Artificial Neural Networks

Neural Network

- **a computing system based on the biological nervous network that creates the human brain.**
- **are not based on a particular computer program, but it can improve and improve its performance over time.**

- Artificial Neural Networks (ANN)
  - Information processing paradigm inspired by biological nervous systems
  - ANN is composed of a system of **neurons**神經元 connected by **synapses**突觸
  - ANN learn by example
    - Adjust synaptic connections between neurons

00.mp4

History

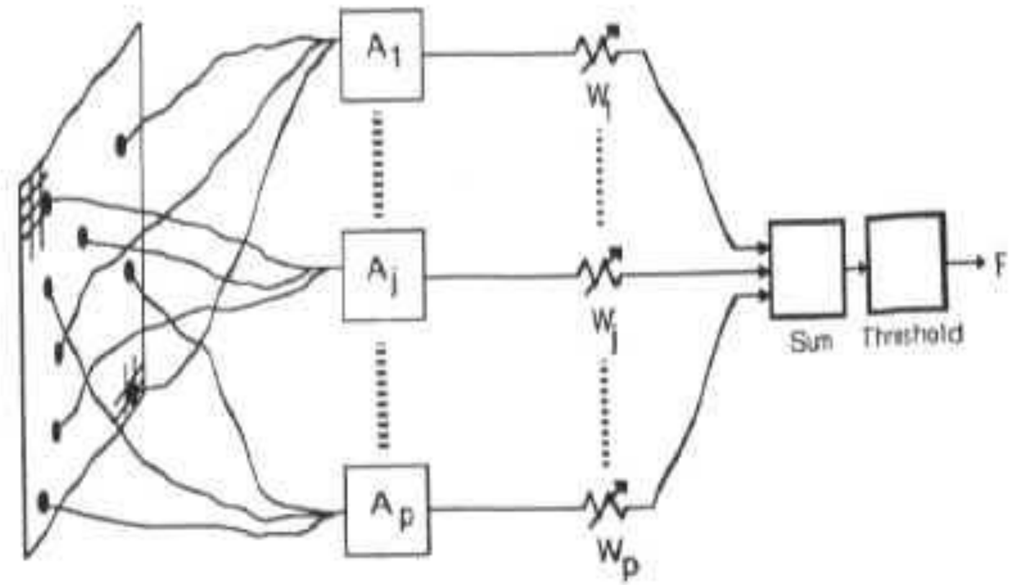
- 1943: McCulloch and Pitts model neural networks based on their understanding of neurology.
  - Neurons embed simple logic functions:
    - a or b
    - a and b
- 1950s:
  - Farley and Clark
    - IBM group that tries to model biological behavior

History

- Perceptron (Rosenblatt 1958)
  - Three layer system:
    - Input nodes
    - Output node
    - Association layer
  - Can learn to connect or associate a given input to a random output unit
- Minsky and Papert
  - Showed that a single layer perceptron cannot learn the XOR of two binary inputs

History

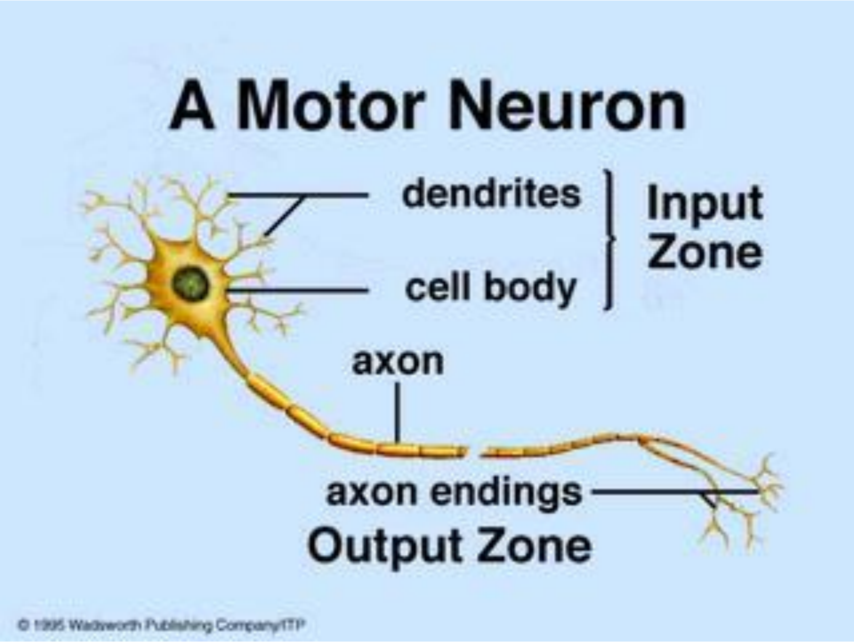
- Perceptron (Rosenblatt 1958)
  - Association units A_1, A_2, … extract features from user input
  - Output is weighted and associated
  - Function fires if weighted sum of input exceeds a threshold.

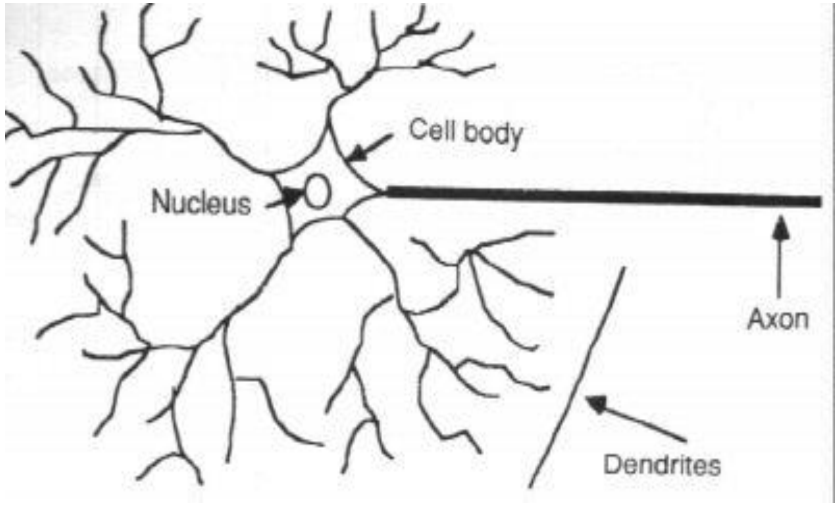


History

- Back-propagation learning method (Werbos 1974)
  - Three layers of neurons
    - Input, Output, Hidden
  - Better learning rule for generic three layer networks
  - Regenerates interest in the 1980s
- Successful applications in medicine, marketing, risk management, … (1990)

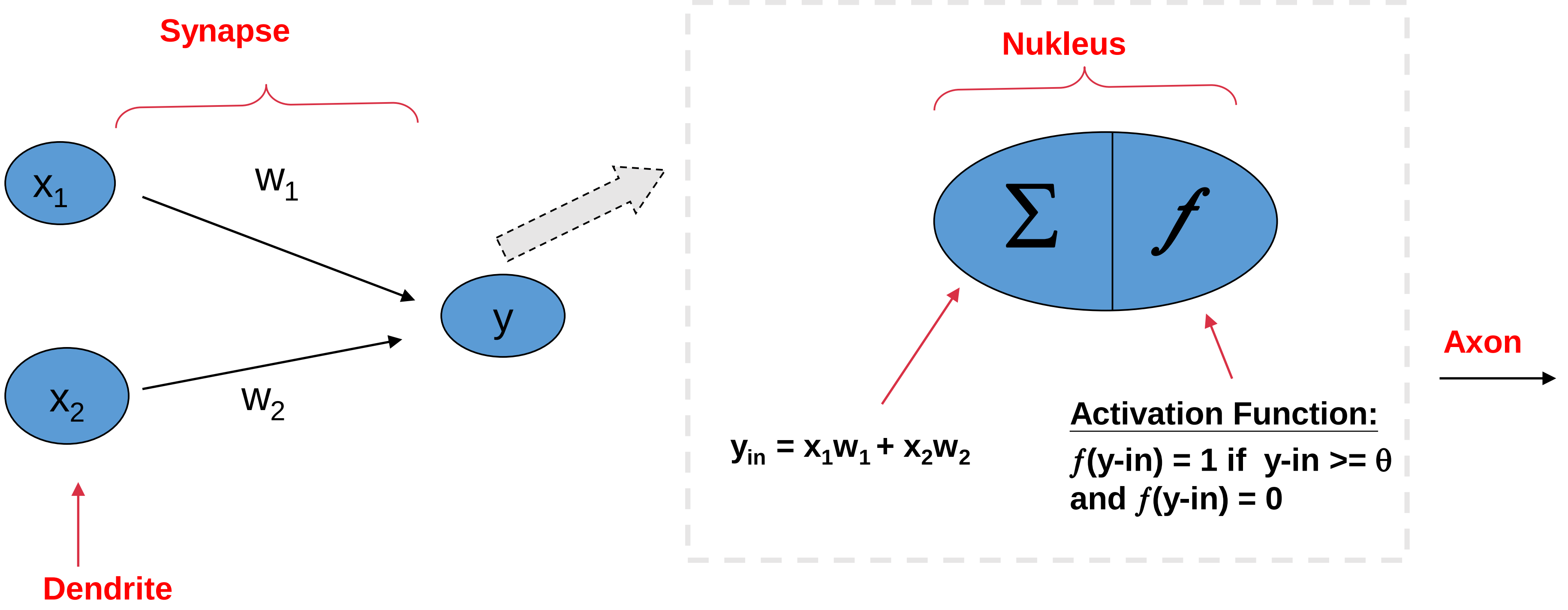
Natural neurons





- Neuron collects signals from *dendrites樹狀突*sends out spikes of electrical activity through an *axon軸索*, which splits into thousands of branches.
- At end of each brand, a *synapses突觸* converts activity into either exciting or inhibiting*抑制*activity of a dendrite at another neuron.
- *Neuron神經元fires* when exciting activity surpasses inhibitory activity
- Learning changes the effectiveness of the synapses

Artificial Neural Network

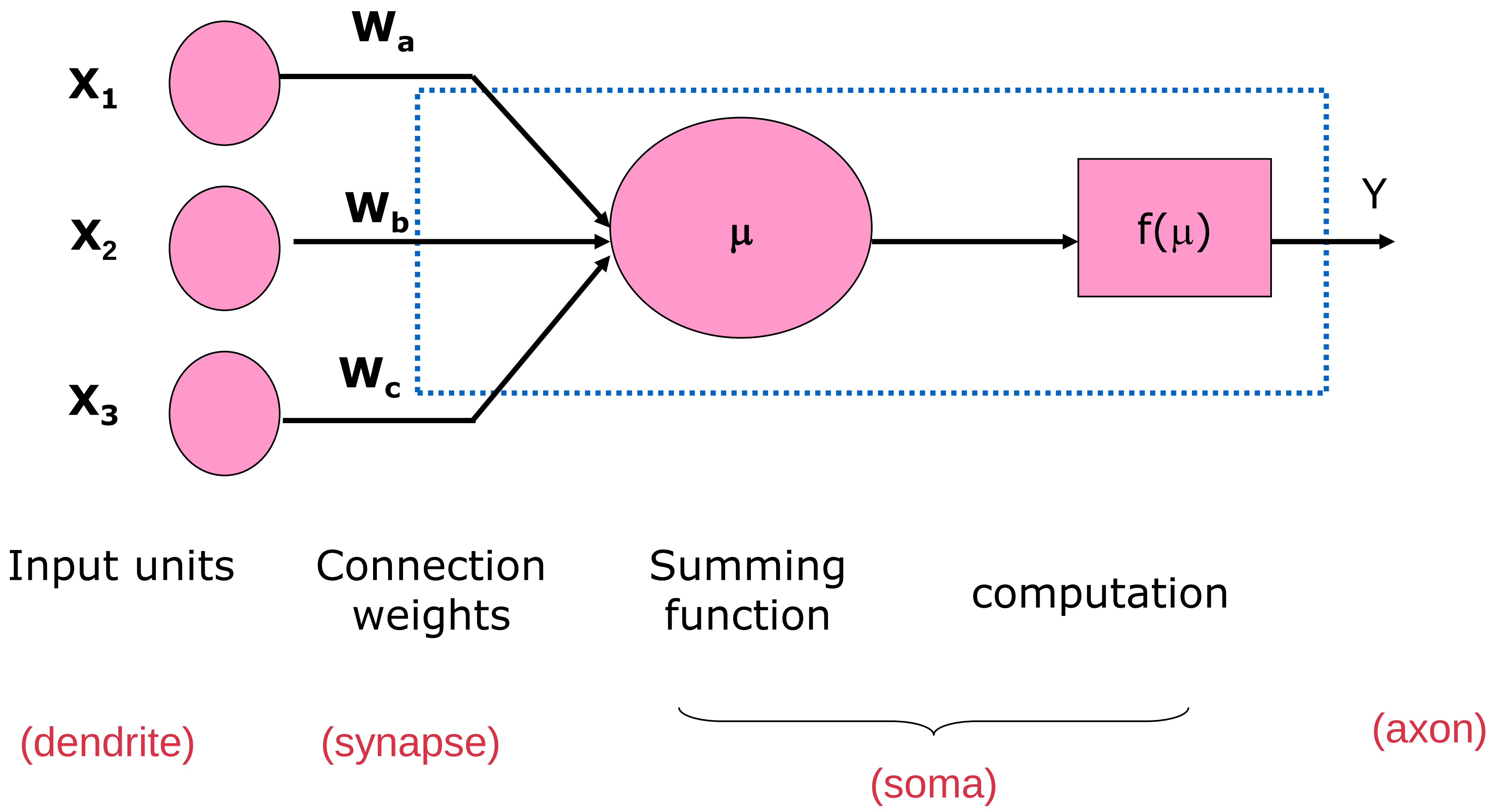


-A neuron receives input, determines the strength or the weight of the input, calculates the total weighted input, and compares the total weighted with a value (threshold)

