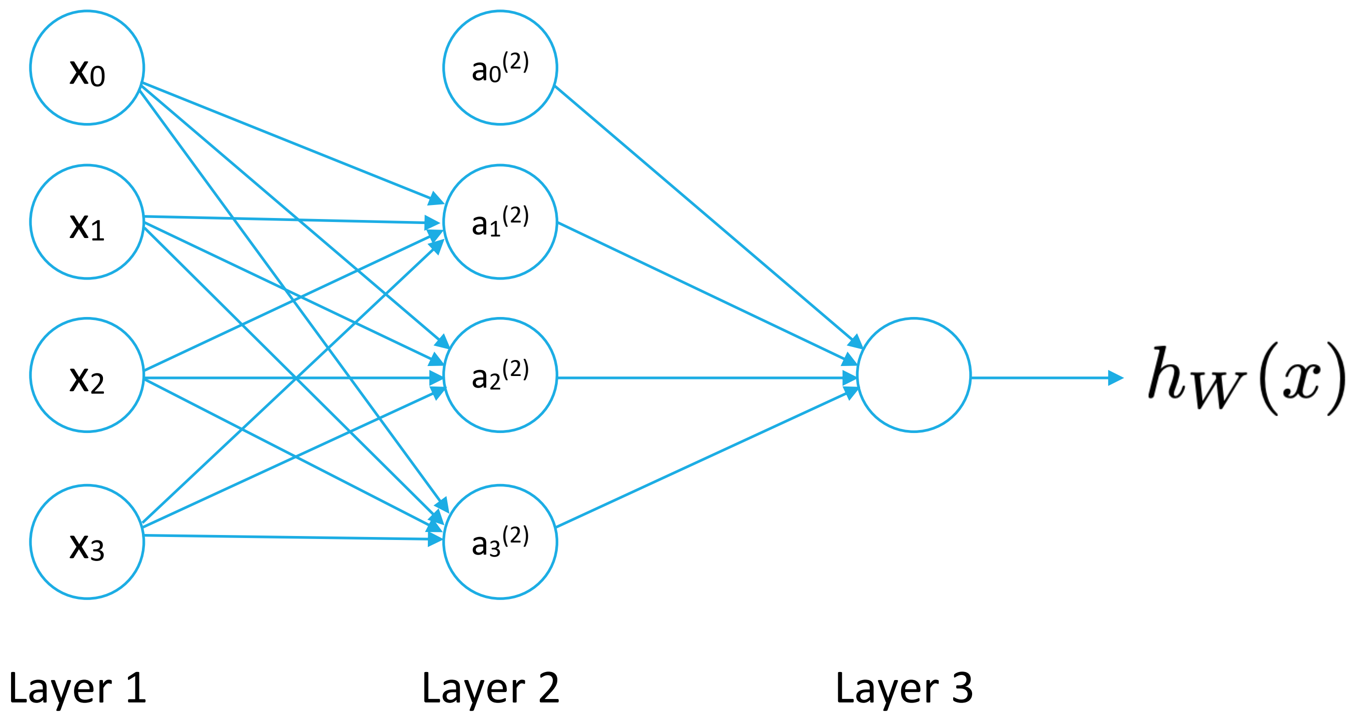
$$g(z) = \frac{1}{1 + e^{-z}}$$

- Tanh
  - $\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$
  - Derivative: $1 - \tanh^2(z)$
- ReLU (rectified linear unit)
  - $g(z) = \max(0, z)$





Weight matrix



$a_i^{(j)}$: activation of i^{th} node in j^{th} layer

$W^{(j)}$: weight matrix mapping activation in j^{th} layer to $j+1^{th}$ layer

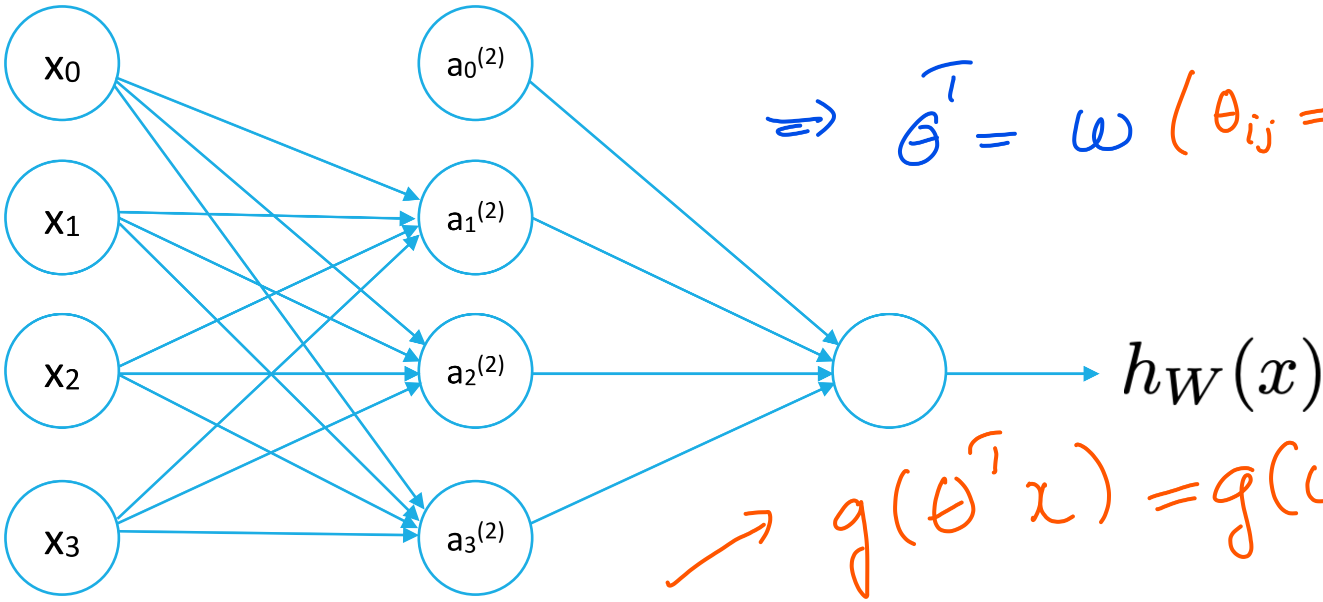
If neural network has s_j nodes in j^{th} layer and s_{j+1} nodes in $j+1^{th}$ layer, $W^{(j)}$ has a size of $s_{j+1} \times (s_j + 1)$

gọi θ_{ij} là trọng số của cạnh i nối với j

$$\theta = \begin{bmatrix} \theta_{01} & \theta_{02} & \theta_{03} \\ \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \\ \theta_{31} & \theta_{32} & \theta_{33} \end{bmatrix}$$

