Introduction to Machine Learning (CSCI-UA 473): Fall 2021

Lecture 8: Neural Networks

Sumit Chopra
Courant Institute of Mathematical Sciences
Department of Radiology - Grossman School of Medicine
NYU

Slides derived from materials from Benjamin Peherstorfer, Kyunghyun Cho, Andrew Gordon Wilson

Lecture Outline

Motivation behind neural networks

The XOR problem

Multi-Layer Perceptron model

Backpropagation algorithm

Regularization and other stuff in NNs

History

Neural Nets have been there since ages

McCulloch and Pitts 1943: ANN circuit

Donald Hebb 1949: Neural pathways are strengthened each time they are used

Rosenblatt 1958: the perceptron algorithm

Minsky & Papert 1969: XOR problem

Fukushima 1975: first convolutional neural network architecture

Rumelhart, Hinton and Williams 1986: Back-propagation algorithm

There was an initial lull in their research and application. Suddenly there has been a surge in interest

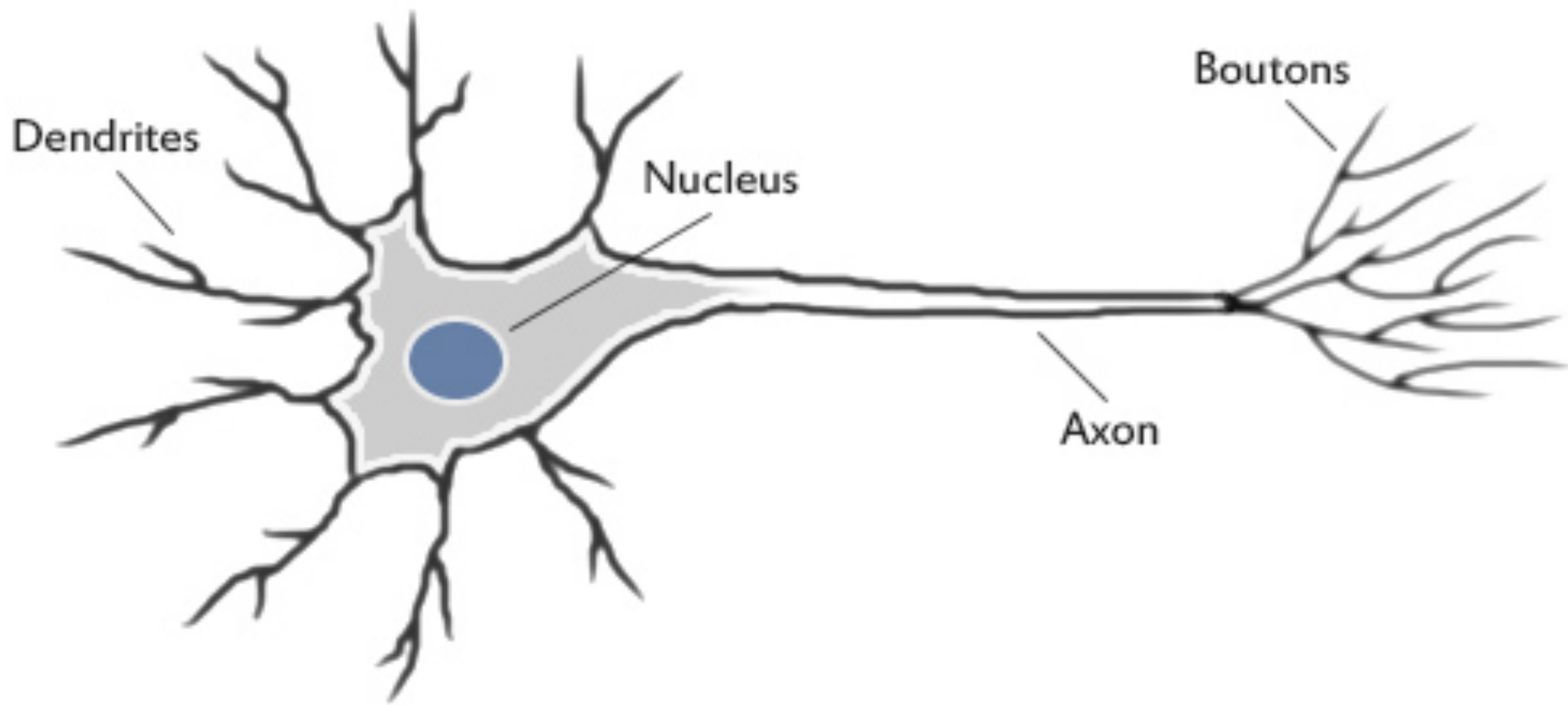
Lack of understanding

Lack of large scale datasets

Lack of hardware resources

Lack of software resources for easy model development

Neuron



Electrical signals are input to a neuron via dendrites

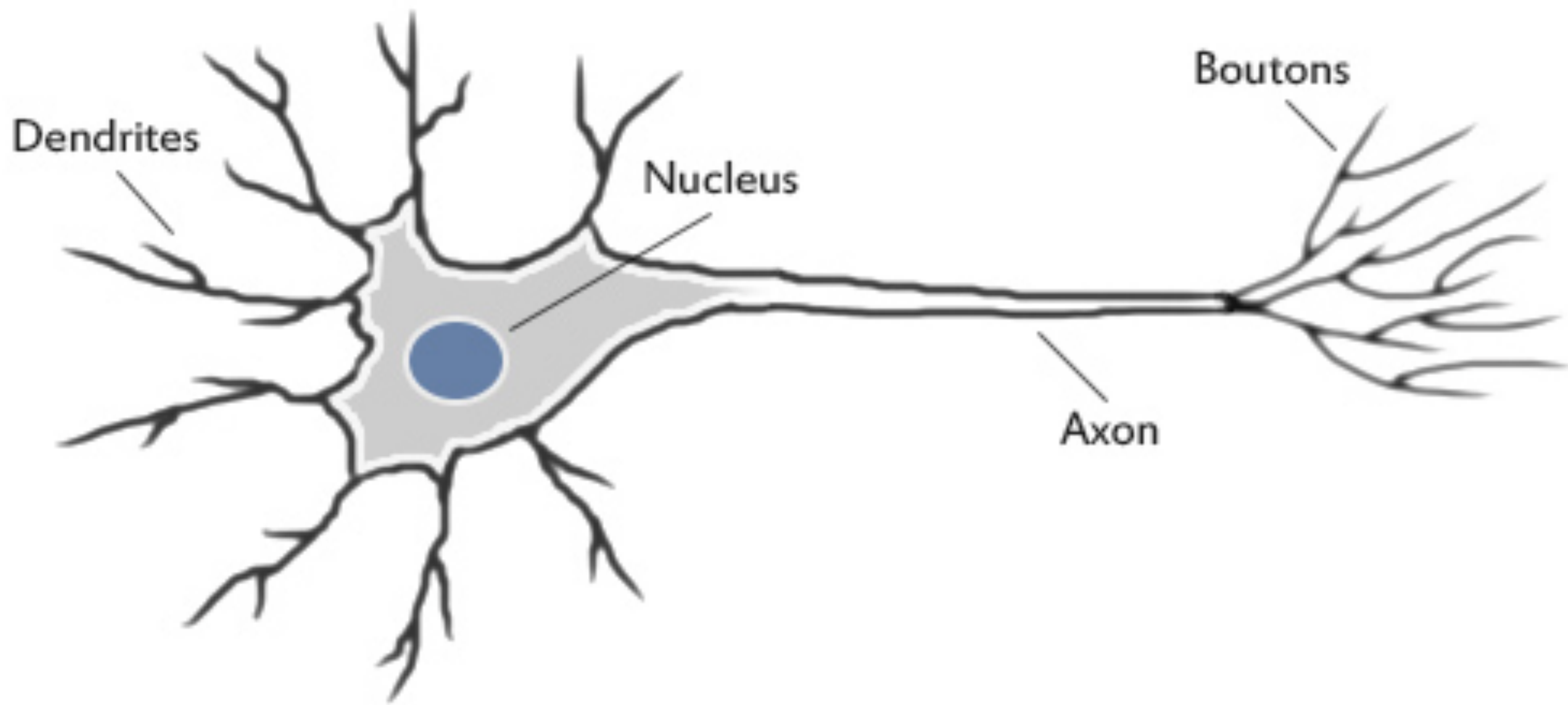
If the sum of these signals exceeds a threshold the neuron fires

Resulting electrical signal is passed to other neurons via Axons

The way these neuron are connected and the threshold of each neuron governs how we learn

About 100 billion neurons in human brain each with about 1000 synaptic connections

Neuron



Electrical signals are input to a neuron via dendrites

If th … ires

Re … ons

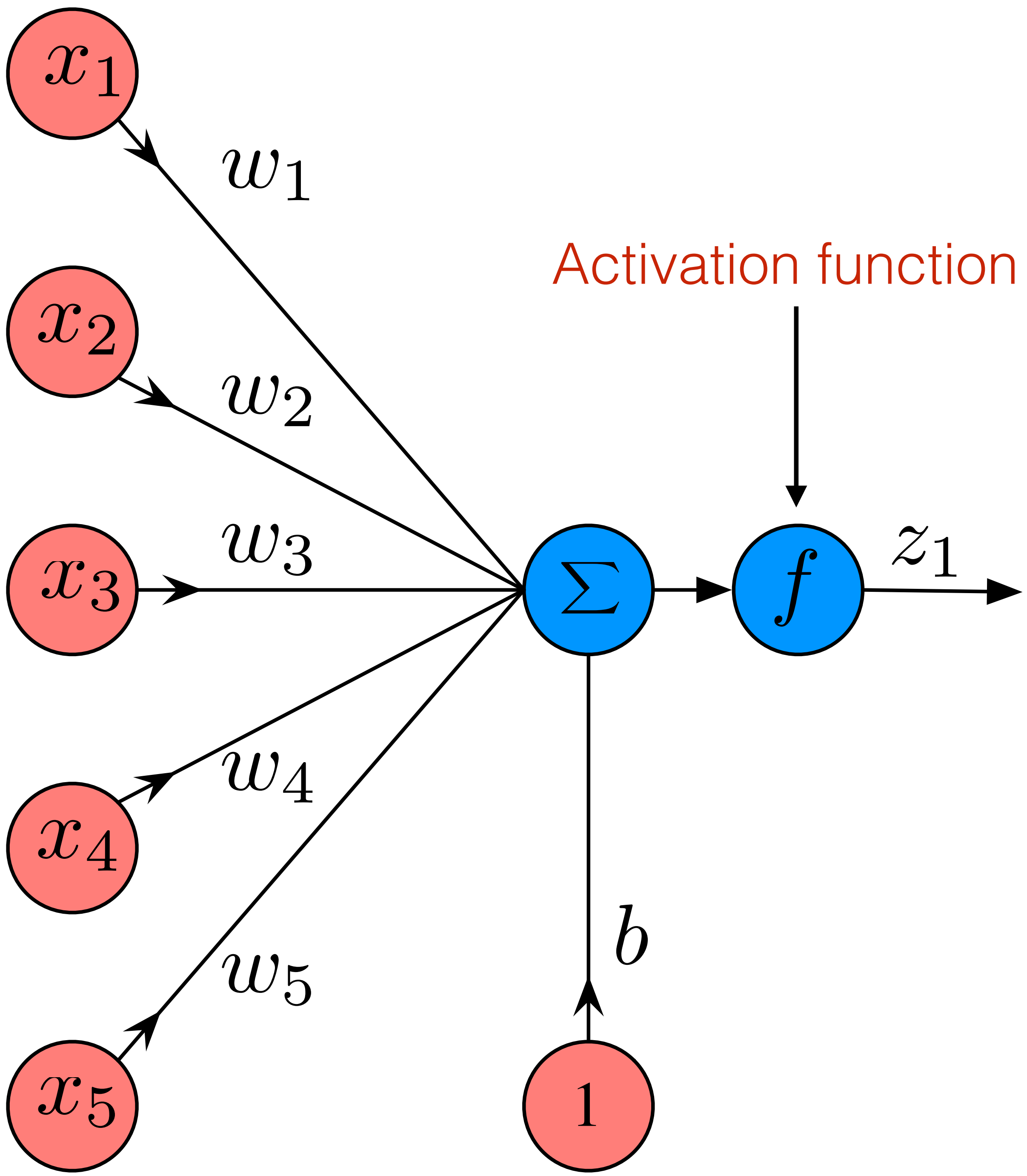
The way thes … governs how

Neurons in an artificial neural network are trying to mimic this behavior

About 100 billion neurons in human brain each with about 1000 synaptic connections

Artificial Neuron

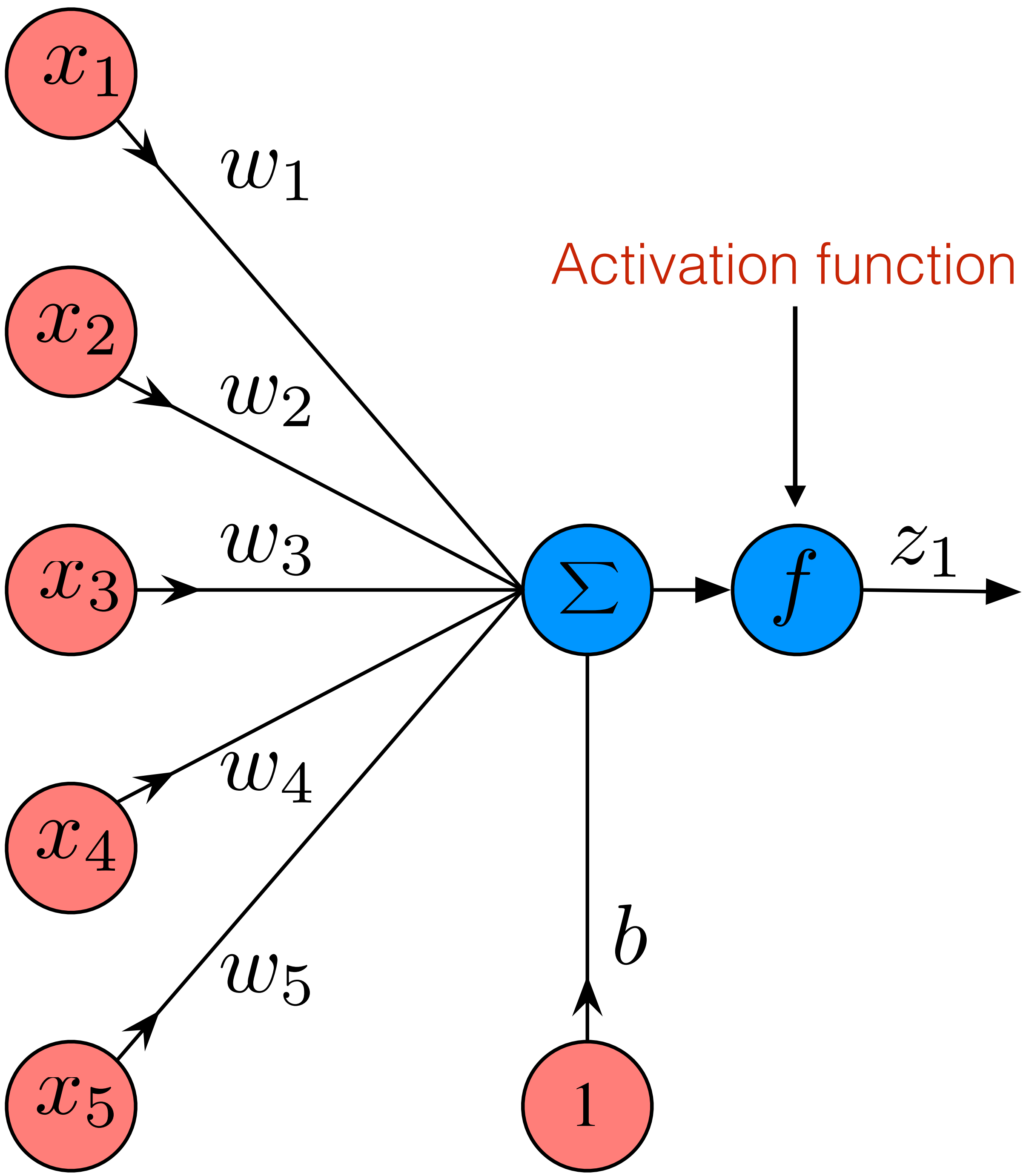
Simplification of a real neuron (McCulloch and Pitts)
Purpose is to develop understanding of what networks of simple units could do



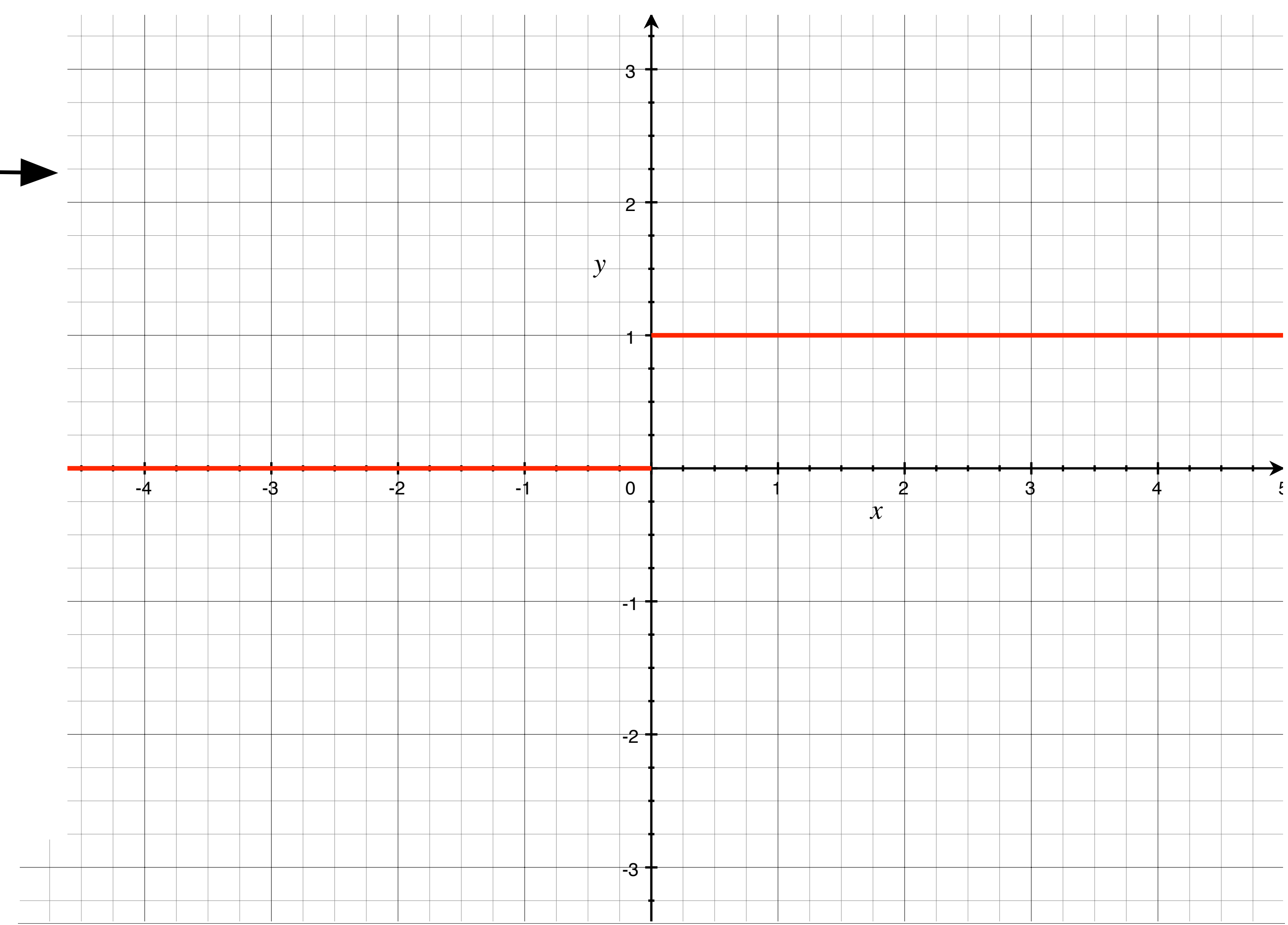
$$z_1 = f\left(\sum_{i=1}^{5} w_i x_i + b\right)$$

