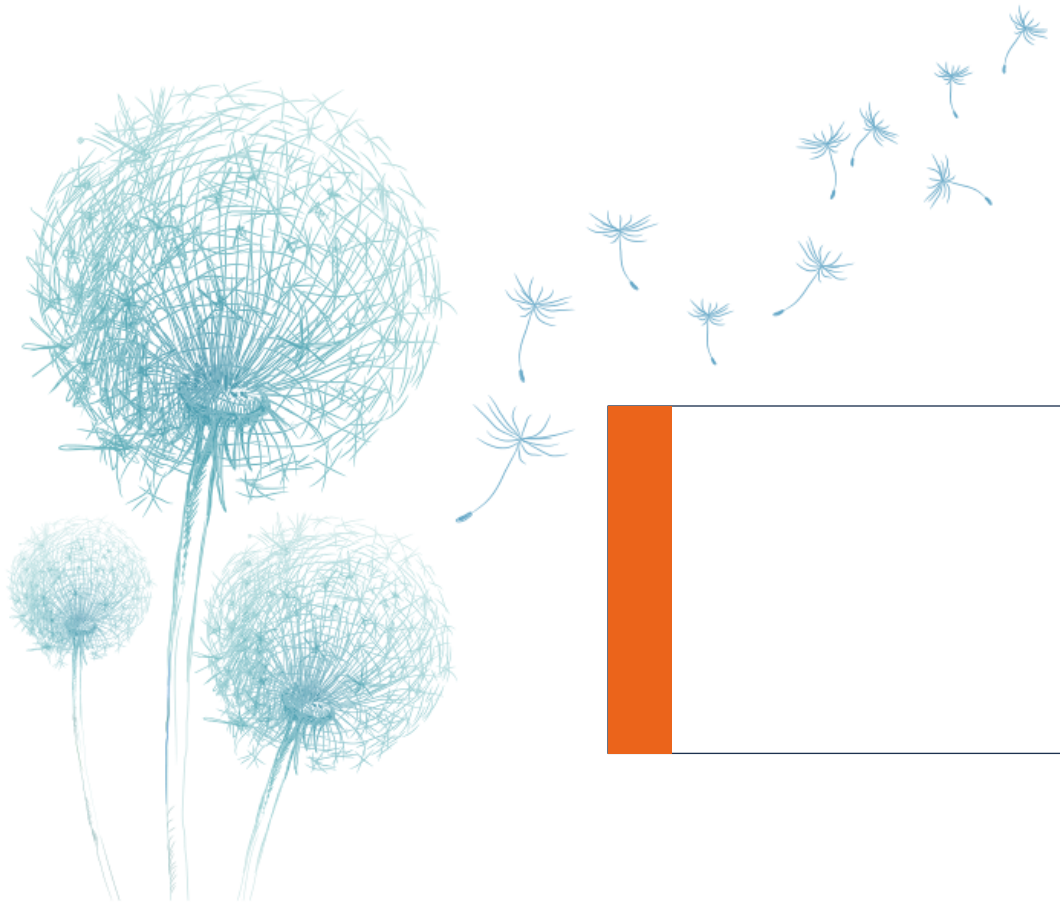
Neural networks and deep learning

Feedforward neural network

Kun Suo
Computer Science, Kennesaw State University
https://kevinsuo.github.io/



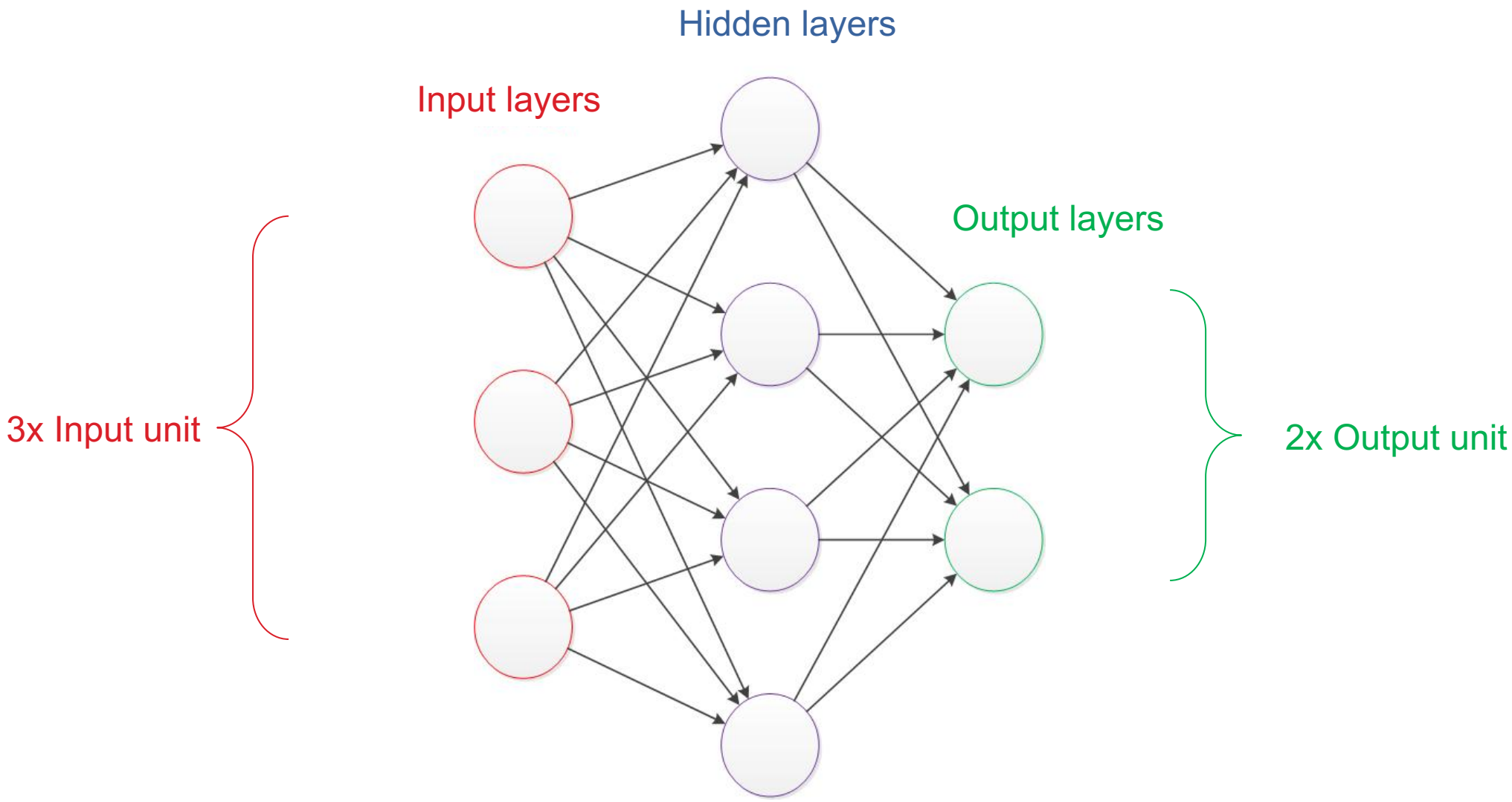
Neural network

Neural networks

- Neural network is an important machine learning technology. It is foundation of deep learning

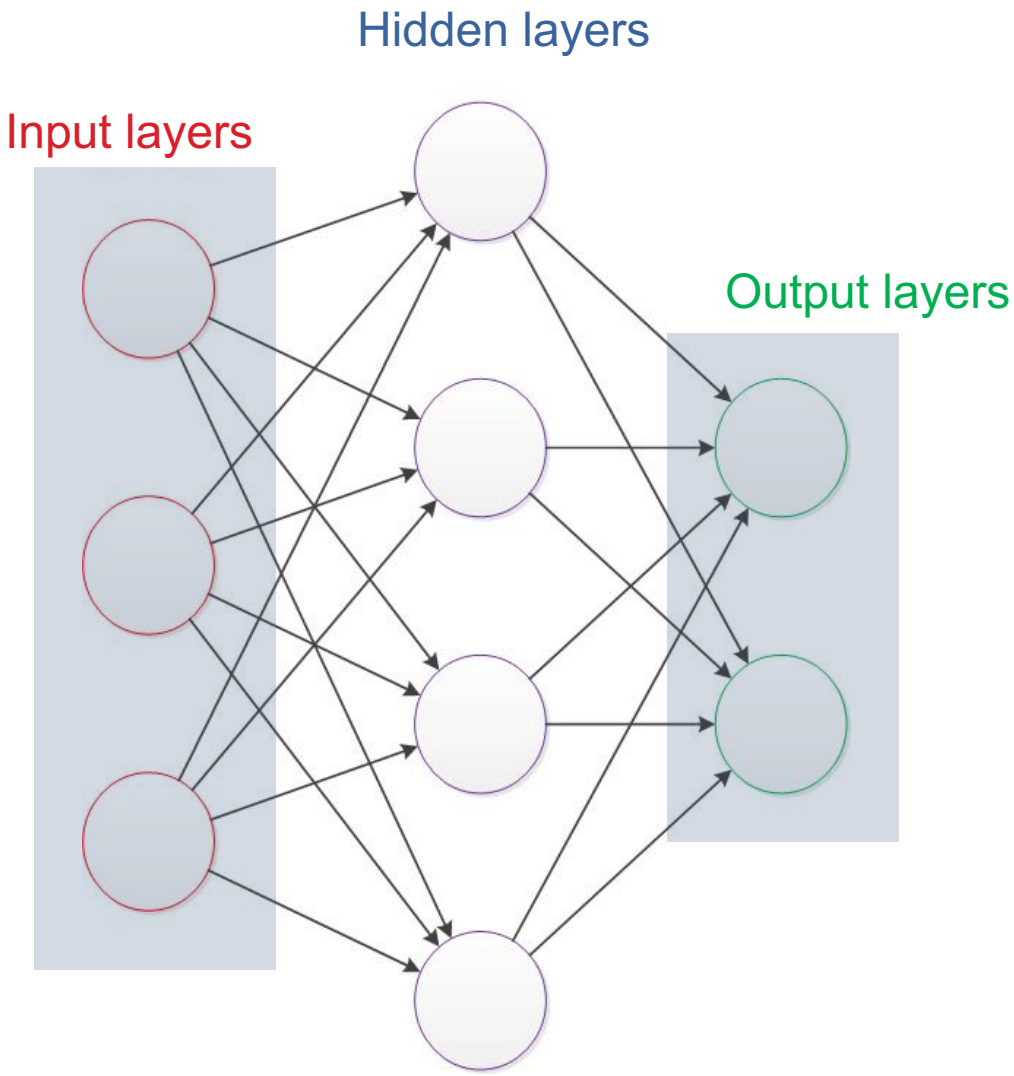
- Neural network is a kind of model that simulates the human brain in order to achieve artificial intelligence and machine learning technology

Neural Networks Example



Neural Networks Example

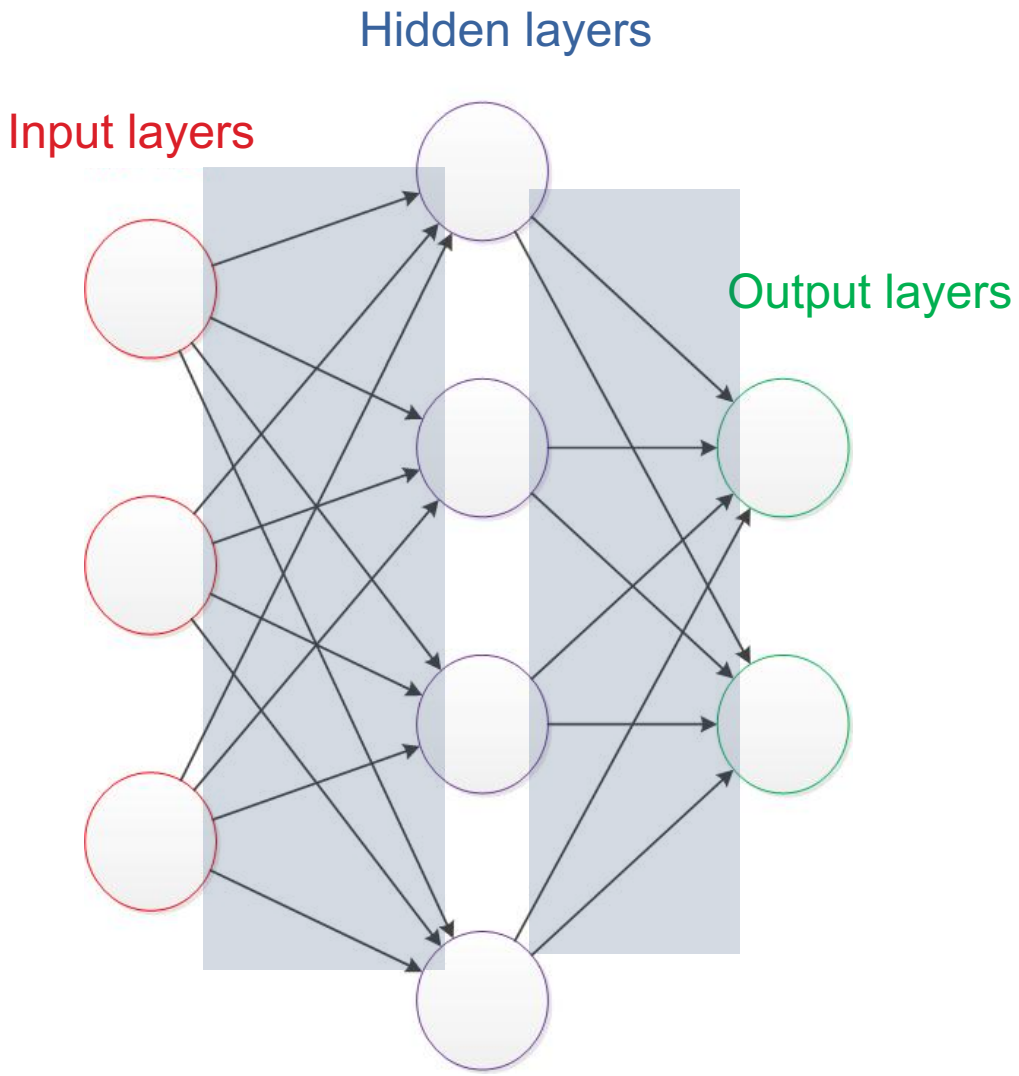
1, When designing a neural network, the number of nodes in the input layer and output layer is often fixed, and the middle layer can be freely specified



Neural Networks Example

1, When designing a neural network, the number of nodes in the input layer and output layer is often fixed, and the middle layer can be freely specified

2, The topology and arrows in the neural network structure diagram represent the flow of data during the prediction process

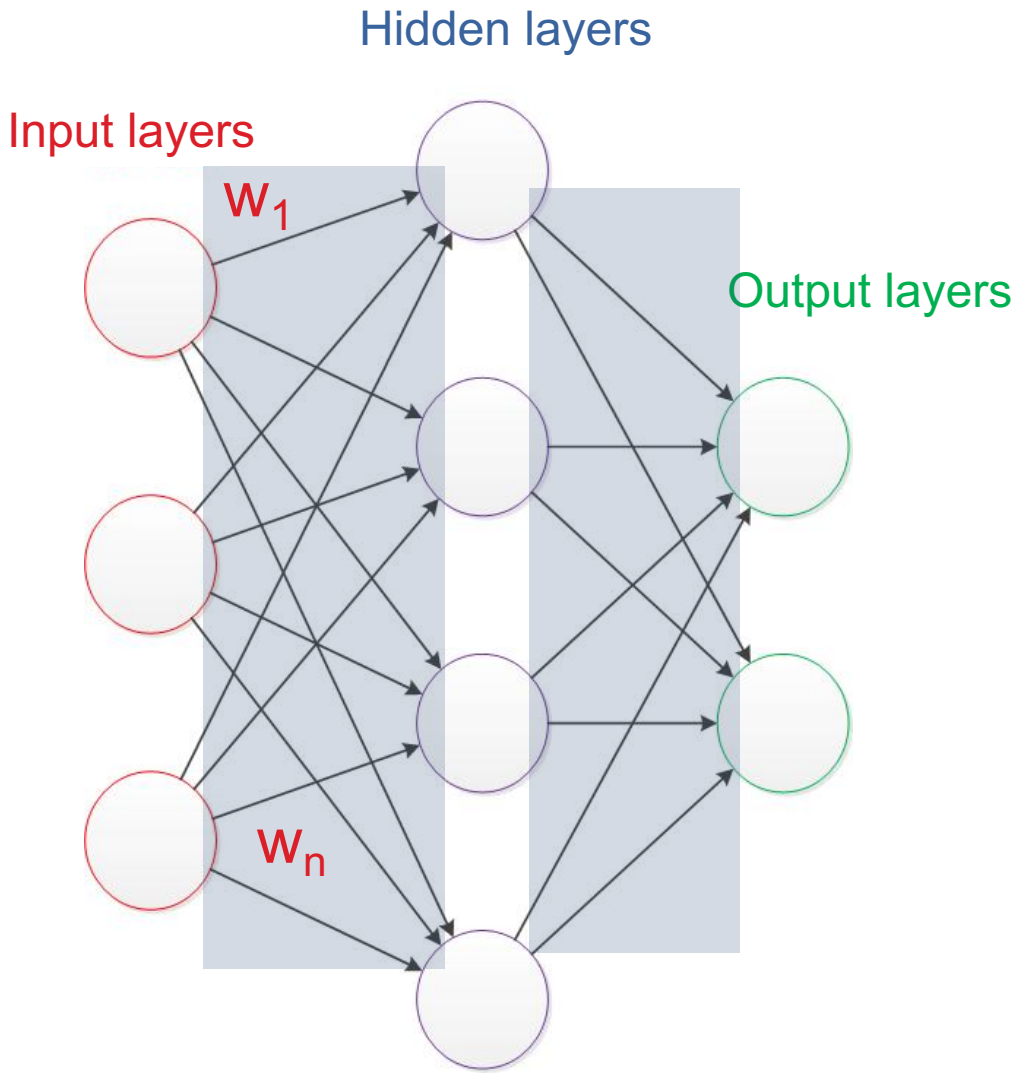


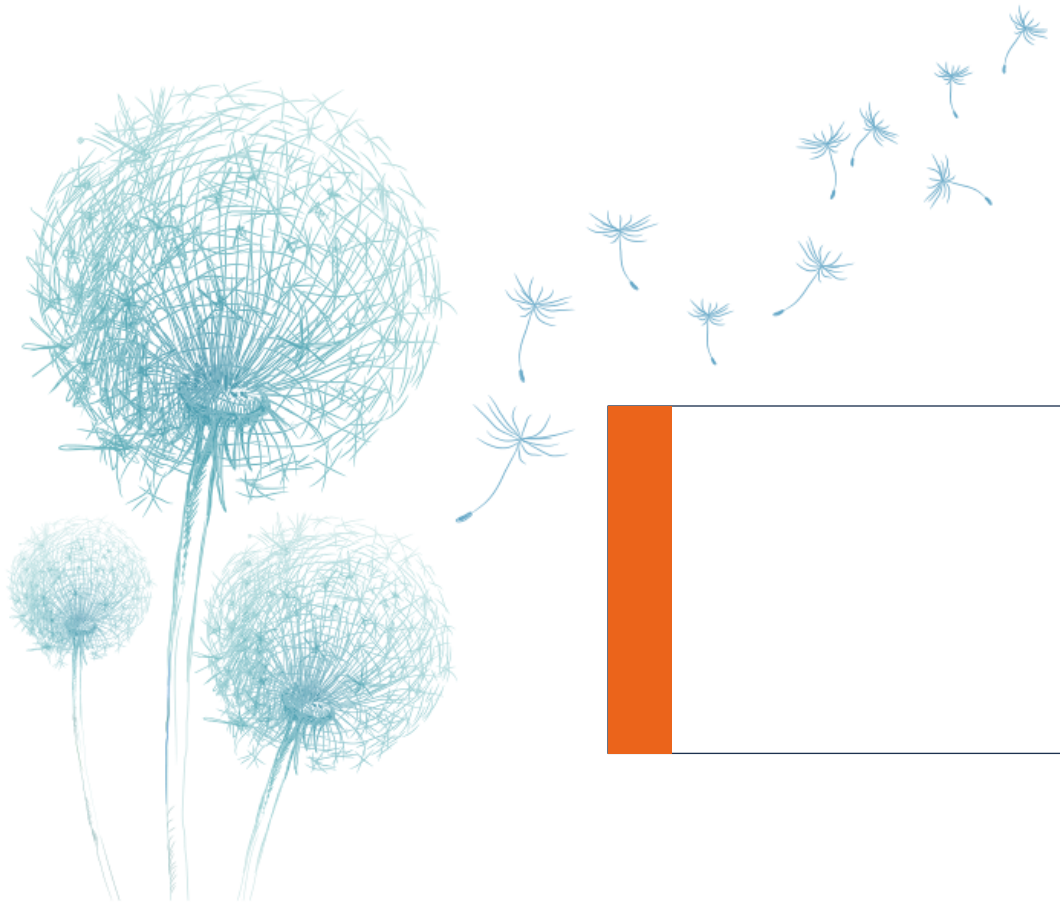
Neural Networks Example

1, When designing a neural network, the number of nodes in the input layer and output layer is often fixed, and the middle layer can be freely specified

2, The topology and arrows in the neural network structure diagram represent the flow of data during the prediction process

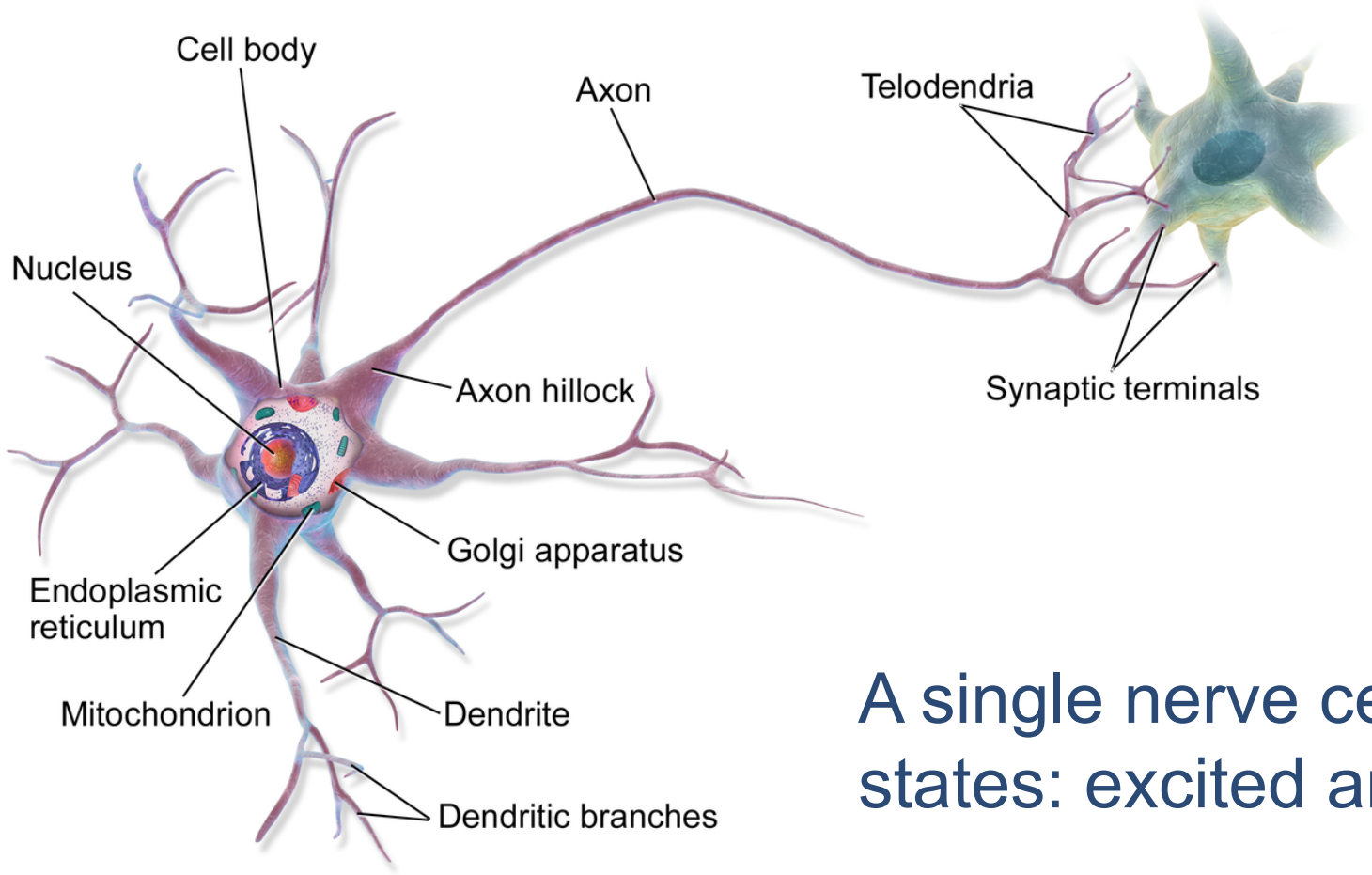
3, The key in the structure diagram is not the circles (representing "neurons"), but the connecting lines (representing the connections between "neurons"). Each connection line corresponds to a different weight (the value is called the weight), which is obtained by training





Neurons

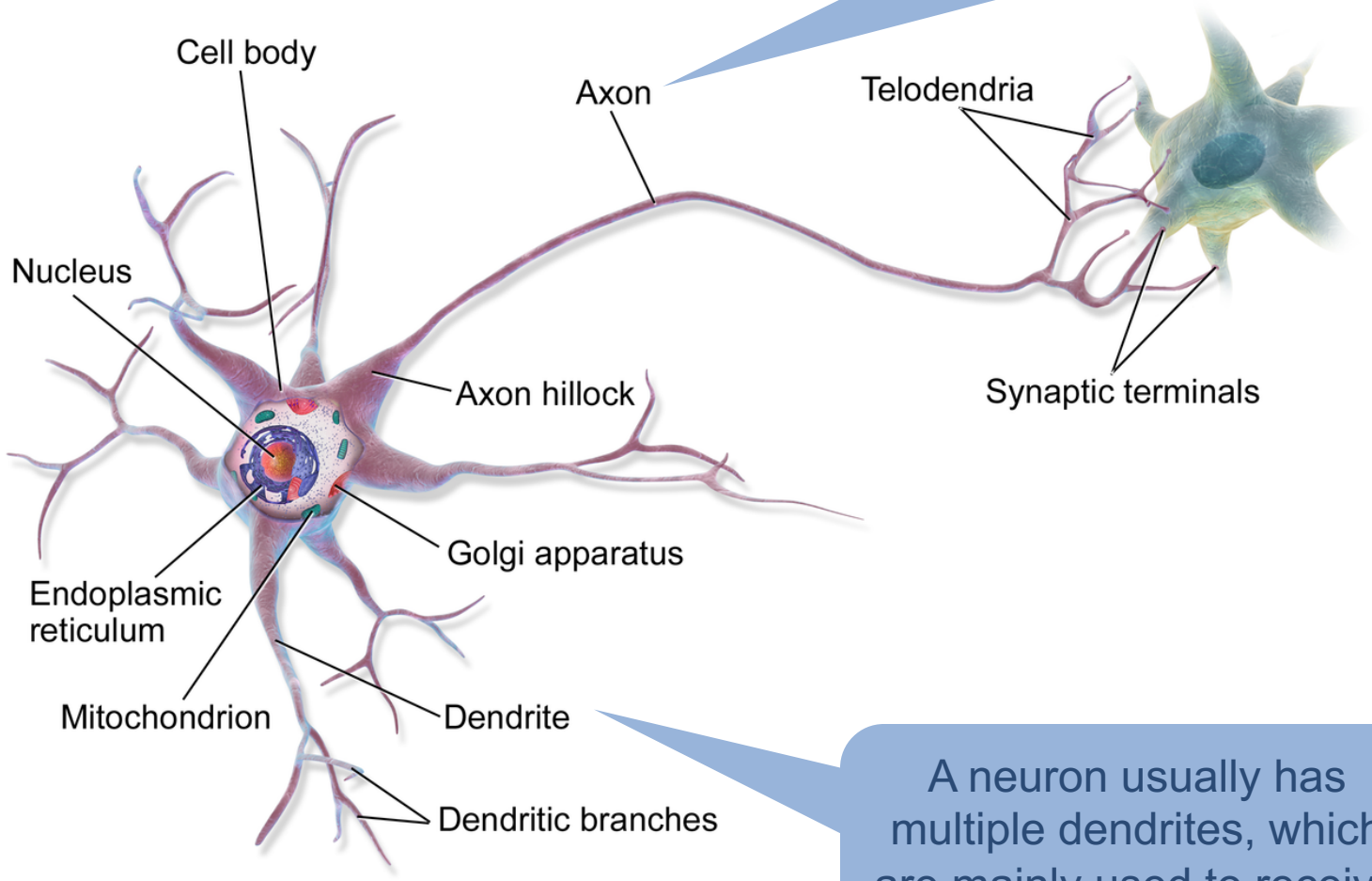
Biological neuron



A single nerve cell has only two states: excited and inhibited

Biological neuron

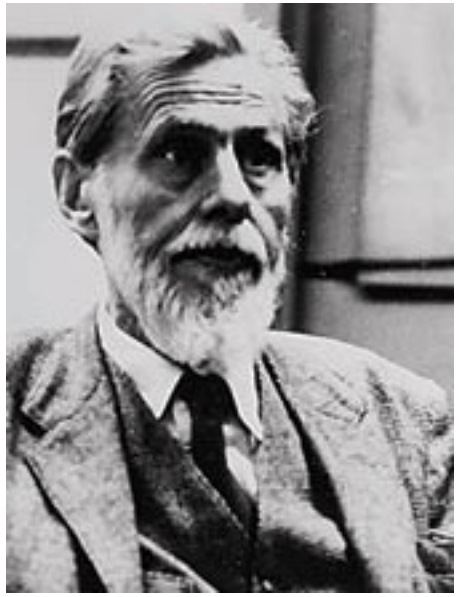
There is only one axon, and the axon tail transmits information to multiple other neurons