Forward propagation

$$\Rightarrow \theta^T = \omega \quad (\theta_{ij} = \omega_{ji})$$

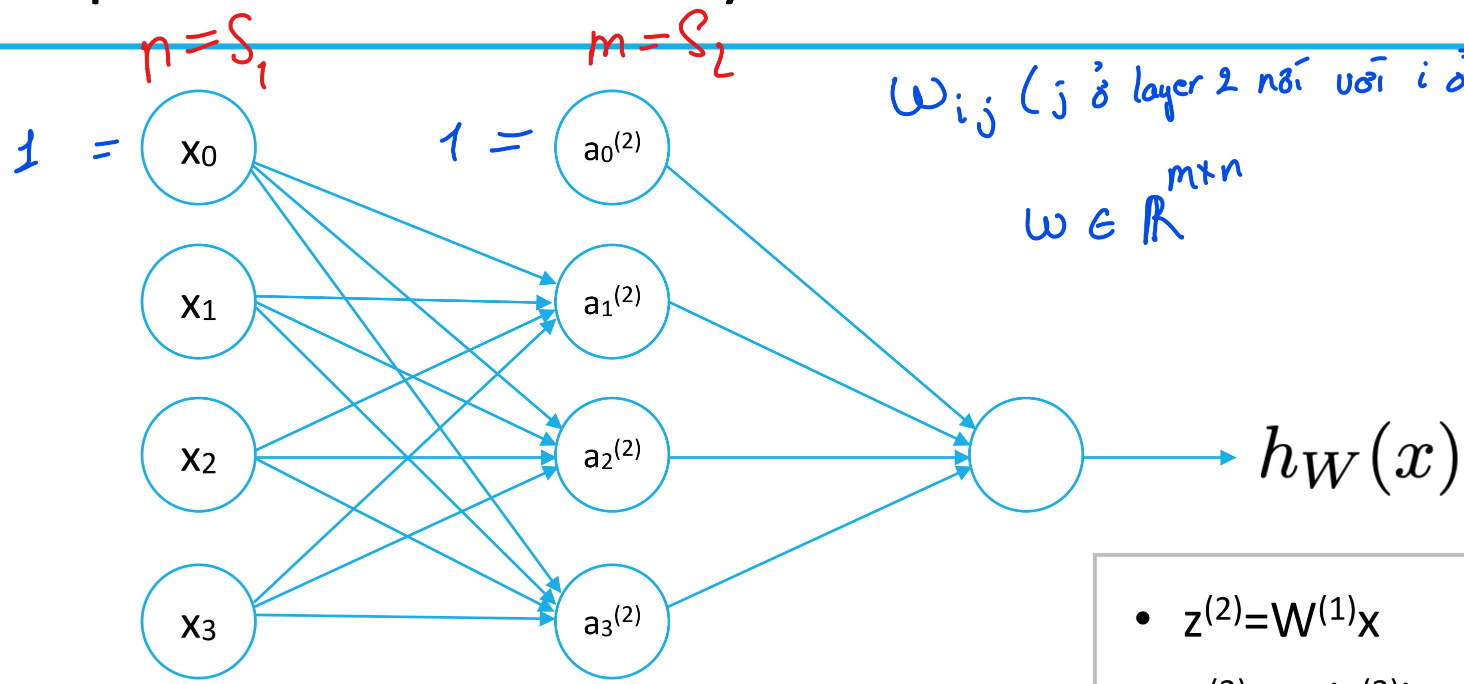


$$g(\theta^T x) = g(\omega x)$$

$$a_1^{(2)} = g\left(W_{10}^{(1)} x_0 + W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3\right)$$
$$a_2^{(2)} = g\left(W_{20}^{(1)} x_0 + W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3\right)$$
$$a_3^{(2)} = g\left(W_{30}^{(1)} x_0 + W_{31}^{(1)} x_1 + W_{32}^{(1)} x_2 + W_{33}^{(1)} x_3\right)$$

$$h_W(x) = a_1^{(3)} = g\left(W_{10}^{(2)} a_0^{(2)} + W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + W_{13}^{(2)} a_3^{(2)}\right)$$

Representation by vector



w_{ij} (j ở layer 2 nối với i ở layer 1)

$w \in \mathbb{R}^{m \times n}$

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

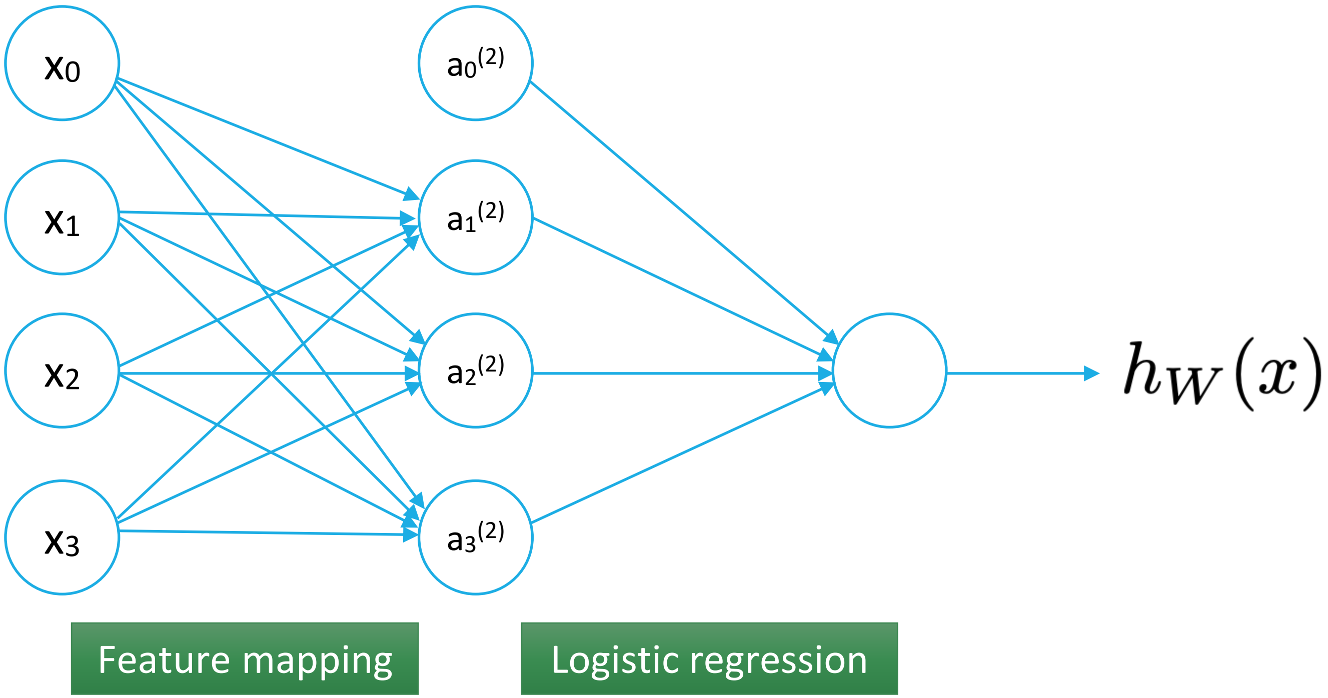
$$a_2^{(2)} = g\left(W_{20}^{(1)}x_0 + W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3\right)$$

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

$$h_W(x) = a_1^{(3)} = g\left(W_{10}^{(2)}a_0^{(2)} + W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)}\right)$$

