Neural Networks: Introduction

Machine Learning

THE UNIVERSITY OF UTAH

Based on slides and material from Geoffrey Hinton, Richard Socher, Dan Roth, Yoav Goldberg, Shai Shalev-Shwartz and Shai Ben-David, and others

Where are we?

Learning algorithms

- Decision Trees
- Perceptron
- AdaBoost
- Support Vector Machines
- Logistic Regression

Where are we?

Learning algorithms

- Decision Trees
- Perceptron
- AdaBoost
- Support Vector Machines
- Logistic Regression

General learning principles

- Overfitting
- Mistake-bound learning
- PAC learning, sample complexity
- Hypothesis choice & VC dimensions
- Training and generalization errors
- Regularized Empirical Loss Minimization
- Bayesian Learning

Where are we?

Learning algorithms

- Decision Trees
- Perceptron
- AdaBoost
- Support Vector Machines
- Logistic Regression

Produce *linear classifiers*

General learning principles

- Overfitting
- Mistake-bound learning
- PAC learning, sample complexity
- Hypothesis choice & VC dimensions
- Training and generalization errors
- Regularized Empirical Loss Minimization
- Bayesian Learning

Where are we?

Learning algorithms

- Decision Trees
- Perceptron
- AdaBoost
- Support Vector Machines
- Logistic Regression

Produce *linear classifiers*

General learning principles

- Overfitting
- Mistake-bound learning
- PAC learning, sample complexity
- Hypothesis choice & VC dimensions
- Training and generalization errors
- Regularized Empirical Loss Minimization
- Bayesian Learning

Not really resolved

What if we want to train non-linear classifiers?

Where do the features come from?

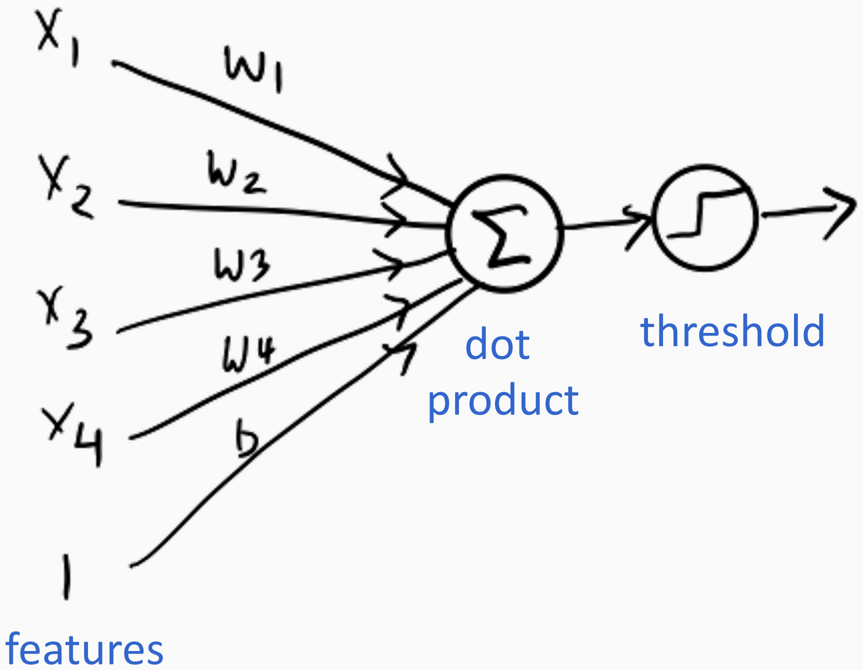
Neural Networks

- What is a neural network?
- Predicting with a neural network
- Training neural networks
- Practical concerns

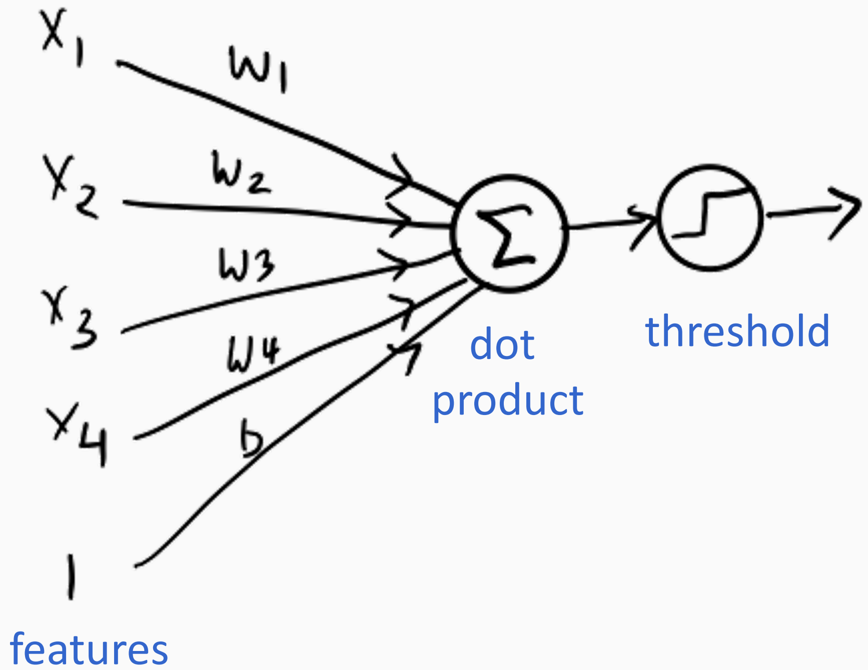
This lecture

- What is a neural network?
  - The hypothesis class
  - Structure, expressiveness
- Predicting with a neural network
- Training neural networks
- Practical concerns