Artificial Neuron

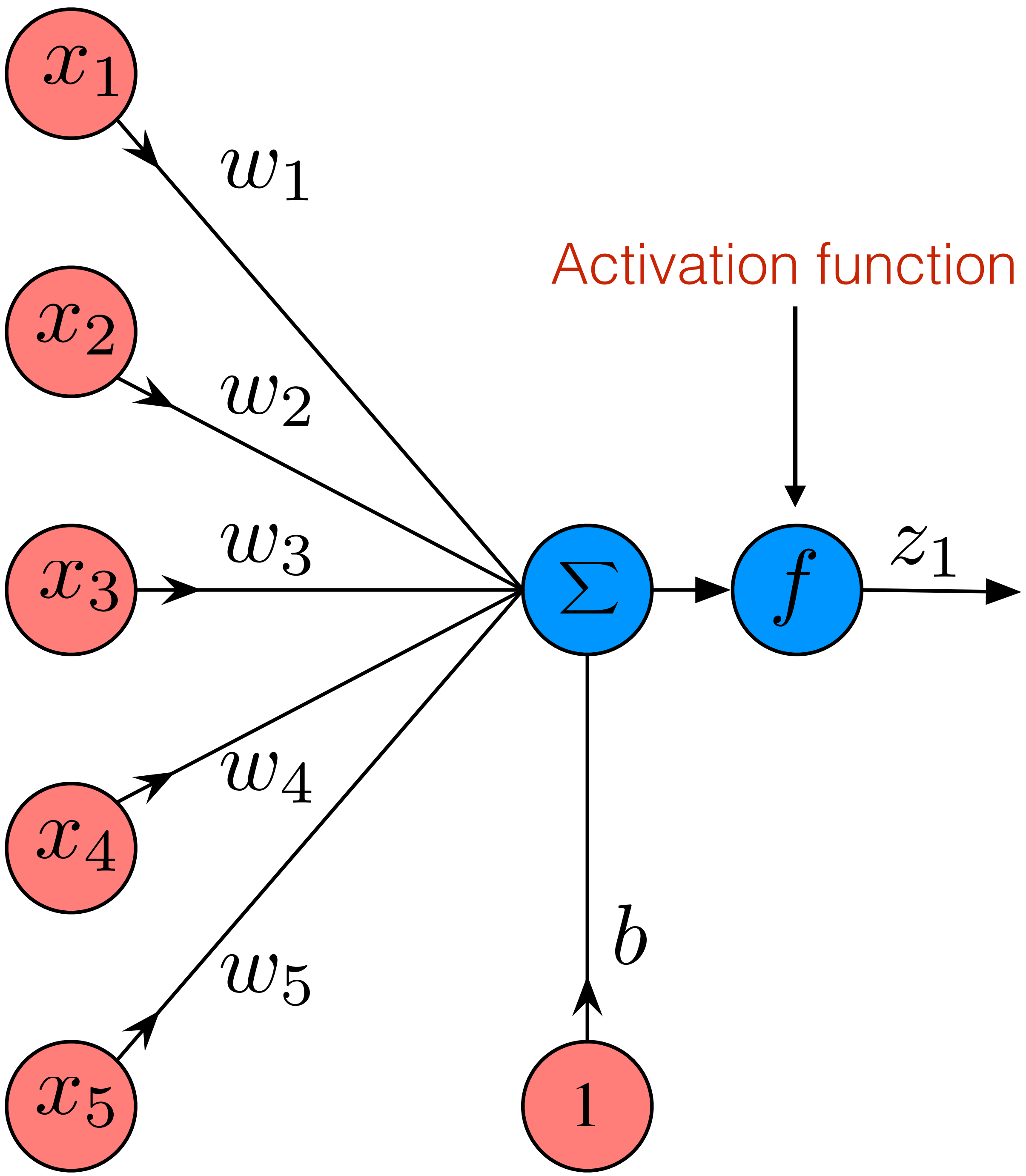
$$z_1 = f\left(\sum_{i=1}^{5} w_i x_i + b\right)$$



Threshold function



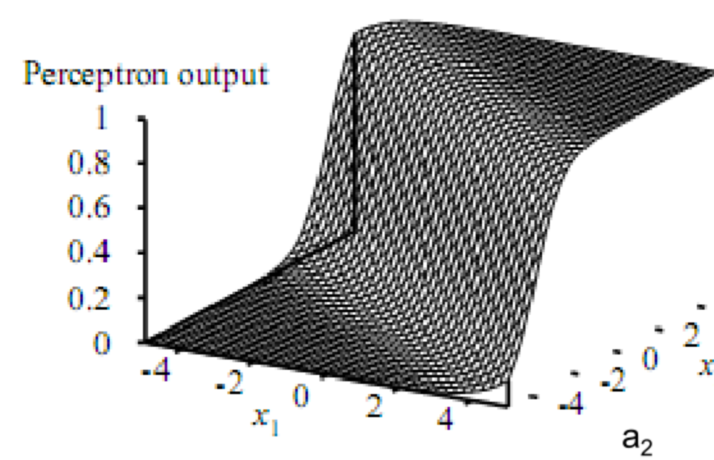
Artificial Neuron



$$z_1 = f\left(\sum_{i=1}^{5} w_i x_i + b\right)$$

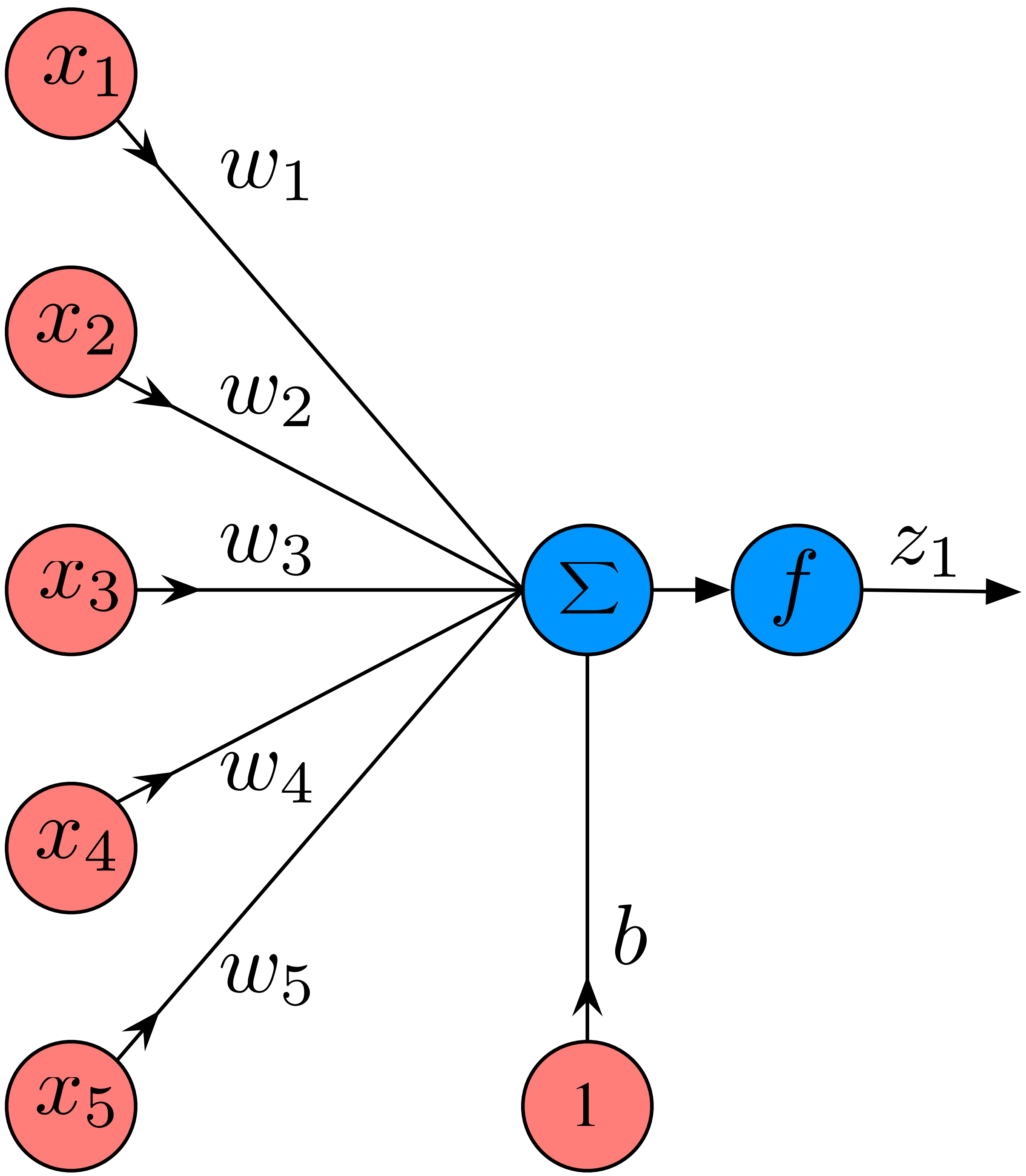
Weights and the biases inform position and slope of the cliff

Learning refers to modifying the weights and biases to get the right position and slope of the cliff