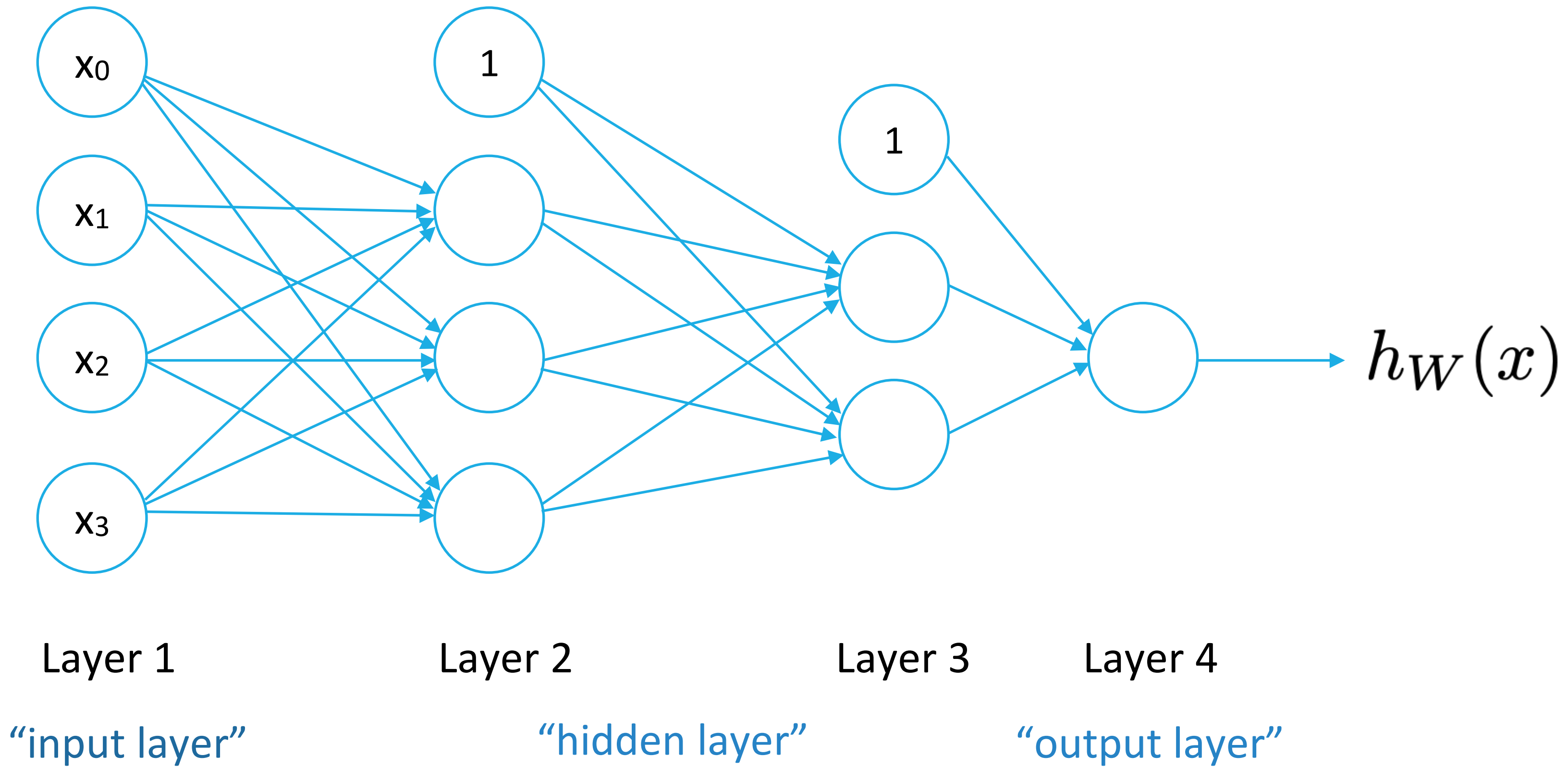
- $z^{(2)} = W^{(1)}x$
- $a^{(2)} = g(z^{(2)})$
- Add $a_0^{(2)} = 1$
- $z^{(3)} = W^{(2)}a^{(2)}$
- $h_W(x) = a^{(3)} = g(z^{(3)})$

Feature self-learning network

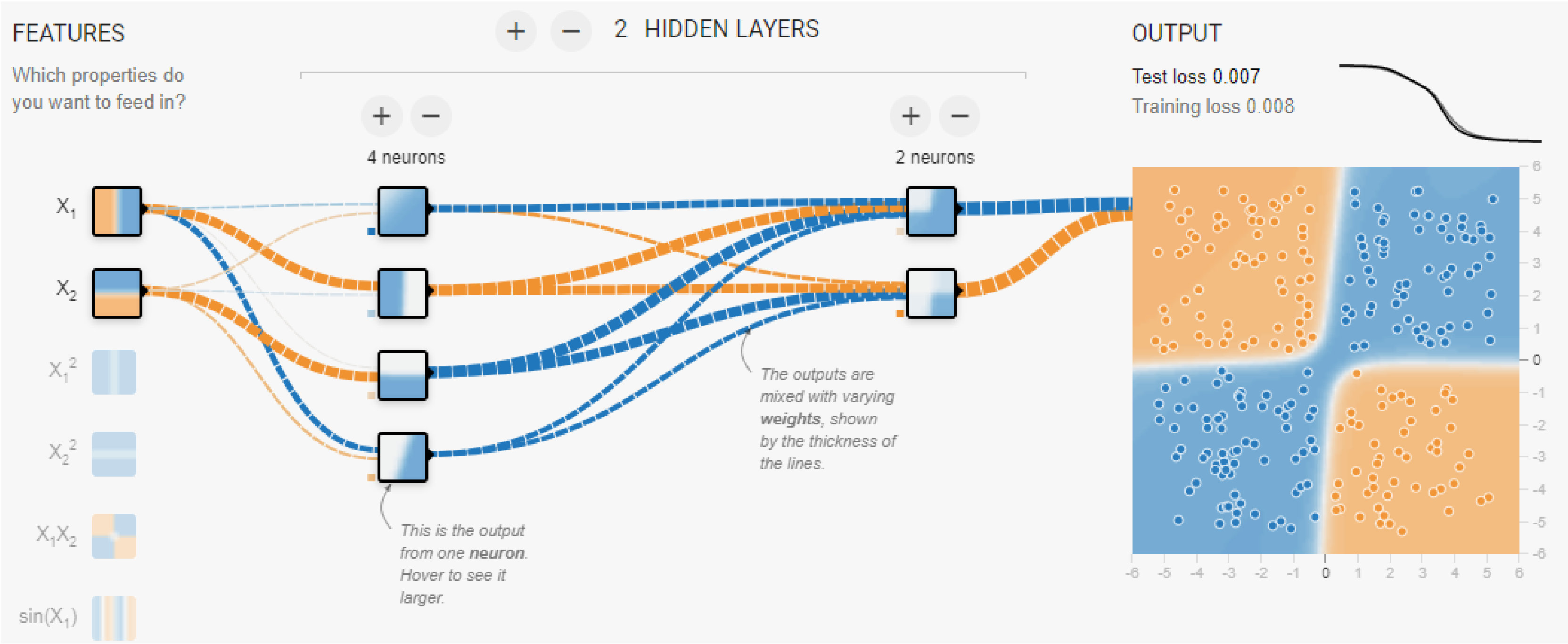


$$h_W(x) = a_1^{(3)} = g\left(W_{10}^{(2)} a_0^{(2)} + W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + W_{13}^{(2)} a_3^{(2)}\right)$$

