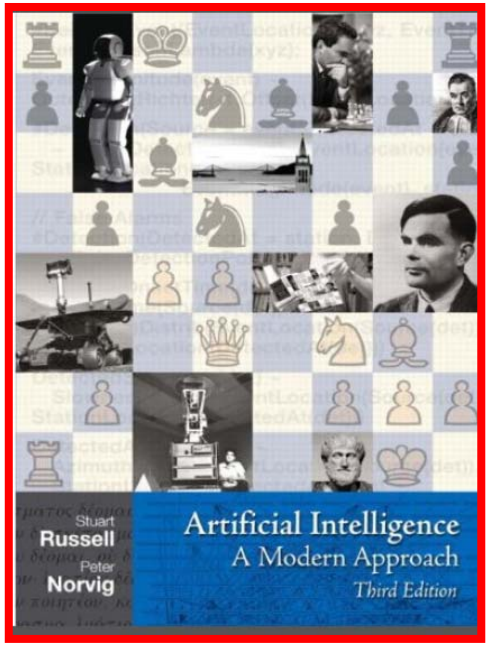


(Chapter-18)

ARTIFICIAL NEURAL NETWORK

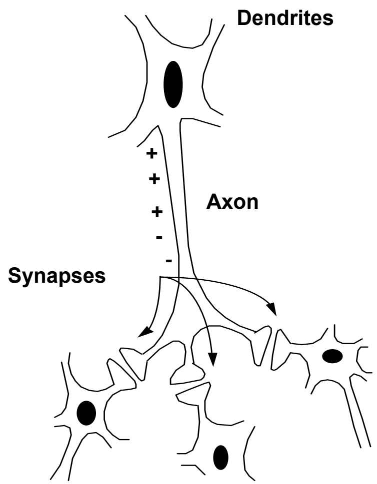
Yanmei Zheng

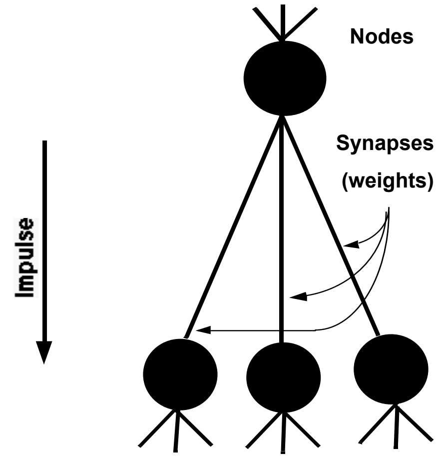
Outline

- History of the Neural Networks
- Neural Networks
- Backpropagation

Connectionist Models

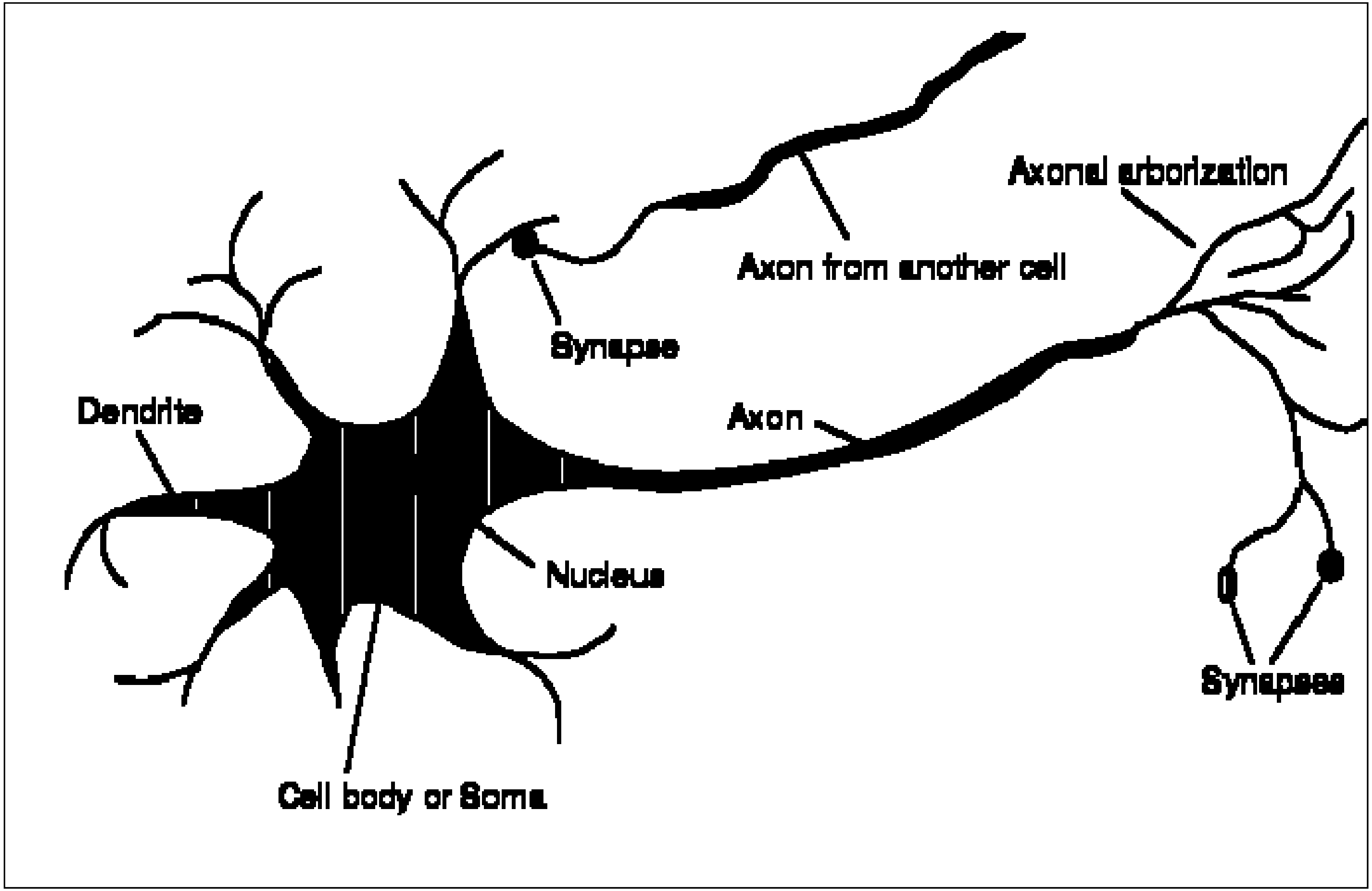
- **Consider humans:**
  - ✓ Neuron switching time ~ 0.001 second
  - ✓ Number of neurons ~ 10^{10}
  - ✓ Connections per neuron ~ 10^{4-5}
  - ✓ Scene recognition time ~ 0.1 second
  - ✓ 100 inference steps doesn't seem like enough → much parallel computation





- **Properties of artificial neural nets (ANN)**
  - ✓ Many neuron-like threshold switching units
  - ✓ Many weighted interconnections among units
  - ✓ Highly parallel, distributed processes

Neural Networks



Neural Networks

- McCulloch & Pitts (1943) are generally recognised as the designers of the first neural network

- Many of their ideas still used today (e.g. many simple units combine to give increased computational power and the idea of a threshold)

Neural Networks

- Hebb (1949) developed the first learning rule (on the premise that if two neurons were active at the same time the strength between them should be increased)

Neural Networks

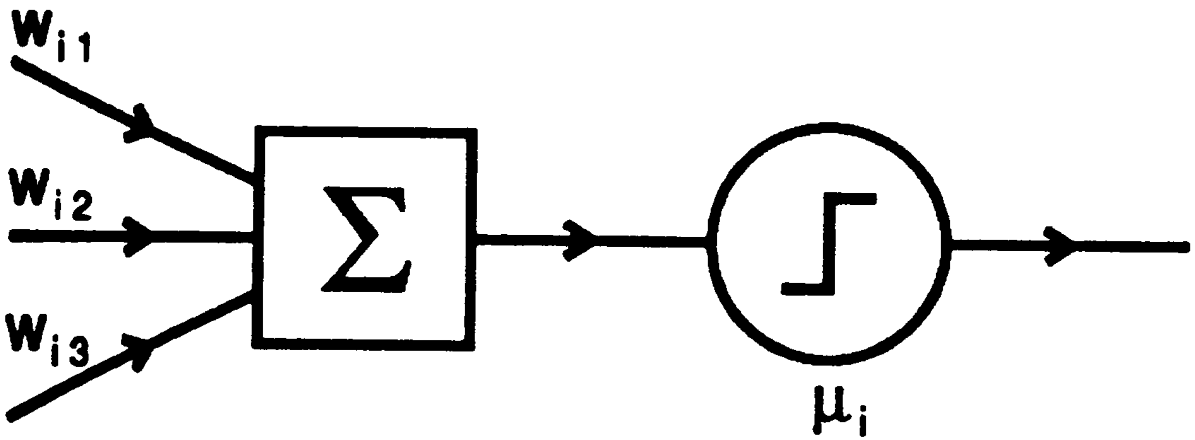
- During the 50's and 60's many researchers worked on the perceptron amidst great excitement.
- 1969 saw the death of neural network research for about 15 years – Minsky & Papert
- Only in the mid 80's (Parker and LeCun) was interest revived (in fact Werbos discovered algorithm in 1974)

History

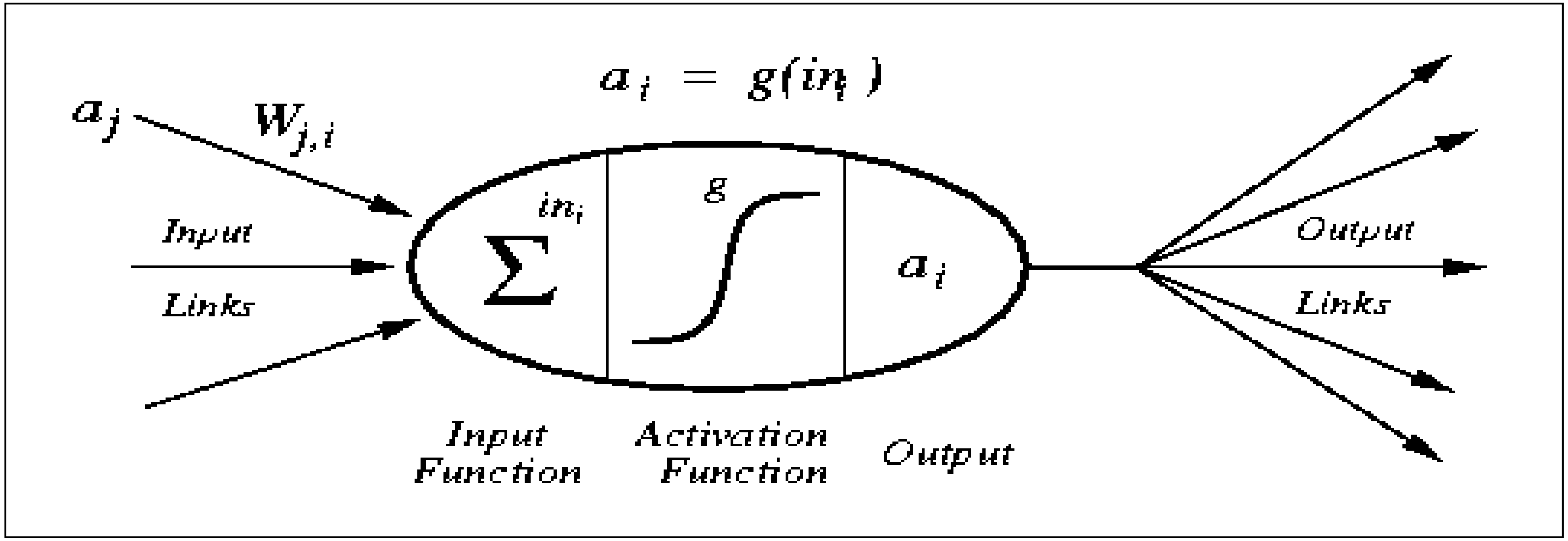
- 1943: McCulloch–Pitts “neuron”
  1. Started the field
- 1962: Rosenblatt’s perceptron
  1. Learned its own weight values; convergence proof
- 1969: Minsky & Papert book on perceptrons
  1. Proved limitations of single-layer perceptron networks
- 1982: Hopfield and convergence in symmetric networks
  1. Introduced energy-function concept
- 1986: Backpropagation of errors
  1. Method for training multilayer networks
- Present: Probabilistic interpretations, Bayesian and spiking networks

McCulloch–Pitts “neuron” (1943)

- **Attributes of neuron**
  1. m binary inputs and 1 output (0 or 1)
  2. Synaptic weights w_{ij}
  3. Threshold μ_i



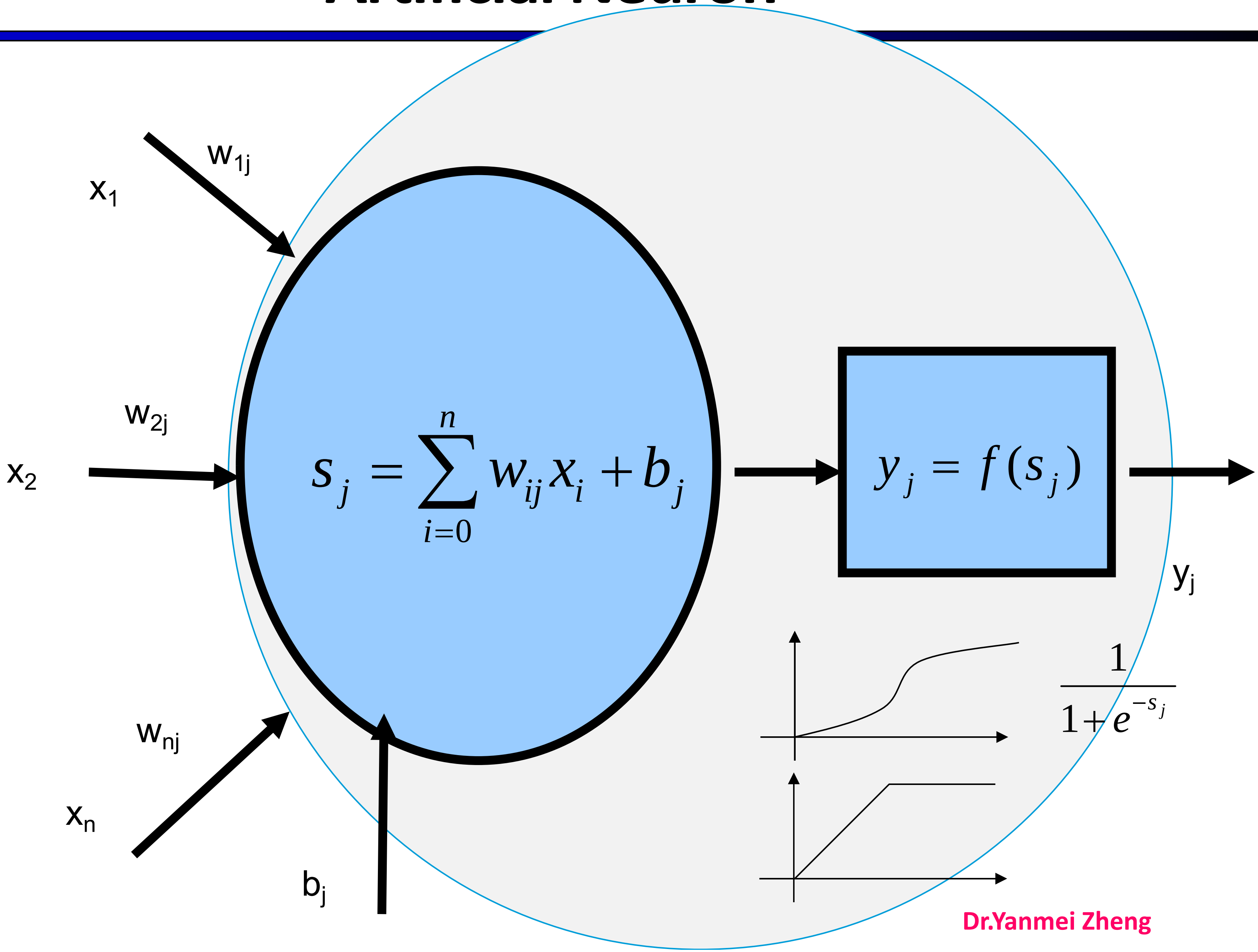
Modelling a Neuron



$$in_i = \sum_j W_{j,i} a_j$$

- a_j :Activation value of unit j
- $w_{j,I}$:Weight on the link from unit j to unit i
- in_I :Weighted sum of inputs to unit i
- a_I :Activation value of unit i
- g :Activation function

Artificial Neuron

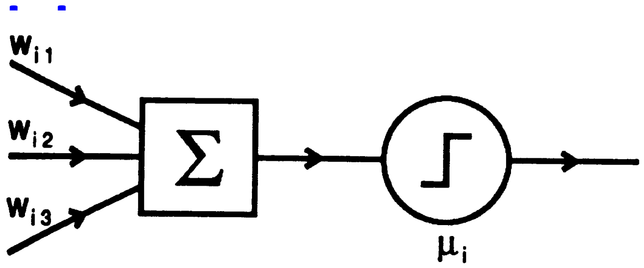


Dr.Yanmei Zheng

McCulloch–Pitts Neural Networks

- **Synchronous discrete time operation**
  1. **Time quantized in units of synaptic**

$$n_i(t+1) = \Theta\left[\sum_j w_{ij} n_j(t) - \mu_i\right]$$



- **Output is 1 if and only if weighted sum of inputs is greater than threshold**
  $\Theta(x) = 1$ if $x \geq 0$ and 0 if $x < 0$

$n_i \equiv$ output of unit i
$\Theta \equiv$ step function
$w_{ij} =$ weight from unit j to i
$\mu_i =$ threshold

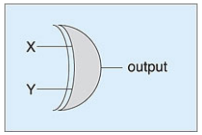
- **Behavior of network can be simulated by a finite automaton**
- **Any FA can be simulated by a McCulloch-Pitts Network**

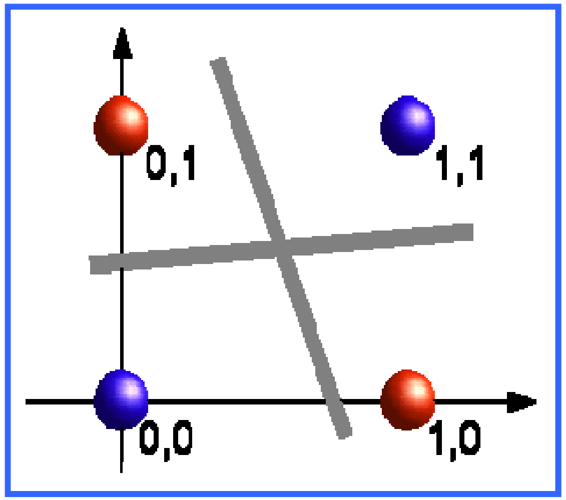
Learning highly non-linear functions

f: X → Y

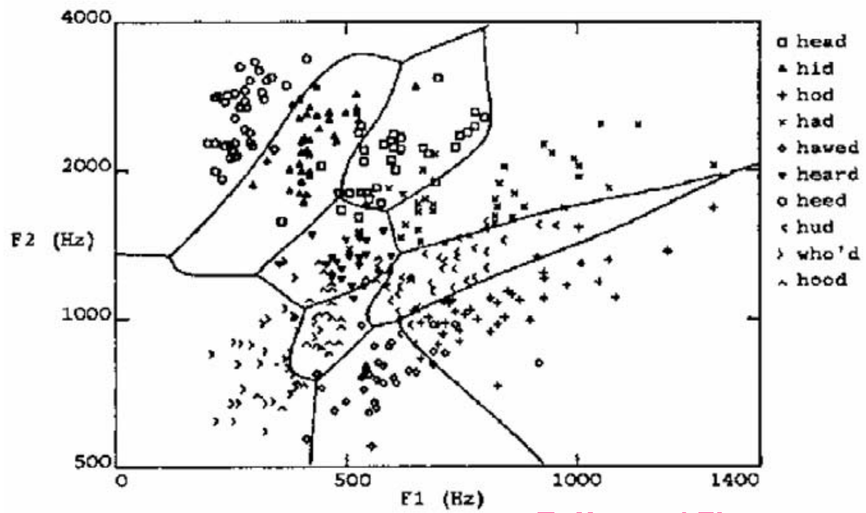
- f might be non-linear function
- X (vector of) continuous and/or discrete vars
- Y (vector of) continuous and/or discrete vars

The XOR gate





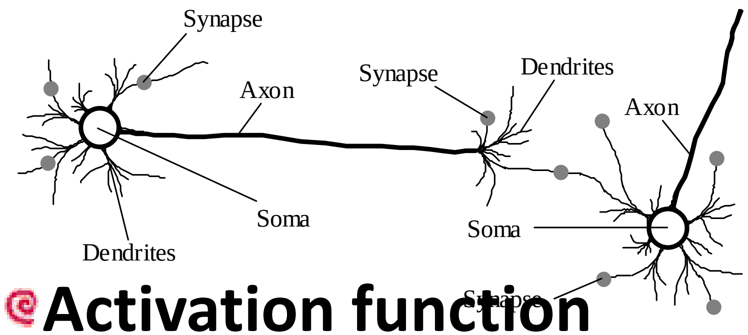
Speech recognition

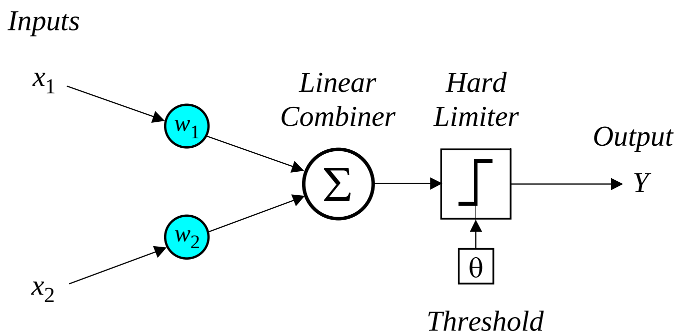


Dr.Yanmei Zheng

Perceptron and Neural Nets

From biological neuron to artificial neuron (perceptron)

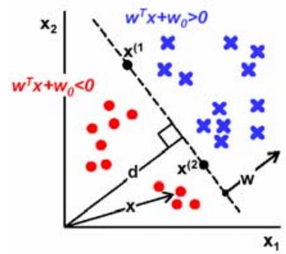




Activation function

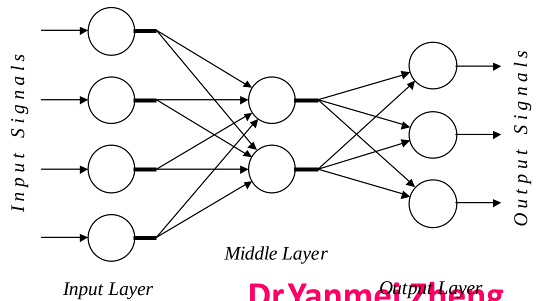
$$X = \sum_{i=1}^{n} x_i w_i$$

$$Y = \begin{cases} +1, & \text{if } X \geq \omega_0 \\ -1, & \text{if } X < \omega_0 \end{cases}$$



Artificial neuron networks

- ✓ supervised learning
- ✓ gradient descent

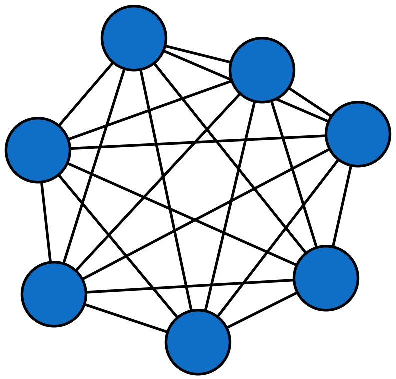


Dr.Yanmei Zheng

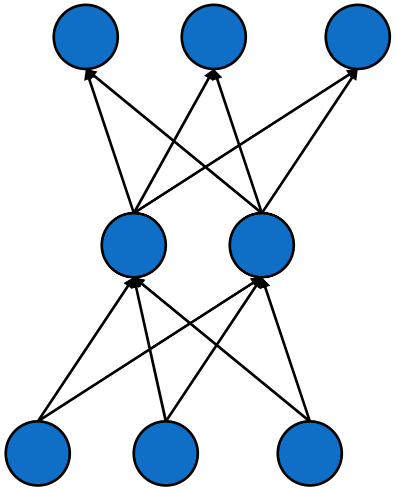
Properties of Artificial Neural Networks

- **High level abstraction of neural input-output transformation**
  1. **Inputs → weighted sum of inputs → nonlinear function → output**
     - **Typically no spikes**
     - **Typically use implausible constraints or learning rules**
- **Often used where data or functions are uncertain**
  1. **Goal is to learn from a set of training data**
  2. **And to generalize from learned instances to new unseen data**
- **Key attributes**
  1. **Parallel computation**
  2. **Distributed representation and storage of data**
  3. **Learning (networks adapt themselves to solve a problem)**
  4. **Fault tolerance (insensitive to component failures)**