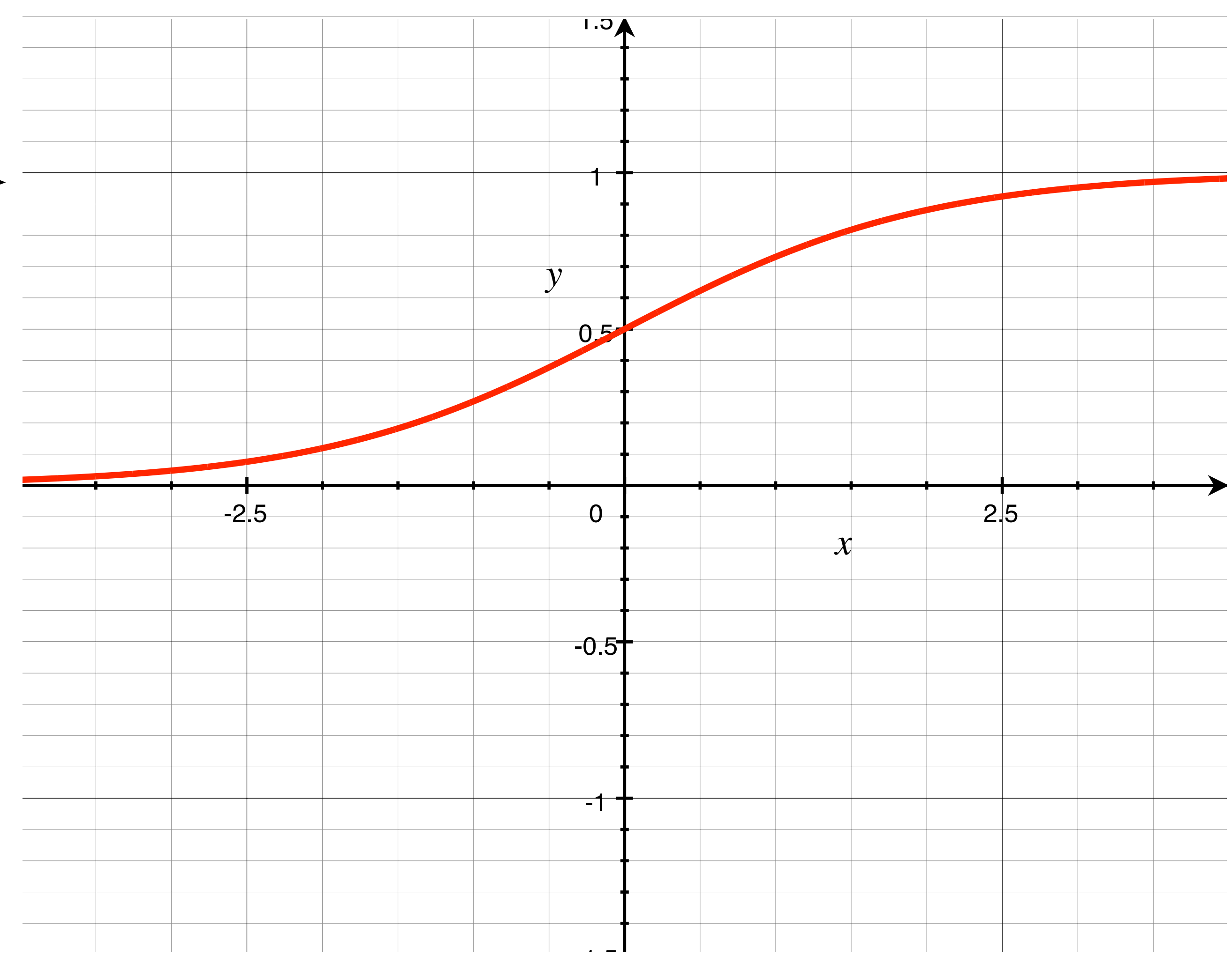


Artificial Neuron

$$z_1 = f\left(\sum_{i=1}^{5} w_i x_i + b\right)$$

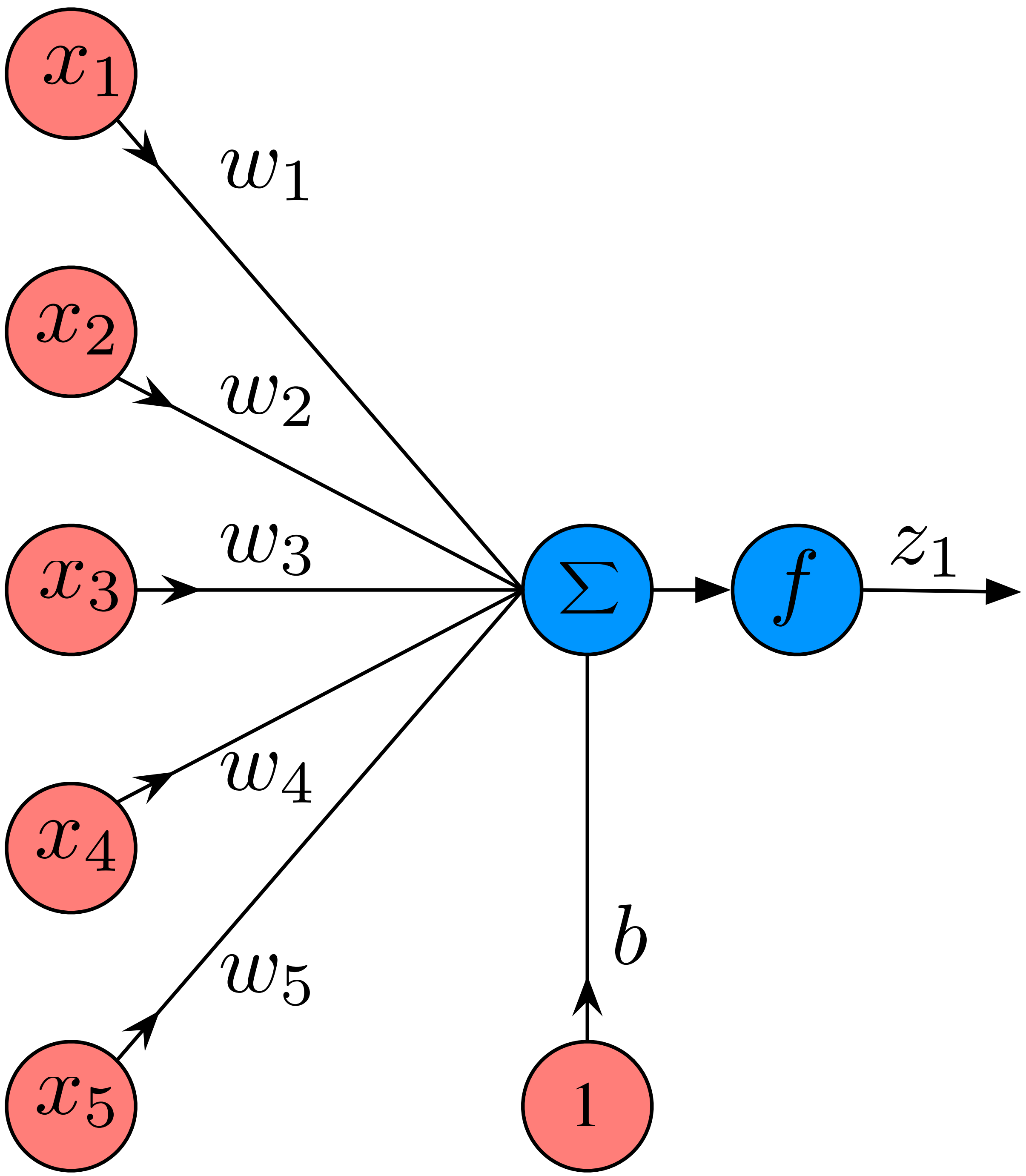


Sigmoid

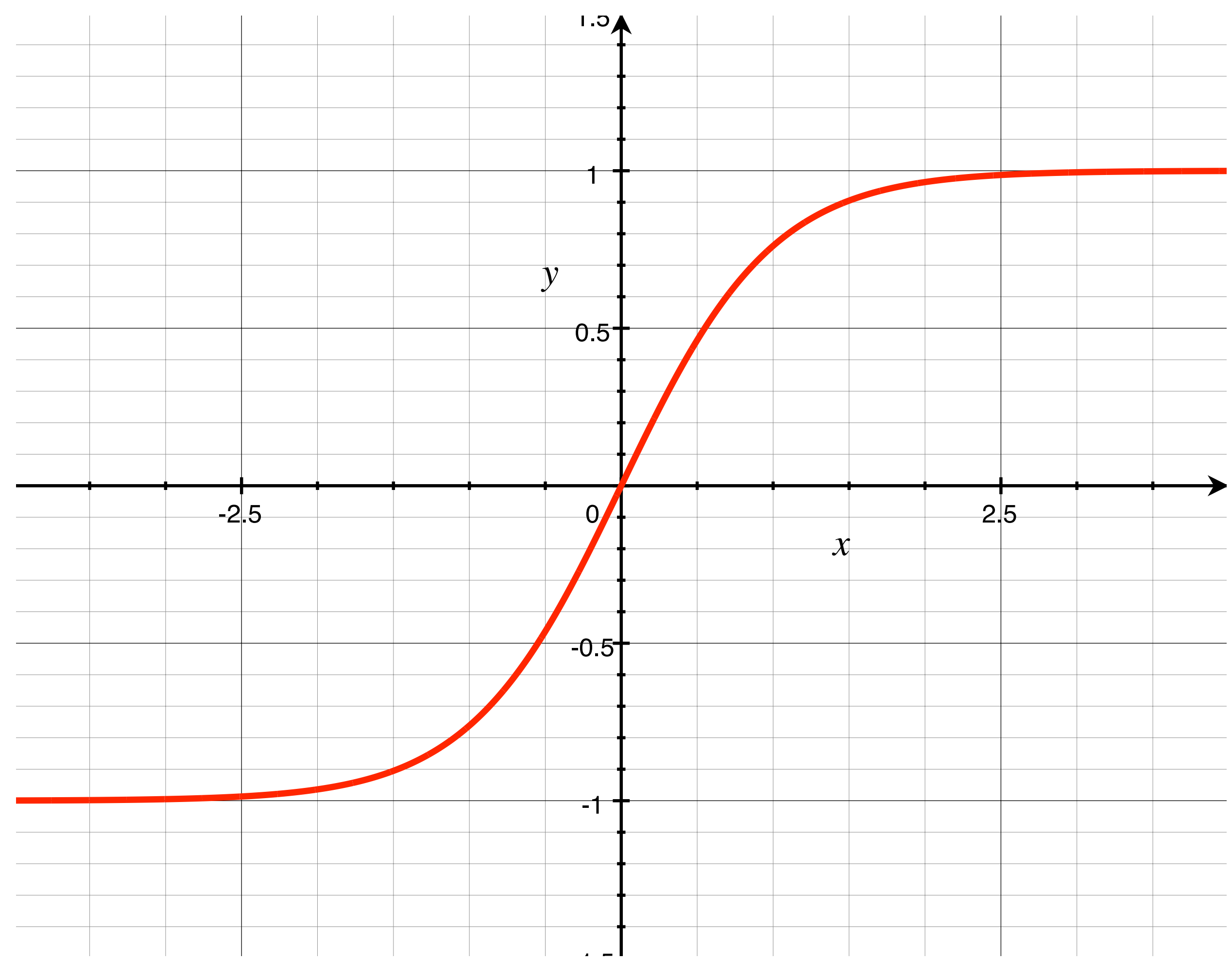


Artificial Neuron

$$z_1 = f\left(\sum_{i=1}^{5} w_i x_i + b\right)$$

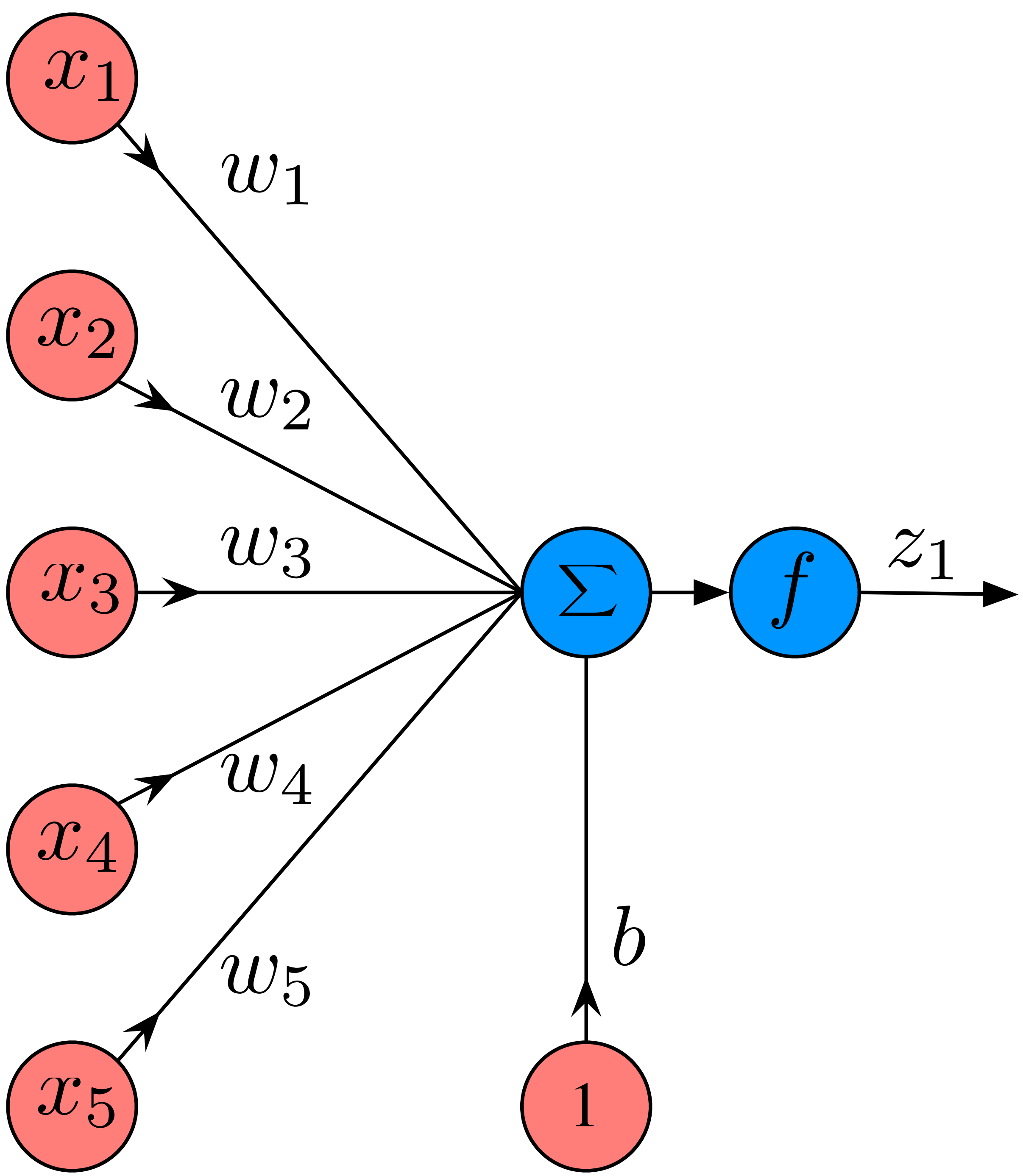


Tanh