We have seen linear threshold units



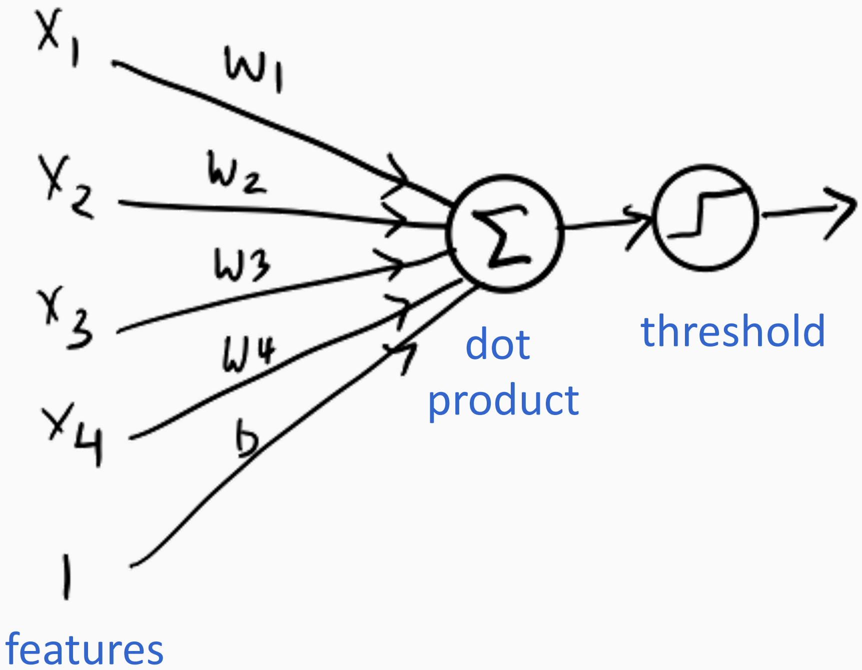
We have seen linear threshold units



Prediction

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

We have seen linear threshold units



Prediction

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

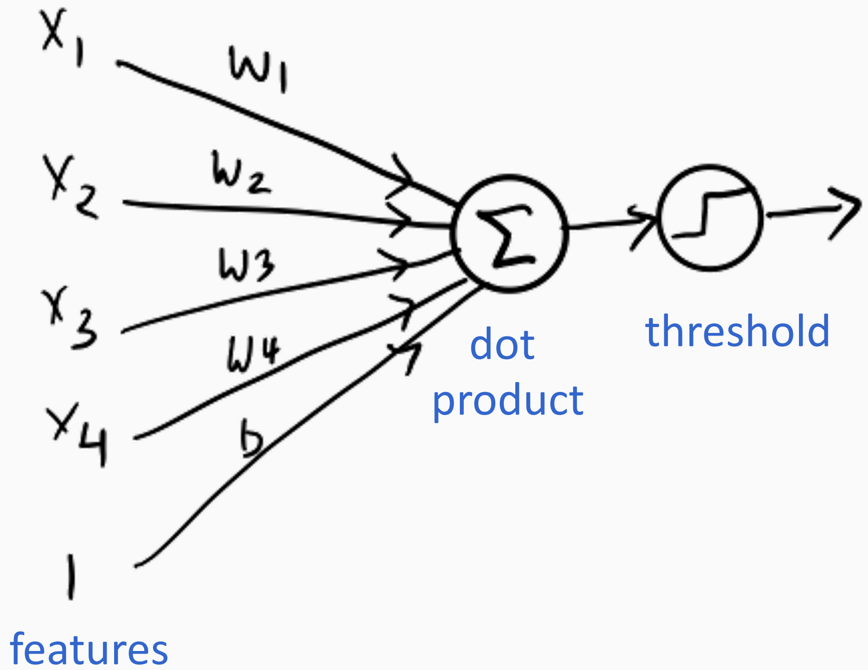
Learning

various algorithms

perceptron, SVM, logistic regression,…