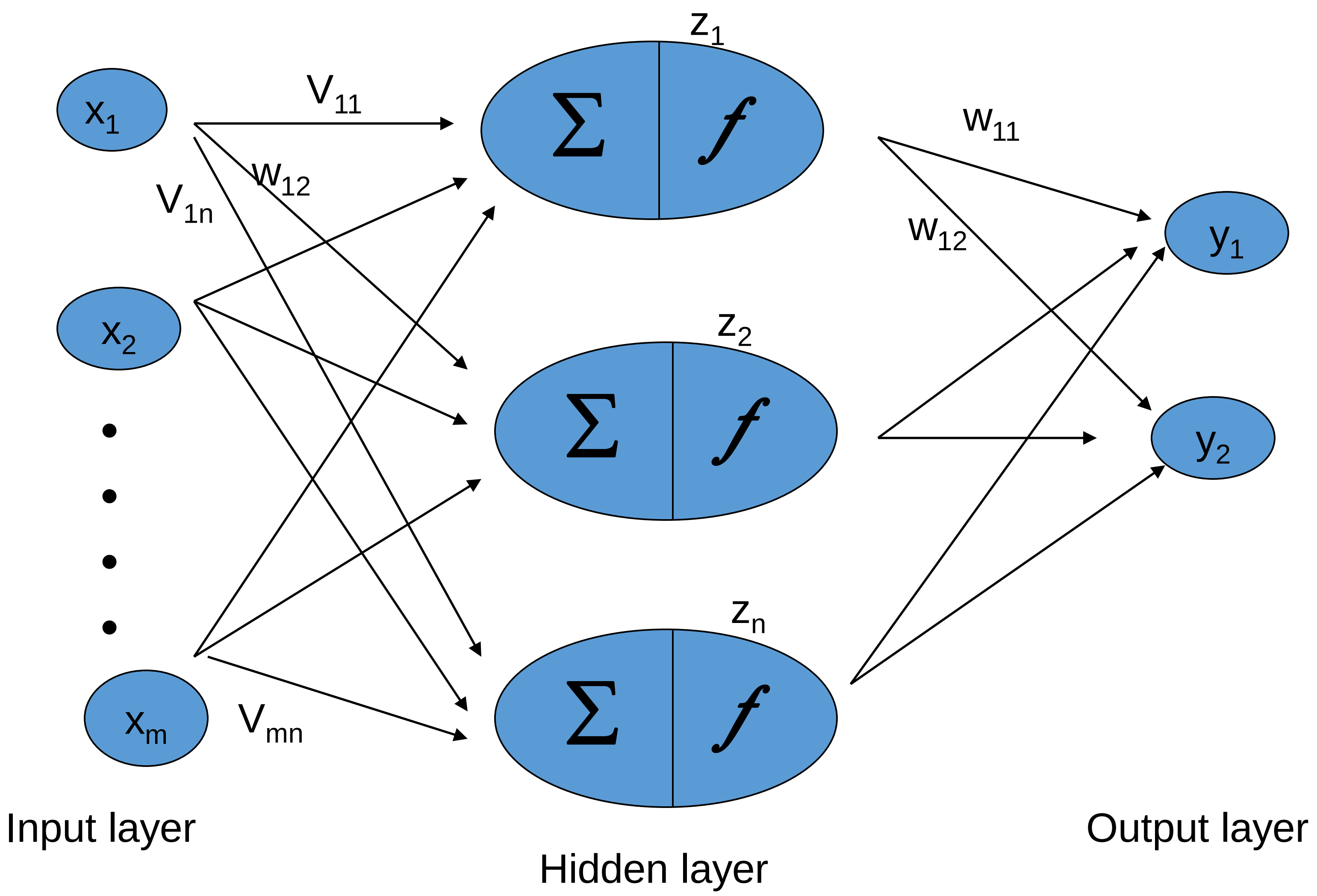
-The value is in the range of 0 and 1

- If the total weighted input greater than or equal the threshold value, the neuron will produce the output, and if the total weighted input less than the threshold value, no output will be produced

Model Of A Neuron



Multilayer Neural Network



CCN, GRNN, MADALINE, MLFF with BP, Neocognitron, RBF, RCE

Strategy / Learning Algorithm

Supervised Learning

- Learning is performed by presenting pattern with target
- During learning, produced output is compared with the desired output
  - The difference between both output is used to modify learning weights according to the learning algorithm
- Recognizing hand-written digits, pattern recognition and etc.
- Neural Network models: perceptron, feed-forward, radial basis function, support vector machine.

Unsupervised Learning

- Targets are not provided
- Appropriate for clustering task
  - Find similar groups of documents in the web, content addressable memory, clustering.
- Neural Network models: Kohonen, self organizing maps, Hopfield networks.

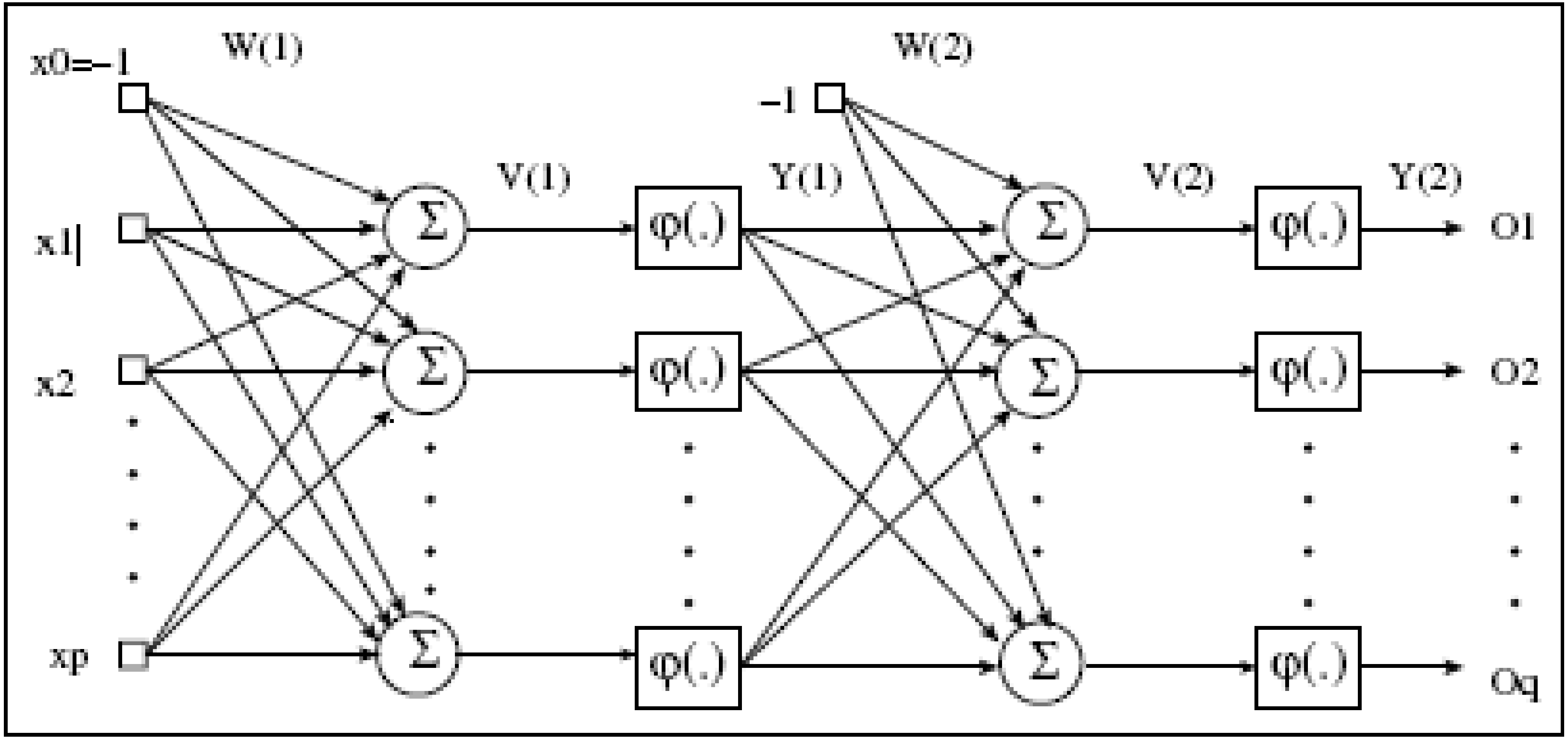
Reinforcement Learning

- Target is provided, but the desired output is absent.
- The net is only provided with guidance to determine the produced output is correct or vise versa.
- Weights are modified in the units that have errors

ANN Forward Propagation

- Bias Nodes
  - Add one node to each layer that has constant output
- Forward propagation
  - Calculate from input layer to output layer
  - For each neuron:
    - Calculate weighted average of input
    - Calculate activation function

ANN Forward Propagation



ANN Forward Propagation Calculations

- Apply input vector **X** to layer of neurons.
- Calculate

$$V_j(n) = \sum_{i=1}^{p} (W_{ji} X_i + Threshold)$$

  - where X_i is the activation of previous layer neuron i
  - W_{ji} is the weight of going from node i to node j
  - p is the number of neurons in the previous layer

- Calculate output activation(Logistic activation function)

$$Y_j(n) = \frac{1}{1 + \exp(-V_j(n))}$$

Neuron Model

- Firing Rules:
  - Threshold rules:
    - Calculate weighted average of input
    - Fire if larger than threshold
  - Perceptron rule
    - Calculate weighted average of input
    - Output activation level is

$$\phi(\nu) = \begin{cases} 1 & \nu \geq \frac{1}{2} \\ \nu & 0 \leq \nu \leq \frac{1}{2} \\ 0 & \nu \leq 0 \end{cases}$$