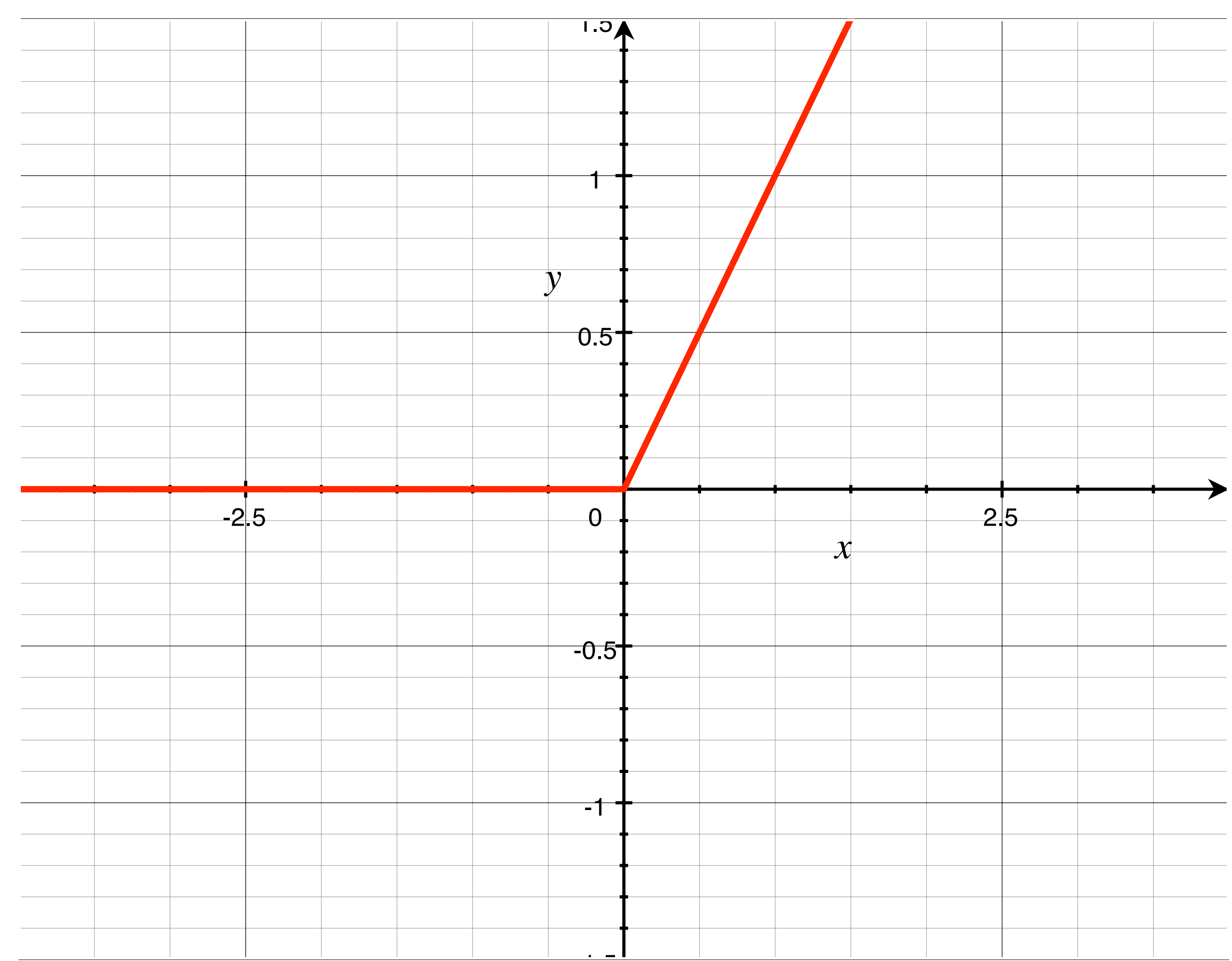


Artificial Neuron

$$z_1 = f\left(\sum_{i=1}^{5} w_i x_i + b\right)$$

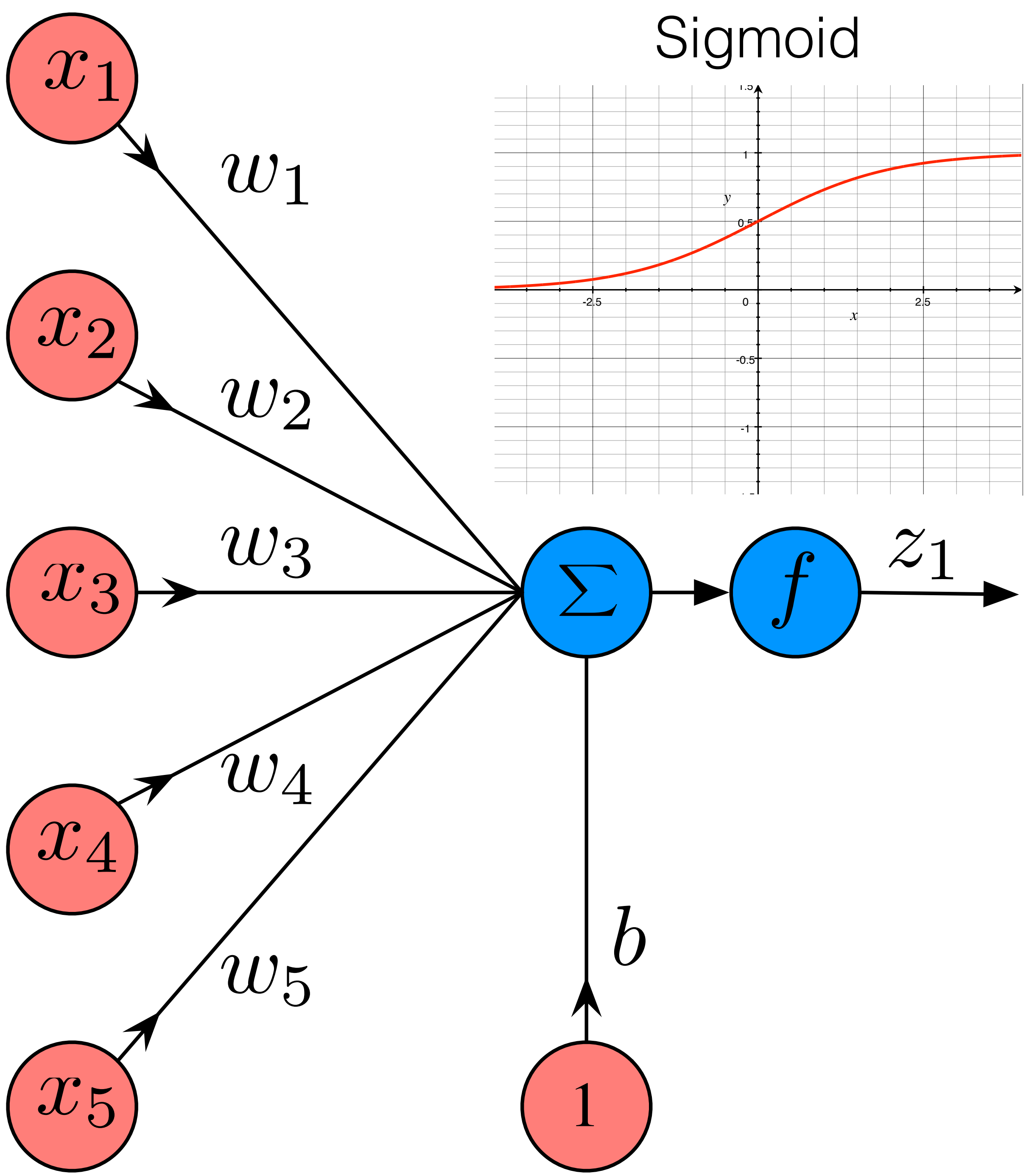


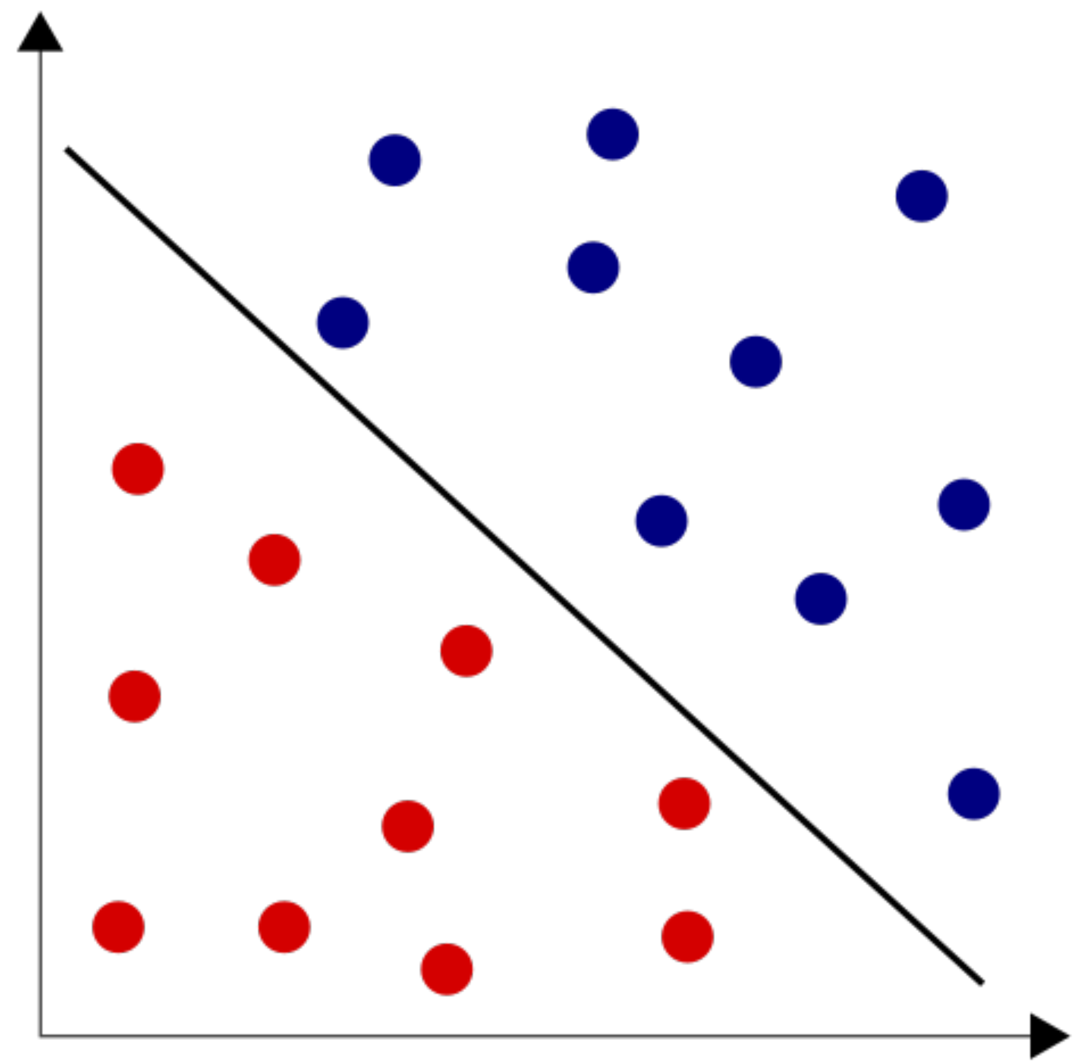
ReLU



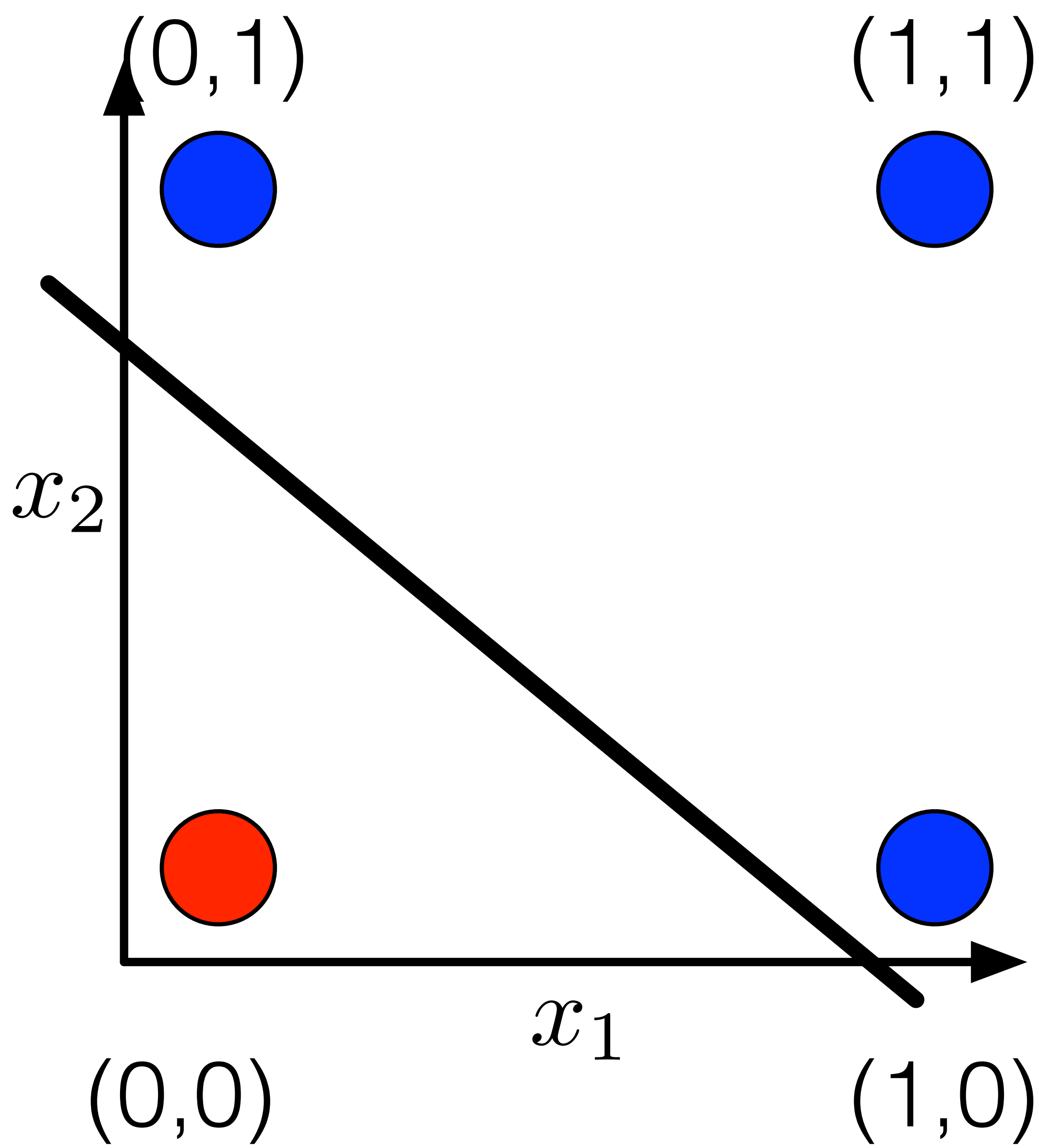
Logistic Regression

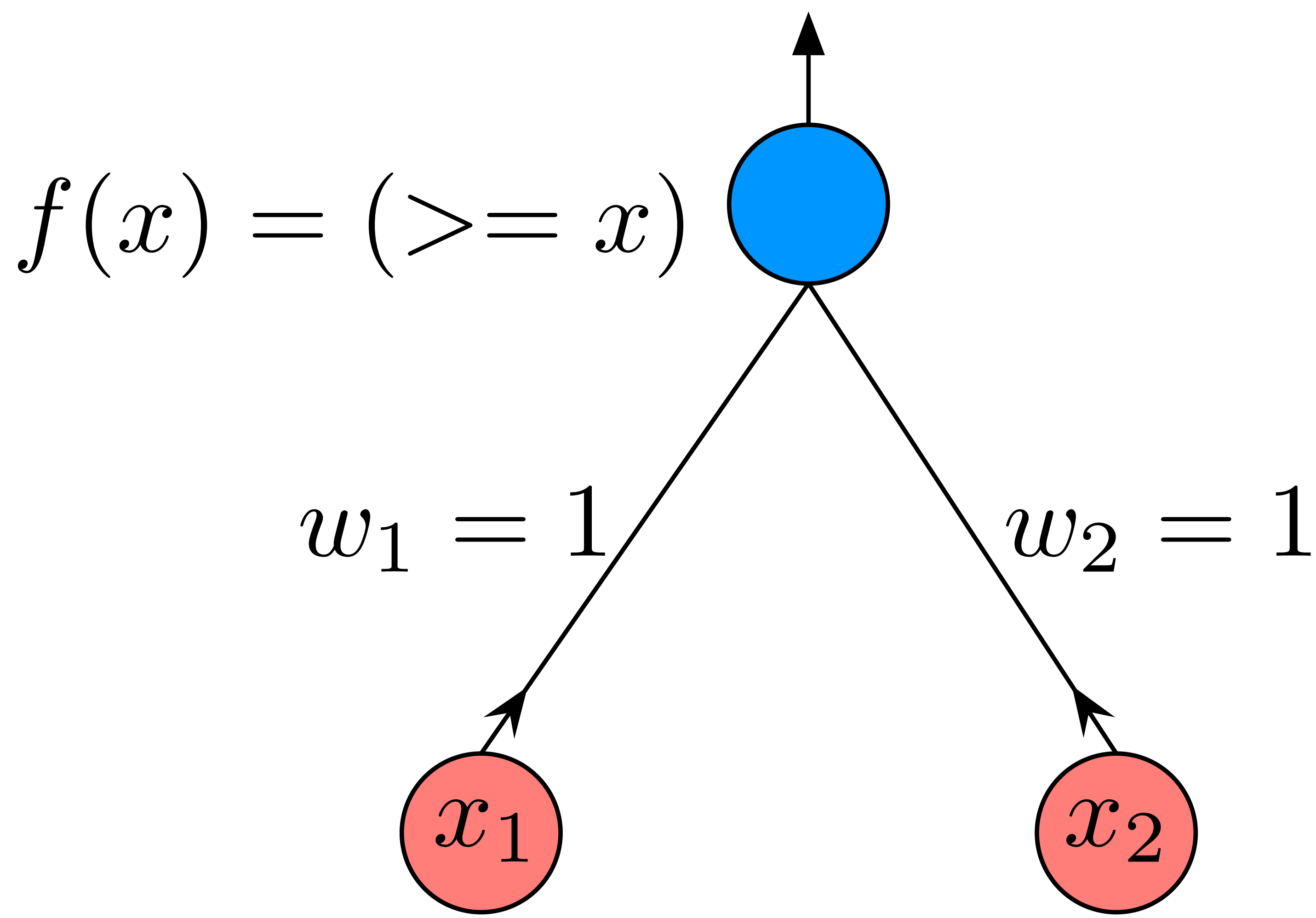
$$z_1 = f\left(\sum_{i=1}^{5} w_i x_i + b\right)$$