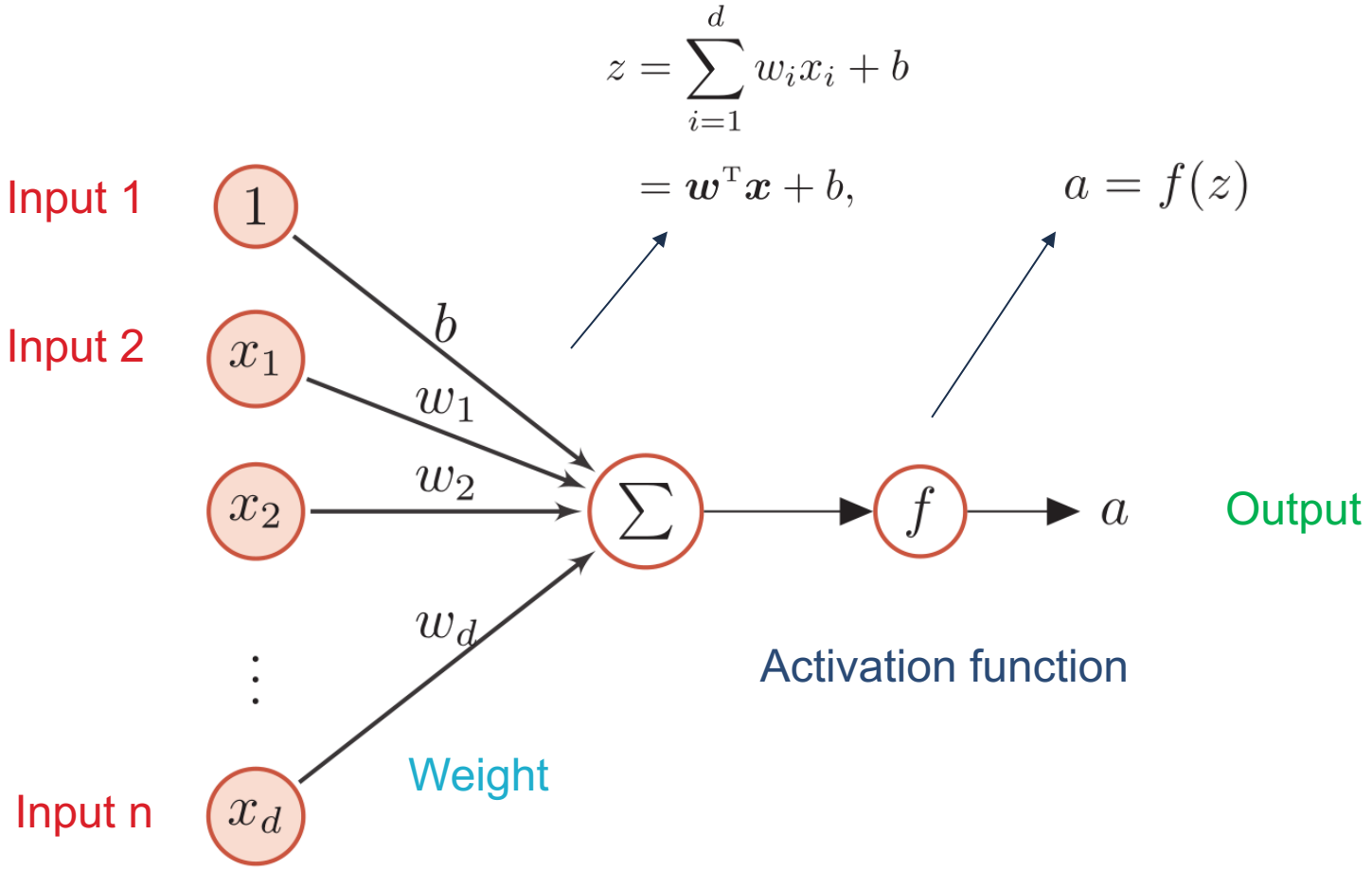
A neuron usually has multiple dendrites, which are mainly used to receive incoming information

Biological neuron

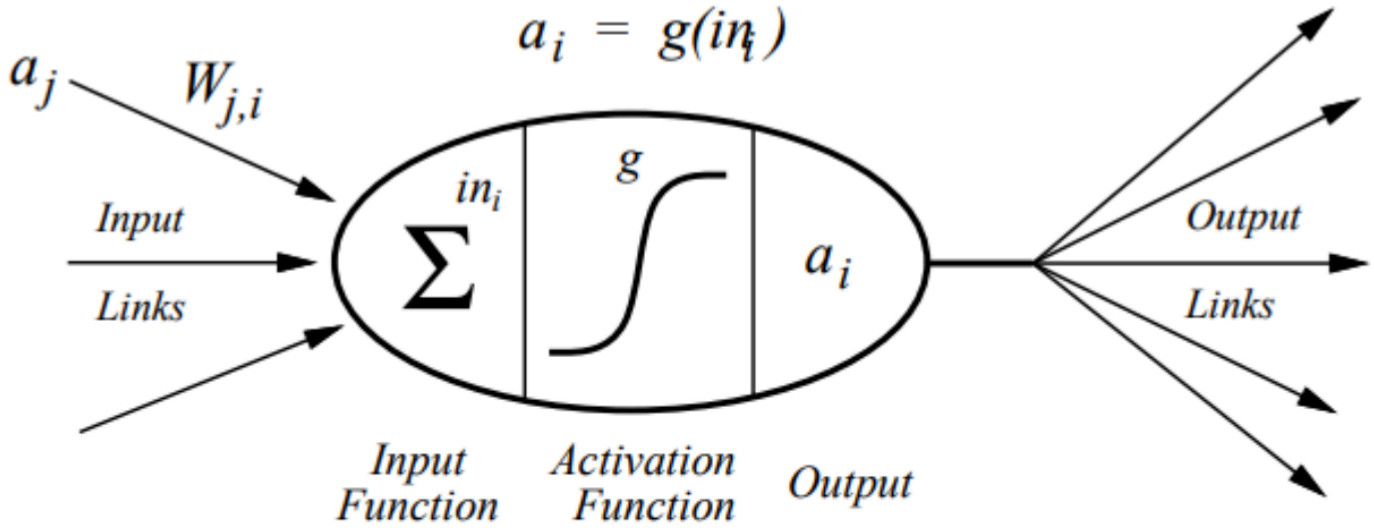
- In 1943, psychologist McCulloch and mathematician Pitts referred to the structure of biological neurons and published an abstract neuron model MP



Artificial neuron



Artificial neuron (Simplified)



$$a_i = g(\sum_j W_{j,i} a_j)$$

Ref: Dr. Manuela Veloso, CMU

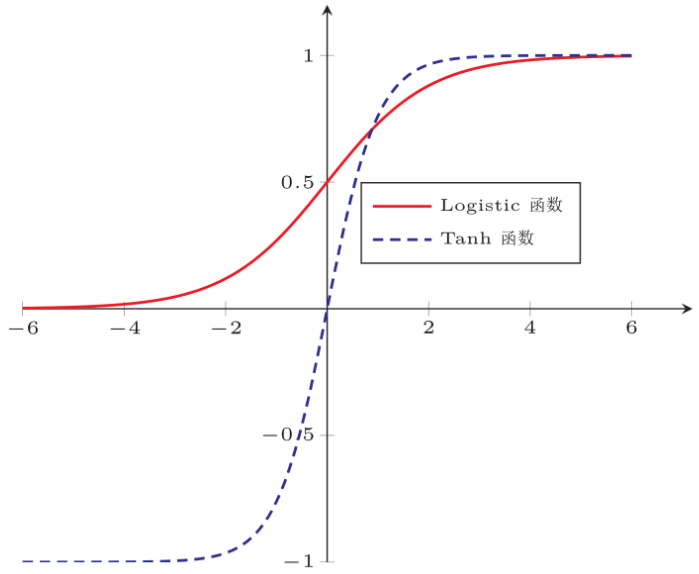
The nature of the activation function

- Continuous and differentiable (allowing a few points to be non-differentiable) nonlinear function.
  - The derivable activation function can directly use numerical optimization methods to learn network parameters.
- The activation function and its derivative function should be as simple as possible
  - Conducive to improving network computing efficiency.
- The value range of the derivative function of the activation function should be in a suitable interval
  - It cannot be too large or too small, otherwise it will affect the efficiency and stability of training.
- Monotonically increasing

Common activation functions

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

$$\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$



- Nature:
  - Saturation function
  - The Tanh function is zero-centralized, and the output of the logistic function is always greater than 0

Common activation functions

$$\mathrm{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$= \max(0, x).$$

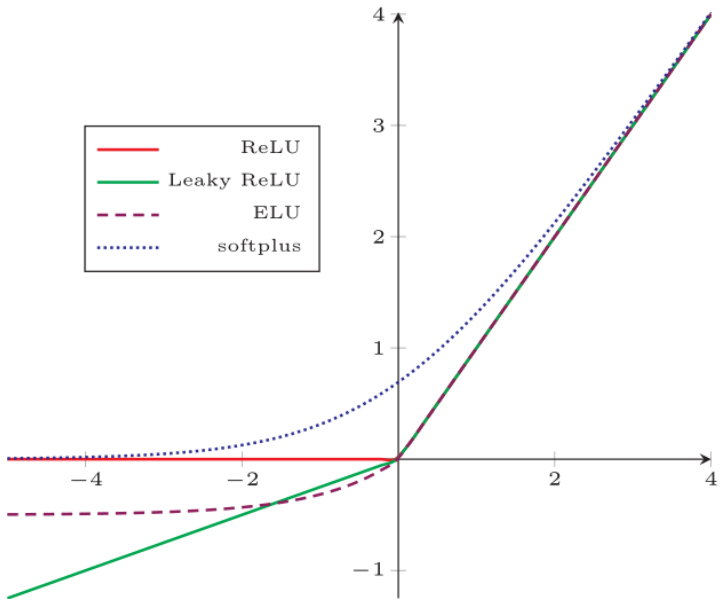
$$\mathrm{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases}$$

$$\mathrm{PReLU}_i(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma_i x & \text{if } x \leq 0 \end{cases}$$

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

$$= \max(0, x) + \min(0, \gamma(\exp(x) - 1))$$

$$\mathrm{softplus}(x) = \log(1 + \exp(x))$$

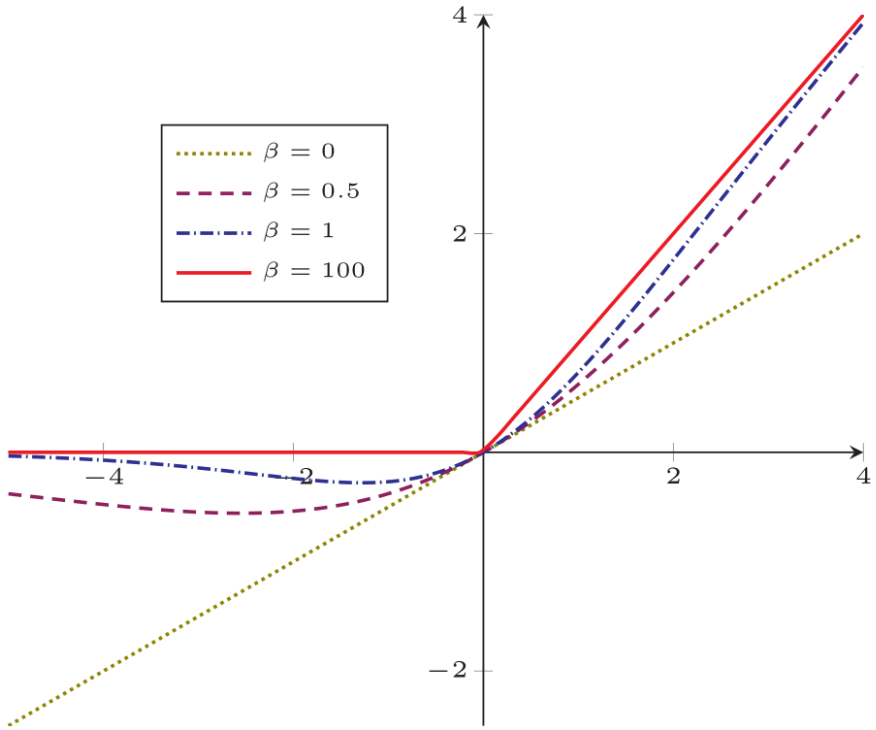


- Computationally more efficient
- Biological rationality
- Unilateral inhibition, wide excitability boundary
- To a certain extent alleviate the problem of vanishing gradient

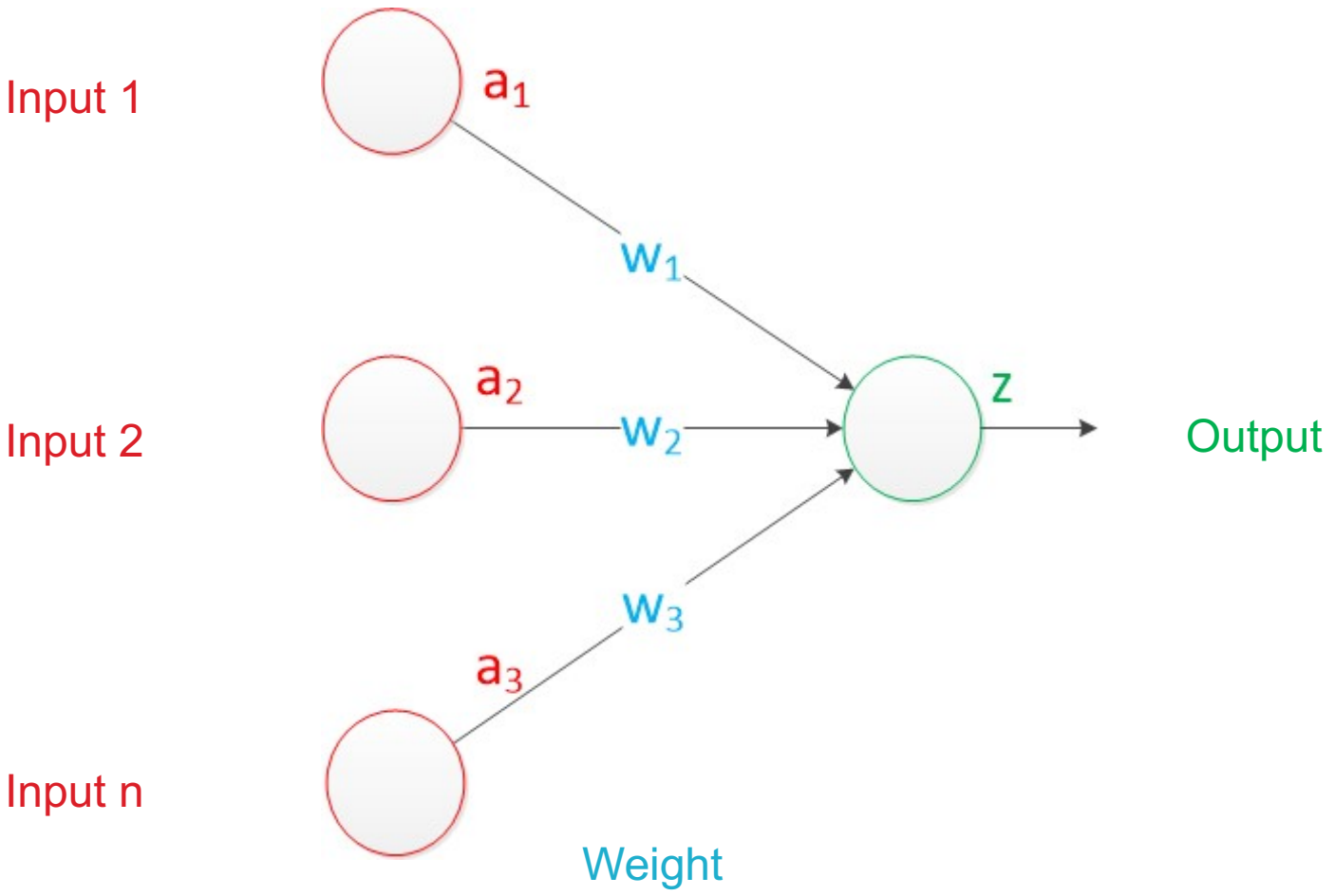
Dying ReLU Problem

Common activation functions

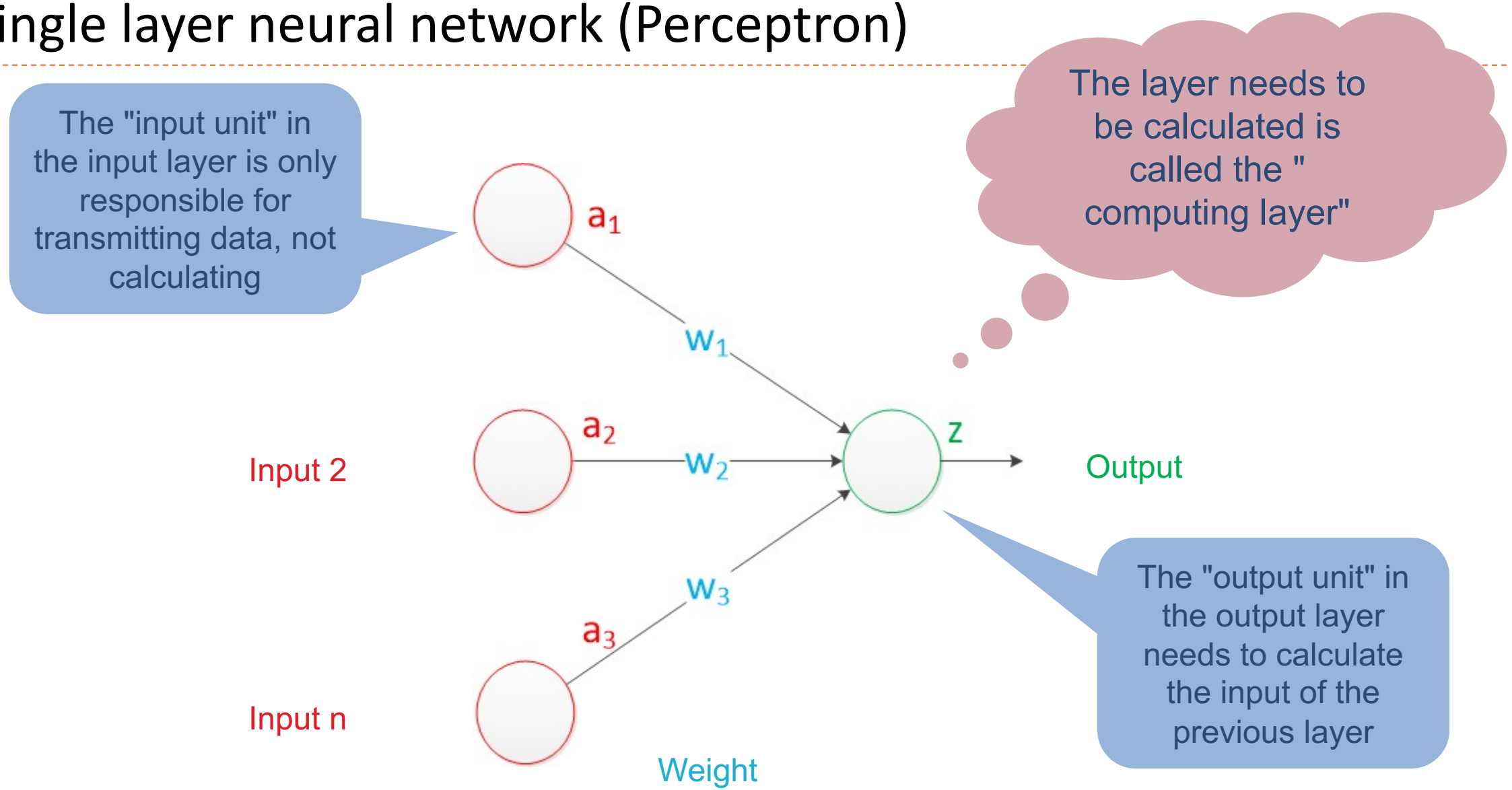
Swish function $\text{swish}(x) = x\sigma(\beta x)$



Single layer neural network (Perceptron)

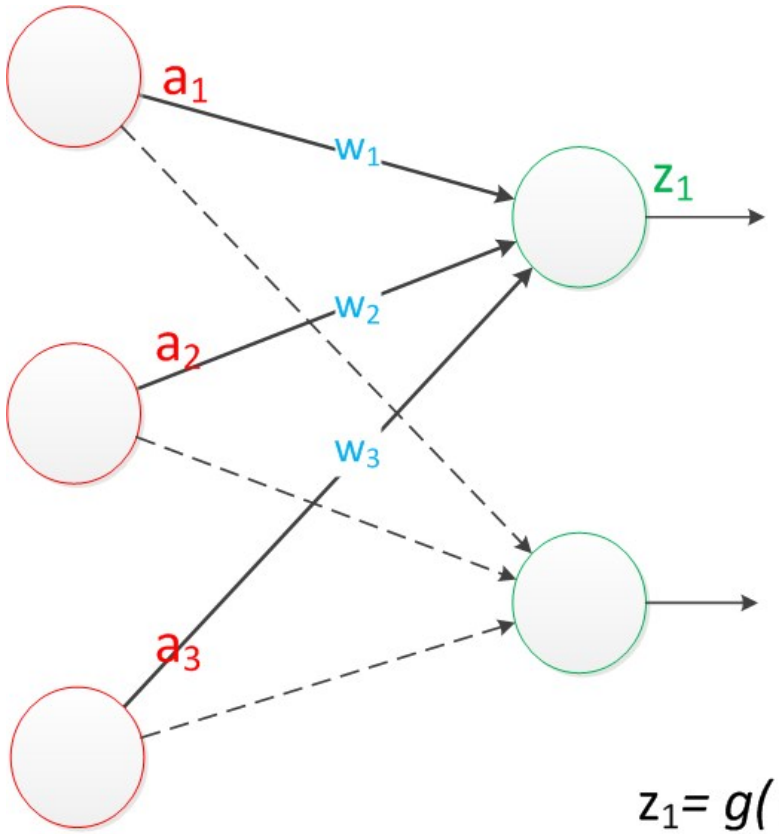


Single layer neural network (Perceptron)



Single layer neural network (Perceptron)

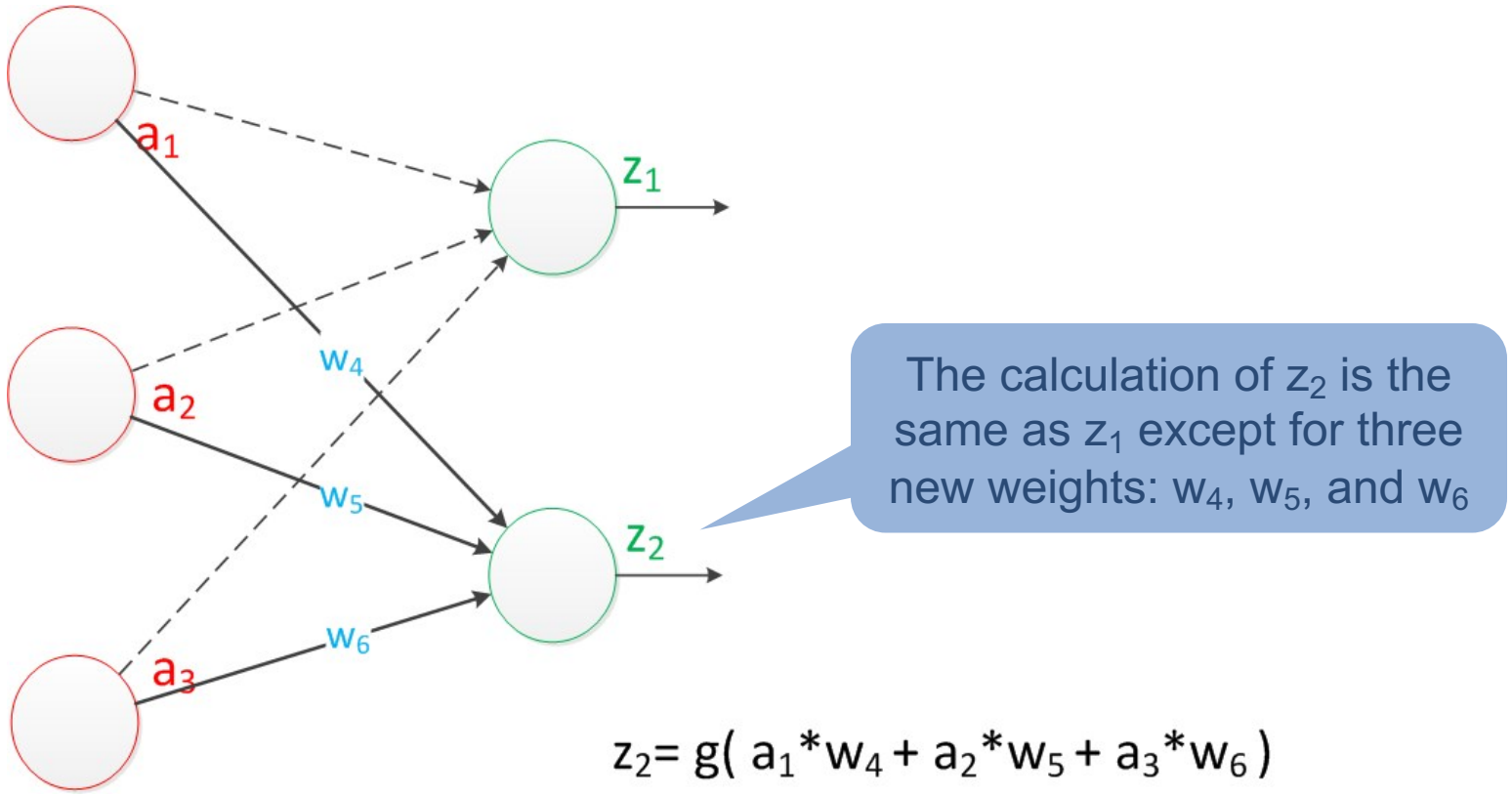
- A network with one computing layer is called a "single layer neural network"



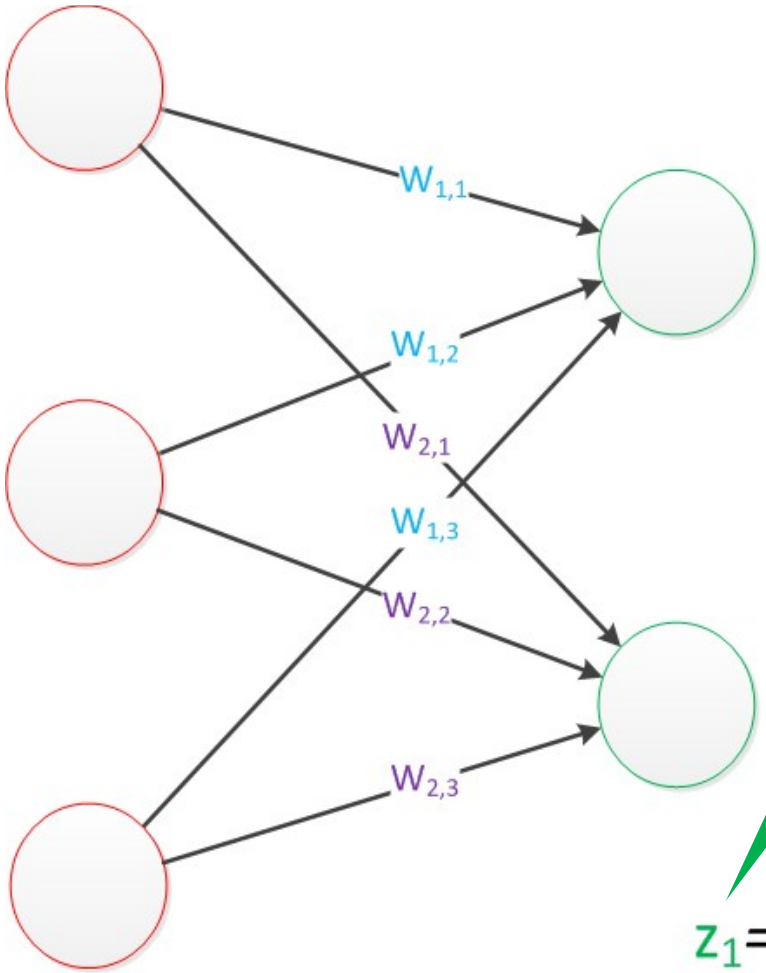
$$z_1 = g(a_1 * w_1 + a_2 * w_2 + a_3 * w_3)$$

Single layer neural network (Perceptron)

- A network with one computing layer is called a "single layer neural network"



Single layer neural network (Perceptron)



$$g(\mathbf{W} * \mathbf{a}) = \mathbf{z}$$

$z = [z_1, z_2]^T$

$a = [a_1, a_2, a_3]^T$

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

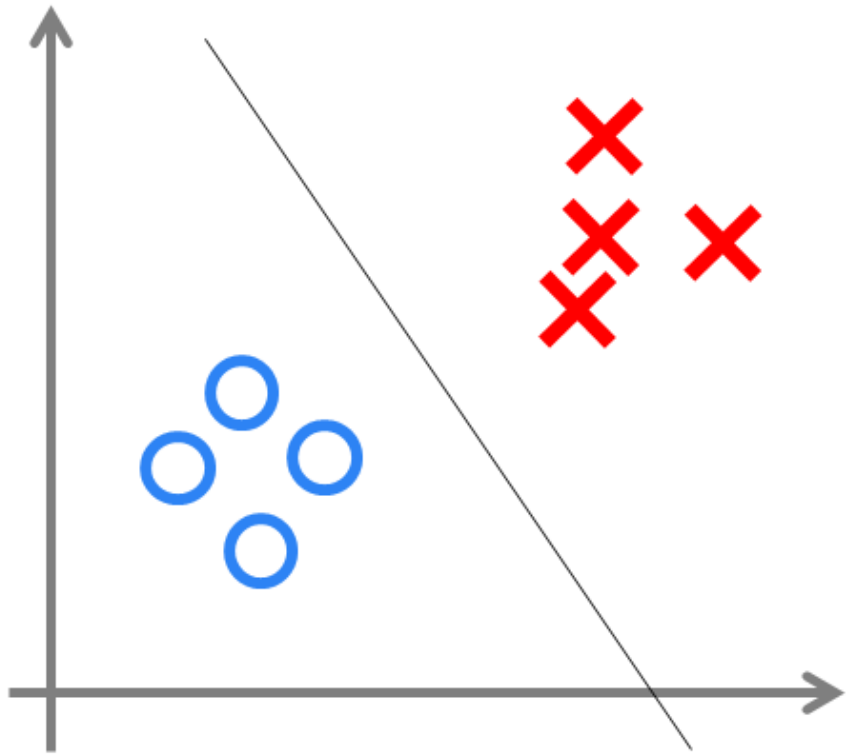
$$z_1 = g(a_1 * w_{1,1} + a_2 * w_{1,2} + a_3 * w_{1,3})$$

$$z_2 = g(a_1 * w_{2,1} + a_2 * w_{2,2} + a_3 * w_{2,3})$$

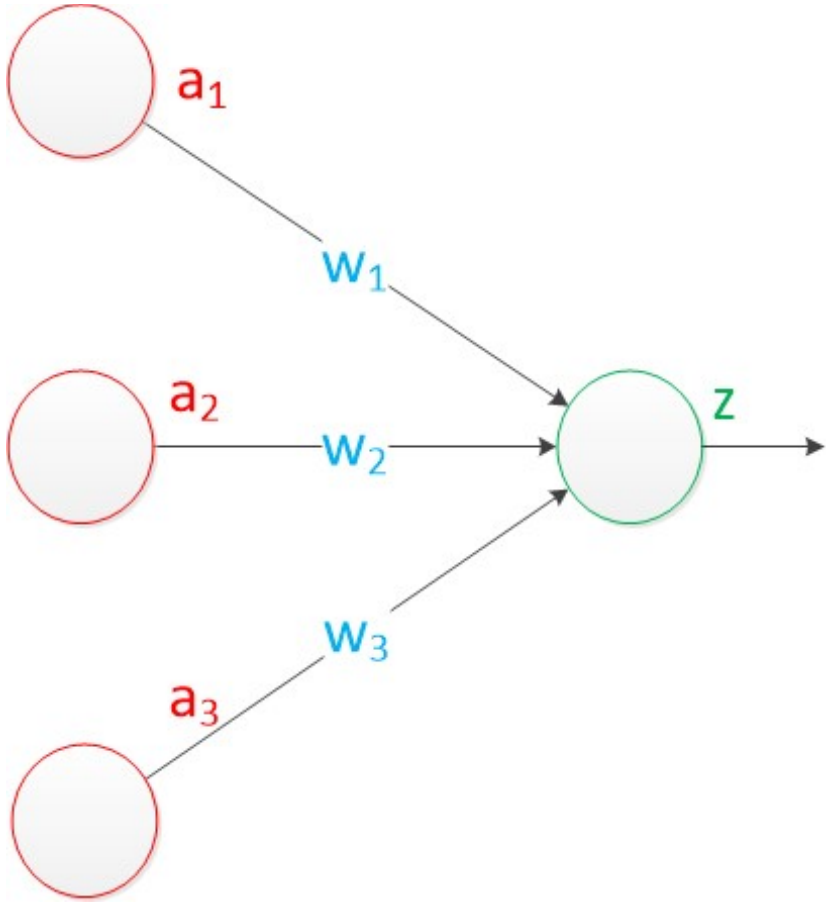
Single layer neural network (Perceptron)