in general, minimize loss

We have seen linear threshold units



Prediction

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

Learning

various algorithms

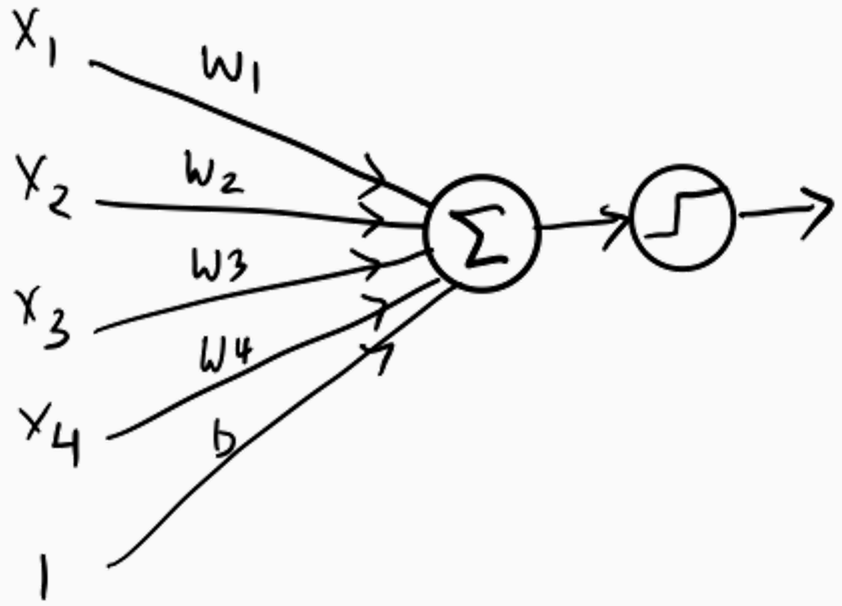
perceptron, SVM, logistic regression,…

in general, minimize loss

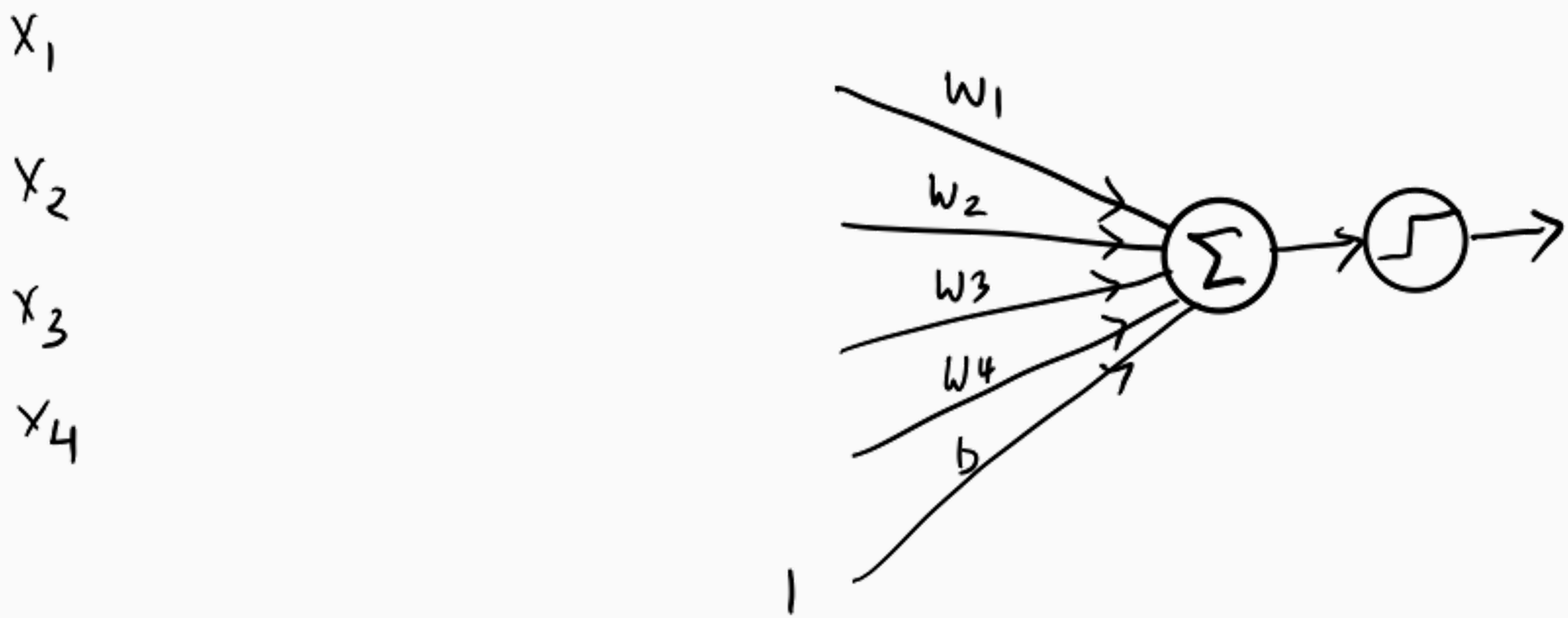
But where do these input features come from?

What if the features were outputs of another classifier?