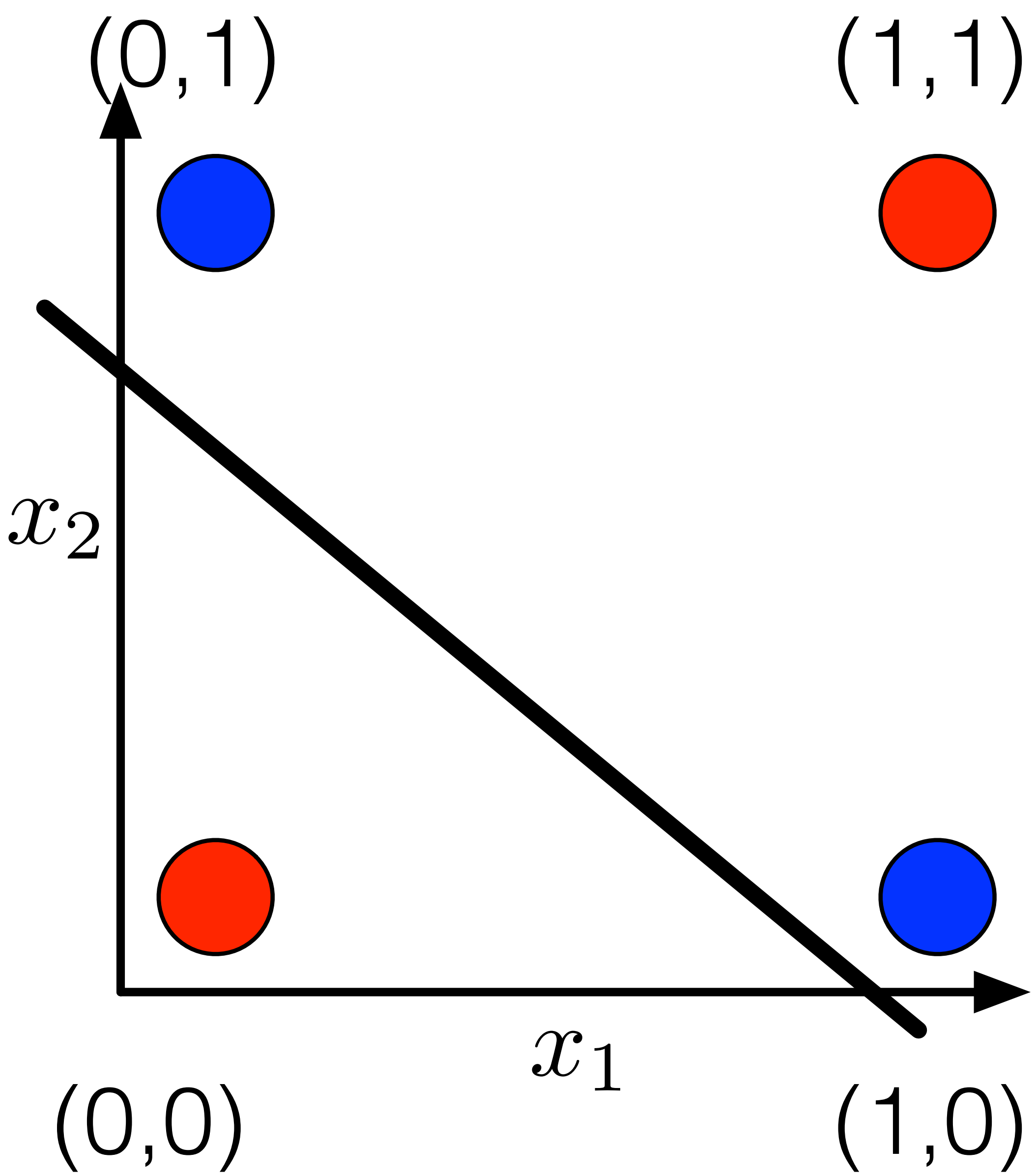


OR vs XOR

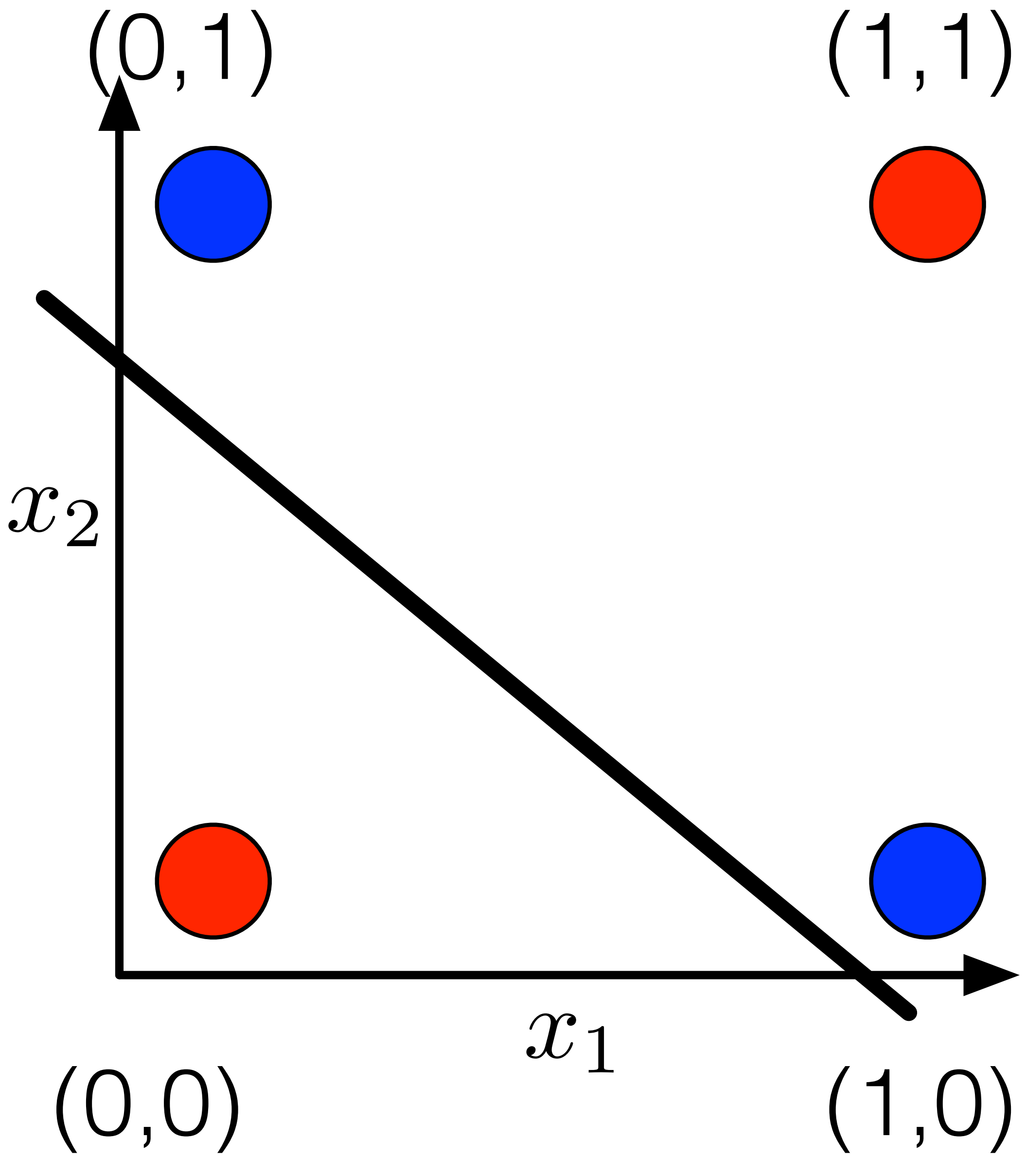


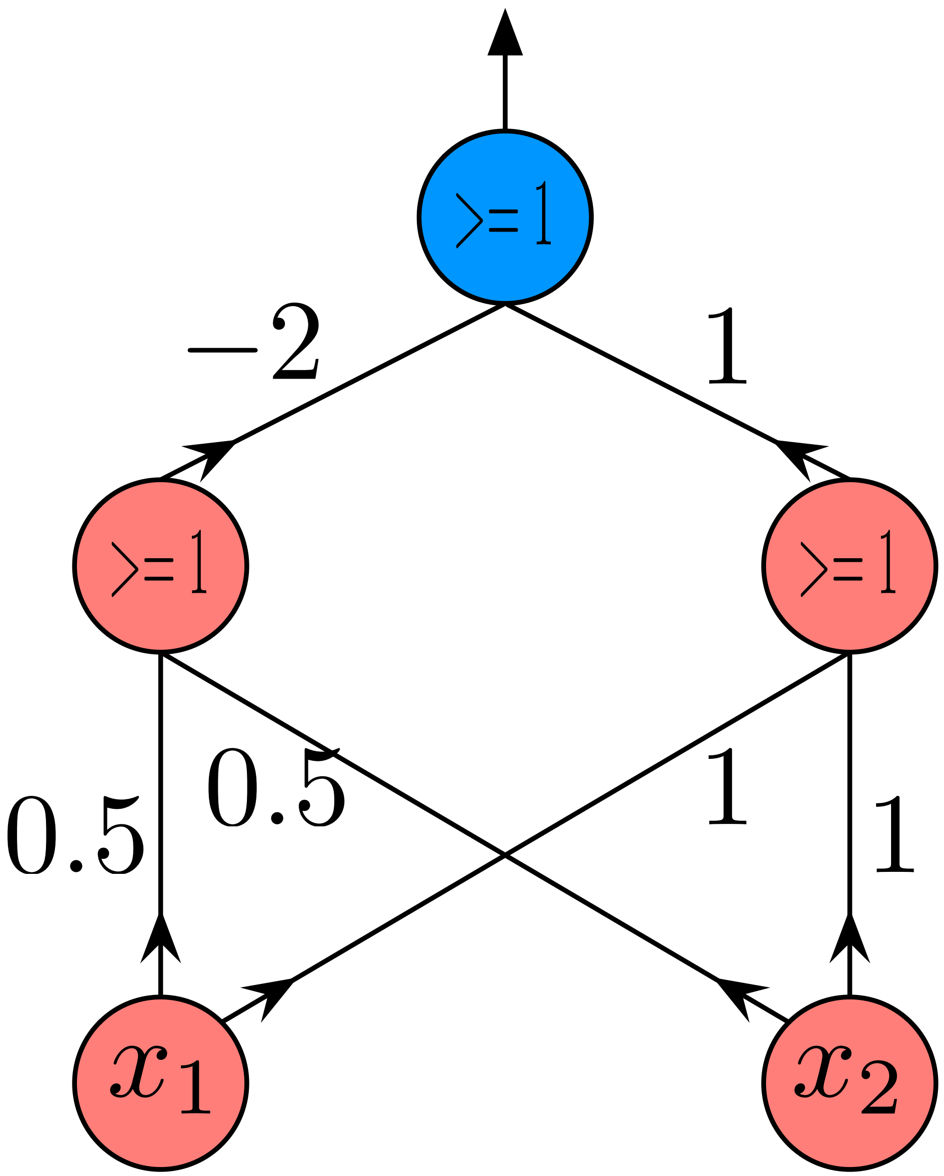


OR vs XOR

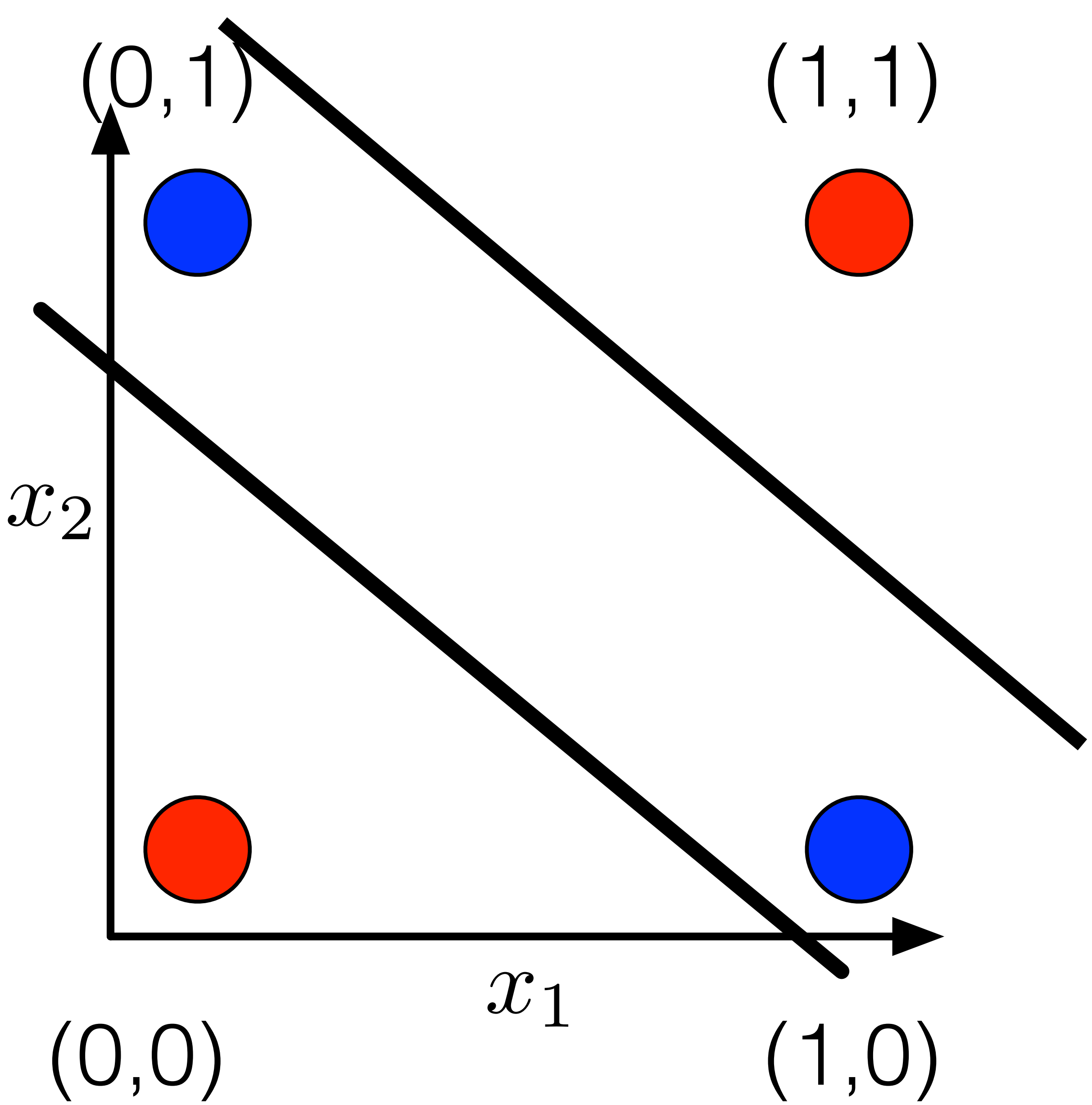


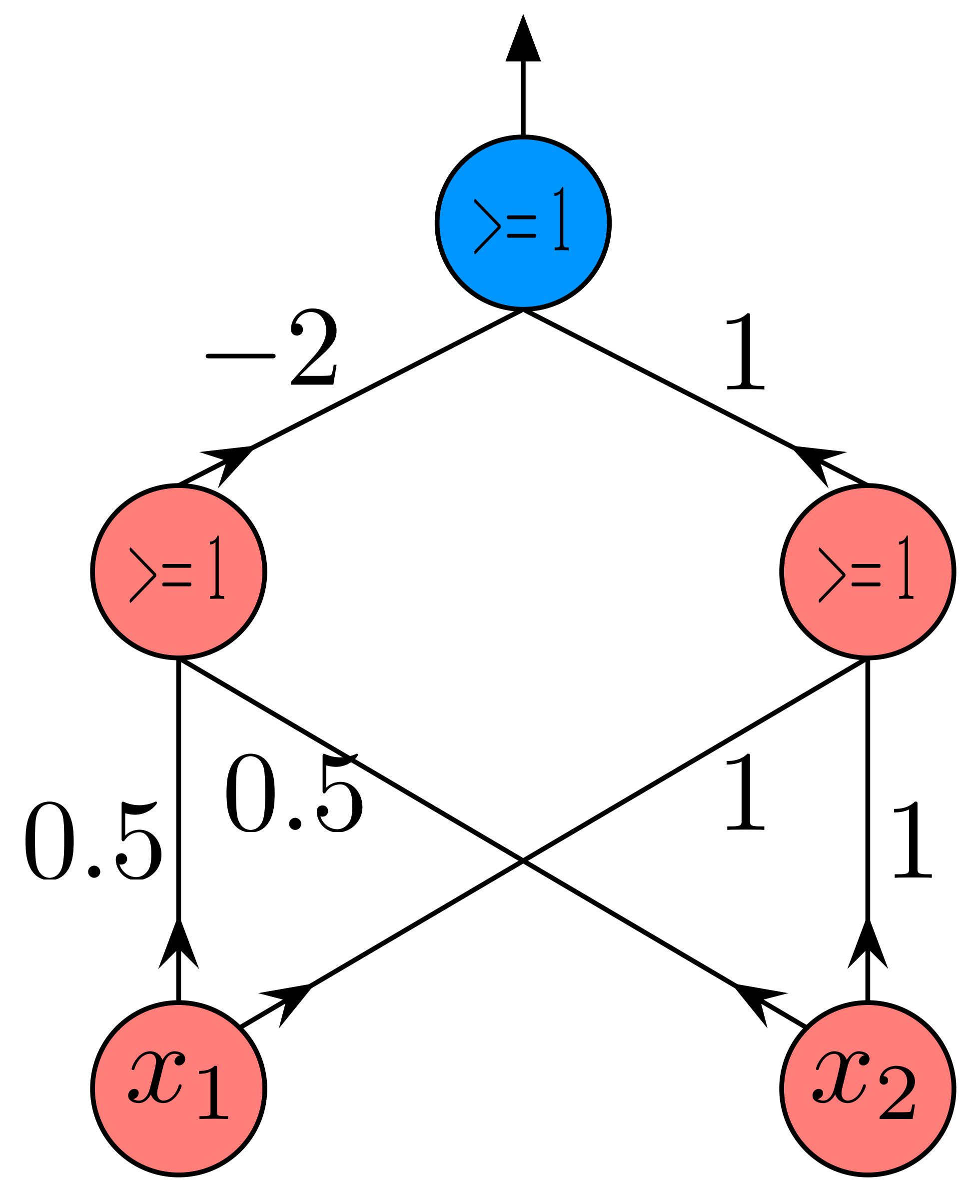
OR vs XOR



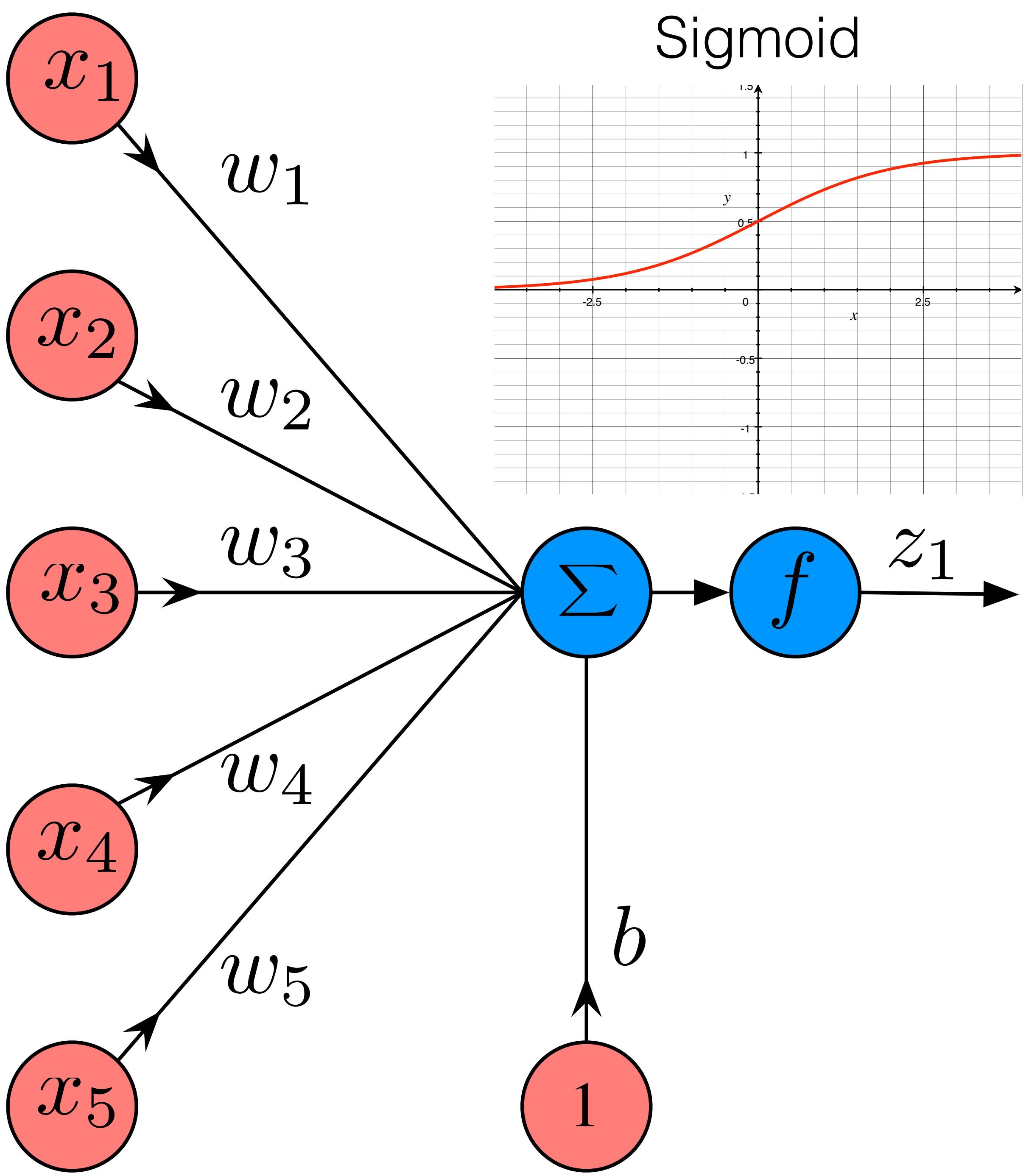


OR vs XOR

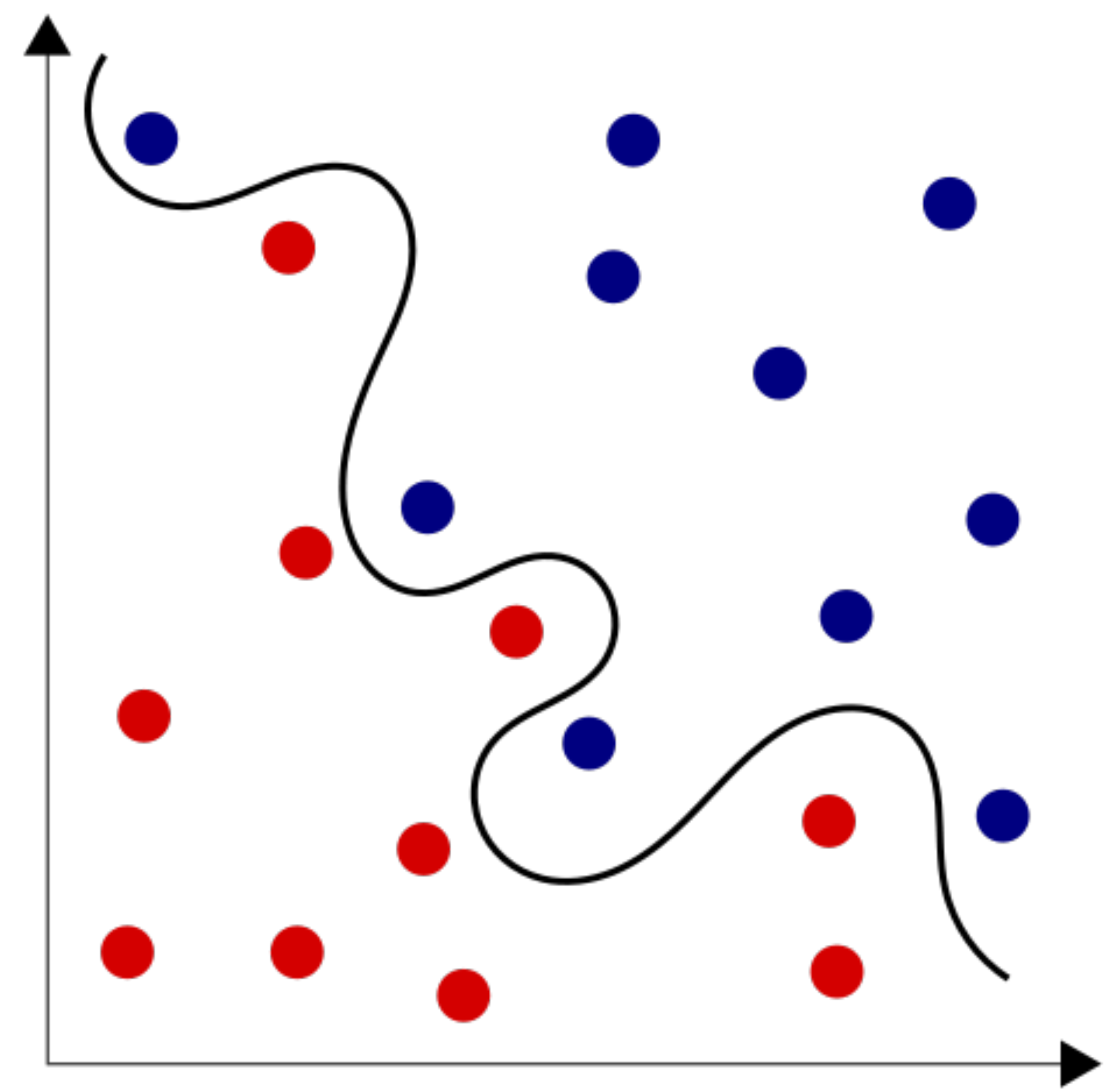




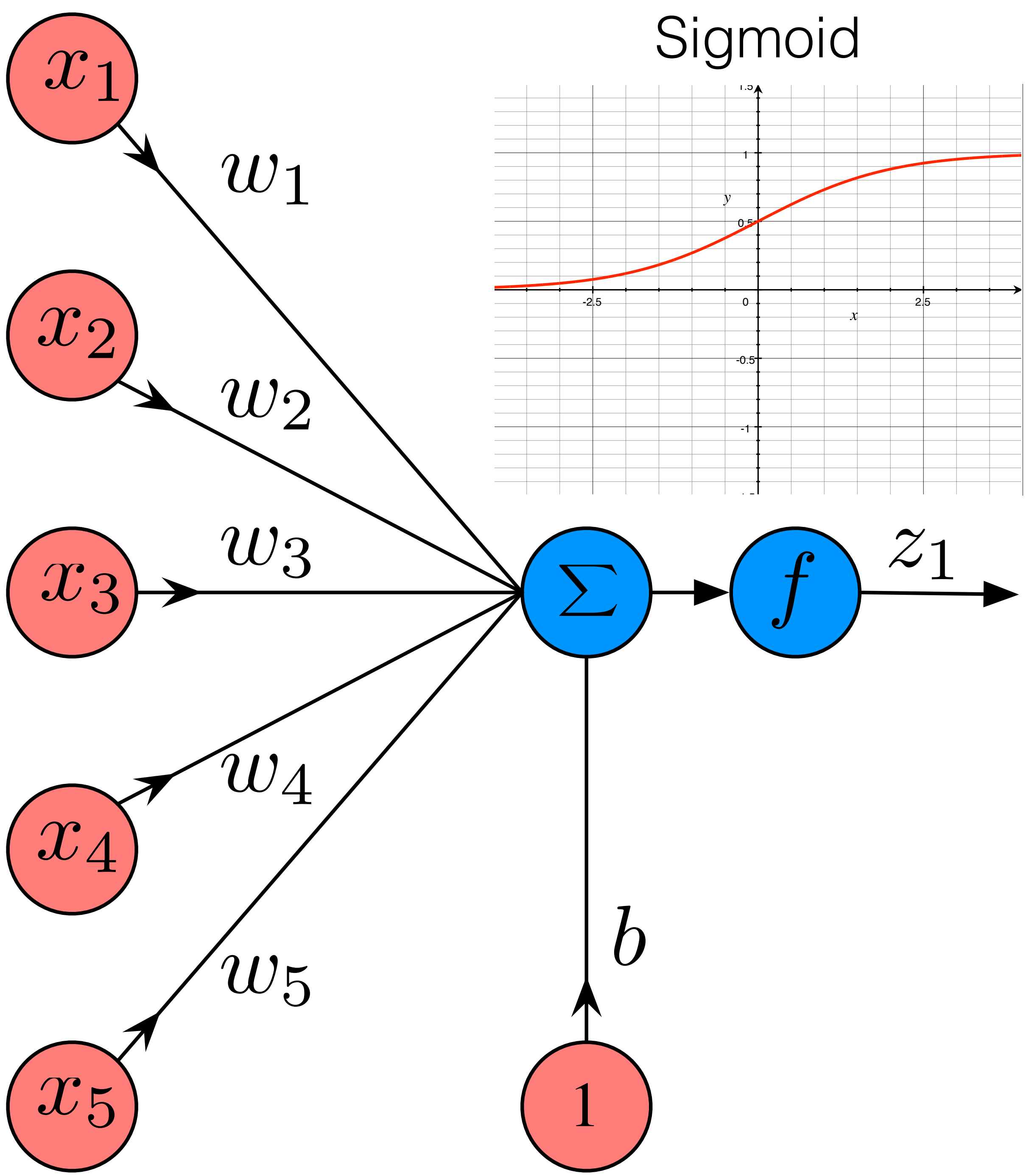
Logistic Regression



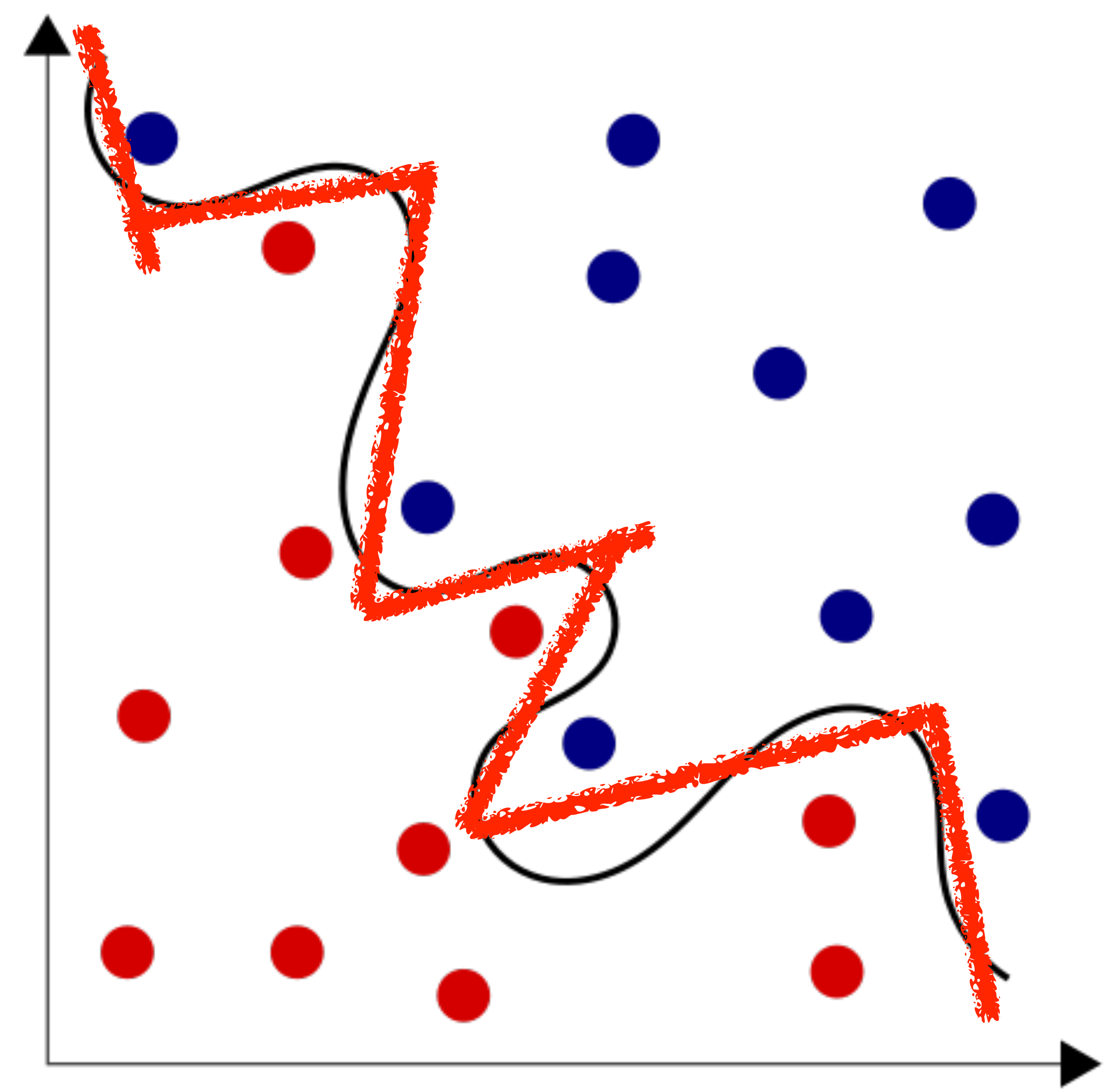
$$z_1 = f\left(\sum_{i=1}^{5} w_i x_i + b\right)$$



