Neural Networks Introduction

Nakul Gopalan
Georgia Tech

These slides are from Vivek Srikumar, Mahdi Roozbahani and Chao Zhang.

Outline

- Perceptron ←
- Stacking Linear Threshold Units
- Neural Networks
- Expressivity of Neural Networks
- Predicting with Neural Networks
- Backpropogation

Linear Classifiers

- Input is a n dimensional vector **x**
- Output is a label $y \in \{-1, 1\}$

- *Linear Threshold Units* (LTUs) classify an example **x** using the following classification rule
  - Output = $\text{sgn}(\mathbf{w}^T\mathbf{x} + b) = \text{sgn}(b + \sum w_i x_i)$
  - $\mathbf{w}^T\mathbf{x} + b \geq 0 \rightarrow$ Predict $y = 1$
  - $\mathbf{w}^T\mathbf{x} + b < 0 \rightarrow$ Predict $y = -1$

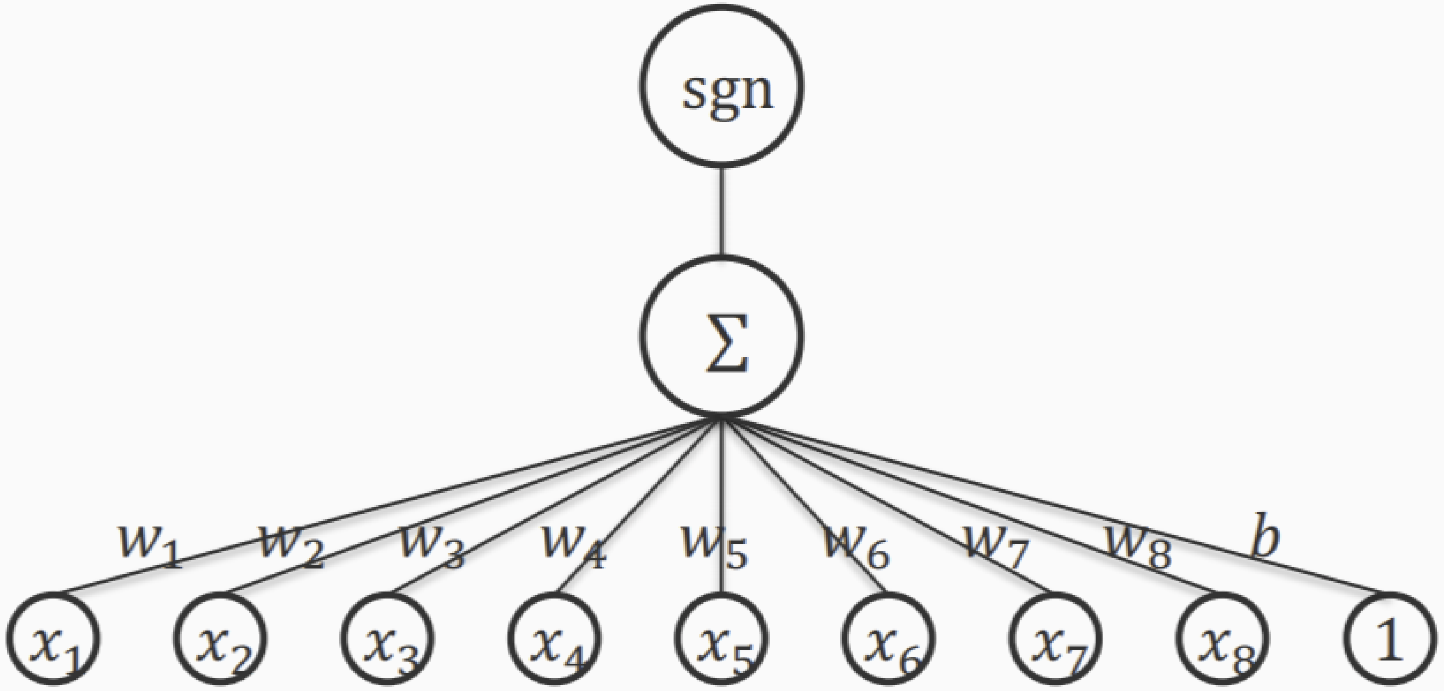
    b is called the bias term

These slides are from Vivek Srikumar

Linear Classifiers

- Input is a n dimensional vector **x**
- Output is a label y ∈ {-1, 1}

- *Linear Threshold Units* (LTUs) classify an example **x** using the following classification rule



These slides are from Vivek Srikumar

Perceptron

- Rosenblatt 1958
- The goal is to find a separating hyperplane
  - For separable data, guaranteed to find one
- An online algorithm
  - Processes one example at a time
- Several variants exist (will discuss briefly at towards the end)

Perceptron Algorithm

Input: A sequence of training examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots$ where all $x_i \in \Re^n$, $y_i \in \{-1,1\}$

- Initialize $\mathbf{w}_0 = 0 \in \Re^n$
- For each training example $(\mathbf{x}_i, y_i)$:
  - Predict $y' = \text{sgn}(\mathbf{w}_t^T \mathbf{x}_i)$
  - If $y_i \neq y'$:
    - Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,(y_i\,\mathbf{x}_i)$
- Return final weight vector

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\,\mathbf{x}_i$

r is the learning rate, a small positive number less than 1

Update only on error. A mistake-driven algorithm

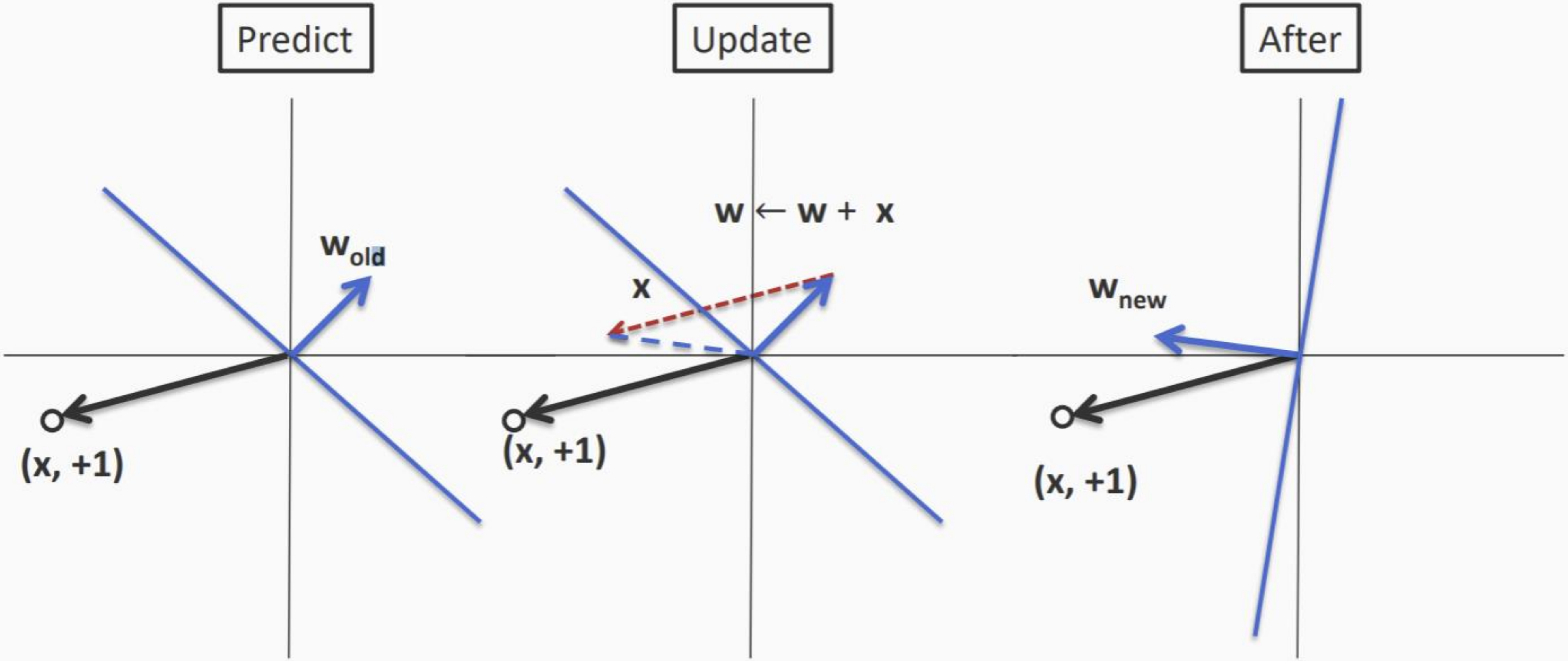
This is the simplest version. We will see more robust versions at the end

Mistake can be written as $y_i \mathbf{w}_t^T \mathbf{x}_i \leq 0$

These slides are from Vivek Srikumar

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\,\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\,\mathbf{x}_i$

Geometry of the perceptron update



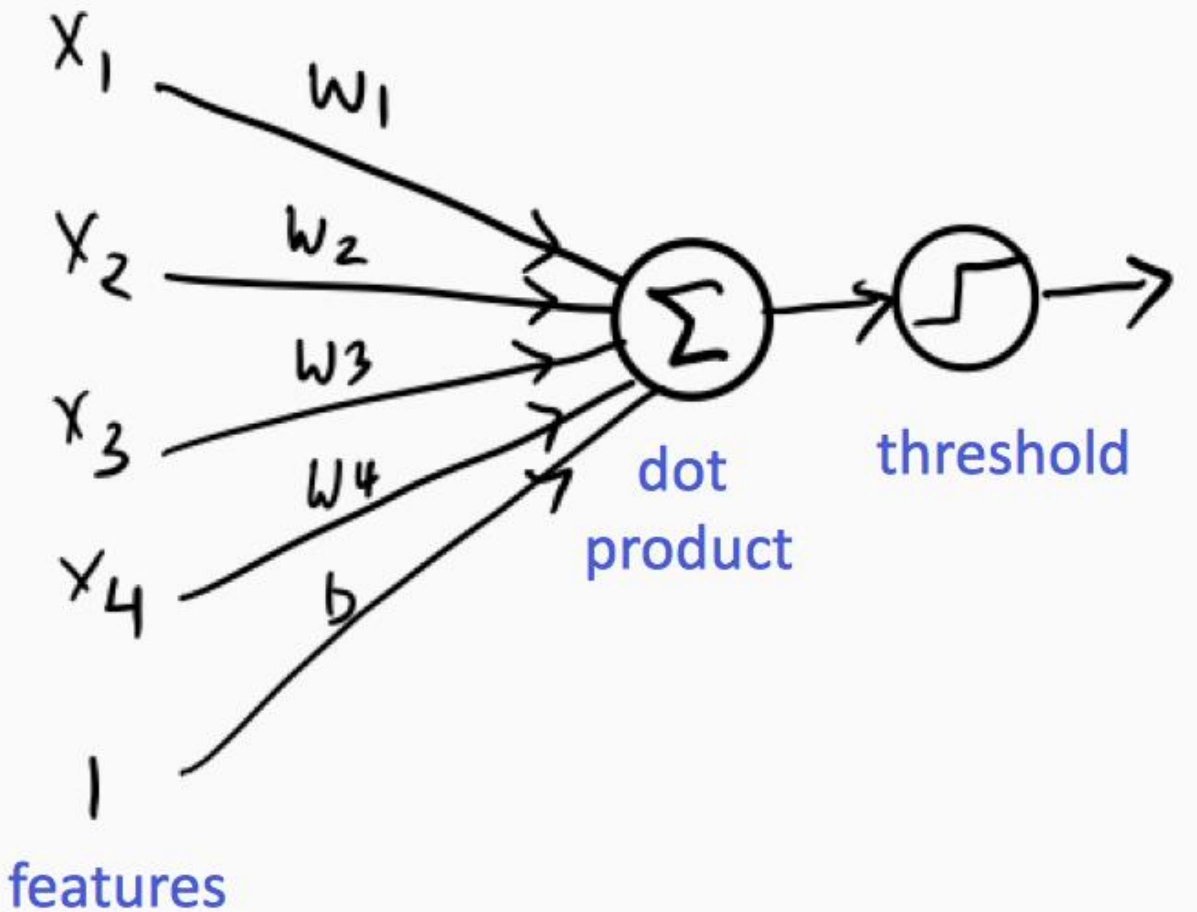
For a mistake on a positive example

These slides are from Vivek Srikumar

Outline

- Perceptron
- Stacking Linear Threshold Units ⬅
- Neural Networks
- Expressivity of Neural Networks
- Predicting with Neural Networks
- Backpropogation