Features from classifiers

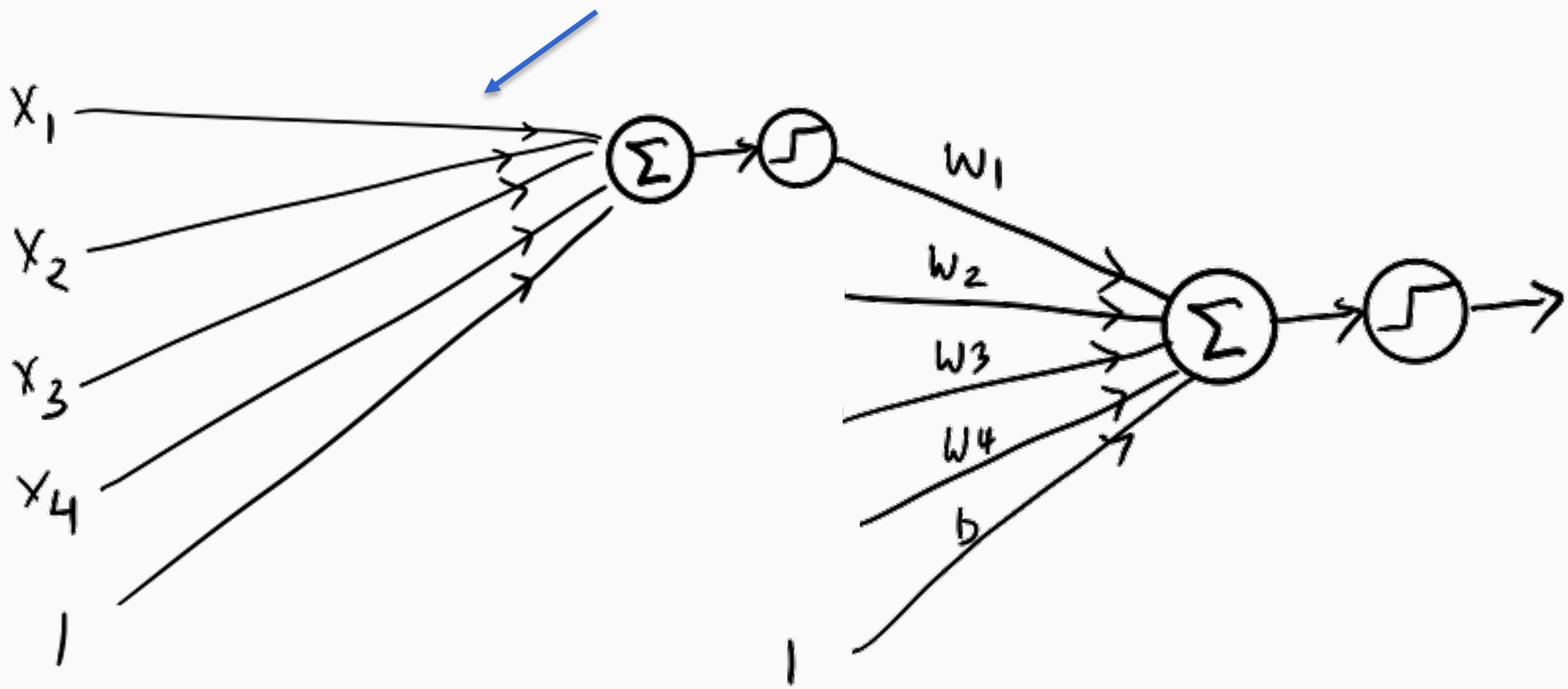


Features from classifiers

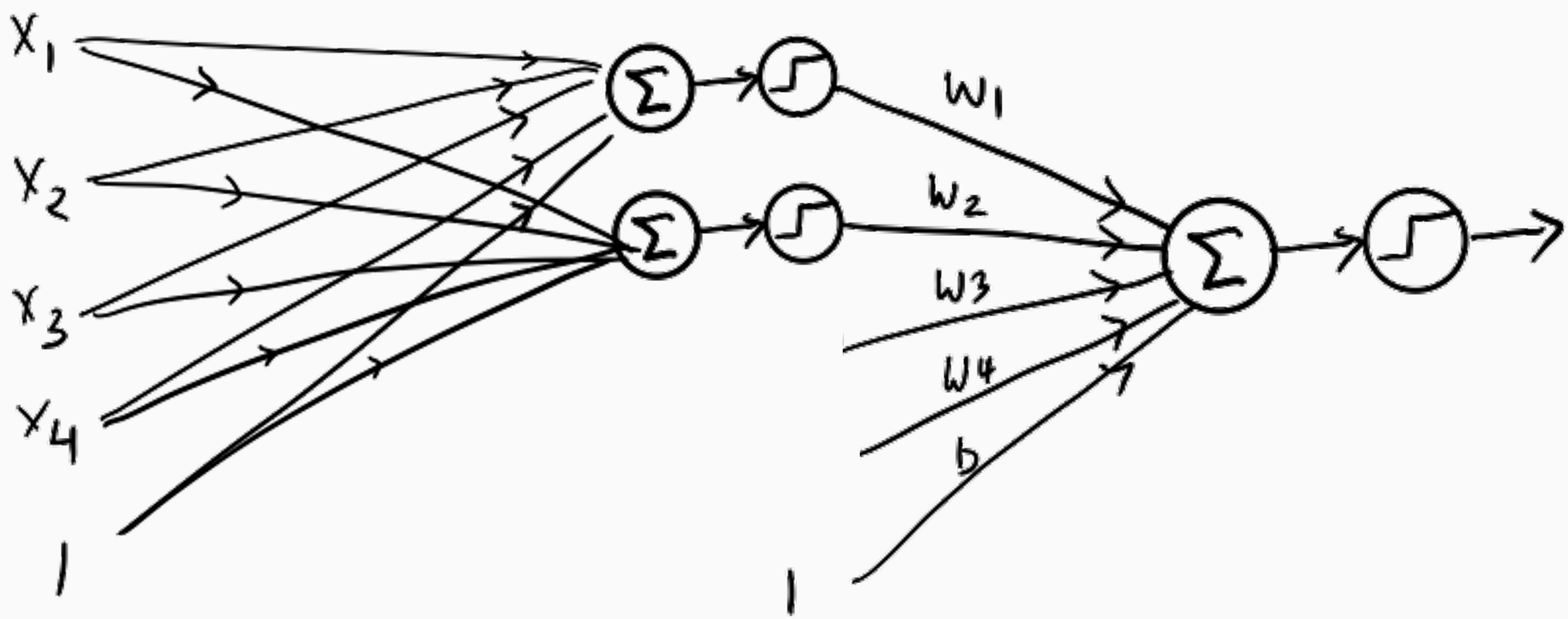


Features from classifiers