- The weights (e.g., w_{ij}) in the perceptron are obtained throug training

- The perceptron is a logistic regression model that can do lin classification tasks



Single layer neural network (Perceptron)



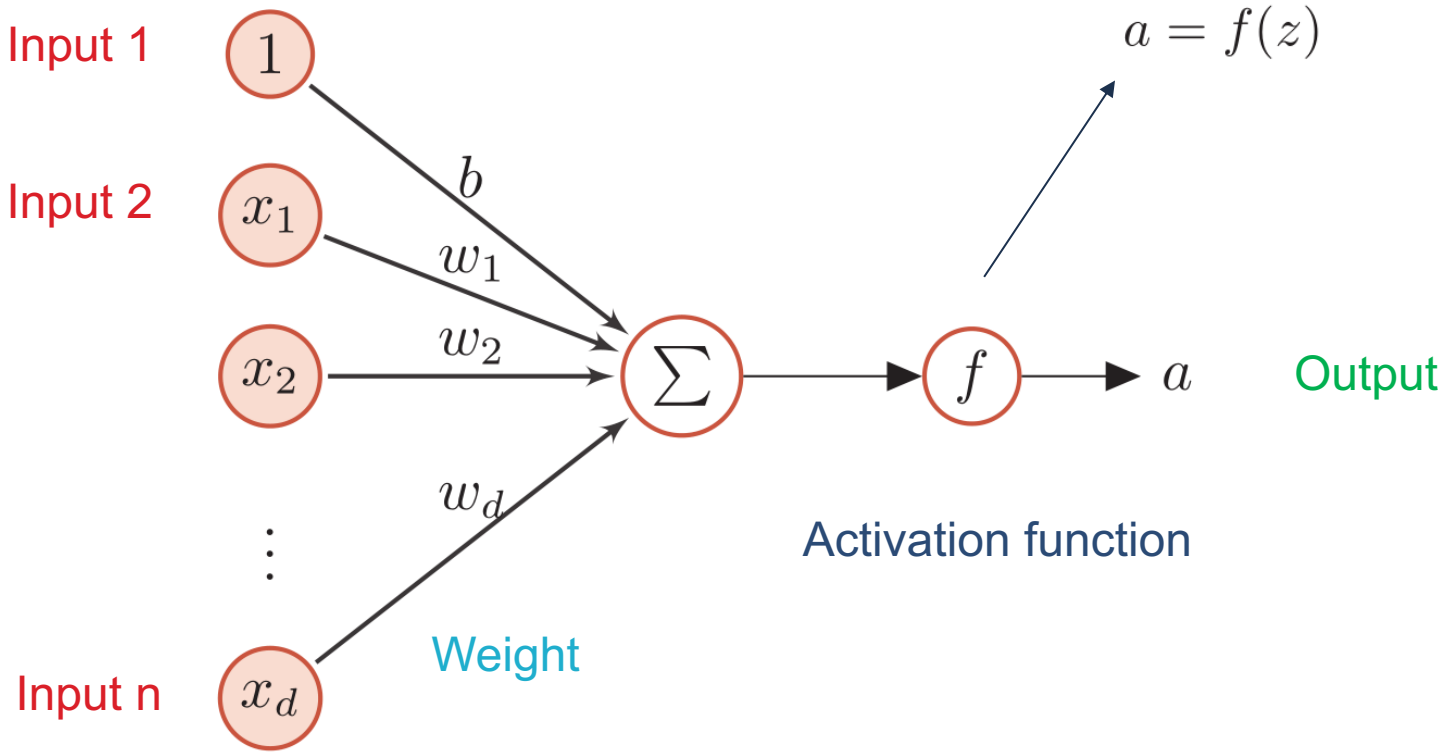
$$z = \sum a_i w_i$$

Let $a_1 = b$, $w_1 = 1$

$a_k = a_{k+1}$, $w_k = w_{k+1}$

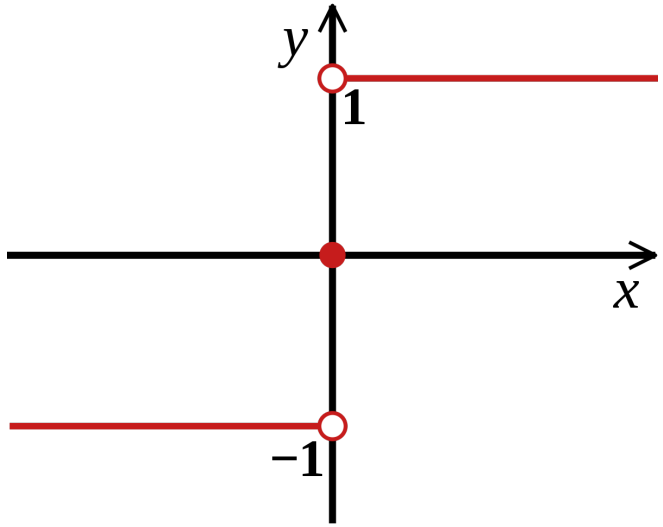
$$z = \sum a_i w_i + b$$

Single layer neural network (Perceptron)



Sgn function

$$f(input) = \begin{cases} 1, input \geq 0 \\ 0, input < 0 \end{cases}$$



Single layer neural network (Perceptron): AND

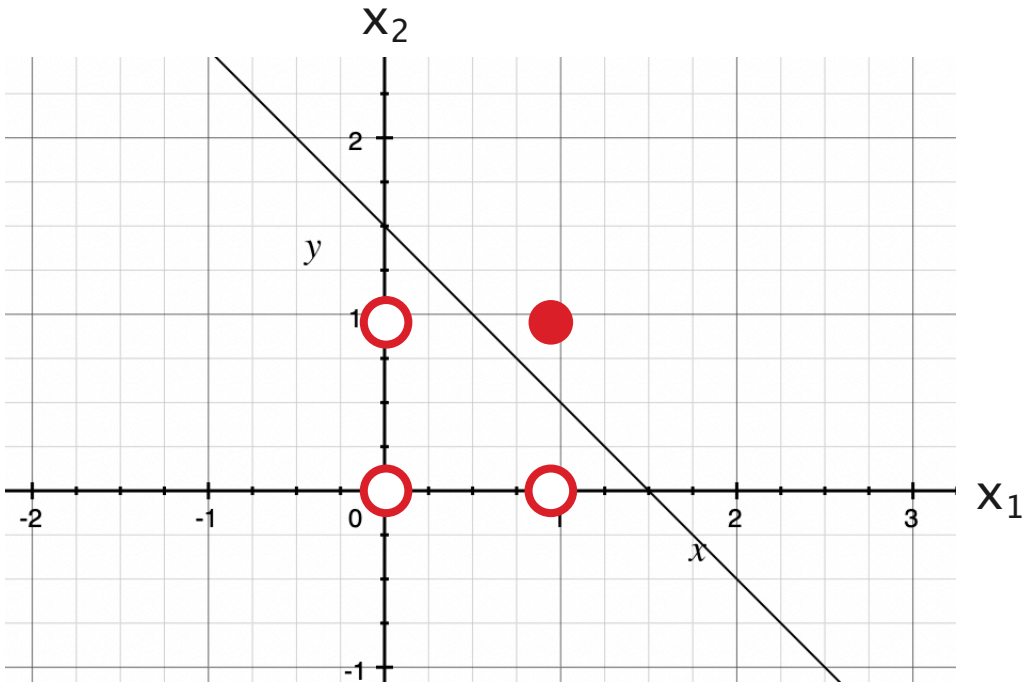
$$z = y(x_1, x_2) = f(\omega_1 x_1 + \omega_2 x_2 - \theta)$$

$$f(input) = \begin{cases} 1, input \geq 0 \\ 0, input < 0 \end{cases}$$

x_1	x_2	$x_1 \wedge x_2$
1	1	1
1	0	0
0	1	0
0	0	0
AND GATE		

ω1 = ω2 = 1, θ = 1.5

y = f(1* x_1 + 1* x_2 – 1.5)



1* x_1 + 1* x_2 – 1.5 = 0

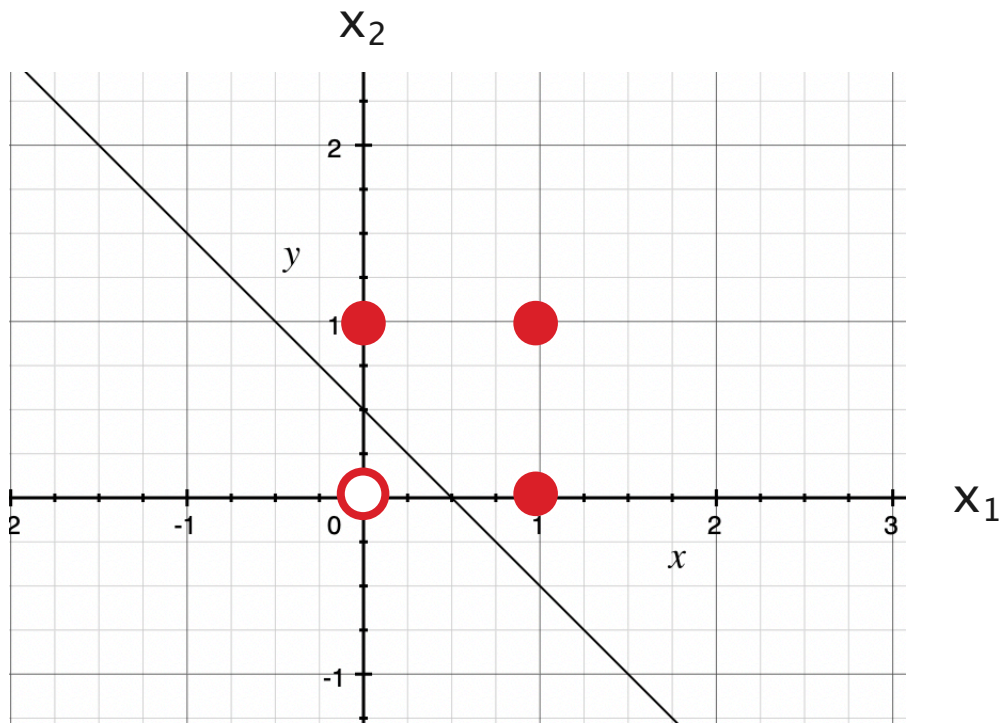
Single layer neural network (Perceptron): OR

$$z = y(x_1, x_2) = f(\omega_1 x_1 + \omega_2 x_2 - \theta) \qquad f(input) = \begin{cases} 1, input \geq 0 \\ 0, input < 0 \end{cases}$$

x_1	x_2	$x_1 \vee x_2$
1	1	1
1	0	1
0	1	1
0	0	0
OR GATE		

ω1 = ω2 = 1, θ = 0.5

y = f(1* x_1 + 1* x_2 – 0.5)



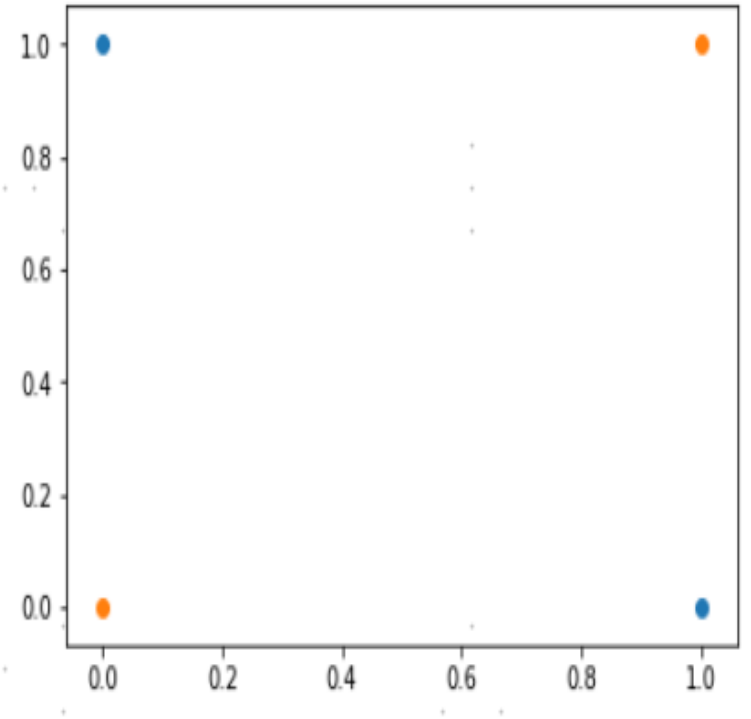
1* x_1 + 1* x_2 – 0.5 = 0

Single layer neural network (Perceptron): XOR

$$z = y(x_1, x_2) = f(\omega_1 x_1 + \omega_2 x_2 - \theta)$$

$$f(input) = \begin{cases} 1, input \geq 0 \\ 0, input < 0 \end{cases}$$

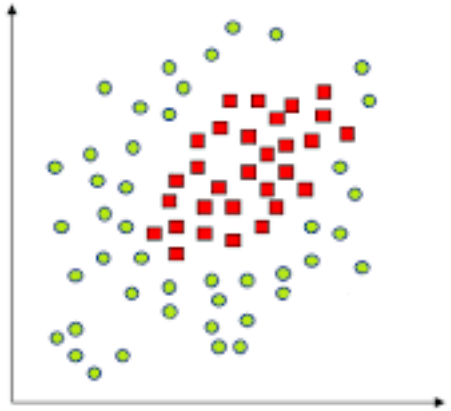
(x1, x2)	y
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	0



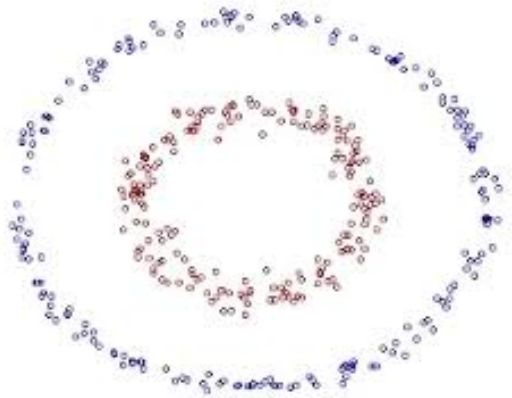
Linear inseparable

Linear inseparable

- We cannot use a straight line to divide the two types of data

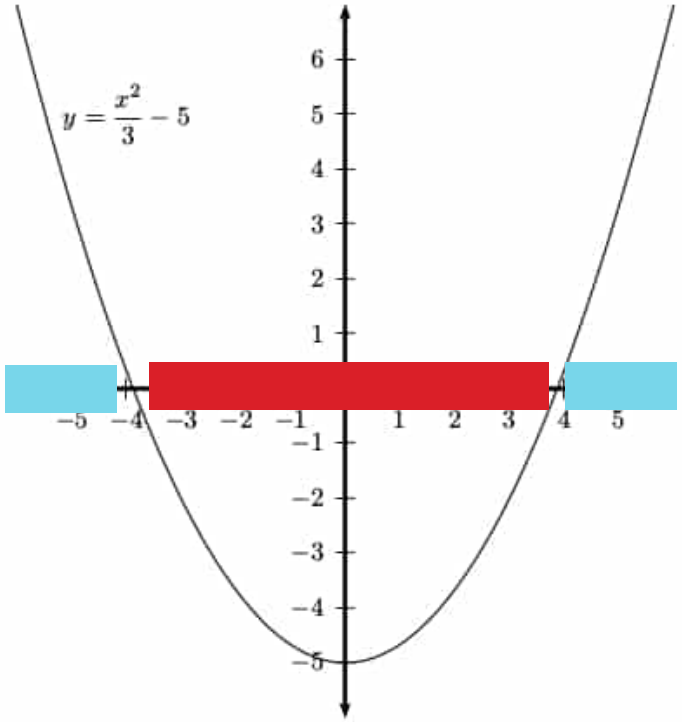


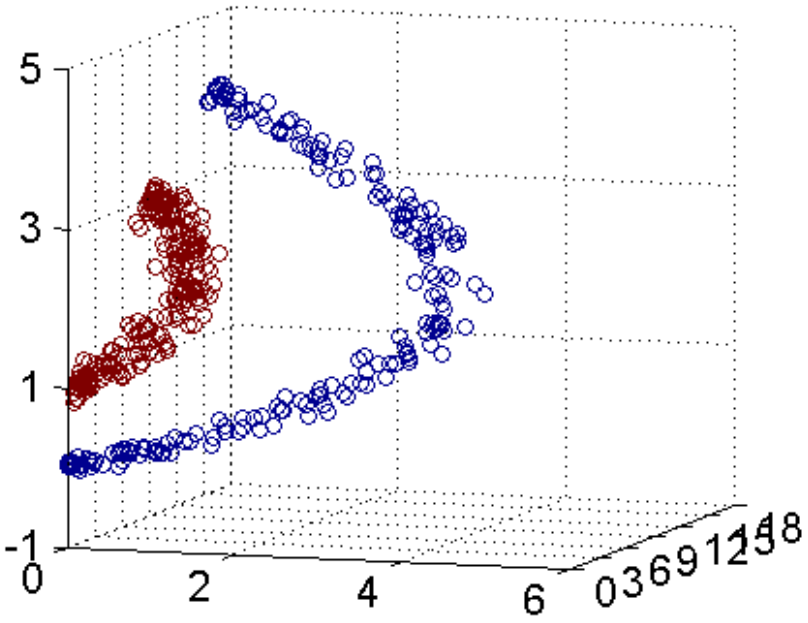
- Such data sets are very common in practic applications, such as: face images, text documents



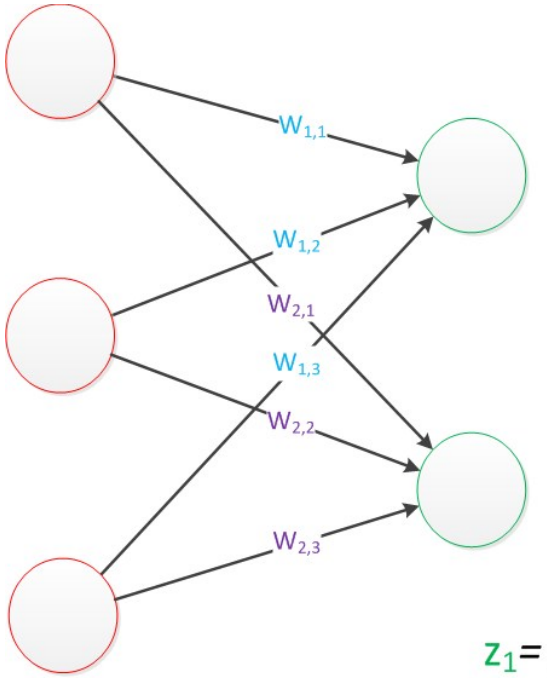
Linear inseparable

- The solution to linear inseparability is to use the kernel function mapping to high-dimensional space



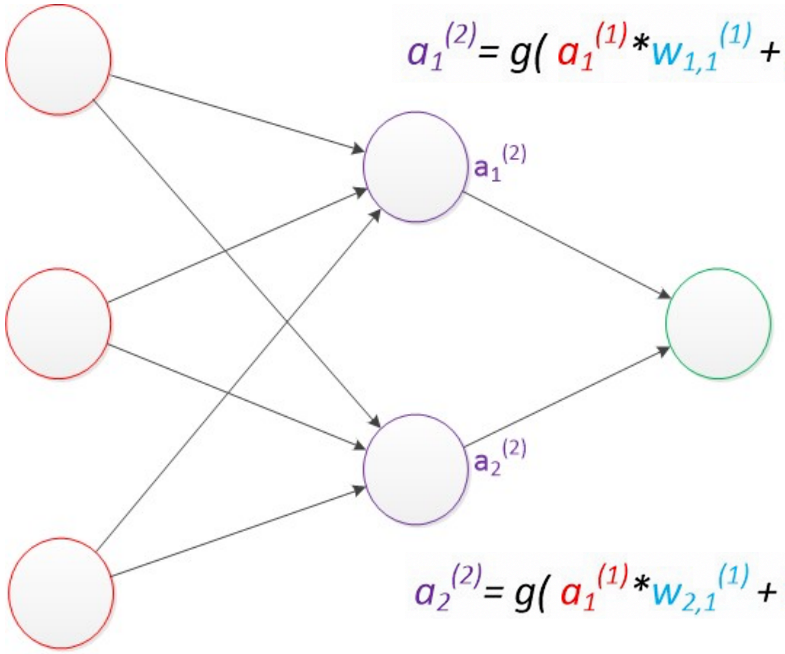


Two-layer neural network



$z_1 = g(a_1 * w_{1,1} + a_2 * w_{1,2} + a_3 * w_{1,3})$

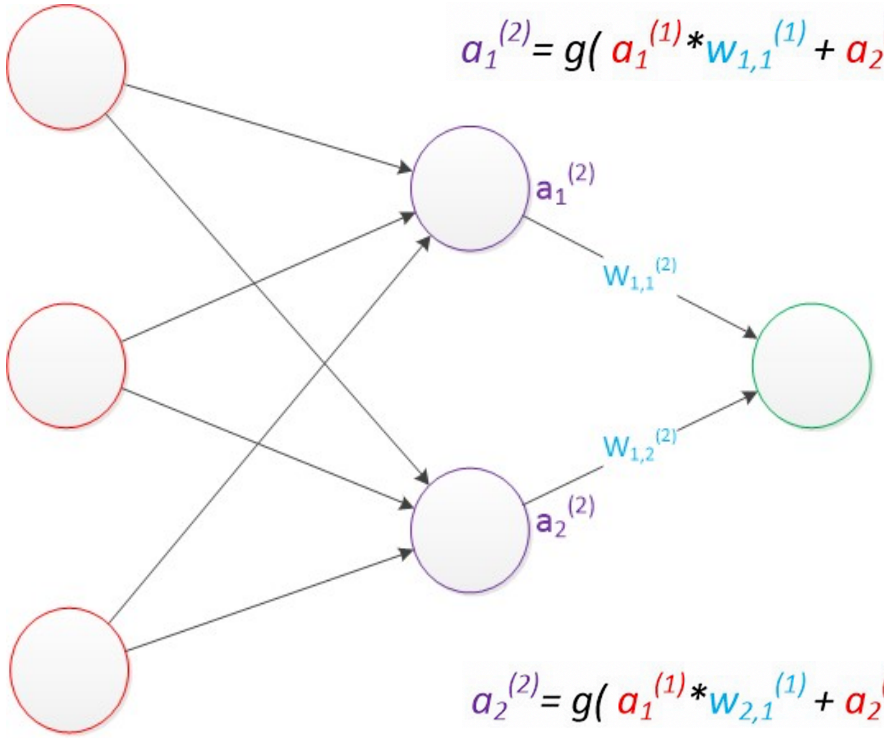
$z_2 = g(a_1 * w_{2,1} + a_2 * w_{2,2} + a_3 * w_{2,3})$



$a_1^{(2)} = g(a_1^{(1)} * w_{1,1}^{(1)} + a_2^{(1)} * w_{1,2}^{(1)} + a_3^{(1)} * w_{1,3}^{(1)})$

$a_2^{(2)} = g(a_1^{(1)} * w_{2,1}^{(1)} + a_2^{(1)} * w_{2,2}^{(1)} + a_3^{(1)} * w_{2,3}^{(1)})$

Two-layer neural network

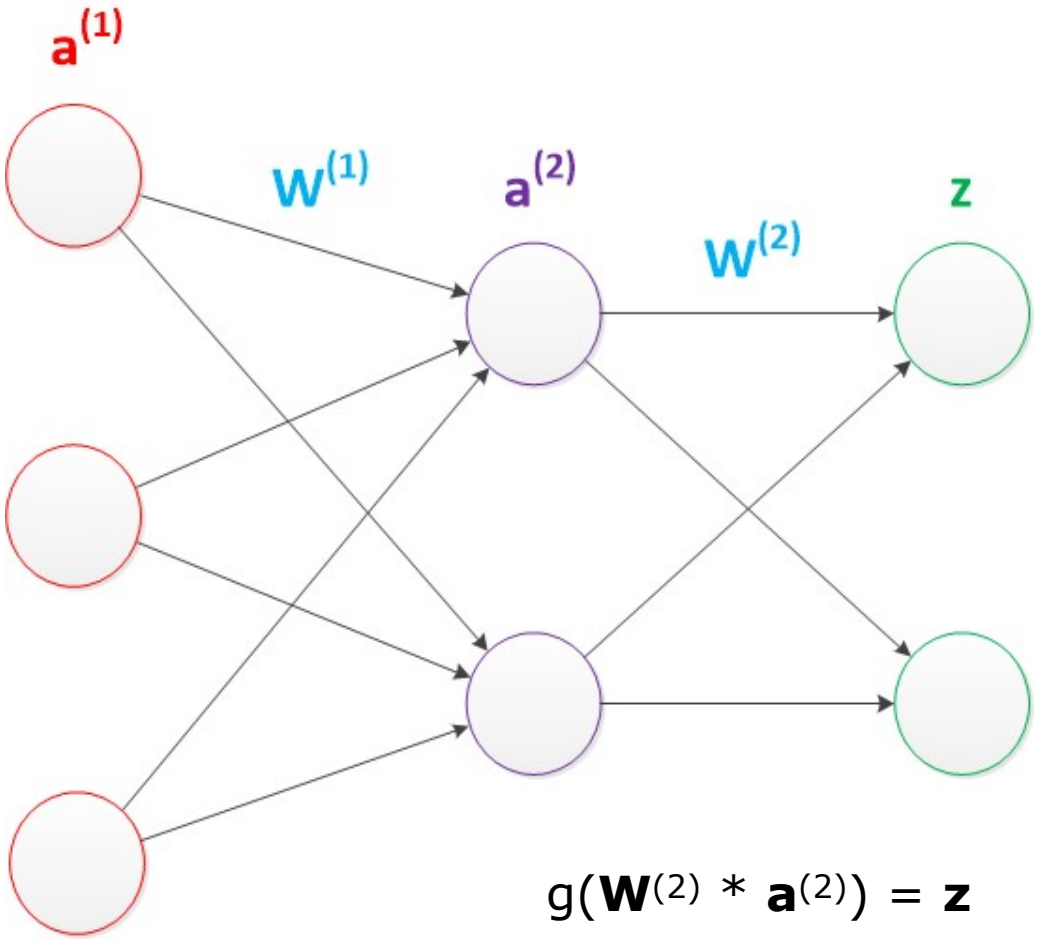


$$a_1^{(2)} = g(a_1^{(1)} * w_{1,1}^{(1)} + a_2^{(1)} * w_{1,2}^{(1)} + a_3^{(1)} * w_{1,3}^{(1)})$$

$$z = g(a_1^{(2)} * W_{1,1}^{(2)} + a_2^{(2)} * W_{1,2}^{(2)})$$

$$a_2^{(2)} = g(a_1^{(1)} * w_{2,1}^{(1)} + a_2^{(1)} * w_{2,2}^{(1)} + a_3^{(1)} * w_{2,3}^{(1)})$$