Linear Threshold Unit



Prediction

$$sgn\,(\boldsymbol{w}^T\boldsymbol{x} + b) = sgn(\sum w_i x_i + b)$$

Learning

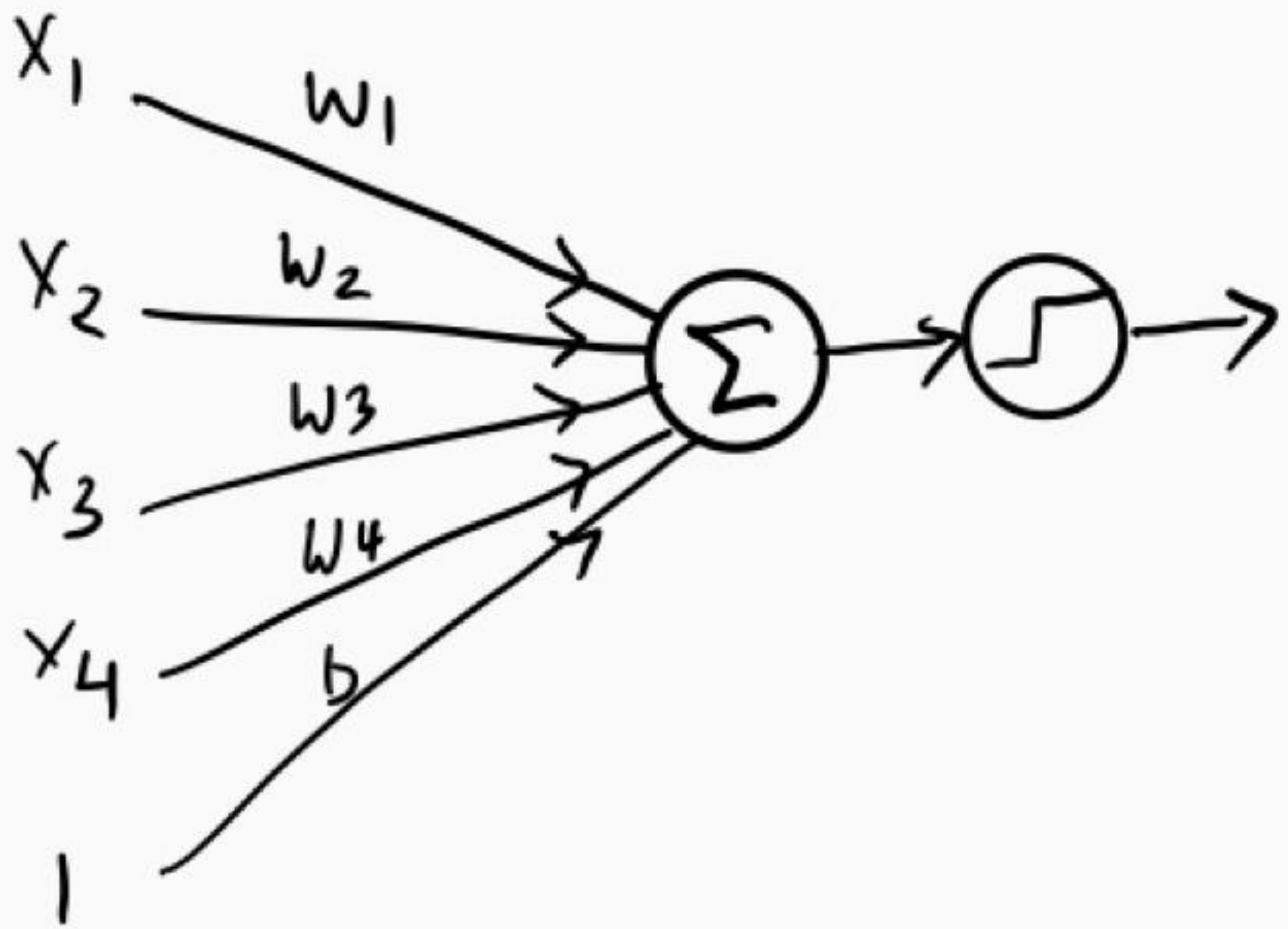
various algorithms
perceptron, SVM, logistic regression,...

in general, minimize loss

But where do these input features come from?

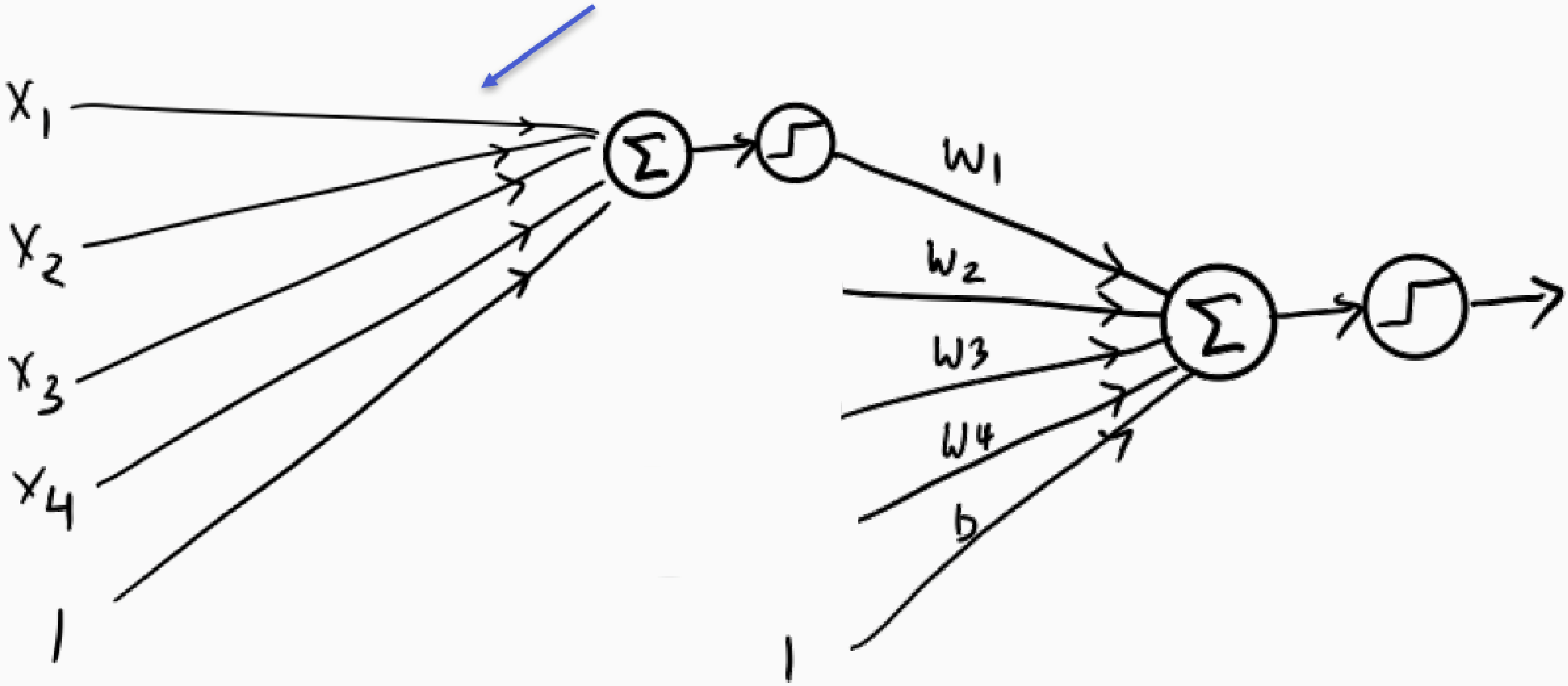
What if the features were outputs of another classifier?