Other architectures

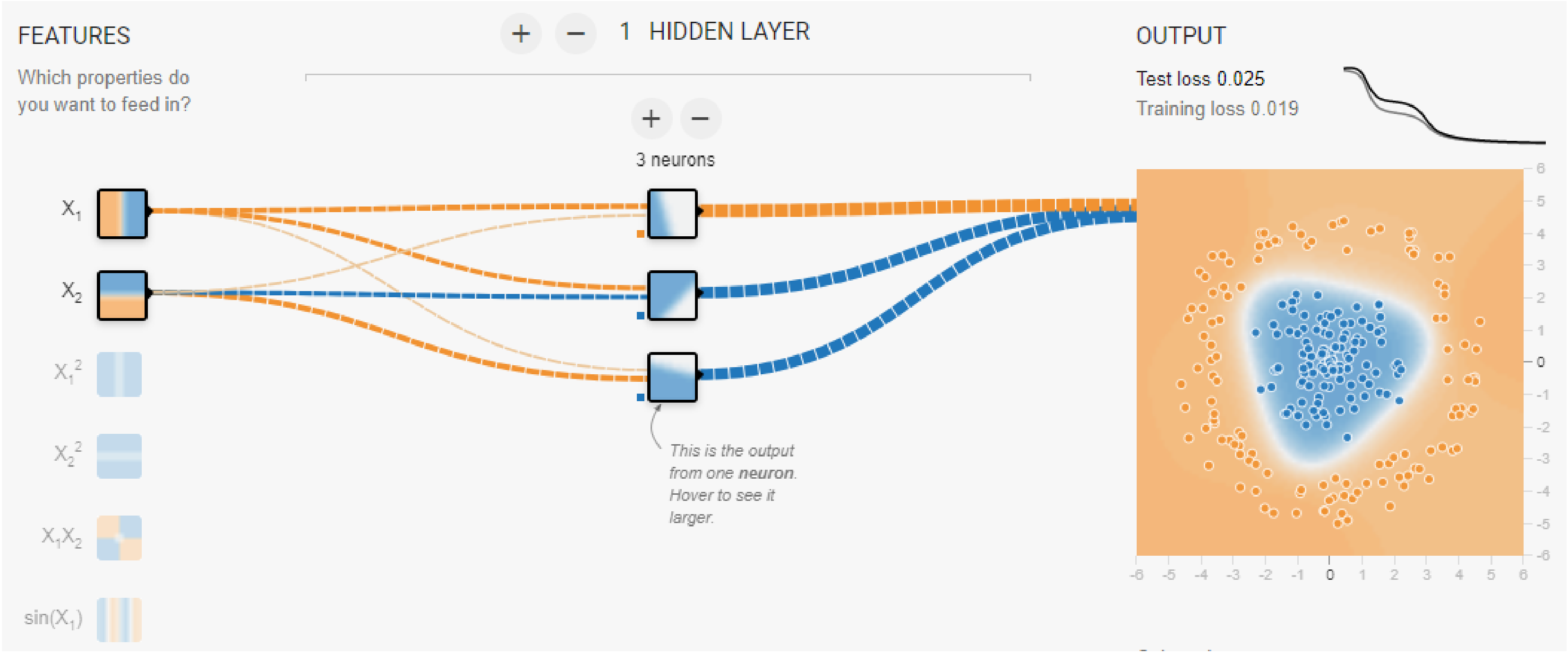


Feature self-learning network



Source: https://playground.tensorflow.org

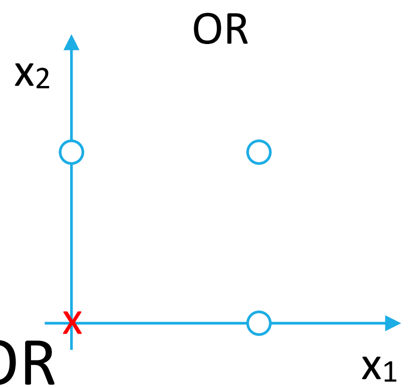
Feature self-learning network

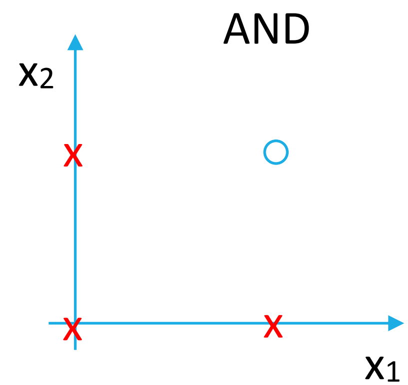


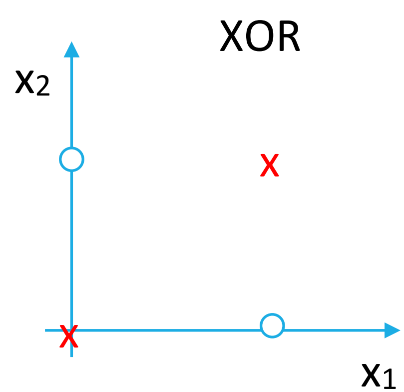
Source: https://playground.tensorflow.org

Non-linear classifier

- Linear function
  - OR
  - AND
- Non-linear function XOR
  - XOR(x1, x2)
  - a1 = AND(x1, NOT(x2))
  - a2 = AND(NOT(x1), x2)
  - y = OR(a1, a2)

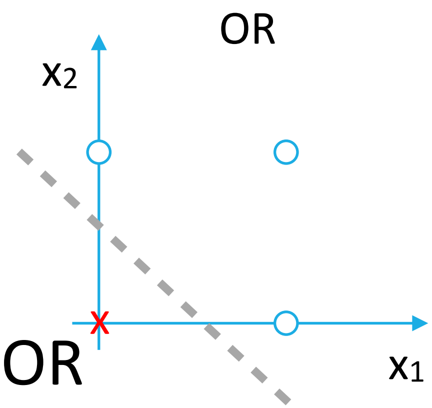


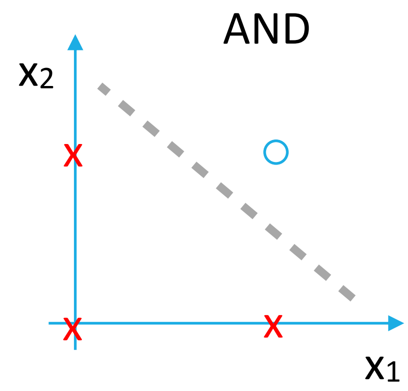


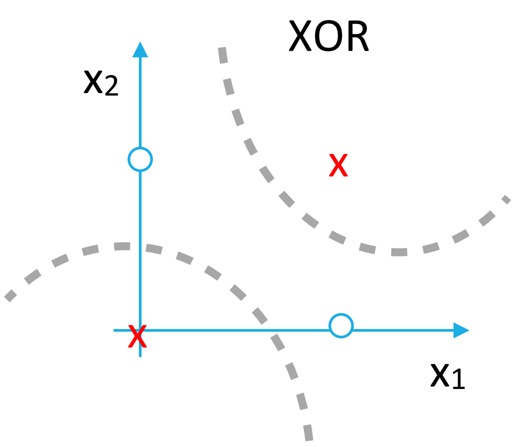


Non-linear classifier

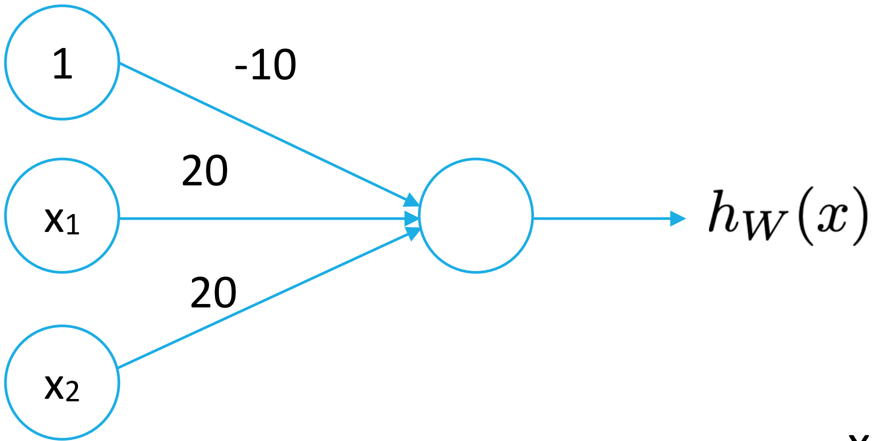
- Linear function
  - OR
  - AND
- Non-linear function XOR
  - XOR(x1, x2)
  - a1 = AND(x1, NOT(x2))
  - a2 = AND(NOT(x1), x2)
  - y = OR(a1, a2)