Each of these connections have their own weights as well

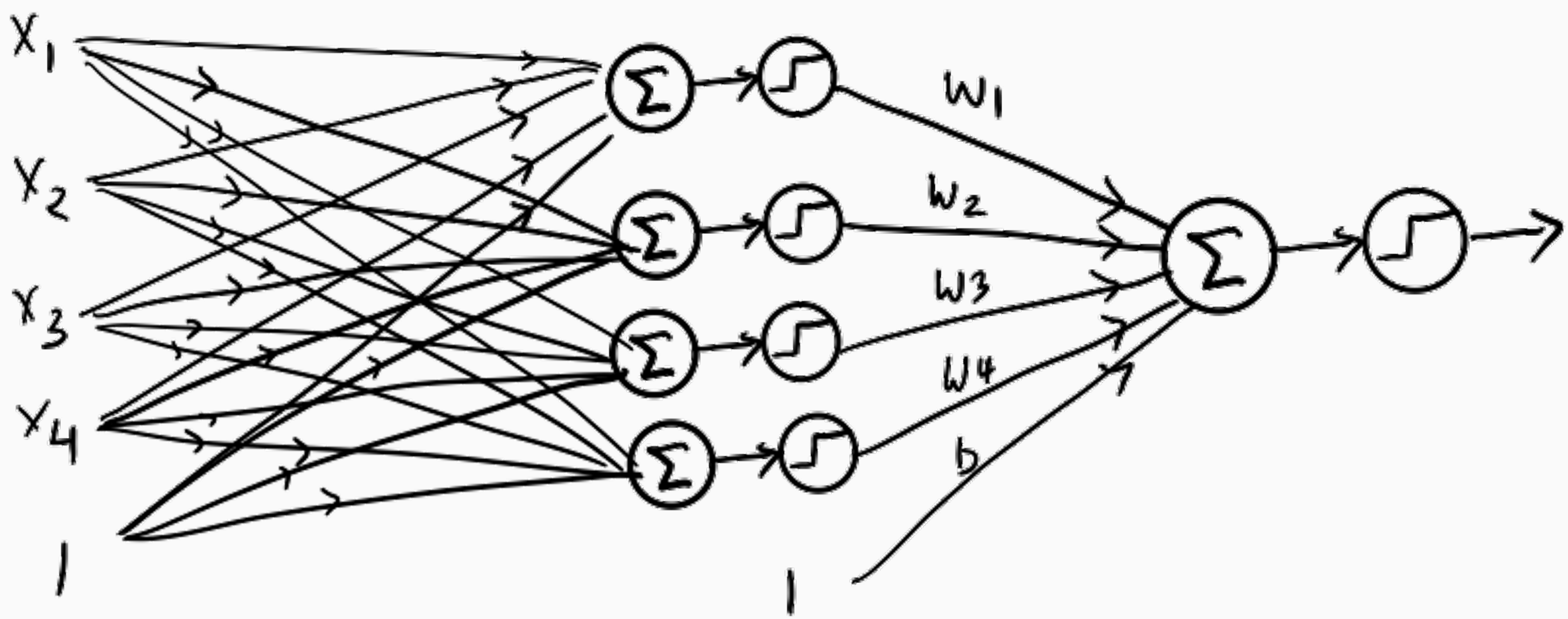


Features from classifiers



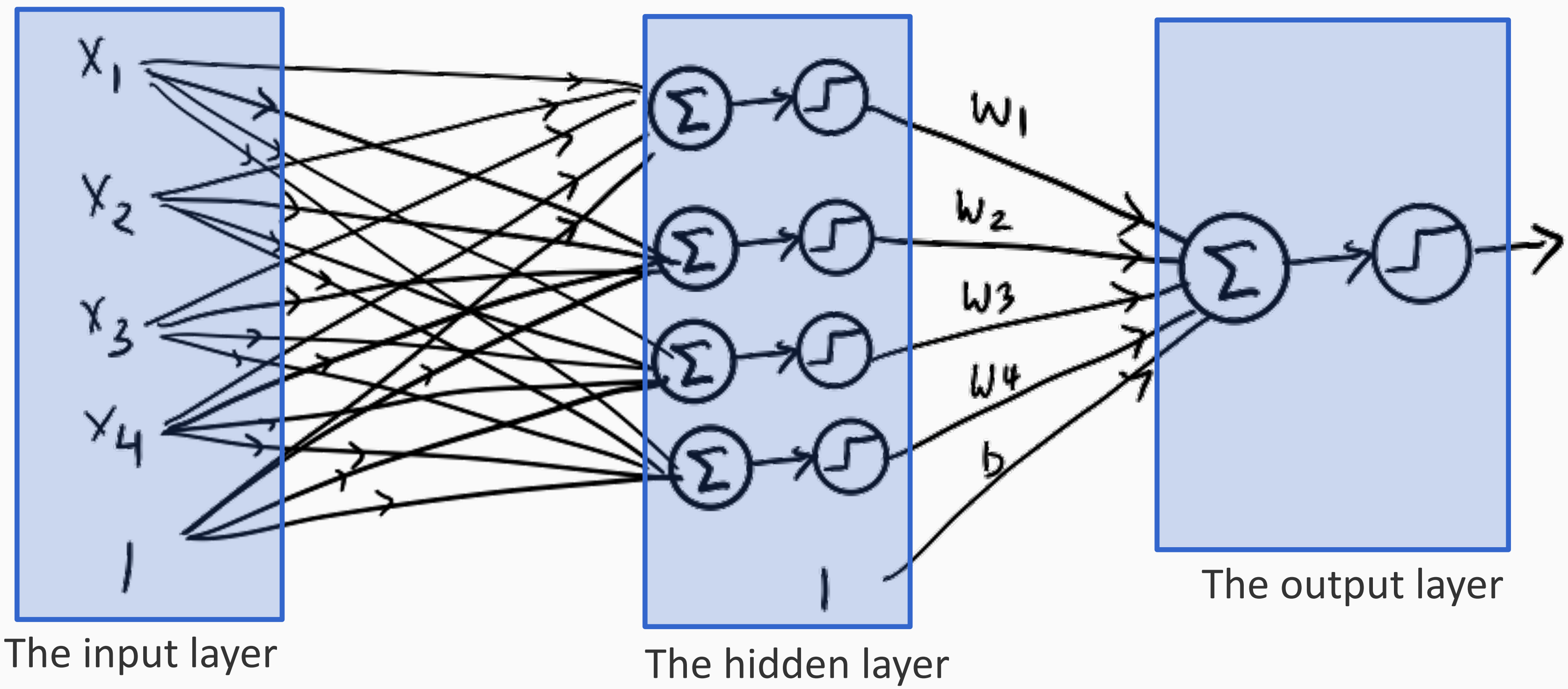
Features from classifiers