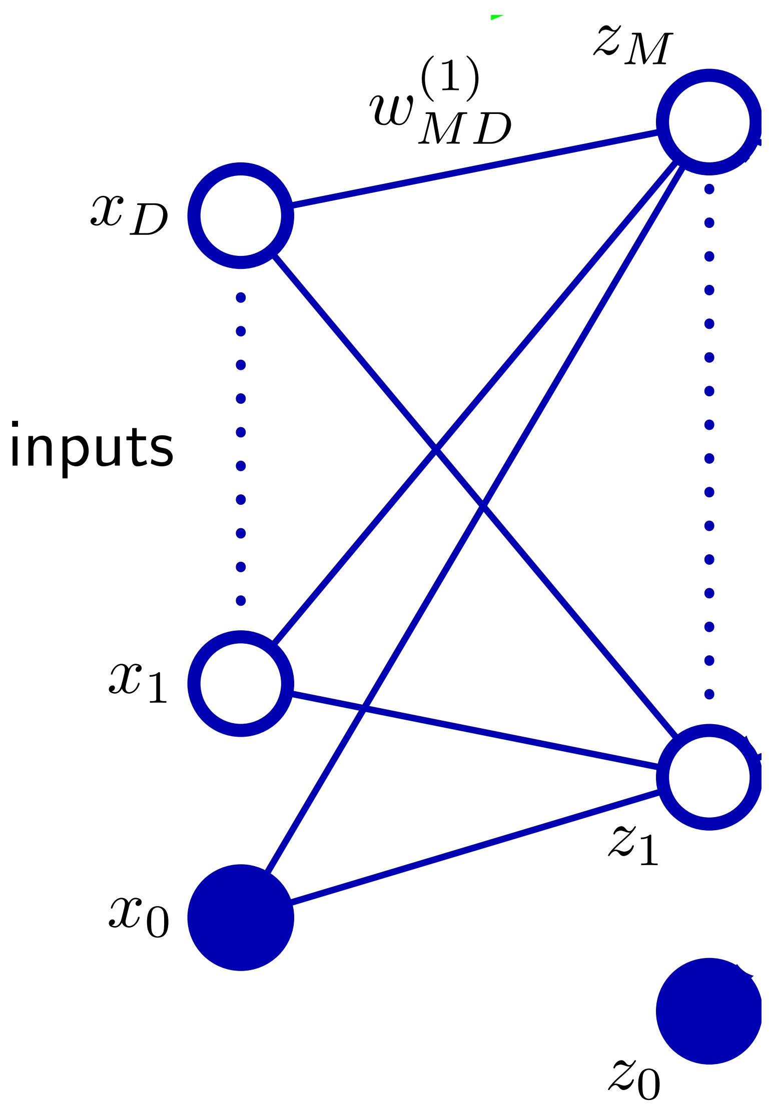
Multi-Layer Perceptron (Neural Network)



$$z_1 = f(\sum_{i=1}^{5} w_i x_i + b)$$



Two-Layer Neural Network



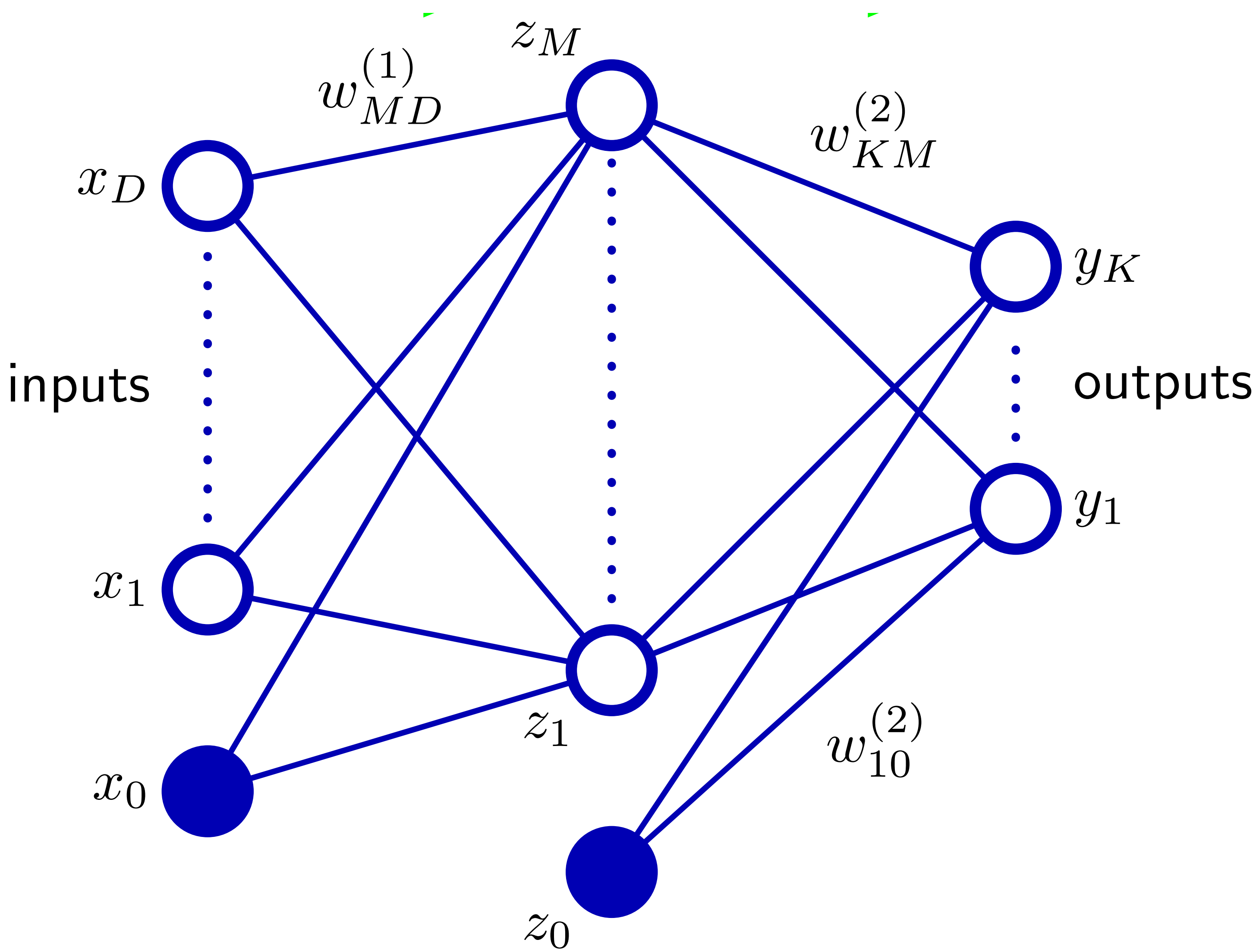
Layer 1

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

$$z_j = h(a_j)$$

[Slide adapted from Pattern Recognition and Machine Learning: Christopher Bishop]

Two-Layer Neural Network



Layer 1

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

$$z_j = h(a_j)$$

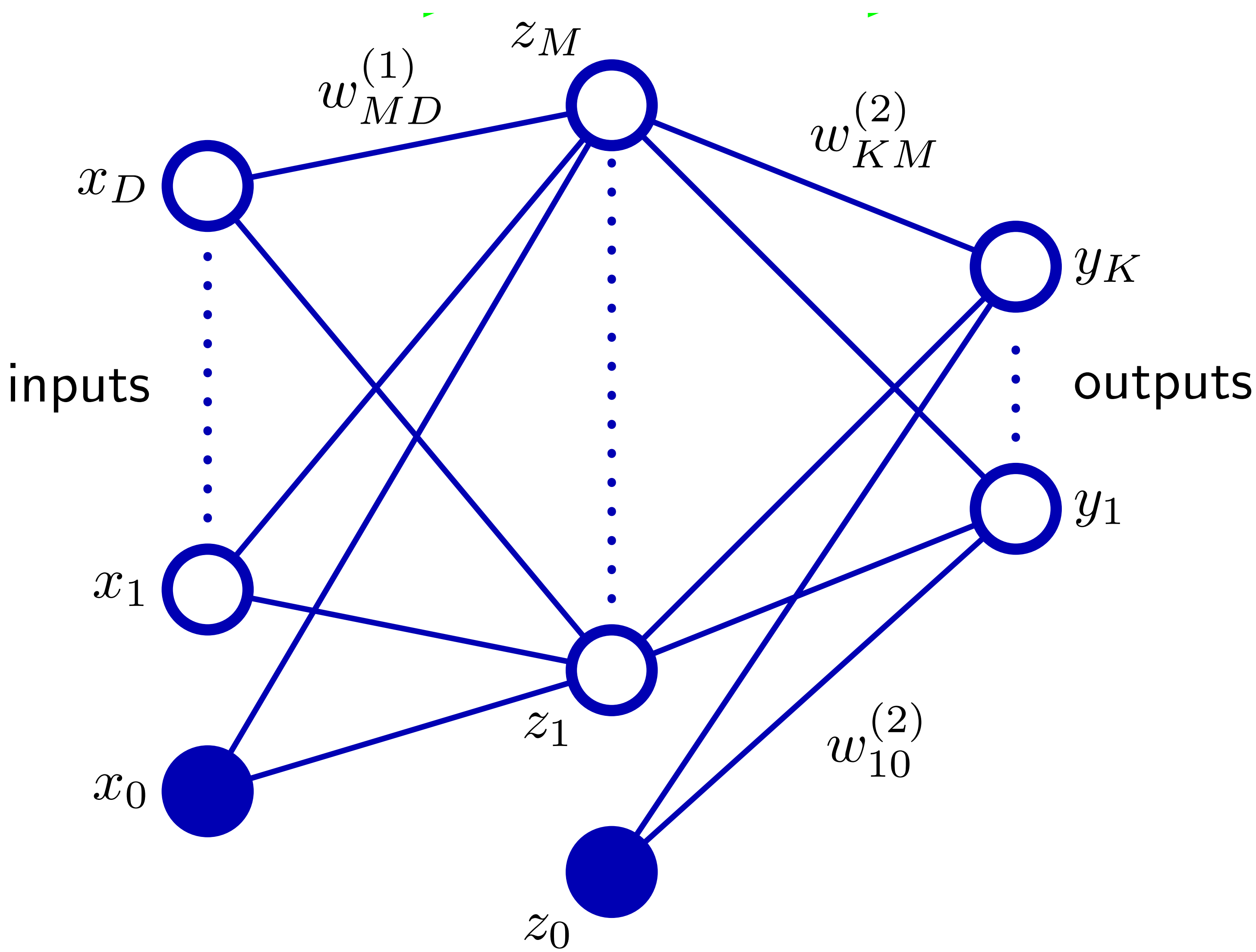
Layer 2

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} x_i + w_{k0}^{(2)}$$

$$y_k = \sigma(a_k)$$

[Slide adapted from Pattern Recognition and Machine Learning: Christopher Bishop]

Two-Layer Neural Network



Layer 1

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

$$z_j = h(a_j)$$

Layer 2

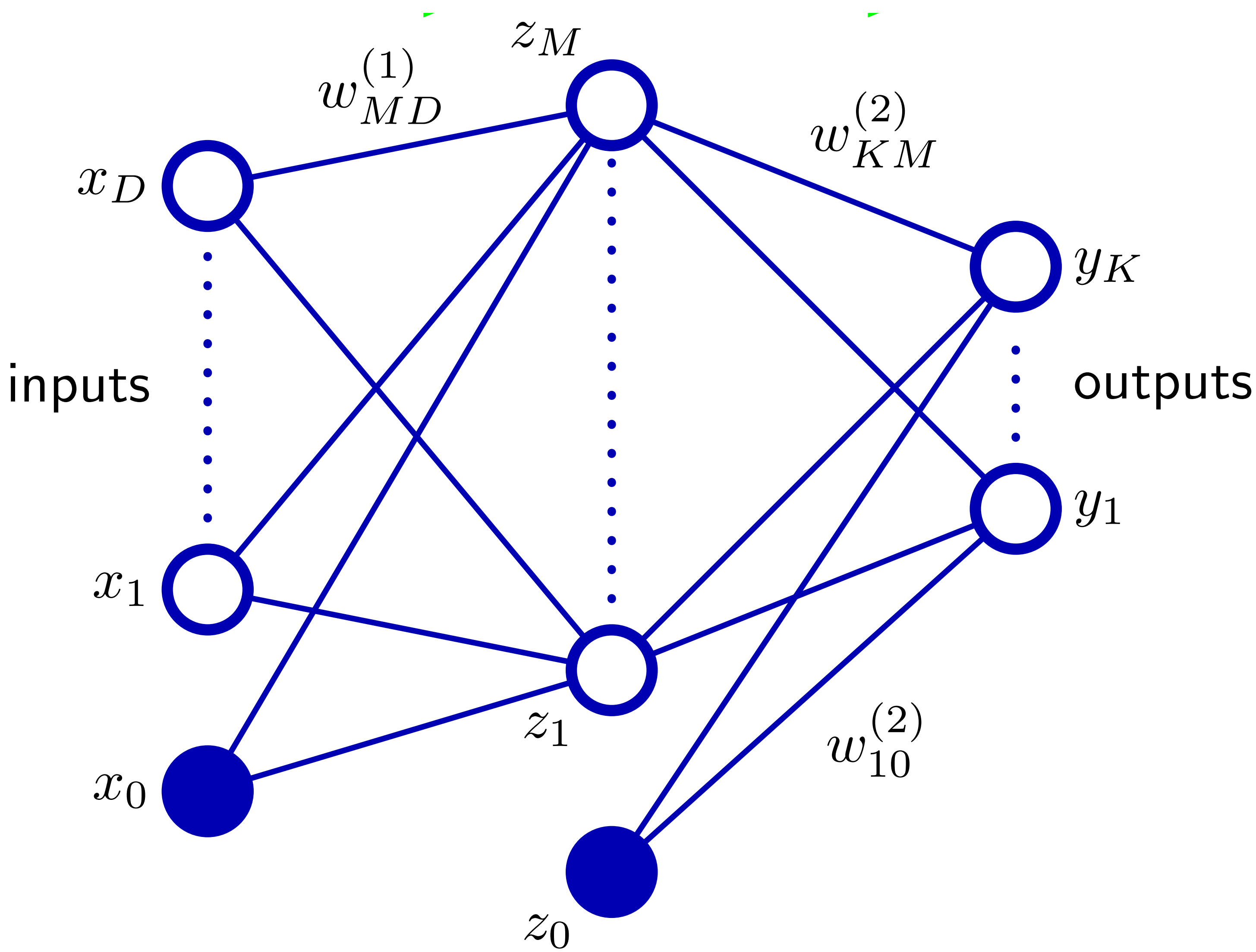
$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} x_i + w_{k0}^{(2)}$$

$$y_k = \sigma(a_k)$$

For regression tasks, $K = 1$ and σ is an identity function

[Slide adapted from Pattern Recognition and Machine Learning: Christopher Bishop]

Two-Layer Neural Network



Layer 1

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

$$z_j = h(a_j)$$

Layer 2

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} x_i + w_{k0}^{(2)}$$

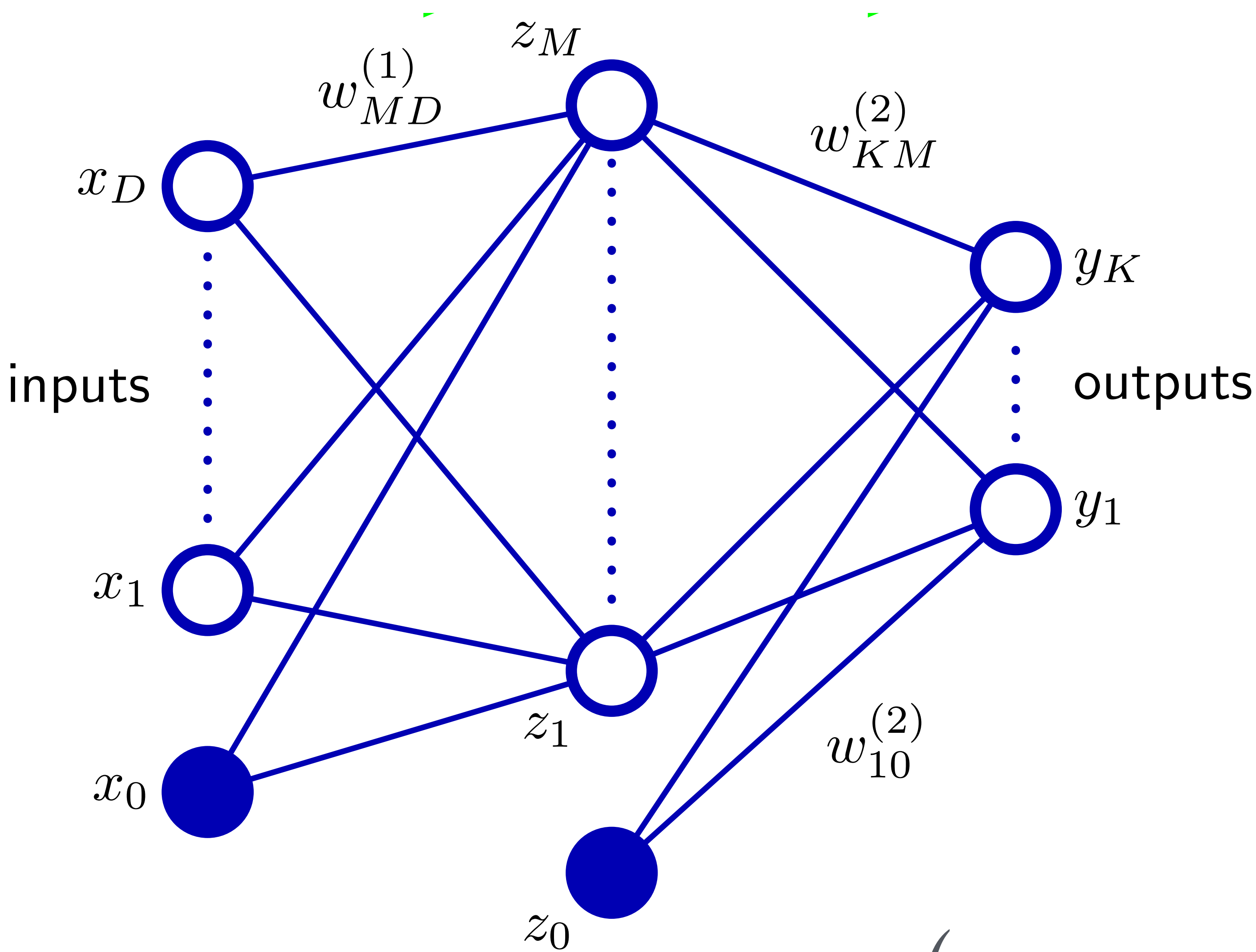
$$y_k = \sigma(a_k)$$

For multiple binary classification task each output unit activation is transformed by a sigmoid function, $\sigma(a) = \dfrac{1}{1+e^{-a}}$

[Slide adapted from Pattern Recognition and Machine Learning: Christopher Bishop]

Two-Layer Neural Network

Subsuming the biases w_{j0} and w_{k0} into the weights we get the expression for the full network



Layer 1

$$a_j = \sum_{i=1}^{D} w^{(1)}_{ji} x_i + w^{(1)}_{j0}$$

$$z_j = h(a_j)$$

Layer 2

$$a_k = \sum_{j=1}^{M} w^{(2)}_{kj} x_i + w^{(2)}_{k0}$$

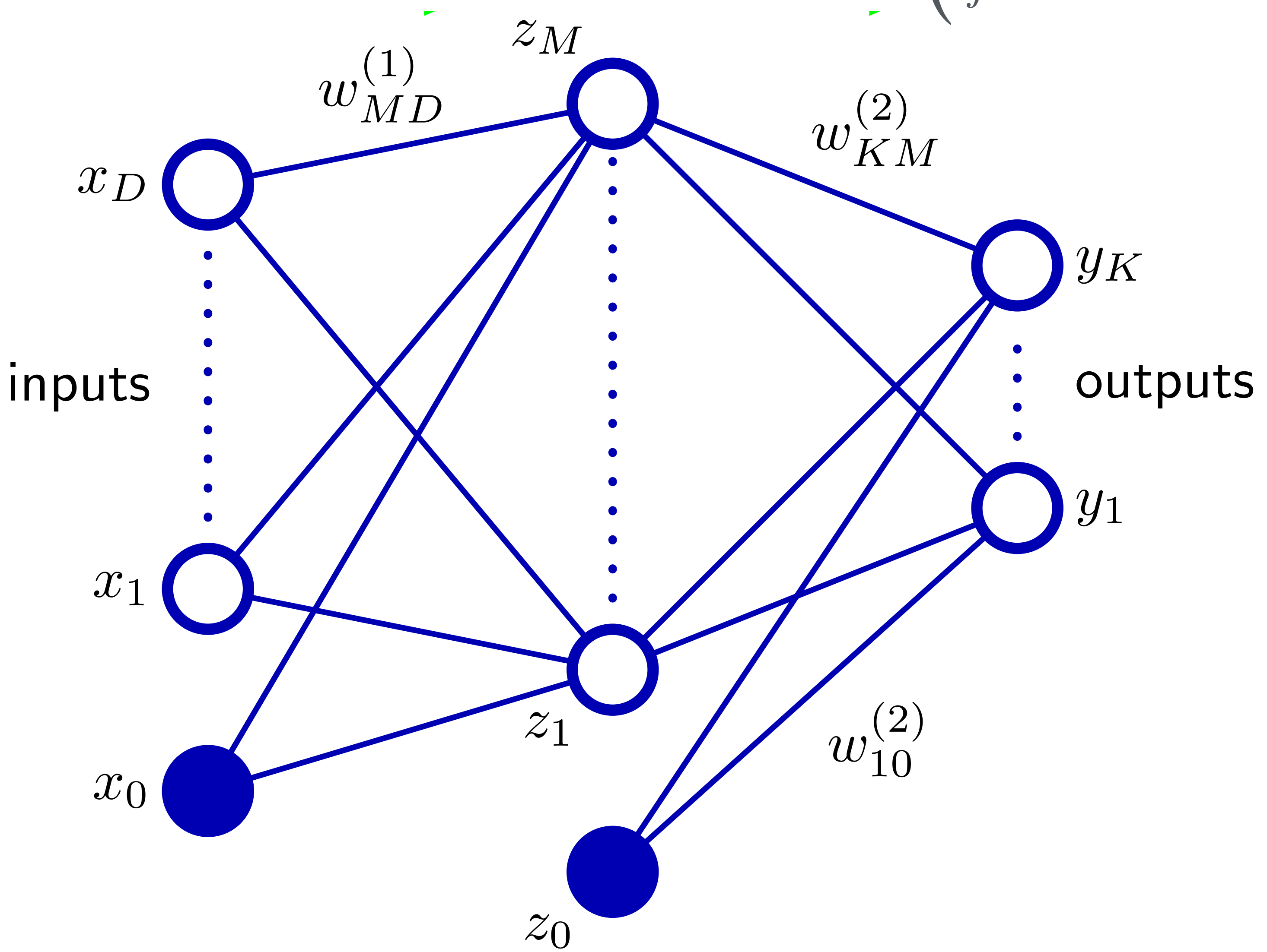
$$y_k = \sigma(a_k)$$

$$y_k(x, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w^{(2)}_{kj} \cdot h\left(\sum_{i=0}^{D} w^{(1)}_{ji} \cdot x_i\right)\right)$$