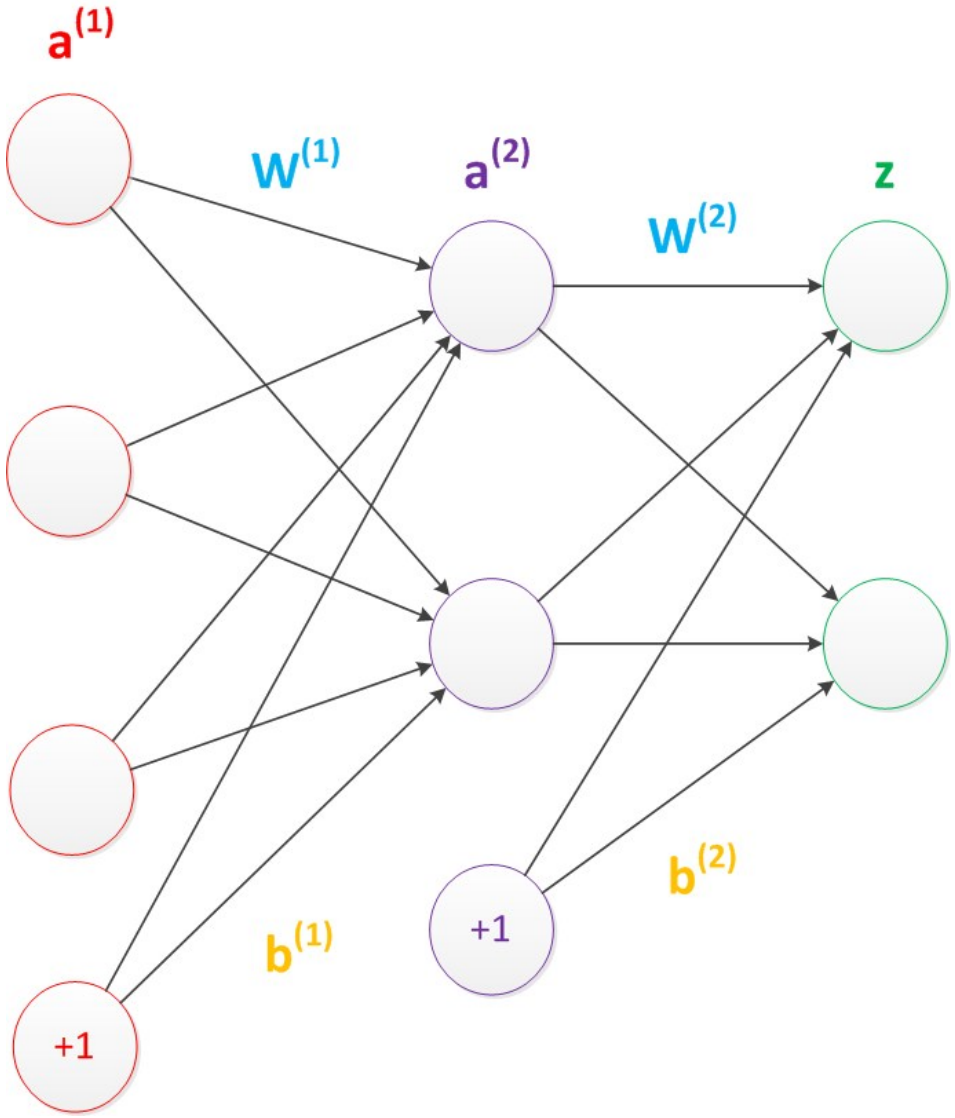
Two-layer neural network



$$g(\mathbf{W}^{(1)} * \mathbf{a}^{(1)}) = \mathbf{a}^{(2)}$$

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

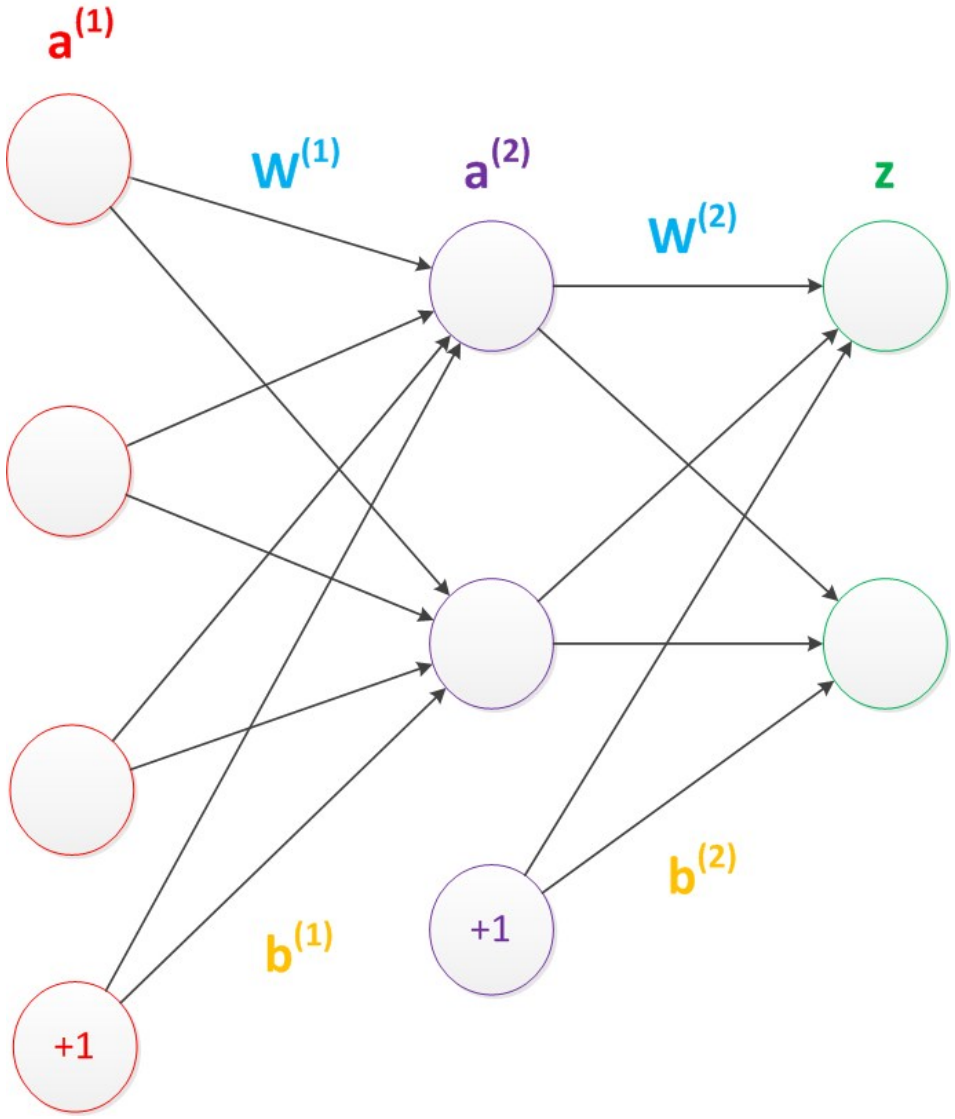
Two-layer neural network (including the bias node)



$$g(\mathbf{W}^{(1)} * \mathbf{a}^{(1)} + \mathbf{b}^{(1)}) = \mathbf{a}^{(2)}$$

$$g(\mathbf{W}^{(2)} * \mathbf{a}^{(2)} + \mathbf{b}^{(2)}) = \mathbf{z}$$

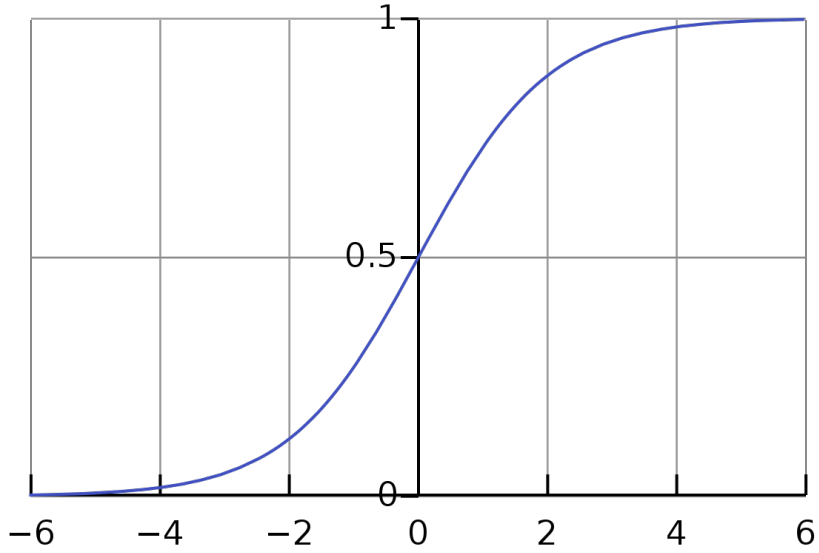
Two-layer neural network (including the bias node)



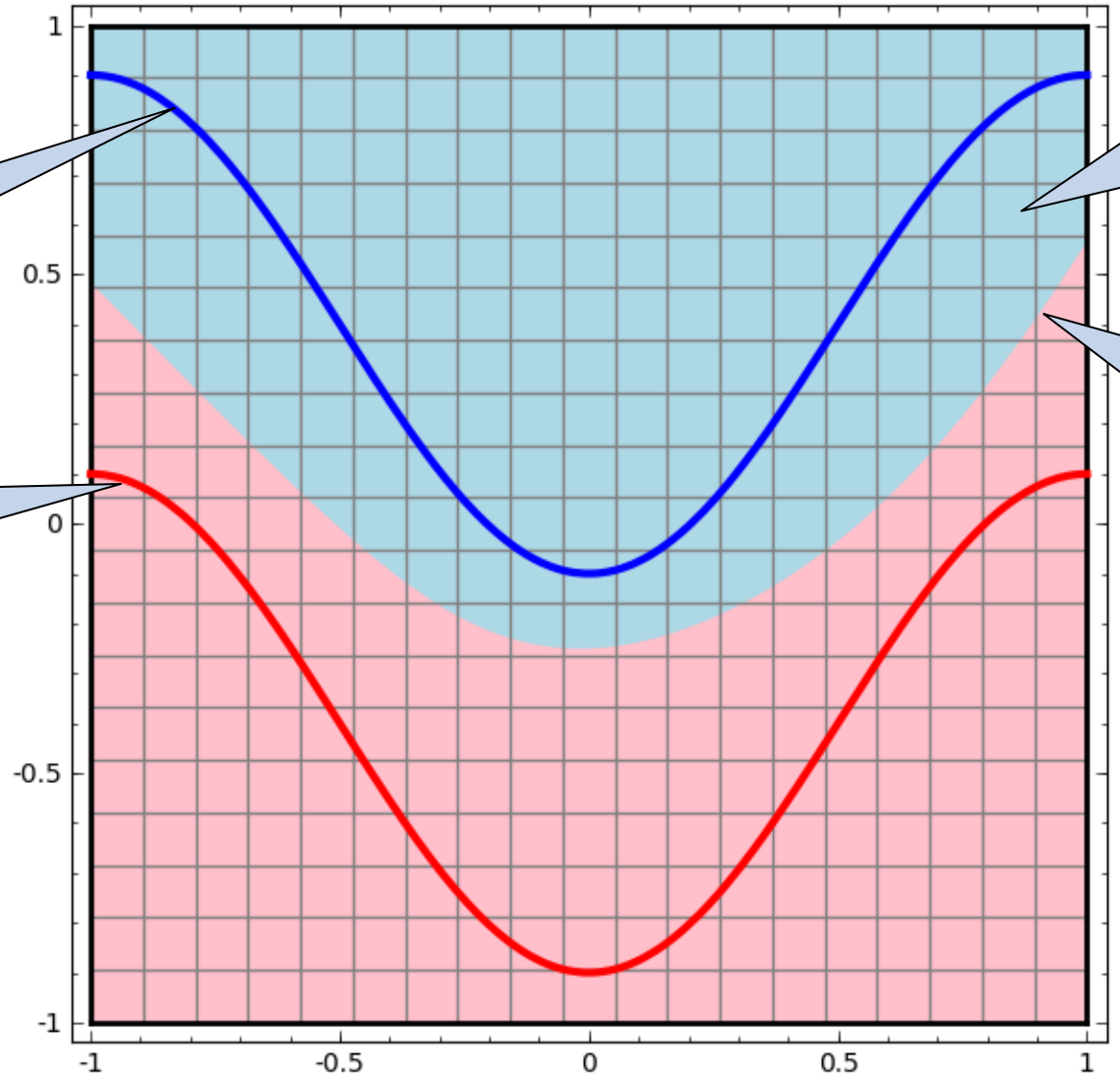
$g(\mathbf{W}^{(1)} * \mathbf{a}^{(1)} + \mathbf{b}^{(1)}) = \mathbf{a}^{(2)}$

$g(\mathbf{W}^{(2)} * \mathbf{a}^{(2)} + \mathbf{b}^{(2)}) = \mathbf{z}$

We used sigmoid function for g()



Non-linear classification task



The red and blue lines represent data

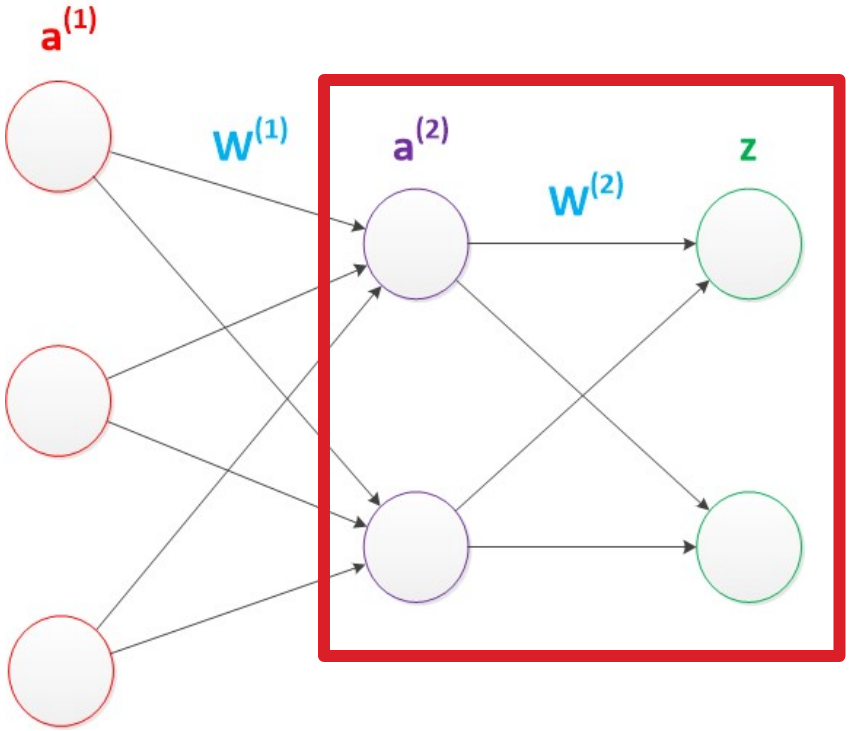
The red and blue lines represent data

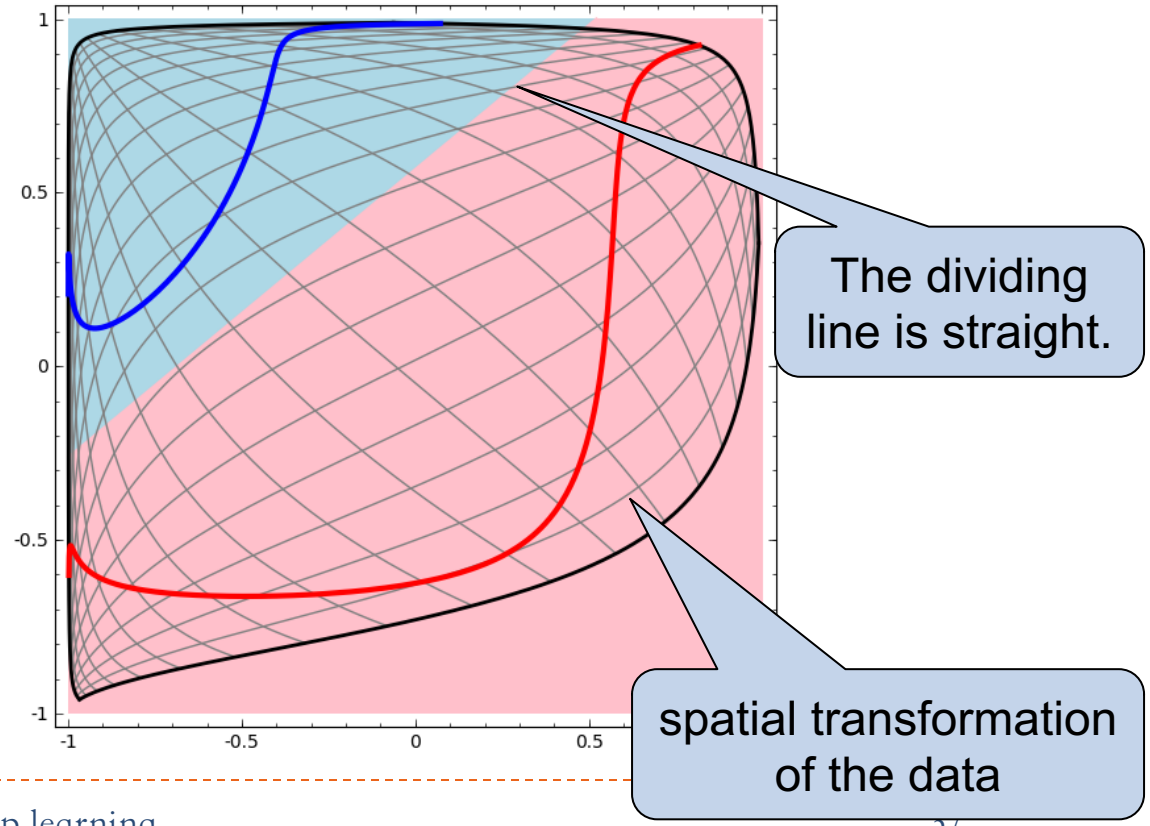
The red area and blue area represent the area divided by the neural network

The dividing line is generated by the NN. The curse is very smooth by NN.

Non-linear classification task

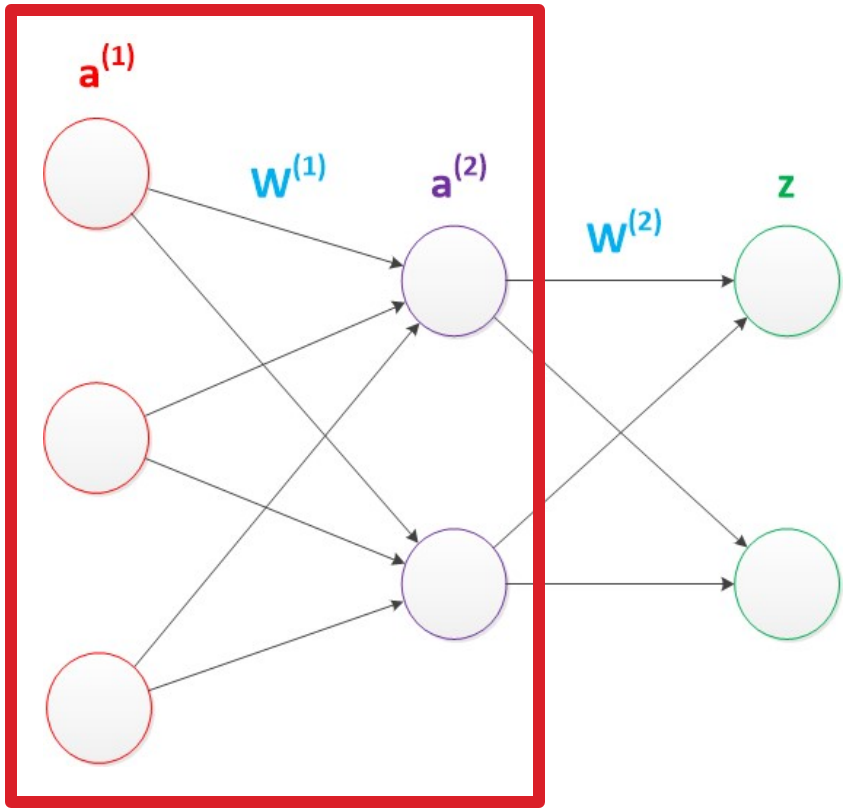
- Single-layer networks can only do linear classification tasks
- Why can a combination of two linear classification tasks be used for nonlinear classification tasks?

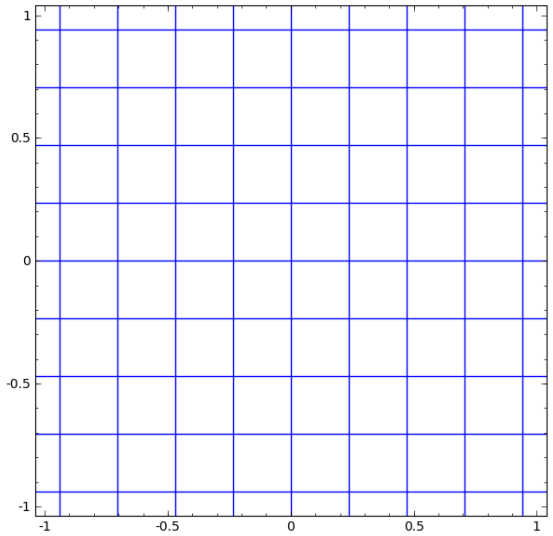


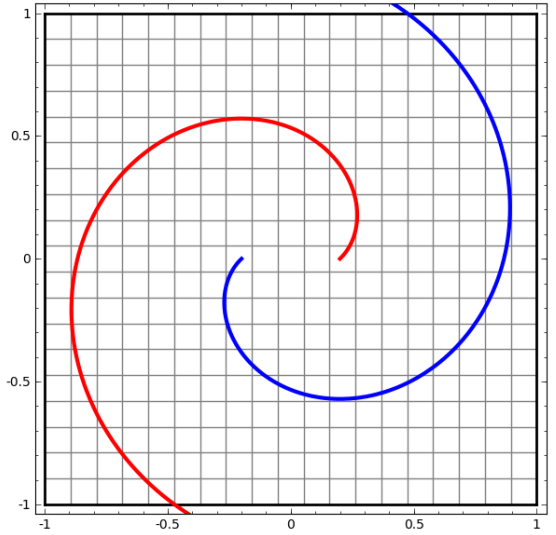


Non-linear classification task

- In the two-layer neural network, the hidden layer performs a **spatial transformation** on the original data so that it can be linearly classified

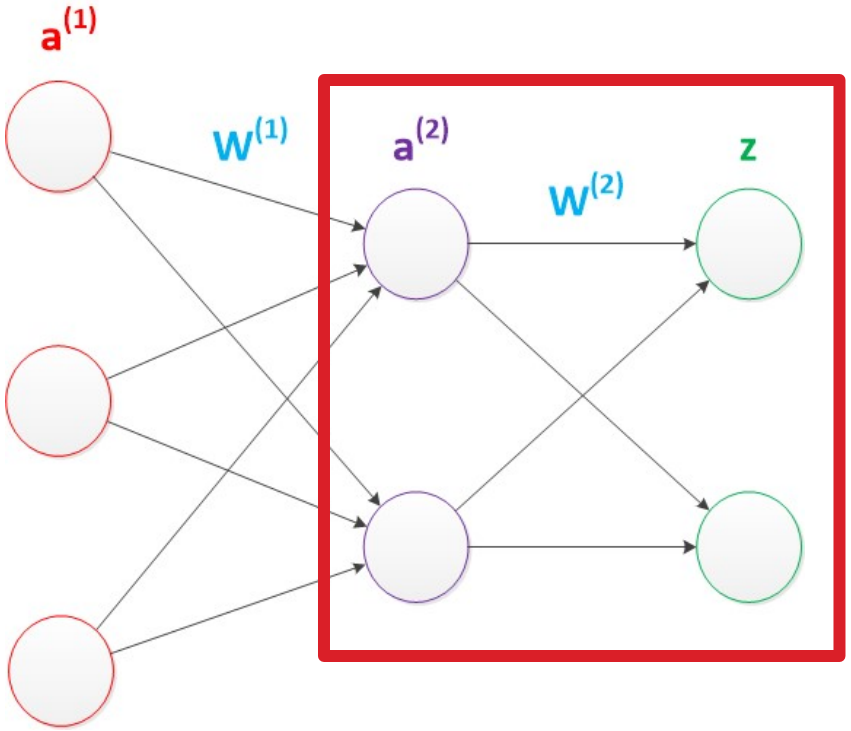


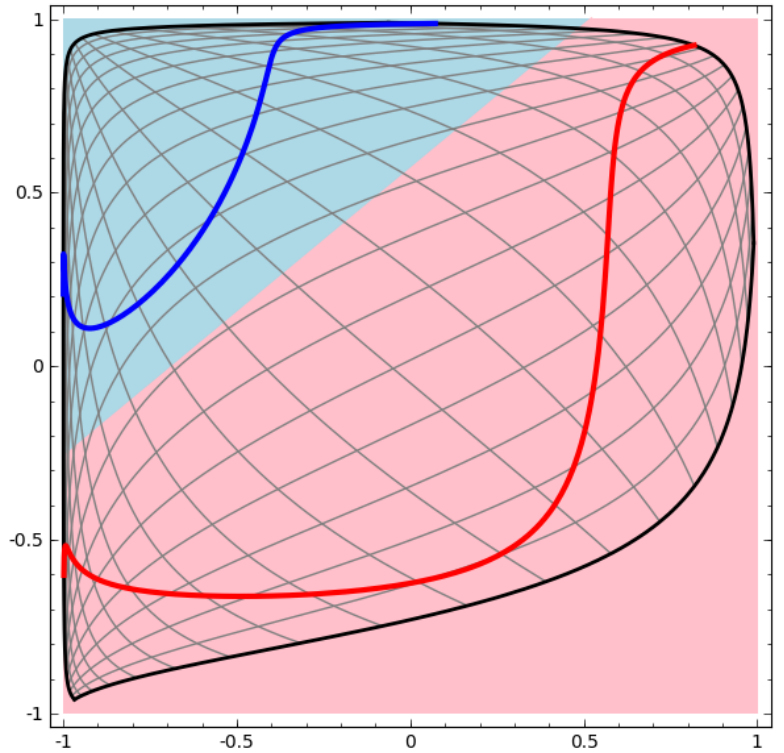




Non-linear classification task

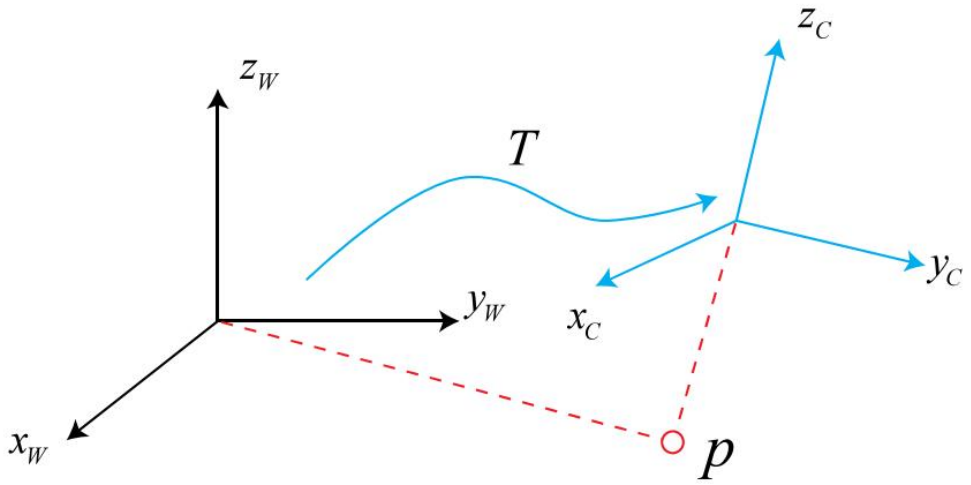
- The decision boundary of the output layer draws a linear classification boundary and classifies it



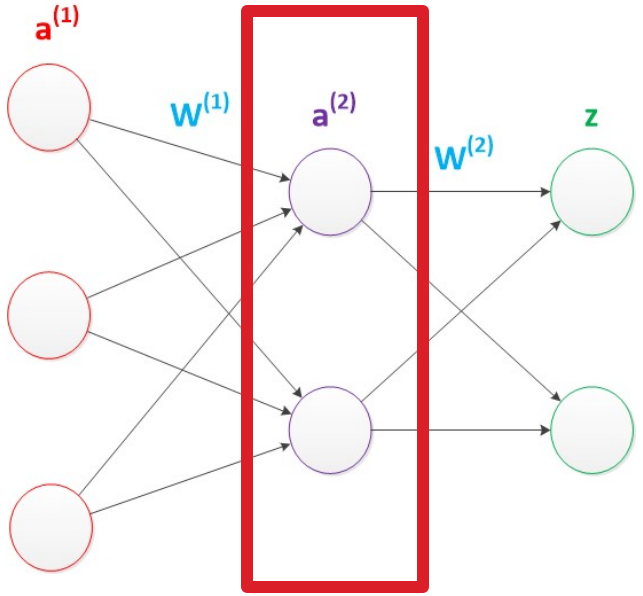


Non-linear classification task

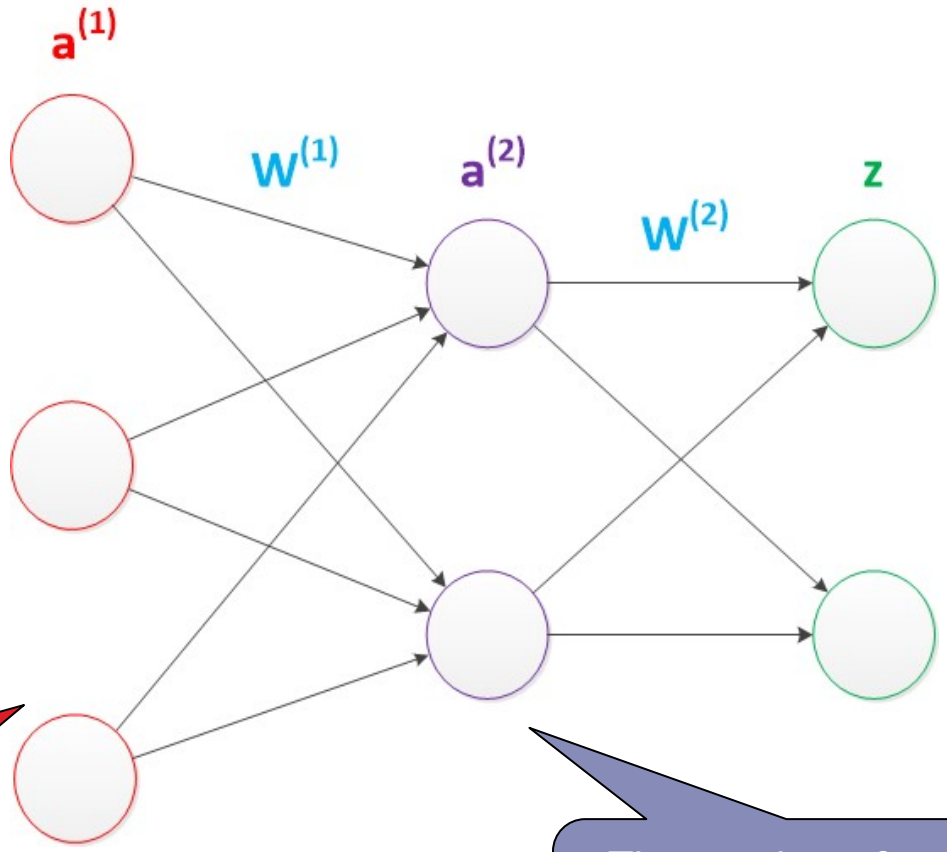
- Multiplying a matrix and a vector is essentially a transformation of the coordinate space of the vector



- The function of the parameter matrix of the hidden layer is to make the original coordinate space of the data from linear indivisible to linear separable



Design of Two-layer neural network



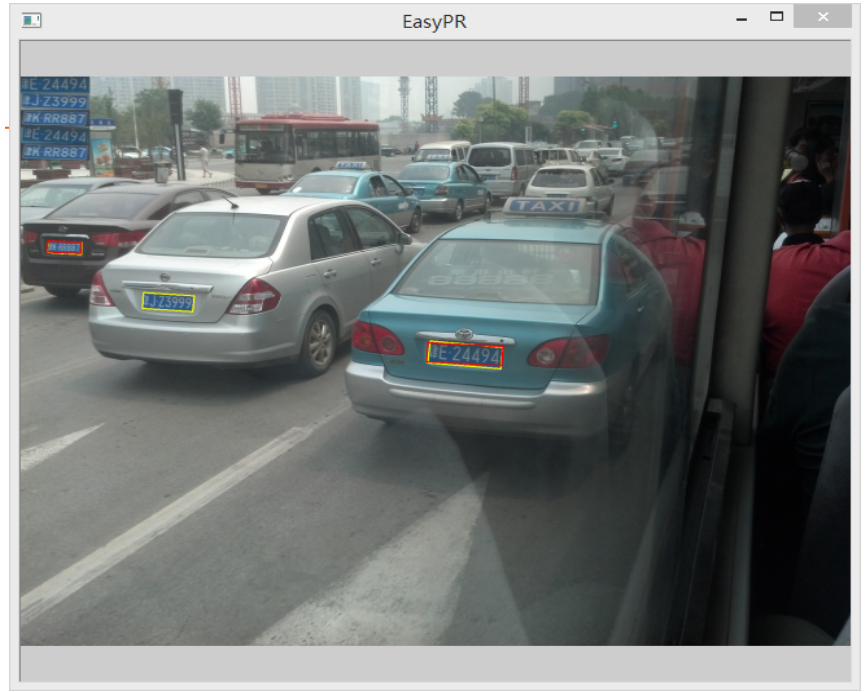
The number of nodes in the output layer must match the dimensionality of the target