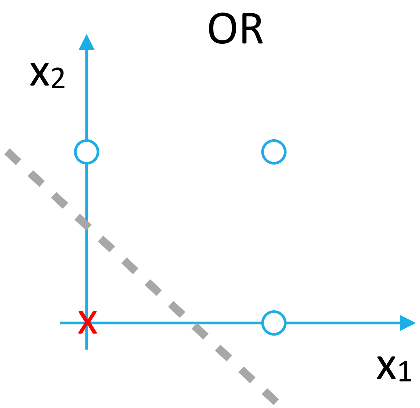




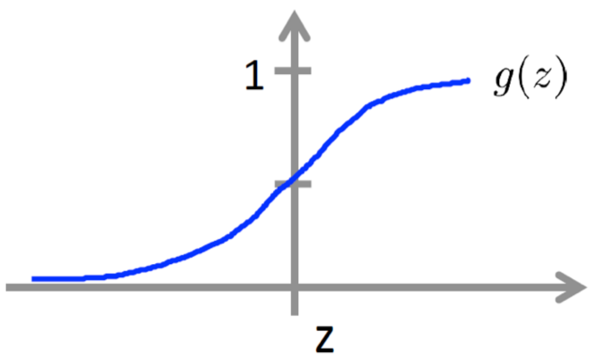


Function OR



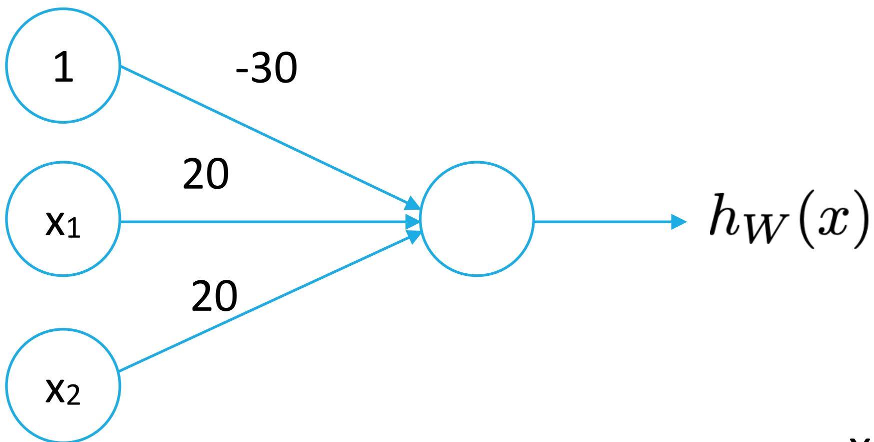


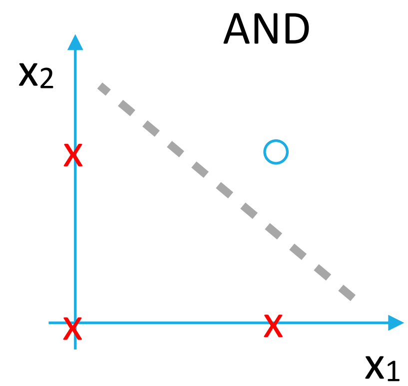
$$h_W(x) = g(-10 + 20x_1 + 20x_2)$$



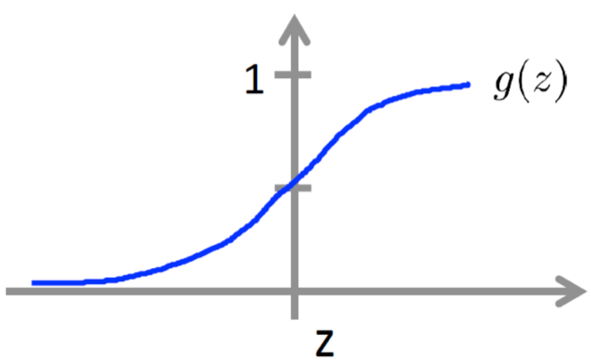
x_1	x_2	$h_W(x)$
0	0	0
0	1	1
1	0	1
1	1	1