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



Features from classifiers

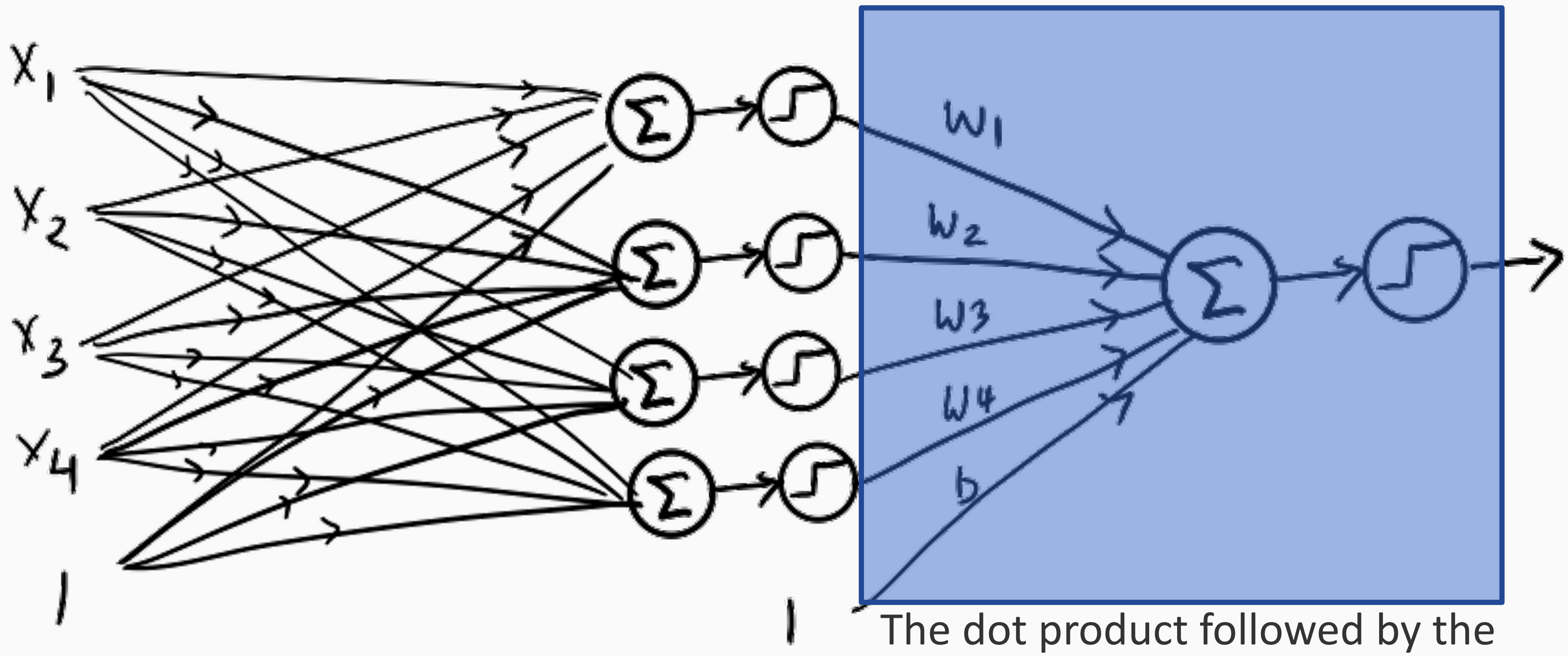
This is a **two layer** feed forward neural network