Features for Linear Threshold Unit



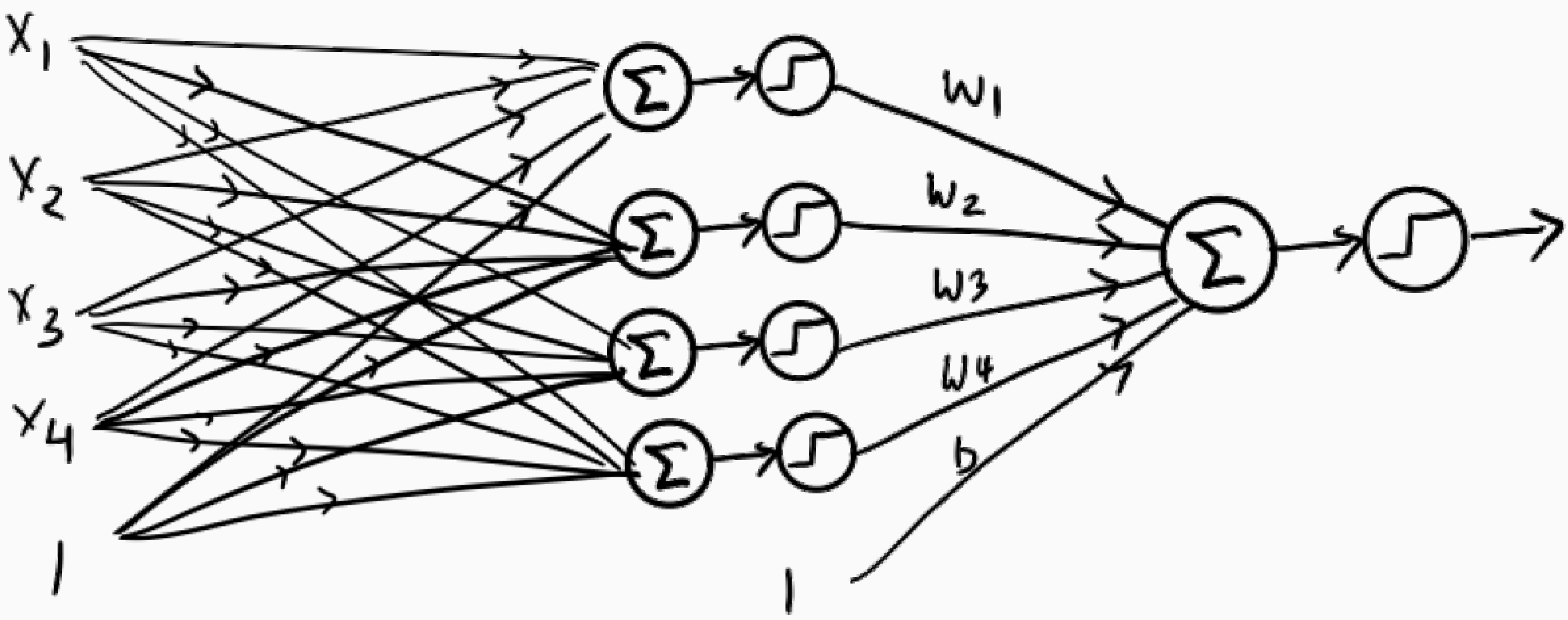
Features from Classifiers

Each of these connections have their own weights as well



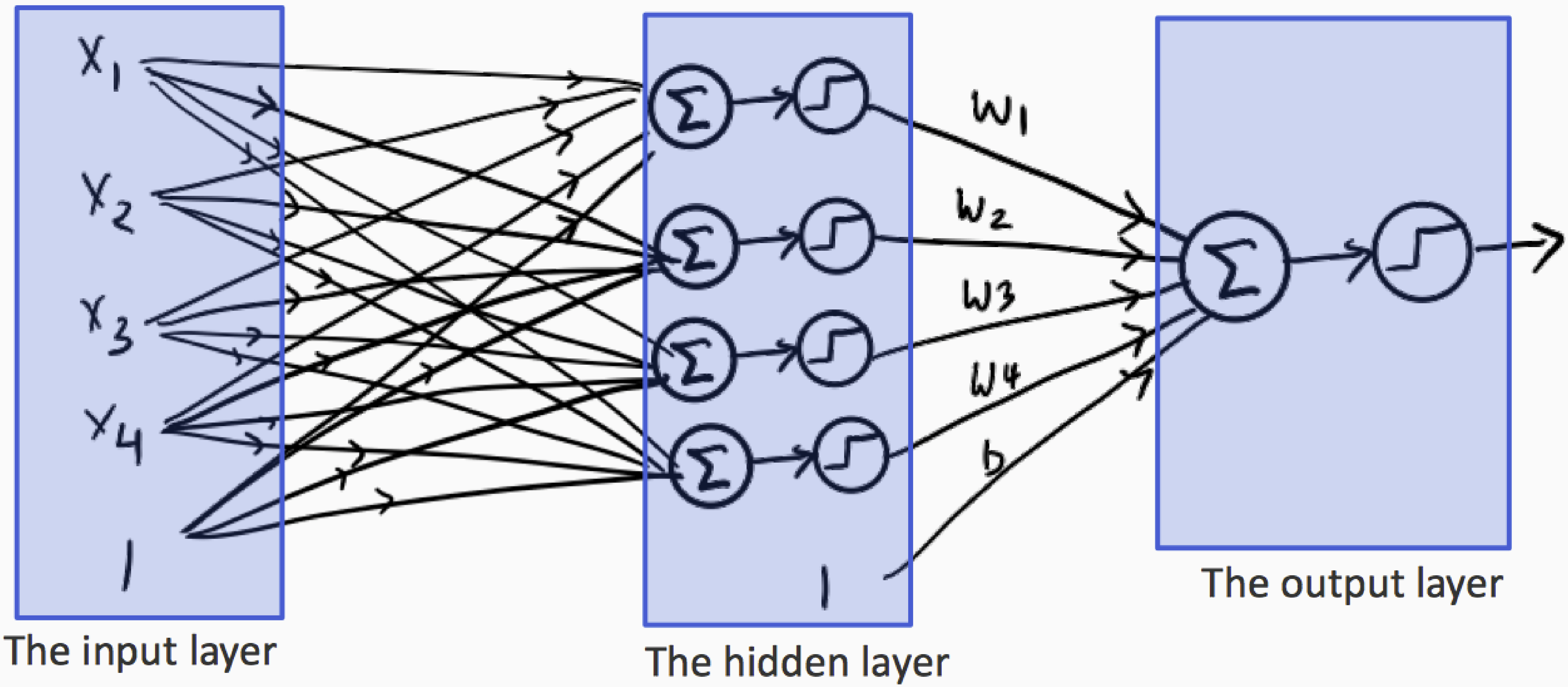
Features from Classifiers

This is a **two layer** feed forward neural network



Features from Classifiers

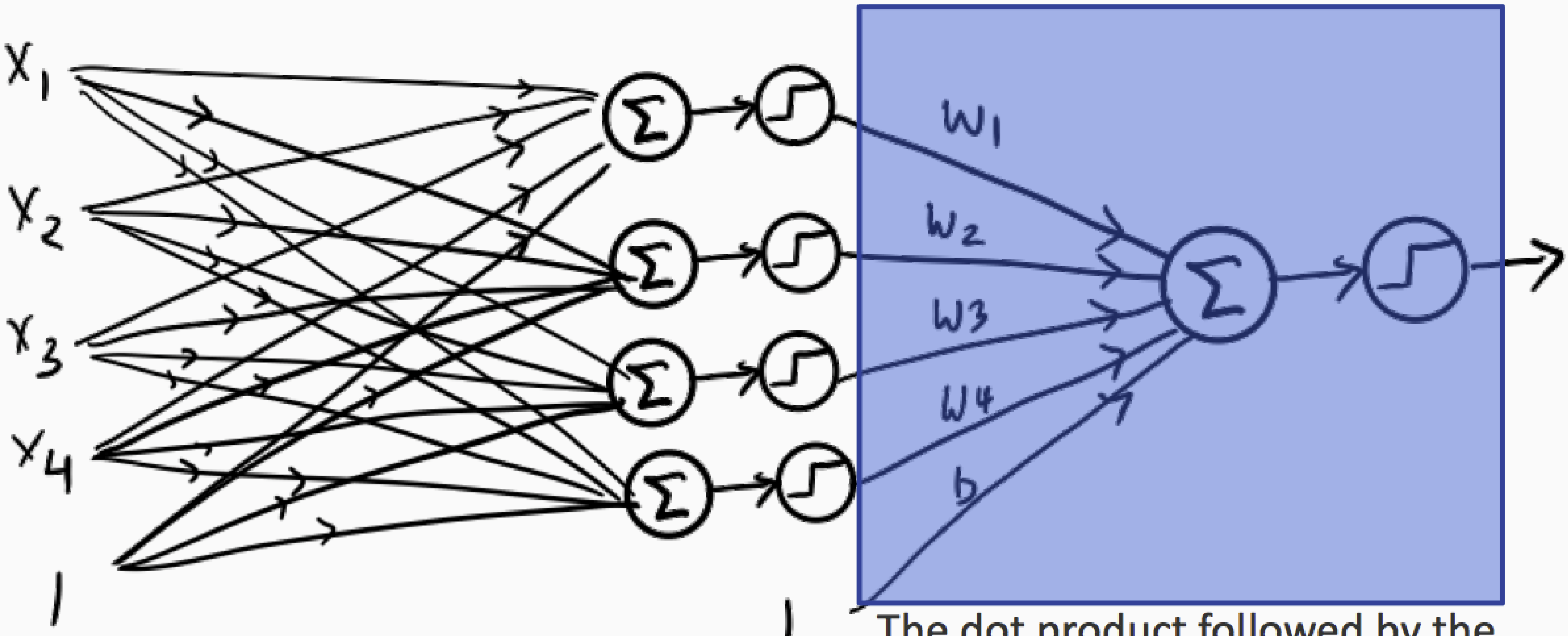
This is a **two layer** feed forward neural network



Think of the hidden layer as learning a good **representation** of the inputs

Features from Classifiers

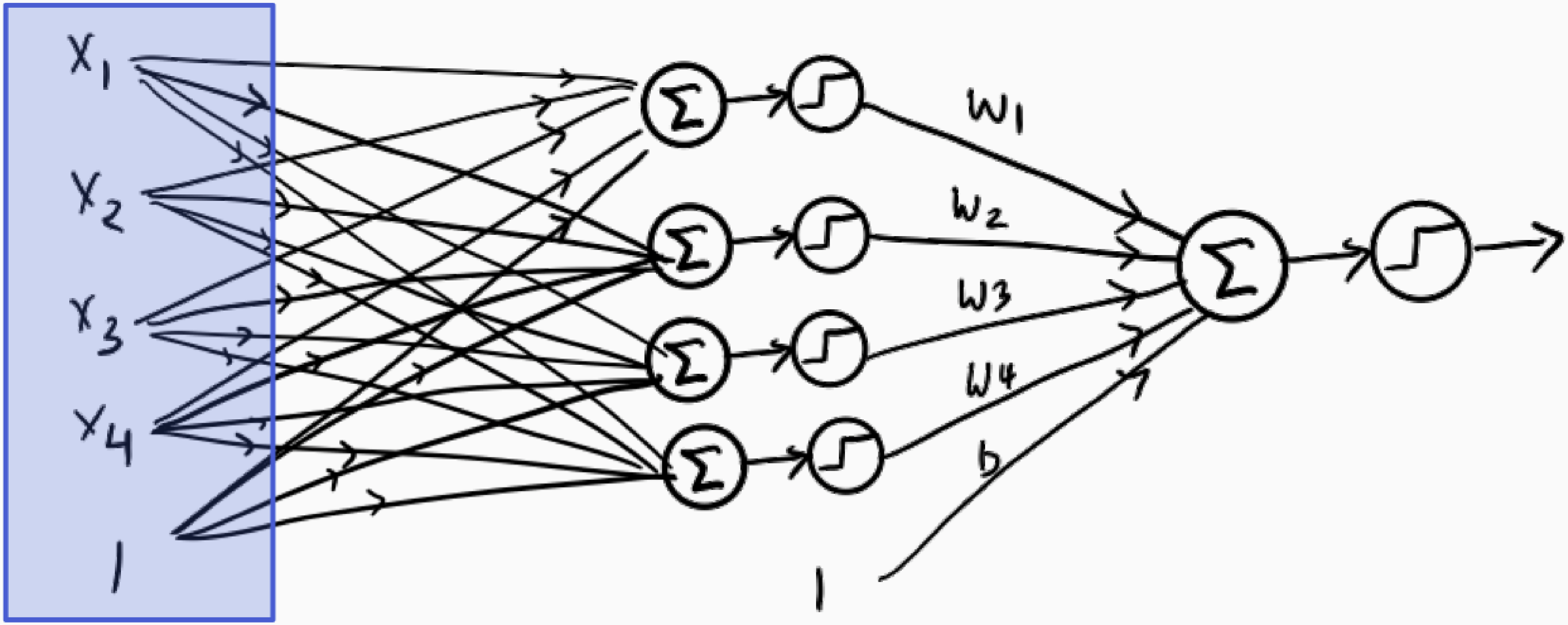
This is a **two layer** feed forward neural network



The dot product followed by the threshold constitutes a ***neuron***

Five neurons in this picture (four in hidden layer and one output)

Features from Classifiers



The input layer

What if the inputs were the outputs of a classifier?

We can make a **three** layer network…. And so on.

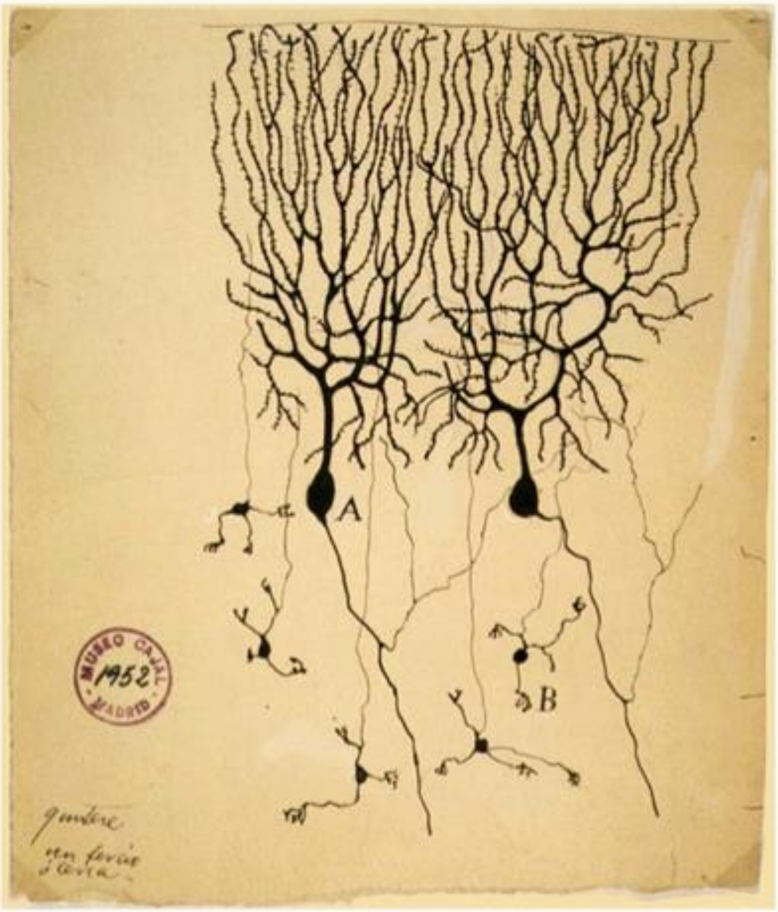
Outline

- Perceptron
- Stacking Linear Threshold Units
- Neural Networks ⬅
- Expressivity of Neural Networks
- Predicting with Neural Networks
- Backpropogation

Neural Networks

- A robust approach for approximating real-valued, discrete-valued or vector valued functions
- Among the most effective **general purpose** supervised learning methods currently known
  - Especially for *complex and hard to interpret data* such as real-world sensory data
- The Backpropagation algorithm for neural networks has been shown successful in many practical problems
  - handwritten character recognition, speech recognition, object recognition, some NLP problems