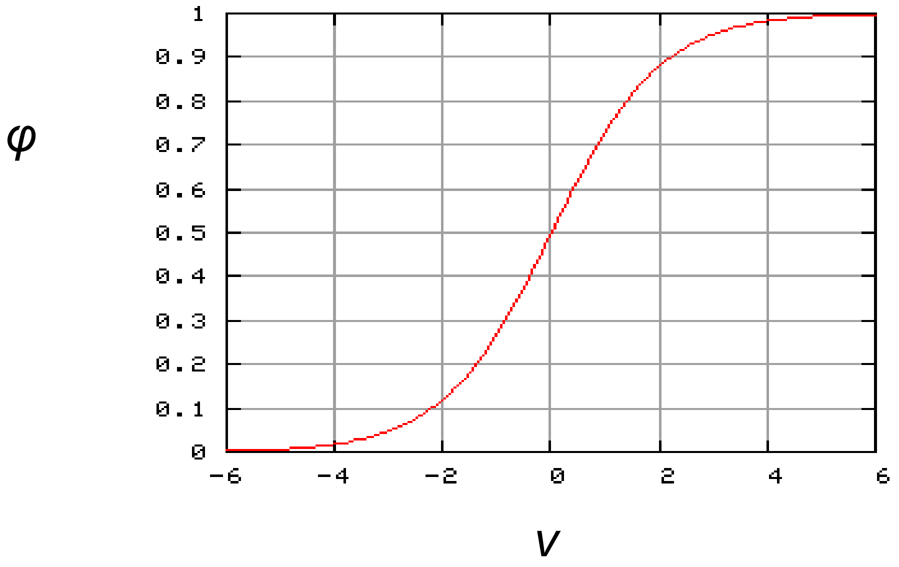
Neuron Model

- Firing Rules: Sigmoid functions
  - Hyperbolic tangent function

$$\varphi(v) = \tanh(v/2) = \frac{1-\exp(-v)}{1+\exp(-v)}$$

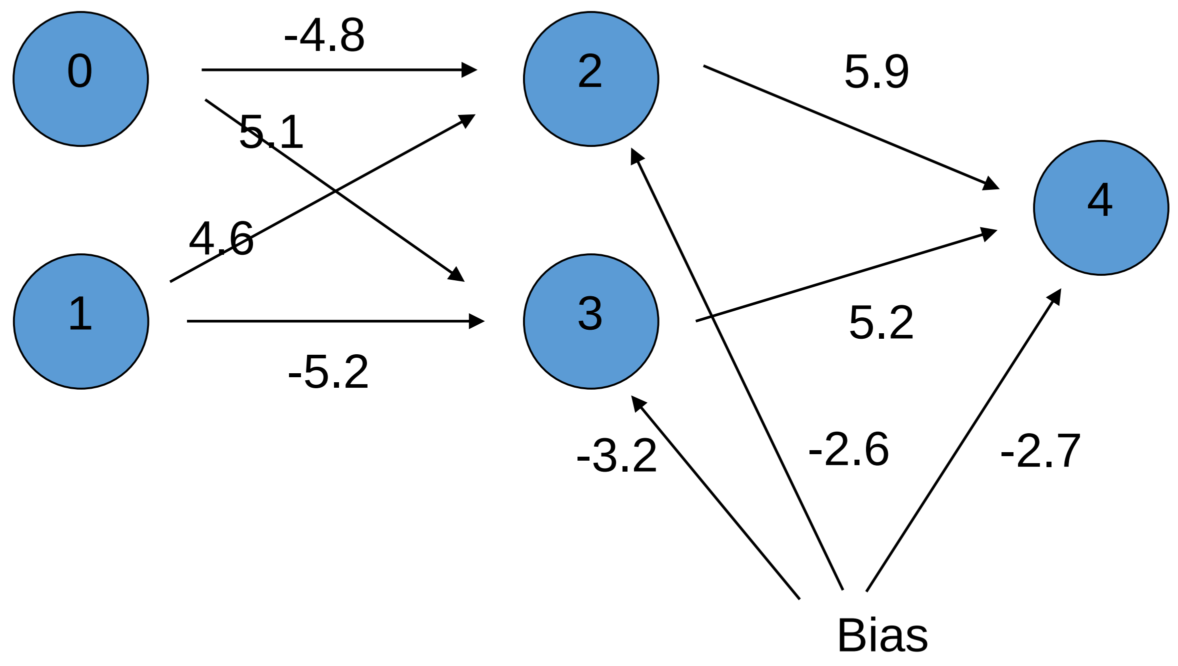
  - Logistic activation function

$$\varphi(v) = \frac{1}{1+\exp(-v)}$$



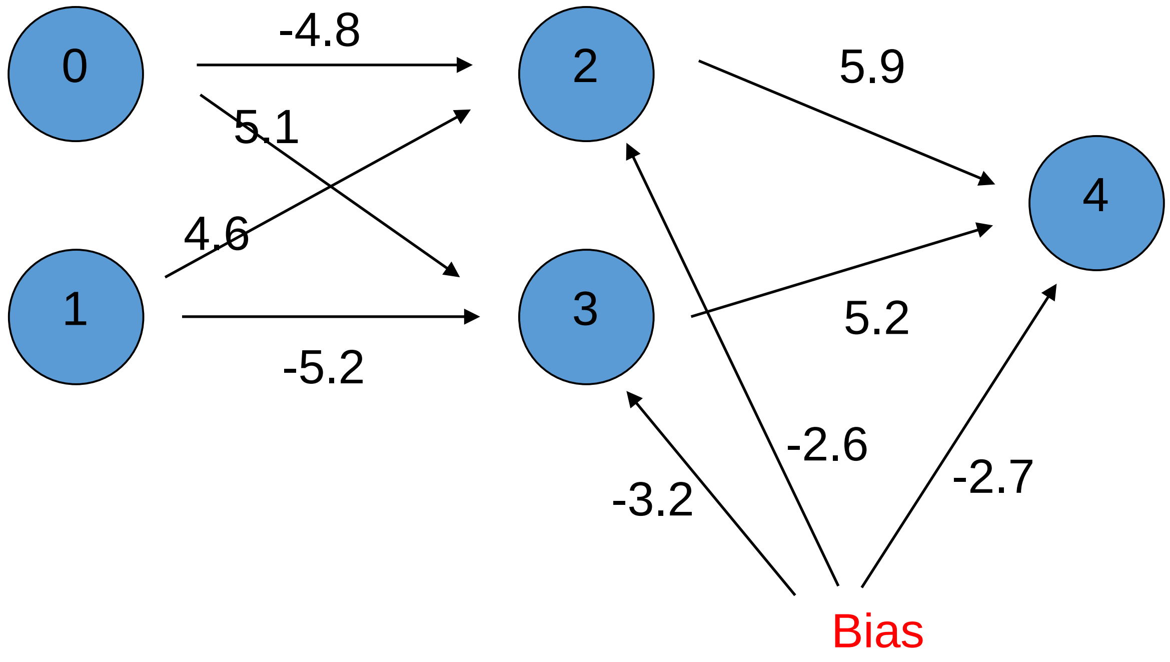
ANN Forward Propagation Step by Step

- Example: Three layer network
  - Calculates xor of inputs



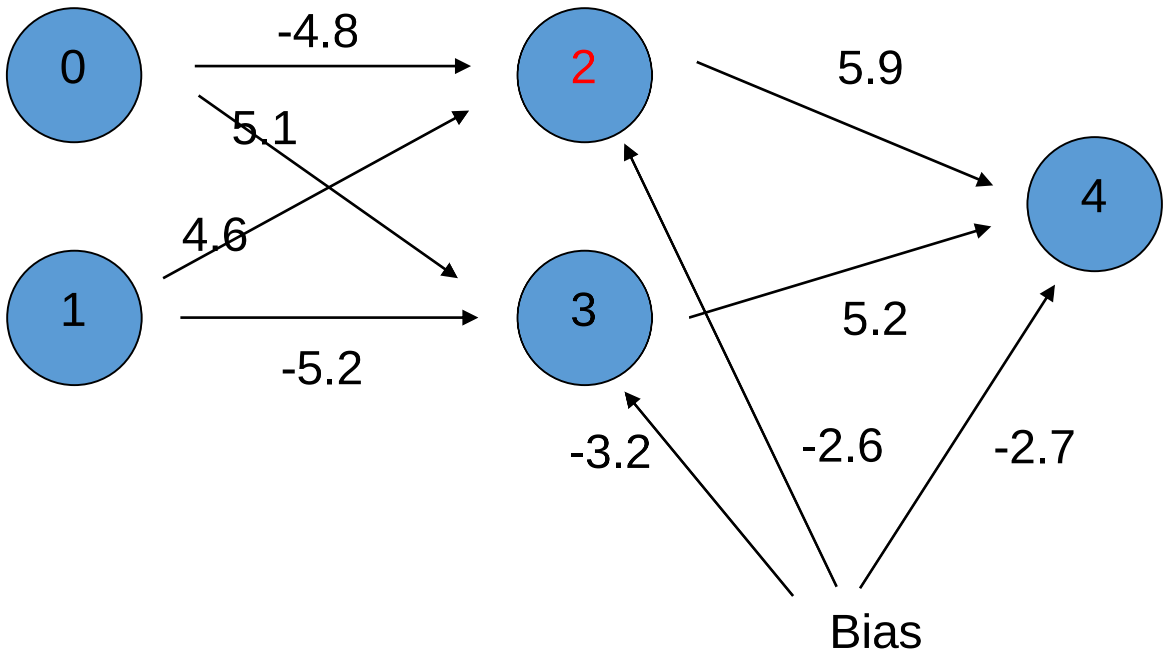
ANN Forward Propagation

- Input (0,0)



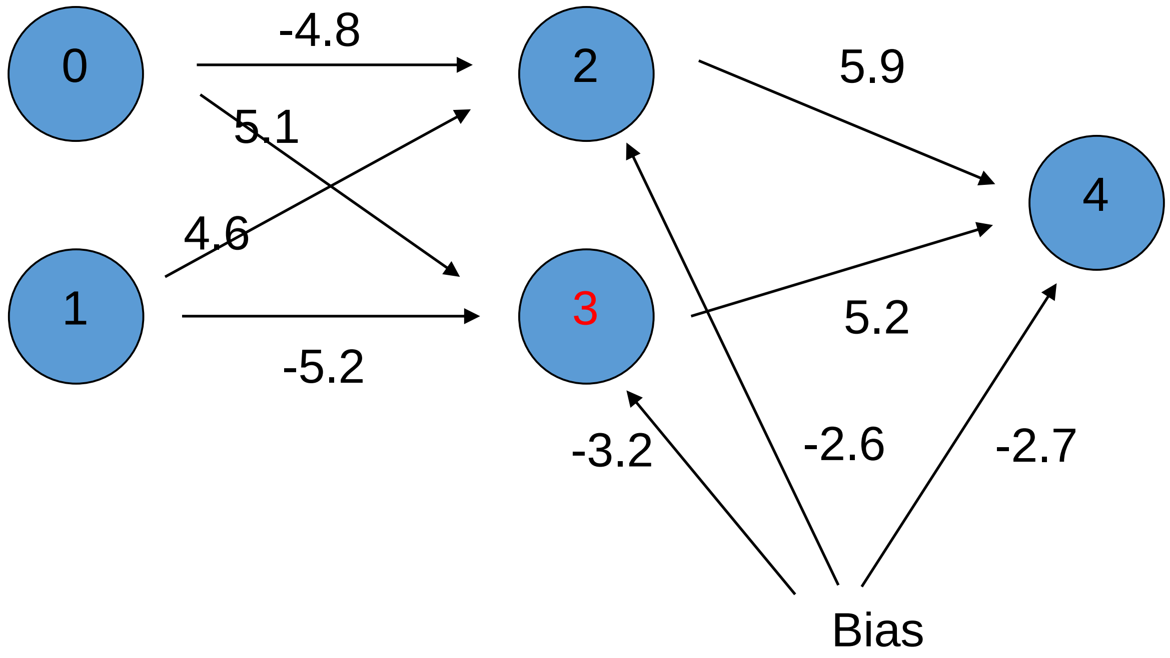
ANN Forward Propagation

- Input (0,0)
  - Node 2 activation is $\varphi(-4.8 \cdot 0+4.6 \cdot 0 - 2.6)$= $1/(1+e^{-(-2.6)})$= 0.0691



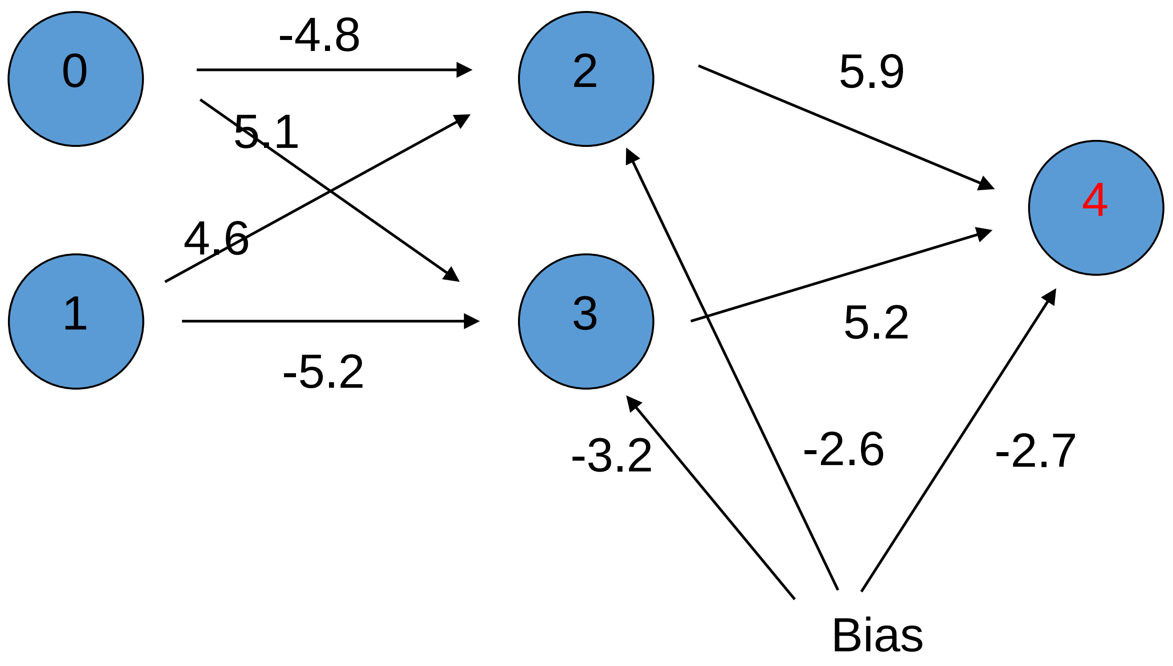
ANN Forward Propagation

- Input (0,0)
  - Node 3 activation is $\varphi(5.1 \cdot 0 - 5.2 \cdot 0 - 3.2) = 0.0392$



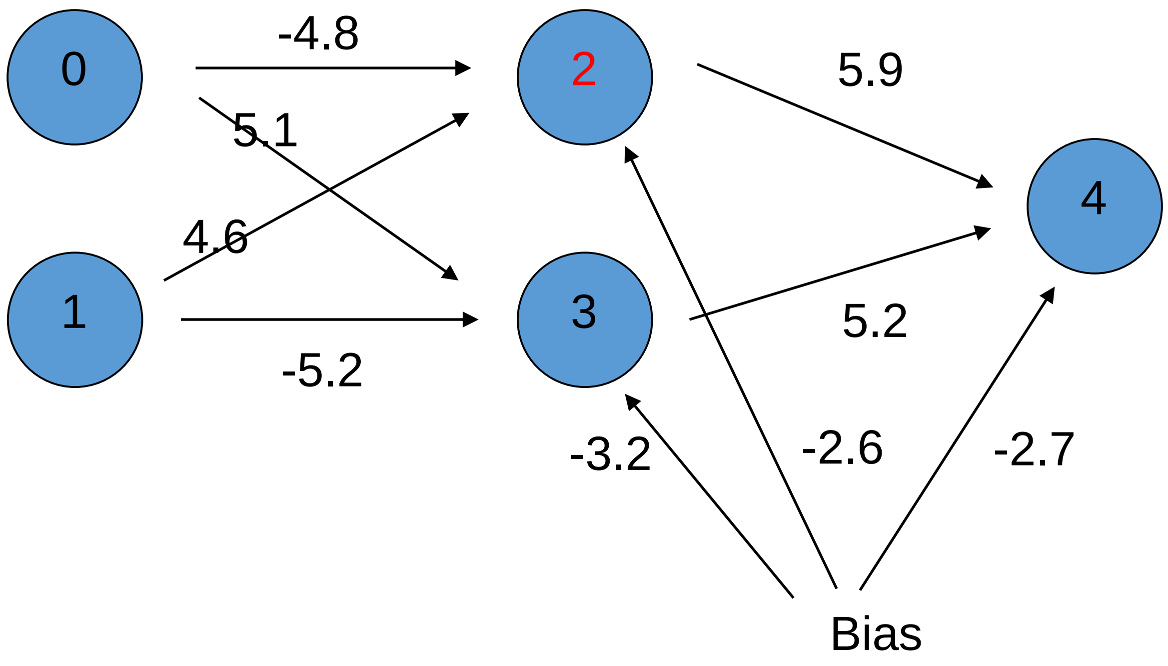
ANN Forward Propagation

- Input (0,0)
  - Node 4 activation is $\varphi(5.9 \cdot 0.069 + 5.2 \cdot 0.0392 - 2.7)=$ 0.110227