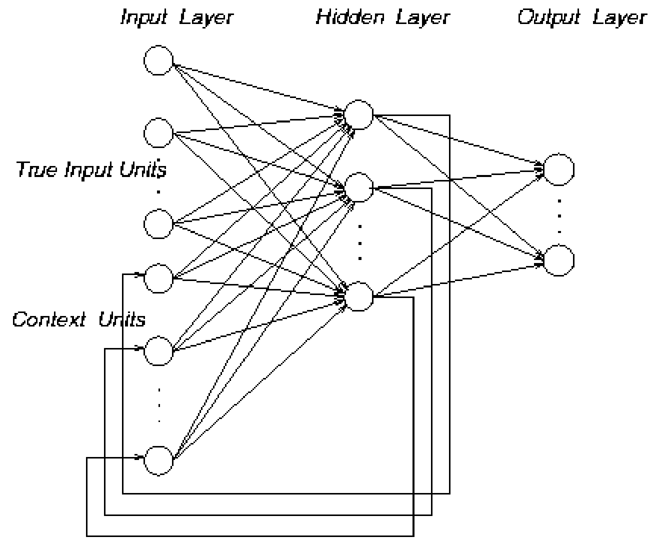
Topologies of Neural Networks



completely connected



feedforward (directed, a-cyclic)



recurrent (feedback connections)

Networks Types

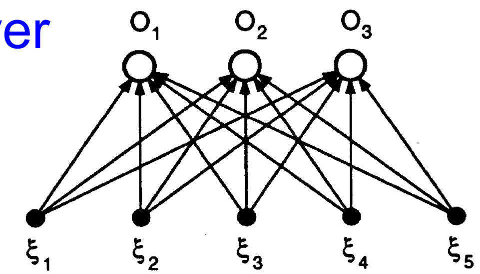
- **Feedforward versus recurrent networks**
  1. **Feedforward: No loops, input → hidden layers → output**
  2. **Recurrent: Use feedback (positive or negative)**
- **Continuous versus spiking**
  1. **Continuous networks model mean spike rate (firing rate)**
     - **Assume spikes are integrated over time**
  2. **Consistent with rate-code model of neural coding**
- **Supervised versus unsupervised learning**
  1. **Supervised networks use a “teacher”**
     - **The desired output for each input is provided by user**
  2. **Unsupervised networks find hidden statistical patterns in input data**
     - **Clustering, principal component analysis**

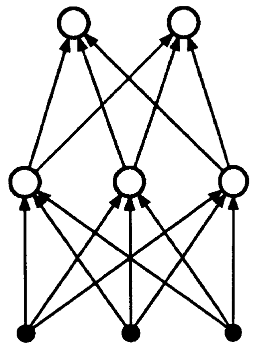
Perceptrons

- Attributes
  1. Layered feedforward networks
  2. Supervised learning
     - Hebbian: Adjust weights to enforce correlations
  3. Parameters: weights W_{ij}
  4. Binary output = Θ(weighted sum of inputs)
     - Take w_o to be the threshold with fixed input –1.

$$Output_i = \Theta\left[\sum_j w_{ij}\xi_j\right]$$

Single-layer





Multilayer

Training Perceptrons to Compute a Function

- **Given inputs ξ_j to neuron i and desired output Y_i, find its weight values by iterative improvement:**
  1. **Feed an input pattern**
  2. **Is the binary output correct?**

     ⇒**Yes: Go to the next pattern**

     ⇒**No: Modify the connection weights using error signal ($Y_i - O_i$)**

     ⇒**Increase weight if neuron didn't fire when it should have and vice versa**

$$= \eta (Y_i - O_i) \xi_j$$

$$= \eta (Y_i - w_{ij} \xi_j) \xi_j$$

$\eta \equiv$ learning rate

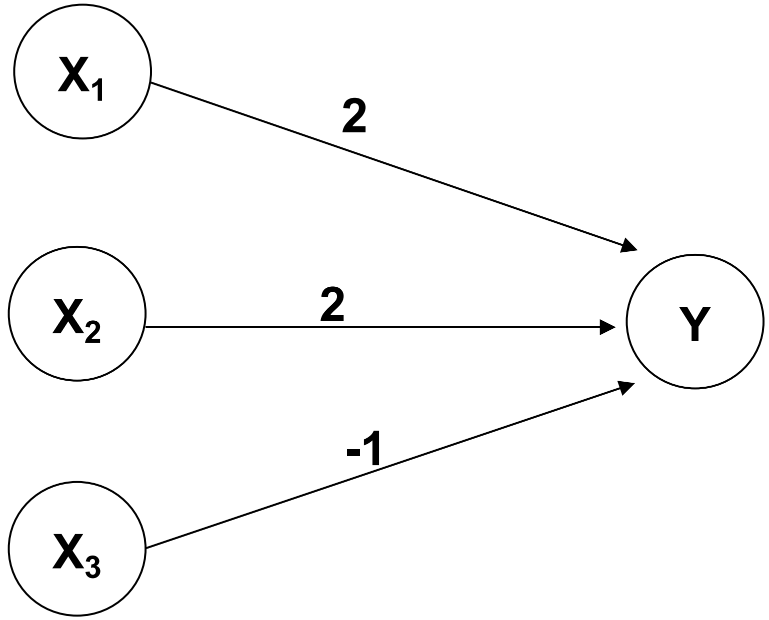
$\xi_j \equiv$ input

$Y_i \equiv$ desired output

$O_i \equiv$ actual output

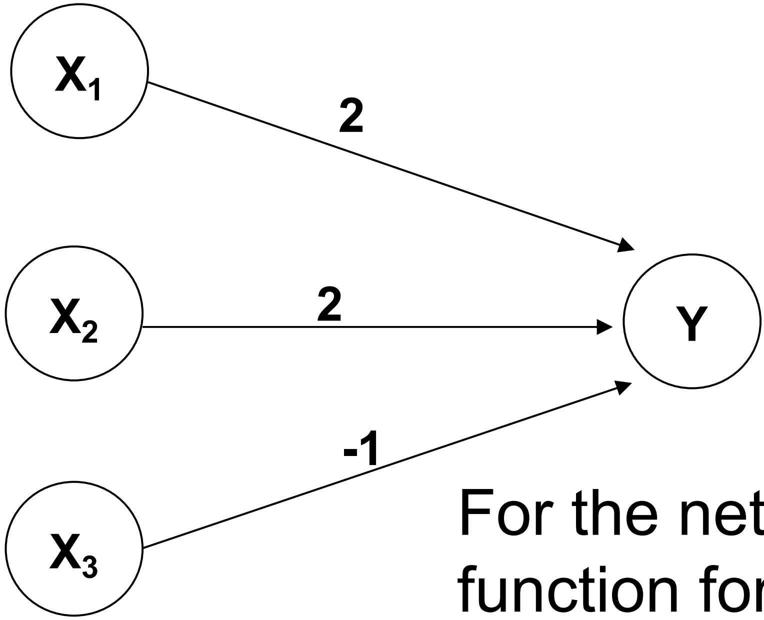
- **Learning rule is Hebbian (based on input/output correlation)**
  1. **converges in a finite number of steps if a solution exists**
  2. **Used in ADALINE (adaptive linear neuron) networks**

The First Neural Neural Networks



The activation of a neuron is binary. That is, the neuron either fires (activation of one) or does not fire (activation of zero).

The First Neural Neural Networks

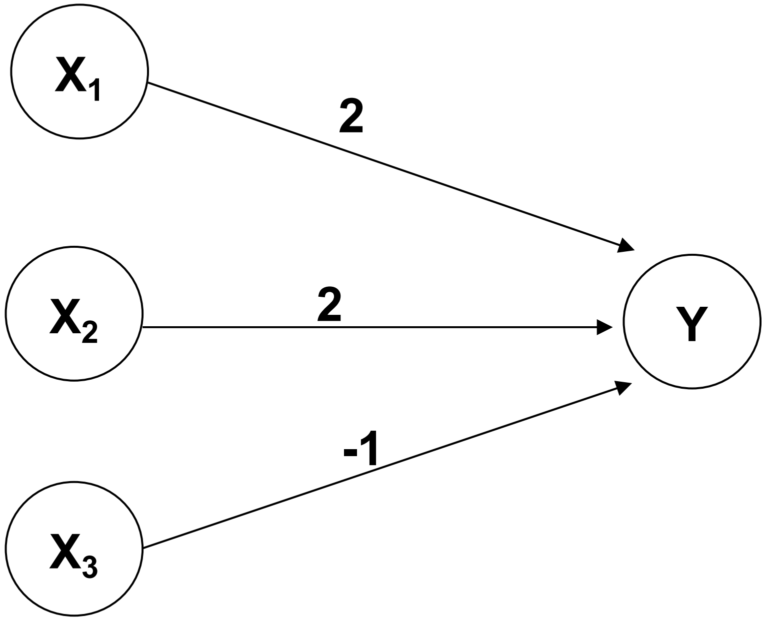


For the network shown here the activation function for unit *Y* is

$$f(y\_in) = 1, \text{ if } y\_in >= \theta \text{ else } 0$$

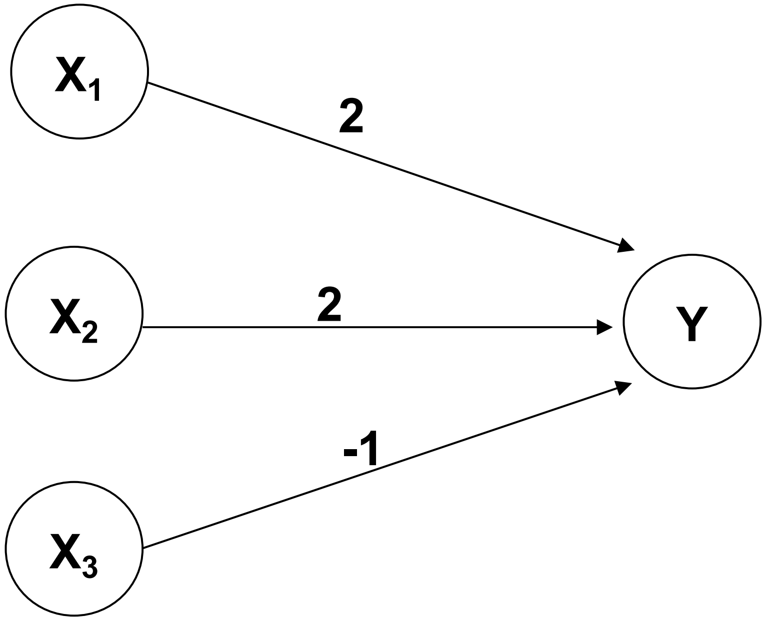
where y_in is the total input signal received
θ is the threshold for *Y*

The First Neural Neural Networks



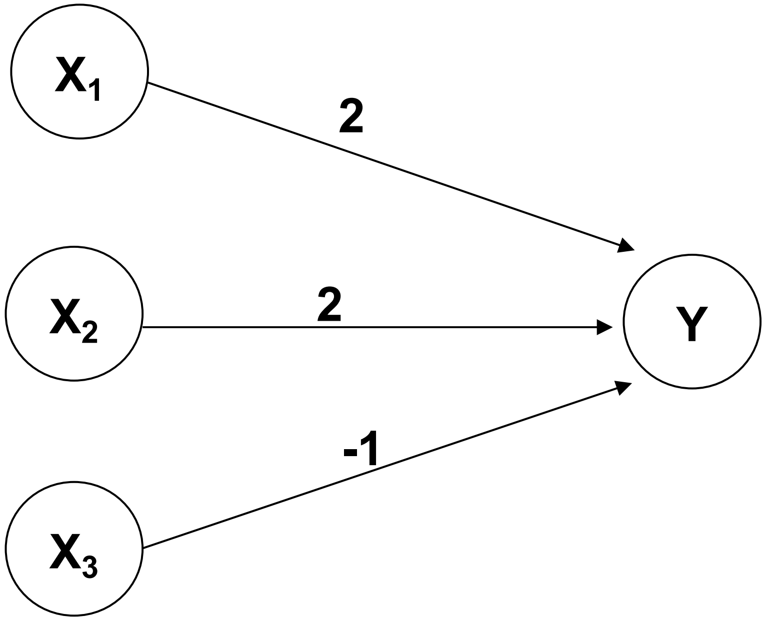
Neurons is a McCulloch-Pitts network are connected by directed, weighted paths

The First Neural Neural Networks



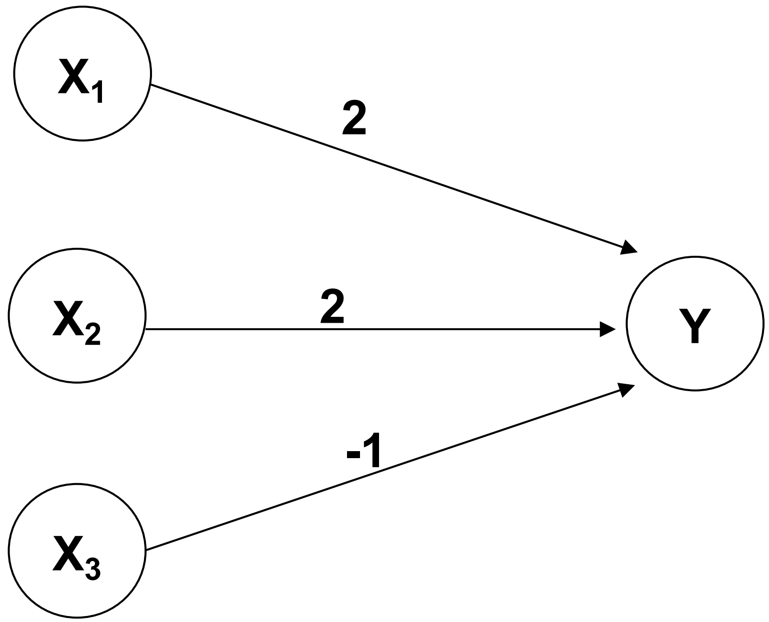
If the weight on a path is positive the path is excitatory, otherwise it is inhibitory

The First Neural Neural Networks



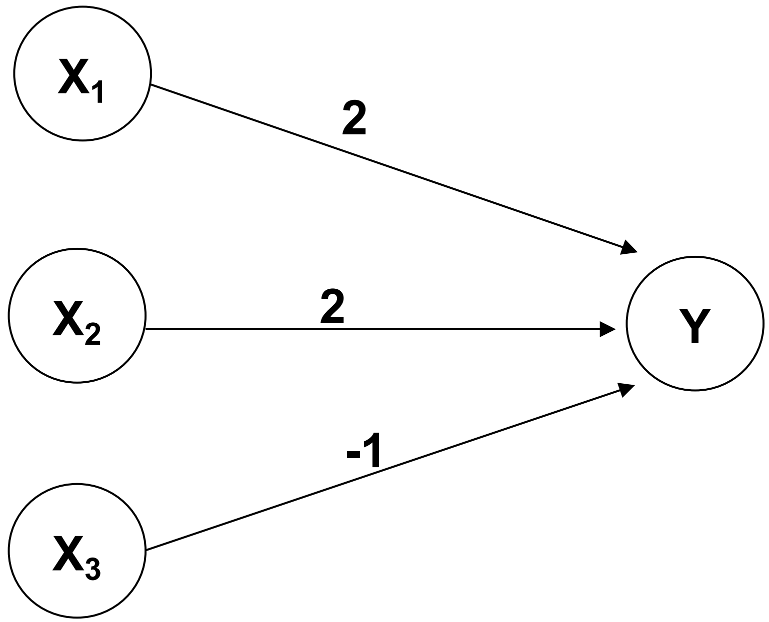
All excitatory connections into a particular neuron have the same weight, although different weighted connections can be input to different neurons

The First Neural Neural Networks



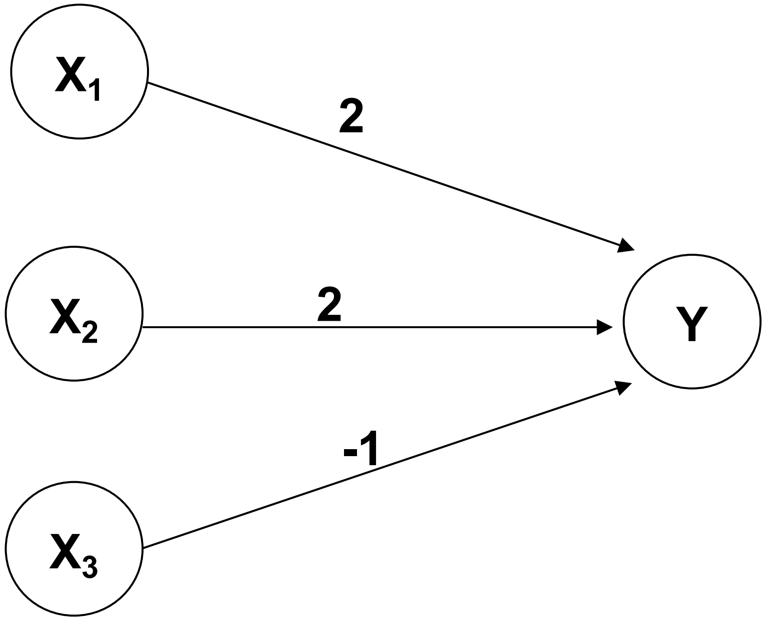
Each neuron has a fixed threshold. If the net input into the neuron is greater than the threshold, the neuron fires

The First Neural Neural Networks



The threshold is set such that any non-zero inhibitory input will prevent the neuron from firing

The First Neural Neural Networks

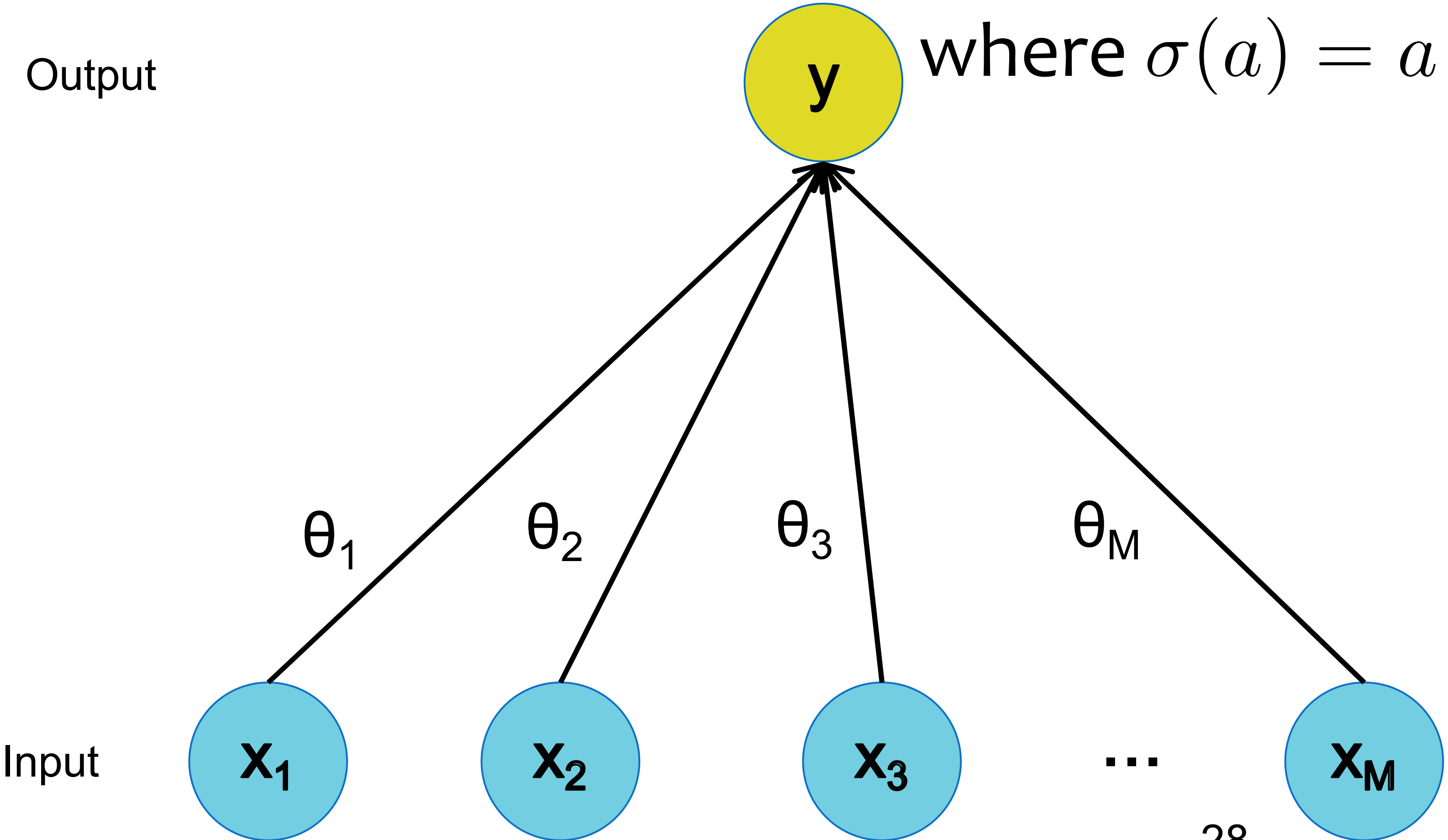


It takes one time step for a signal to pass over one connection.

Linear Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = a$

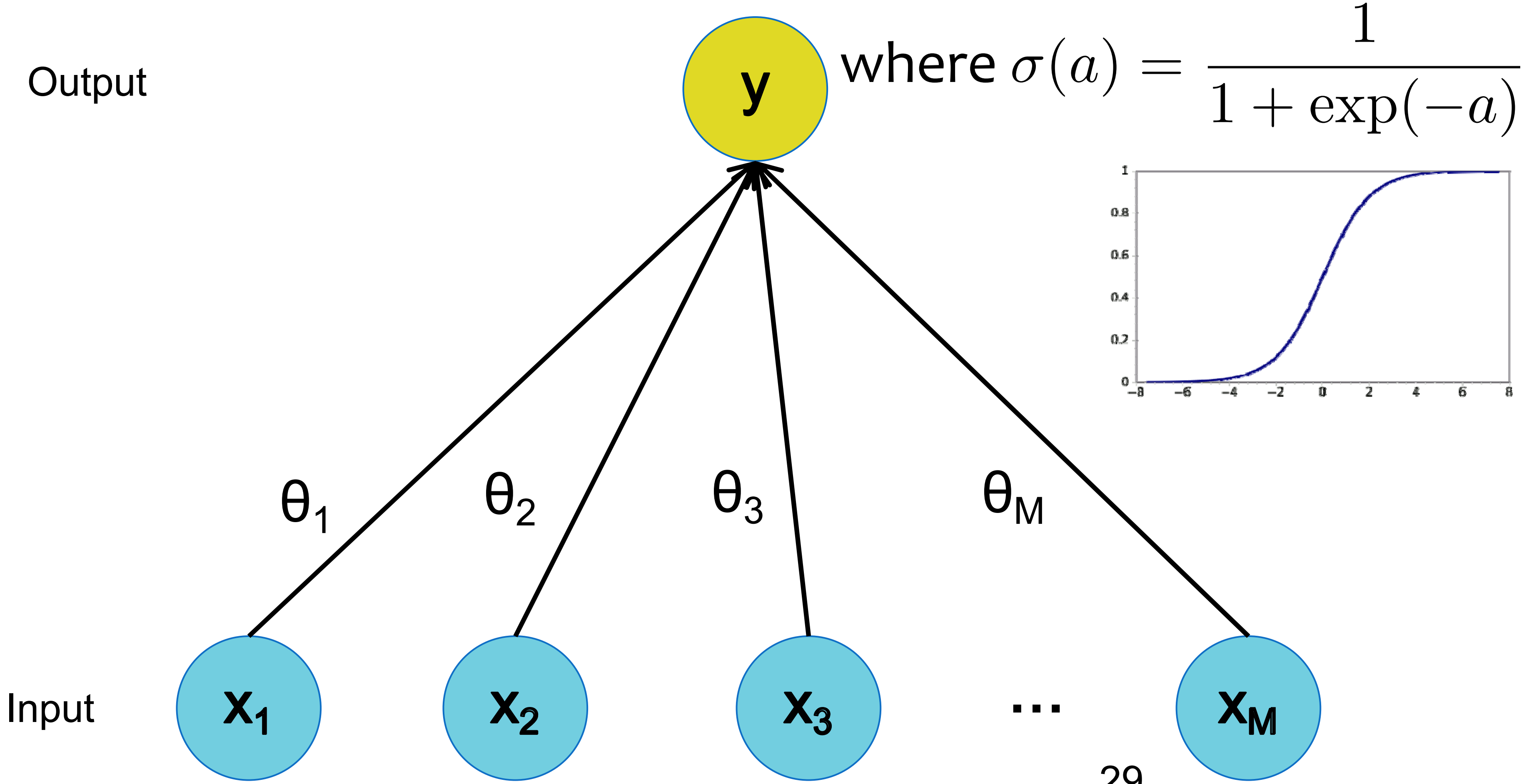


Dr.Yanmei Zheng

Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$



Dr.Yanmei Zheng

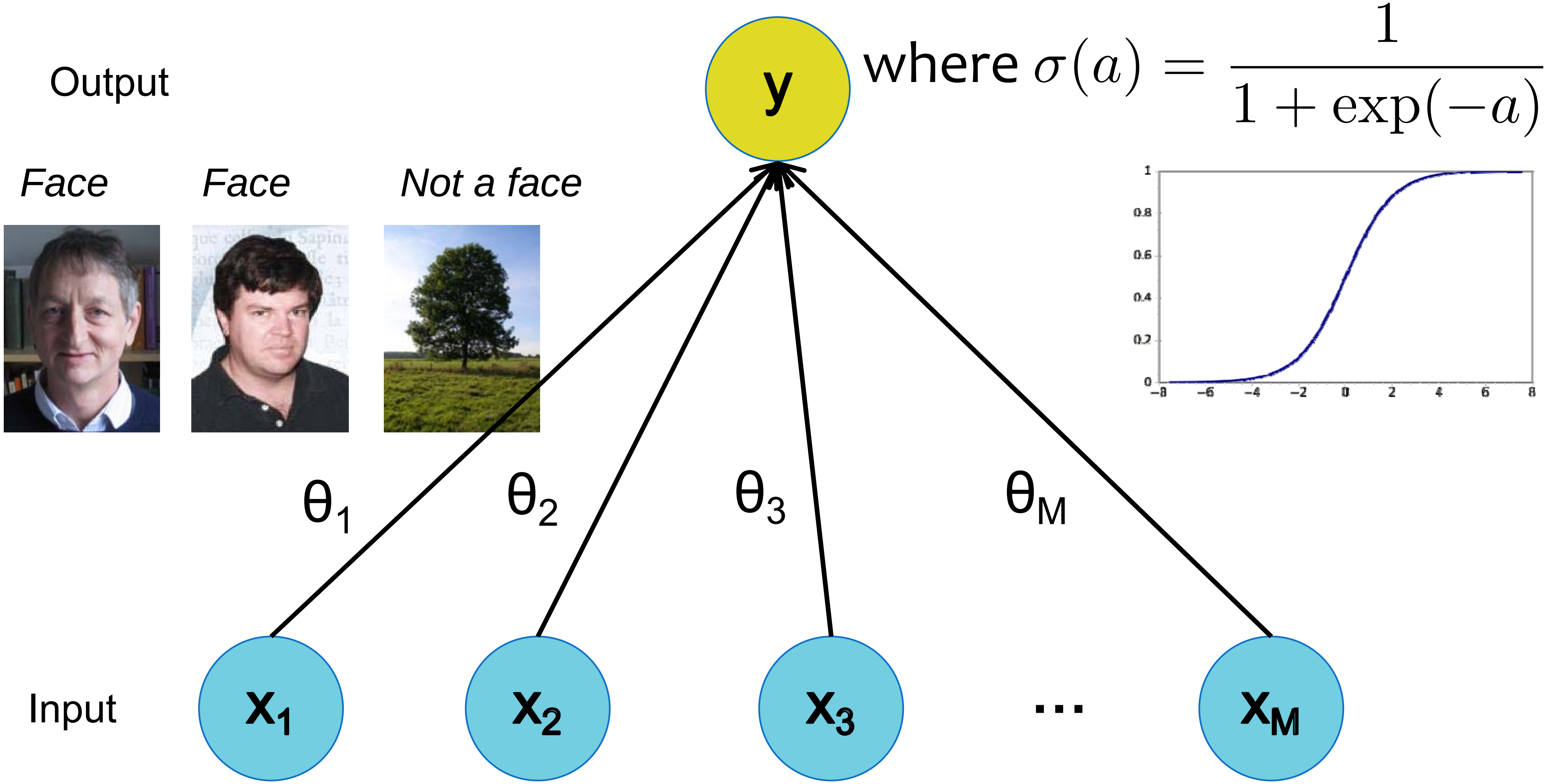
Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$