Inspiration from Biological Neurons



The first drawing of a brain cells by Santiago Ramón y Cajal in 1899

Neurons: core components of brain and the nervous system consisting of

1. Dendrites that collect information from other neurons
2. An axon that generates outgoing spikes

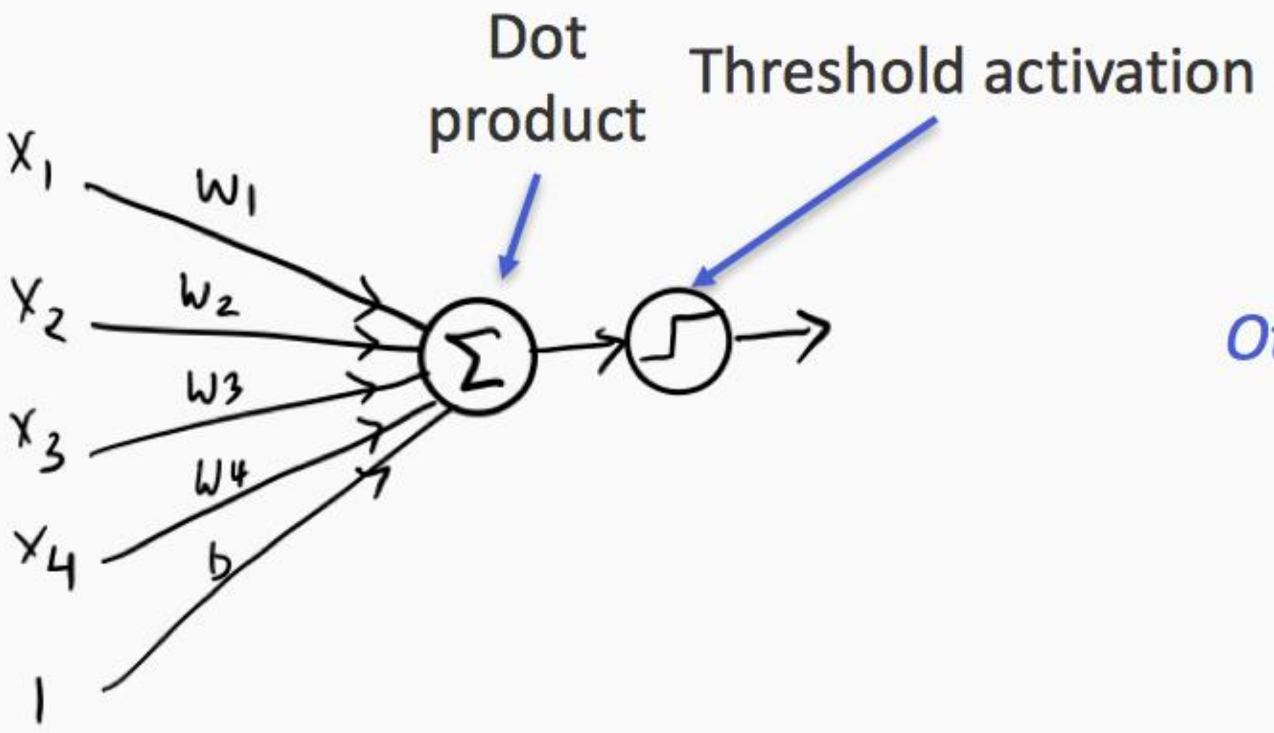
Artificial Neurons

*Functions that **very loosely** mimic a biological neuron*

A neuron accepts a collection of inputs (a vector **x**) and produces an output by:

1. Applying a dot product with weights **w** and adding a bias b
2. Applying a (possibly non-linear) transformation called an ***activation***

$$output = activation(\boldsymbol{w}^T \boldsymbol{x} + b)$$



Other activations are possible

Activation Functions

$$output = activation(\mathbf{w}^T \boldsymbol{x} + b)$$

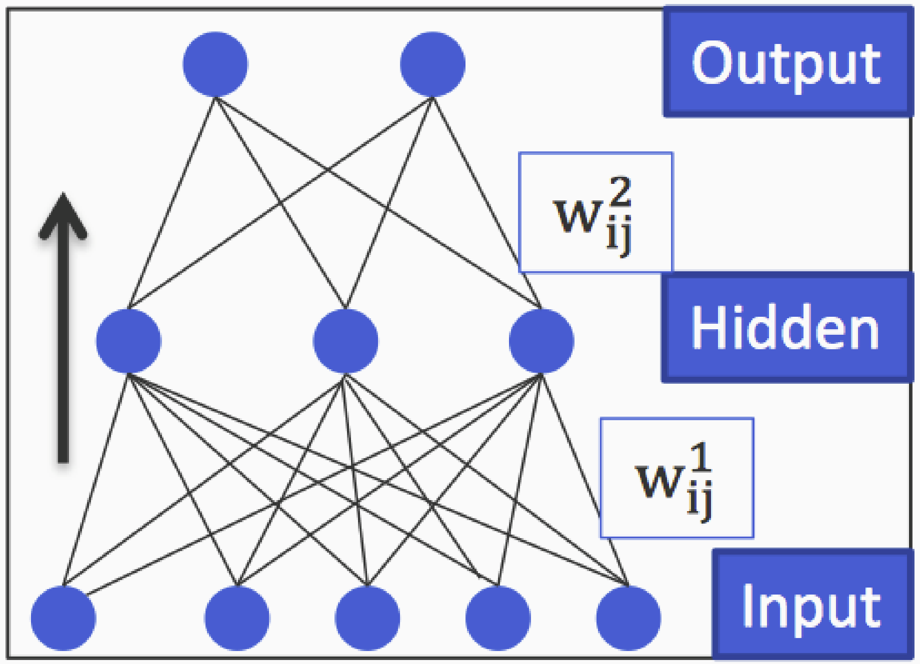
Name of the neuron	Activation function: $activation(z)$
Linear unit	z
Threshold/sign unit	$\text{sgn}(z)$
Sigmoid unit	$\frac{1}{1 + \exp(-z)}$
Rectified linear unit (ReLU)	$\max(0, z)$
Tanh unit	$\tanh(z)$

Many more activation functions exist (sinusoid, sinc, gaussian, polynomial…)

Neural Network

A function that converts inputs to outputs defined by a directed acyclic graph

- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights

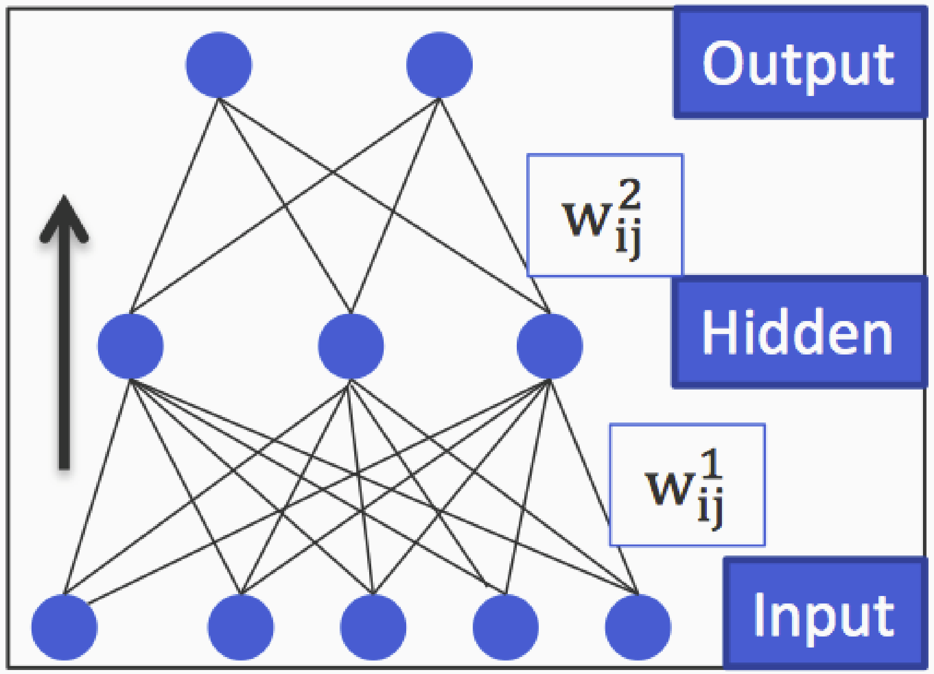


- To define a neural network, we need to specify:
  - The structure of the graph
    - How many nodes, the connectivity
  - The activation function on each node
  - The edge weights

Neural Network

A function that converts inputs to outputs defined by a directed acyclic graph

- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights



- To define a neural network, we need to specify:
  - The structure of the graph
    - How many nodes, the connectivity
  - The activation function on each node
  - The edge weights

Called the *architecture* of the network
Typically predefined, part of the design of the classifier

Learned from data

A Brief History of Neural Network

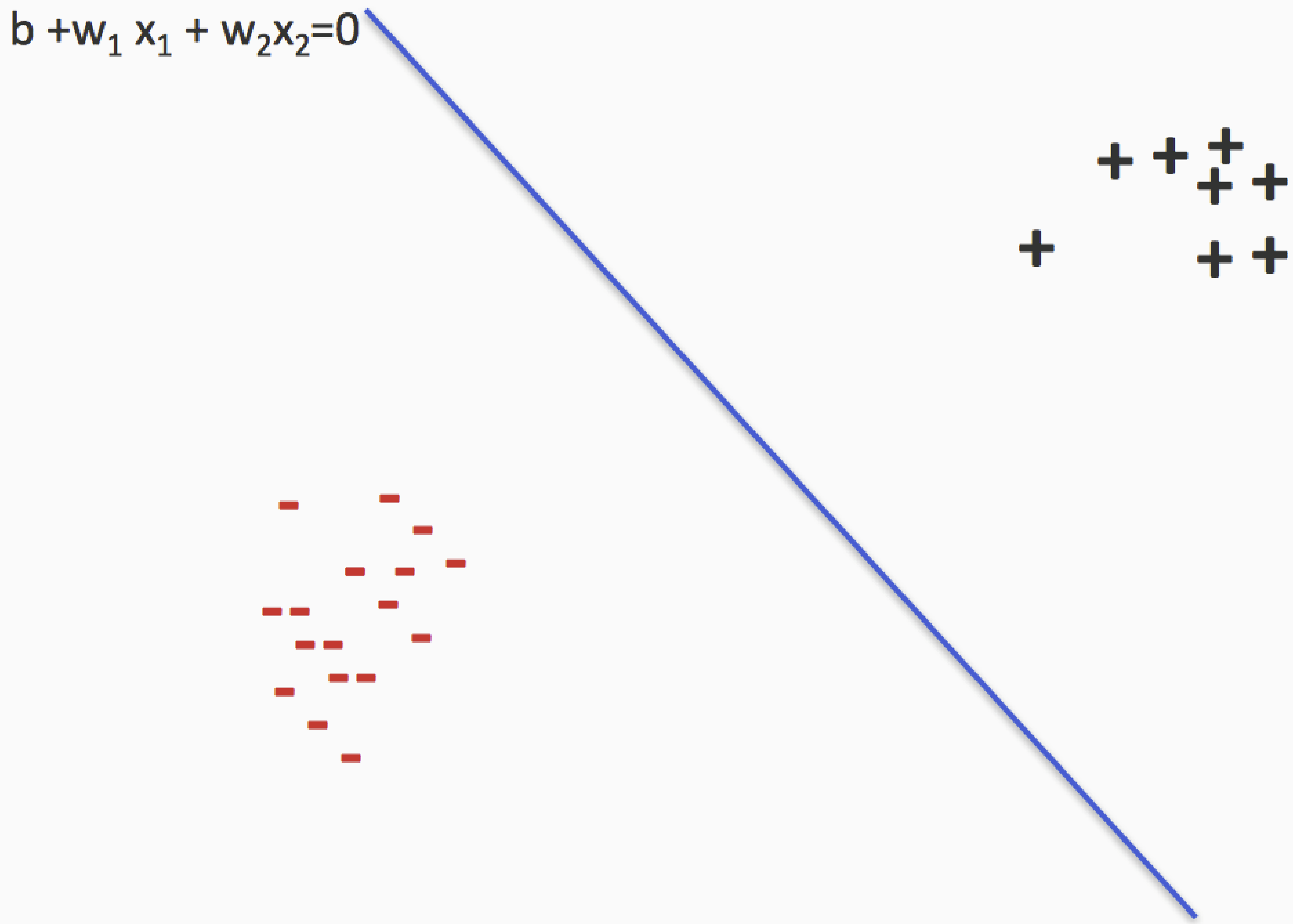
- 1943: McCullough and Pitts showed how linear threshold units can compute logical functions
- 1949: Hebb suggested a learning rule that has some physiological plausibility
- 1950s: Rosenblatt, the Peceptron algorithm for a single threshold neuron
- 1969: Minsky and Papert studied the neuron from a geometrical perspective
- 1980s: Convolutional neural networks (Fukushima, LeCun), the backpropagation algorithm (various)
- 2003-today: More compute, more data, deeper networks