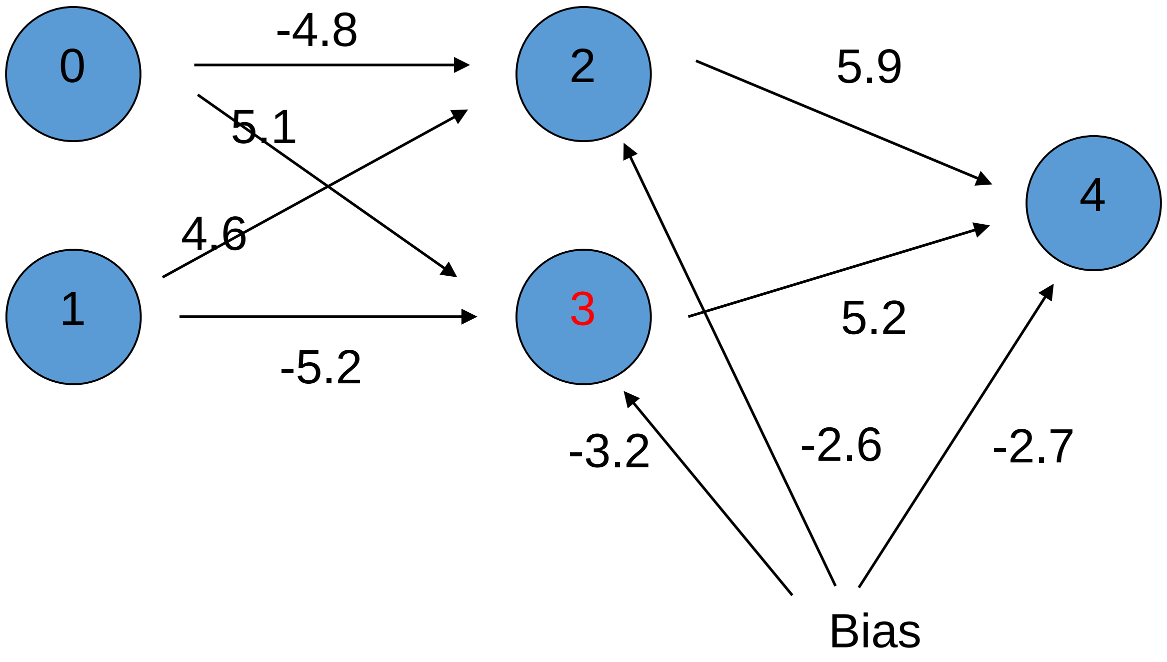
ANN Forward Propagation

- Input (0,1)
  - Node 2 activation is $\varphi(0x(-4.8)+1x4.6 -2.6)=$ 0.153269



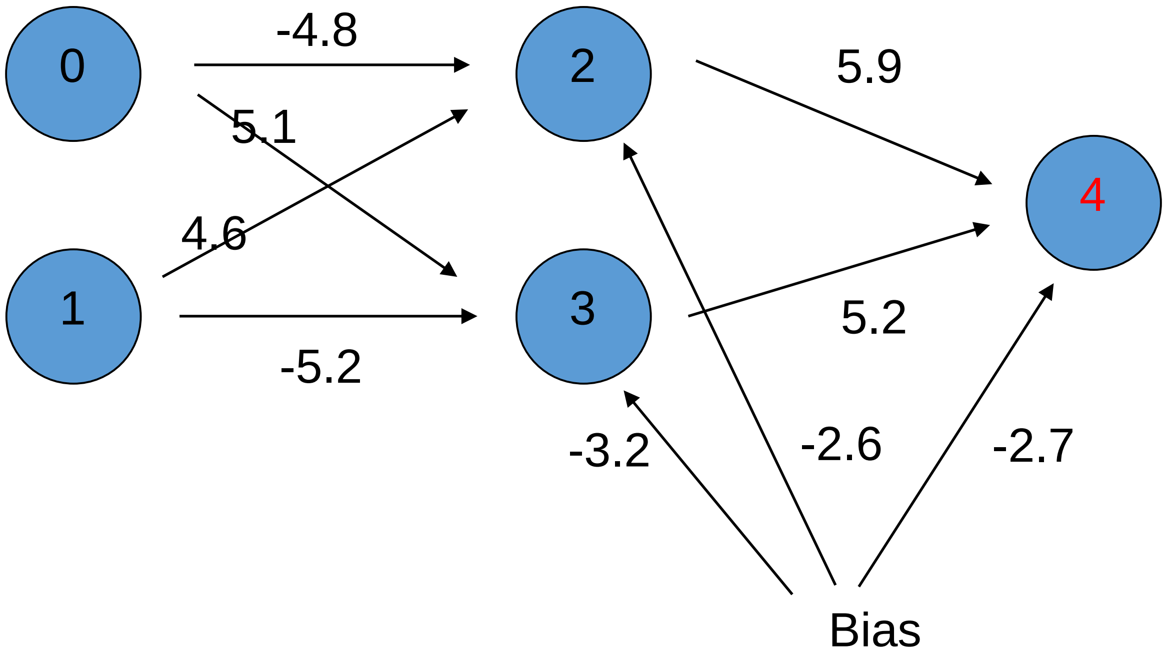
ANN Forward Propagation

- Input (0,1)
  - Node 3 activation is $\varphi(-5.2\ -3.2)=\ 0.000224817$

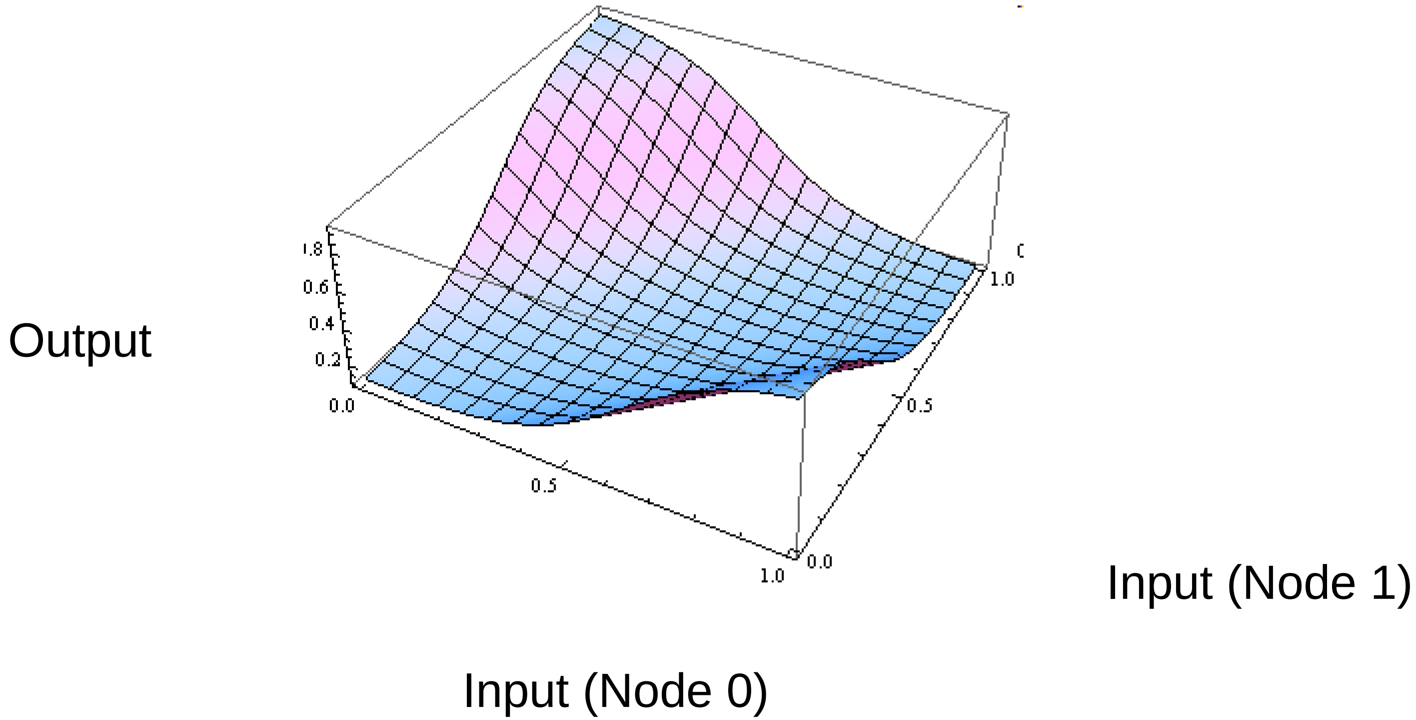


ANN Forward Propagation

- Input (0,1)
  - Node 4 activation is $\varphi(5.9 \cdot 0.153269 + 5.2 \cdot 0.000224817 - 2.7) = 0.923992$



ANN Forward Propagation



ANN Forward Propagation

- Network can learn a non-linearly separated set of outputs.
- Need to map output (real value) into binary values.

Back Propagation

- 01.mp4

ANN Training

- Weights are determined by *training*
  - Back-propagation:
    - On given input, compare actual output to desired output.
    - Adjust weights to output nodes.
    - Work backwards through the various layers
  - Start out with initial random weights
    - Best to keep weights close to zero (<<10)

ANN Training

- Weights are determined by *training*
  - Need a training set
    - Should be representative of the problem
  - During each training epoch:
    - Submit training set element as input
    - Calculate the error for the output neurons
    - Calculate average error during epoch
    - Adjust weights

ANN Training

- Error is the mean square of differences in output layer

$$E(\vec{x}) = \frac{1}{2}\sum_{k=1}^{K}(y_k(\vec{x}) - t_k(\vec{x}))^2$$

y – observed output

t – target output

ANN Training

- Error of training epoch is the average of all errors.

ANN Training

- Update weights and thresholds using
  - Weights

$$w_{j,k} = w_{j,k} + (-\eta)\frac{\partial E(\vec{x})}{\partial w_{jk}}$$

  - Bias

$$\theta_k = \theta_k + (-\eta)\frac{\partial E(\vec{x})}{\partial \theta_k}$$

  - η is a possibly time-dependent factor that should prevent overcorrection

ANN Training

- Using a sigmoid function, we get

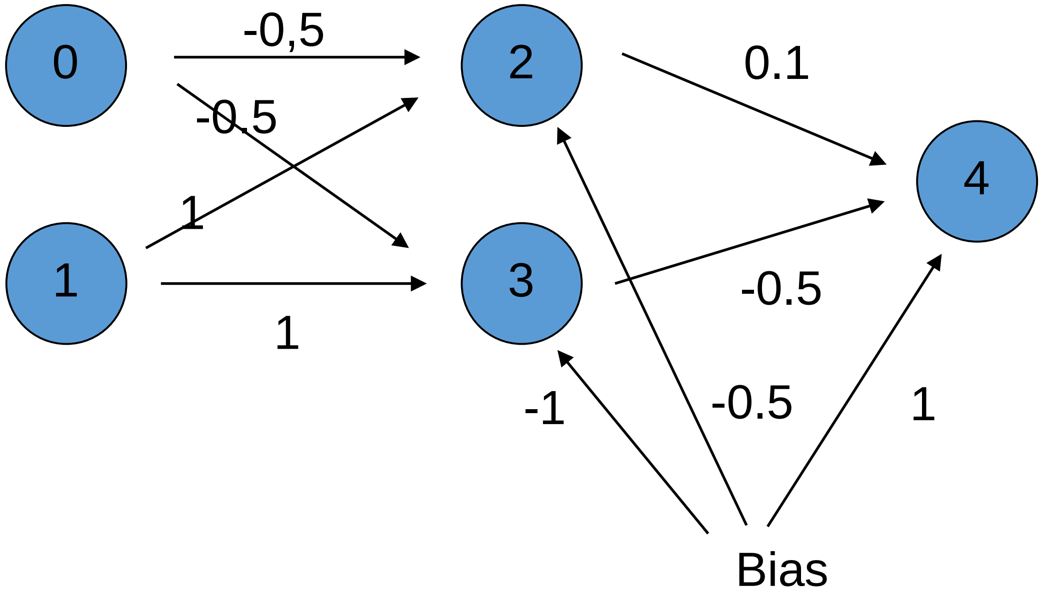
$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\mathrm{net}_j)(t_j - y_j)$$

  - Logistics function φ has derivative φ'(t) = φ(t)(1- φ(t))