Output

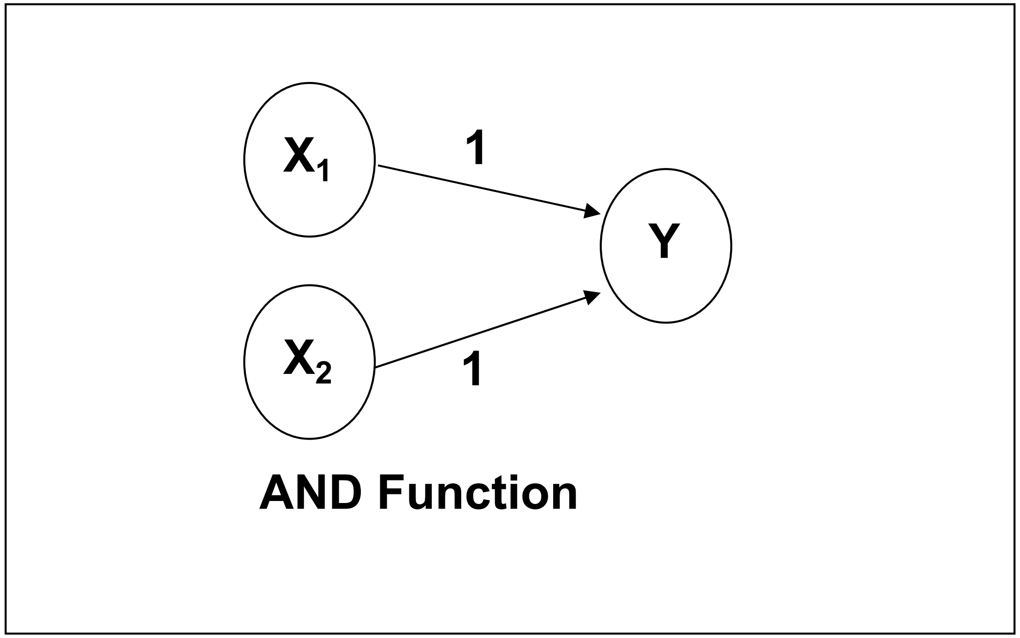
Face *Face* *Not a face*



Input

Dr.Yanmei Zheng

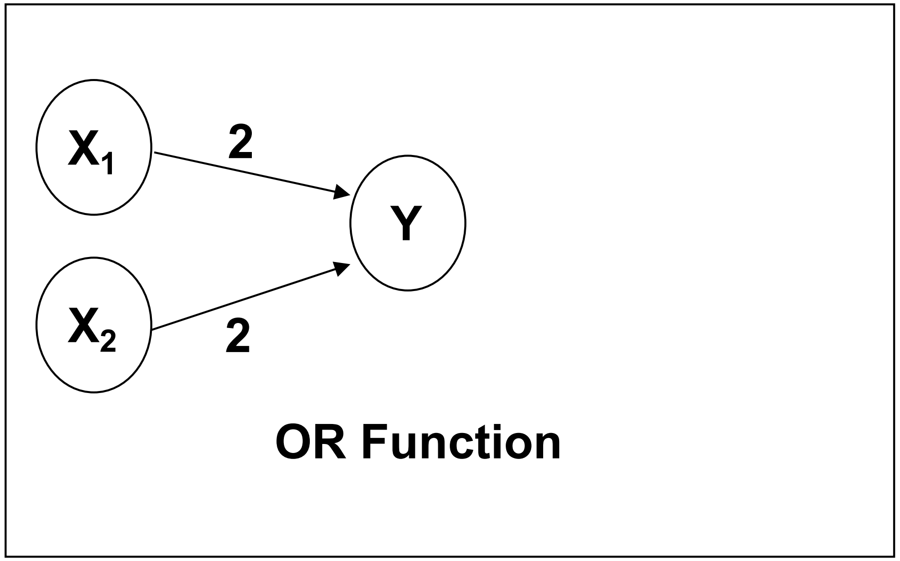
The First Neural Neural Networks



AND		
X1	X2	Y
1	1	1
1	0	0
0	1	0
0	0	0

Threshold(Y) = 2

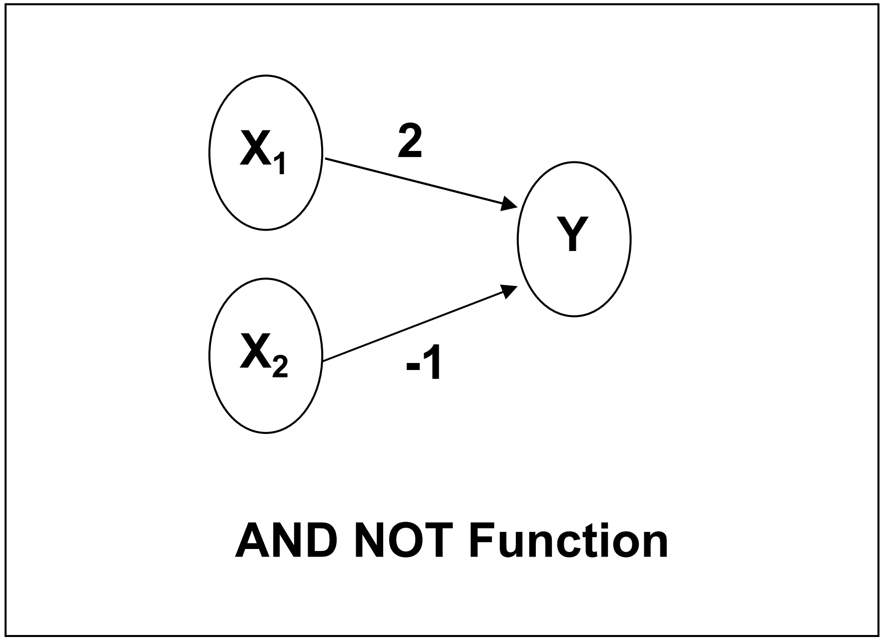
The First Neural Neural Networks



OR		
X1	X2	Y
1	1	1
1	0	1
0	1	1
0	0	0

Threshold(Y) = 2

The First Neural Neural Networks

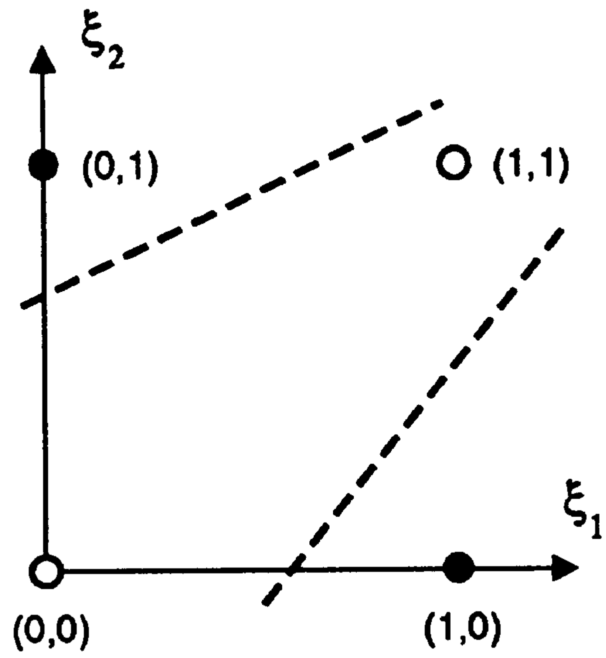


Threshold(Y) = 2

AND NOT		
X1	X2	Y
1	1	0
1	0	1
0	1	0
0	0	0

Linear inseparability

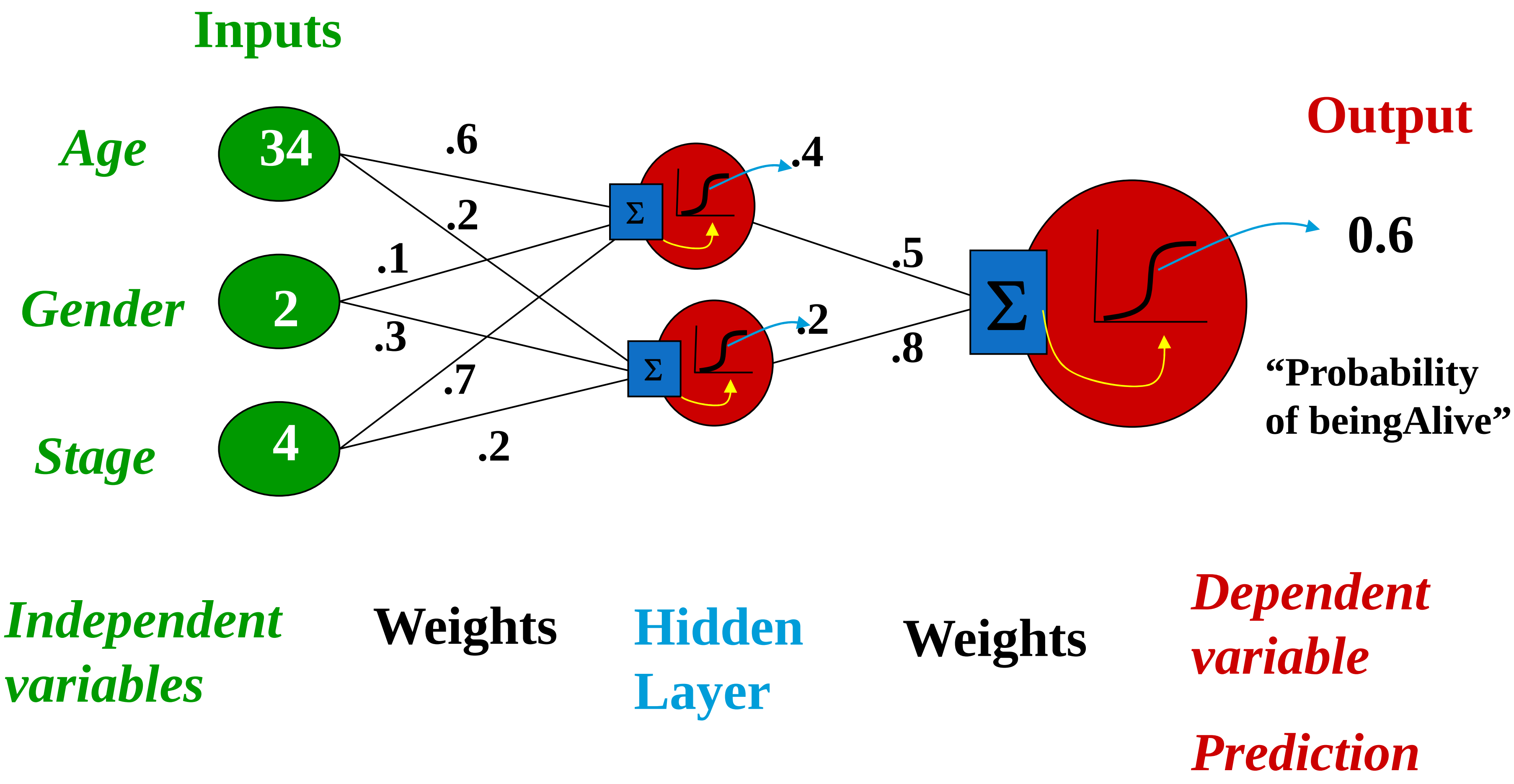
- Single-layer perceptron with threshold units fails if problem is not linearly separable
  1. Example: XOR



Multilayer perceptrons

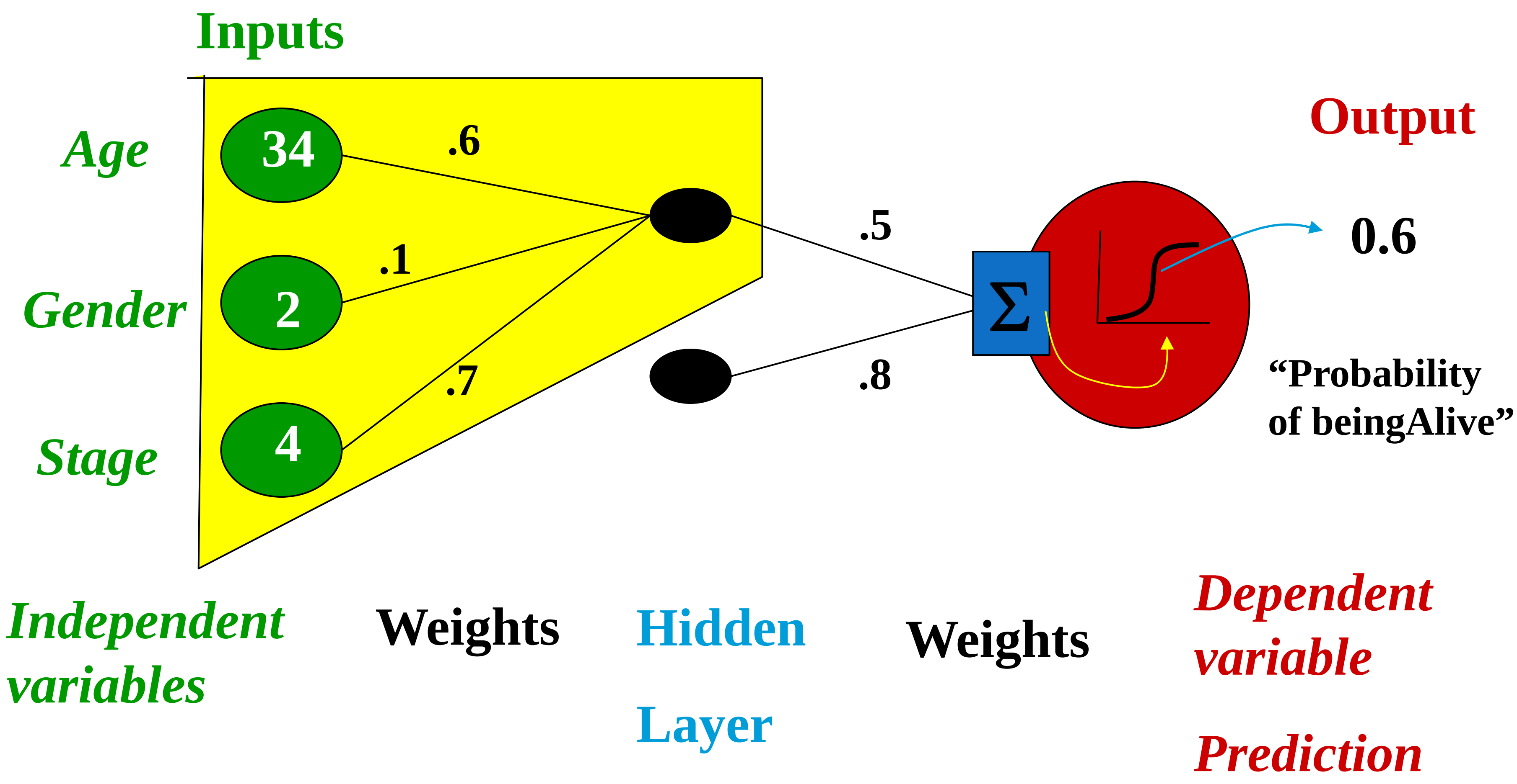
Dr.Yanmei Zheng

Neural Network Model



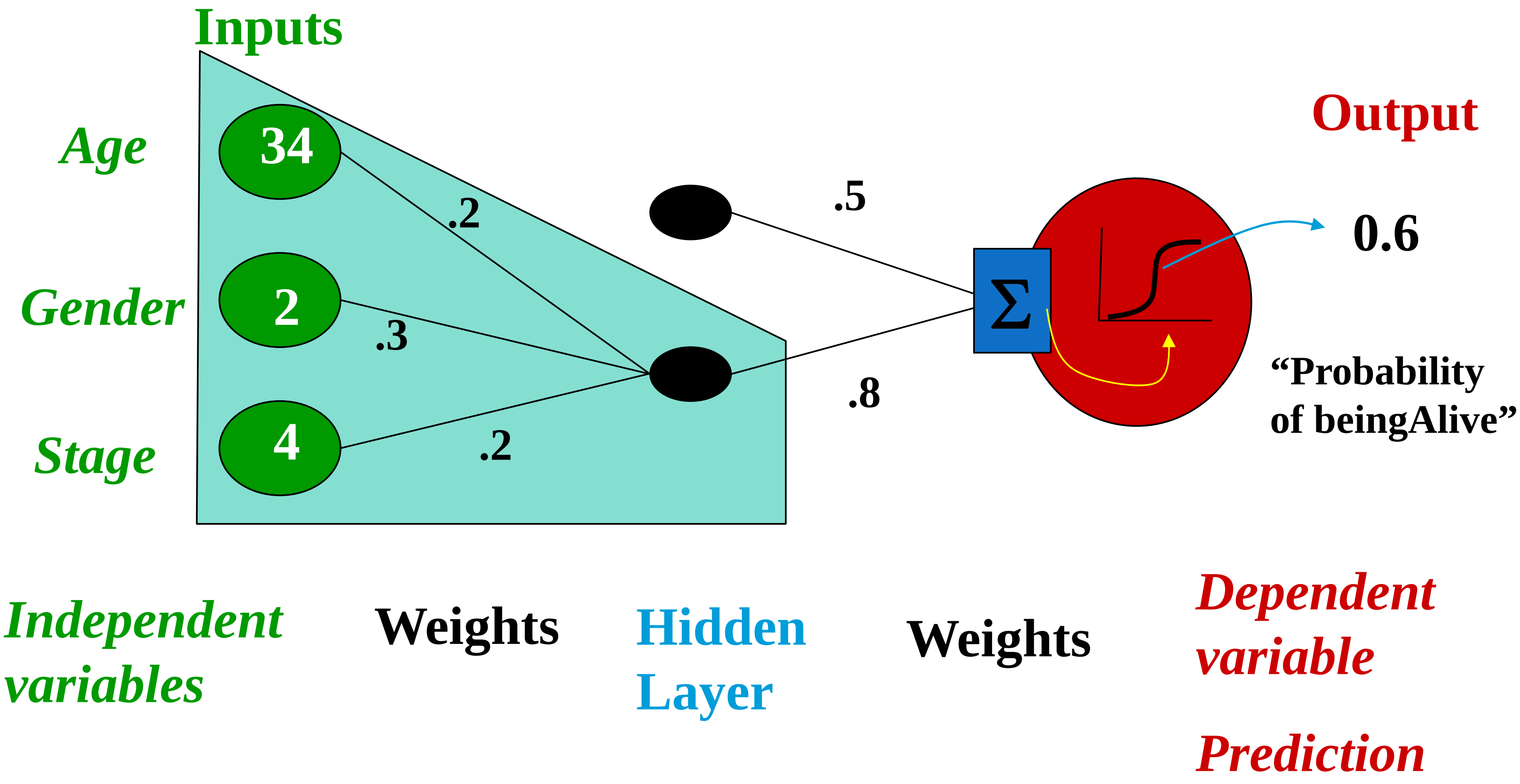
Dr.Yanmei Zheng

"Combined logistic models"

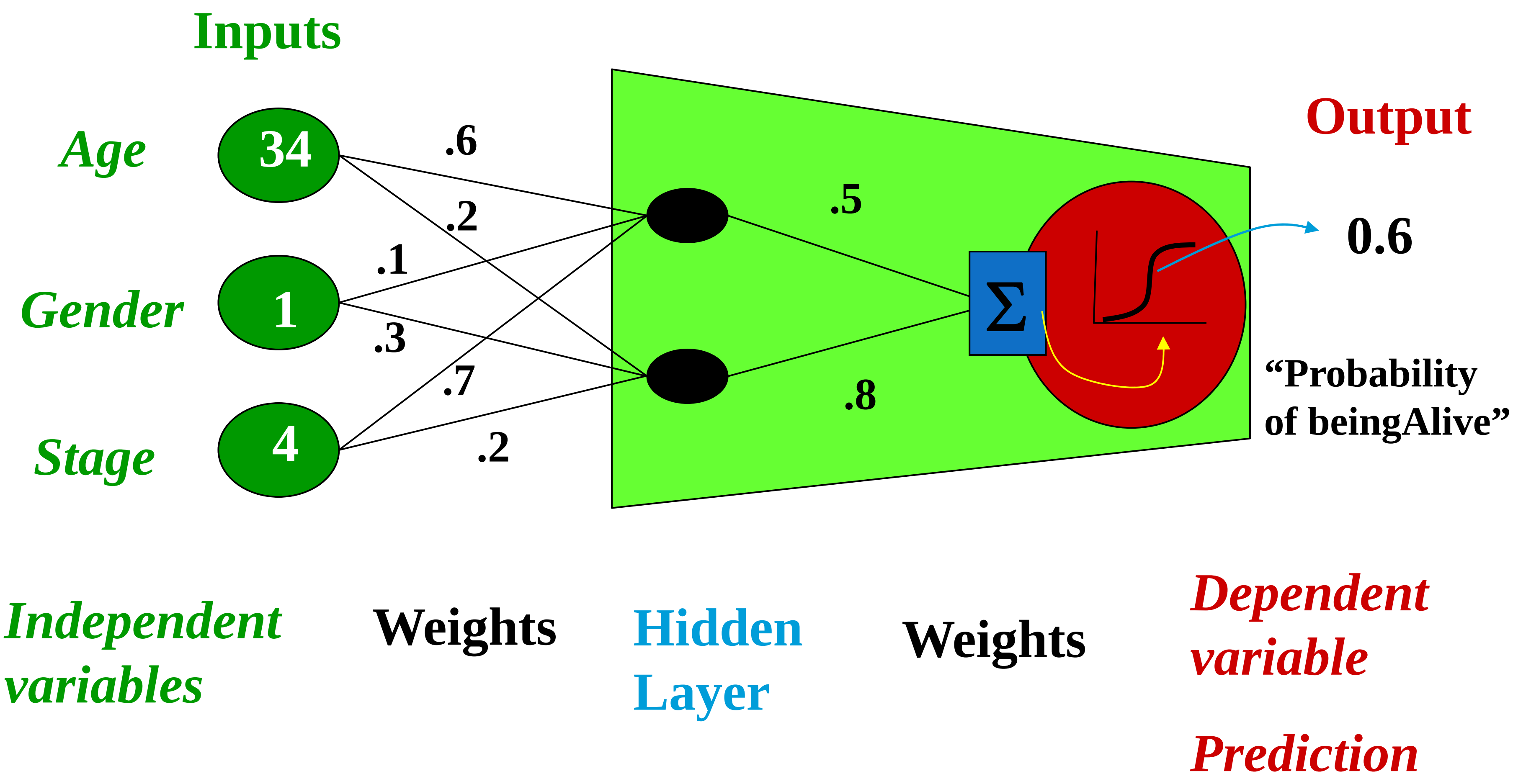


Dr.Yanmei Zheng

"Combined logistic models"