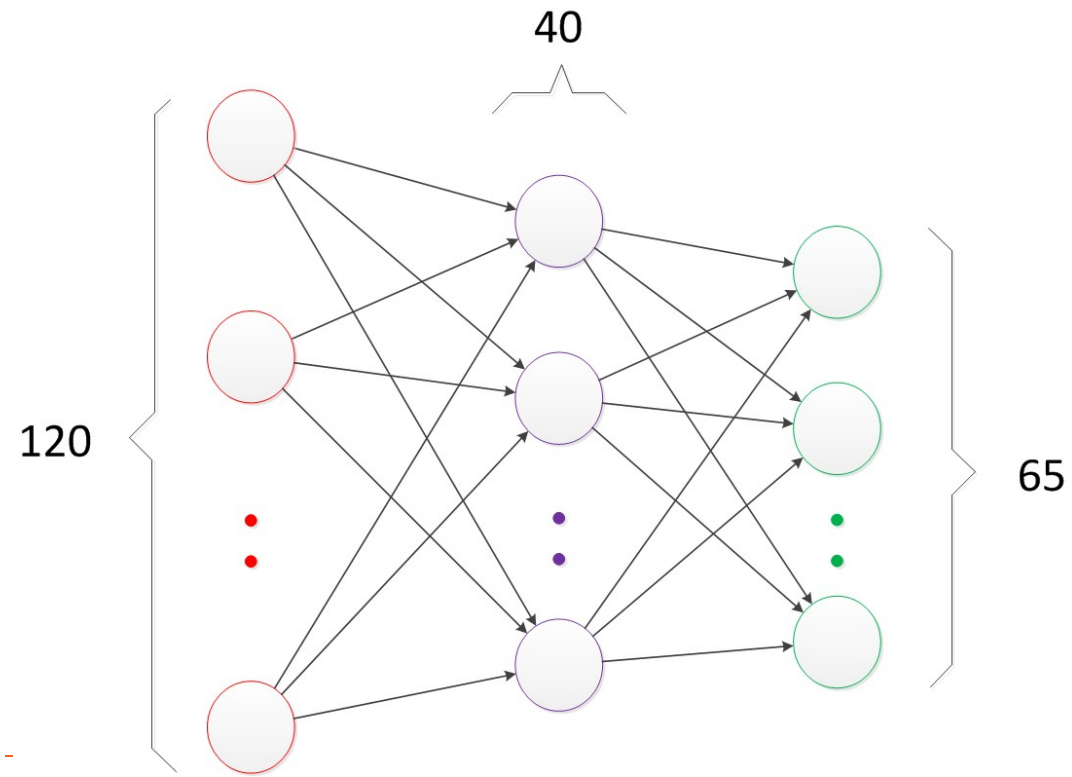
The number of nodes in the input layer needs to match the dimension of the feature

The number of nodes in the middle layer is specified by the designer

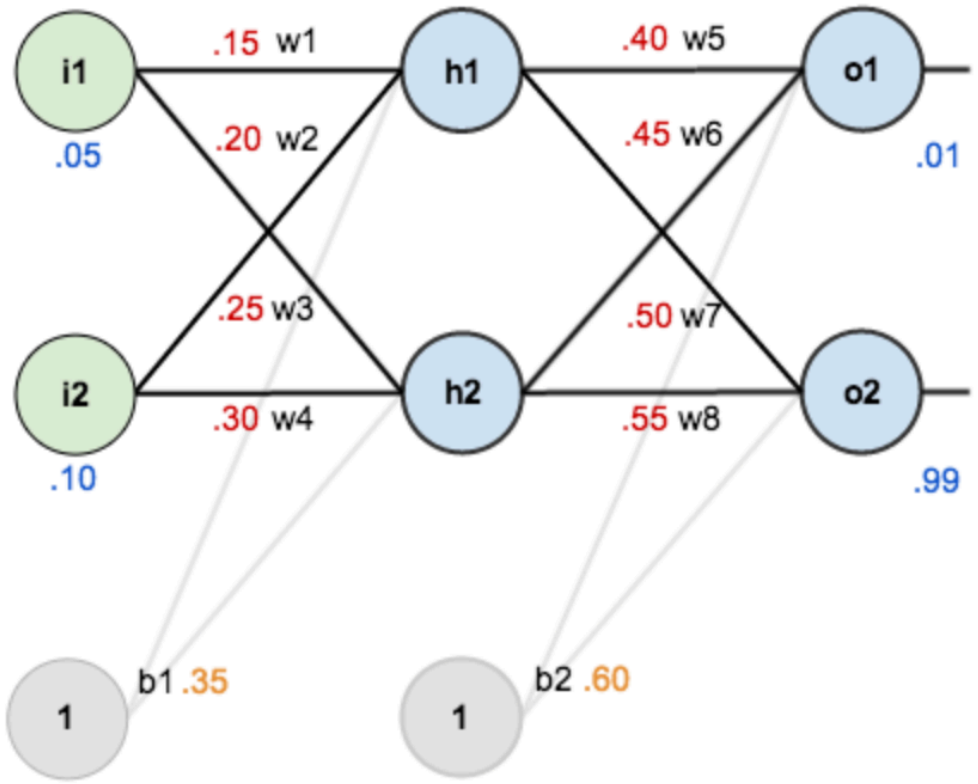
Design of Two-layer neural network

- EasyPR: the input is a 120-dimensional vector. The output is 65 types of text. They found the performance is the best when hidden layer number is 40





Back Propagation



- Suppose input data i_1=0.05, i_2=0.10
  - Initial weights w1=0.15, w2=0.20, w3=0.25, w4=0.30;
  - w5=0.40, w6=0.45, w7=0.50, w8=0.55;

h_1=0.15*0.5+0.1*0.25+0.35=0.45

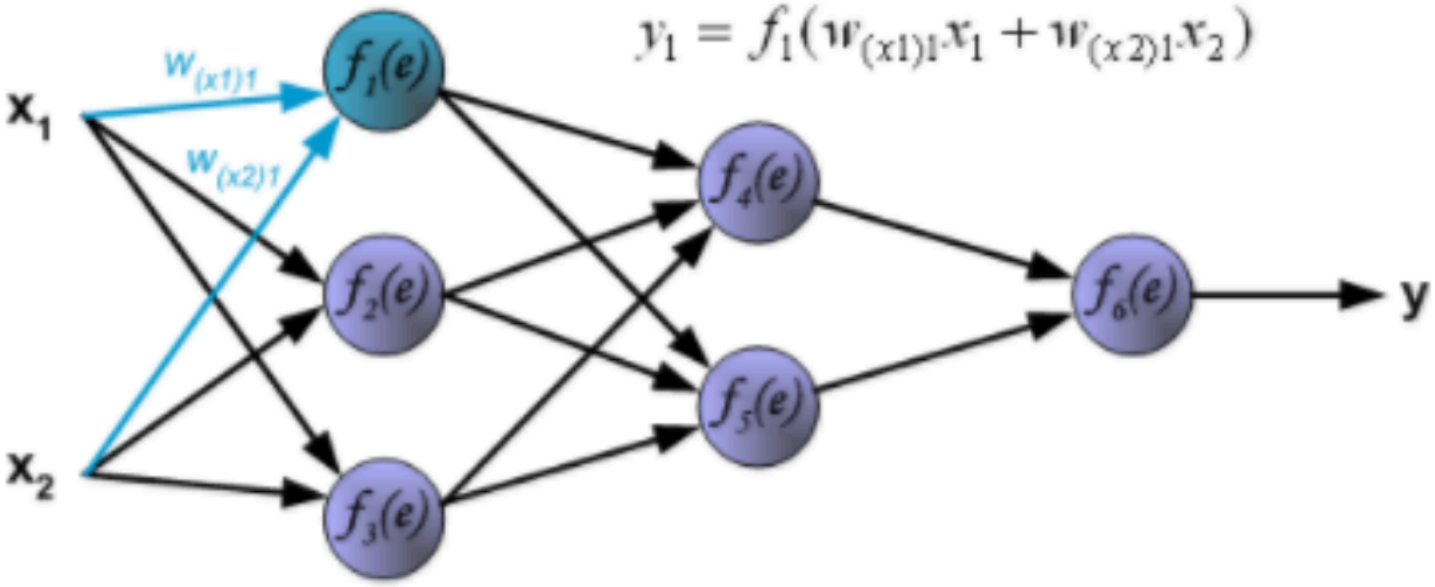
h_2=0.25*0.5+0.1*0.3+0.35=0.505

o_1=0.4*0.45+0.5*0.505+0.6=1.0325

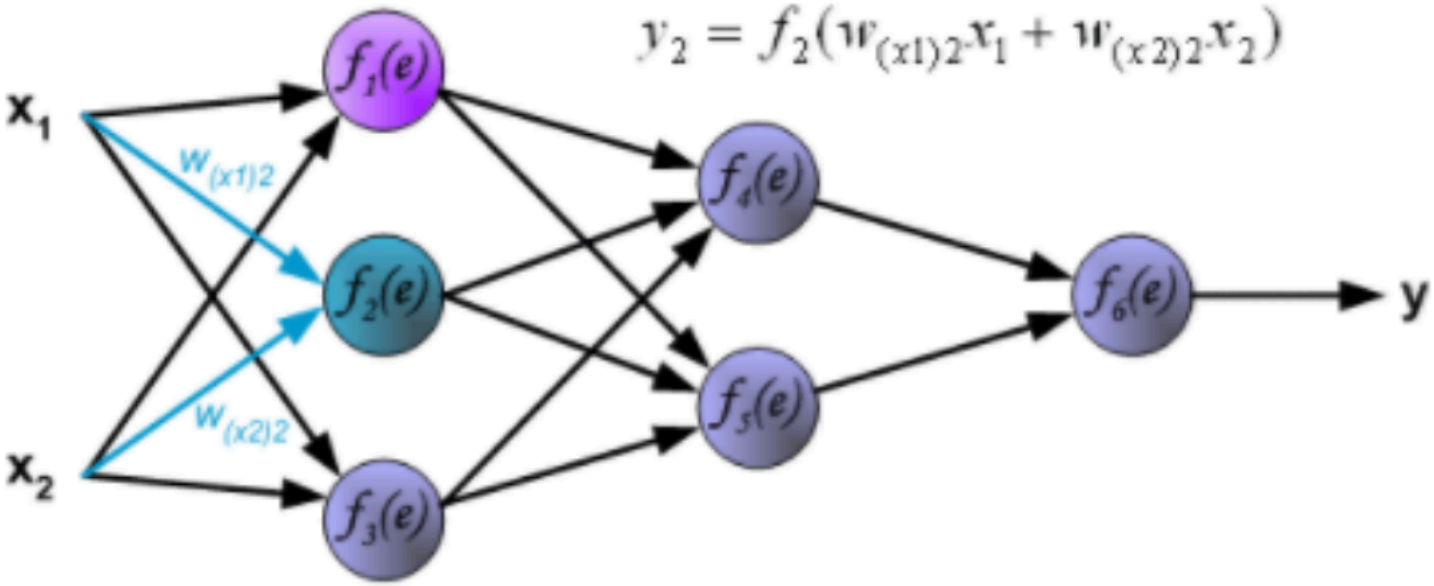
o_2=0.5*0.45+0.55*0.505+0.6=1.10275

- If we want o_1=0.01, o_2=0.99, how to adjust w_1~w_8

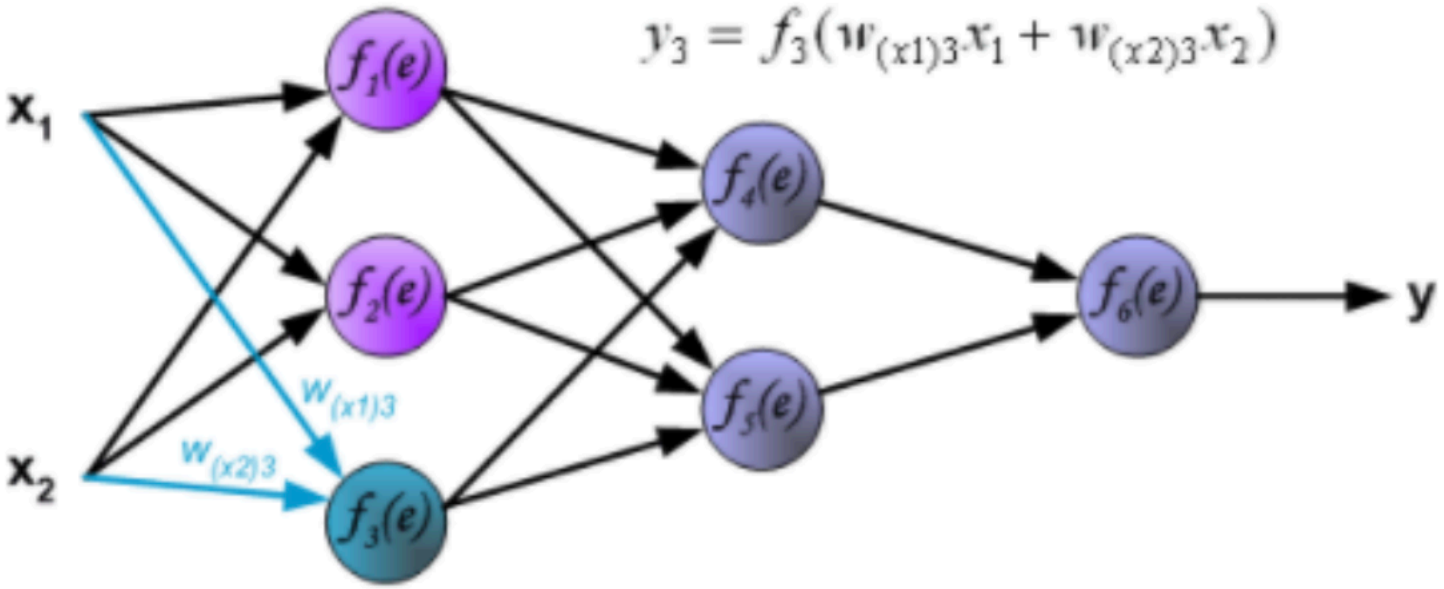
Back Propagation



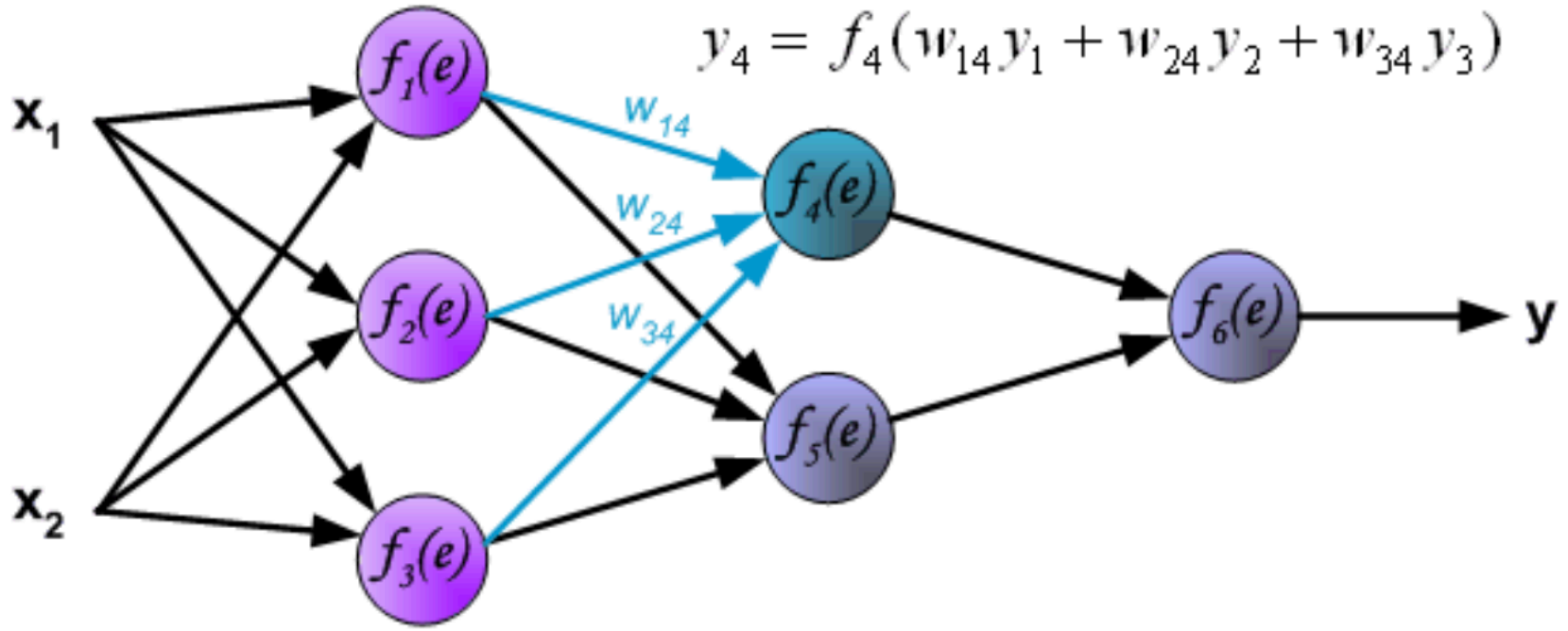
Back Propagation



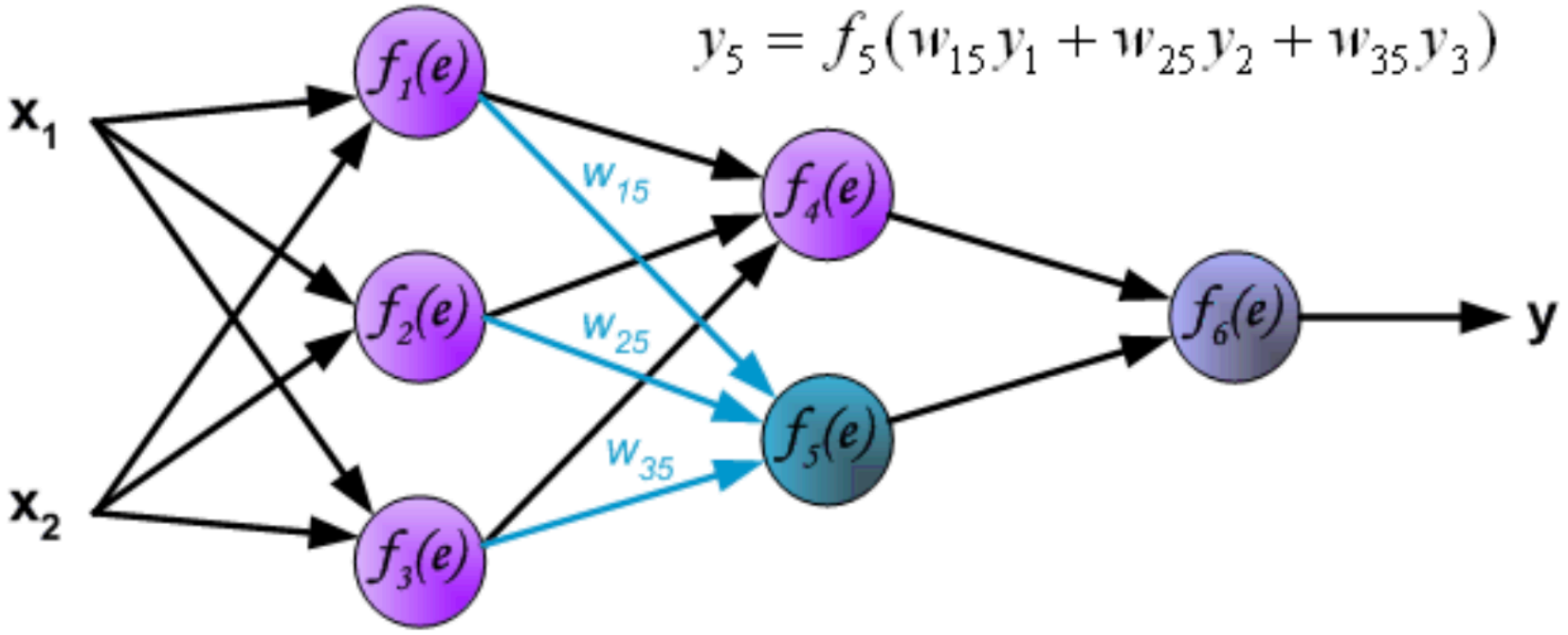
Back Propagation



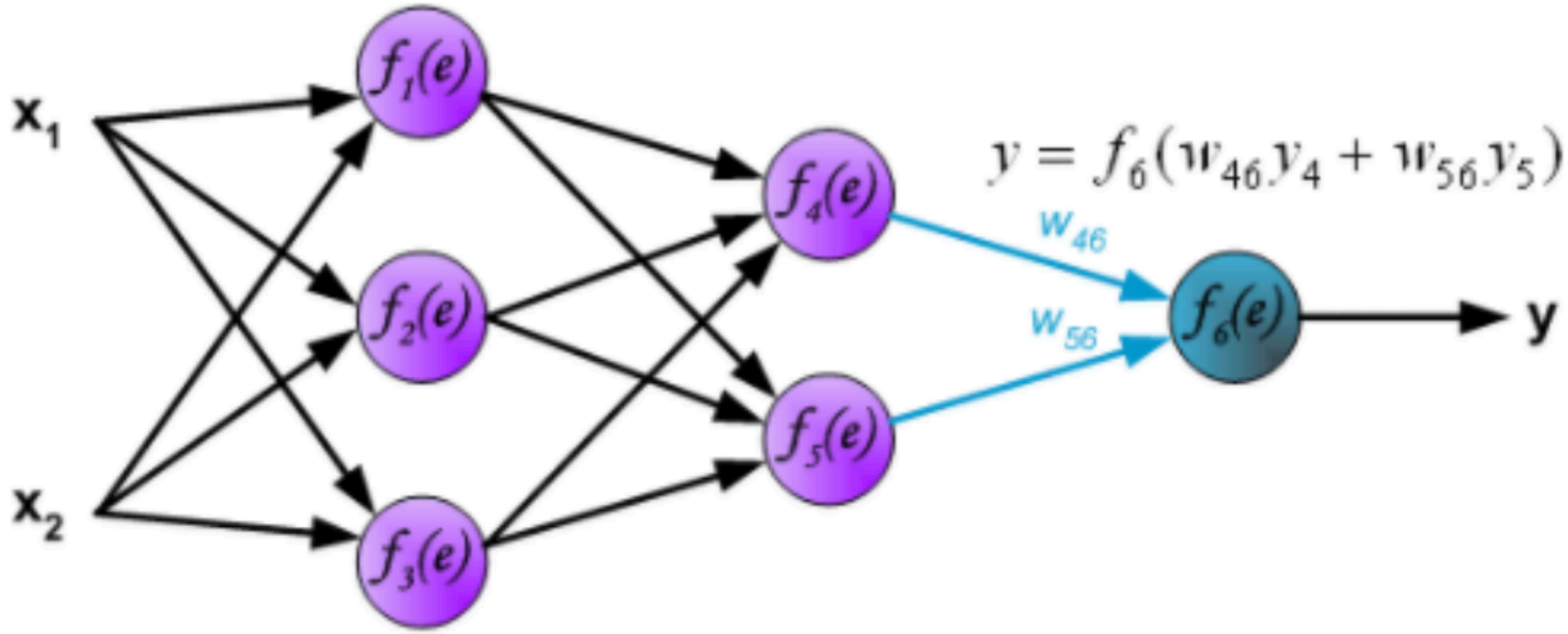
Back Propagation



Back Propagation

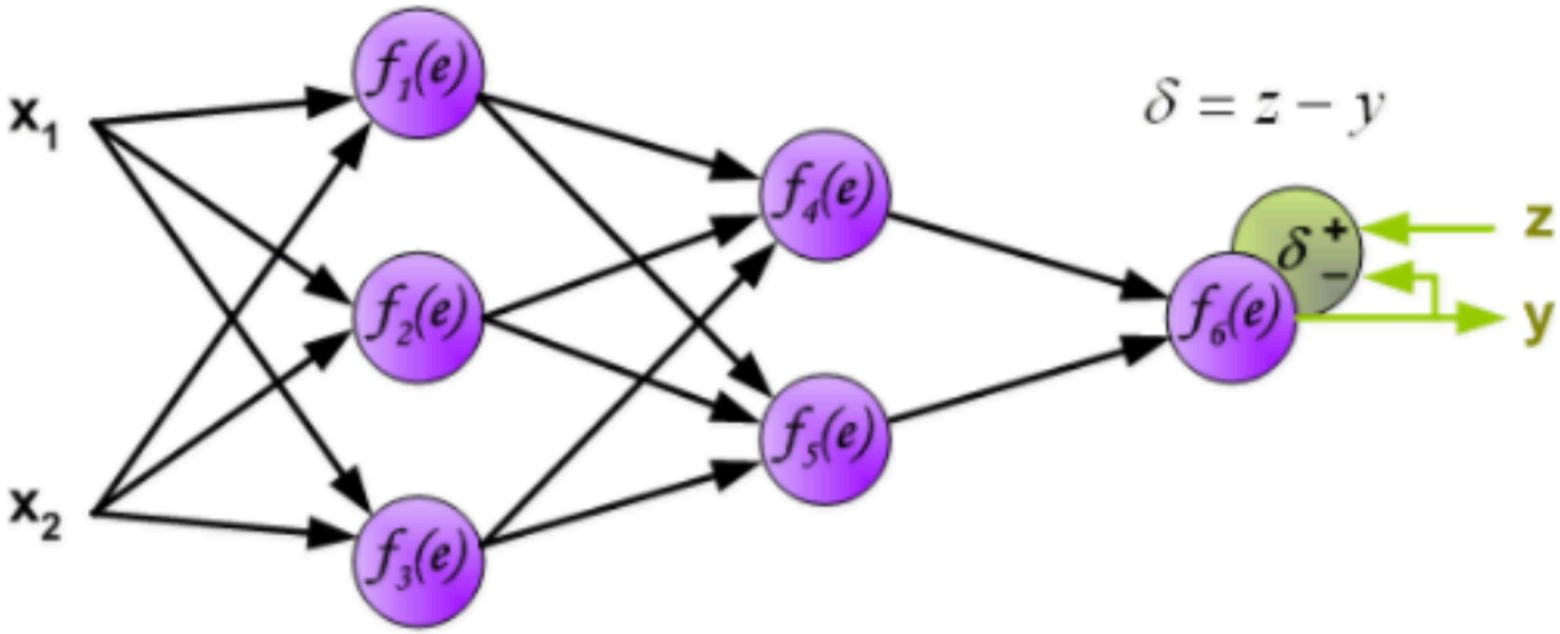


Back Propagation

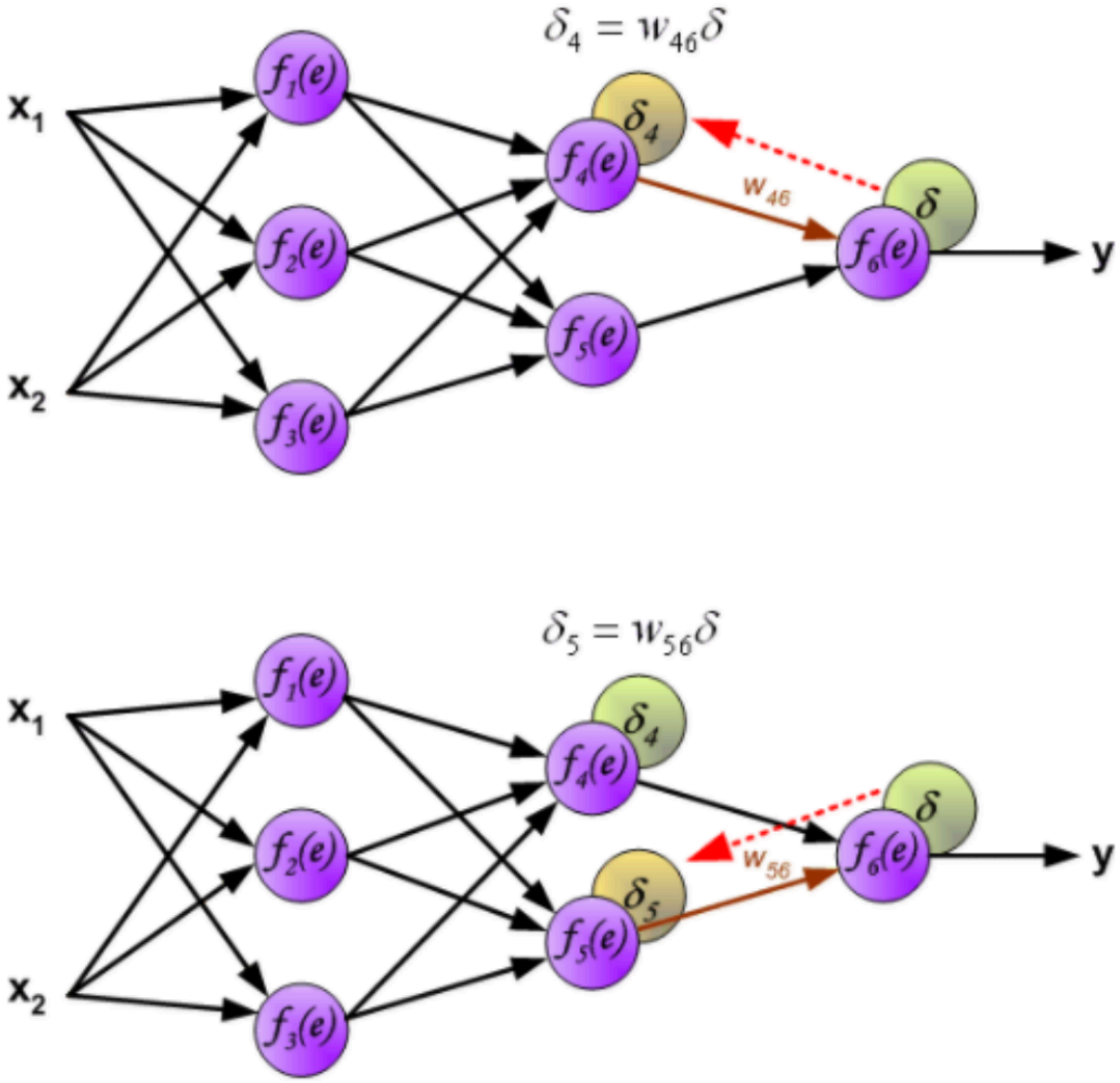


Back Propagation

There is a certain error between the calculated output y and the true result z of the training set sum