Outline

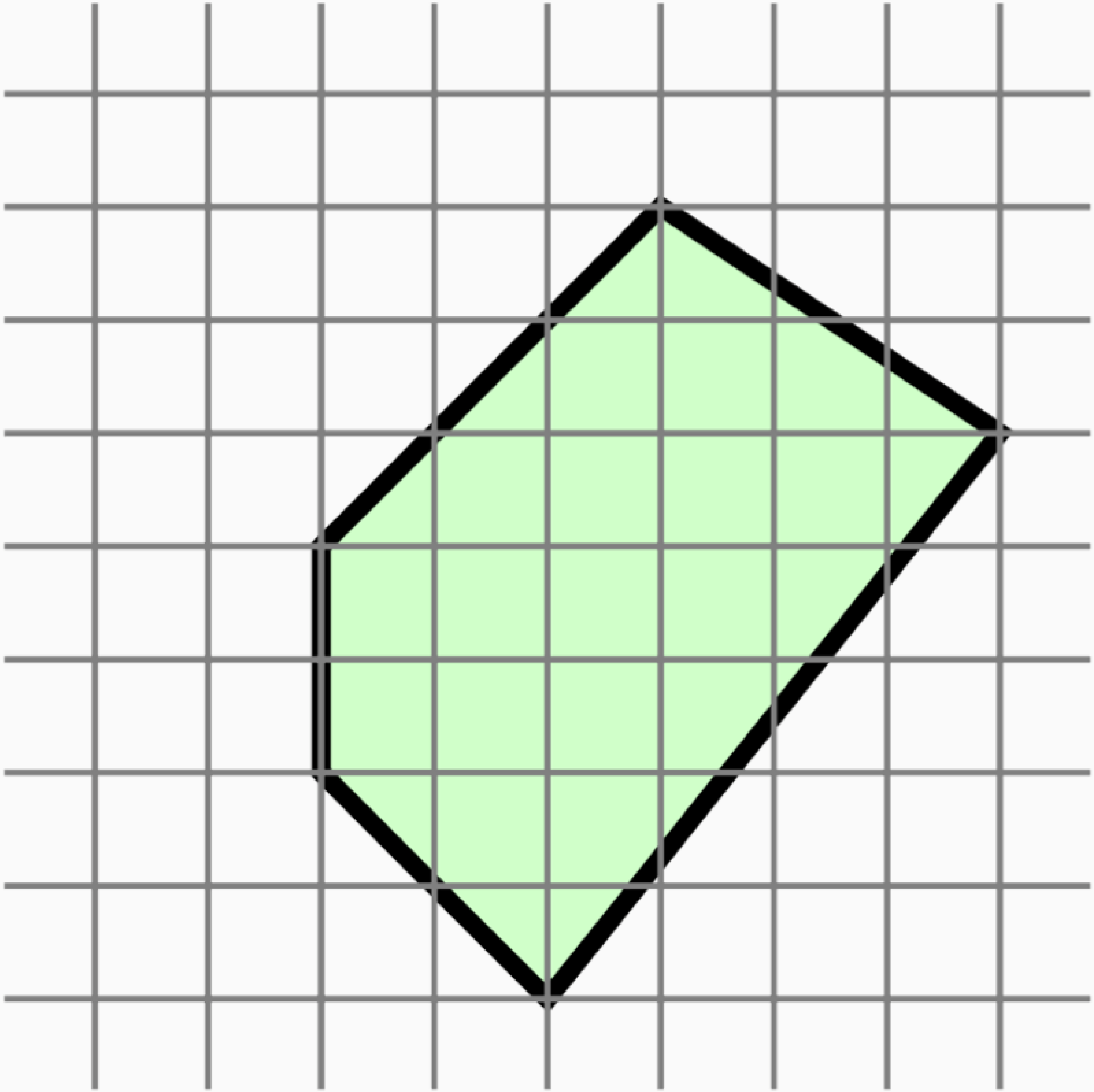
- Perceptron
- Stacking Linear Threshold Units
- Neural Networks
- Expressivity of Neural Networks ←
- Predicting with Neural Networks
- Backpropogation

A Single Neuron with Threshold Activation

Prediction = *sgn*$(b + w_1 x_1 + w_2 x_2)$



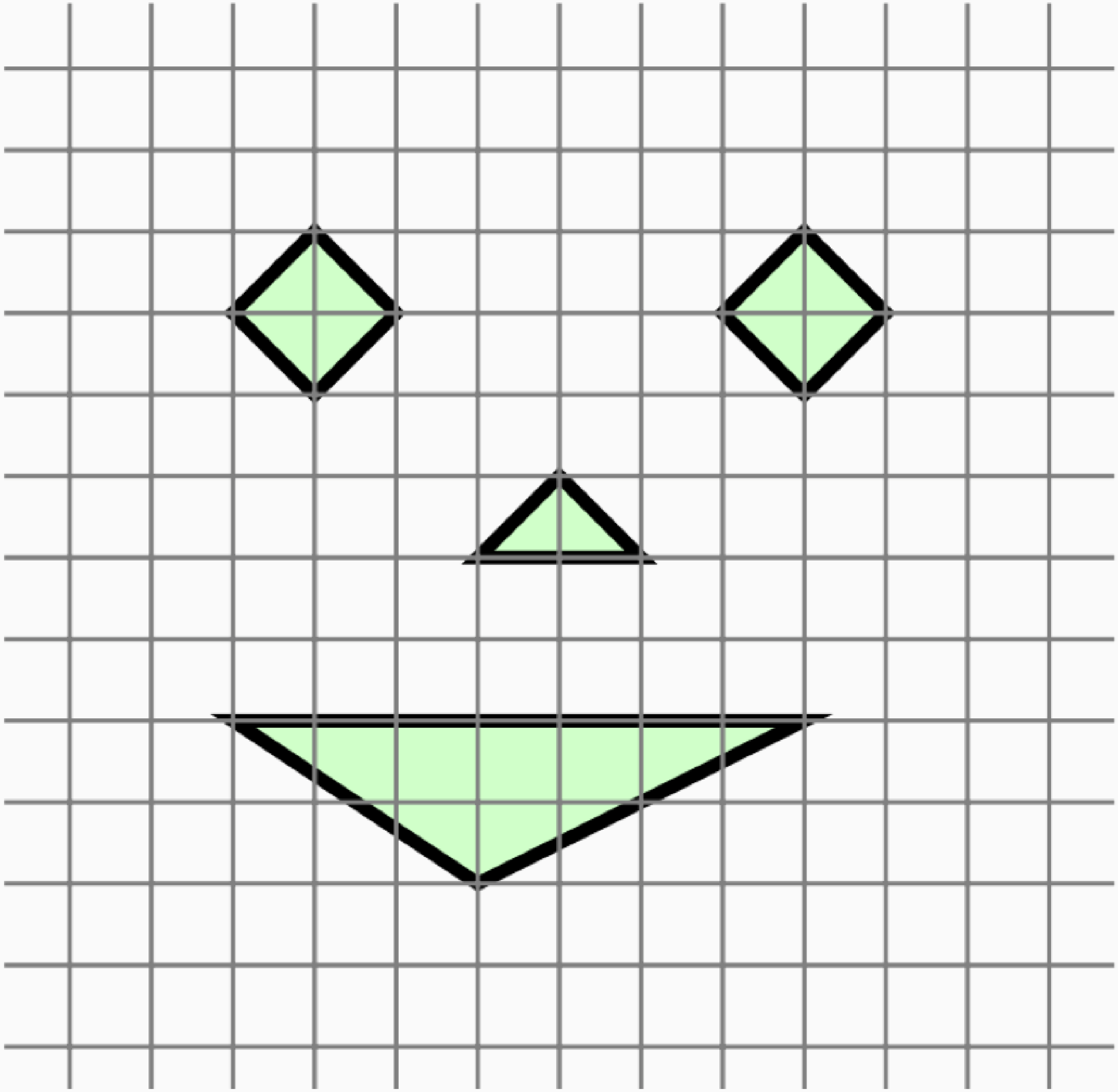
Two Layers with Threshold Activation



In general,
convex
polygons

Figure from [Shai Shalev-Shwartz and Shai Ben-David, 2014]

Three Layers with Threshold Activation



In general, unions of convex polygons

Figure from [Shai Shalev-Shwartz and Shai Ben-David, 2014]

NNs are Universal Function Approximators

- Any continuous function can be approximated to arbitrary accuracy using one hidden layer of sigmoid units [Cybenko 1989]

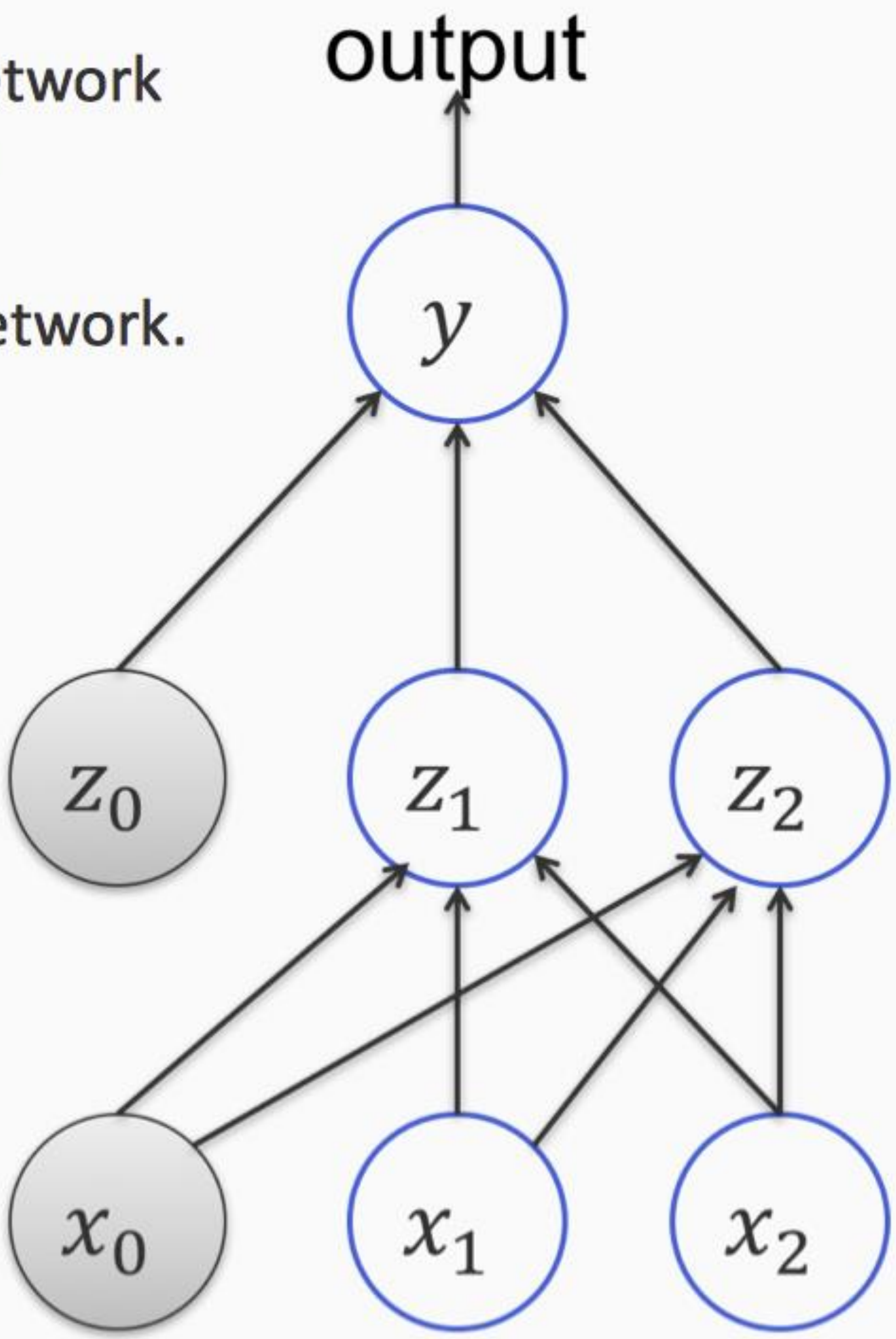
- Approximation error is insensitive to the choice of activation functions [DasGupta et al 1993]