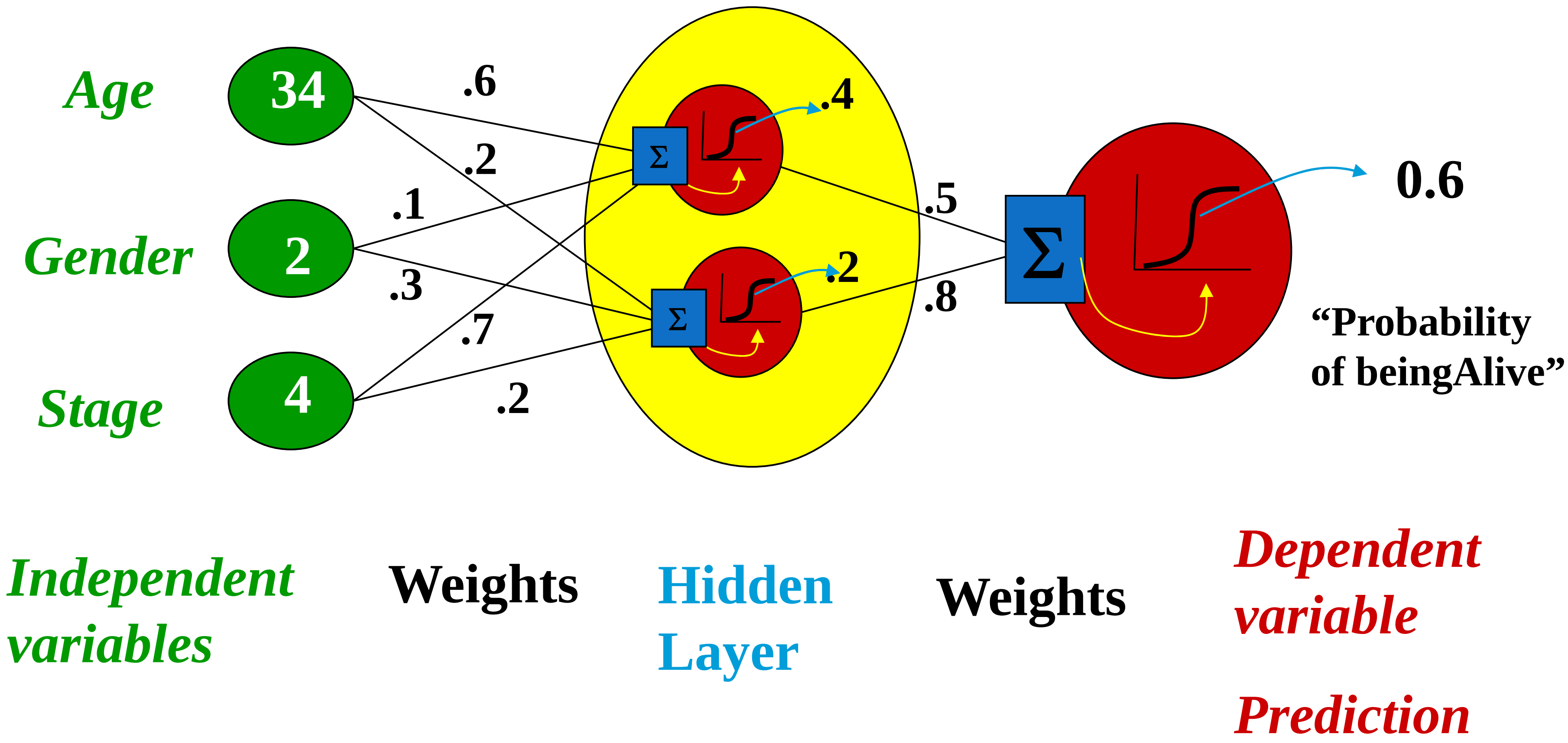


“Combined logistic models”



Dr.Yanmei Zheng

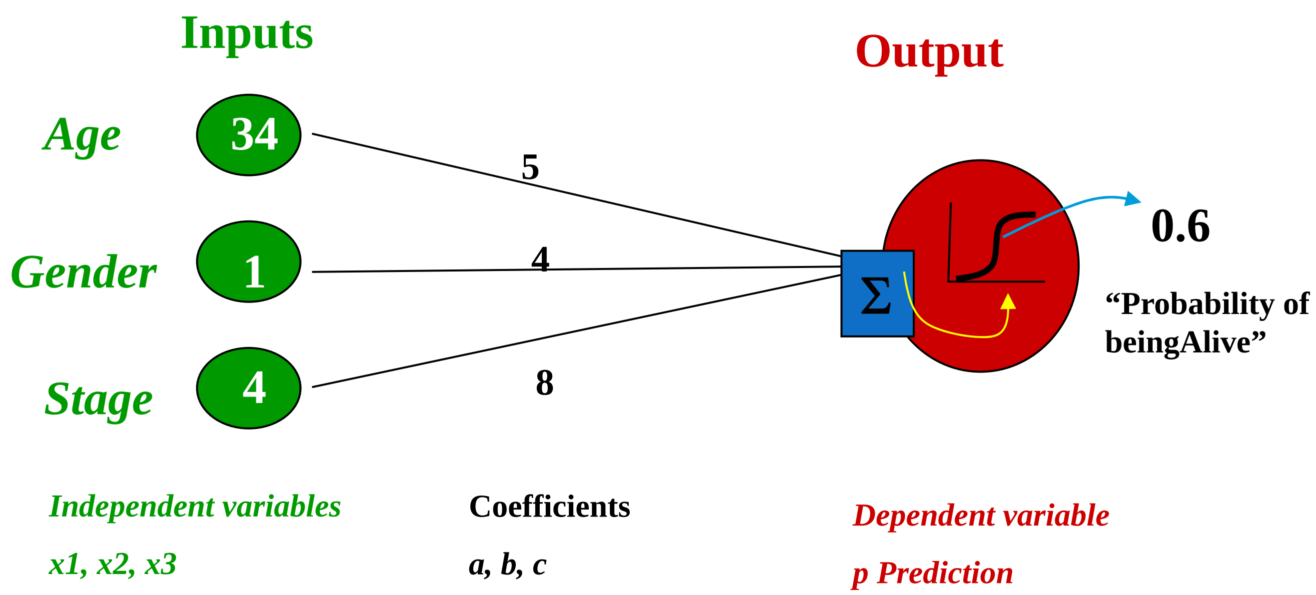
Not really, no target for hidden units...



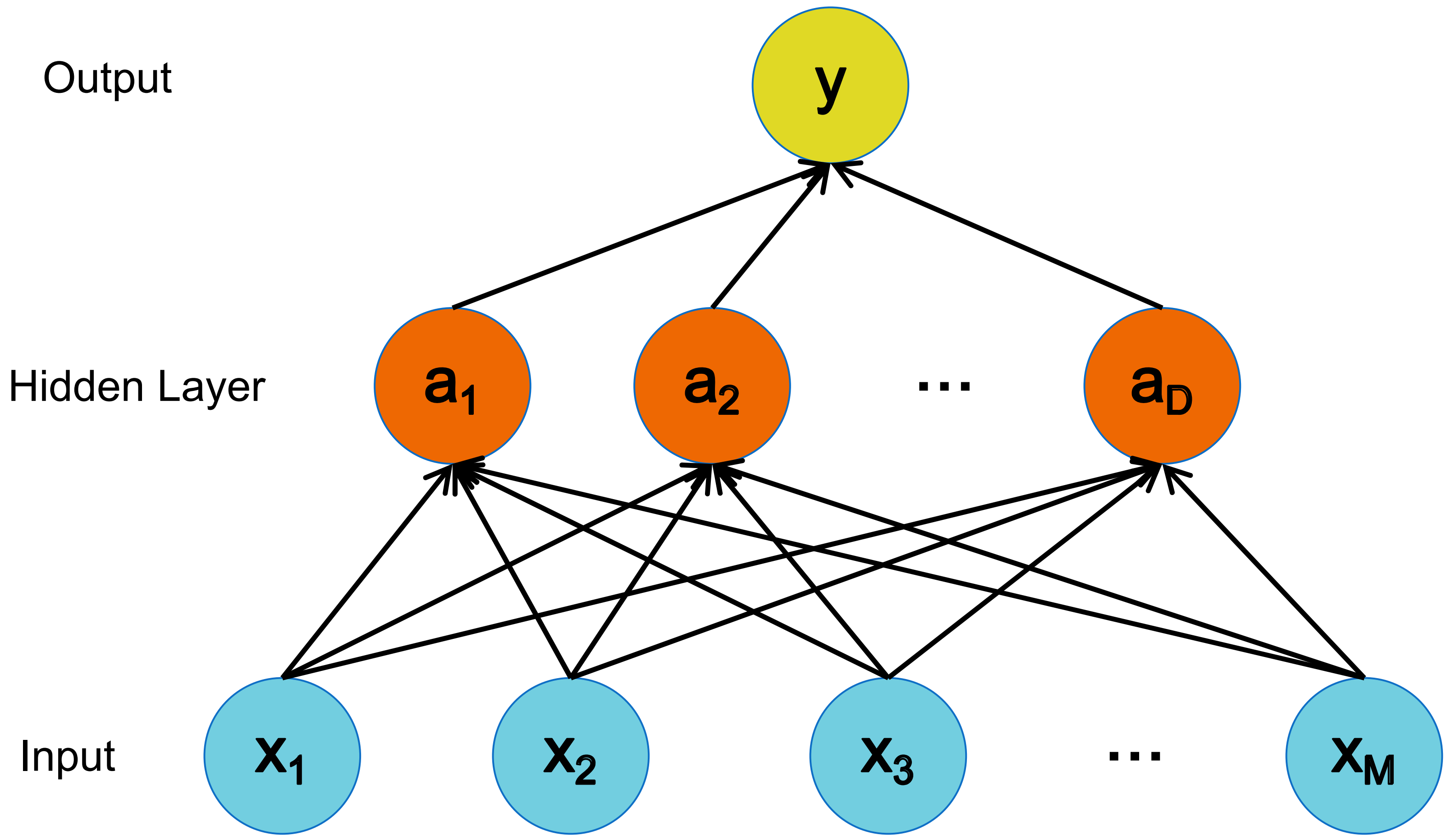
Jargon Pseudo-Correspondence

- Independent variable = input variable
- Dependent variable = output variable
- Coefficients = “weights”
- Estimates = “targets”

Logistic Regression Model (the sigmoid unit)

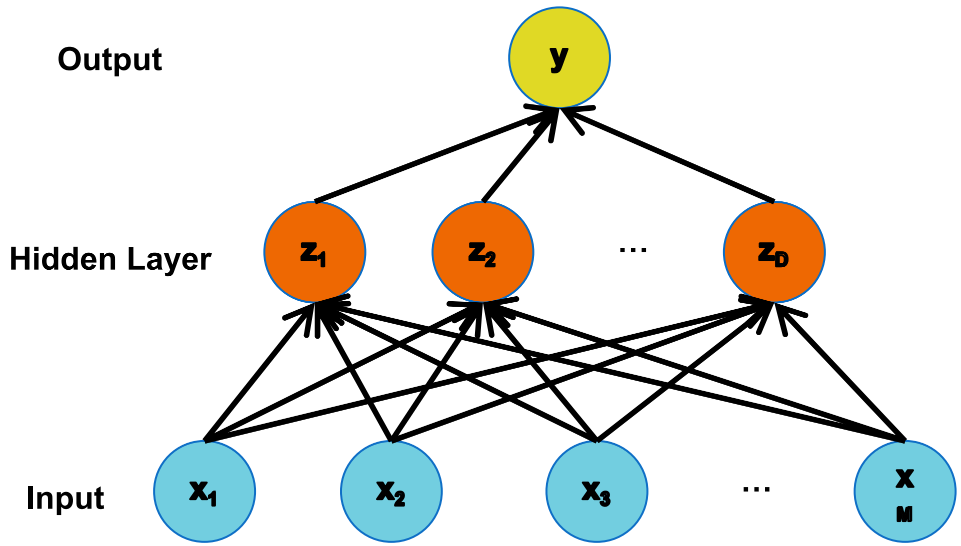


Neural Network



Dr.Yanmei Zheng

Neural Network



(E) Output (sigmoid)

$$y = \frac{1}{1+\exp(-b)}$$

(D) Output (linear)

$$b = \sum_{j=0}^{D} \beta_j z_j$$

(C) Hidden (sigmoid)

$$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$$

(B) Hidden (linear)

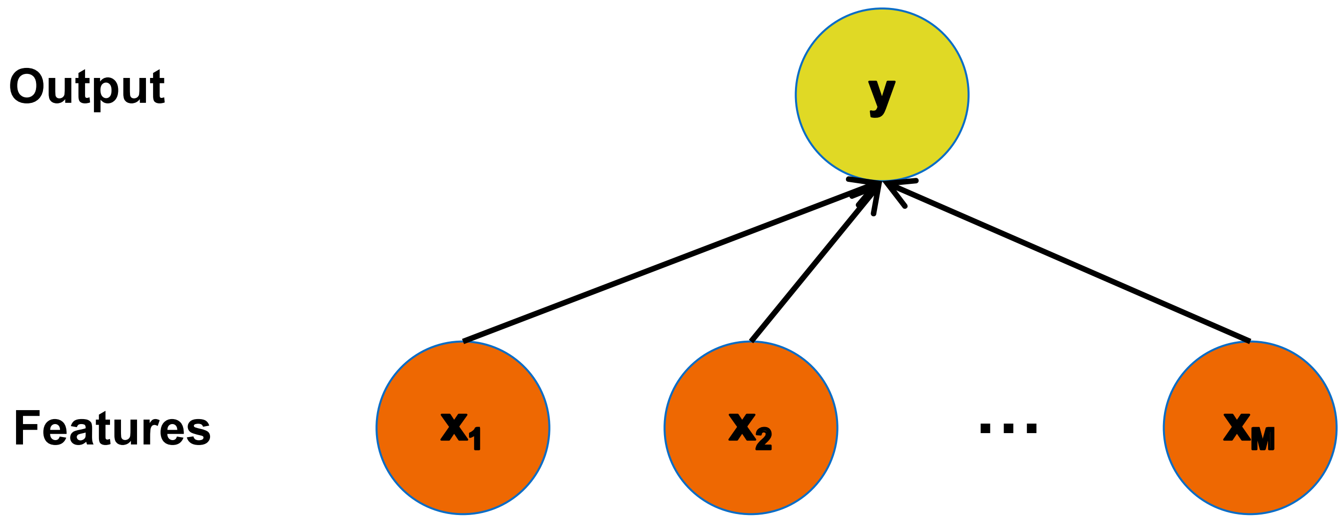
$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$$