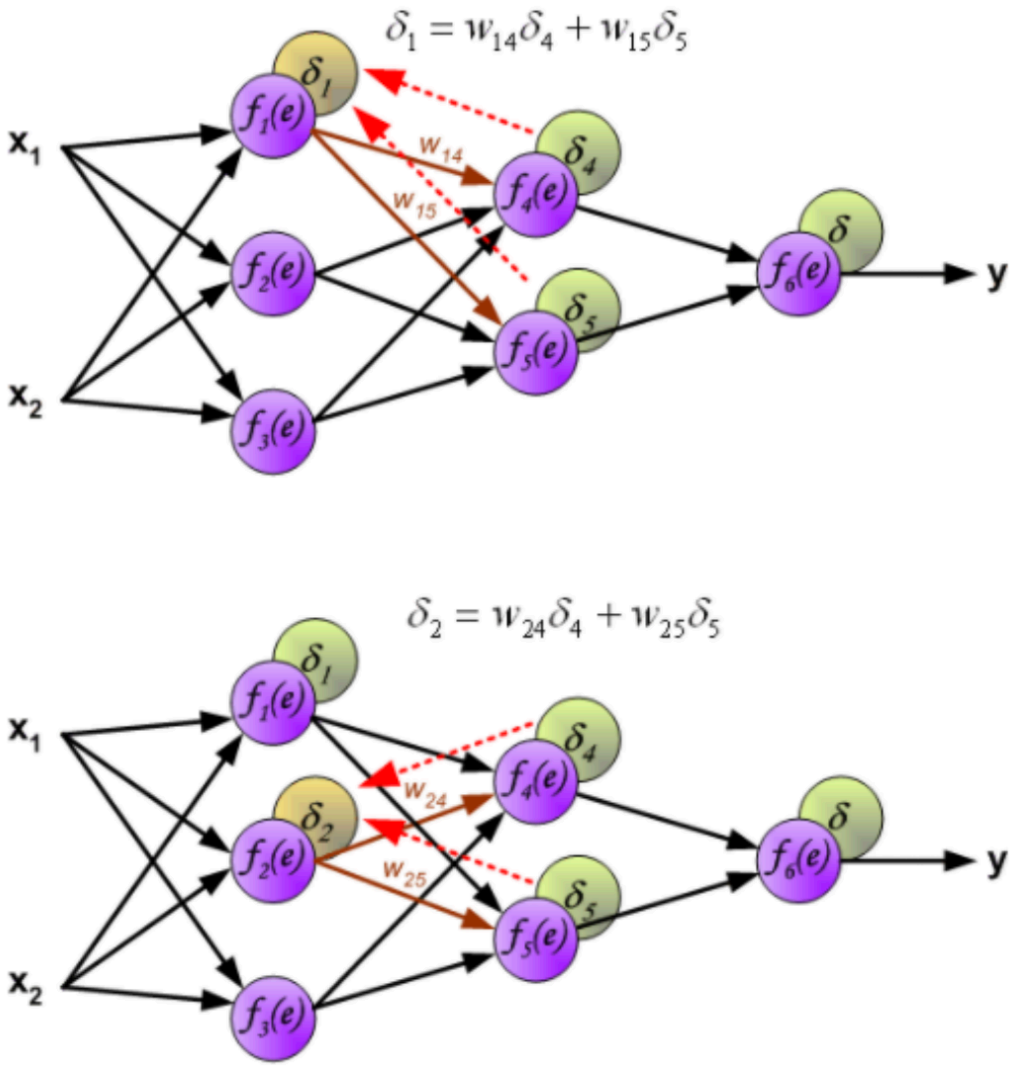


Back Propagation



The weight coefficient w_{mn} is used to propagate the error signal δ, and a new error δ_m will be obtained

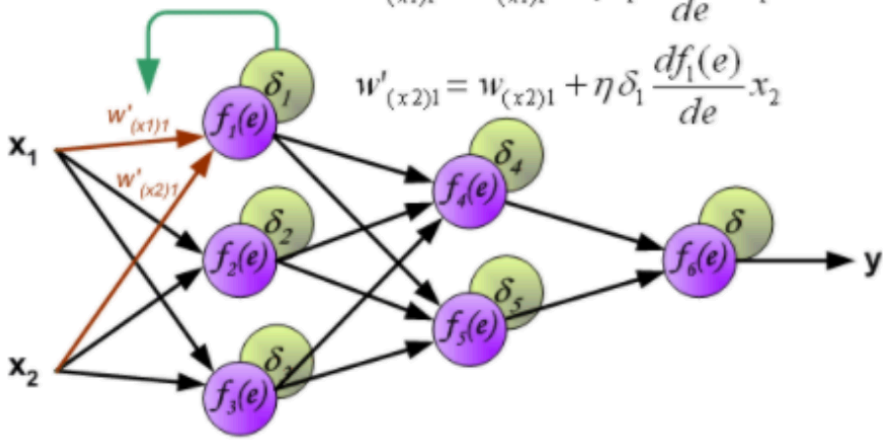
Back Propagation



Back Propagation

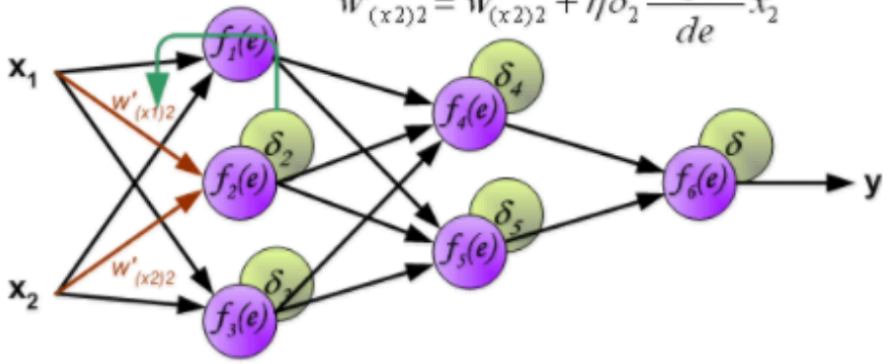
$$w'_{(x1)1} = w_{(x1)1} + \eta\delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta\delta_1 \frac{df_1(e)}{de} x_2$$



$$w'_{(x1)2} = w_{(x1)2} + \eta\delta_2 \frac{df_2(e)}{de} x_1$$

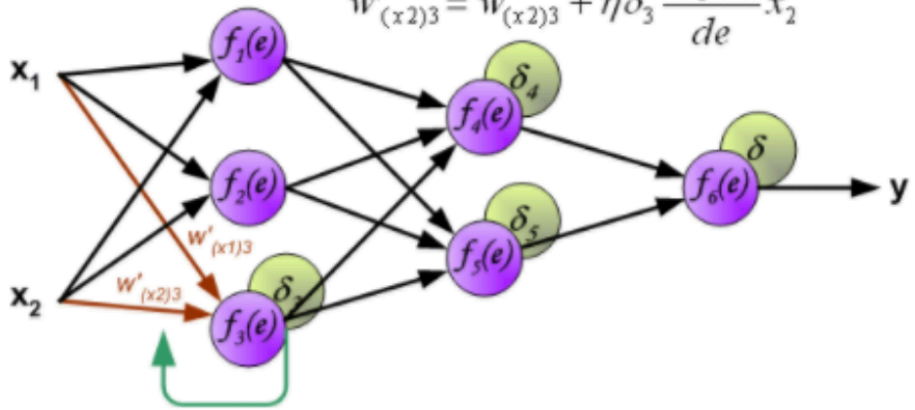
$$w'_{(x2)2} = w_{(x2)2} + \eta\delta_2 \frac{df_2(e)}{de} x_2$$



η is a coefficient, which affects the range of weight change. There are many ways to choose it

$$w'_{(x1)3} = w_{(x1)3} + \eta\delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta\delta_3 \frac{df_3(e)}{de} x_2$$



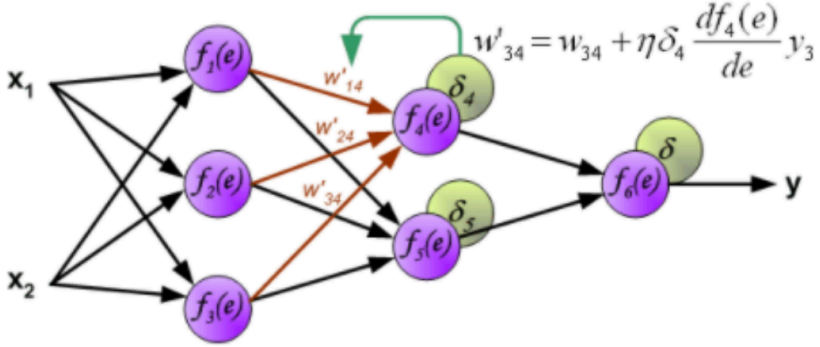
Back Propagation

Pass the input value in the forward direction to obtain the error. According to the error back propagation error, the weight is updated to achieve the correction of the entire network

$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

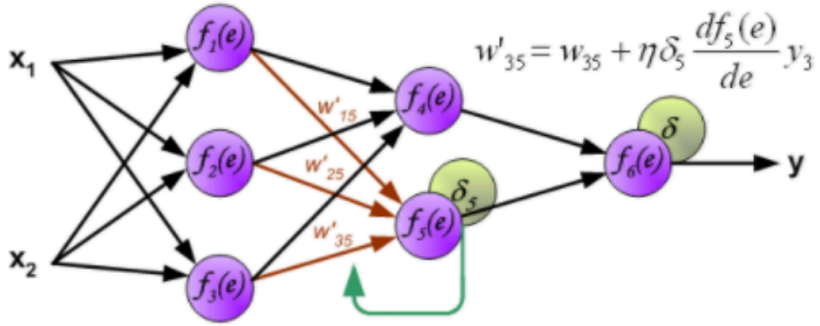
$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$



$$w'_{15} = w_{15} + \eta\delta_5 \frac{df_5(e)}{de} y_1$$

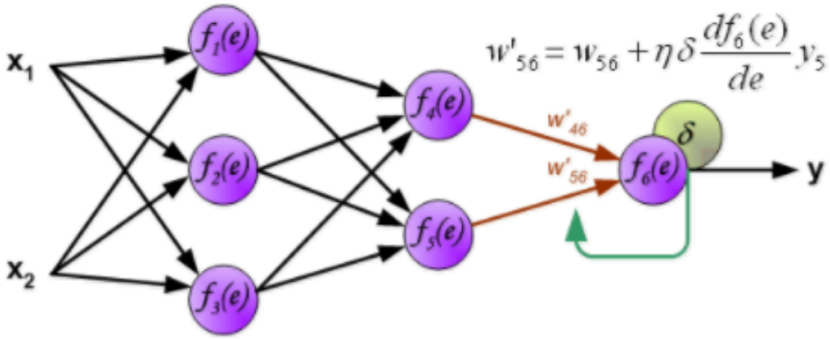
$$w'_{25} = w_{25} + \eta\delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta\delta_5 \frac{df_5(e)}{de} y_3$$

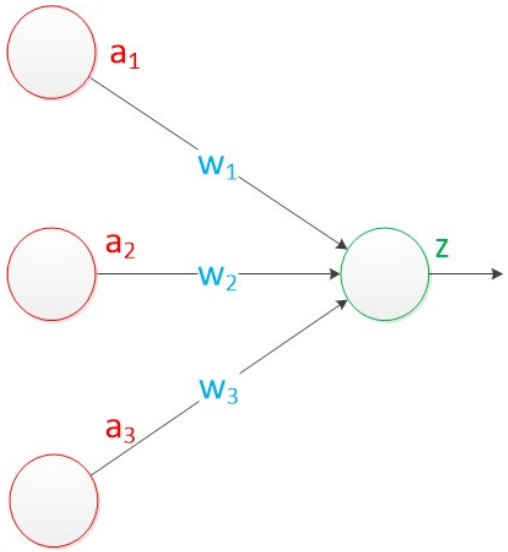


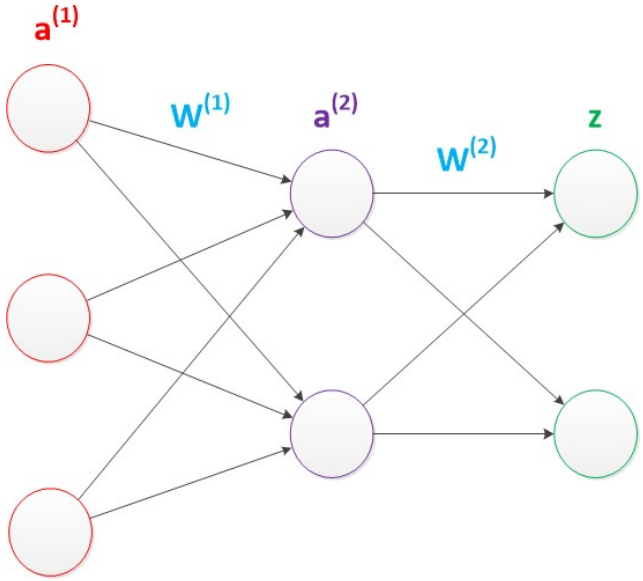
$$w'_{46} = w_{46} + \eta\delta \frac{df_6(e)}{de} y_4$$

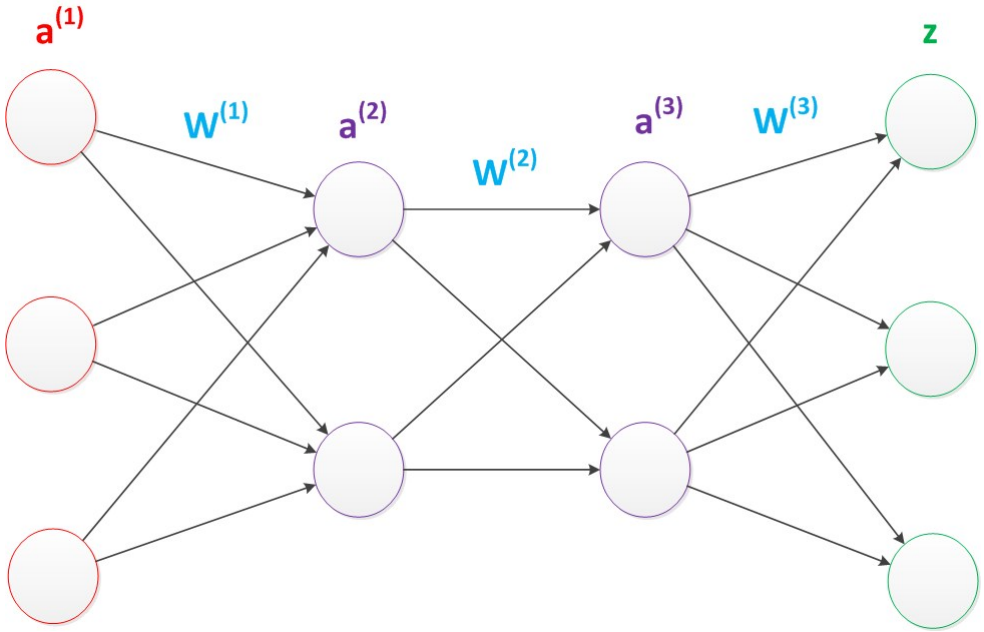
$$w'_{56} = w_{56} + \eta\delta \frac{df_6(e)}{de} y_5$$



Multi-layer neural network (deep learning)



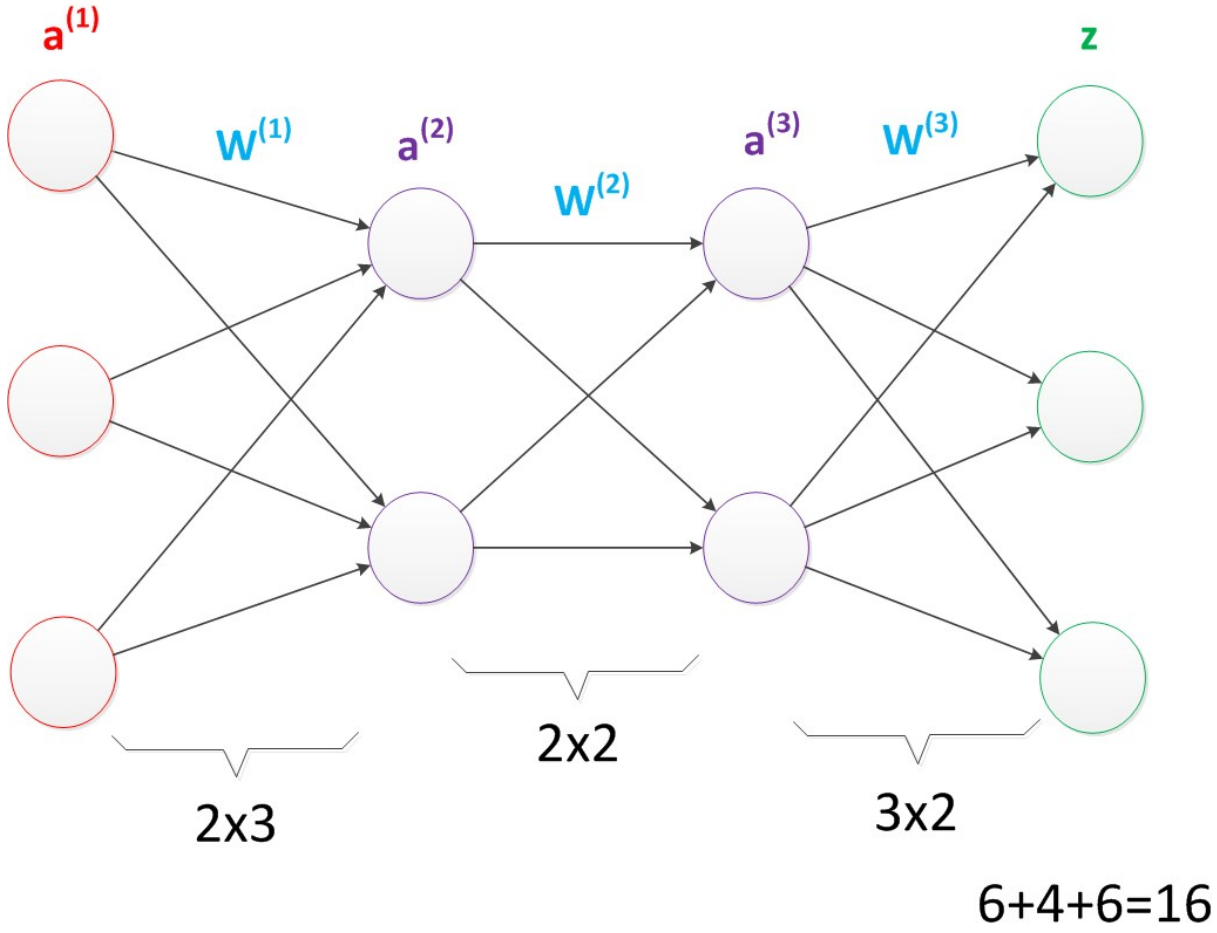




$g(\mathbf{W}^{(1)} * \mathbf{a}^{(1)}) = \mathbf{a}^{(2)};$
$g(\mathbf{W}^{(2)} * \mathbf{a}^{(2)}) = \mathbf{a}^{(3)};$
$g(\mathbf{W}^{(3)} * \mathbf{a}^{(3)}) = \mathbf{z};$

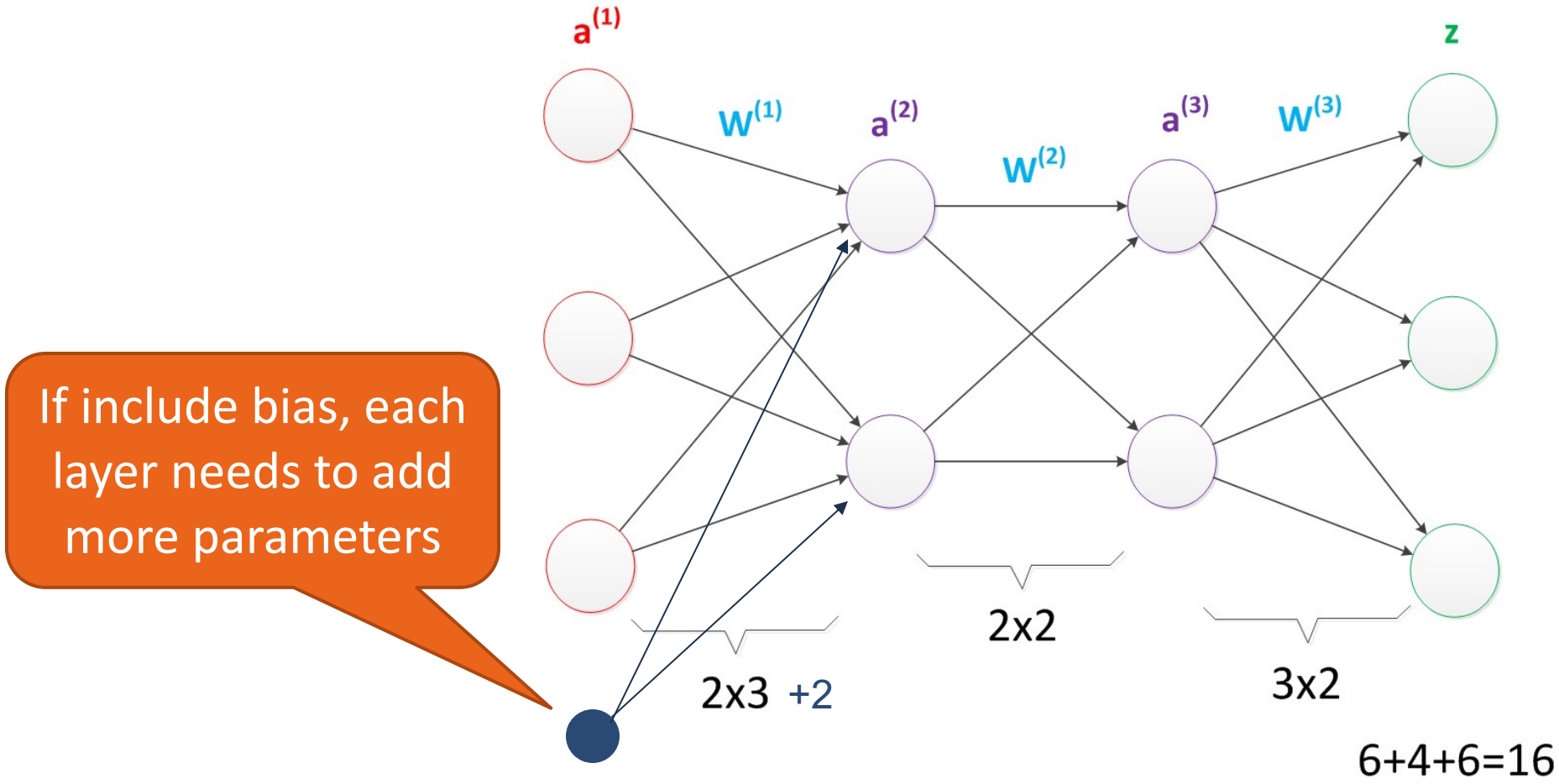
Multi-layer neural network (deep learning)

- There are 6 parameters in W(1), 4 parameters in W(2), and 6 parameters in W(3)



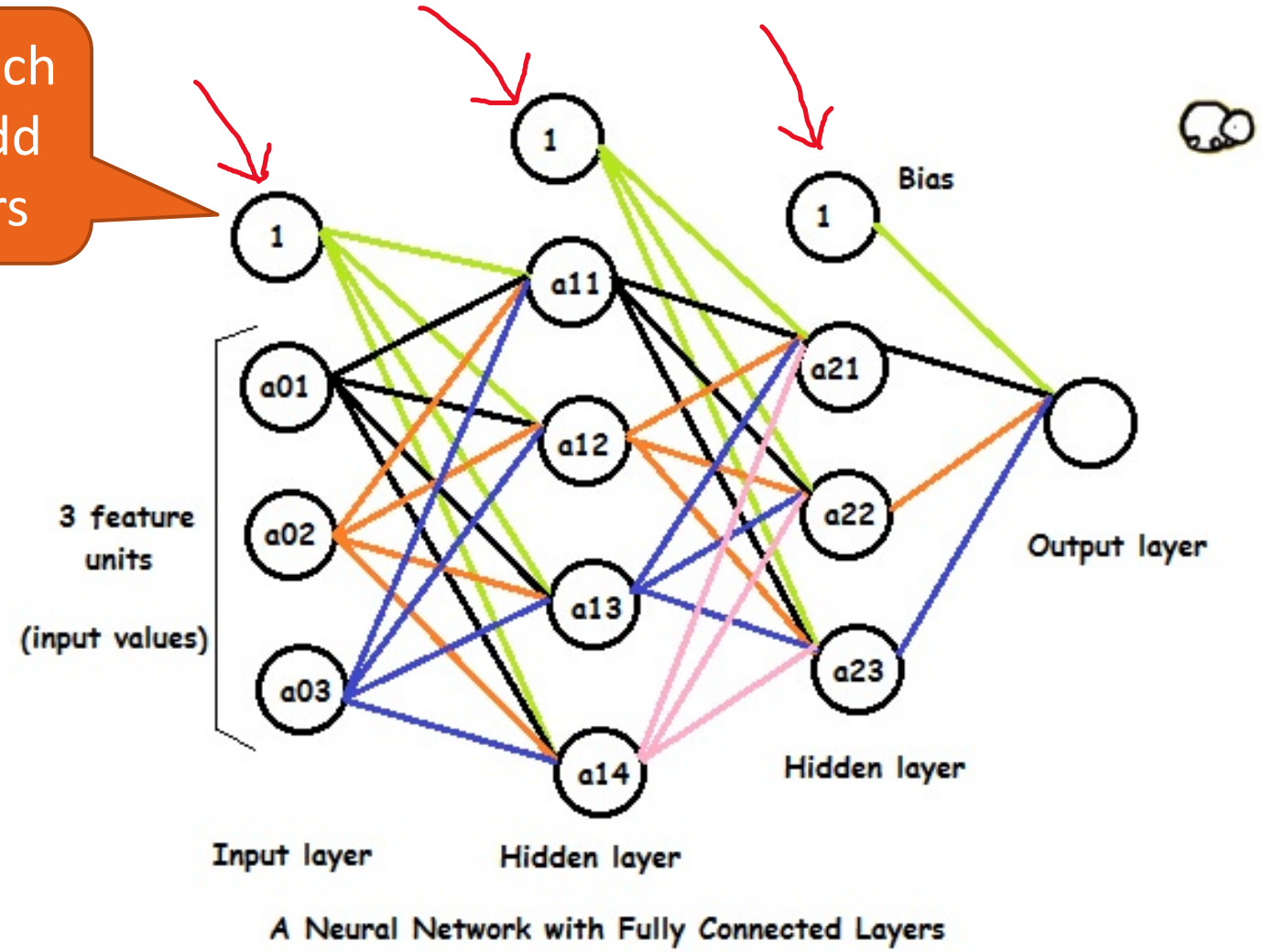
Multi-layer neural network (deep learning)

- There are 6 parameters in W(1), 4 parameters in W(2), and 6 parameters in W(3)



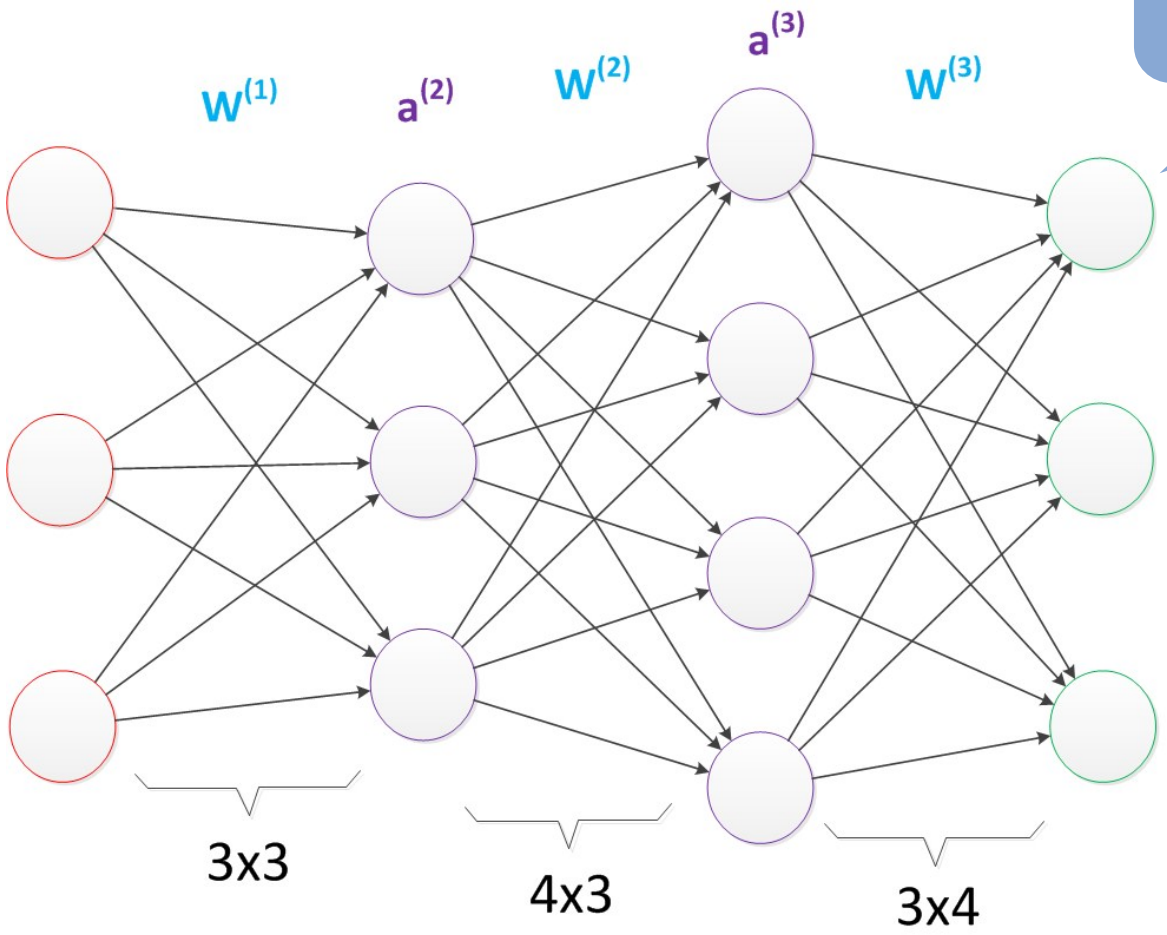
Multi-layer neural network (deep learning)

If include bias, each layer needs to add more parameters



Multi-layer neural network (deep learning)

- How many parameters in this network?