[Slide adapted from Pattern Recognition and Machine Learning: Christopher Bishop]

Two-Layer Neural Network

$$y_k(x, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} \cdot h\left(\sum_{i=0}^{D} w_{ji}^{(1)} \cdot x_i\right)\right)$$



Different notions of layers

Single hidden layer neural network

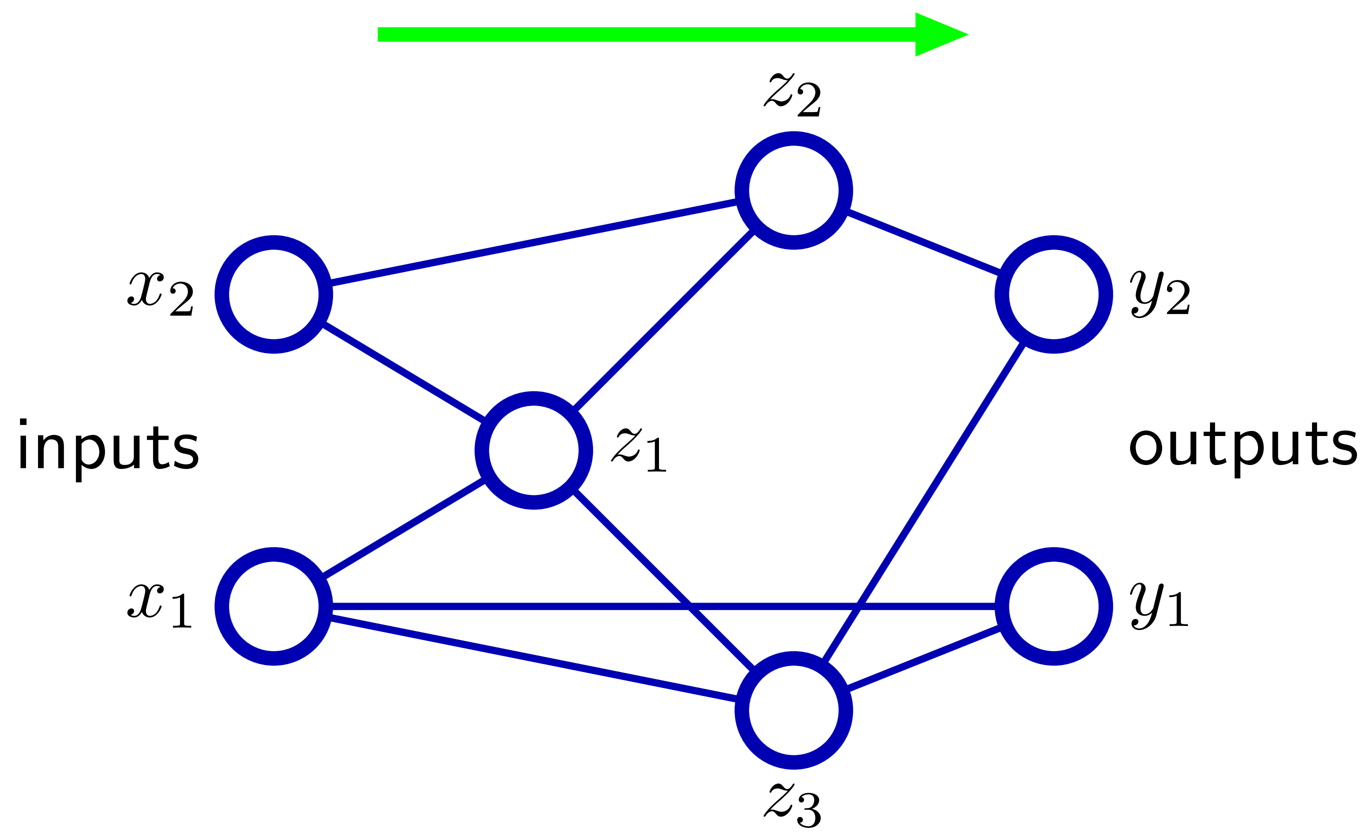
2 layer neural network (two layers of trainable weights)

3 layer neural network (3 sets of units: input, hidden, output)

[Slide adapted from Pattern Recognition and Machine Learning: Christopher Bishop]

General Neural Network

In general this forms a directed acyclic graph where the information is flowing from left to right



$$z_k = h\left(\sum_j w_{kj} z_j\right)$$

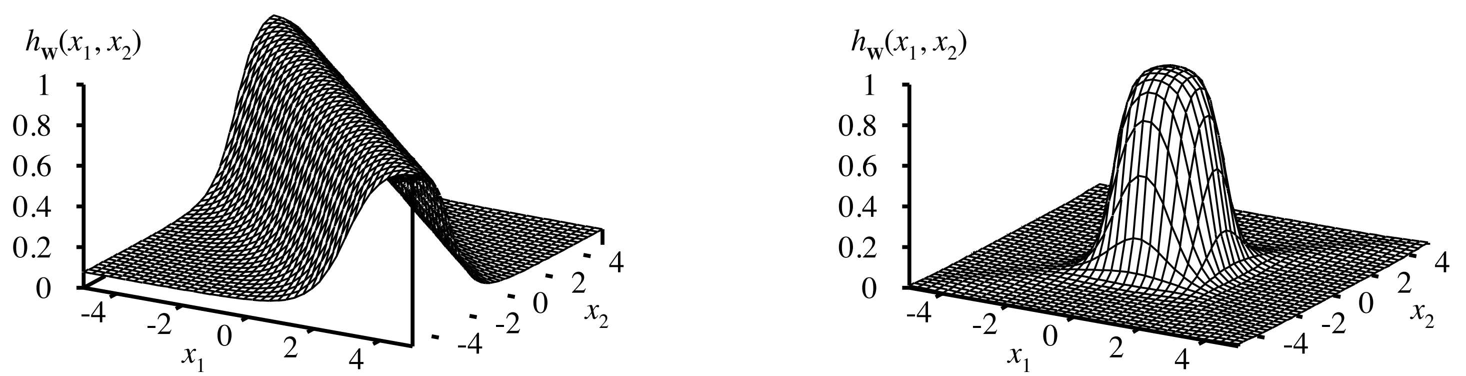
[Slide adapted from Pattern Recognition and Machine Learning: Christopher Bishop]

Universal Approximator

This repetitive composition operation is a really power concept

A two layer neural network can approximate any function under the sun

Hence it is called a Universal Approximator



Combine two opposite-facing threshold functions to make a ridge

Combine two perpendicular ridges to make a bump

Add bumps of various sizes and locations to fit any surface

Proof requires exponentially many hidden units

[Slide adapted from Stuart Russel]

Universal Approximator

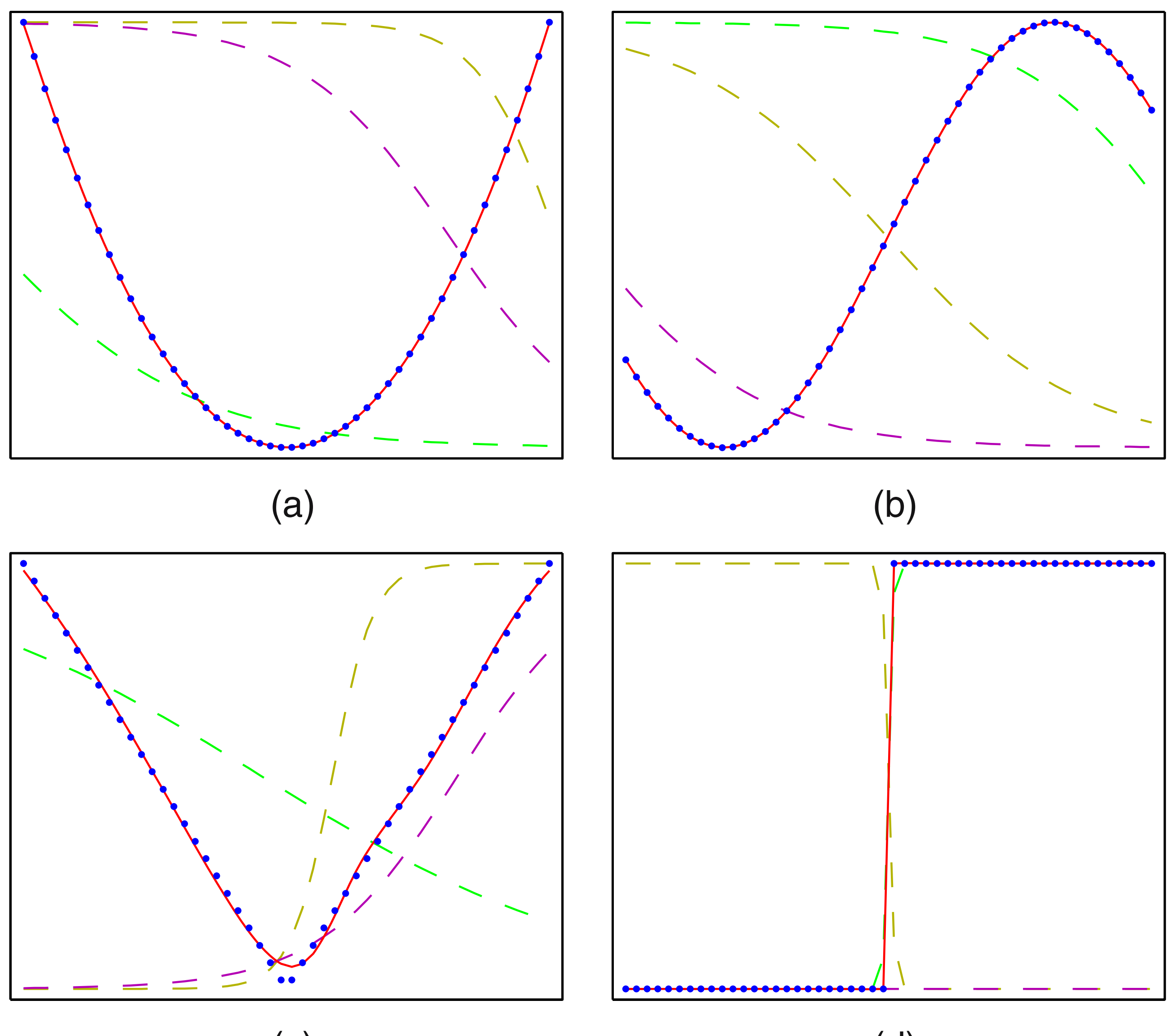
This repetitive composition operation is a really power concept

A two layer neural network can approximate any function under the sun

Hence it is called a Universal Approximator

$f(x) = x^2$

$f(x) = sin(x)$



$f(x) = |x|$

$f(x) = H(x)$

[Slide adapted from Pattern Recognition and Machine Learning: Christopher Bishop]

Training of Neural Networks

Consider the regression task

Define the likelihood function

$$p(t \,|\, x_n, \mathbf{w}) = \mathcal{N}\left(t \,|\, y(x_n, \mathbf{w}), \beta^{-1}\right)$$

$$p(\mathbf{t} \,|\, X, \mathbf{w}, \beta) = \prod_{i=1}^{N} p(t_n \,|\, x_n, \mathbf{w}, \beta)$$

Then the error (loss) function is given by taking the negative log of the likelihood

$$E(\mathbf{w}) = -\log \prod_{n=1}^{N} p(t_n \,|\, y(x_n, \mathbf{w}, \beta)$$

$$= \frac{\beta}{2} \sum_{n=1}^{N} \left[y(x_n, \mathbf{w}) - t_n\right]^2 - \frac{N}{2} \log \beta + \frac{N}{2} \log(2\pi)$$

$$\approx \frac{1}{2} \sum_{n=1}^{N} \left[y(x_n, \mathbf{w}) - t_n\right]^2$$

Training of Neural Networks

Similarly for a classification task we can define the error (loss)

Cross Entropy Loss

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{j=1}^{K} t_{kn} \log y_k(x_n, \mathbf{w})$$

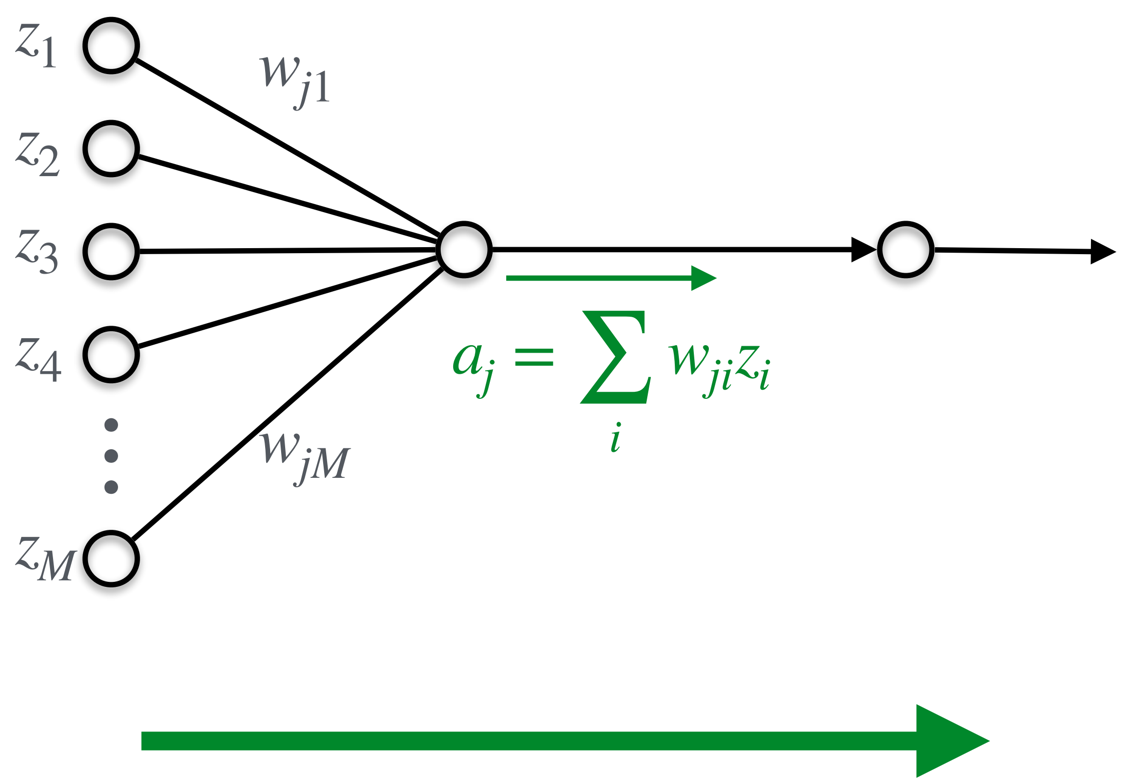
Gradient descent is the algorithm of choice for training neural networks

$$\mathbf{w}^{t+1} \leftarrow w^t - \eta \nabla E(\mathbf{w})$$

The question remains how to compute $\nabla E(\mathbf{w})$?

Backpropagation Algorithm

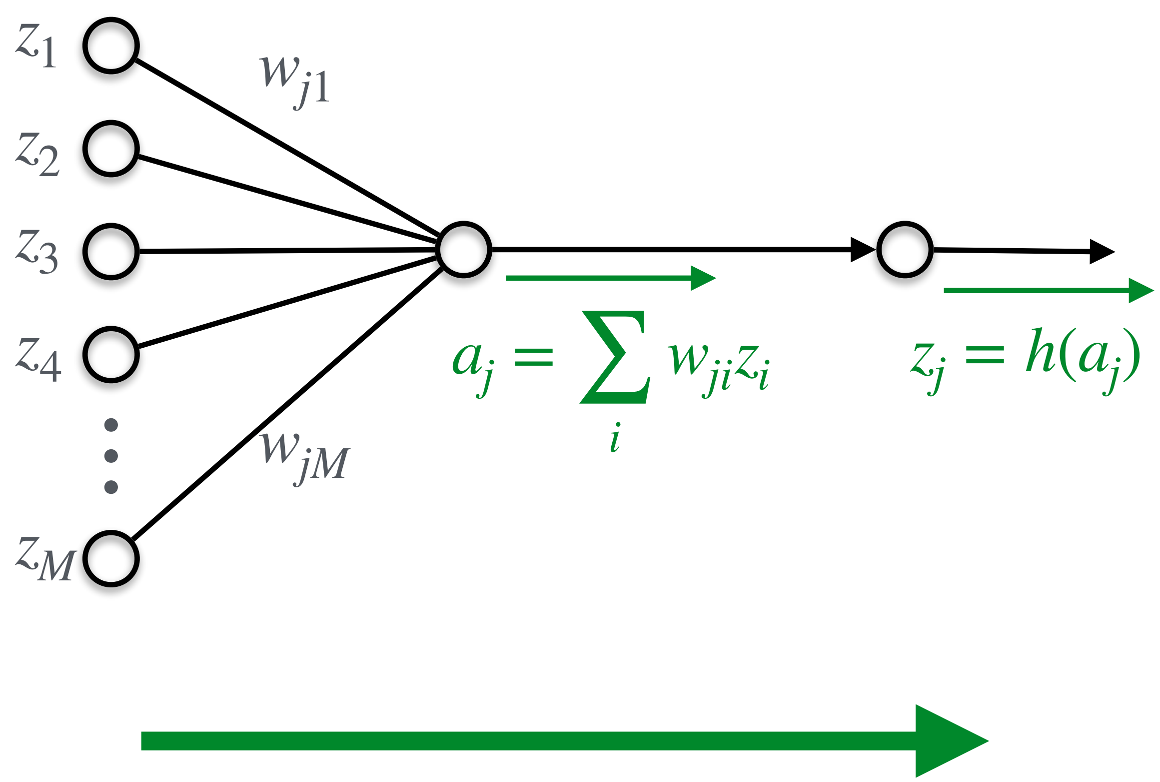
Its a message passing algorithm which has two steps: Forward pass and a Backward Pass



Forward pass: starting from the inputs x successively compute the activations of each unit