Outline

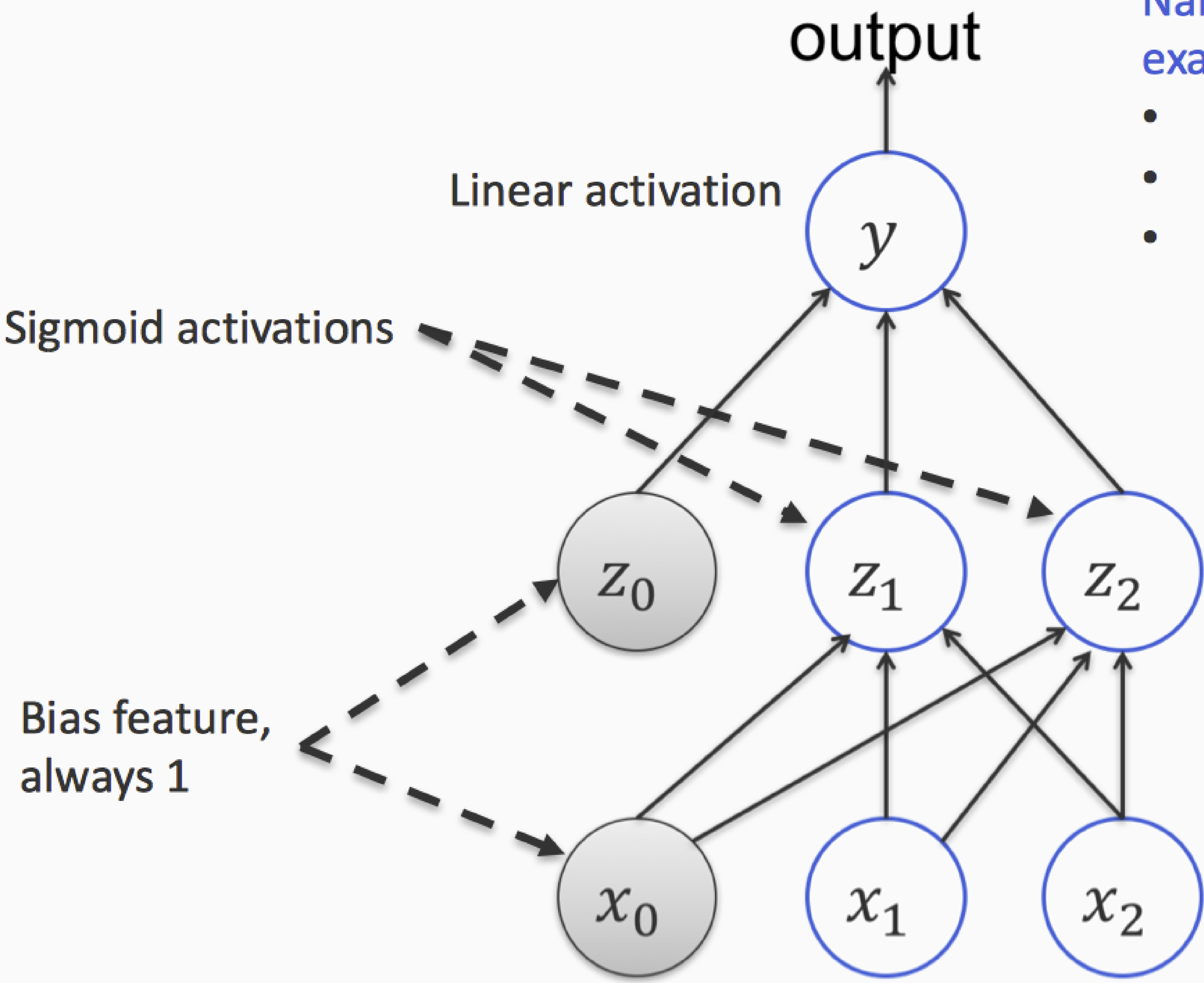
- Perceptron
- Stacking Linear Threshold Units
- Neural Networks
- Expressivity of Neural Networks
- Predicting with Neural Networks ←
- Backpropogation

Predicting with Neural Networks

We will use this example network as to introduce the general principle of how to make predictions with a neural network.