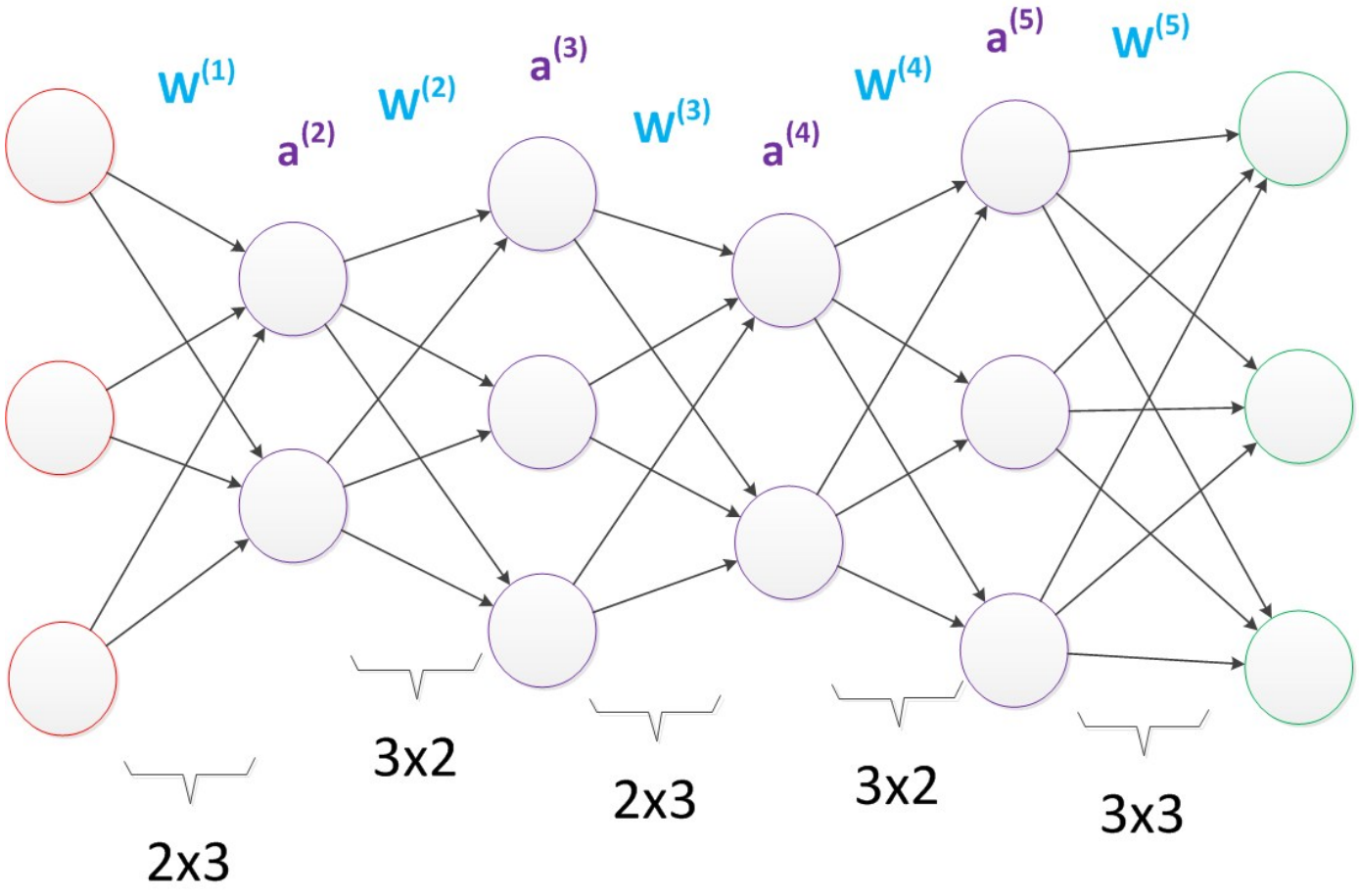


The number of parameters of the second neural network is nearly twice that of the first neural network, resulting in better representation capabilities

9+12+12=33

Multi-layer neural network (deep learning)

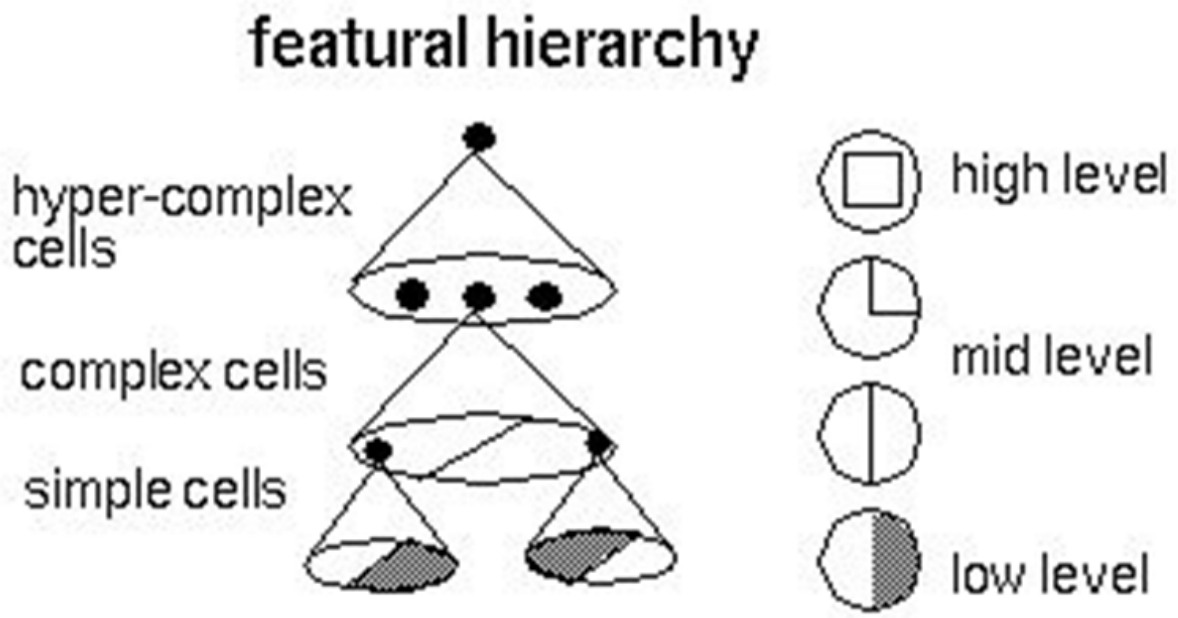
- How many parameters in this network?



The same number of parameters can be expressed at a deeper level

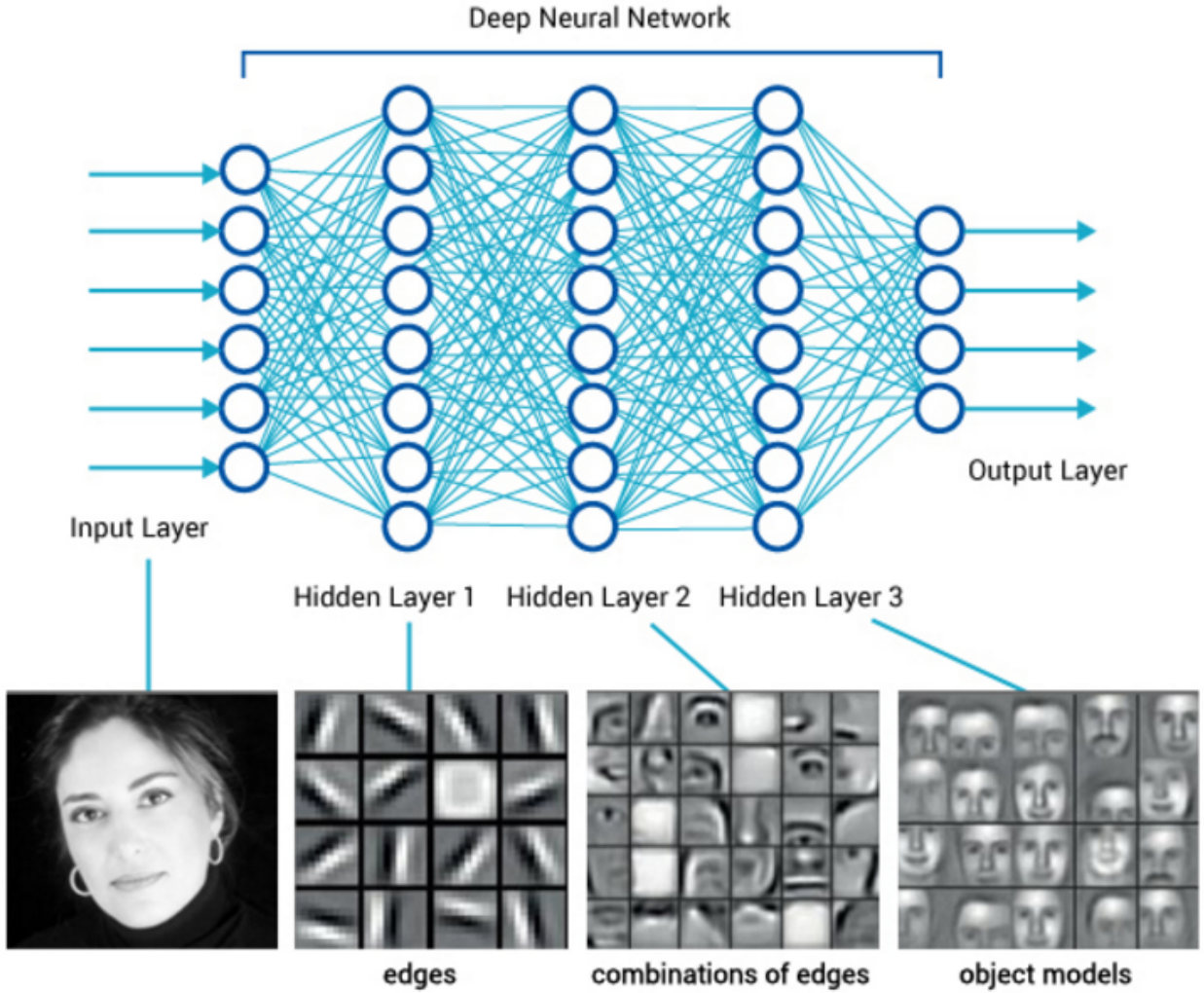
6+6+6+6+9=33

What are the benefits of adding more levels?



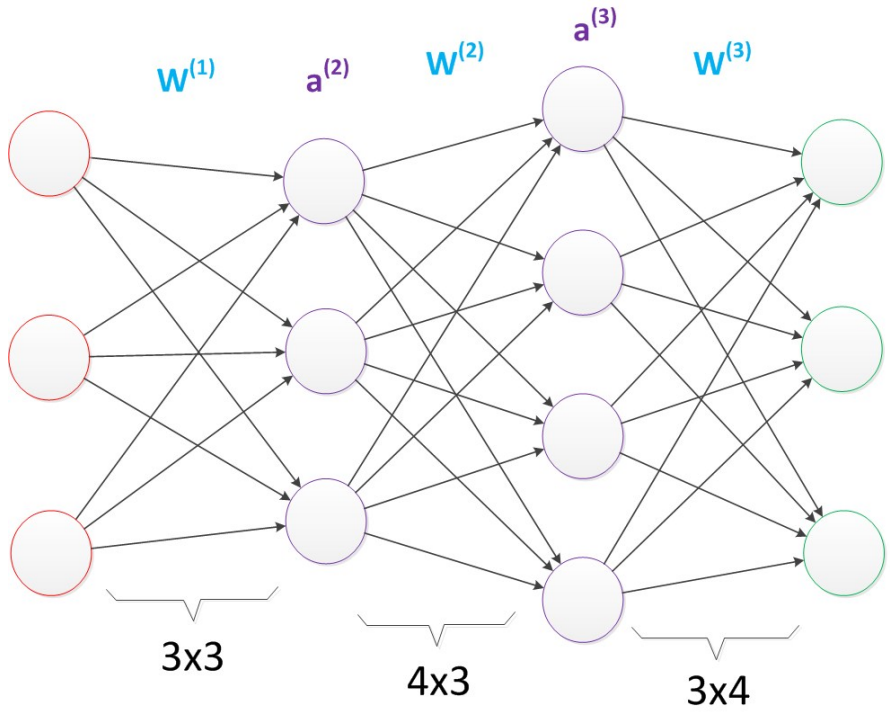
- More in-depth representation features, and stronger function simulation capabilities
- As the number of layers in the network increases, each layer has a deeper abstraction of the previous level
  - 1st layer: margin
  - 2nd layer: shape
  - 3rd layer: pattern
  - ...
  - nth layer: target

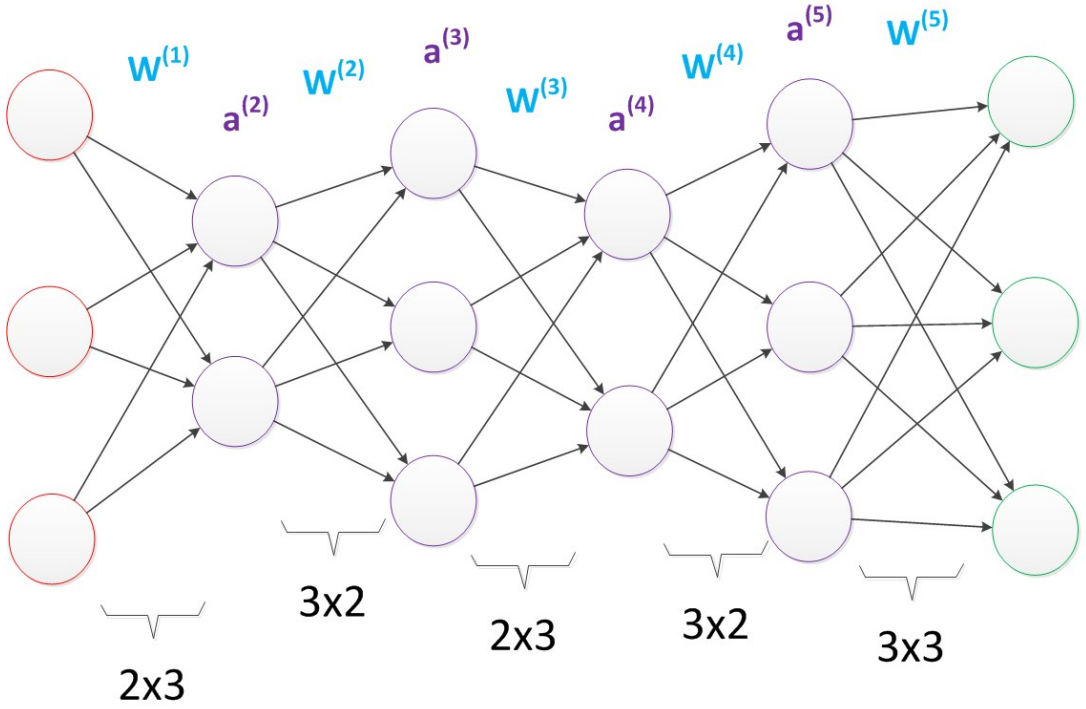
What are the benefits of adding more levels?



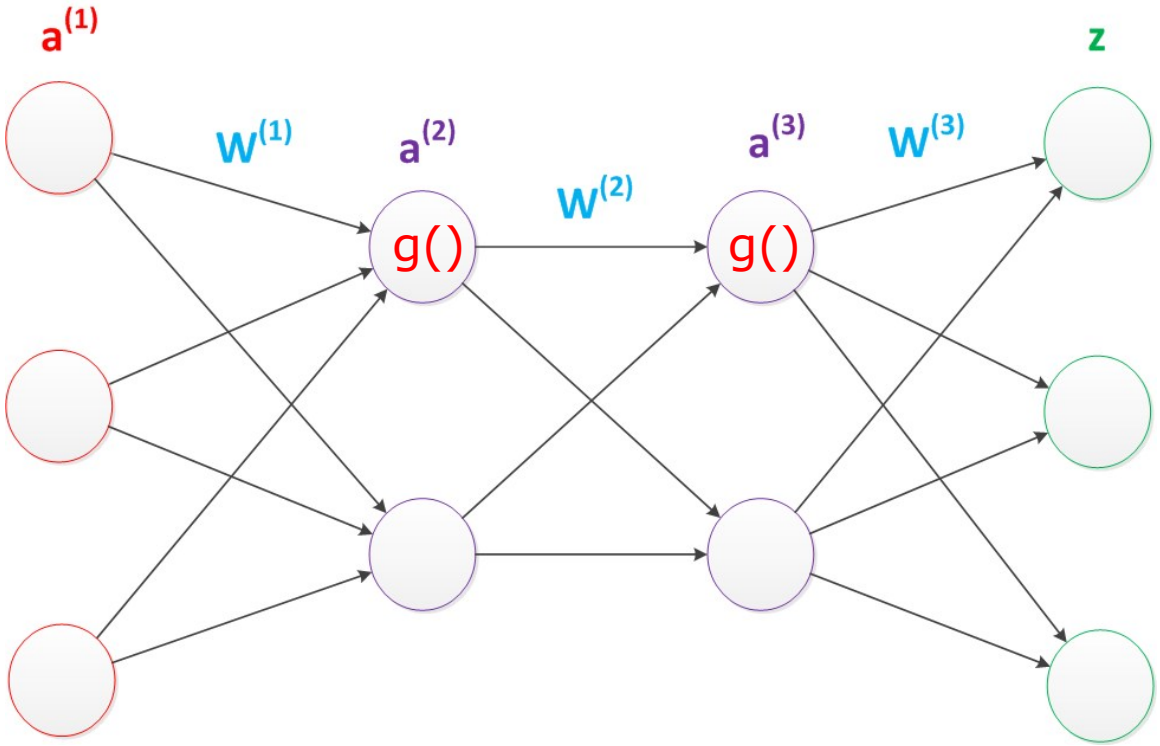
Multi-layer neural network (deep learning)

- Research found that in the case of the same number of parameters, deeper networks tend to have better recognition efficiency than shallow networks



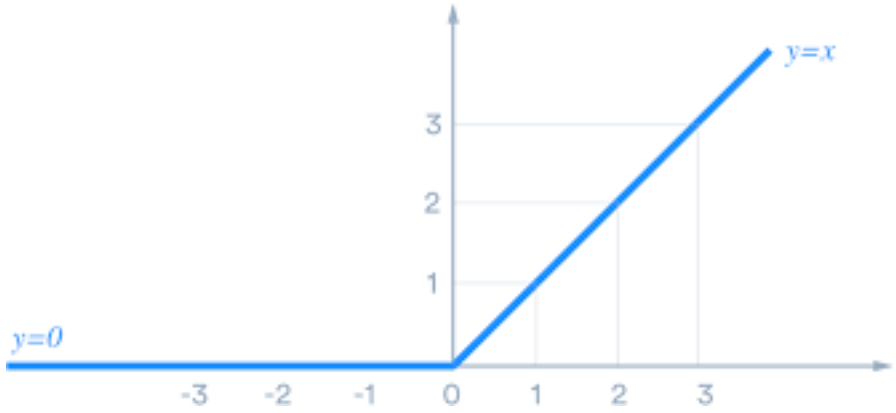


Multi-layer neural network (deep learning)

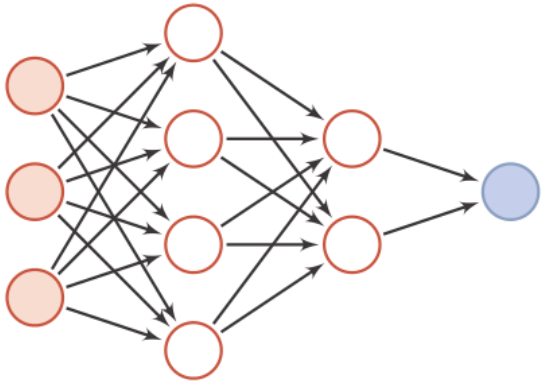


$g(\mathbf{W}^{(1)} * \mathbf{a}^{(1)}) = \mathbf{a}^{(2)};$
$g(\mathbf{W}^{(2)} * \mathbf{a}^{(2)}) = \mathbf{a}^{(3)};$
$g(\mathbf{W}^{(3)} * \mathbf{a}^{(3)}) = \mathbf{z};$

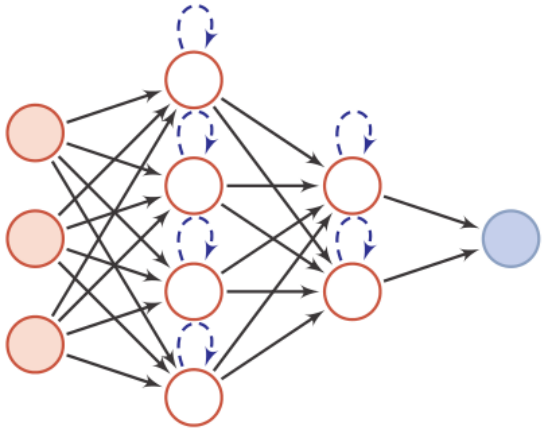
We used ReLU function for g()



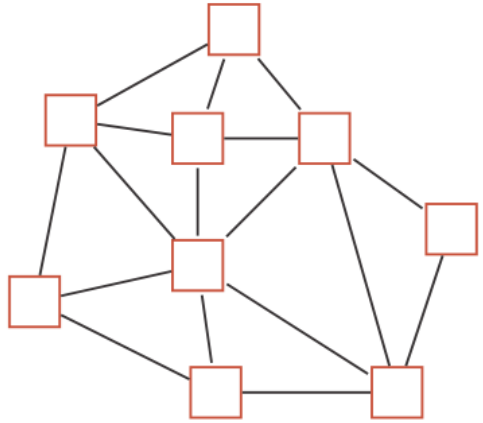
Other neural networks



Feedforward network



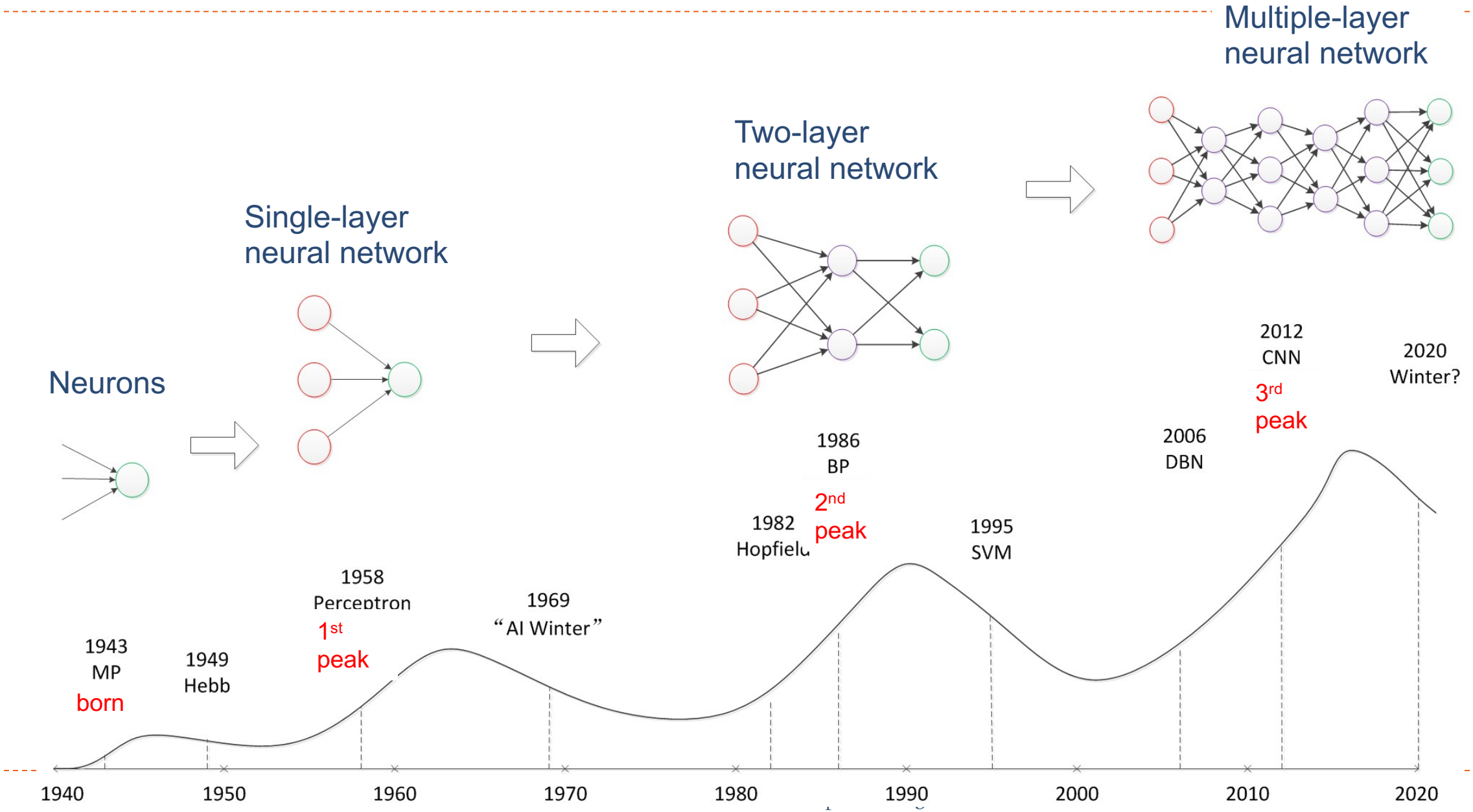
Memory Network



Graph Network

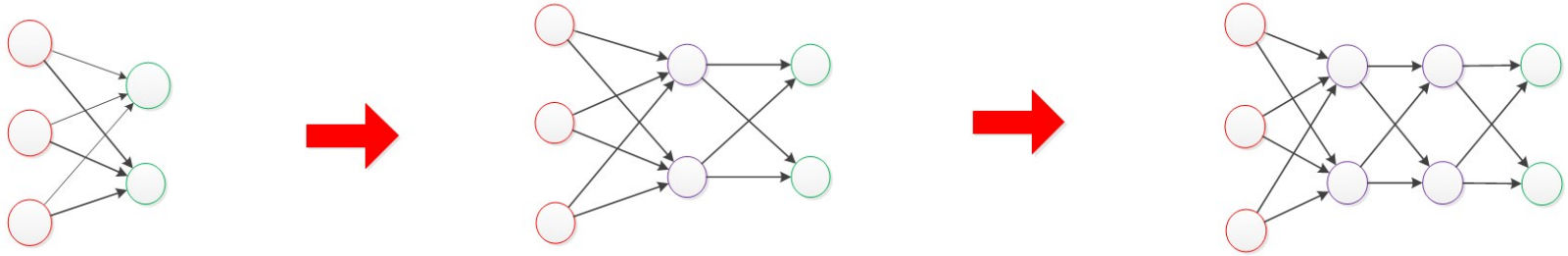
A round node represents a neuron, and a square node represents a group of neurons.

Conclusion

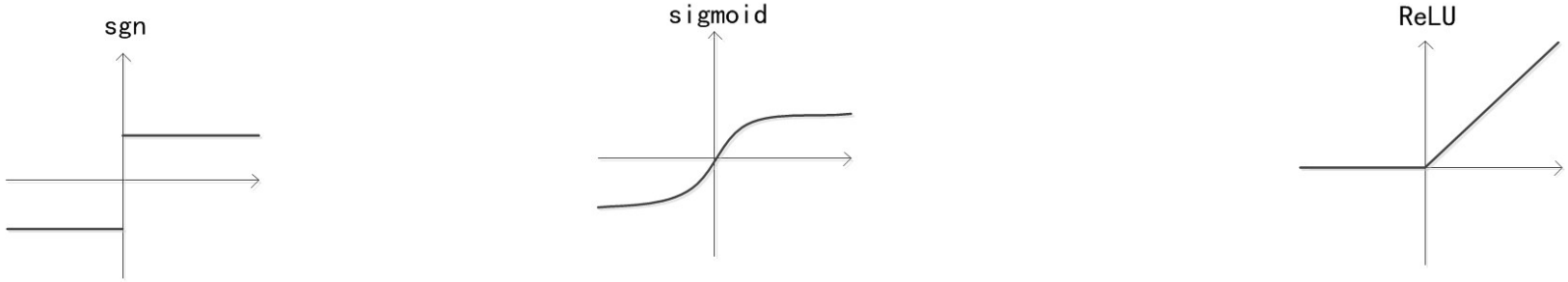

Multiple-layer neural network
Two-layer neural network
Single-layer neural network
Neurons
1943 MP born
1949 Hebb
1958 Perceptron 1st peak
1969 "AI Winter"
1982 Hopfield
1986 BP 2nd peak
1995 SVM
2006 DBN
2012 CNN 3rd peak
2020 Winter?
1940
1950
1960
1970
1980
1990
2000
2010
2020

Conclusion

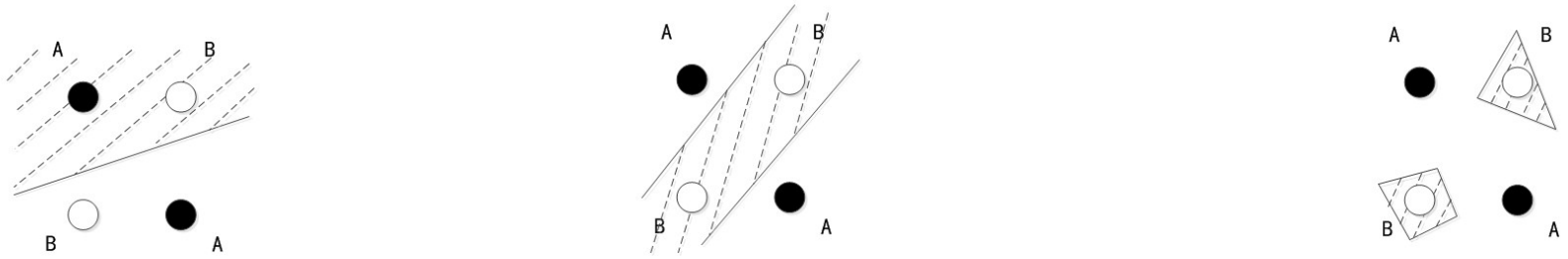
structure



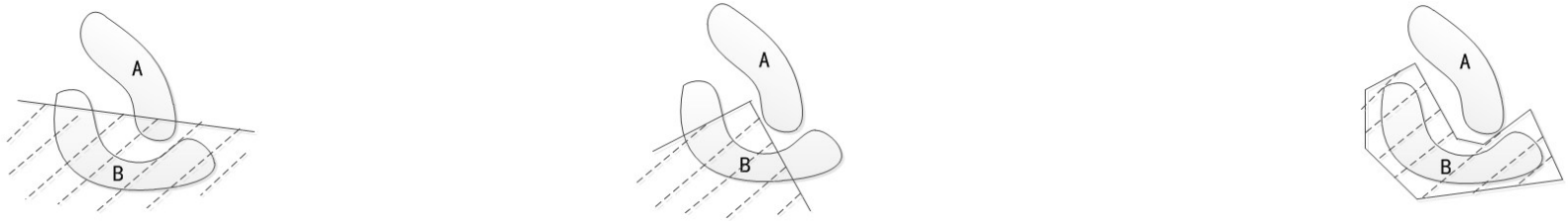
function



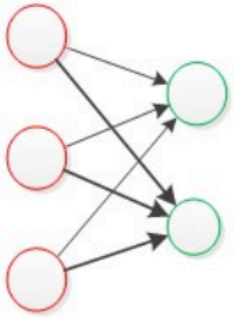
XOR problem

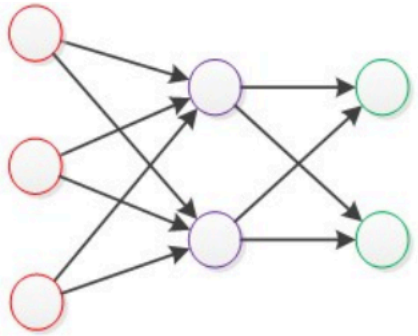


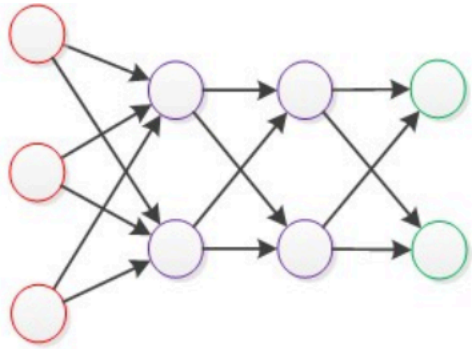
real problem



Conclusion



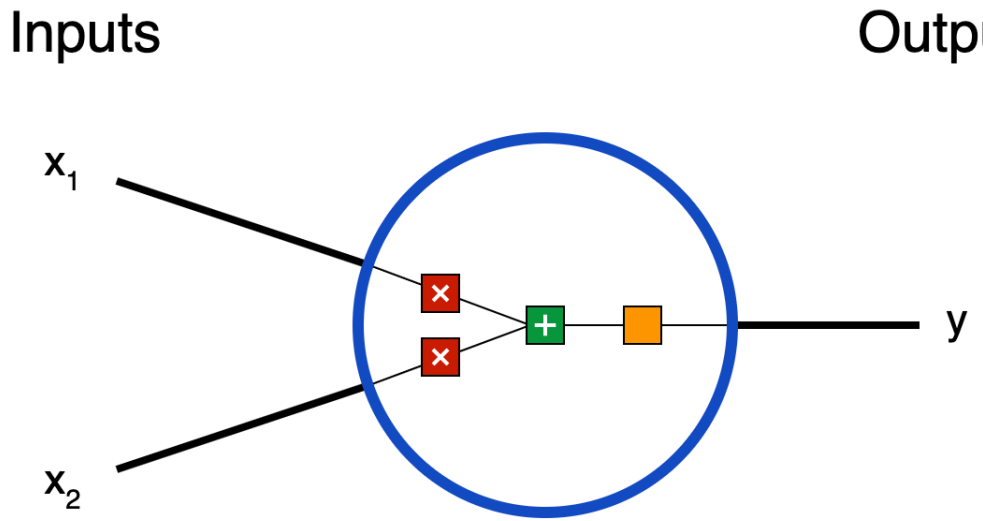




	Single-layer neural network	Two-layer neural network	Multiple-layer neural network
Year	60-70s	85-95s	10-20s
Computer	Transistor	CPU	GPU
Data	1-10	1k-10k	1M-10M
Algorithm	Basic learning	BP	Deep learning