(A) Input

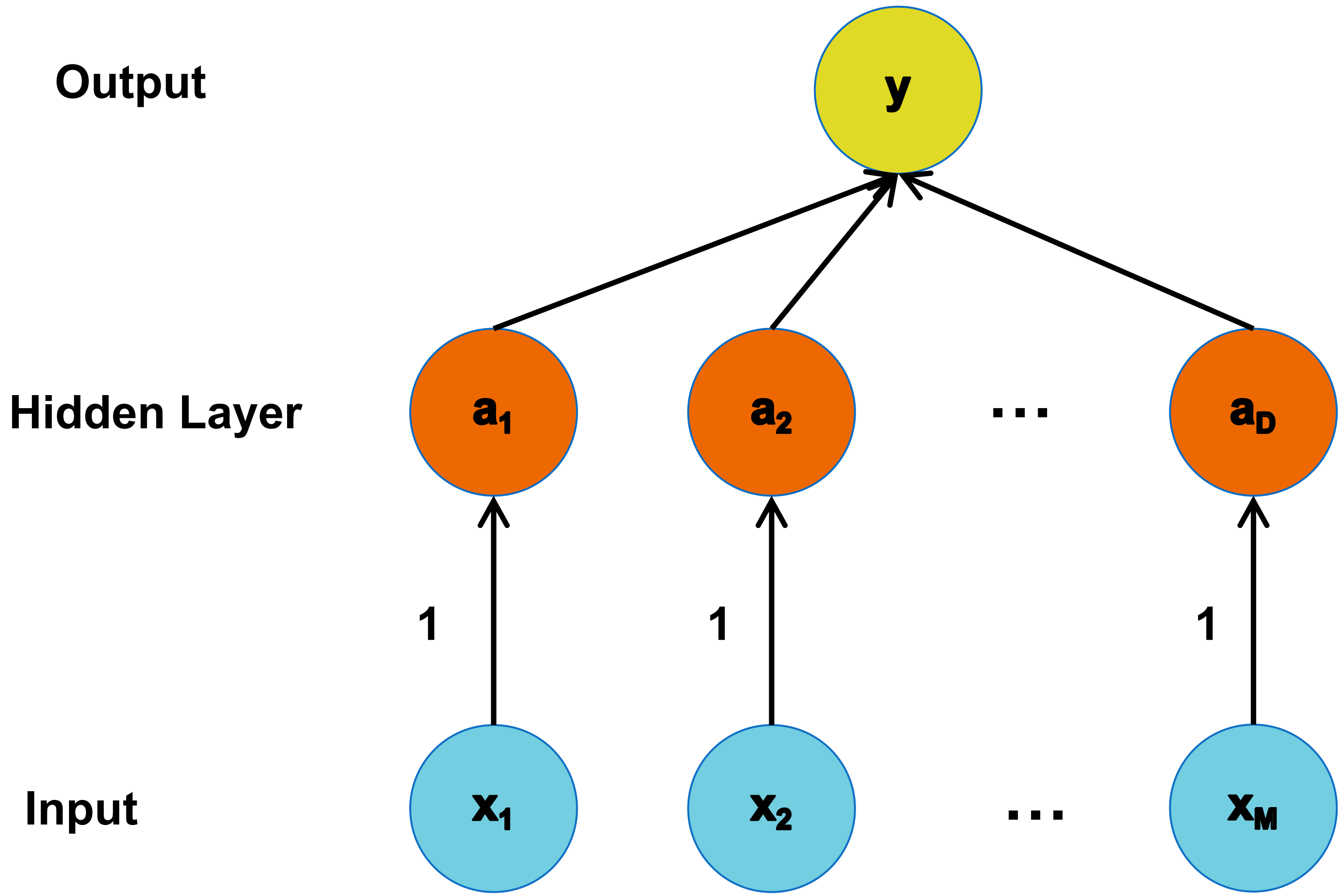
Given $x_i, \ \forall i$

Dr.Yanmei Zheng

Building a Neural Net



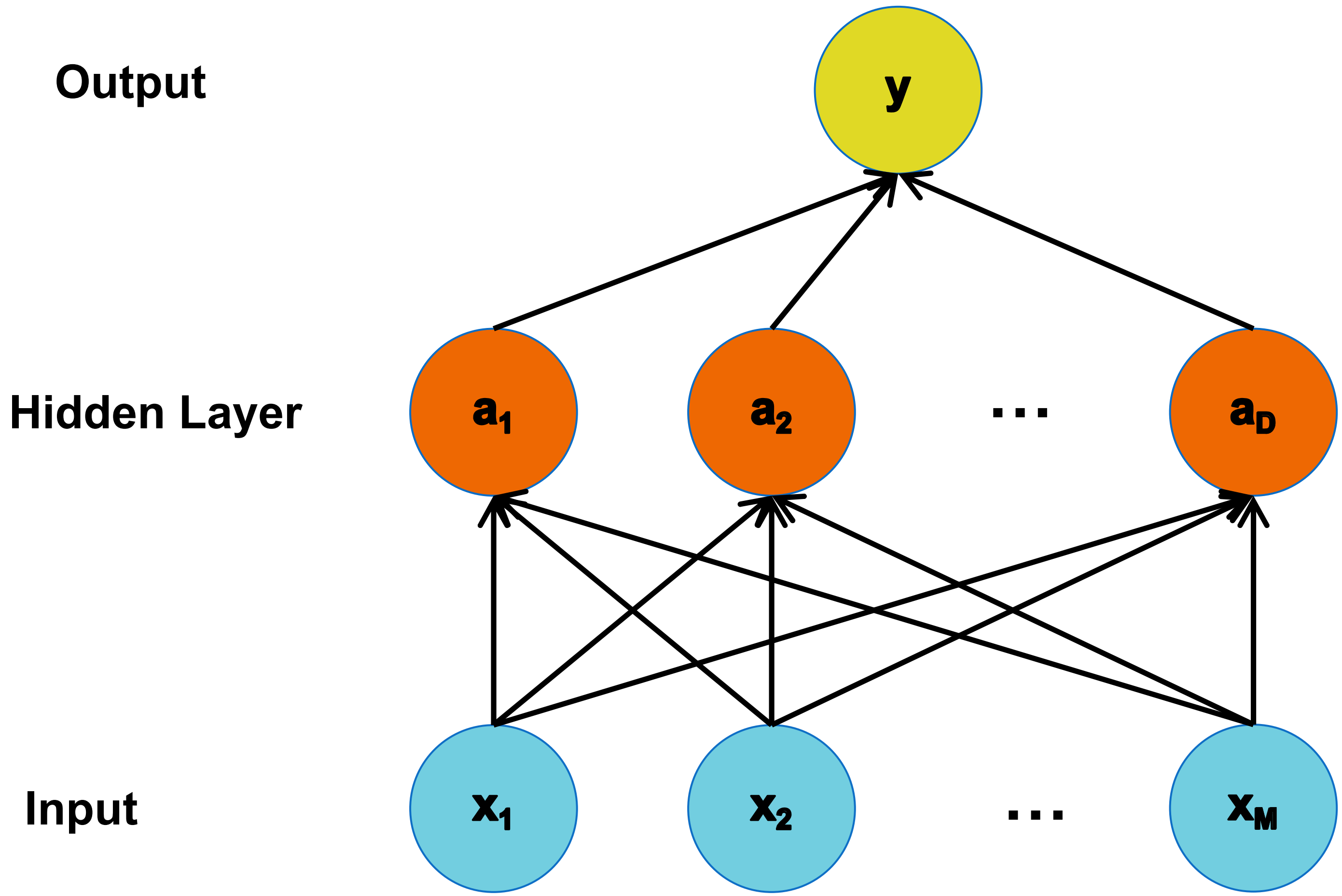
Building a Neural Net



D = M

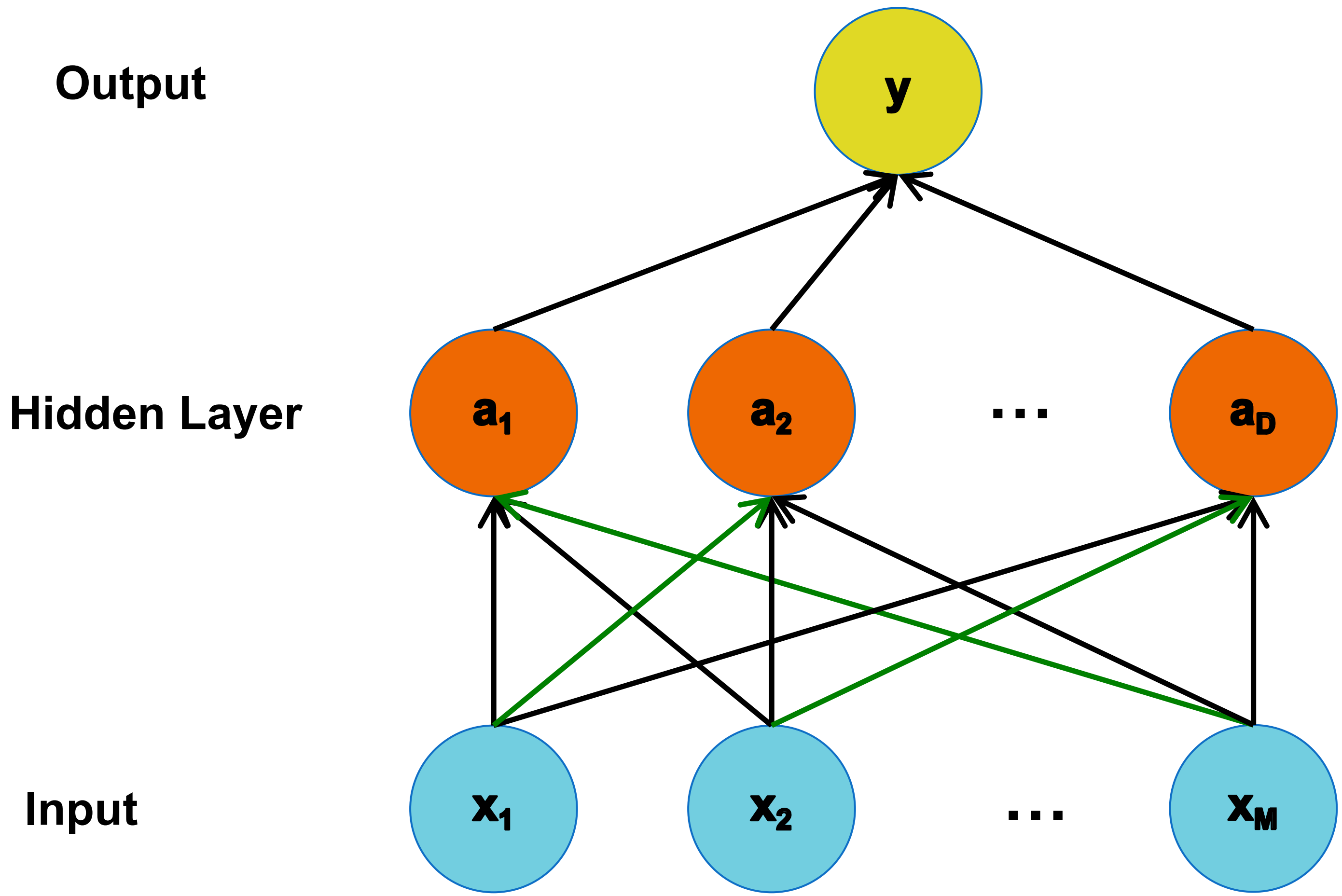
Dr.Yanmei Zheng

Building a Neural Net



D = M

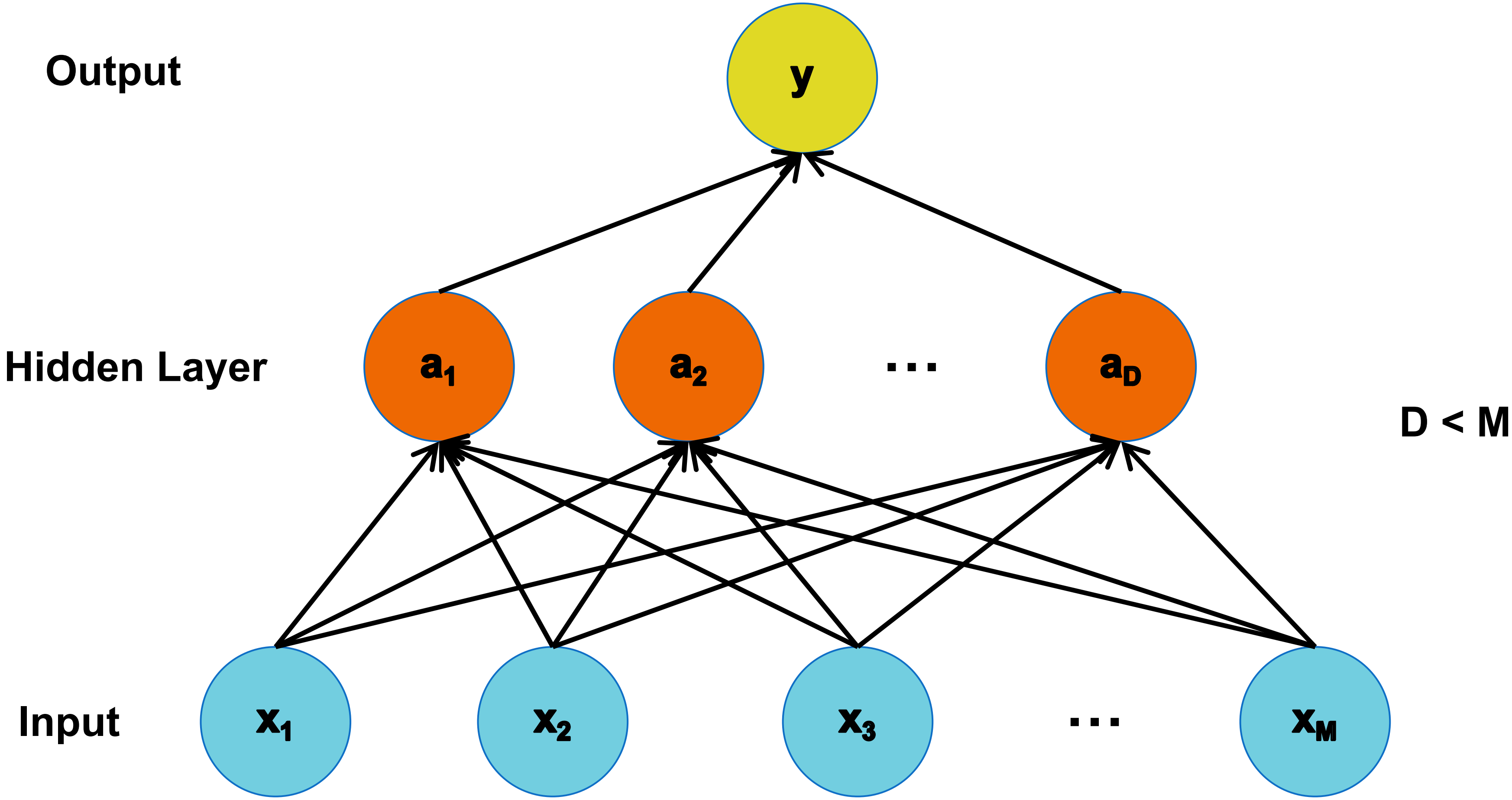
Building a Neural Net



D = M

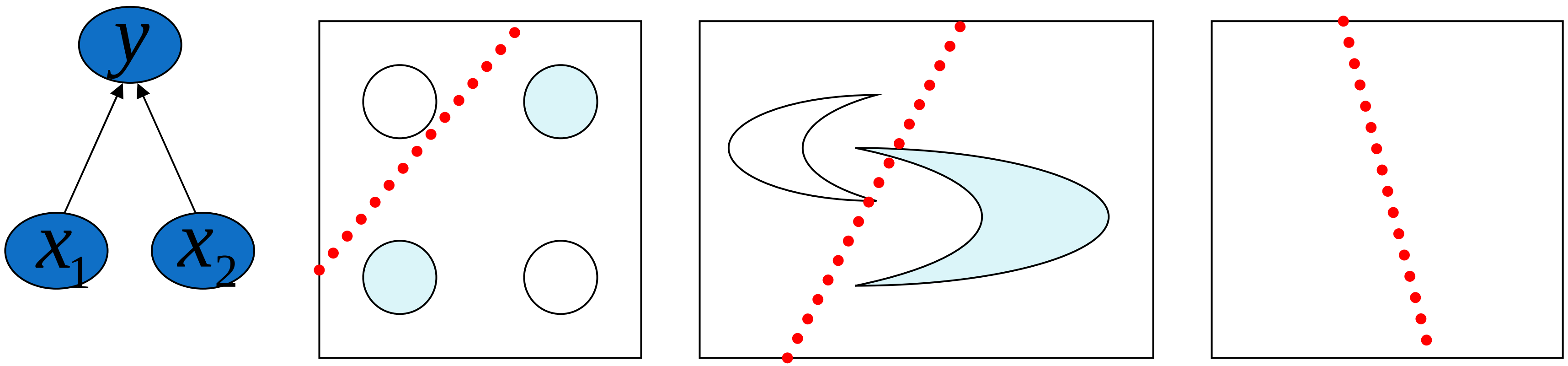
Dr.Yanmei Zheng

Building a Neural Net



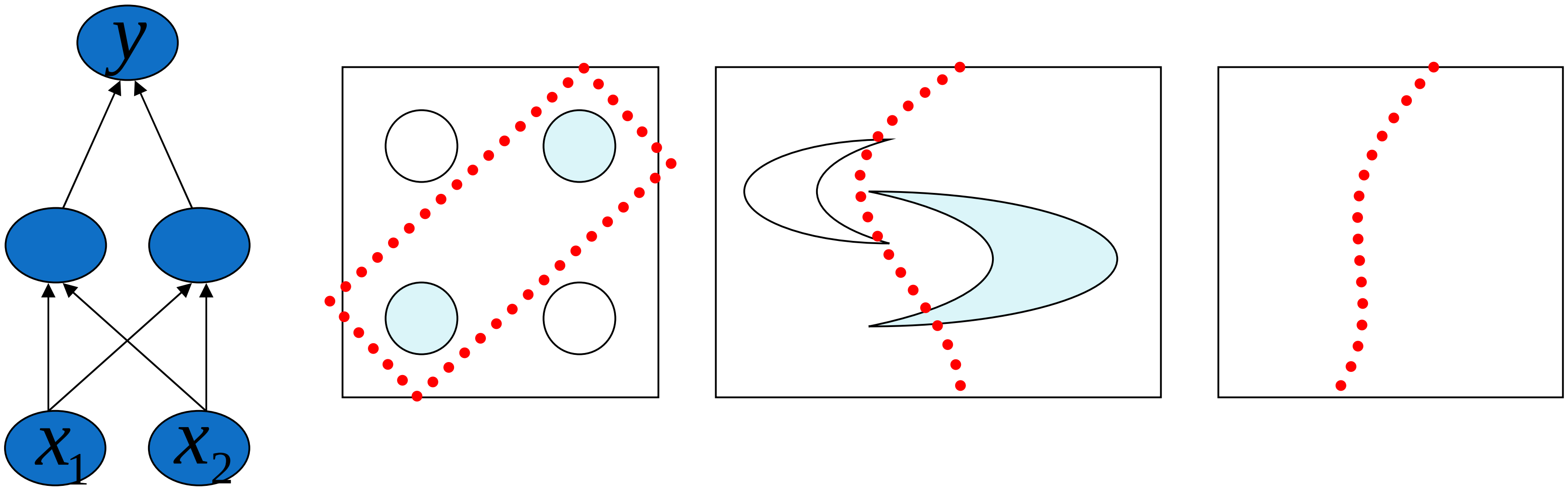
Decision Boundary

- 0 hidden layers: linear classifier
  1. Hyperplanes



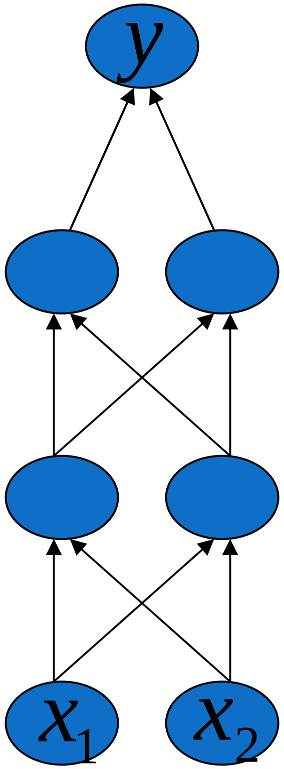
Decision Boundary

- 1 hidden layer
  1. Boundary of convex region (open or closed)

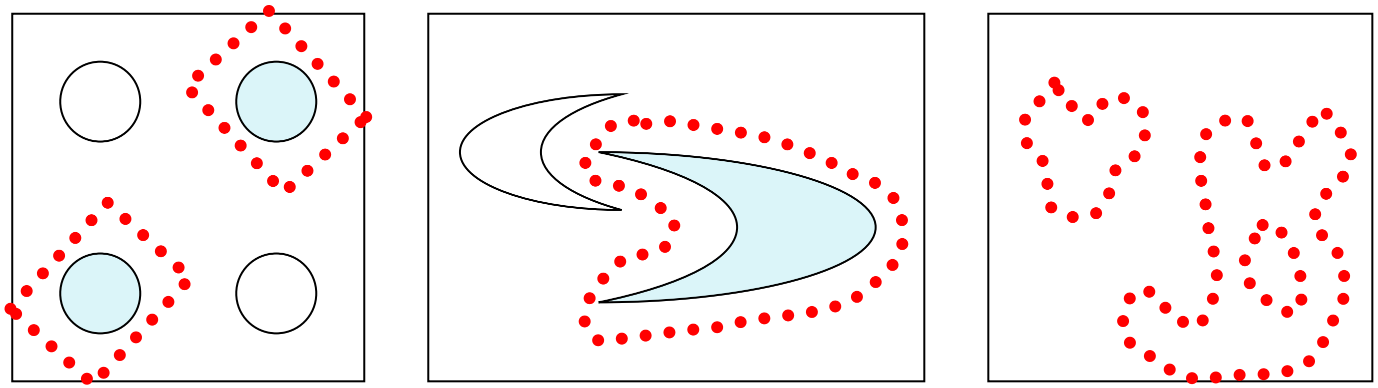


Dr.Yanmei Zheng

Decision Boundary



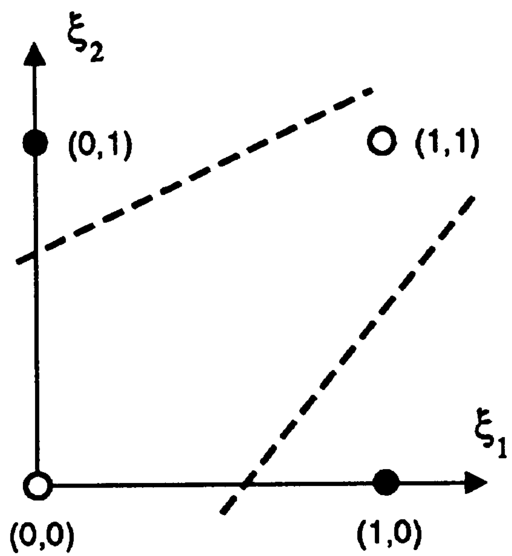
- 2 hidden layers
  1. Combinations of convex regions



Solution in 1980s: Multilayer perceptrons

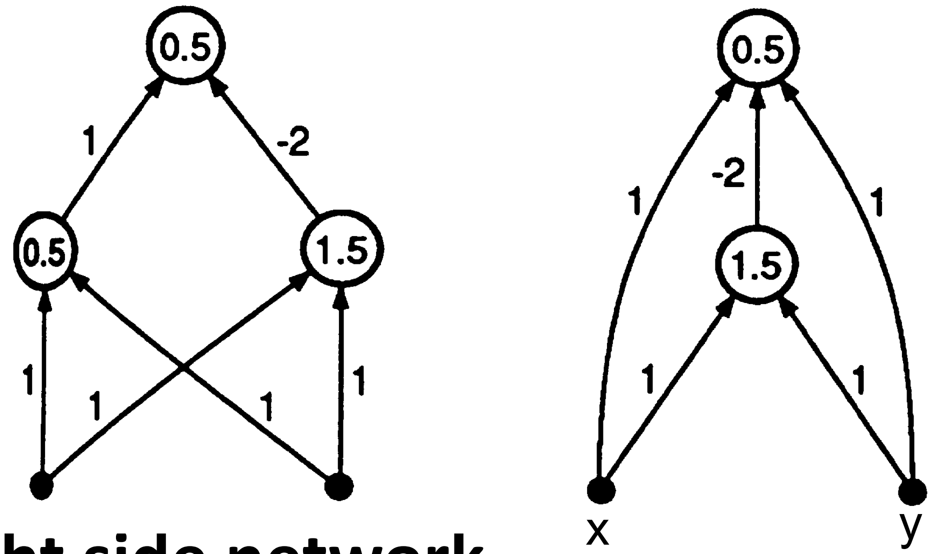
- Removes many limitations of single-layer networks
  1. Can solve XOR

- Exercise: Draw a two-layer perceptron that computes the XOR function
  1. 2 binary inputs ξ_1 and ξ_2
  2. 1 binary output
  3. One "hidden" layer
  4. Find the appropriate weights and threshold



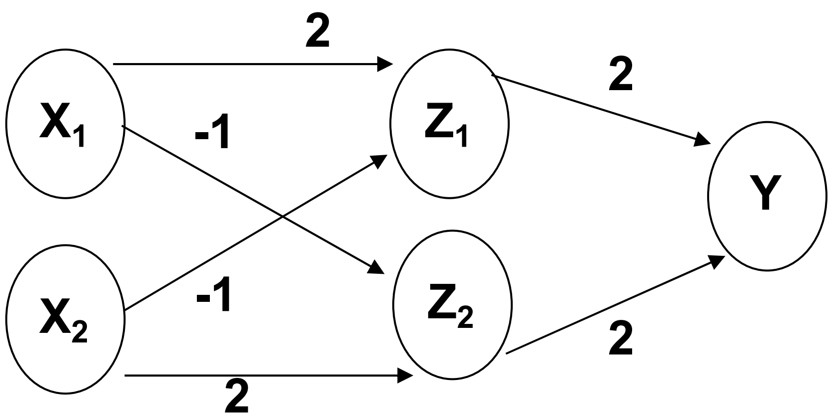
Solution in 1980s: Multilayer perceptrons

- Examples of two-layer perceptrons that compute XOR



- E.g. Right side network
  1. Output is 1 if and only if x + y – 2(x + y – 1.5 > 0) – 0.5 > 0

The First Neural Neural Networks



Threshold(*Y*) = 2 Threshold(*Y*) = 2
XOR Function

XOR		
X1	**X2**	**Y**
1	1	0
1	0	1
0	1	1
0	0	0

X_1 XOR X_2 = (X_1 AND NOT X_2) OR (X_2 AND NOT X_1)

X1	X2	Z1	Z2	Z1'	Z2'	Y	Y'
1	1	-1	-1	0	0	-2	0
1	0	0	-3	1	0	0	1
0	1	-3	0	0	1	2	1
0	0	-2	-2	0	0	-2	0

The First Neural Neural Networks

If we touch something cold we perceive heat

If we keep touching something cold we will perceive cold

If we touch something hot we will perceive heat

The First Neural Neural Networks

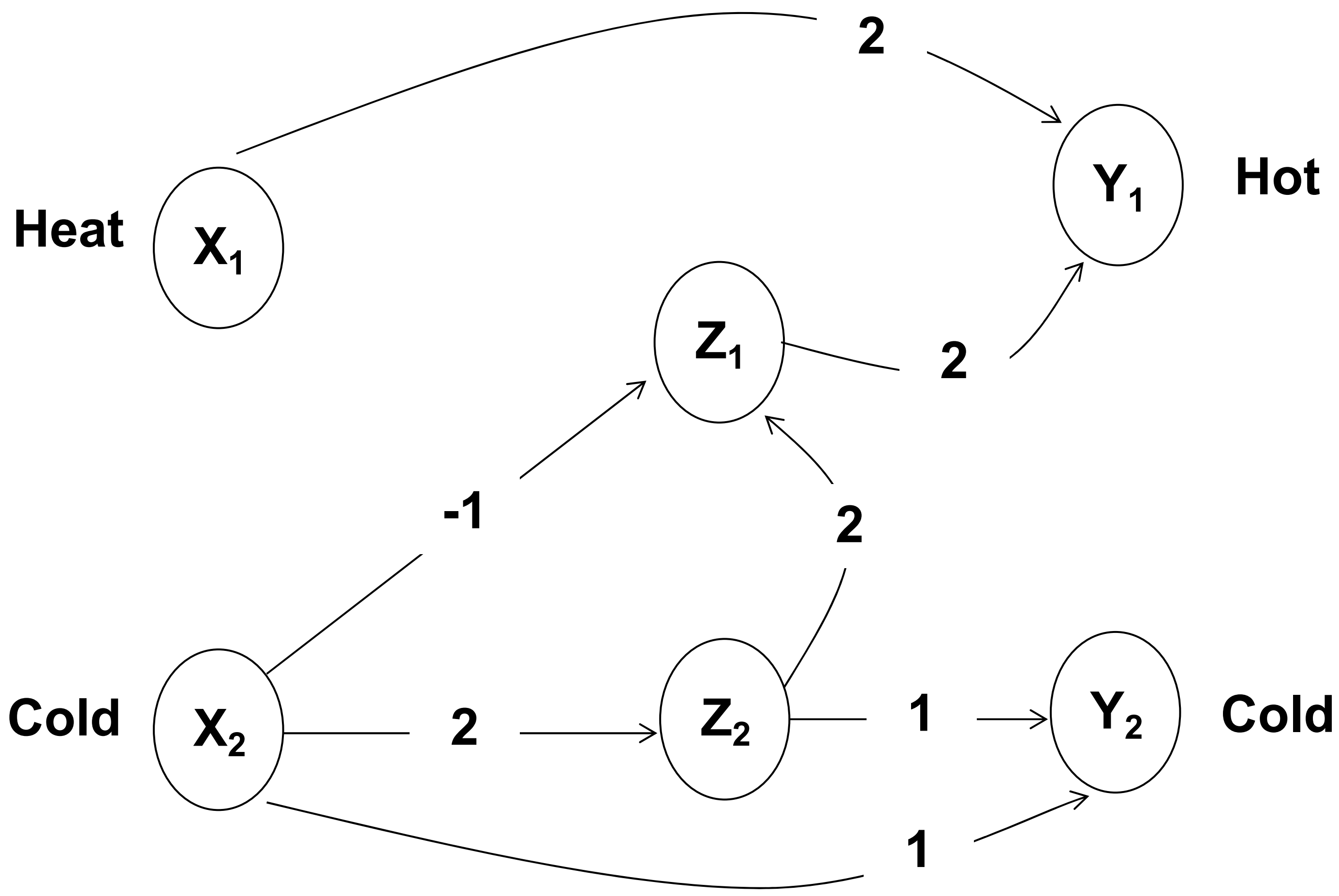
To model this we will assume that time is discrete

If cold is applied for one time step then heat will be perceived

If a cold stimulus is applied for two time steps then cold will be perceived

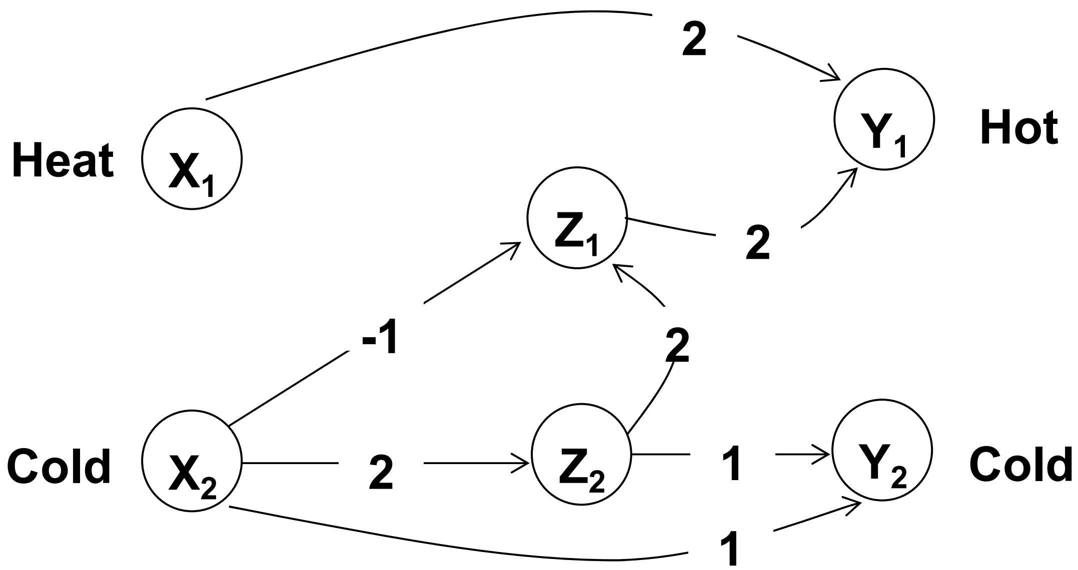
If heat is applied then we should perceive heat

The First Neural Neural Networks



Dr.Yanmei Zheng

The First Neural Neural Networks



- It takes time for the stimulus (applied at X_1 and X_2) to make its way to Y_1 and Y_2 where we perceive either heat or cold

- At t(0), we apply a stimulus to X_1 and X_2
- At t(1) we can update Z_1, Z_2 and Y_1
- At t(2) we can perceive a stimulus at Y_2
- At t(2+n) the network is fully functional

The First Neural Neural Networks

We want the system to perceive cold if a cold stimulus is applied for two time steps

$$Y_2(t) = X_2(t-2) \text{ } AND \text{ } X_2(t-1)$$

$X_2(t-2)$	$X_2(t-1)$	$Y_2(t)$
1	1	1
1	0	0
0	1	0
0	0	0

The First Neural Neural Networks

We want the system to perceive heat if either a hot stimulus is applied or a cold stimulus is applied (for one time step) and then removed

$$Y_1(t) = [\ X_1(t-1)\ ]\ \text{OR}\ [\ X_2(t-3)\ \text{AND NOT}\ X_2(t-2)\ ]$$

X2(t – 3)	X2(t – 2)	AND NOT	X1(t – 1)	OR
1	1	0	1	1
1	0	1	1	1
0	1	0	1	1
0	0	0	1	1
1	1	0	0	0
1	0	1	0	1
0	1	0	0	0
0	0	0	0	0

The First Neural Neural Networks

The network shows

$$Y_1(t) = X_1(t-1) \text{ OR } Z_1(t-1)$$

$$Z_1(t-1) = Z_2(t-2) \text{ AND NOT } X_2(t-2)$$

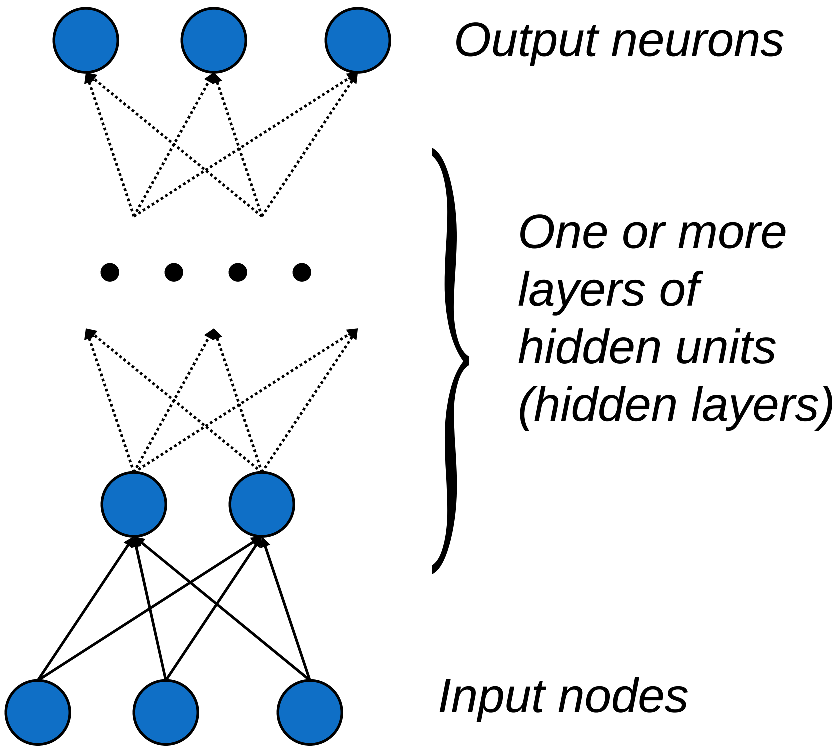
$$Z_2(t-2) = X_2(t-3)$$

Substituting, we get

$$Y_1(t) = [\ X_1(t-1)\ ] \text{ OR } [\ X_2(t-3) \text{ AND NOT } X_2(t-2)\ ]$$

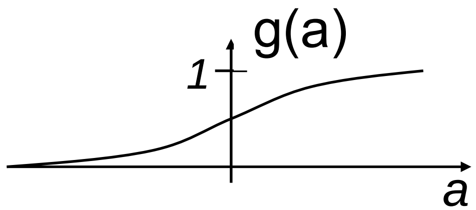
which is the same as our original requirements