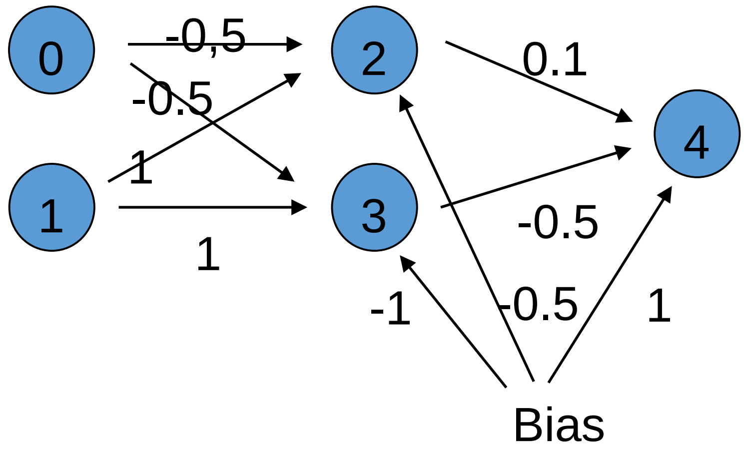
ANN Training Example

- Start out with random, small weights



x1	x2	y
0	0	0.687349
0	1	0.667459
1	0	0.698070
1	1	0.676727

ANN Training Example



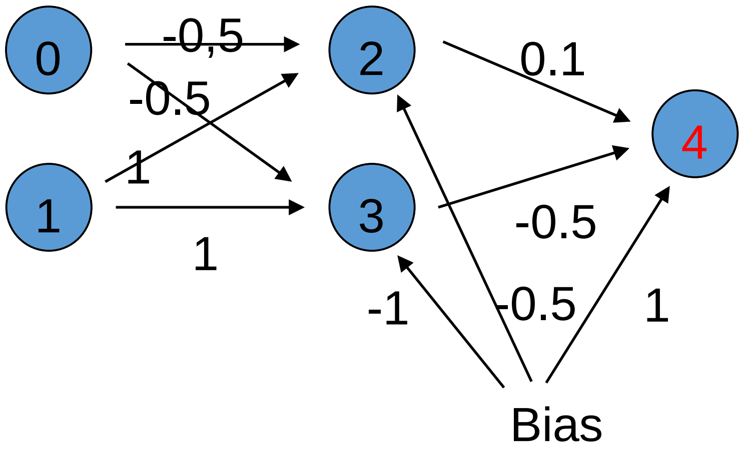
x1	x2	y	Error
0	0	0.69	0.472448
0	1	0.67	0.110583
1	0	0.70	0.0911618
1	1	0.68	0.457959

Average Error is 0.283038

ANN Training Example

- Calculate the derivative of the error with respect to the weights and bias into the output layer neurons

ANN Training Example



New weights going into node 4

We do this for all training inputs, then average out the changes

net_4 is the weighted sum of input going into neuron 4:

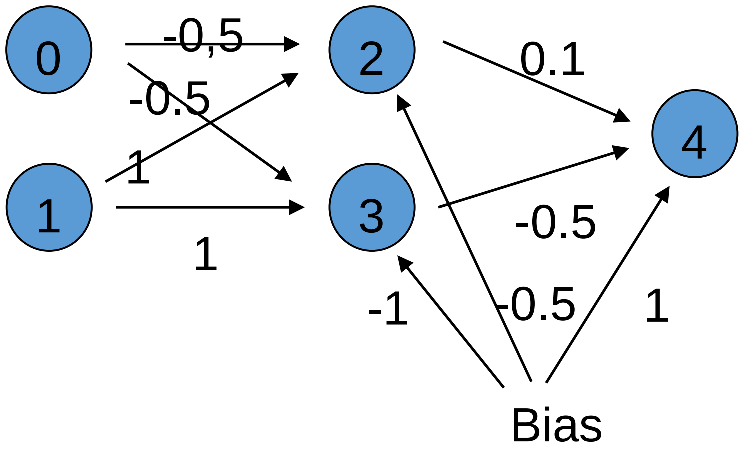
$net_4(0,0)$= 0.787754

net4(0,1)= 0.696717

net4(1,0)= 0.838124

net4(1,1)= 0.73877

ANN Training Example



New weights going into node 4

We calculate the derivative of the activation function at the point given by the net-input.

Recall our cool formula

$$\varphi'(t) = \varphi(t)(1- \varphi(t))$$

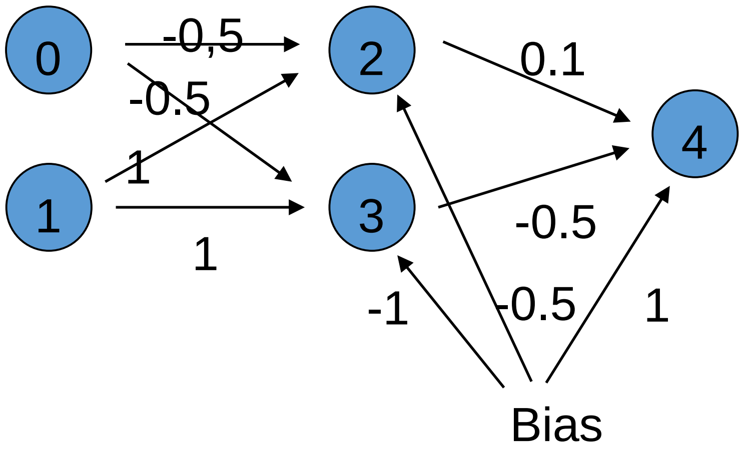
$\varphi'(\, net_4(0,0)) = \varphi'(\, 0.787754) = 0.214900$

$\varphi'(\, net_4(0,1)) = \varphi'(\, 0.696717) = 0.221957$

$\varphi'(\, net_4(1,0)) = \varphi'(\, 0.838124) = 0.210768$

$\varphi'(\, net_4(1,1)) = \varphi'(\, 0.738770) = 0.218768$

ANN Training Example



$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\text{net}_j)(t_j - y_j)$$

New weights going into node 4

We now obtain δ values for each input separately:

Input 0,0:

δ_4= φ'(net$_4$(0,0)) *(0-y$_4$(0,0)) = -0.152928

Input 0,1:

δ_4= φ'(net4(0,1)) *(1-y$_4$(0,1)) = 0.0682324

Input 1,0:

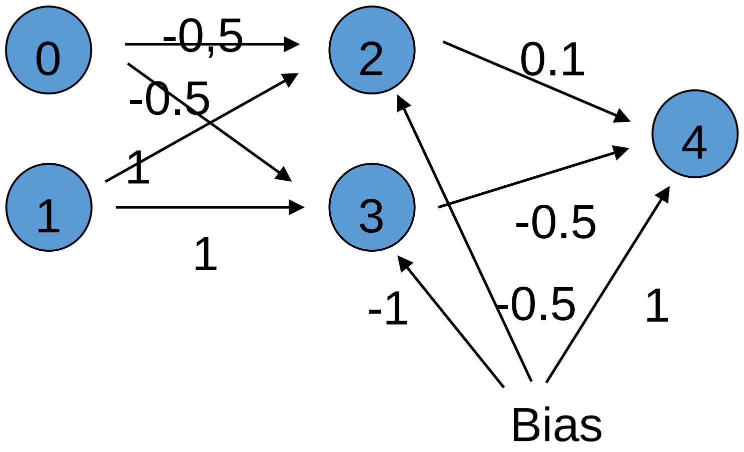
δ_4= φ'(net$_4$(1,0)) *(1-y$_4$(1,0)) = 0.0593889

Input 1,1:

δ_4= φ'(net$_4$(1,1)) *(0-y$_4$(1,1)) = -0.153776

Average: δ_4 = -0.0447706

ANN Training Example



$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\text{net}_j)(t_j - y_j)$$

New weights going into node 4

Average: δ_4 = -0.0447706

We can now update the weights going into node 4:

Let's call: E_{ji} the derivative of the error function with respect to the weight going from neuron *i* into neuron *j*.

We do this for every possible input:

$E_{4,2}$ = - output(neuron(2)* δ_4

For (0,0): $E_{4,2}$ = 0.0577366

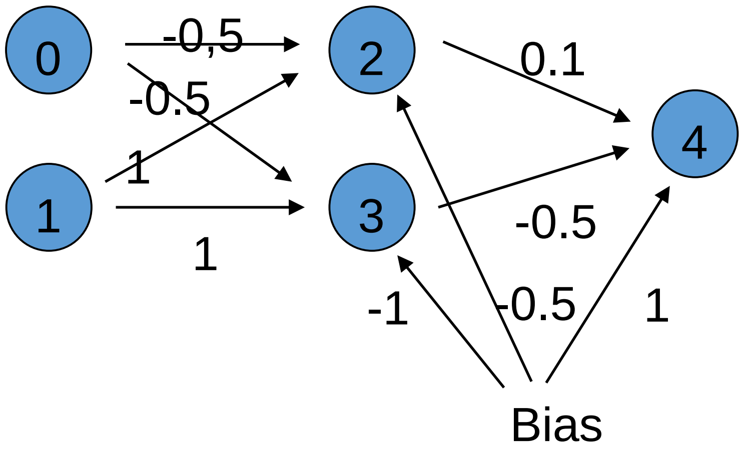
For (0,1): $E_{4,2}$ = -0.0424719

For(1,0): $E_{4,2}$ = -0.0159721

For(1,1): $E_{4,2}$ = 0.0768878

Average is 0.0190451

ANN Training Example

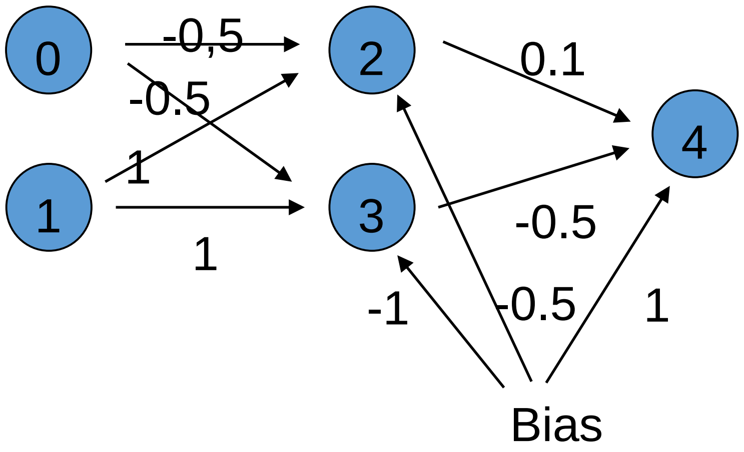


New weight from 2 to 4 is now going to be 0.1190451.

$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\mathrm{net}_j)(t_j - y_j)$$

ANN Training Example



$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\text{net}_j)(t_j - y_j)$$

New weights going into node 4

For (0,0): $E_{4,3}$ = 0.0411287

For (0,1): $E_{4,3}$ = -0.0341162

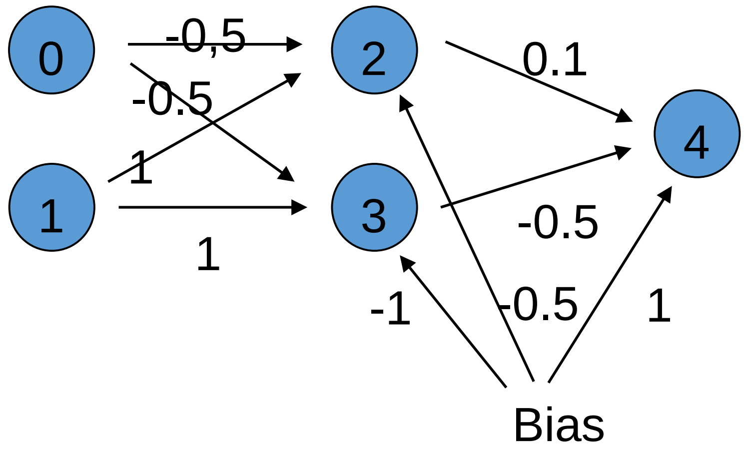
For(1,0): $E_{4,3}$ = -0.0108341

For(1,1): $E_{4,3}$ = 0.0580565

Average is 0.0135588

New weight is -0.486441

ANN Training Example



$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\text{net}_j)(t_j - y_j)$$

New weights going into node 4:

We also need to change the bias node

For (0,0): $E_{4,B}$ = 0.0411287

For (0,1): $E_{4,B}$ = -0.0341162

For(1,0): $E_{4,B}$ = -0.0108341

For(1,1): $E_{4,B}$ = 0.0580565

Average is 0.0447706

New weight is 1.0447706

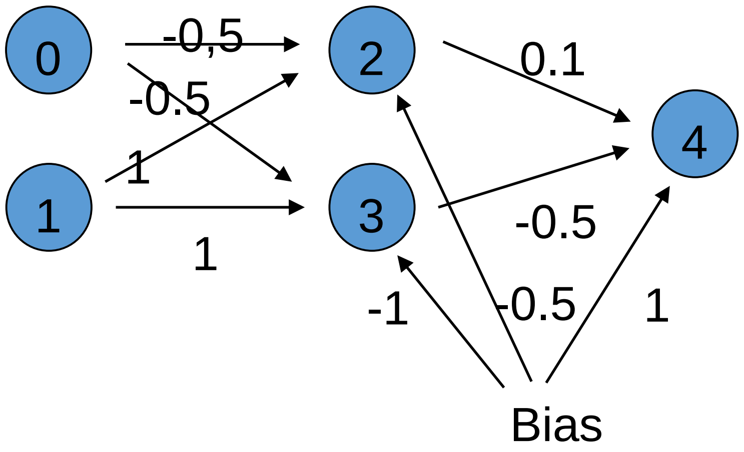
ANN Training Example

- We now have adjusted all the weights into the output layer.
- Next, we adjust the hidden layer
- The target output is given by the delta values of the output layer
- More formally:
  - Assume that *j* is a hidden neuron
  - Assume that δ_k is the delta-value for an output neuron *k*.
  - While the example has only one output neuron, most ANN have more. When we sum over *k,* this means summing over all output neurons.
  - w_{kj} is the weight from neuron *j* into neuron *k*

$$\delta_j = \varphi'(\text{net}_j) \cdot \sum_k (\delta_k w_{kj})$$

$$\frac{\partial E}{\partial w_{ji}} = -y_i \delta_j$$

ANN Training Example



$$\delta_j = \varphi'(\text{net}_j) \cdot \sum_k (\delta_k w_{kj})$$

$$\frac{\partial E}{\partial w_{ji}} = -y_i \delta_j$$

We now calculate the updates to the weights of neuron 2.