Backpropagation Algorithm

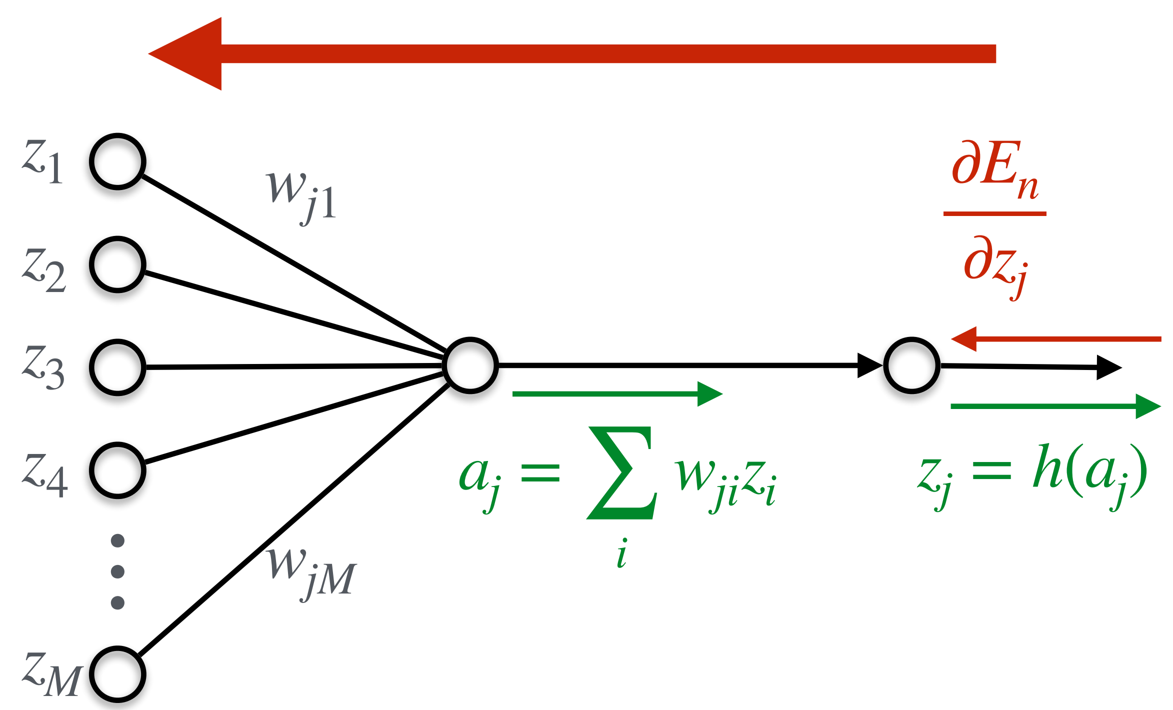
Its a message passing algorithm which has two steps: Forward pass and a Backward Pass



Forward pass: starting from the inputs x successively compute the activations of each unit

Backpropagation Algorithm

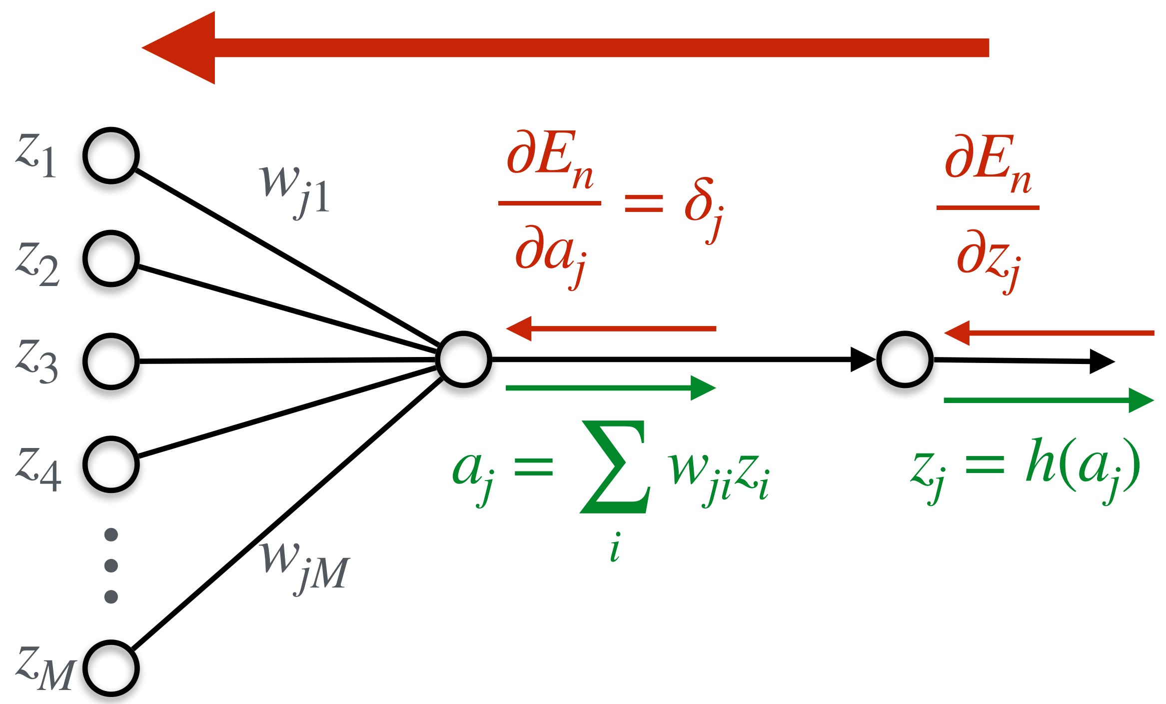
Its a message passing algorithm which has two steps: Forward pass and a Backward Pass



Backward pass: successively compute the gradients of the error function with respect to the activations and the weights

Backpropagation Algorithm

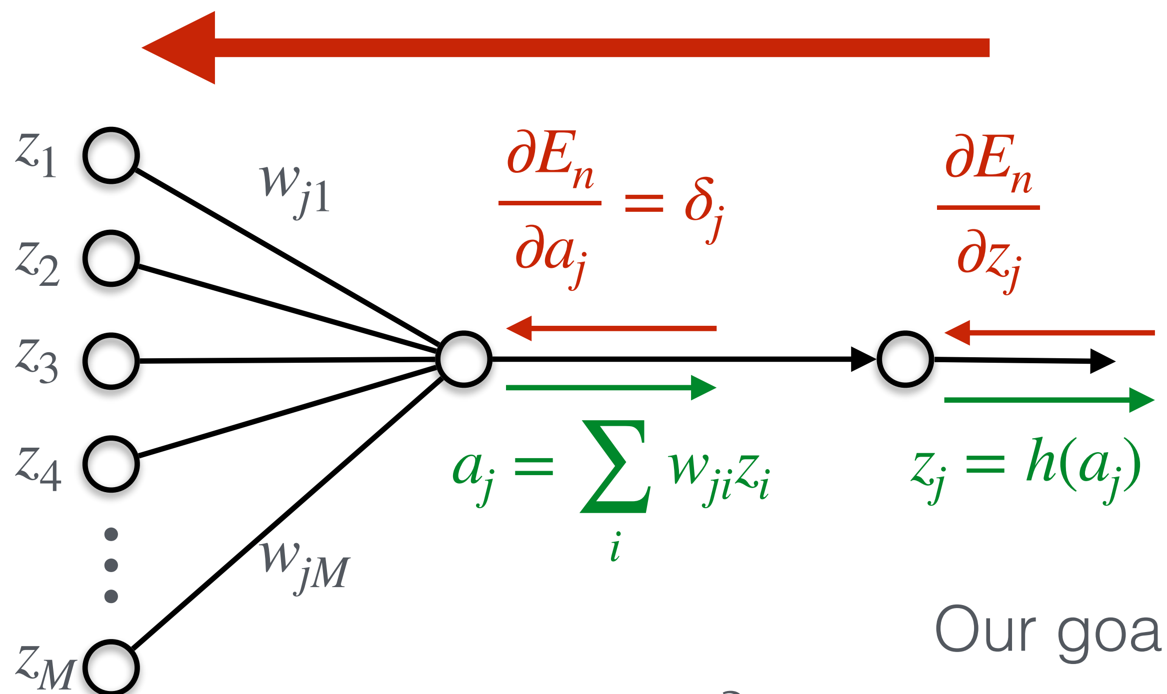
Its a message passing algorithm which has two steps: Forward pass and a Backward Pass



Backward pass: successively compute the gradients of the error function with respect to the activations and the weights

Backpropagation Algorithm

Its a message passing algorithm which has two steps: Forward pass and a Backward Pass



Our goal is to compute $\frac{\partial E_n}{w_{ji}}$

By chain rule we have $\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$

We know that $\frac{\delta a_j}{\partial w_{ji}} = z_i$

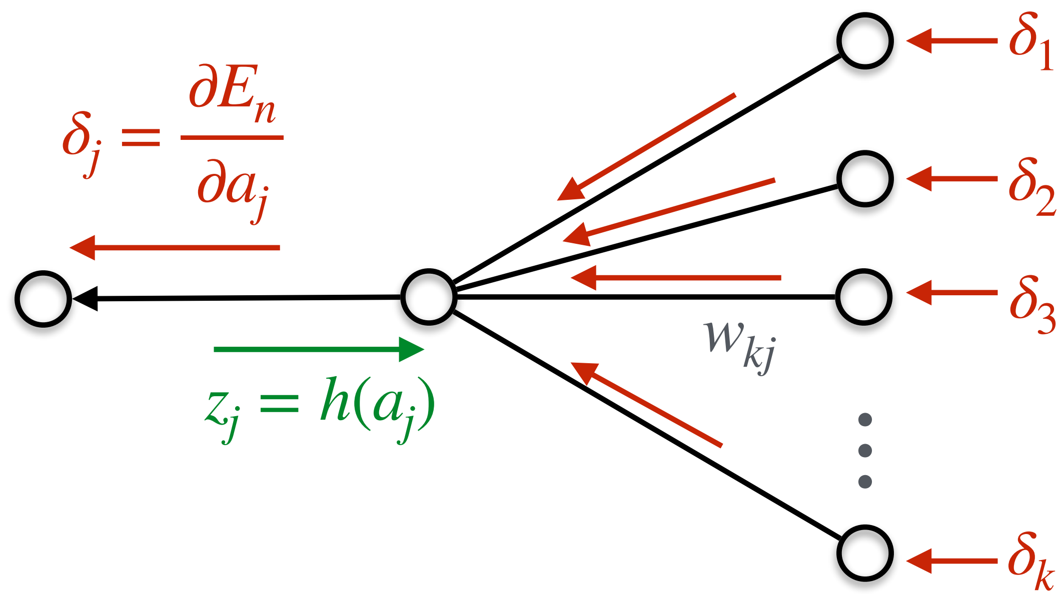
Denote $\frac{\partial E_n}{\partial a_j}$ by δ_j

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

Thus all we need is to compute the value of δ_j for each hidden unit

Backpropagation Algorithm

Its a message passing algorithm which has two steps: Forward pass and a Backward Pass



For the output unit
$\delta_k = y_k - t_k$

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \frac{\partial E_n}{\partial z_j} \cdot \frac{\partial z_j}{\partial a_j} = \left[\sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial z_j} \right] \cdot h'(a_j)$$

We know that $\frac{\delta a_k}{\partial z_j} = w_{kj}$ and $\frac{\delta E_n}{\delta a_k} = \delta_k$

The value of δ_j for hidden unit j $\qquad \delta_j = h'(a_j) \sum_k w_{kj} \delta_k$

Backpropagation Algorithm

Apply input x_n to the network to compute the activations of all units (hidden and output) in the forward pass

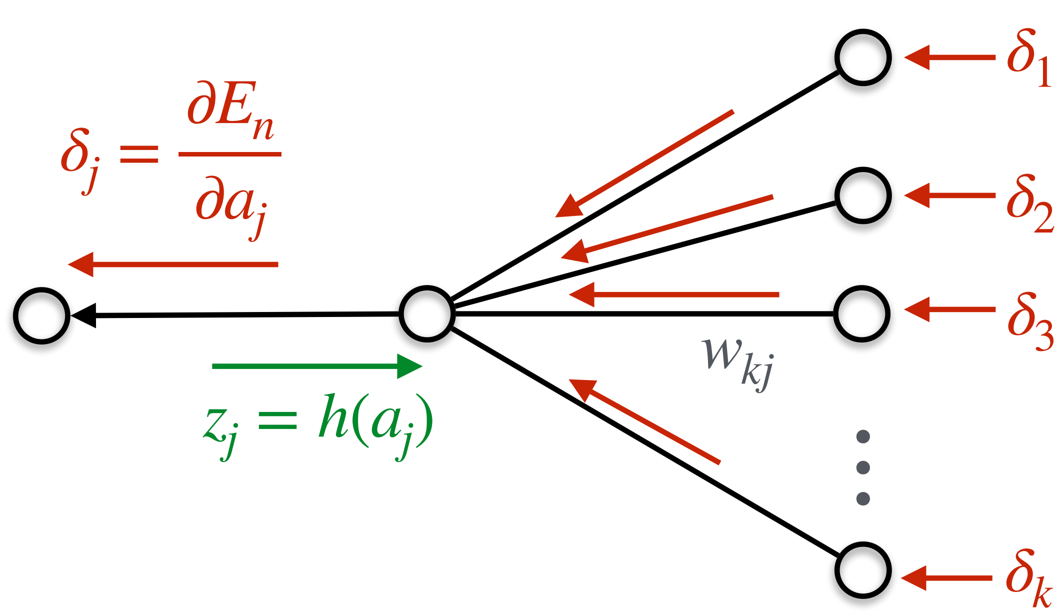
Evaluate δ_k for all output units

Backpropagate the $\delta's$ to compute the $\delta's$ for all hidden units using the equation

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

Evaluate the derivatives with respect to the weights w_{ji} using equation $\dfrac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$

This is a general technique and can be used to compute the Jacobian matrix (derivative of outputs wrt inputs) and Hessian matrix (second derivative of weights)



Regularization in Neural Networks

Weight Decay

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \boxed{\mathbf{w}^T\mathbf{w}}$$

While this is a popular choice in practice it has certain flaws

It is not invariant to arbitrary scaling/shifts of the inputs or output

$$z_j = h\left(\sum_i w_{ji}x_i + w_{j0}\right) \qquad y_k = \sum_j w_{kj}z_j + w_{k0}$$

$$x_i \rightarrow ax_i + b$$

$$w_{ji} \rightarrow \frac{1}{a}w_{ji} \qquad w_{j0} \rightarrow w_{j0} - \frac{b}{a}\sum_i w_{ji}$$

Weight Decay by definition is not invariant to such scaling

Regularization in Neural Networks

Weight Decay

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \boxed{\mathbf{w}^T\mathbf{w}}$$

While this is a popular choice in practice it has certain flaws

It is not invariant to arbitrary scaling/shifts of the inputs or output

$$z_j = h\left(\sum_i w_{ji}x_i + w_{j0}\right) \qquad y_k = \sum_j w_{kj}z_j + w_{k0}$$

$$x_i \to ax_i + b$$

$$w_{ji} \to \frac{1}{a}w_{ji} \qquad w_{j0} \to w_{j0} - \frac{b}{a}\sum_i w_{ji}$$

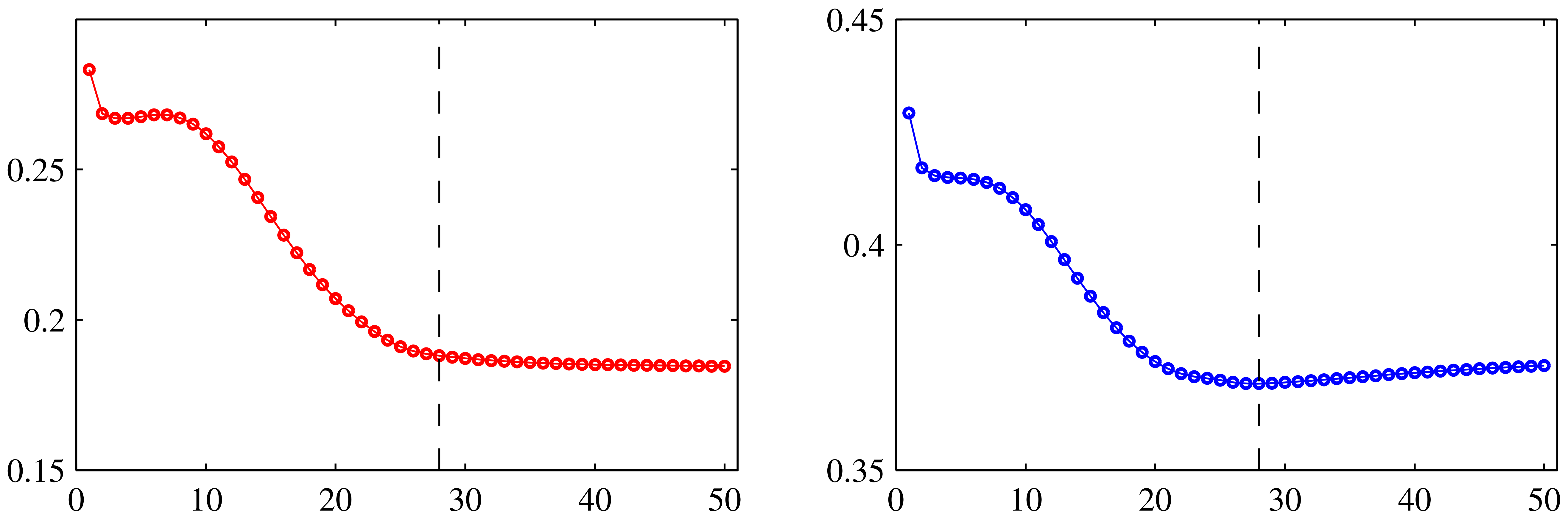
A better regularizer will be

$$\frac{\lambda_1}{2}\sum_{w\in\mathcal{W}_1} w^2 + \frac{\lambda_2}{2}\sum_{w\in\mathcal{W}_2} w^2$$

Regularization in Neural Networks

Early stopping

Constantly monitor the model performance on the validation set and stop as soon as the validation error starts to creep up



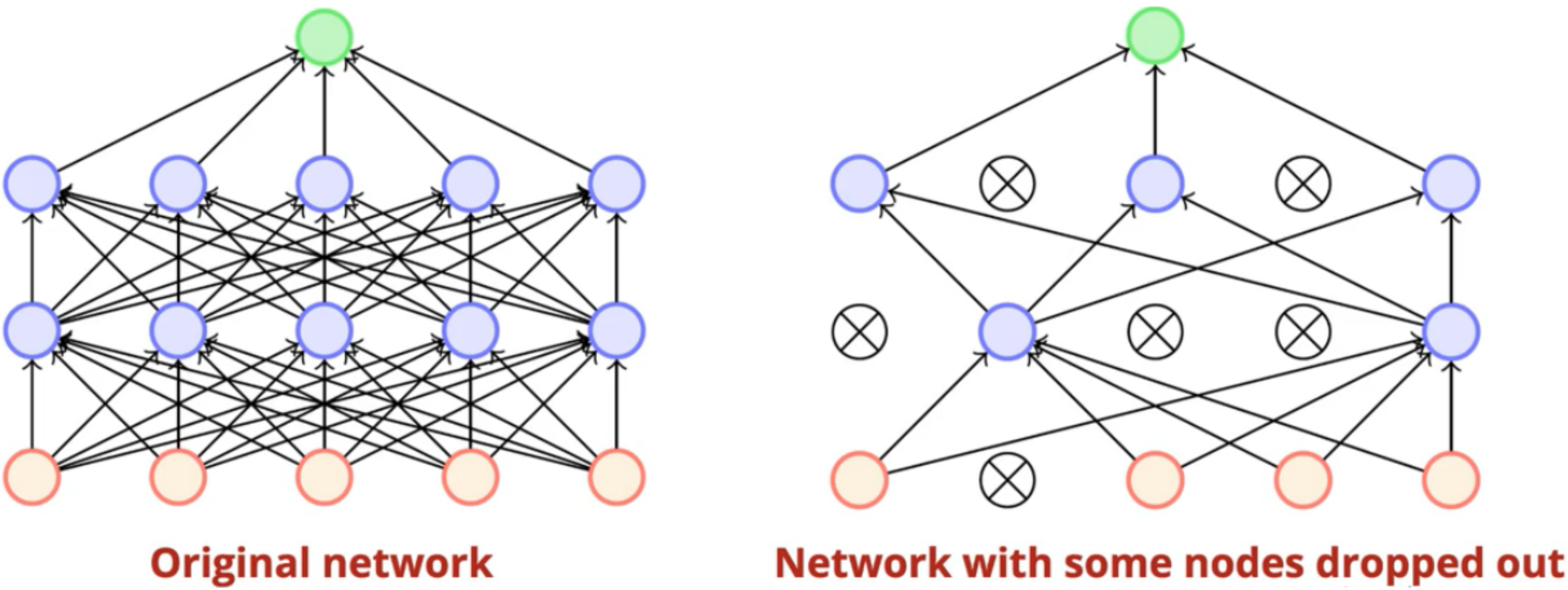
[Slide adapted from Pattern Recognition and Machine Learning: Christopher Bishop]

Regularization in Neural Networks

Dropout

For each hidden unit, flip a biased coin with probability of heads equal to p

Switch the unit off if its a head



End of Lecture 08