Example: how to create a feedforward neural network

1. Building Blocks: Neurons



each input is multiplied by a weight: ■

$$x_1 \rightarrow x_1 * w_1$$

$$x_2 \rightarrow x_2 * w_2$$

all the weighted inputs are added together with a bias b: ■

$$(x_1 * w_1) + (x_2 * w_2) + b$$

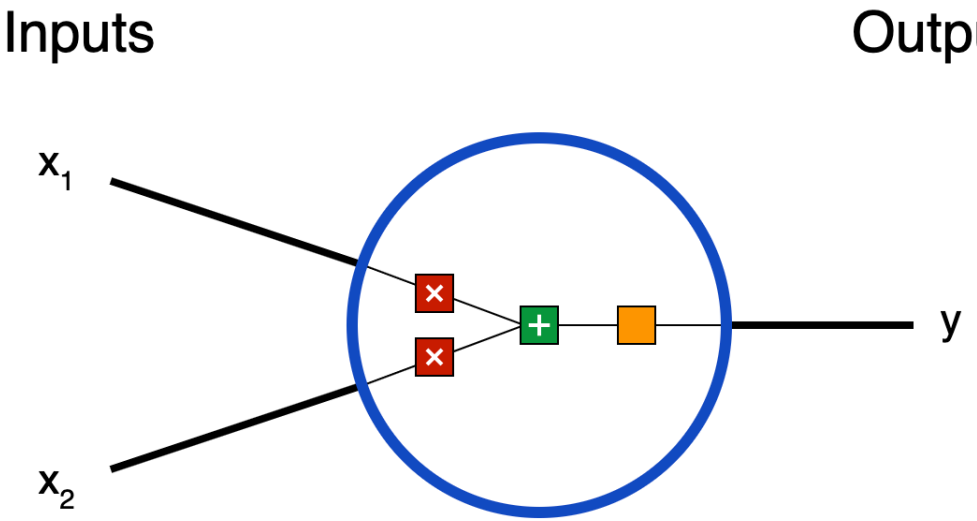
the sum is passed through an activation function: ■

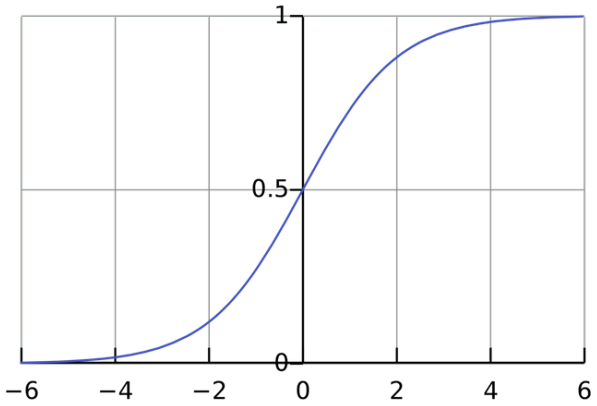
$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

Example: how to create a feedforward neural network

1. Building Blocks: Neurons

Activation function is used to turn an unbounded input into an output

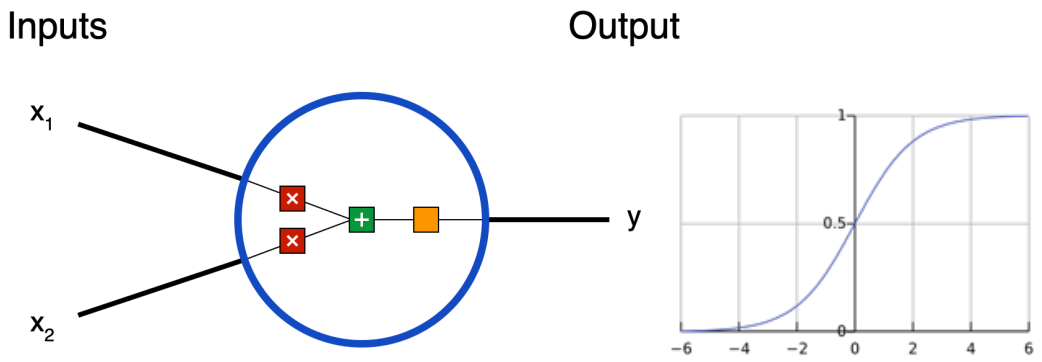




$(-\infty,+\infty)$ ⇨ ⇨ $(0,1)$

Example: how to create a feedforward neural network

1. Building Blocks: Neurons



$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

Suppose $w = [0, 1]$ and $x = [2, 3]$

$$b = 4$$

Then,

$$\begin{aligned}(w \cdot x) + b &= ((w_1 * x_1) + (w_2 * x_2)) + b \\ &= 0 * 2 + 1 * 3 + 4 \\ &= 7\end{aligned}$$

$$y = f(w \cdot x + b) = f(7) = \boxed{0.999}$$

Example: how to create a feedforward neural network

1. Building Blocks: Neurons

```
import numpy as np

def sigmoid(x):
  # Our activation function: f(x) = 1 / (1 + e^(-x))
  return 1 / (1 + np.exp(-x))

class Neuron:
  def __init__(self, weights, bias):
    self.weights = weights
    self.bias = bias

  def feedforward(self, inputs):
    # Weight inputs, add bias, then use the activation function
    total = np.dot(self.weights, inputs) + self.bias
    return sigmoid(total)

weights = np.array([0, 1]) # w1 = 0, w2 = 1
bias = 4                   # b = 4
n = Neuron(weights, bias)

x = np.array([2, 3])       # x1 = 2, x2 = 3
print(n.feedforward(x))    # 0.9990889488055994
```

Suppose $w = [0, 1]$ and $x = [2, 3]$

$$b = 4$$

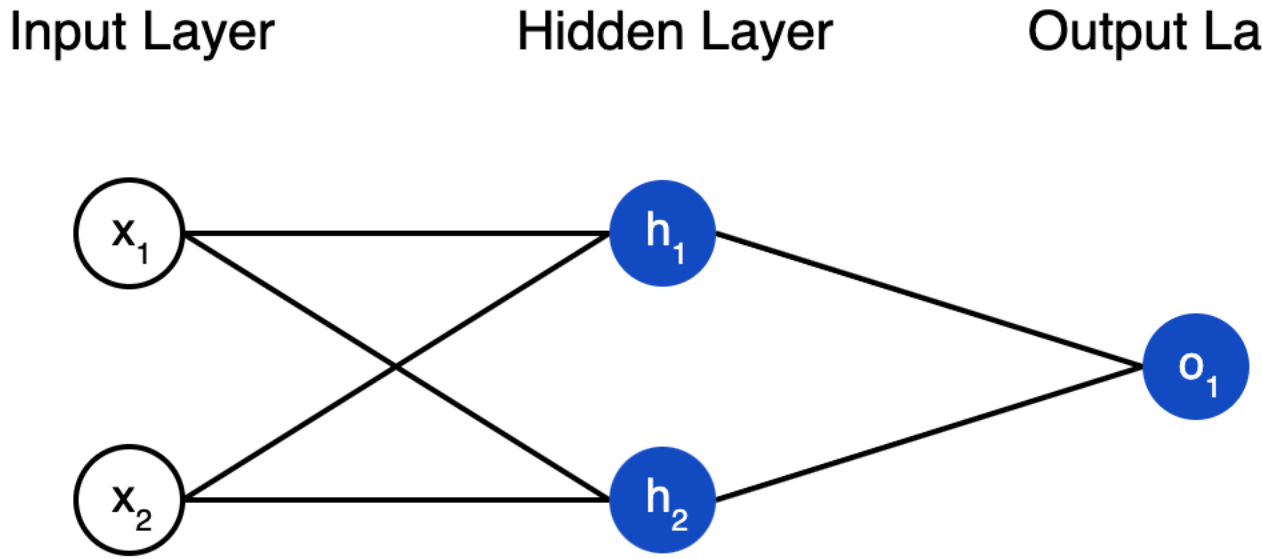
Then,

$$\begin{aligned}(w \cdot x) + b &= ((w_1 * x_1) + (w_2 * x_2)) + b \\ &= 0 * 2 + 1 * 3 + 4 \\ &= 7\end{aligned}$$

$$y = f(w \cdot x + b) = f(7) = \boxed{0.999}$$

Example: how to create a feedforward neural network

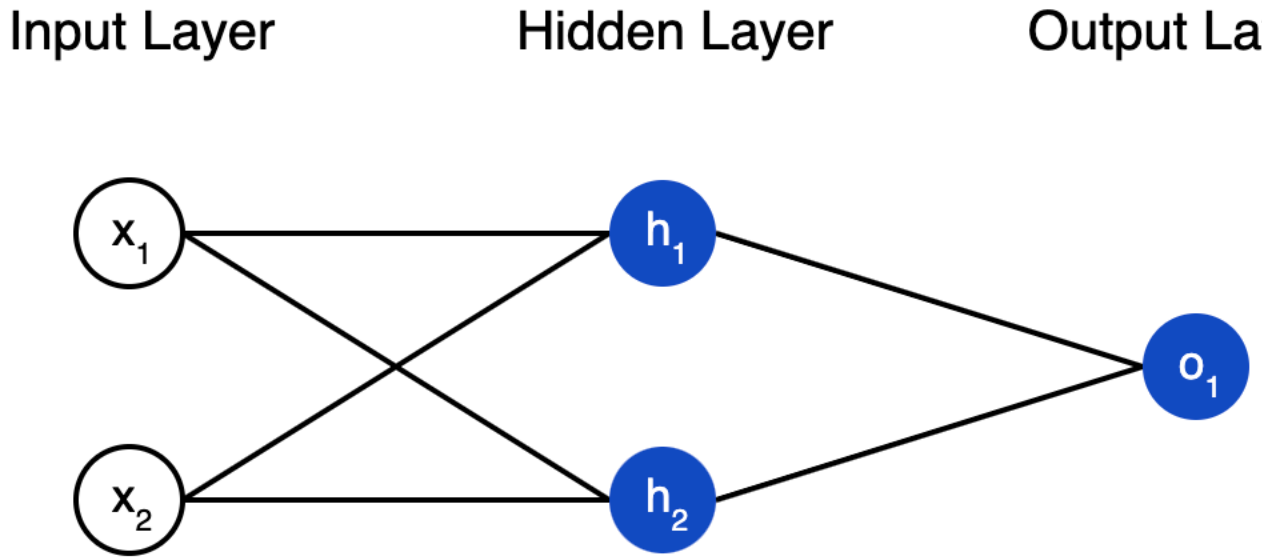
2. Combining Neurons into a Neural Network



This network has 2 inputs, a hidden layer with 2 neurons (h1, and h2), and an output layer with 1 neuron (o1)

Example: how to create a feedforward neural network

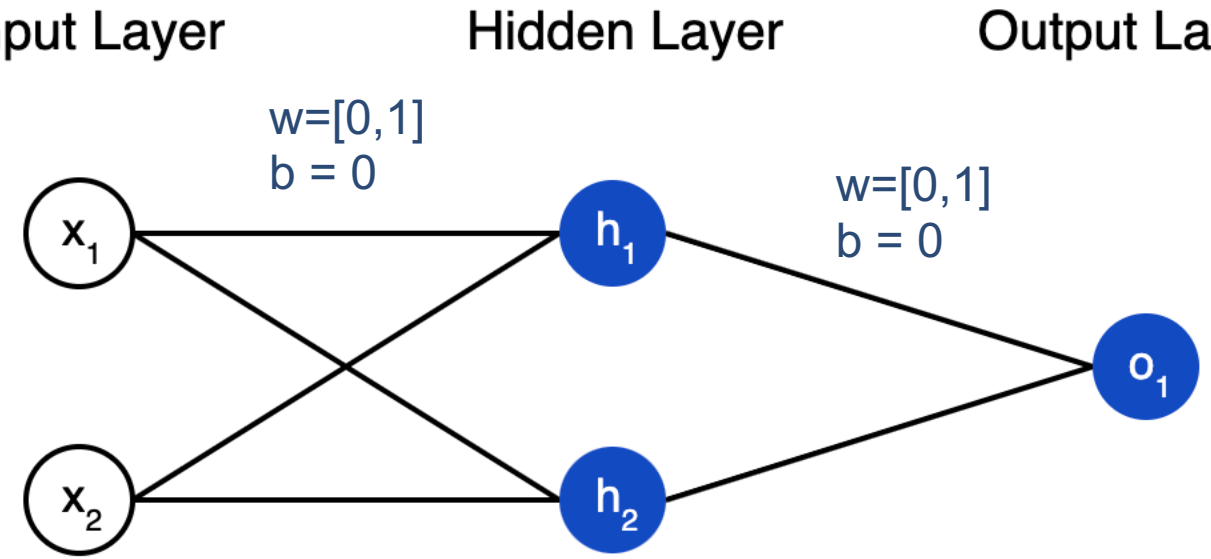
2. Combining Neurons into a Neural Network



Suppose all w=[0,1] and all b = 0, what is the output if we pass in the input x = [2, 3] ?

Example: how to create a feedforward neural network

2. Combining Neurons into a Neural Network

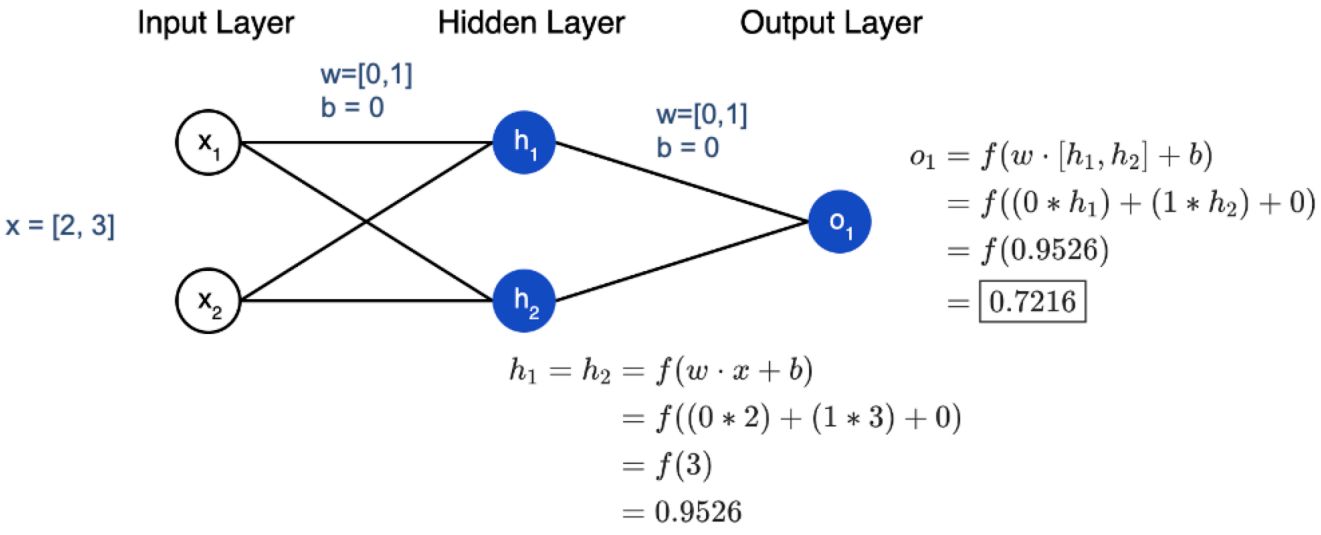


$$o_1 = f(w \cdot [h_1, h_2] + b)$$
$$= f((0 * h_1) + (1 * h_2) + 0)$$
$$= f(0.9526)$$
$$= \boxed{0.7216}$$

$$h_1 = h_2 = f(w \cdot x + b)$$
$$= f((0 * 2) + (1 * 3) + 0)$$
$$= f(3)$$
$$= 0.9526$$

Example: how to create a feedforward neural network

2. Combining Neurons into a Neural Network



$$o_1 = f(w \cdot [h_1, h_2] + b)$$
$$= f((0 * h_1) + (1 * h_2) + 0)$$
$$= f(0.9526)$$
$$= \boxed{0.7216}$$

$$h_1 = h_2 = f(w \cdot x + b)$$
$$= f((0 * 2) + (1 * 3) + 0)$$
$$= f(3)$$
$$= 0.9526$$

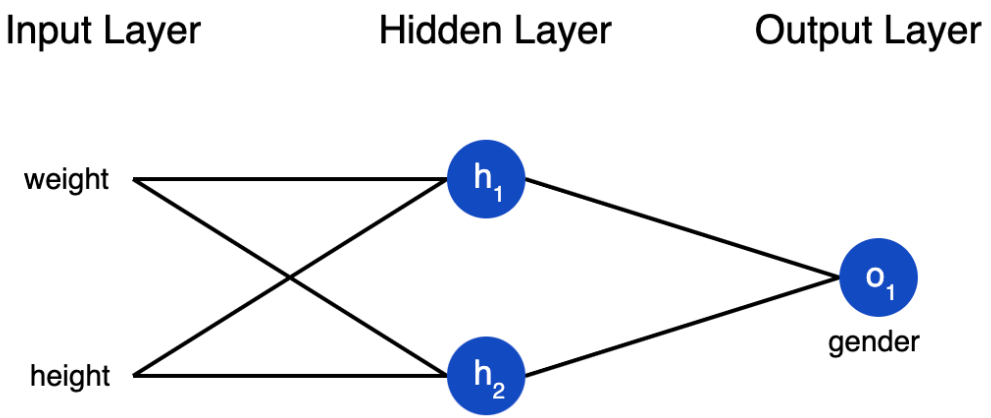
```python
class OurNeuralNetwork:
  '''
  A neural network with:
    - 2 inputs
    - a hidden layer with 2 neurons (h1, h2)
    - an output layer with 1 neuron (o1)
  Each neuron has the same weights and bias:
    - w = [0, 1]
    - b = 0
  '''
  def __init__(self):
    weights = np.array([0, 1])
    bias = 0

    # The Neuron class here is from the previous section
    self.h1 = Neuron(weights, bias)
    self.h2 = Neuron(weights, bias)
    self.o1 = Neuron(weights, bias)

  def feedforward(self, x):
    out_h1 = self.h1.feedforward(x)
    out_h2 = self.h2.feedforward(x)

    # The inputs for o1 are the outputs from h1 and h2
    out_o1 = self.o1.feedforward(np.array([out_h1, out_h2]))

    return out_o1

network = OurNeuralNetwork()
x = np.array([2, 3])
print(network.feedforward(x)) # 0.7216325609518421
```

Example: how to create a feedforward neural network

3. Training a Neural Network

Name	Weight (lb)	Height (in)	Gender
Alice	133	65	F
Bob	160	72	M
Charlie	152	70	M
Diana	120	60	F

Let's train our network to predict someone's gender given their weight and height:

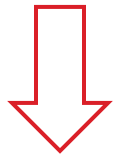


Example: how to create a feedforward neural network

3. Training a Neural Network

Name	Weight (lb)	Height (in)	Gender
Alice	133	65	F
Bob	160	72	M
Charlie	152	70	M
Diana	120	60	F

Represent Male with a 0 and Female with a 1, shift the data



Name	Weight (minus 135)	Height (minus 66)	Gender
Alice	-2	-1	1
Bob	25	6	0
Charlie	17	4	0
Diana	-15	-6	1

Example: how to create a feedforward neural network

3. Training a Neural Network: define loss function

We'll use the mean squared error (MSE) loss:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2$$

Let's break this down:

- n is the number of samples, which is 4 (Alice, Bob, Charlie, Diana).
- y represents the variable being predicted, which is Gender.
- y_{true} is the *true* value of the variable (the "correct answer"). For example, y_{true} for Alice would be 1 (Female).
- y_{pred} is the *predicted* value of the variable. It's whatever our network outputs.

Example: how to create a feedforward neural network

3. Training a Neural Network: define loss function

An Example Loss Calculation:

Name	y_{true}	y_{pred}	$(y_{true} - y_{pred})^2$
Alice	1	0	1
Bob	0	0	0
Charlie	0	0	0
Diana	1	0	1

$$\text{MSE} = \frac{1}{4}(1 + 0 + 0 + 1) = \boxed{0.5}$$

Example: how to create a feedforward neural network

3. Training a Neural Network: define loss function

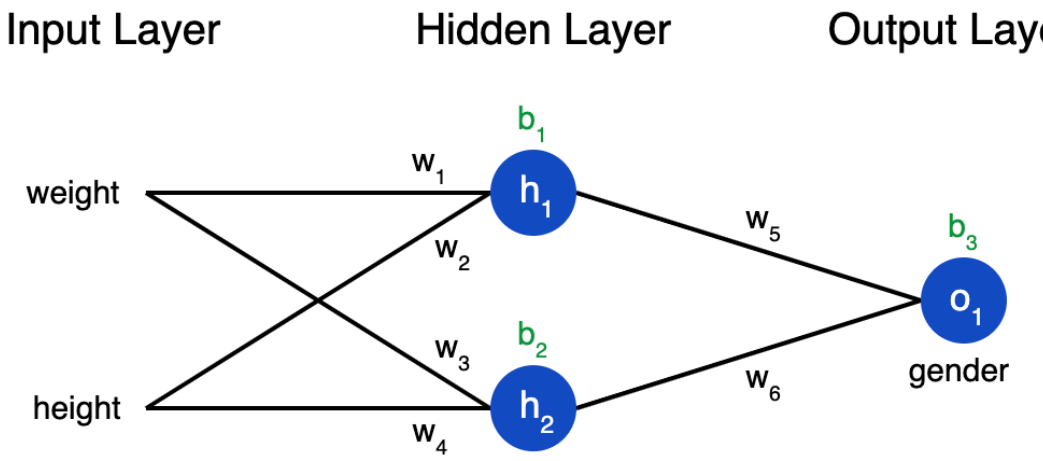
Code for Loss Calculation:

```
import numpy as np

def mse_loss(y_true, y_pred):
  # y_true and y_pred are numpy arrays of the same length.
  return ((y_true - y_pred) ** 2).mean()

y_true = np.array([1, 0, 0, 1])
y_pred = np.array([0, 0, 0, 0])

print(mse_loss(y_true, y_pred)) # 0.5
```

Example: how to create a feedforward neural network

3. Training a Neural Network: minimize loss of the neural network

Change the network's weights and biases to influence its predictions, we can write loss as a multivariable function: $L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$



How would loss L change if we changed w_1?

Partial derivative $\frac{\partial L}{\partial w_1}$

Example: how to create a feedforward neural network

3. Training a Neural Network: minimize loss of the neural network

How would loss L change if we changed w_1? Partial derivative $\frac{\partial L}{\partial w_1}$

update equation:

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

η is a constant called the **learning rate** that controls how fast we train. All we're doing is subtracting $\eta \frac{\partial L}{\partial w_1}$ from w_1:

- If $\frac{\partial L}{\partial w_1}$ is positive, w_1 will decrease, which makes L decrease.
- If $\frac{\partial L}{\partial w_1}$ is negative, w_1 will increase, which makes L increase

Example: how to create a feedforward neural network

3. Training a Neural Network:

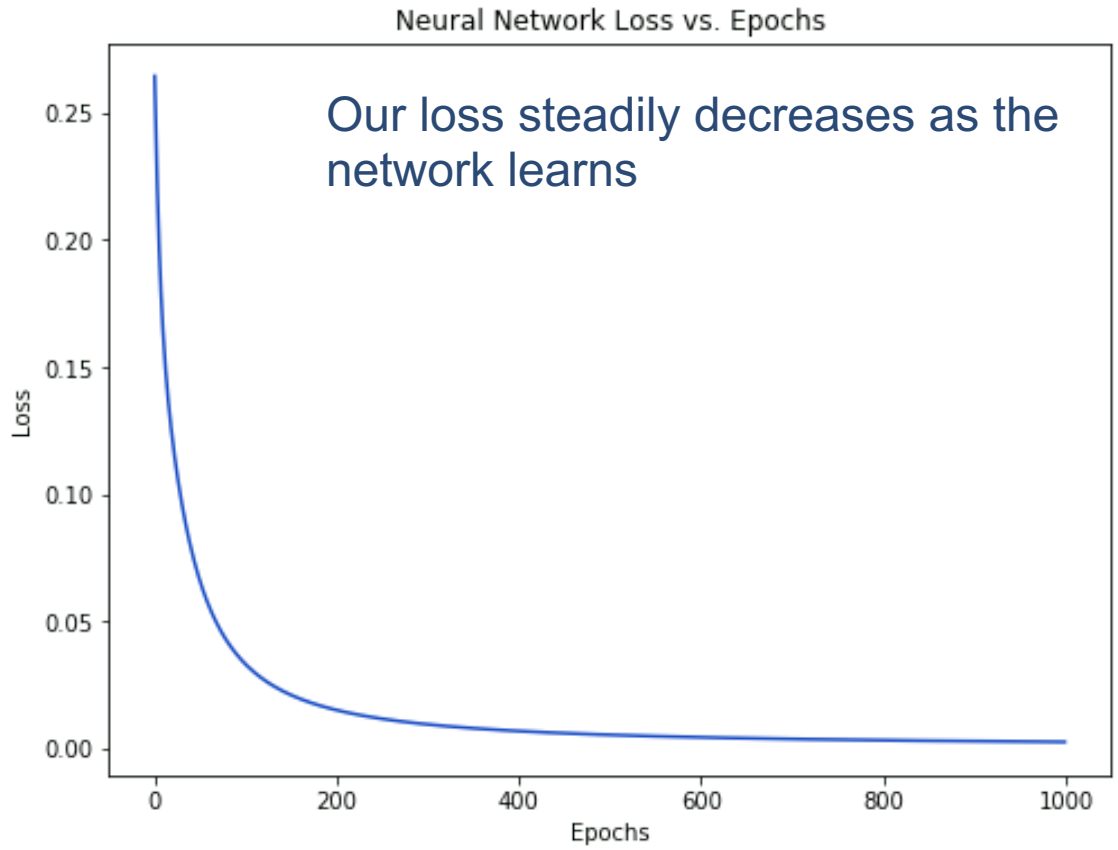
Our training process will look like this:

1. Choose **one** sample from our dataset. This is what makes it *stochastic* gradient descent - we only operate on one sample at a time.
2. Calculate all the partial derivatives of loss with respect to weights or biases (e.g. $\frac{\partial L}{\partial w_1}$, $\frac{\partial L}{\partial w_2}$, etc).
3. Use the update equation to update each weight and bias.
4. Go back to step 1.

Example: how to create a feedforward neural network

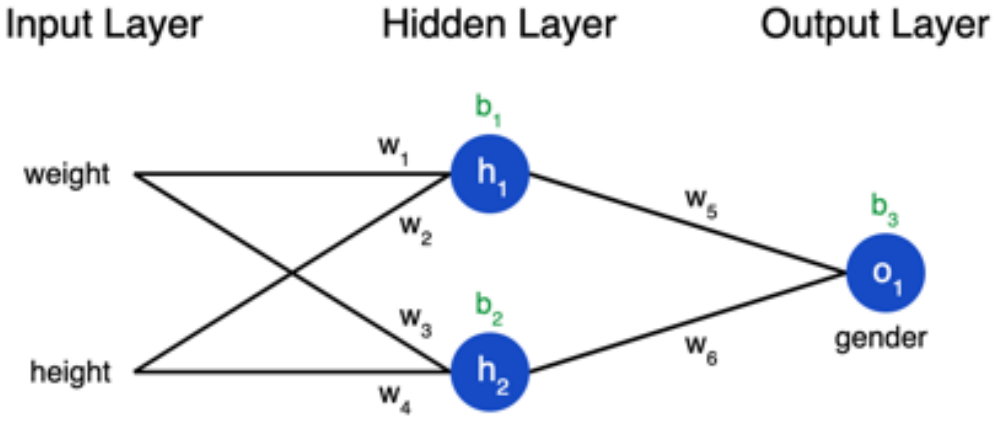
https://github.com/kevinsuo/CS7357/blob/master/fnn/fnn1.py

```
fish /Users/ksuo/Google Drive/github/CS7357/fnn                    ⌥⌘1
Epoch 730 loss: 0.003
Epoch 740 loss: 0.003
Epoch 750 loss: 0.003
Epoch 760 loss: 0.003
Epoch 770 loss: 0.003
Epoch 780 loss: 0.003
Epoch 790 loss: 0.003
Epoch 800 loss: 0.003
Epoch 810 loss: 0.003
Epoch 820 loss: 0.003
Epoch 830 loss: 0.003
Epoch 840 loss: 0.003
Epoch 850 loss: 0.003
Epoch 860 loss: 0.003
Epoch 870 loss: 0.003
Epoch 880 loss: 0.003
Epoch 890 loss: 0.003
Epoch 900 loss: 0.003
Epoch 910 loss: 0.003
Epoch 920 loss: 0.003
Epoch 930 loss: 0.003
Epoch 940 loss: 0.003
Epoch 950 loss: 0.002
Epoch 960 loss: 0.002
Epoch 970 loss: 0.002
Epoch 980 loss: 0.002
Epoch 990 loss: 0.002
ksuo@ltksup50143mac ~/G/g/C/fnn>
```



Example: how to create a feedforward neural network

Use the network to predict genders




```
# Make some predictions
emily = np.array([-7, -3]) # 128 pounds, 63 inches
frank = np.array([20, 2])  # 155 pounds, 68 inches
print("Emily: %.3f" % network.feedforward(emily)) # 0.951 - F
print("Frank: %.3f" % network.feedforward(frank)) # 0.039 - M
```