First, we calculate the net-input into 2.

This is really simple because it is just a linear functions of the arguments x_1 and x_2

$net_2 = -0.5\ x_1 + x2 - 0.5$

We obtain

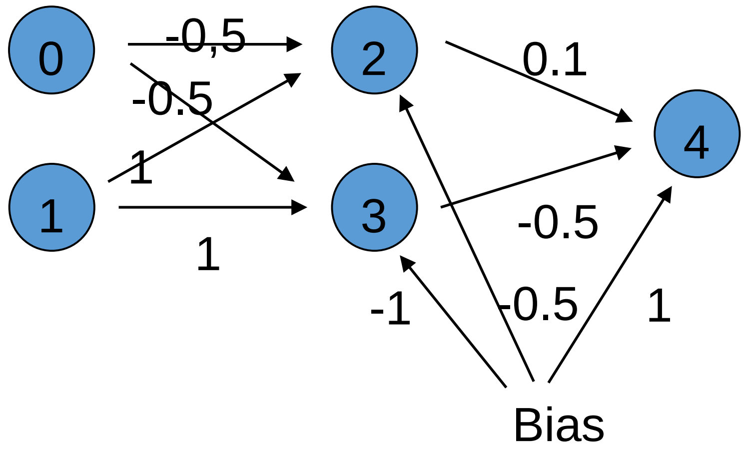
$\delta_2 (0,0) = -\ 0.00359387$

$\delta_2(0,1) = \quad 0.00160349$

$\delta_2(1,0) = \quad 0.00116766$

$\delta_2(1,1) = -\ 0.00384439$

ANN Training Example



Call E_{20} the derivative of E with respect to w_{20}. We use the output activation for the neurons in the previous layer (which happens to be the input layer)

$E_{20}(0,0) = -\varphi(0)\cdot\delta_2(0,0) = 0.00179694$

$E_{20}(0,1) = 0.00179694$

$E_{20}(1,0) = -0.000853626$

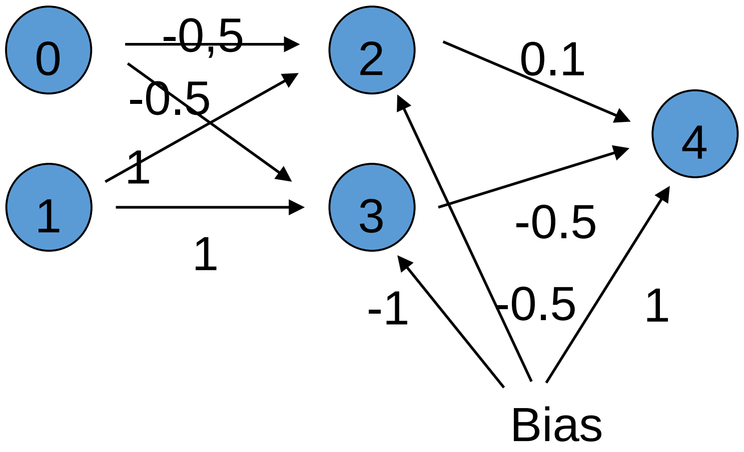
$E_{20}(1,1) = 0.00281047$

$$\delta_j = \varphi'(\text{net}_j)\cdot\sum_k(\delta_k w_{kj})$$

$$\frac{\partial E}{\partial w_{ji}} = -y_i\delta_j$$

The average is 0.00073801 and the new weight is -0.499262

ANN Training Example



$$\delta_j = \varphi'(\mathrm{net}_j) \cdot \sum_k (\delta_k w_{kj})$$

$$\frac{\partial E}{\partial w_{ji}} = -y_i \delta_j$$

Call E_{21} the derivative of E with respect to w_{21}. We use the output activation for the neurons in the previous layer (which happens to be the input layer)

$E_{21}(0,0) = -\varphi(1)\cdot\delta_2(0,0) =$ 0.00179694

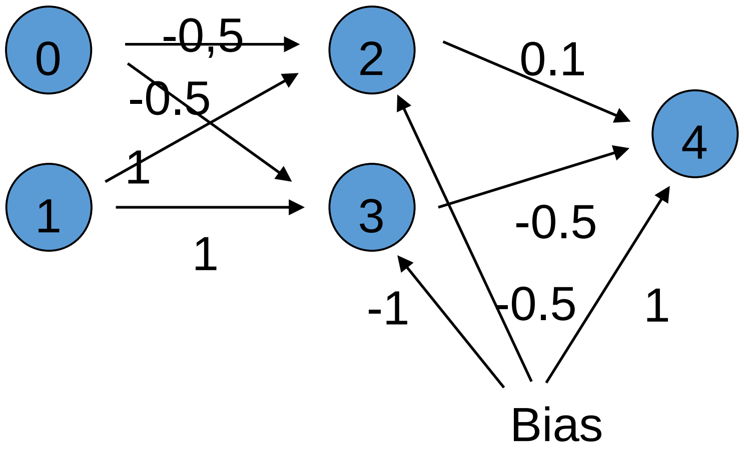
$E_{21}(0,1) =$ -0.00117224

$E_{21}(1,0) =$ -0.000583829

$E_{21}(1,1) =$ 0.00281047

The average is 0.000712835 and the new weight is 1.00071

ANN Training Example



Call E_{2B} the derivative of E with respect to w_{2B}. Bias output is always -0.5

$E_{2B}(0,0)$ = - -0.5 $\cdot\delta_2(0,0)$ = 0.00179694

$E_{2B}(0,1)$ = -0.00117224

$E_{2B}(1,0)$ = -0.000583829

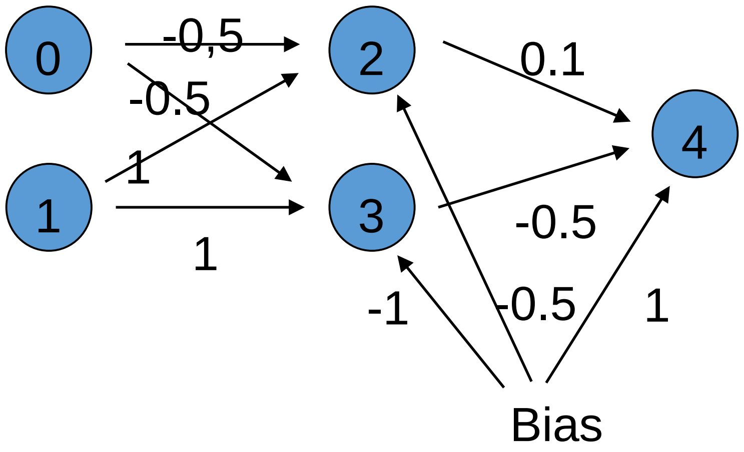
$E_{2B}(1,1)$ = 0.00281047

The average is 0.00058339 and the new weight is -0.499417

$$\delta_j = \varphi'(\text{net}_j) \cdot \sum_k (\delta_k w_{kj})$$

$$\frac{\partial E}{\partial w_{ji}} = -y_i \delta_j$$

ANN Training Example



We now calculate the updates to the weights of neuron 3.

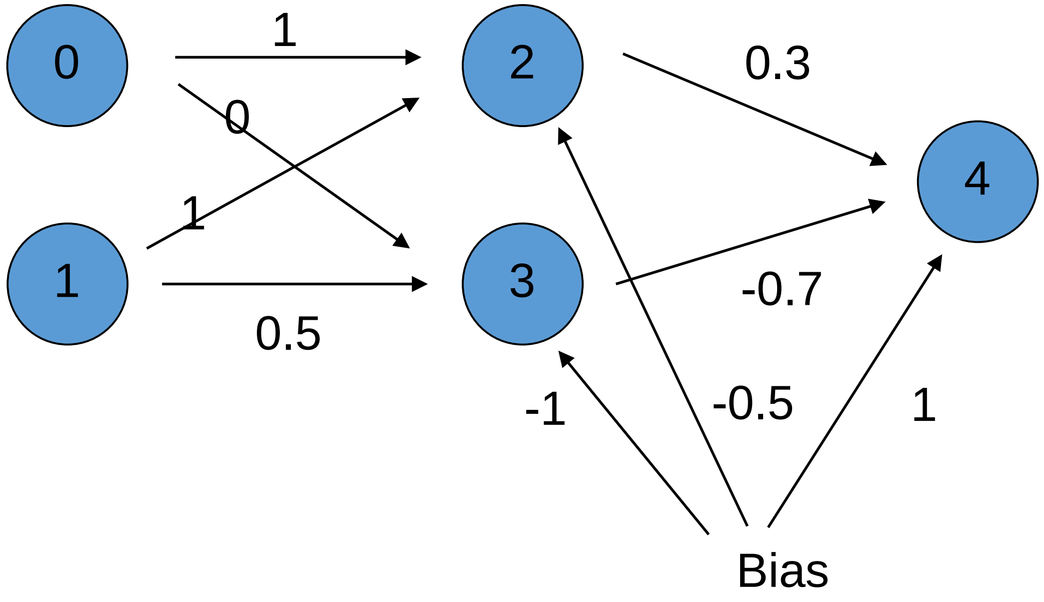
…

$$\delta_j = \varphi'(\text{net}_j) \cdot \sum_k (\delta_k w_{kj})$$

$$\frac{\partial E}{\partial w_{ji}} = -y_i \delta_j$$

ANN Training Example 2

- Start out with random, small weights



Node 0: x_0

Node 1: x_1

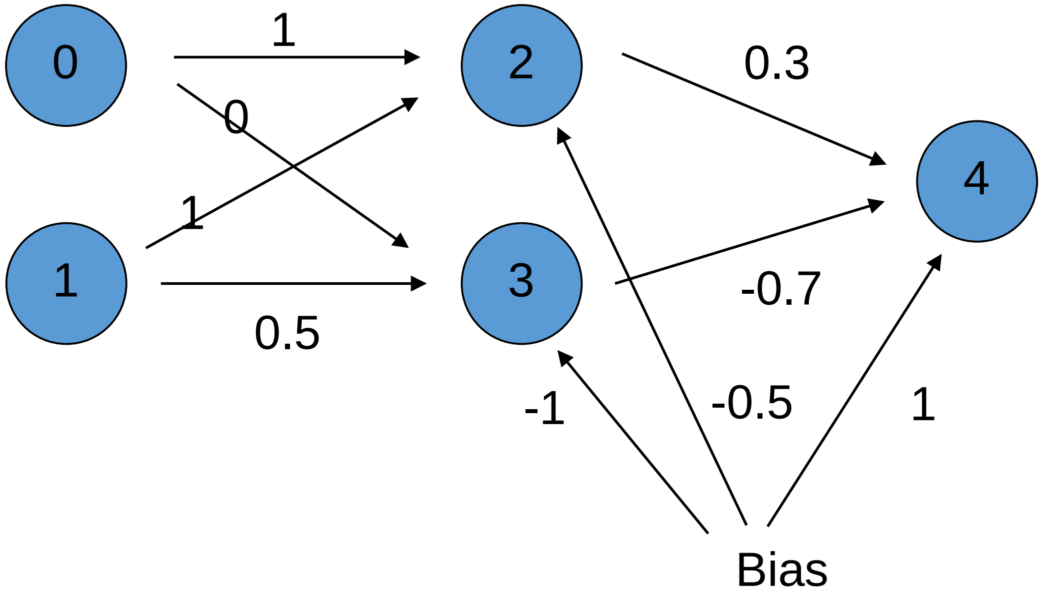
Node 2: $o_2 = \varphi(x_0 + x_1 - 0.5)$