Multilayer Perceptron



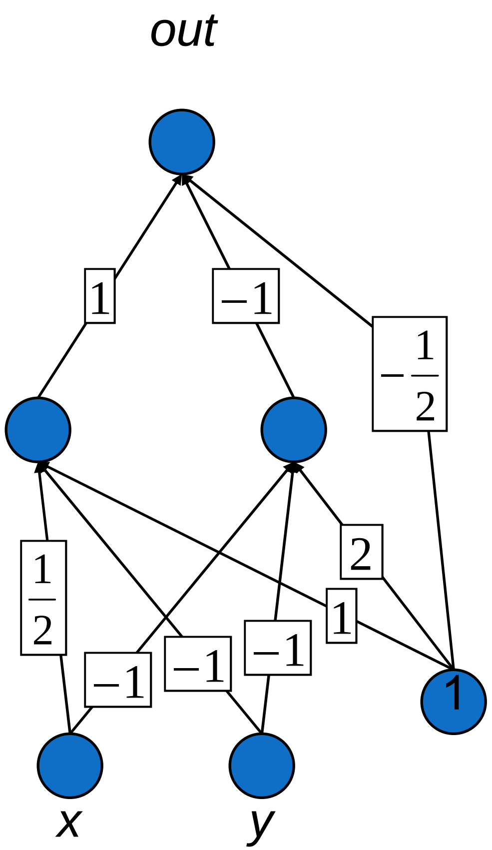
The most common output function (Sigmoid):

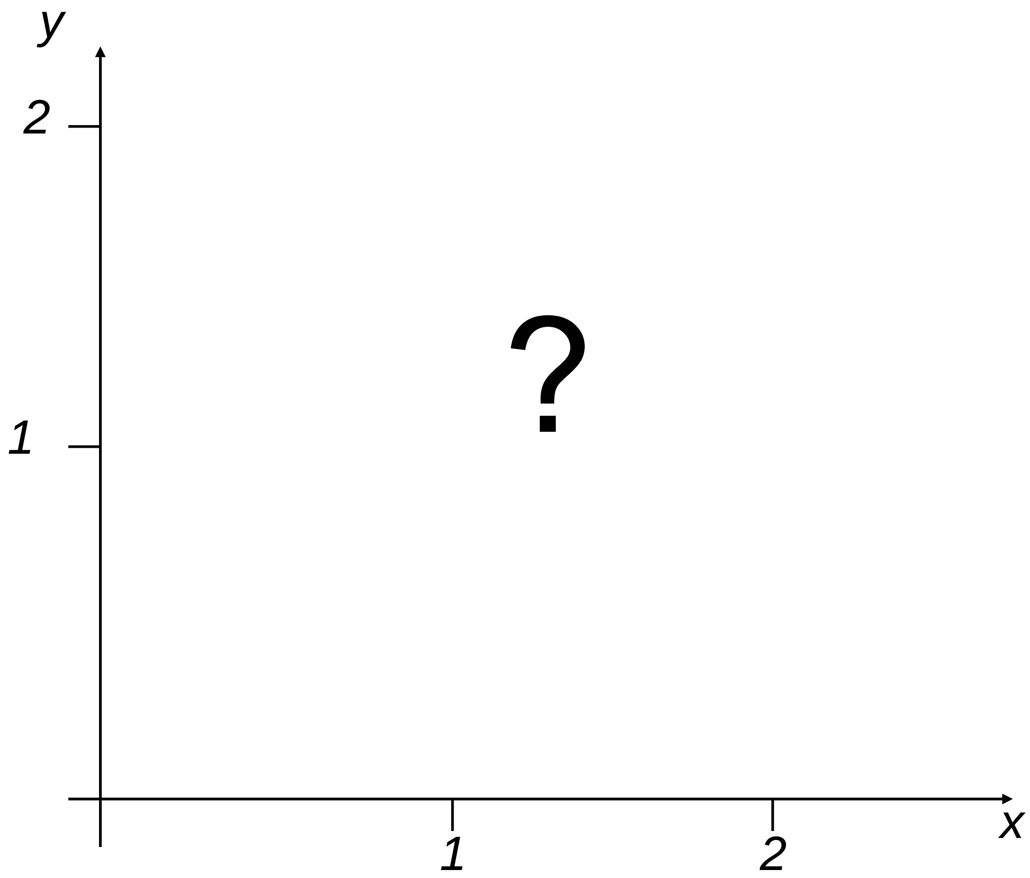
$$g(a) = \frac{1}{1 + e^{-\beta a}}$$



(non-linear squashing function)

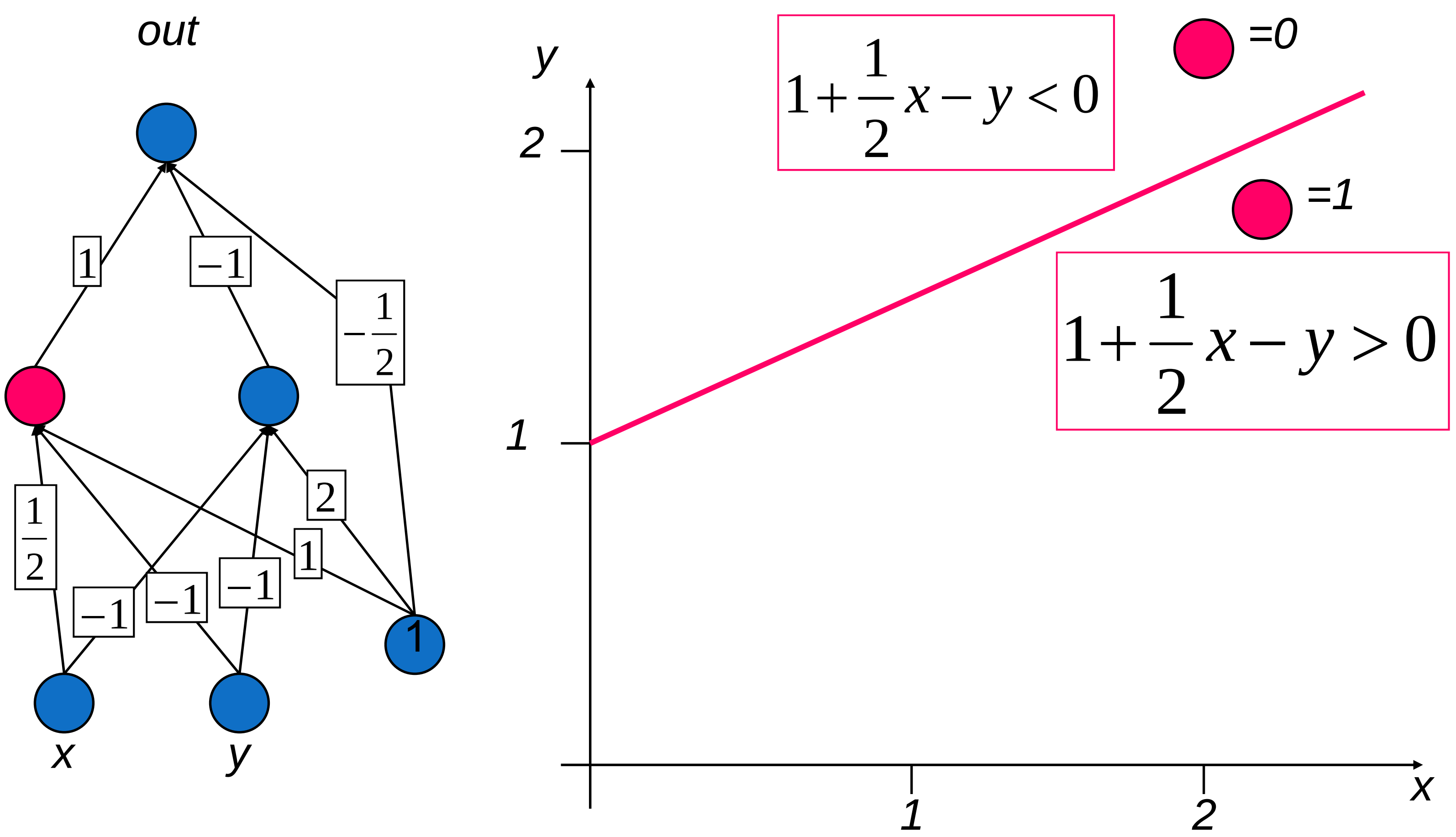
Example: Perceptrons as Constraint Satisfaction Networks



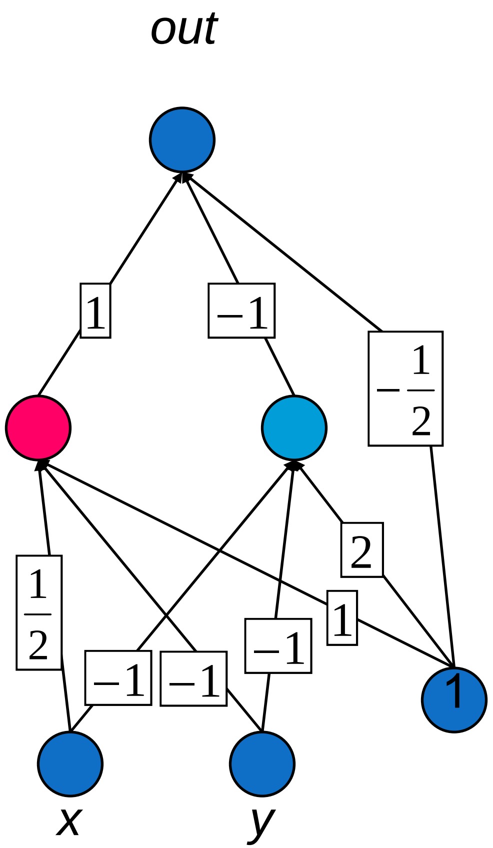


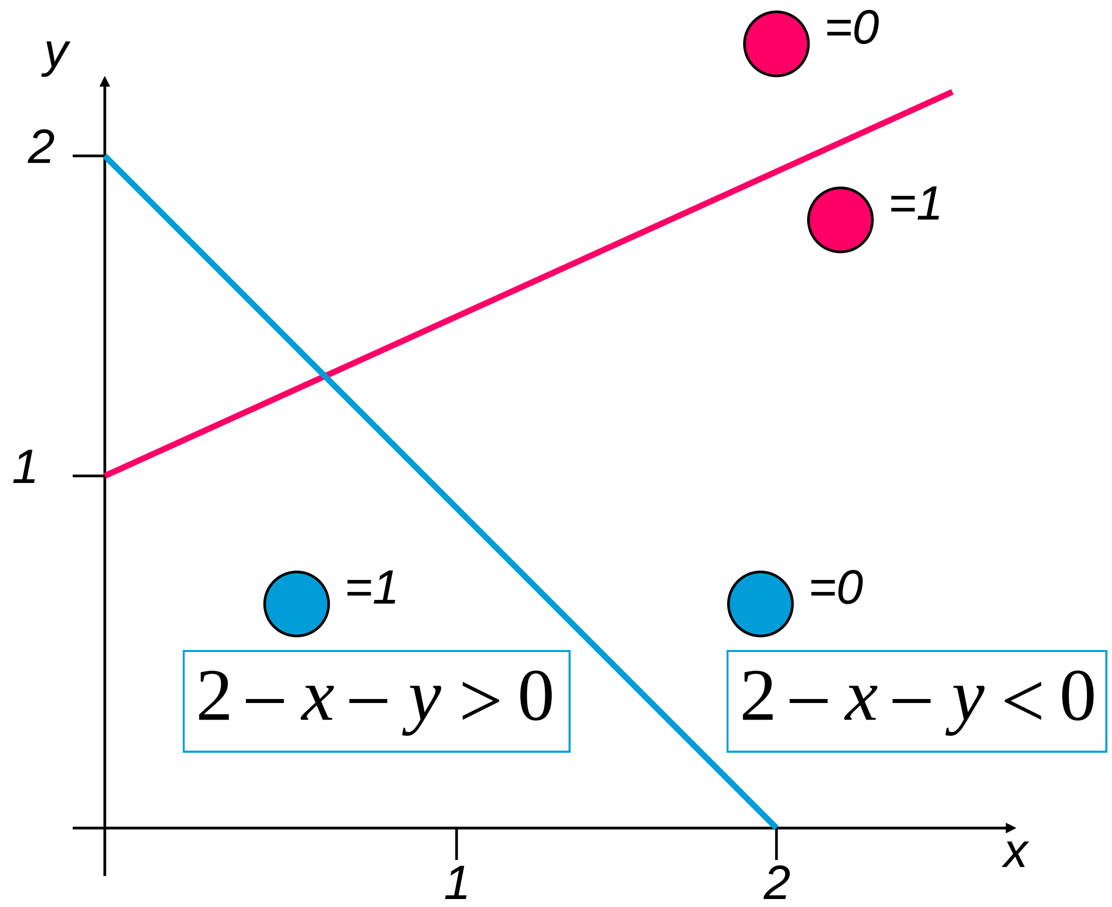
Dr.Yanmei Zheng

Example: Perceptrons as Constraint Satisfaction Networks

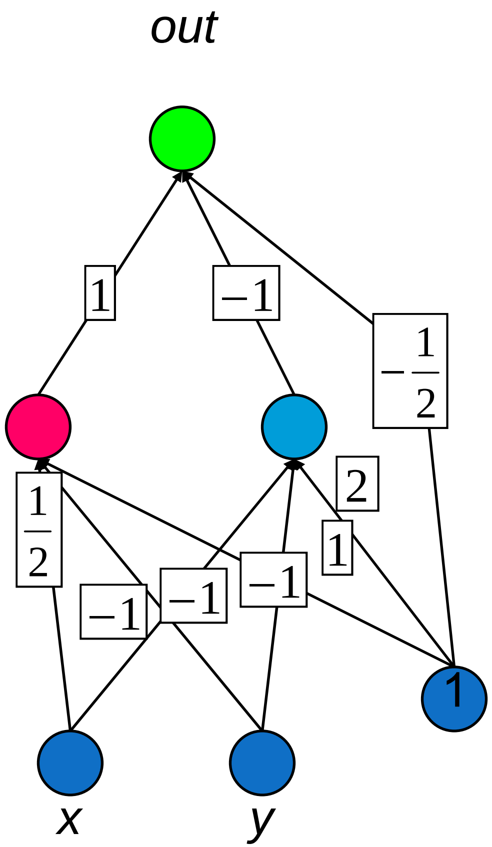


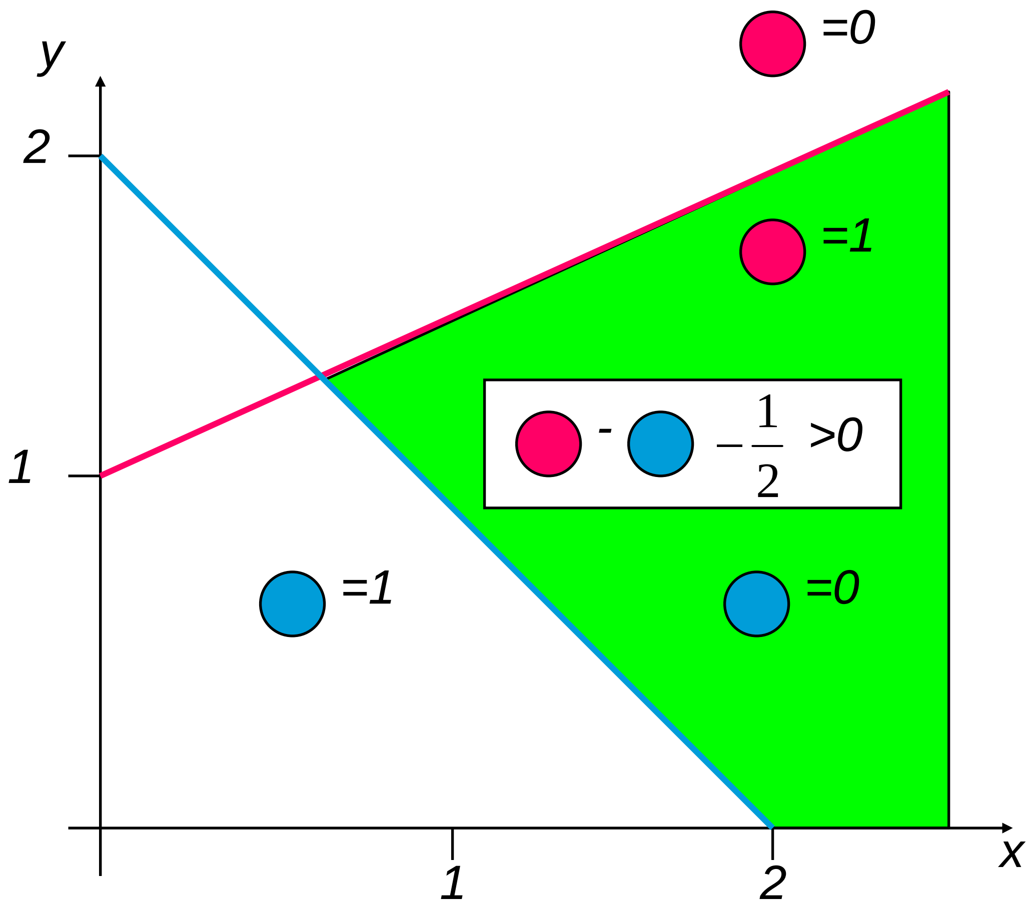
Example: Perceptrons as Constraint Satisfaction Networks





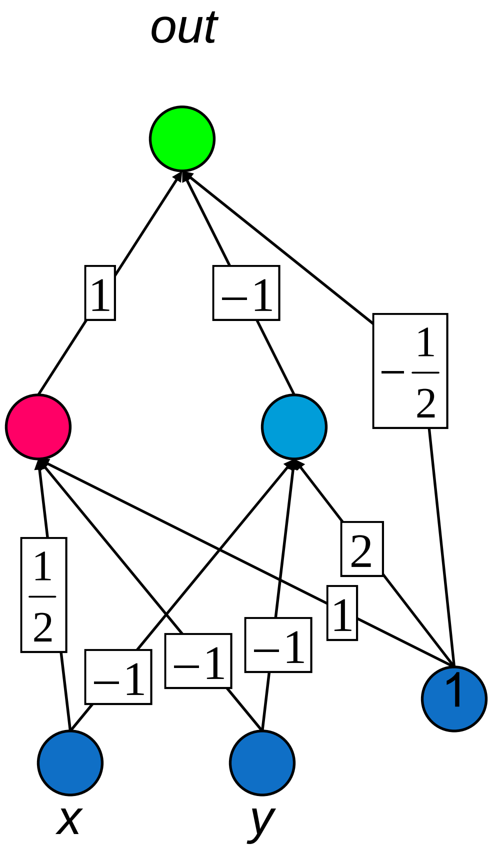
Example: Perceptrons as Constraint Satisfaction Networks

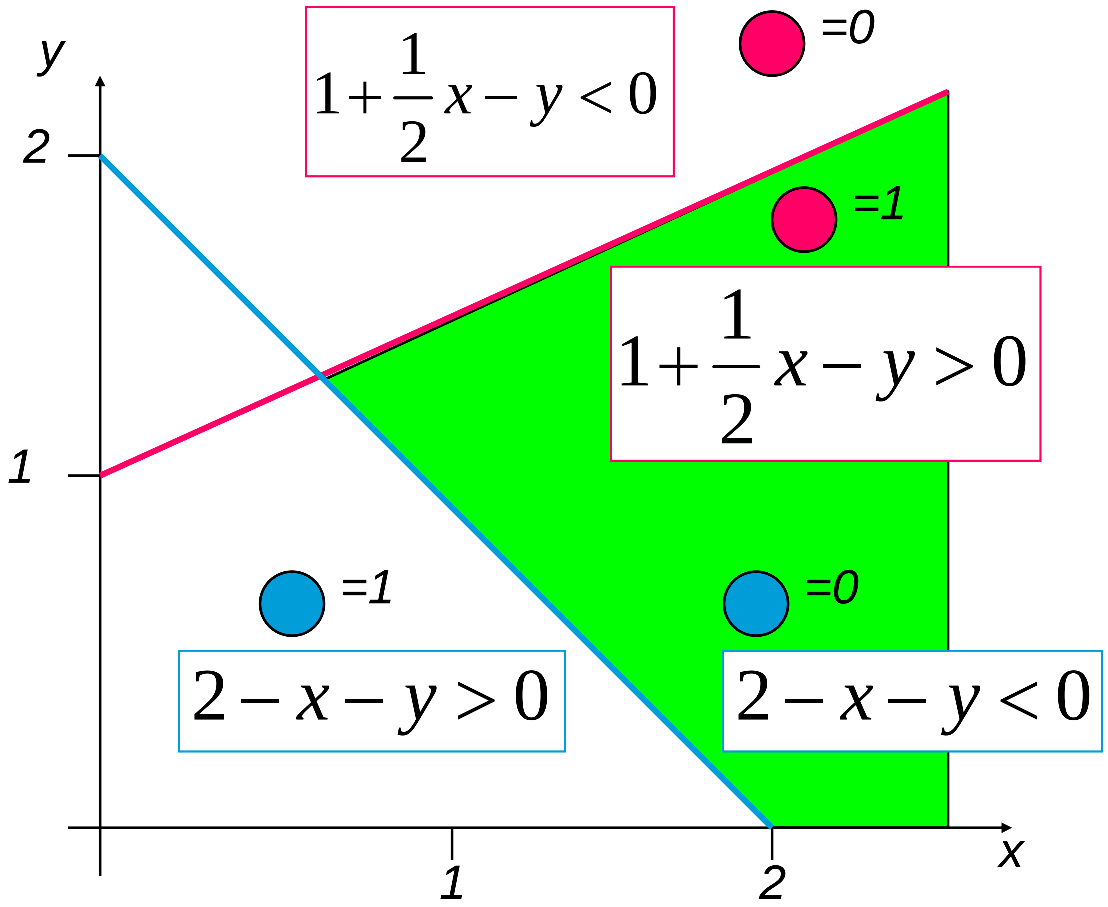




Dr.Yanmei Zheng

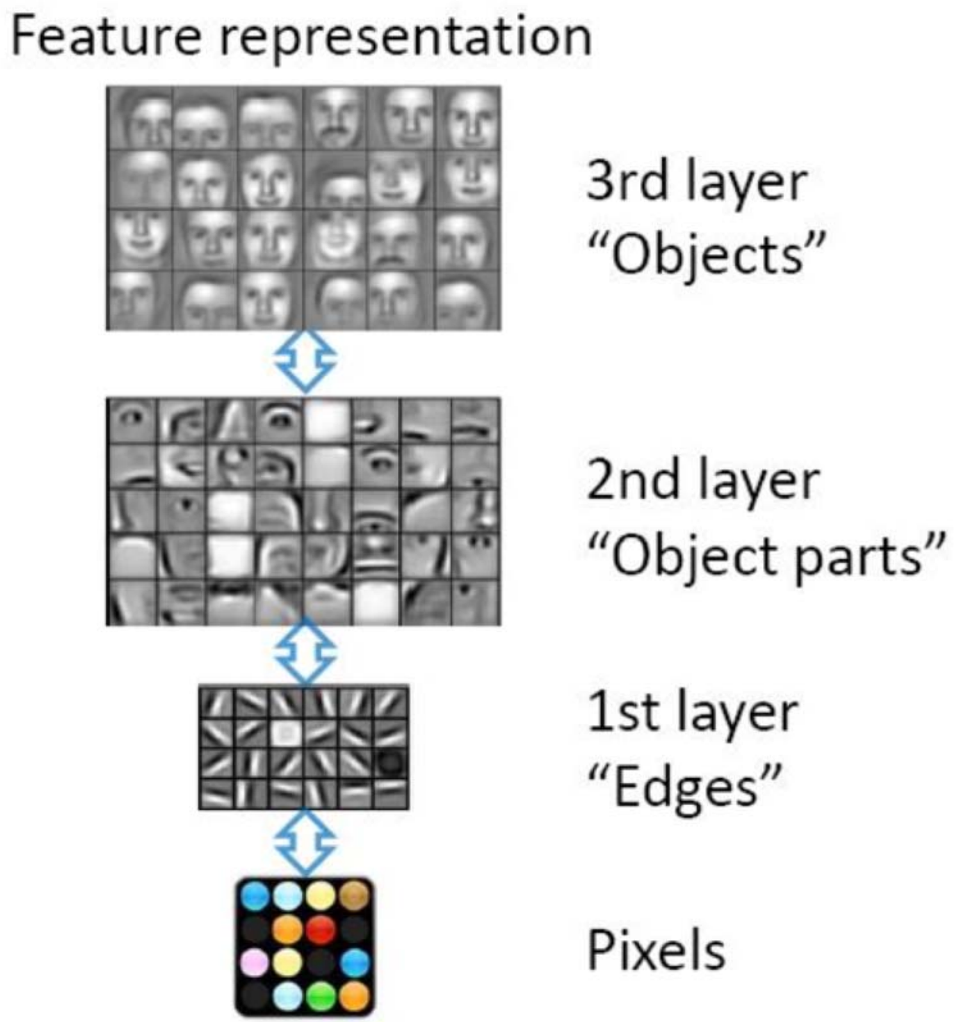
Example: Perceptrons as Constraint Satisfaction Networks