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

Features from classifiers

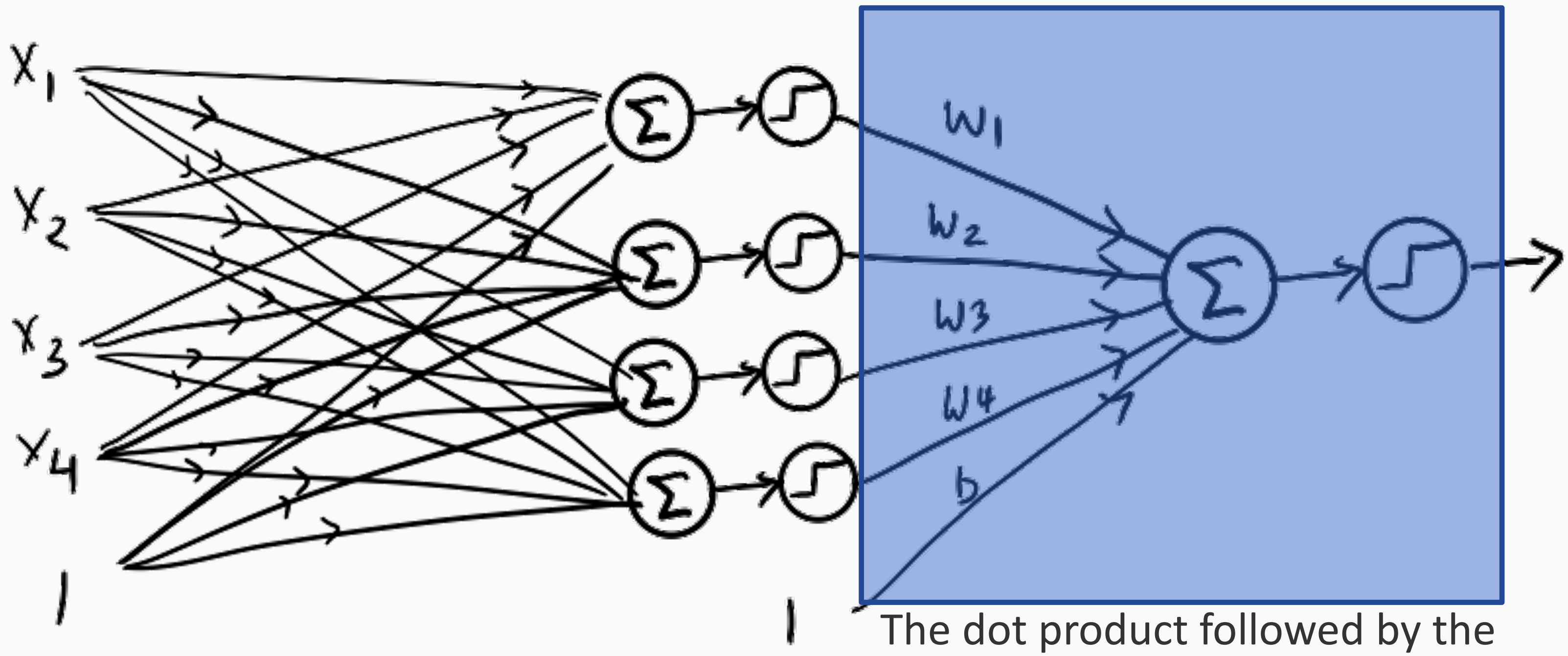
This is a **two layer** feed forward neural network