Predicting with Neural Networks



Naming conventions for this example

- Inputs: x
- Hidden: z
- Output: y

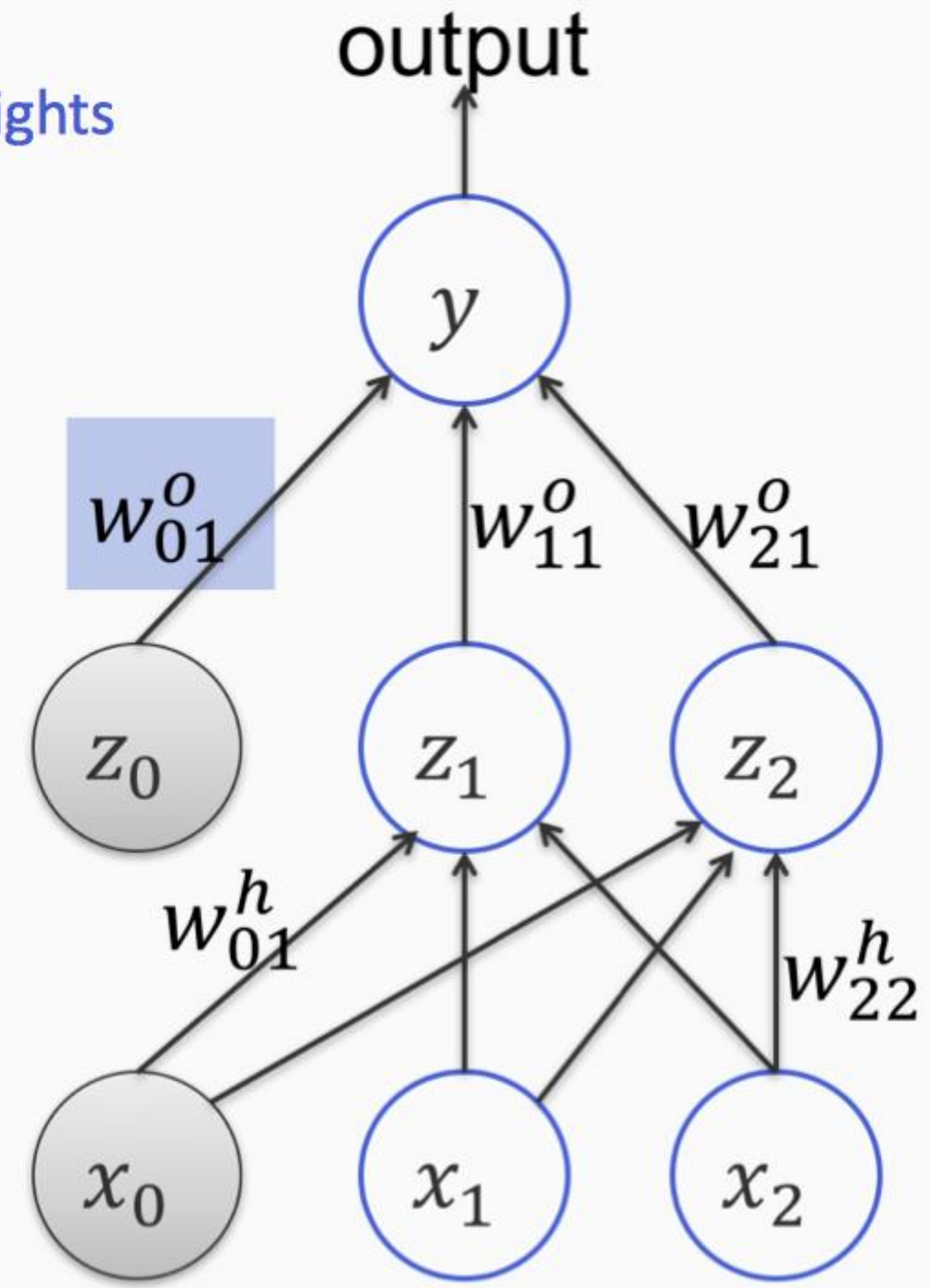
Predicting with Neural Networks

Naming Convention for Weights

$$w_{from,to}^{target\_layer}$$

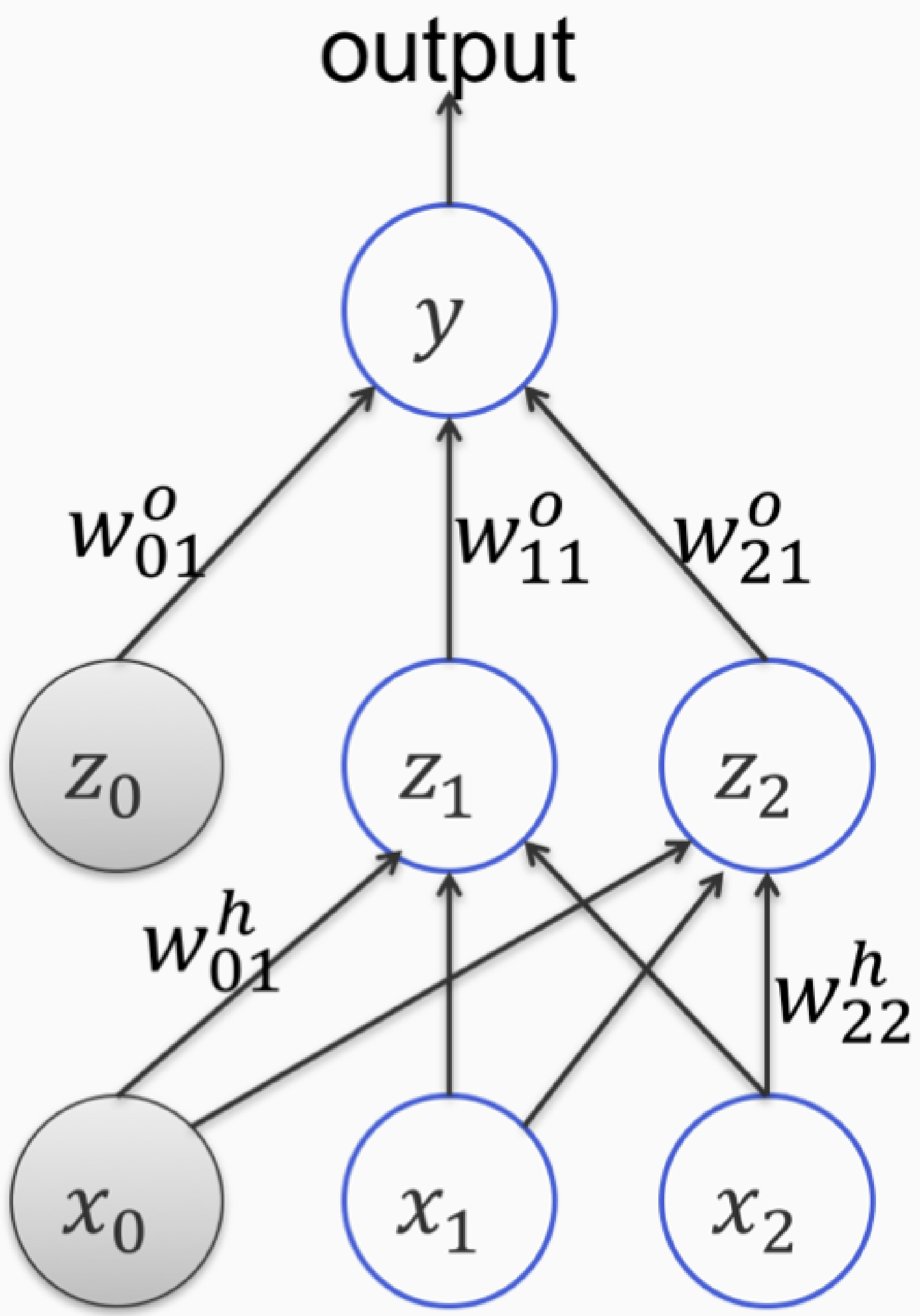
$$w_{01}^{o}$$

From neuron #0 to neuron #1 in output layer



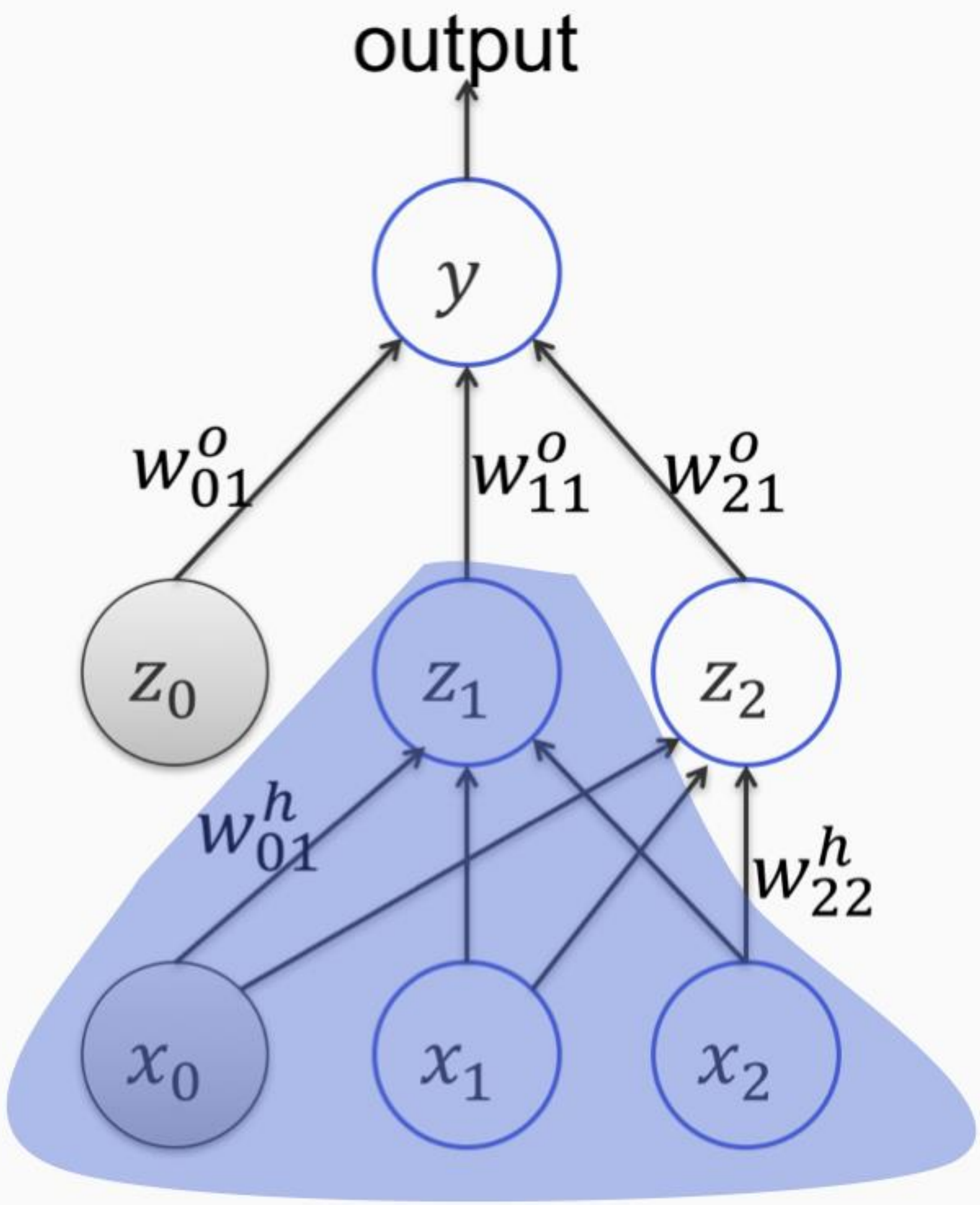
Predicting with Neural Nets: The Forward Pass

Given an input **x**, how is the output predicted



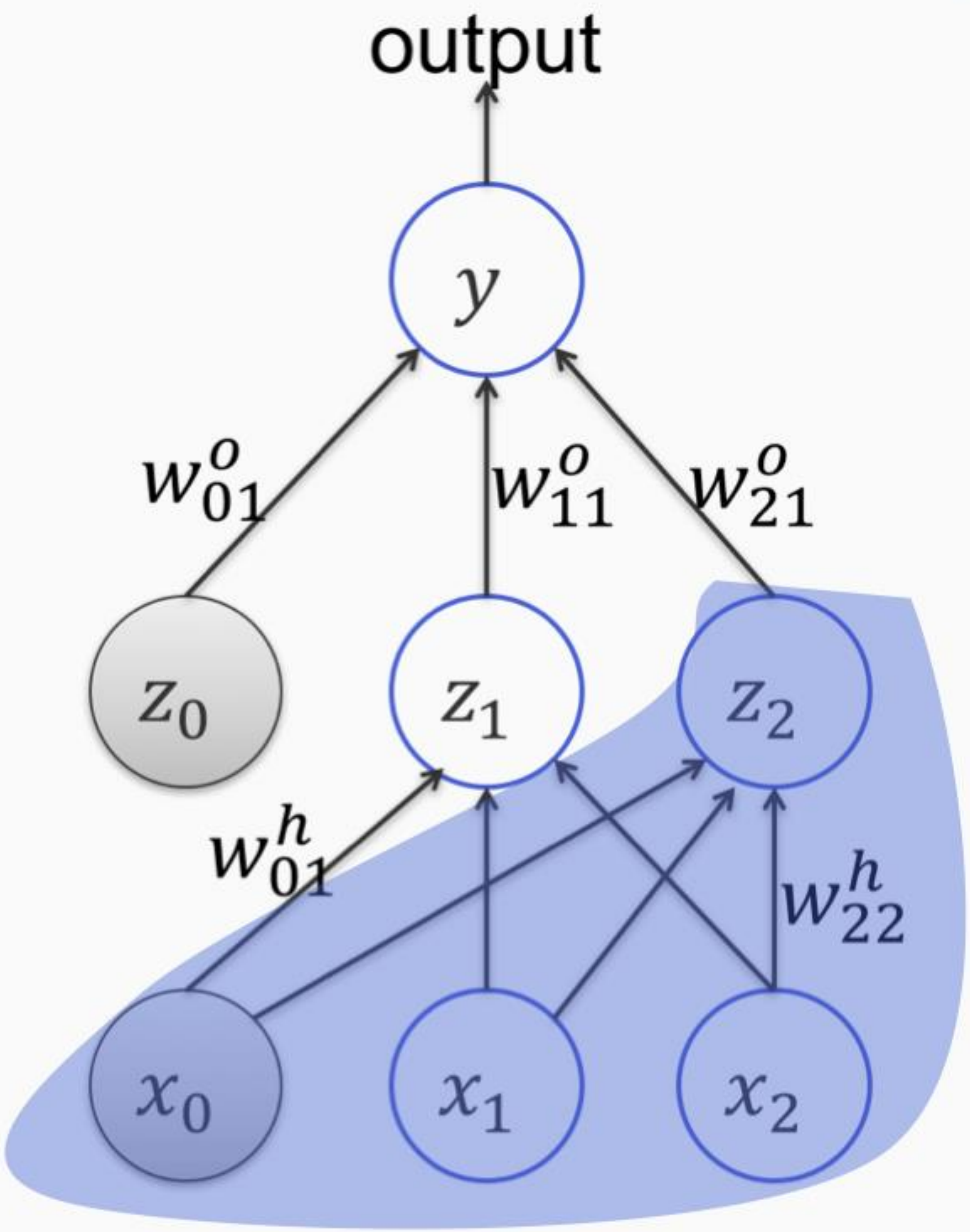
The Forward Pass

Given an input **x**, how is the output predicted



$$z_1 = \sigma(w_{01}^h + w_{11}^h x_1 + w_{21}^h x_2)$$

The Forward Pass

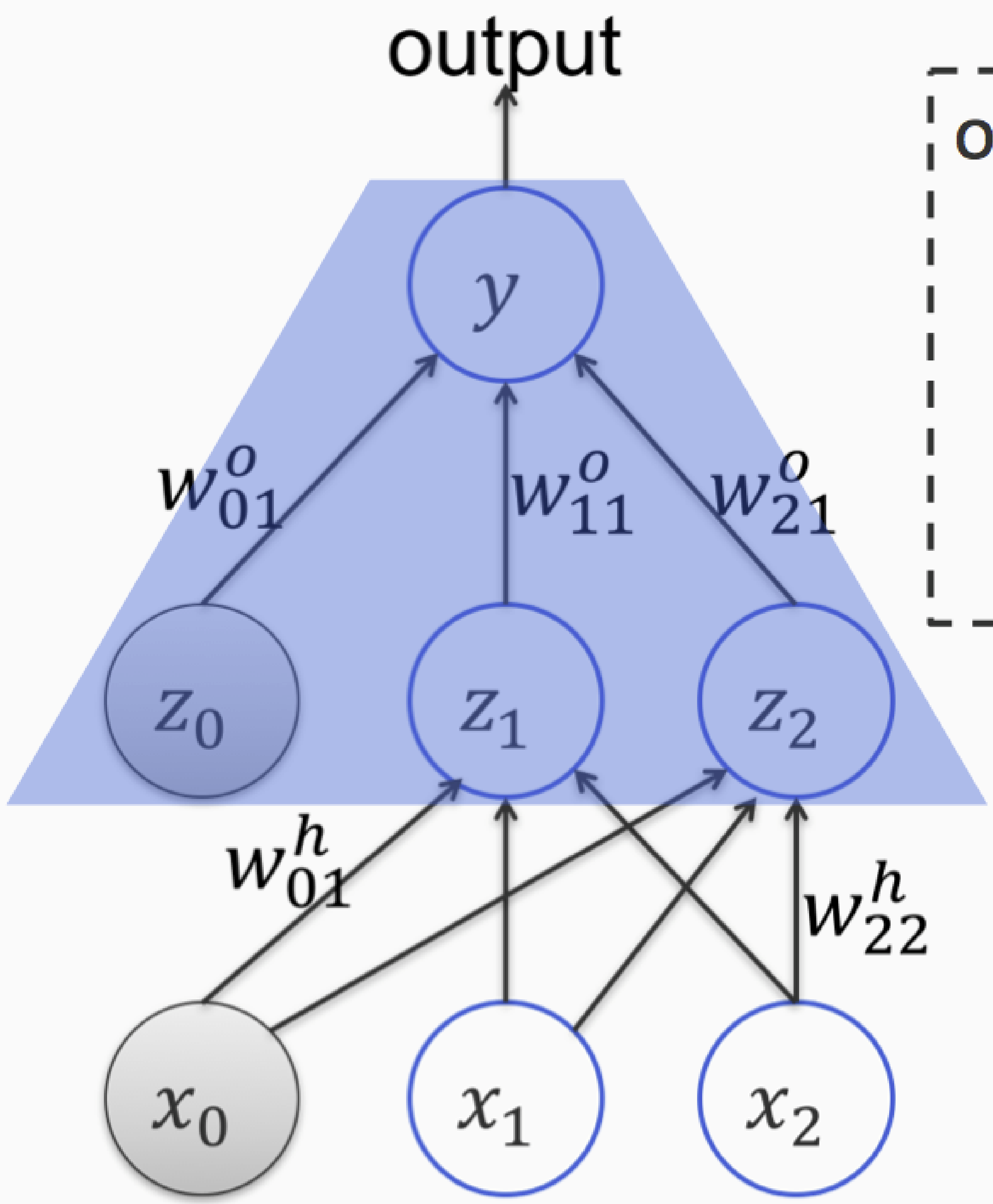


Given an input **x**, how is the output predicted

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

$$z_1 = \sigma(w_{01}^h + w_{11}^h x_1 + w_{21}^h x_2)$$

The Forward Pass



Given an input **x**, how is the output predicted

output $y = w^o_{01} + w^o_{11} z_1 + w^o_{21} z_2$

$$z_2 = \sigma(w^h_{02} + w^h_{12} x_1 + w^h_{22} x_2)$$

$$z_1 = \sigma(w^h_{01} + w^h_{11} x_1 + w^h_{21} x_2)$$

Outline

- Perceptron
- Stacking Linear Threshold Units
- Neural Networks
- Expressivity of Neural Networks
- Predicting with Neural Networks
- Backpropogation ⬅