Different Levels of Abstraction

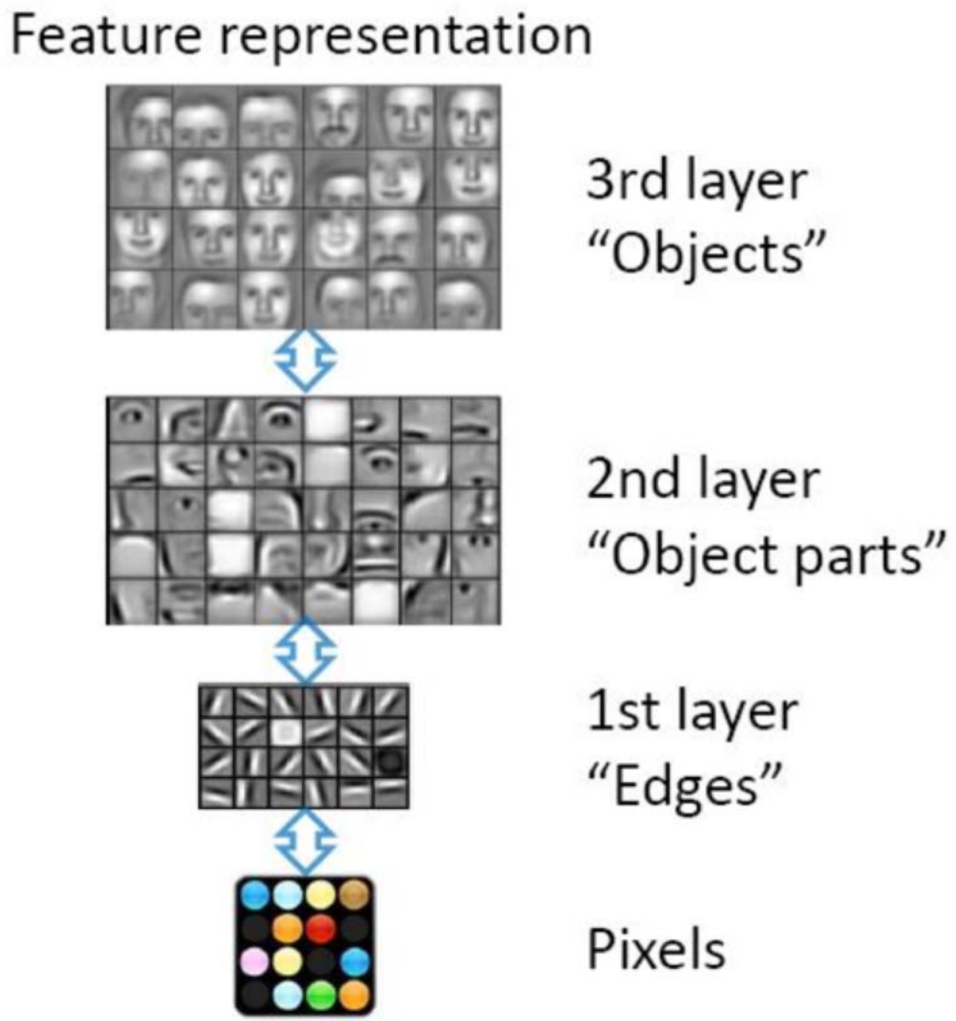
- We don't know the "right" levels of abstraction
- So let the model figure it out!



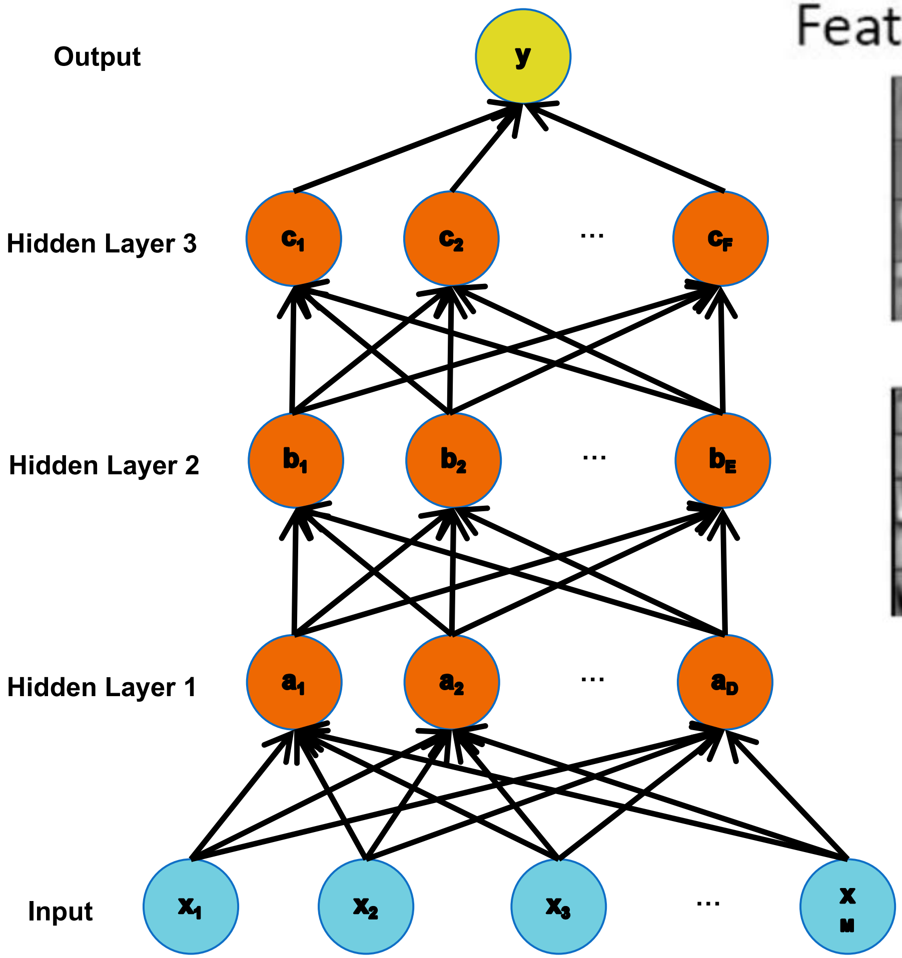
Different Levels of Abstraction

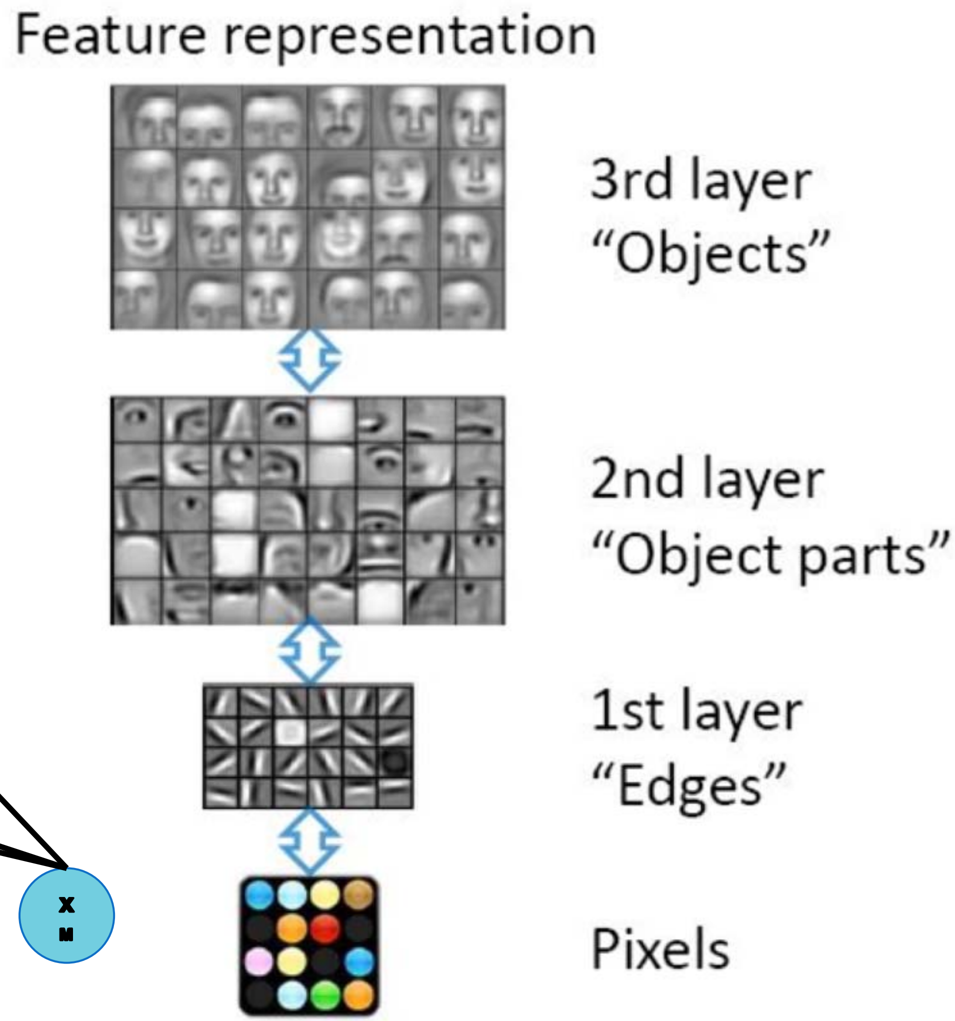
Face Recognition:

1. Deep Network can build up increasingly higher levels of abstraction
2. Lines, parts, regions



Different Levels of Abstraction





Architectures

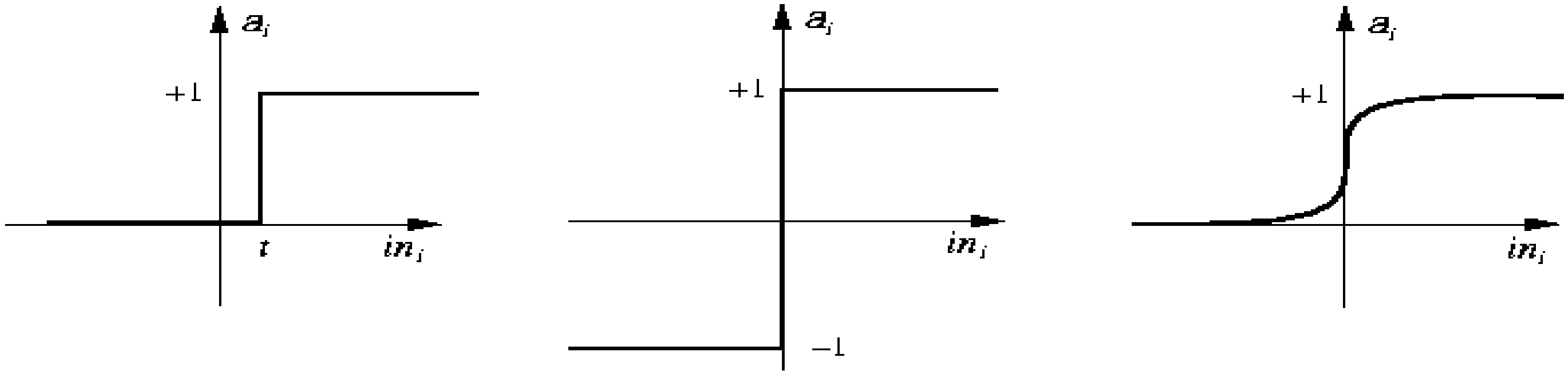
Neural Network Architectures

Even for a basic Neural Network, there are many design decisions to make:

1. **# of hidden layers (depth)**
2. **# of units per hidden layer (width)**
3. **Type of activation function (nonlinearity)**
4. **Form of objective function**

Activation Functions

Activation Functions

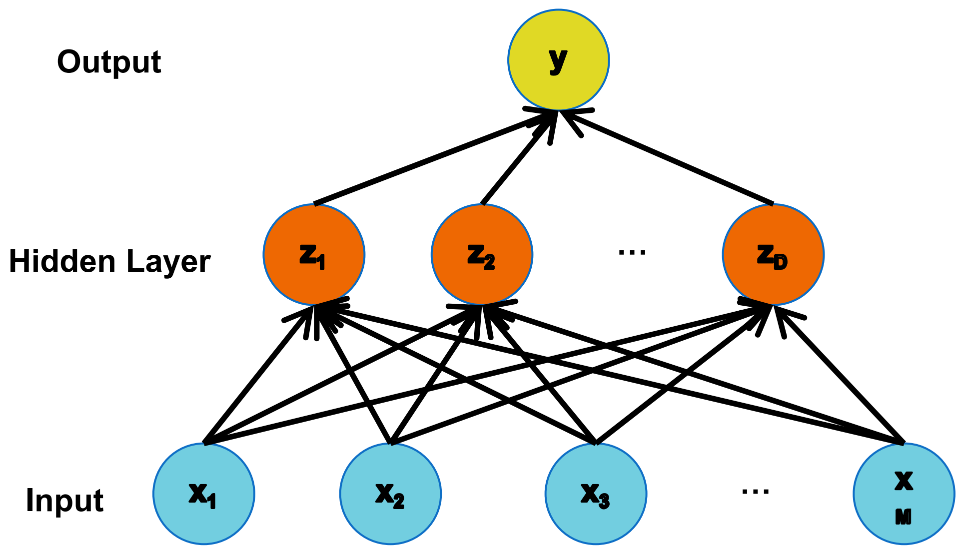


(a) Step function (b) Sign function (c) Sigmoid function

- $Step_t(x)$ = 1 if x >= t, else 0
- Sign(x) = +1 if x >= 0, else –1
- Sigmoid(x) = $1/(1+e^{-x})$
- Identity Function

Activation Functions

Neural Network with sigmoid activation functions



(F) Loss
$J = \frac{1}{2}(y - y^*)^2$

(E) Output (sigmoid)
$y = \frac{1}{1+\exp(-b)}$

(D) Output (linear)
$b = \sum_{j=0}^{D} \beta_j z_j$

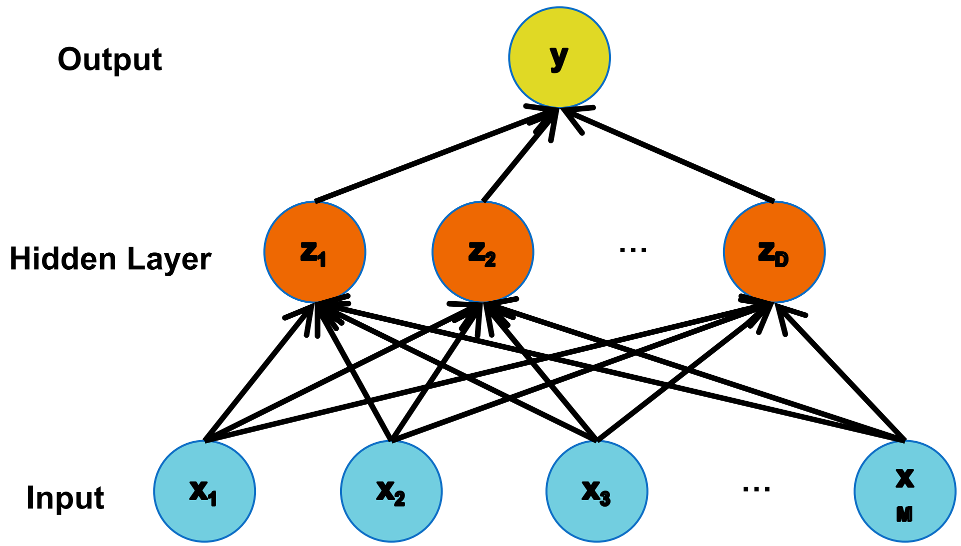
(C) Hidden (sigmoid)
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

(B) Hidden (linear)
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) Input
Given $x_i, \ \forall i$

Activation Functions

Neural Network with arbitrary nonlinear activation functions



(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

(E) **Output (nonlinear)**
$y = \sigma(b)$

(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

(C) **Hidden (nonlinear)**
$z_j = \sigma(a_j), \ \forall j$

(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

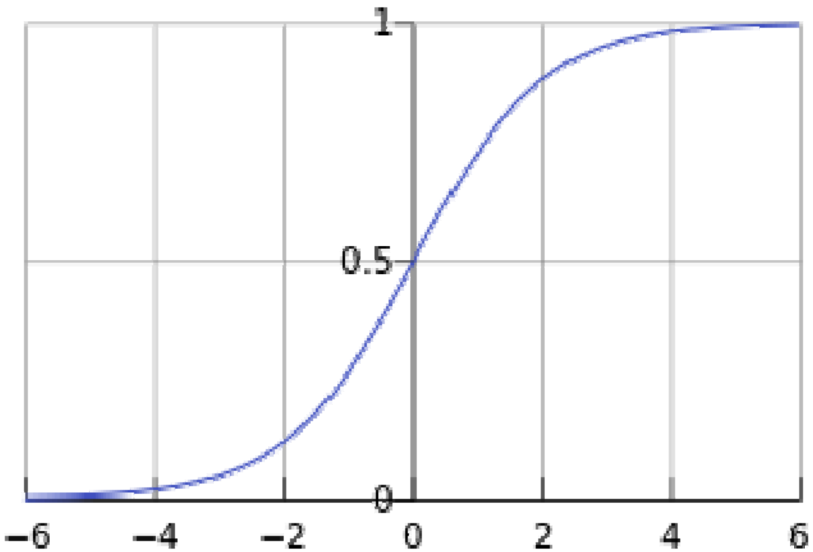
(A) **Input**
Given $x_i, \ \forall i$

Dr.Yanmei Zheng

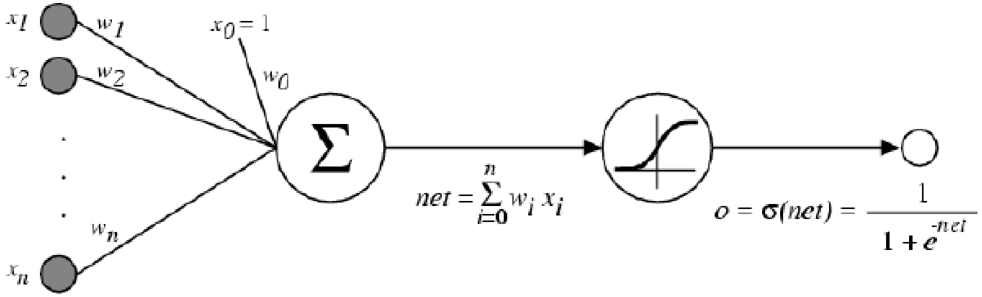
Activation Functions

Sigmoid / Logistic Function

$$\text{logistic}(u) \equiv \frac{1}{1+e^{-u}}$$

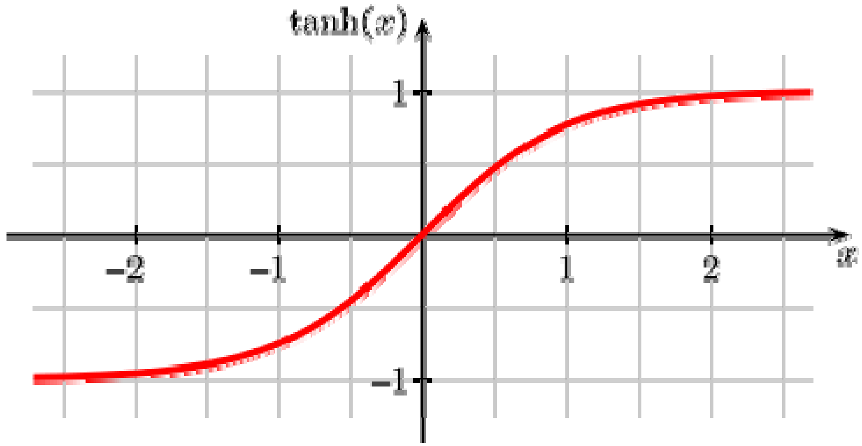


So far, we've assumed that the activation function (nonlinearity) is always the sigmoid function…



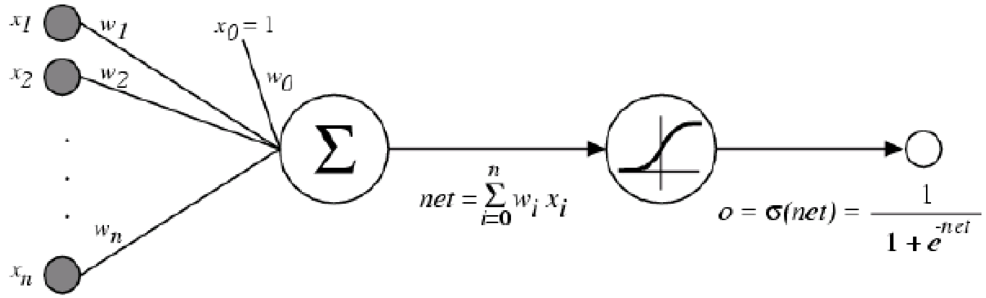
Activation Functions

- A new change: modifying the nonlinearity
  - ✓ The logistic is not widely used in modern ANNs



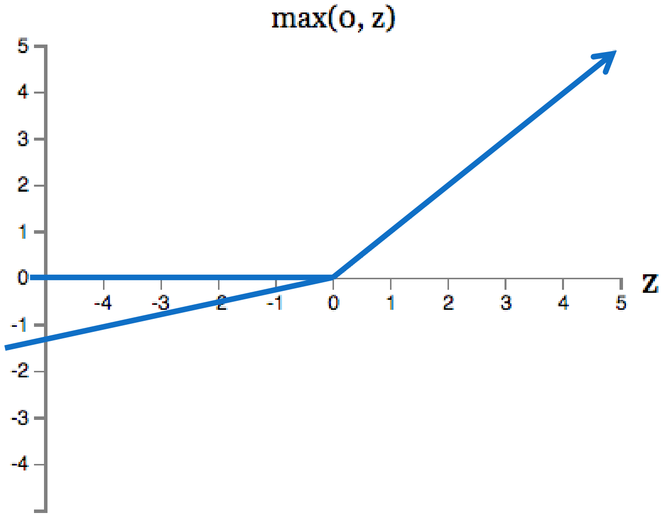
Alternate 1: tanh

Like logistic function but shifted to range [-1, +1]



Activation Functions

- A new change: modifying the nonlinearity
  - ✓ reLU often used in vision tasks

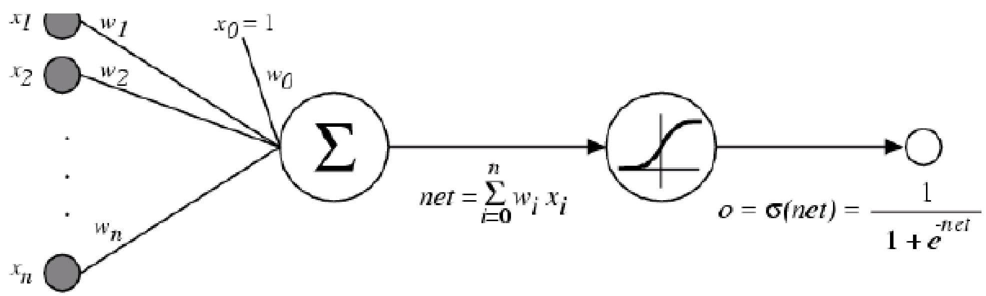


Alternate 2: rectified linear unit

Linear with a cutoff at zero

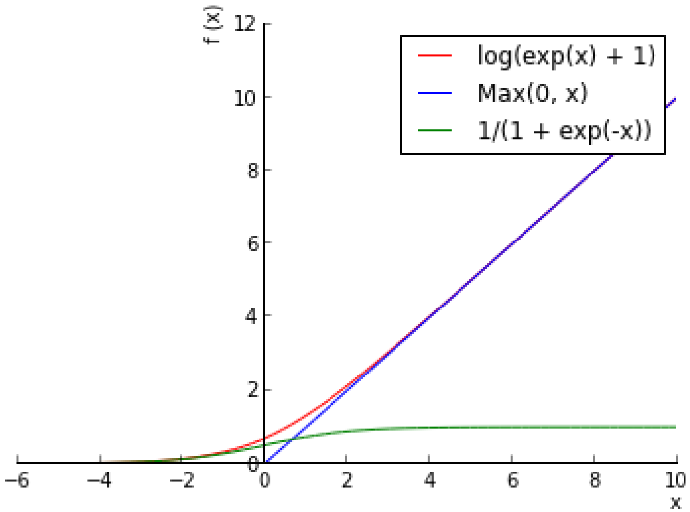
(Implementation: clip the gradient when you pass zero)

$$\max(0, w \cdot x + b).$$



Activation Functions

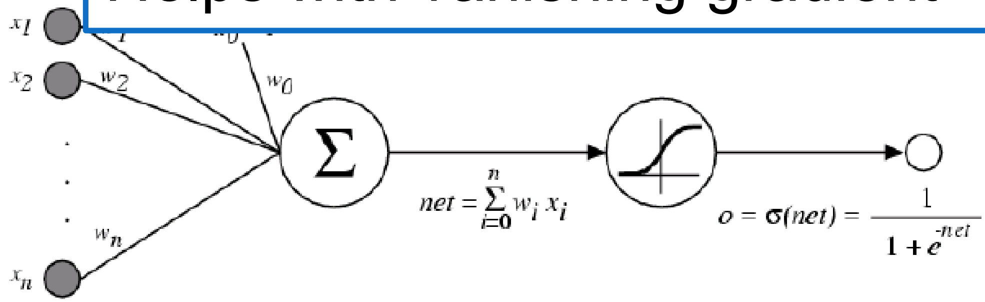
- **A new change: modifying the nonlinearity**
  - ✓ **reLU often used in vision tasks**