Function AND



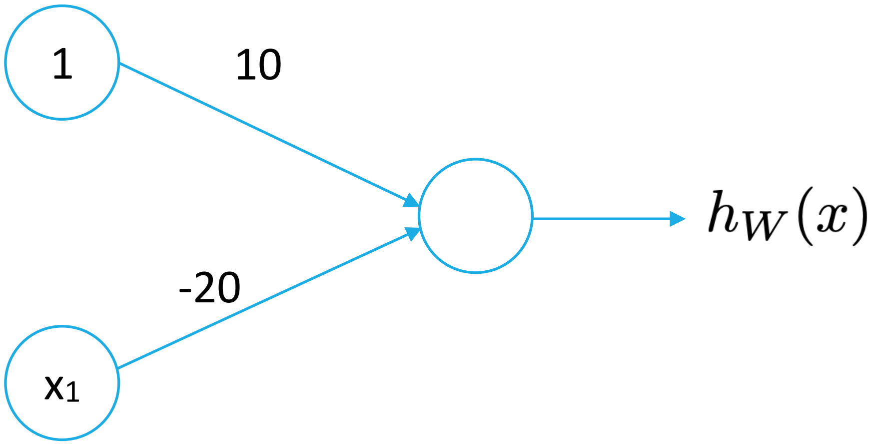


$$h_W(x) = g(-30 + 20x_1 + 20x_2)$$

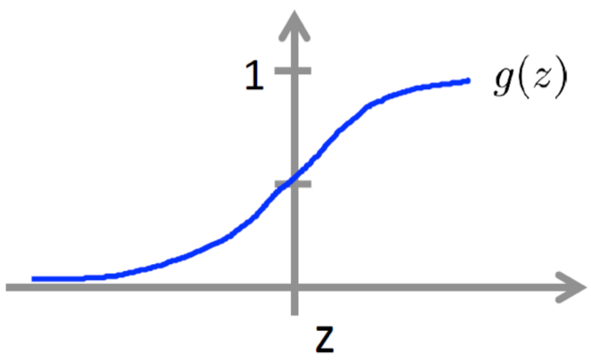


x_1	x_2	$h_W(x)$
0	0	0
0	1	0
1	0	0
1	1	1

Function NOT

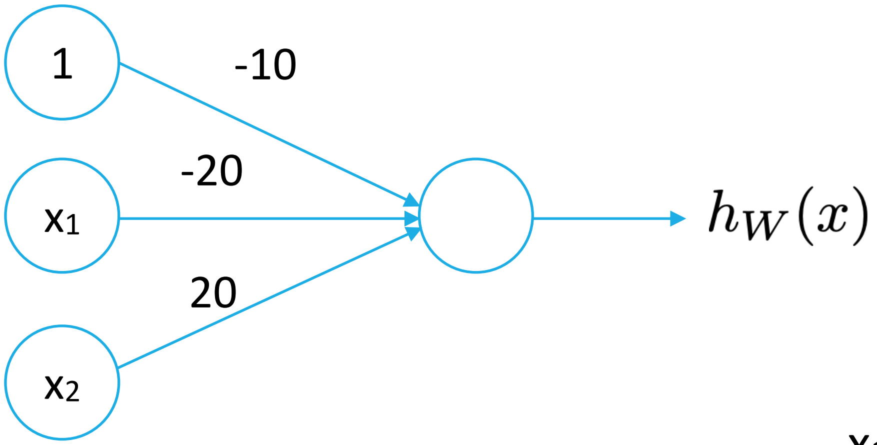


$$h_W(x) = g(10 - 20x_1)$$

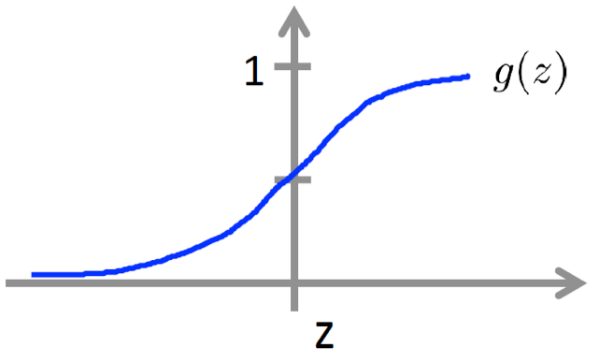


x_1	$h_W(x)$
0	1
1	0

Function AND(NOT(x1), x2)

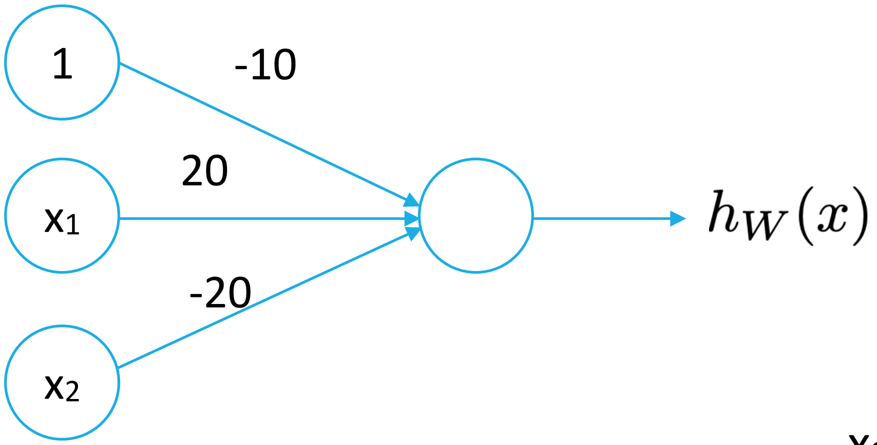


$$h_W(x) = g(-10 - 20x_1 + 20x_2)$$

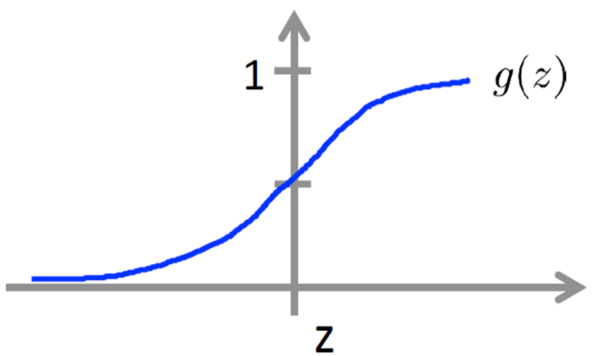


x_1	x_2	$h_W(x)$
0	0	0
0	1	1
1	0	0
1	1	0

Function AND(x1, NOT(x2))

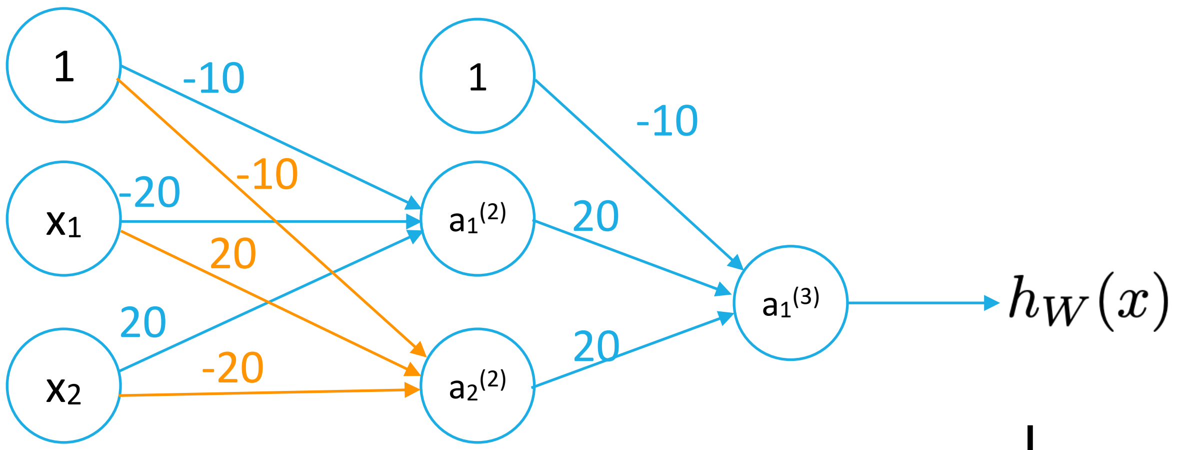


$$h_W(x) = g(-10 + 20x_1 - 20x_2)$$



x_1	x_2	$h_W(x)$
0	0	0
0	1	0
1	0	1
1	1	0

Function XOR



x_1	x_2	$h_W(x)$
0	0	0
0	1	1
1	0	1
1	1	0

$\mathrm{XOR}(x_1, x_2) = \mathrm{OR}(\mathrm{AND}(\mathrm{NOT}(x_1), x_2), \mathrm{AND}(x_1, \mathrm{NOT}(x_2)))$

Exercise with playground

Playground: https://playground.tensorflow.org

In this exercise, we will train our first little neural net. Neural nets will give us a way to learn nonlinear models without the use of explicit feature crosses.

Task 1: The model as given combines our two input features into a single neuron. Will this model learn any nonlinearities? Run it to confirm your guess.

Task 2: Try increasing the number of neurons in the hidden layer from 1 to 2, and also try changing from a Linear activation to a nonlinear activation like ReLU. Can you create a model that can learn nonlinearities? Can it model the data effectively?

Task 3: Try increasing the number of neurons in the hidden layer from 2 to 3, using a nonlinear activation like ReLU. Can it model the data effectively? How does model quality vary from run to run?

Task 4: Continue experimenting by adding or removing hidden layers and neurons per layer. Also feel free to change learning rates, regularization, and other learning settings. What is the smallest number of neurons and layers you can use that gives test loss of 0.177 or lower?