Node 3: $o_3 = \varphi(0.5\, x_1 - 1)$

Node 4: $o_4 = \varphi(0.3\, o_2 - 0.7\, o_3 + 1)$

ANN Training Example 2

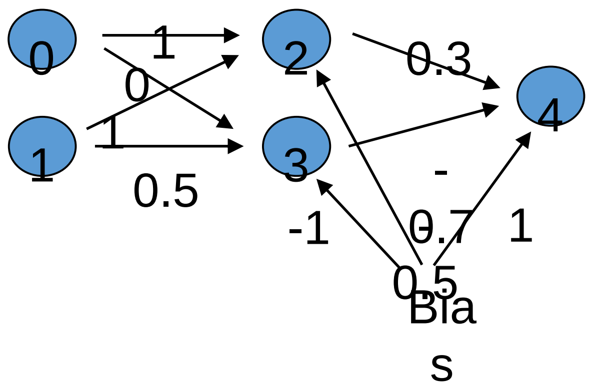
- Calculate outputs



x1	x2	$y=o_4$
0	0	0.7160
0	1	0.7155
1	0	0.7308
1	1	0.7273

ANN Training Example 2

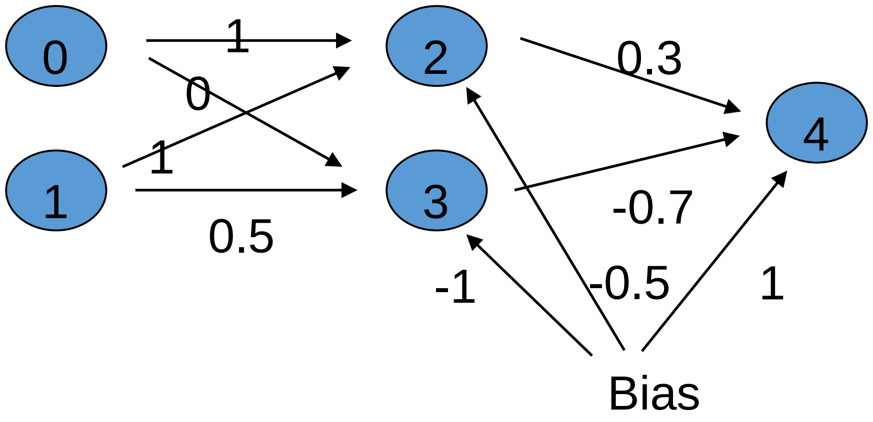
- Calculate average error to be E = 0.14939



x_0	x_1	y	t	$E=(y-t)^2/2$
0	0	0.7160	0	0.2564
0	1	0.7155	1	0.0405
1	0	0.7308	1	0.0362
1	1	0.7273	0	0.264487

ANN Training Example 2

- Calculate the change for node 4



Need to calculate net_4, the weighted input of all input into node 4

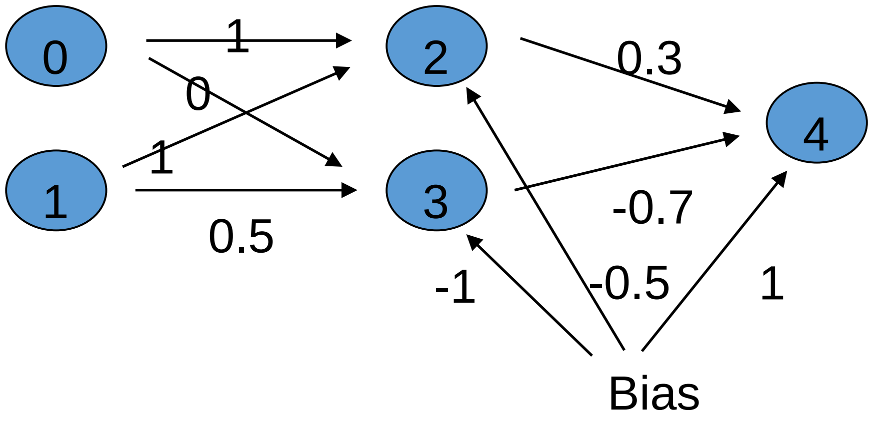
$net_4(x_0,x_1) = 0.3\ o_2(x_0,x_1) – 0.7\ o_3(x0,x1) + 1$

$net_4 = (net_4(0,0) + net_4(0,1) + net_4(1,0) + net_4(1,1))/4$

This gives 0.956734

ANN Training Example 2

- Calculate the change for node 4



$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\text{net}_j)(t_j - y_j)$$

We now calculate

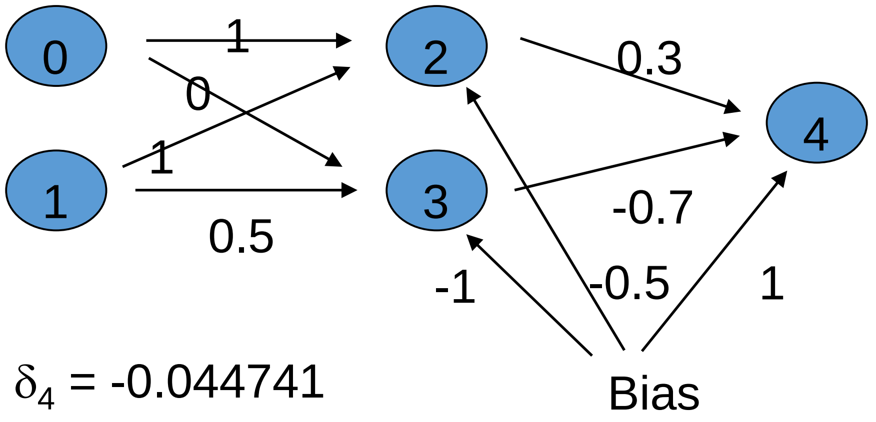
$\delta_4(0,0) = \varphi'(\text{net}_4(0,0)(0 - o_4(0,0)) = - 0.14588$

$\delta_4(0,1) = \varphi'(\text{net}4(0,1)(1 - o4(0,1)) = 0.05790$
$\delta_4(1,1) = \varphi'(\text{net}4(1,0)(0 - o4(1,0)) = 0.05297$
$\delta_4(1,1) = \varphi'(\text{net}4(1,1)(0 - o4(1,1)) = -0.14425$

On average δ_4 = -0.044741

ANN Training Example 2

- Calculate the change for node 4



$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\mathrm{net}_j)(t_j - y_j)$$

$\delta_4 = -0.044741$

We can now update the weights for node 4

$E_{4,2}(0,0) = -o_2(0,0)^* \ \delta_4 = 0.01689$

$E_{4,2}(0,1) = -o_2(0,1)^* \ \delta_4 = 0.02785$

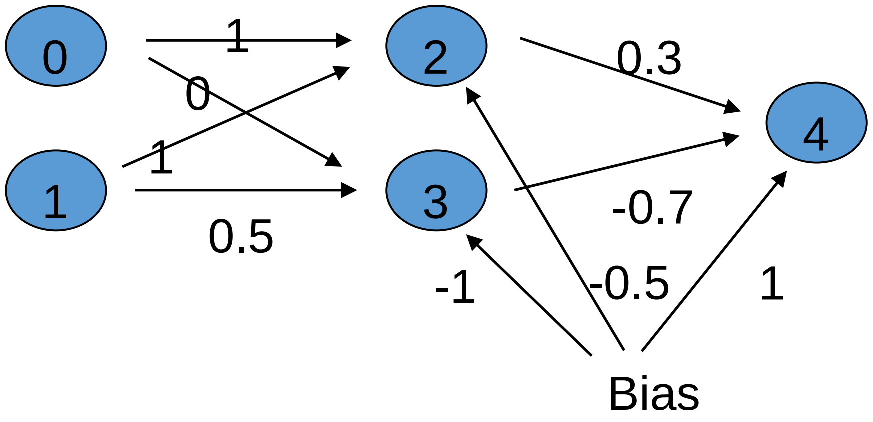
$E_{4,2}(1,0) = -o_2(1,0)^* \ \delta_4 = 0.02785$

$E_{4,2}(0,0) = -o_2(0,0)^* \ \delta_4 = 0.03658$

with average 0.00708

ANN Training Example 2

- Calculate the change for node 4



$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\text{net}_j)(t_j - y_j)$$

$E_{4,2} = 0.00708$

Therefore, new weight w_{42} is 0.2993

ANN Training

- ANN Back-propagation is an empirical algorithm

ANN Training

- XOR is too simple an example, since quality of ANN is measured on a finite sets of inputs.
- More relevant are ANN that are trained on a training set and unleashed on real data

ANN Training

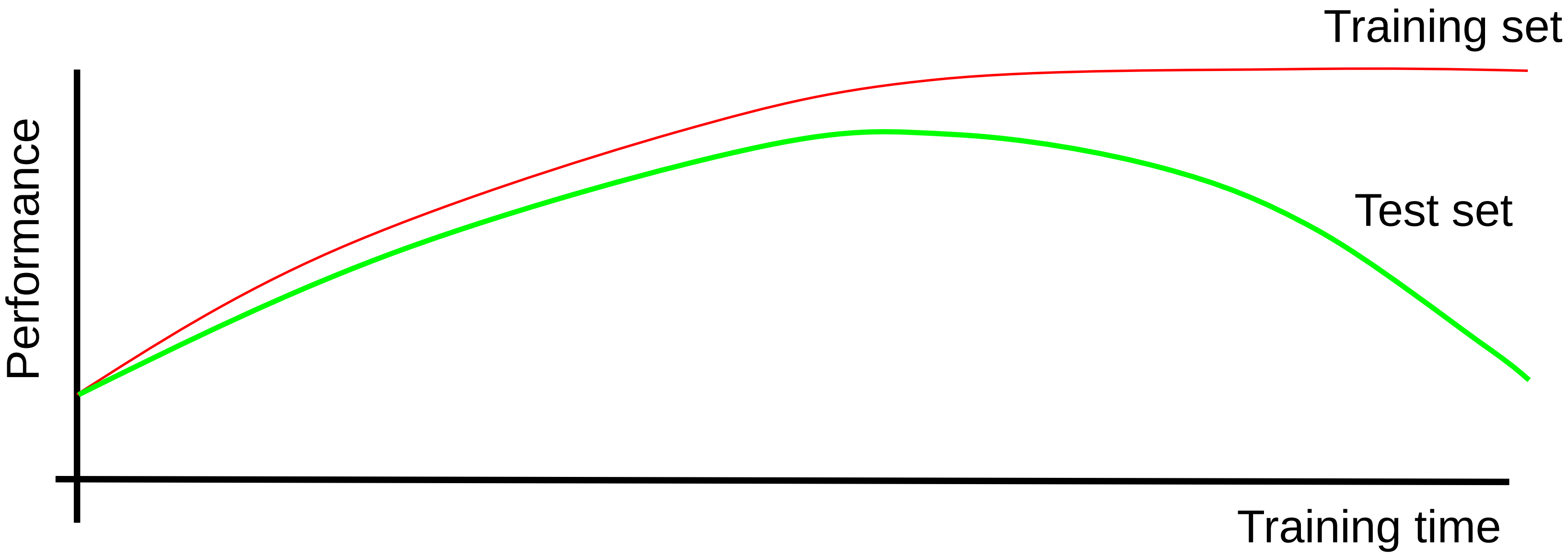
- Need to measure effectiveness of training
  - Need training sets
  - Need test sets.
- There can be no interaction between test sets and training sets.
  - Example of a Mistake:
      - Train ANN on training set.
      - Test ANN on test set.
      - Results are poor.
      - Go back to training ANN.
    - After this, there is no assurance that ANN will work well in practice.
      - In a subtle way, the test set has become part of the training set.

ANN Training

- Convergence
  - ANN back propagation uses gradient decent.
  - Naïve implementations can
    - overcorrect weights
    - undercorrect weights
  - In either case, convergence can be poor
- Stuck in the wrong place
  - ANN starts with random weights and improves them
  - If improvement stops, we stop algorithm
  - No guarantee that we found the best set of weights
  - Could be stuck in a local minimum

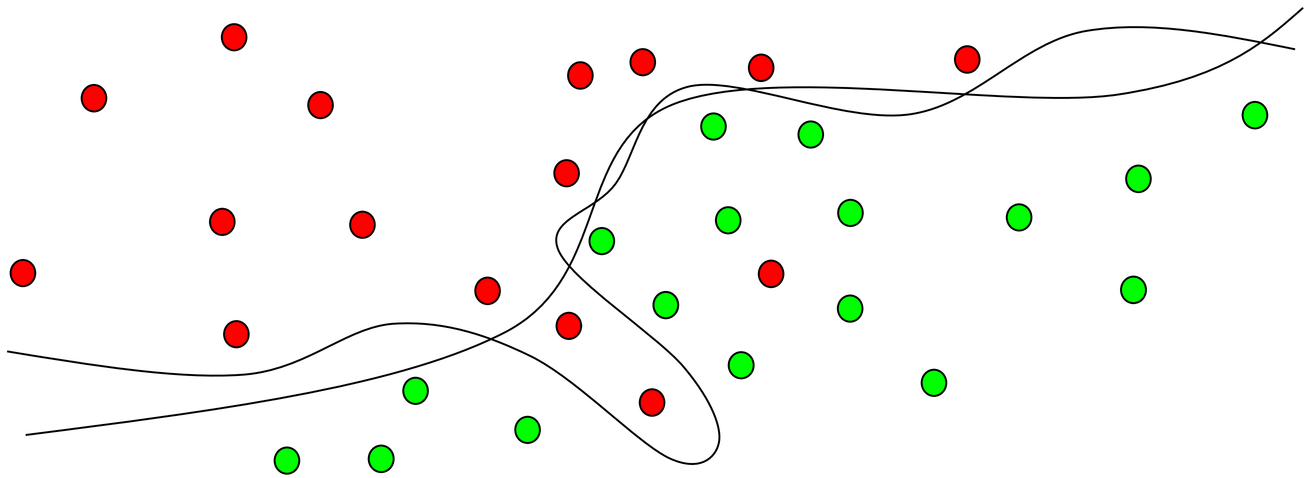
ANN Training

- Overtraining
  - An ANN can be made to work too well on a training set
  - But loose performance on test sets