Alternate 2: rectified linear unit

Soft version: log(exp(x)+1)

Doesn't saturate (at one end)
Sparsifies outputs
Helps with vanishing gradient



Objective Functions for NNs

- **Regression:**
  - ✓ **Use the same objective as Linear Regression**
  - ✓ **Quadratic loss (i.e. mean squared error)**
- **Classification:**
  - ✓ **Use the same objective as Logistic Regression**
  - ✓ **Cross-entropy (i.e. negative log likelihood)**
  - ✓ **This requires probabilities, so we add an additional "softmax" layer at the end of our network**

	Forward	Backward
Quadratic	$J = \frac{1}{2}(y - y^*)^2$	$\frac{dJ}{dy} = y - y^*$
Cross Entropy	$J = y^* \log(y) + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{y - 1}$

Cross-entropy vs. Quadratic loss

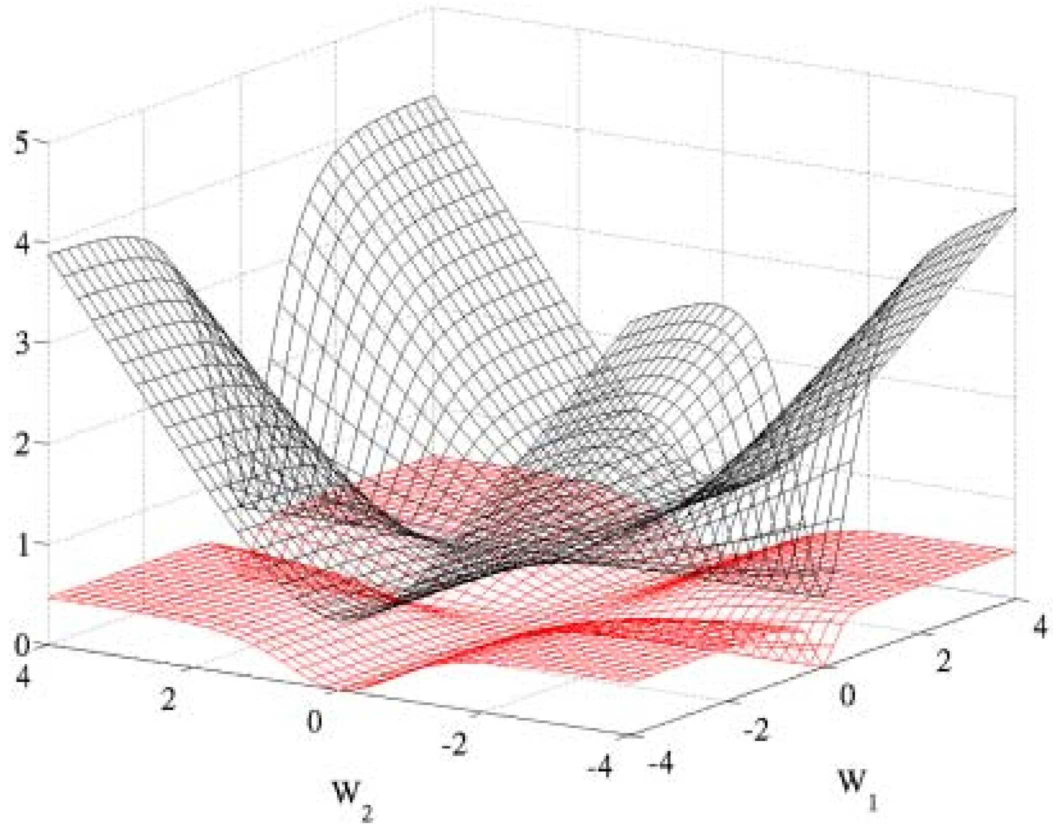


Figure 5: *Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.*

Figure from Glorot & Bentio (2010)

Backpropagation

Backpropagation

- **Question 1:**
  When can we compute the gradients of the parameters of an arbitrary neural network?

- **Question 2:**
  When can we make the gradient computation efficient?

Backpropagation

- In order to adapt the weights from input to hidden units, we again want to apply the delta rule. In this case, however, we do not have a value for the hidden units.

Backpropagation

- Calculate the activation of the hidden units

$$h_j = f\left(\sum_{k=0}^{n} v_{jk} x_k\right)$$

- And the activation of the output units

$$y_i = f\left(\sum_{j=0} w_{ij} h_j\right)$$

Backpropagation

- If we have μ pattern to learn the error is

$$E = \frac{1}{2}\sum_{\mu}\sum_{i}\left(t_i^{\mu} - y_i^{\mu}\right)^2 = \frac{1}{2}\sum_{\mu}\sum_{i}\left[t_i^{\mu} - f\left(\sum_{j} w_{ij} h_j^{\mu}\right)\right]^2$$

$$= \frac{1}{2}\sum_{\mu}\sum_{i}\left[t_i^{\mu} - f\left(\sum_{j} w_{ij}\, f\left(\sum_{k=0}^{n} v_{jk} x_k^{\mu}\right)\right)\right]^2$$

Backpropagation

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta \sum_{\mu} \left(t_i^{\mu} - y_i^{\mu} \right) \dot{f} \left(A_i^{\mu} \right) h_j^{\mu} = \eta \sum_{\mu} \delta_i^{\mu} h_j^{\mu}$$

$$\delta_i^{\mu} = \left(t_i^{\mu} - y_i^{\mu} \right) \dot{f} \left(A_i^{\mu} \right)$$

$$E = \tfrac{1}{2} \sum_{\mu} \sum_{i} \left(t_i^{\mu} - y_i^{\mu} \right)^2 = \tfrac{1}{2} \sum_{\mu} \sum_{i} \left[t_i^{\mu} - f \left(\sum_{j} w_{ij} h_j^{\mu} \right) \right]^2$$

$$= \tfrac{1}{2} \sum_{\mu} \sum_{i} \left[t_i^{\mu} - f \left(\sum_{j} w_{ij} f \left(\sum_{k=0}^{n} v_{jk} x_k^{\mu} \right) \right) \right]^2$$

Backpropagation

$$\Delta v_{jk} = -\eta \frac{\partial E}{\partial v_{jk}} = -\eta \sum_{\mu} \frac{\partial E}{\partial h_j^{\mu}} \frac{\partial h_j^{\mu}}{\partial v_{jk}}$$

$$= \eta \sum_{\mu} \sum_{i} \left(t_i^{\mu} - y_i^{\mu} \right) \dot{f}\left(A_i^{\mu} \right) w_{ij} \; \dot{f}\left(A_j^{\mu} \right) x_k^{\mu}$$

$$= \eta \sum_{\mu} \sum_{i} \delta_i^{\mu} w_{ij} \; \dot{f}\left(A_j^{\mu} \right) x_k^{\mu}$$

$$E = \tfrac{1}{2} \sum_{\mu} \sum_{i} \left(t_i^{\mu} - y_i^{\mu} \right)^2 = \tfrac{1}{2} \sum_{\mu} \sum_{i} \left[t_i^{\mu} - f\left(\sum_{j} w_{ij} h_j^{\mu} \right) \right]^2$$

$$= \tfrac{1}{2} \sum_{\mu} \sum_{i} \left[t_i^{\mu} - f\left(\sum_{j} w_{ij} \, f\left(\sum_{k=0}^{n} v_{jk} x_k^{\mu} \right) \right) \right]^2$$

Backpropagation

- **The weight correction is given by :**

$$\Delta w_{mn} = \eta \sum_{\nu} \delta_m^{\mu} x_n^{\mu}$$

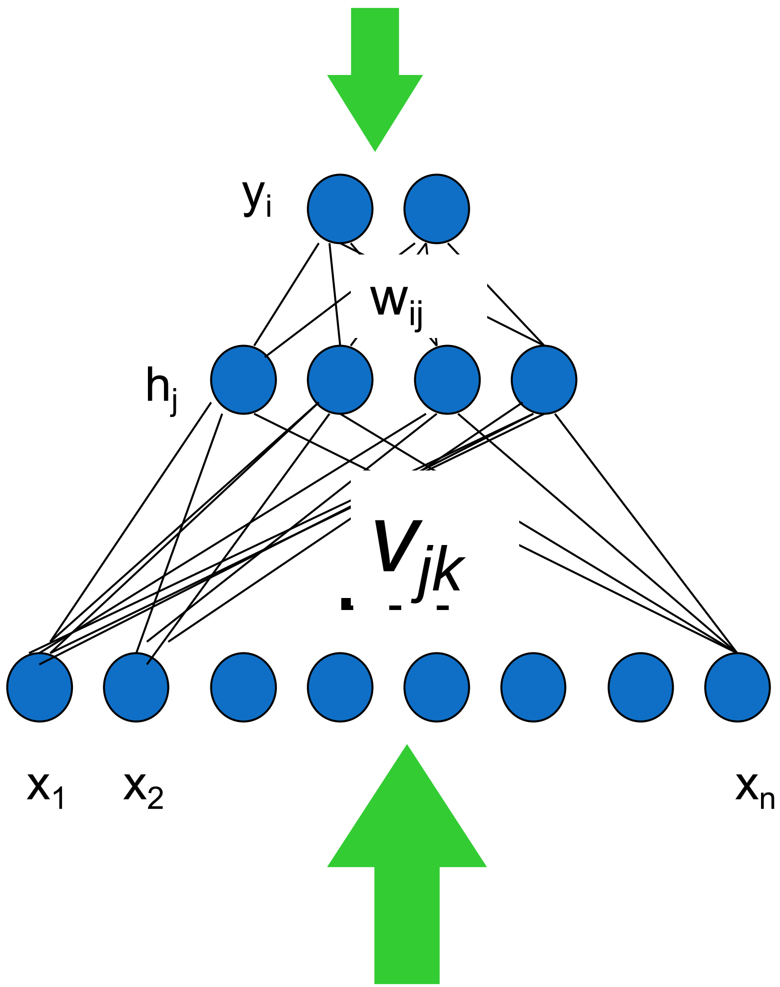
Where

$$\delta_m^{\mu} = \left(t_m^{\mu} - y_m^{\mu}\right) f'\left(A_m^{\mu}\right)$$ If *m* is the output layer

or

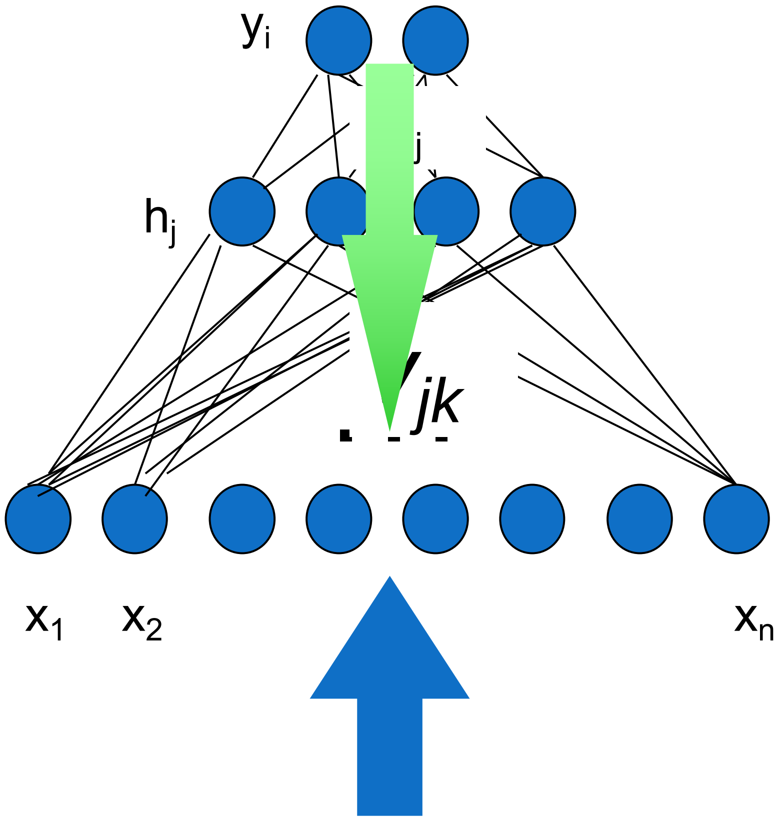
$$\delta_m^{\mu} = f'\left(A_m^{\mu}\right) \sum_{s} w_{sm} \delta_s^{\mu}$$ If *m* is an hidden layer

Backpropagation



Dr.Yanmei Zheng

Backpropagation

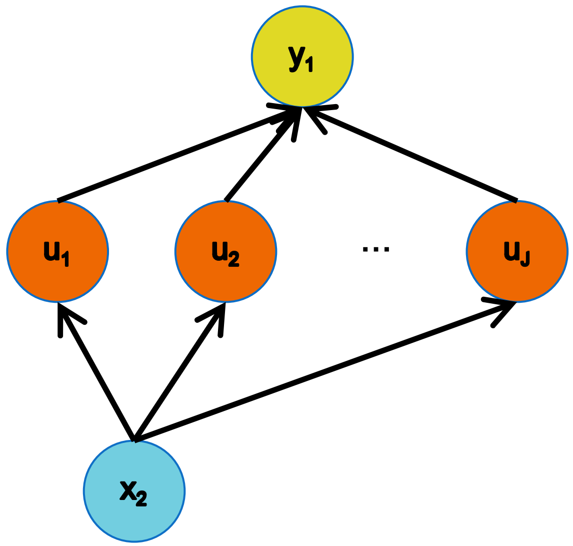


Chain Rule

Given $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j}\frac{du_j}{dx_k}, \quad \forall i, k$$

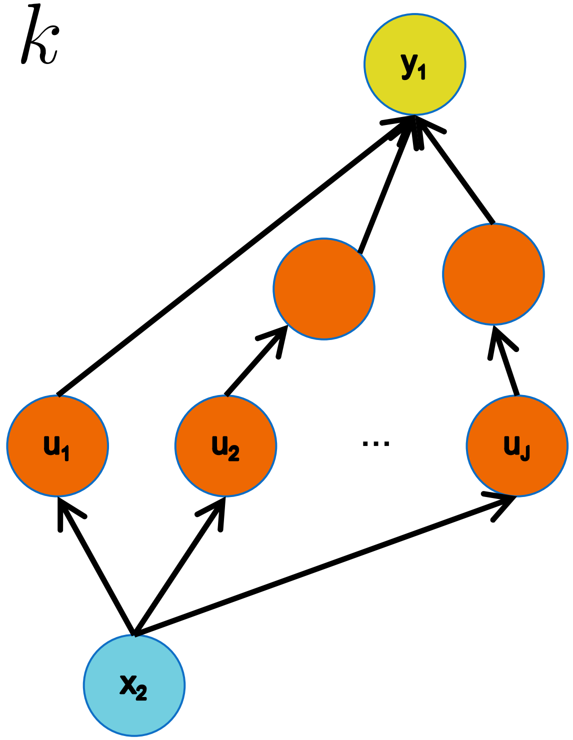


Chain Rule

Given: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j}\frac{du_j}{dx_k}, \qquad \forall i, k$$



Backpropagation is just repeated application of the **chain rule**

Chain Rule

- **Backpropagation:**

1. **Instantiate the computation as a directed acyclic graph, where each intermediate quantity is a node**
2. **At each node, store**

   (a) the quantity computed in the forward pass

   (b) the partial derivative of the goal with respect to that node's intermediate quantity.
3. **Initialize all partial derivatives to 0.**
4. **Visit each node in reverse topological order. At each node, add its contribution to the partial derivatives of its parents**

Backpropagation

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

$J = cos(u)$

$u = u_1 + u_2$

$u_1 = sin(t)$

$u_2 = 3t$

$t = x^2$

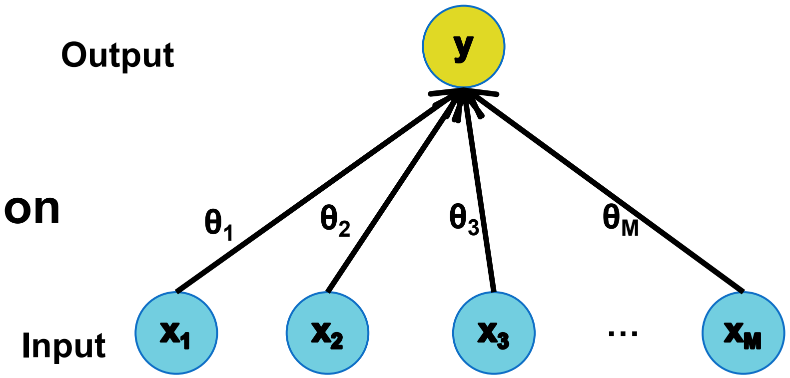
Backpropagation

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward	Backward
$J = cos(u)$	$\frac{dJ}{du}$ += $-sin(u)$
$u = u_1 + u_2$	$\frac{dJ}{du_1}$ += $\frac{dJ}{du}\frac{du}{du_1}$, $\frac{du}{du_1} = 1$ $\frac{dJ}{du_2}$ += $\frac{dJ}{du}\frac{du}{du_2}$, $\frac{du}{du_2} = 1$
$u_1 = sin(t)$	$\frac{dJ}{dt}$ += $\frac{dJ}{du_1}\frac{du_1}{dt}$, $\frac{du_1}{dt} = \cos(t)$
$u_2 = 3t$	$\frac{dJ}{dt}$ += $\frac{dJ}{du_2}\frac{du_2}{dt}$, $\frac{du_2}{dt} = 3$
$t = x^2$	$\frac{dJ}{dx}$ += $\frac{dJ}{dt}\frac{dt}{dx}$, $\frac{dt}{dx} = 2x$

Backpropagation

Case 1:
Logistic Regression



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backward

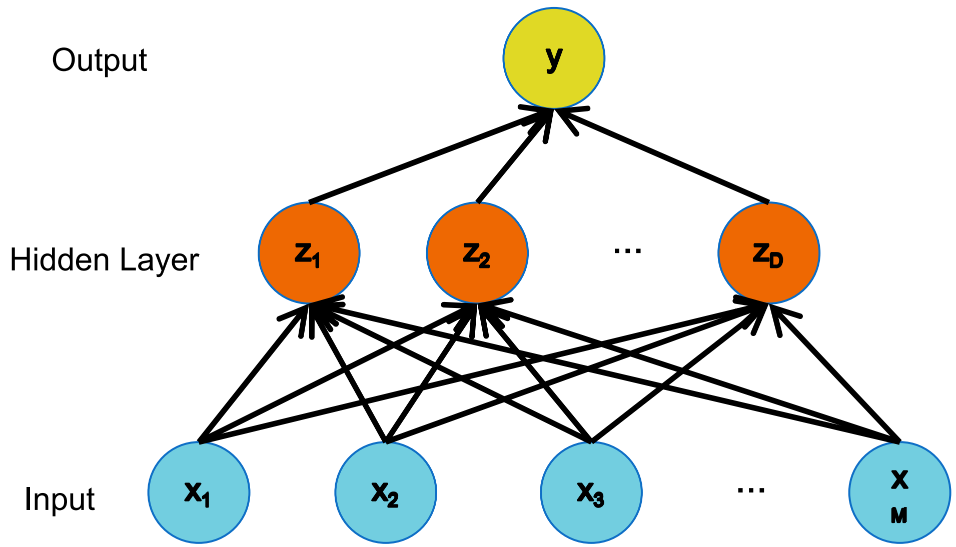
$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{da} = \frac{dJ}{dy}\frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

$$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j$$

$$\frac{dJ}{dx_j} = \frac{dJ}{da}\frac{da}{dx_j}, \quad \frac{da}{dx_j} = \theta_j$$

Backpropagation



(E) **Output (sigmoid)**
$$y = \frac{1}{1+\exp(-b)}$$

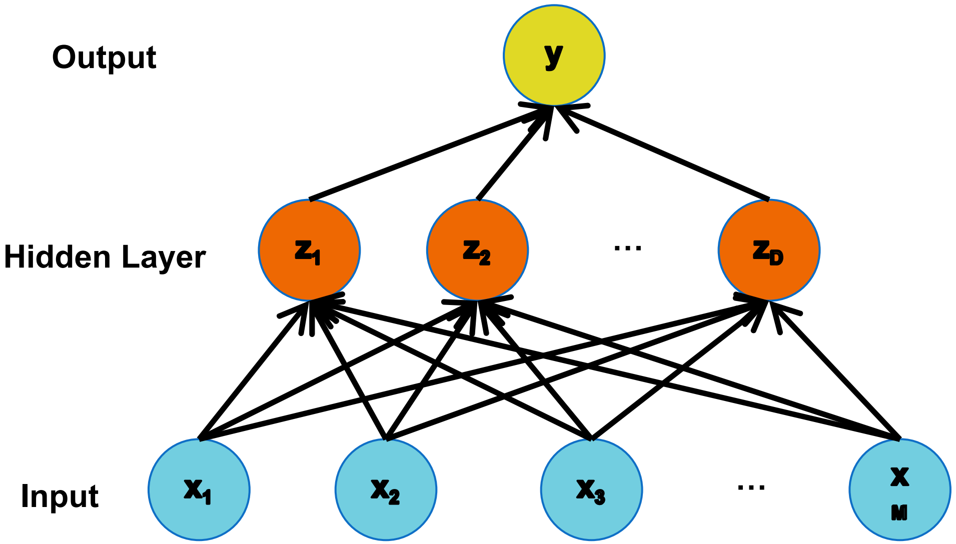
(D) **Output (linear)**
$$b = \sum_{j=0}^{D} \beta_j z_j$$

(C) **Hidden (sigmoid)**
$$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$$

(B) **Hidden (linear)**
$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$$

(A) **Input**
Given $x_i, \ \forall i$

Backpropagation



(F) Loss
$J = \frac{1}{2}(y - y^*)^2$

(E) Output (sigmoid)
$y = \frac{1}{1+\exp(-b)}$

(D) Output (linear)
$b = \sum_{j=0}^{D} \beta_j z_j$

(C) Hidden (sigmoid)
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

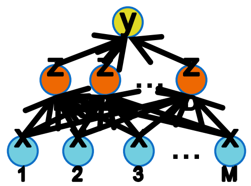
(B) Hidden (linear)
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) Input
Given $x_i, \ \forall i$

Dr.Yanmei Zheng

Backpropagation

Case 2: Neural Network



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$$

Chain Rule

Backpropagation:

1. **Instantiate the computation as a directed acyclic graph**, where each intermediate quantity is a node
2. At each node, store (a) the quantity computed in the forward pass and (b) the **partial derivative** of the goal with respect to that node's intermediate quantity.
3. **Initialize** all partial derivatives to 0.
4. Visit each node in **reverse topological order**. At each node, add its contribution to the partial derivatives of its parents

Chain Rule

Backpropagation:

1. **Instantiate the computation as a directed acyclic graph**, where each node represents a Tensor.
2. At each node, store (a) the quantity computed in the forward pass and (b) the **partial derivatives** of the goal with respect to that node's Tensor.
3. **Initialize** all partial derivatives to 0.
4. Visit each node in **reverse topological order**. At each node, add its contribution to the partial derivatives of its parents

Backpropagation

Case 2:

	Forward	Backward
Module 5	$J = y^* \log y + (1-y^*)\log(1-y)$	$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1-y^*)}{y-1}$
Module 4	$y = \frac{1}{1+\exp(-b)}$	$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b)+1)^2}$
Module 3	$b = \sum_{j=0}^{D} \beta_j z_j$	$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$ $\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$
Module 2	$z_j = \frac{1}{1+\exp(-a_j)}$	$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j)+1)^2}$
Module 1	$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$	$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$ $\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$

Summary: Biology and Neural Networks

- So many similarities
  1. Information is contained in synaptic connections
  2. Network learns to perform specific functions
  3. Network generalizes to new inputs
- But NNs are woefully inadequate compared with biology
  1. Simplistic model of neuron and synapse, implausible learning rules
  2. Hard to train large networks
  3. Network construction (structure, learning rate etc.) is a heuristic art
- One obvious difference: Spike representation
  1. Recent models explore spikes and spike-timing dependent plasticity
- Other Recent Trends: Probabilistic approach
  1. NNs as Bayesian networks (allows principled derivation of dynamics, learning rules, and even structure of network)
  2. Not clear how neurons encode probabilities in spikes

Summary

1. **Neural Networks…**
   - ✓ provide a way of learning features
   - ✓ are highly nonlinear prediction functions
   - ✓ (can be) a highly parallel network of logistic regression classifiers
   - ✓ discover useful hidden representations of the input
2. **Backpropagation…**
   - ✓ provides an efficient way to compute gradients
   - ✓ is a special case of reverse-mode automatic differentiation

Thank you

End of

ARTIFICIAL NEURAL NETWORK

Dr.Yanmei Zheng