Does increasing the model size improve the fit, or how quickly it converges? Does this change how often it converges to a good model? For example, try the following architecture:

- First hidden layer with 3 neurons.
- Second hidden layer with 3 neurons.
- Third hidden layer with 2 neurons.

Source: Google Crash Course

Training and backward propagation

Part 2

Cost function

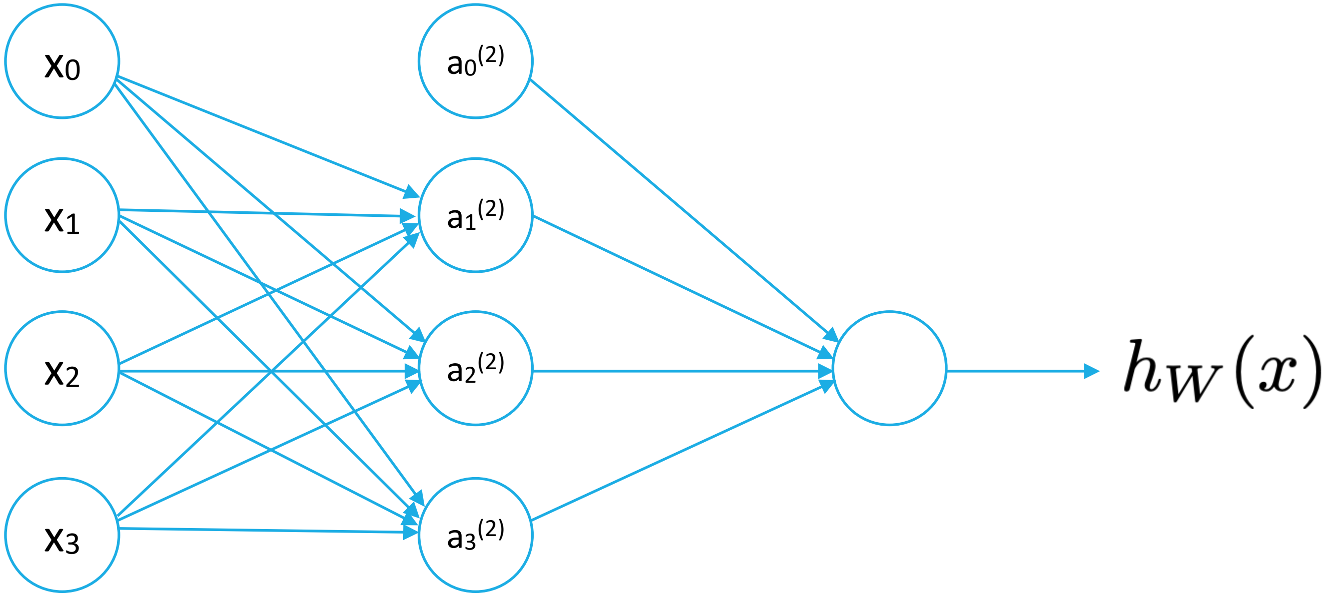
$$J(W) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log(h_W(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_W(x^{(i)}))\right]$$

Gradient

$$\frac{dJ}{dW} = \left[\frac{dJ}{dW_{10}^{(1)}}, \frac{dJ}{dW_{11}^{(1)}}, \ldots, \frac{dJ}{dW_{10}^{(L-1)}}, \ldots, \frac{dJ}{dW_{s_{l+1}s_l}^{(L-1)}}\right]$$

Derivative

$$\frac{dJ}{dW_{ij}^{(l)}} = \sum_{k=1}^{s_{(l+1)}} \frac{dJ}{dz_k^{(l+1)}} \frac{dz_k^{(l+1)}}{dW_{ij}^{(l)}}$$



$$\frac{dJ}{dz_i^{(l)}} = \sum_{j=1}^{s_{(l+1)}} \frac{dJ}{dz_j^{(l+1)}} \frac{dz_j^{(l+1)}}{dz_i^{(l)}}$$

$$= \sum_{j=1}^{s_{(l+1)}} \boxed{\delta_j^{(l+1)}} \frac{dz_j^{(l+1)}}{dz_i^{(l)}}$$

$$= \sum_{j=1}^{s_{(l+1)}} \delta_j^{(l+1)} \boxed{\frac{d}{dz_i^{(l)}} \sum_{k=0}^{s_l} W_{jk}^{(l)} g(z_k^{(l)})}$$

$$= \sum_{j=1}^{s_{(l+1)}} \delta_j^{(l+1)} \boxed{W_{ji}^{(l)} g'(z_i^{(l)})}$$

$$= g'(z_i^{(l)}) \sum_{j=1}^{s_{(l+1)}} (W^{(l)})_{ij}^T \delta_j^{(l+1)}$$

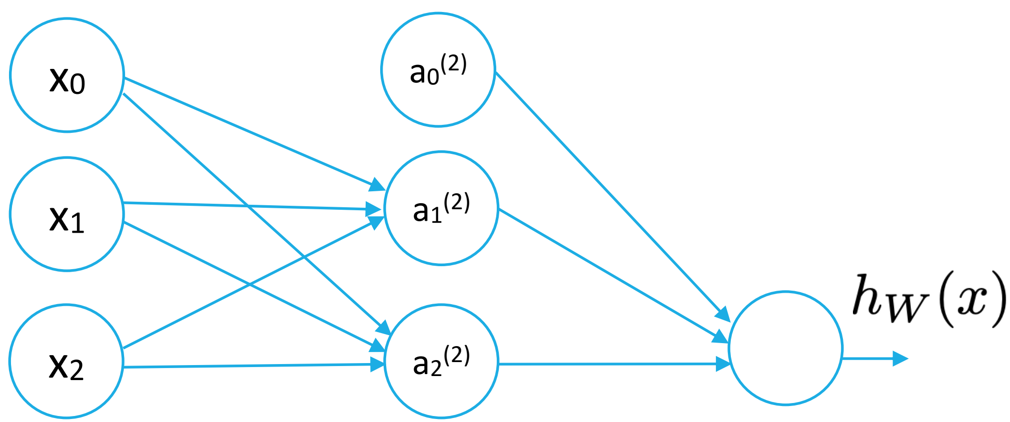
sigmoid | ...

Notation

$$\delta_j^{(l)} = \frac{dJ}{dz_j^{(l)}}$$

s_{L+1} = Số lượng nút ở layer(L+1)

$z_k^{(l)} = x_0 w_0 + x_1 w_1 + \dots$ (layer l)