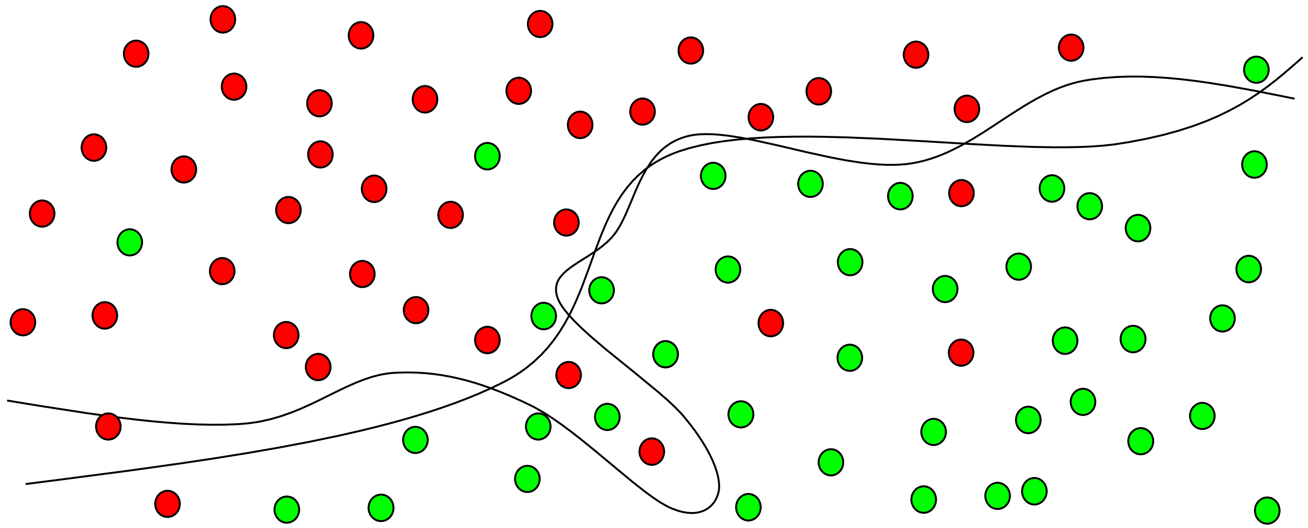
ANN Training

- Overtraining
  - Assume we want to separate the red from the green dots.
  - Eventually, the network will learn to do well in the training case
  - But have learnt only the particularities of our training set



ANN Training

- Overtraining



ANN Training

- Improving Convergence
  - Many Operations Research Tools apply
    - Simulated annealing
    - Sophisticated gradient descent

ANN Design

- ANN is a largely empirical study
  - “Seems to work in almost all cases that we know about”
- Known to be statistical pattern analysis

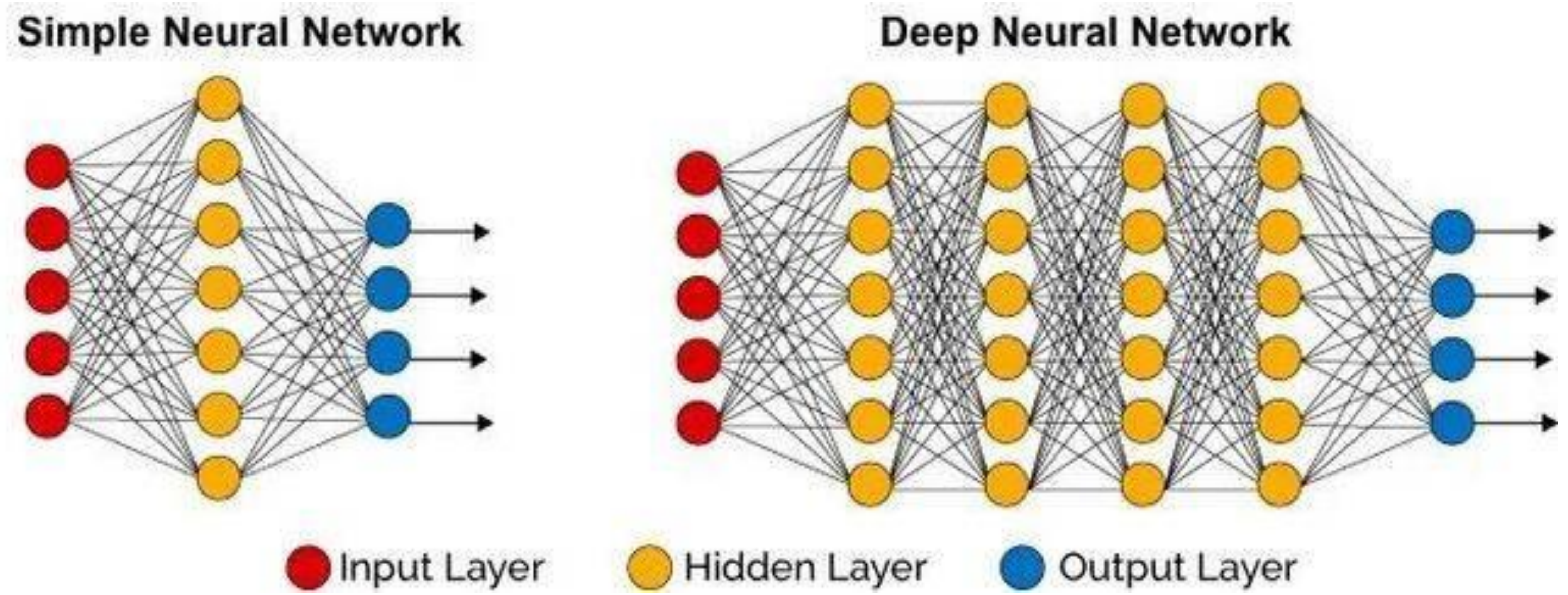
ANN Design

- Number of layers
  - Apparently, three layers is almost always good enough and better than four layers.
  - Also: fewer layers are faster in execution and training
- How many hidden nodes?
  - Many hidden nodes allow to learn more complicated patterns
  - Because of overtraining, almost always best to set the number of hidden nodes too low and then increase their numbers.

ANN Design

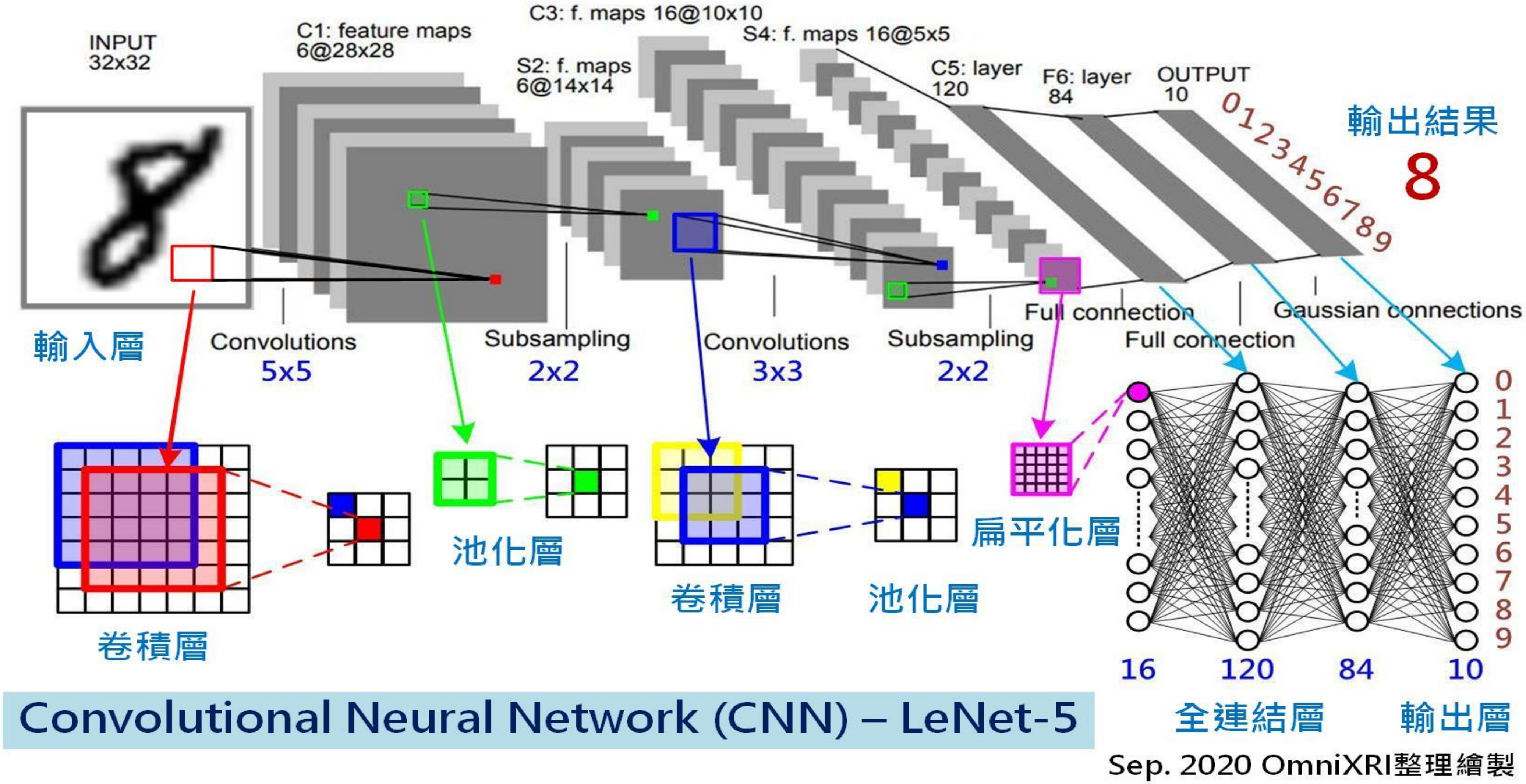
- Interpreting Output
  - ANN’s output neurons do not give binary values.
    - Good or bad
    - Need to define what is an accept.
  - Can indicate *n* degrees of certainty with *n*-1 output neurons.
    - Number of firing output neurons is degree of certainty

Deep Neural Network深度神經網路



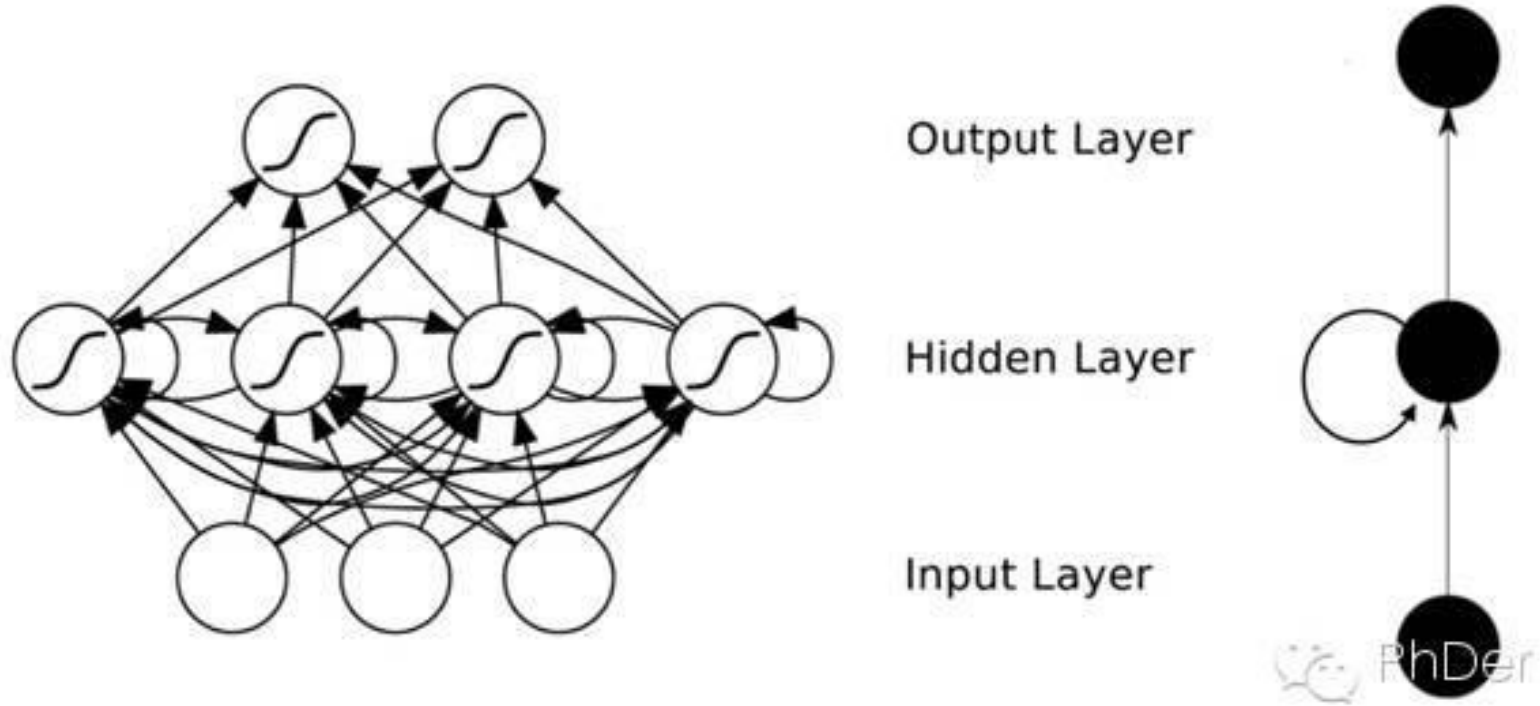
02.mp4

Convolutional Neural Network卷積神經網路



03.mp4

Recurrent NN 循環神經網絡



- connections between nodes can create a **cycle**, allowing output from some nodes to affect subsequent input to the same nodes

ART, BAM, BSB, Boltzman Machine, Cauchy Machine, Hopfield, RNN