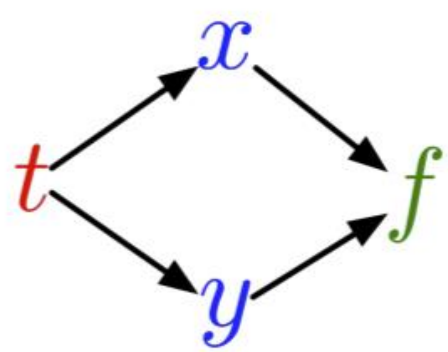
Backpropogation

- Smarter chain rule for derivatives
- What is chain rule:
  - Univariate case:

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t)) = \frac{\mathrm{d}f}{\mathrm{d}x} \cdot \frac{\mathrm{d}x}{\mathrm{d}t}$$

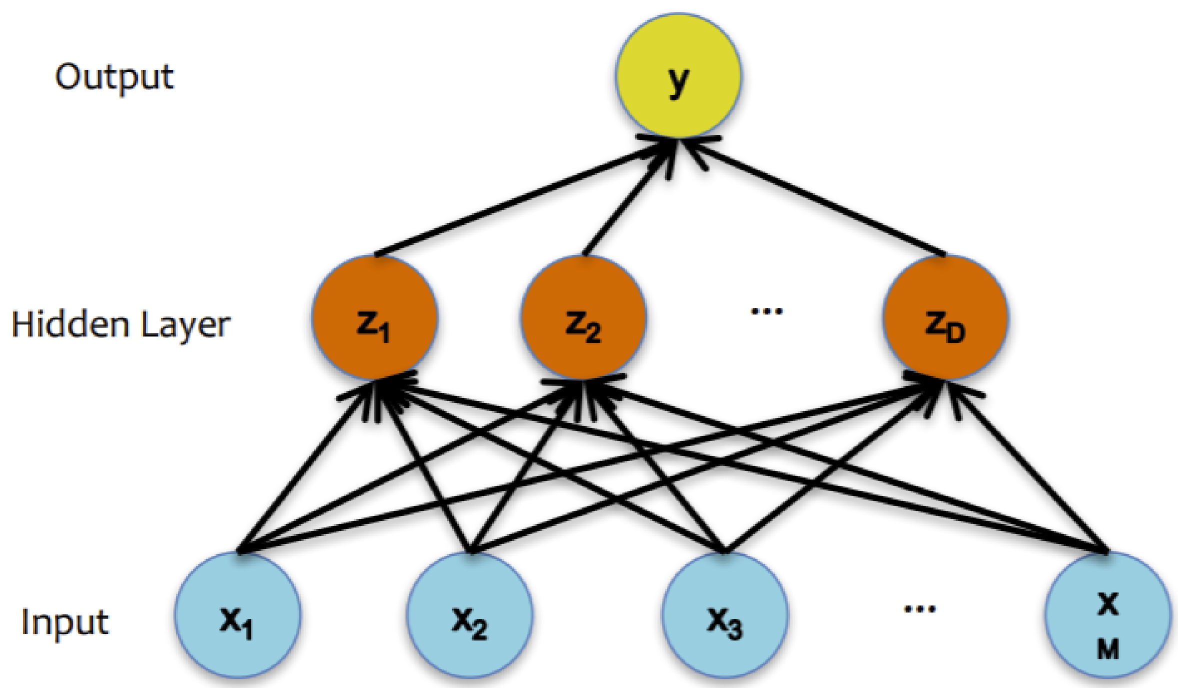
  - Multivariate case (Partial derivatives):

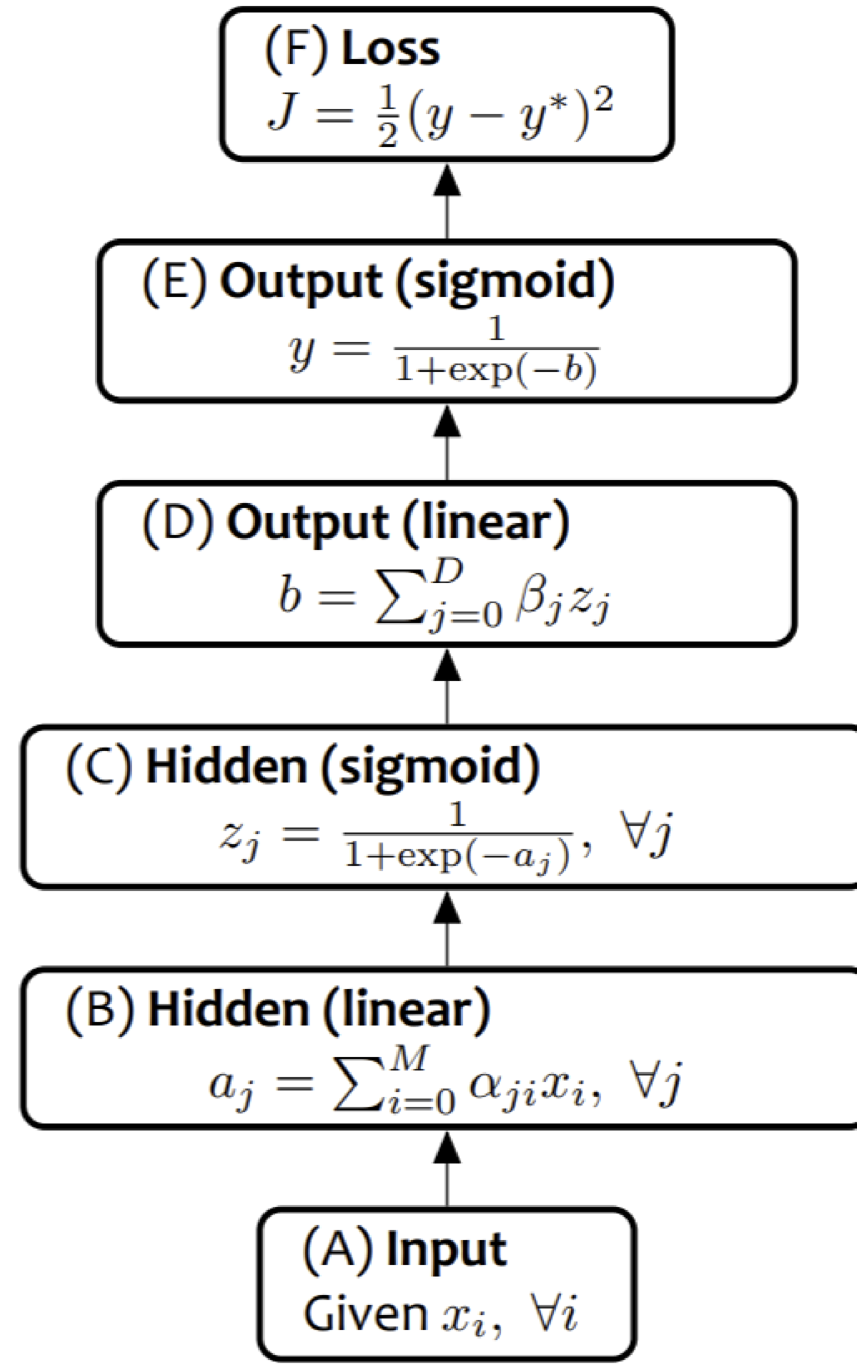
$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t), y(t)) = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$



These slides are from Roger Grosse

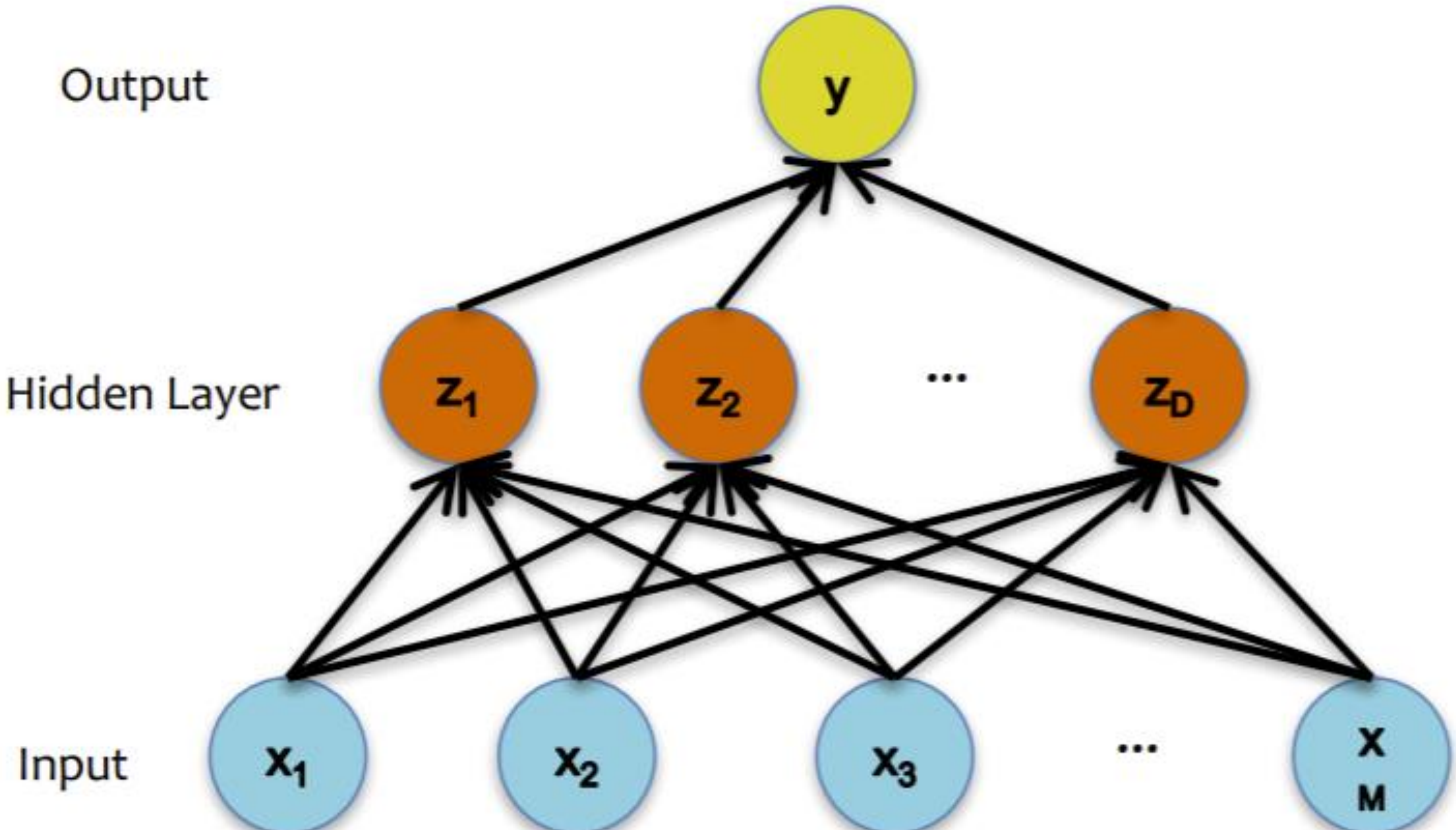
Backpropogation





These slides are from Matt Gromley

Backpropogation



These slides are from Matt Gromley

Backpropogtion

Forward

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

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

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

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

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

Backward

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

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

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

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

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

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

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

These slides are from Matt Gromley

Backpropagation

- Backprop is used to train the overwhelming majority of neural nets today.
  - Even optimization algorithms much fancier than gradient descent (e.g. second-order methods) use backprop to compute the gradients.
- Despite its practical success, backprop is believed to be neurally implausible. No evidence for biological signals analogous to error derivatives.
  - All the biologically plausible alternatives we know about learn much more slowly (on computers). So how on earth does the brain learn?

Slide from Roger Grosse

Take-Home Messages

- Stacking Linear Threshold Units
- Neural Networks
- Expressivity of Neural Networks
- Predicting with Neural Networks
- Backprop is chain rule with some book-keeping