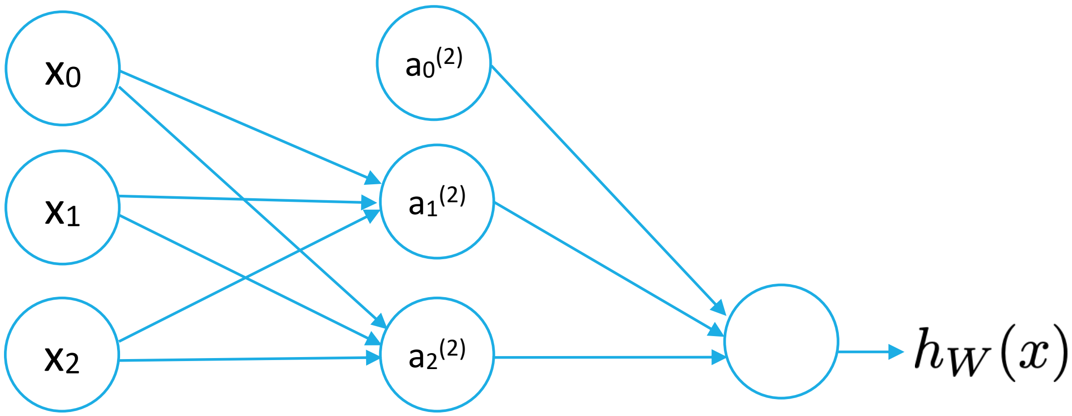
$$\frac{dJ}{dW_{ij}^{(l)}} = \sum_{k=1}^{s_{(l+1)}} \frac{dJ}{dz_k^{(l+1)}} \frac{dz_k^{(l+1)}}{dW_{ij}^{(l)}}$$

$$= \sum_{k=1}^{s_{(l+1)}} \delta_k^{(l+1)} \boxed{\frac{d}{dW_{ij}^{(l)}} \sum_{t=0}^{s_l} W_{kt}^{(l)} a_t^{(l)}}$$

$$= \delta_i^{(l+1)} \frac{d}{dW_{ij}^{(l)}} W_{ij}^{(l)} a_j^{(l)}$$

$$= \delta_i^{(l+1)} a_j^{(l)}$$

$$\frac{dJ}{dW^{(l)}} = \delta^{(l+1)} a^{(l)T}$$



Backward propagation algorithm

(1) Apply forward propagation to calculate:

$z^{(1)}, \ldots, z^{(L)}, a^{(1)}, \ldots, a^{(L)}$ and $J(z^{(L)})$

(2) $\delta^{(L)} = \dfrac{dJ}{dz^{(L)}}$

(3) for l = L-1 to 1

(4) $\dfrac{dJ}{dz^{(l)}} = g'(z^{(l)})(W^{(l)T}\delta^{(l+1)})$

(5) $\dfrac{dJ}{dW^{(l)}} = \delta^{(l+1)}a^{(l)T}$

(6) end for

Regularization

- Cost function

$$J(W) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log(h_W(x^{(i)})) + (1-y^{(i)})\log(1-h_W(x^{(i)}))\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{j=1}^{s_{l+1}}\sum_{i=1}^{s_l}(W_{ji}^{(l)})^2$$

(L2)

Regularization

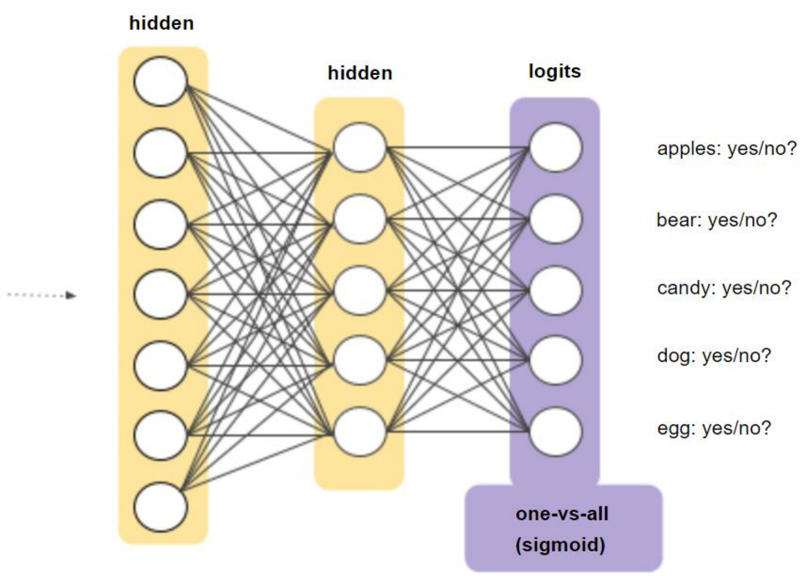
- Gradient

$$\frac{dJ}{dz^{(l)}} = g'(z^{(l)})(W^{(l)T}\delta^{(l+1)}) + \lambda W^{(l)} \qquad \text{if} \quad j \neq 0$$

$$\frac{dJ}{dz^{(l)}} = g'(z^{(l)})(W^{(l)T}\delta^{(l+1)}) \qquad \text{if} \quad j = 0$$

Neural network for multi-classes

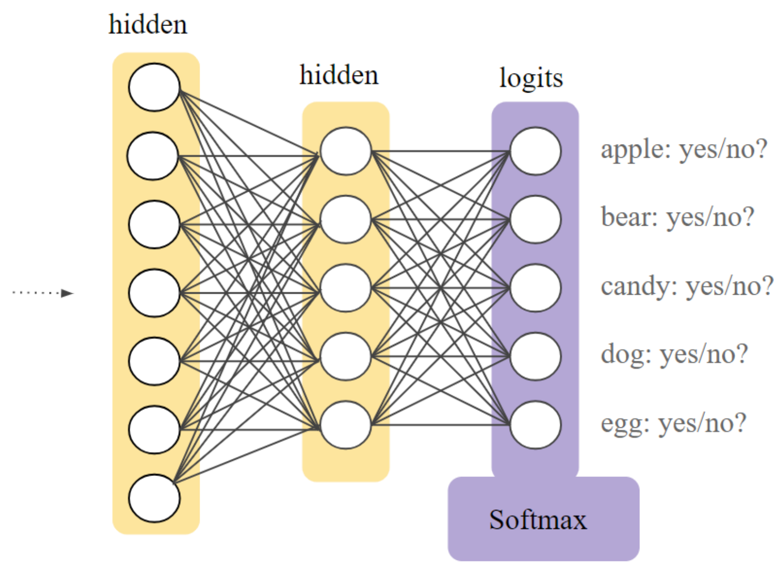
- One vs. All
  - Train K classifiers



Source: Google Crash Course

Neural network for multi-classes

❑ Softmax



$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

- x: output of last hidden layer
- w_j: weights connected to class #j

Example

- P(y="apple") = 0.76
- P(y="bear") = 0.12
- P(y="candy") = 0.03
- P(y="dog") = 0.02
- P(y="egg") = 0.07

$\Sigma = 1$

Source: Google Crash Course

Neural network for multi-classes

- One hot encoding: represent for output with vectors
  - Classes: {apple, bear, candy, dog, egg}
  - One hot encodings for
    - apple: $[1, 0, 0, 0, 0]^T$
    - bear: $[0, 1, 0, 0, 0]^T$
    - candy: $[0, 0, 1, 0, 0]^T$
    - dog: $[0, 0, 0, 1, 0]^T$
    - egg: $[0, 0, 0, 0, 1]^T$

Neural network for multi-classes

- Cross Entropy loss

$$-\sum_{c=1}^{K} y_c \log(p_c)$$

P(y="apple")

P(y="bear")

Example: y = [1, 0, 0, 0, 0] (apple), $\hat{y} = [0.76, 0.12, 0.03, 0.02, 0.07]$

- p_1: probability the sample belonging to **apple** given by the model
- p_2: probability the sample belonging to **bear** given by the model
- …

Loss = $y_1\log(p_1) + y_2\log(p_2) + y_3\log(p_3) + y_4\log(p_4) + y_5\log(p_5)$

= $y_1\log(p_1)$ = 0.76

Practice

- Predict house price with California Housing Dataset
  - [Intro to Neural Nets.ipynb]
- Recognize hand-written digits with neural network
  - [Multi-class classification with MNIST.ipynb]