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

Features from classifiers

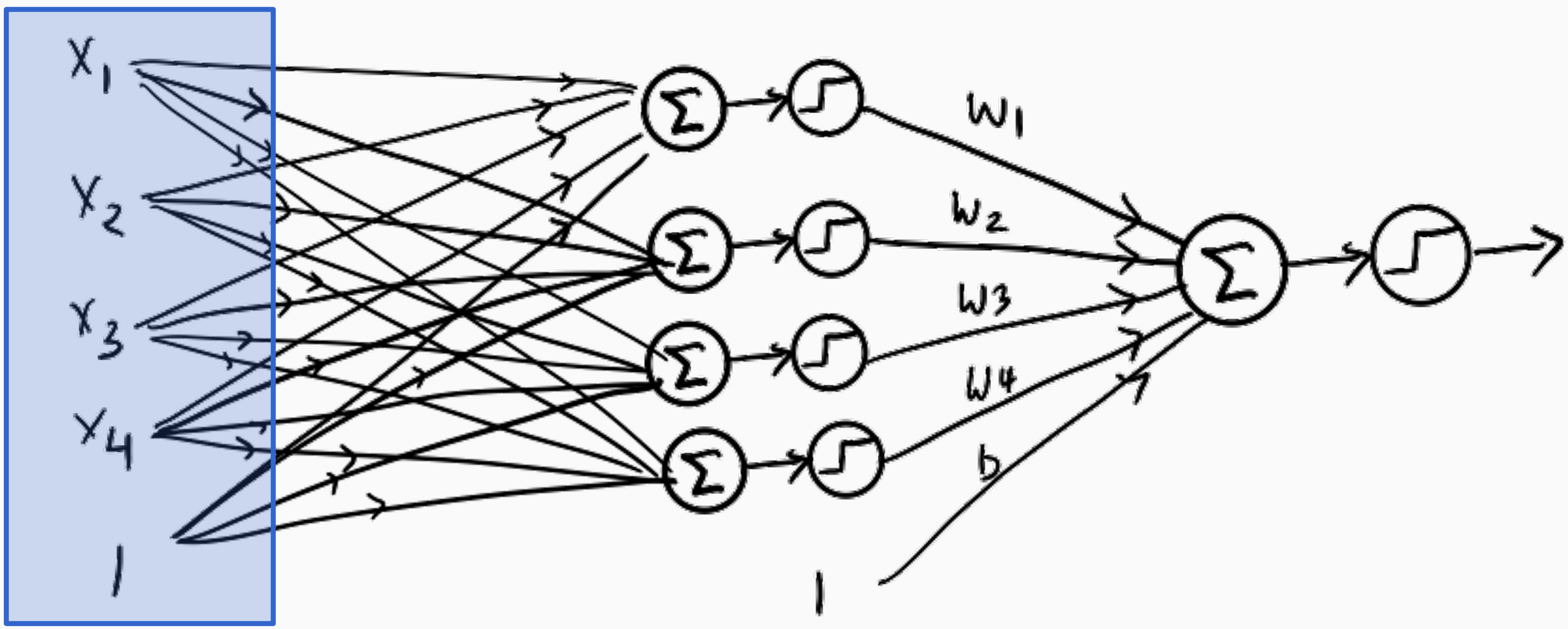
This is a **two layer** feed forward neural network



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

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

But where do the inputs come from?



The input layer

What if the inputs were the outputs of a classifier?

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

Let us try to formalize this

Neural networks

A robust approach for approximating real-valued, discrete-valued or vector valued functions

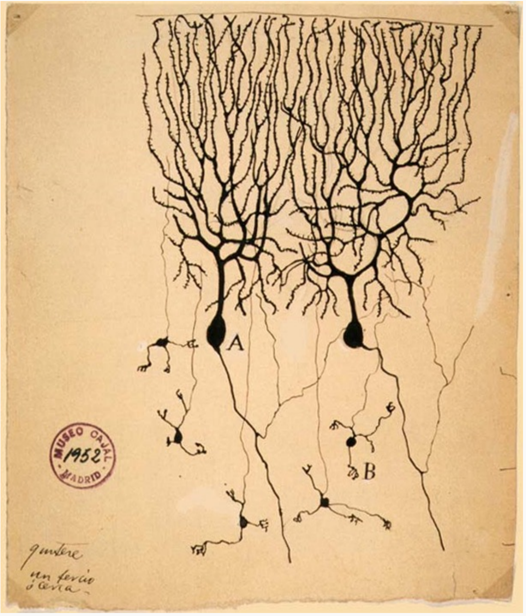
Among the most effective **general purpose** supervised learning methods currently known

- Especially for *complex and hard to interpret data* such as real-world sensory data

The Backpropagation algorithm to enable gradient-based learning of neural networks has been shown successful in many practical problems

- Across various application domains

Biological neurons

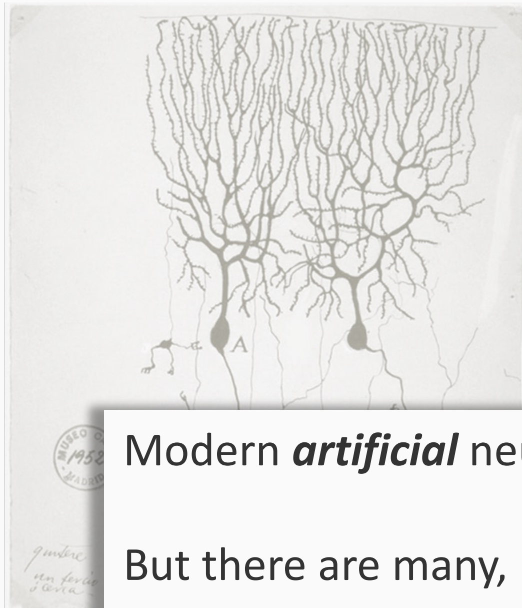


The first drawing of a brain cells by Santiago Ramón y Cajal in 1899

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

Biological neurons



The
cells
Caja…

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

Modern ***artificial*** neurons are **loosely** “inspired” by biological neurons

But there are many, many fundamental differences

Don’t take the similarity seriously (as also other AI related hyperbole in the news or social media)

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***

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

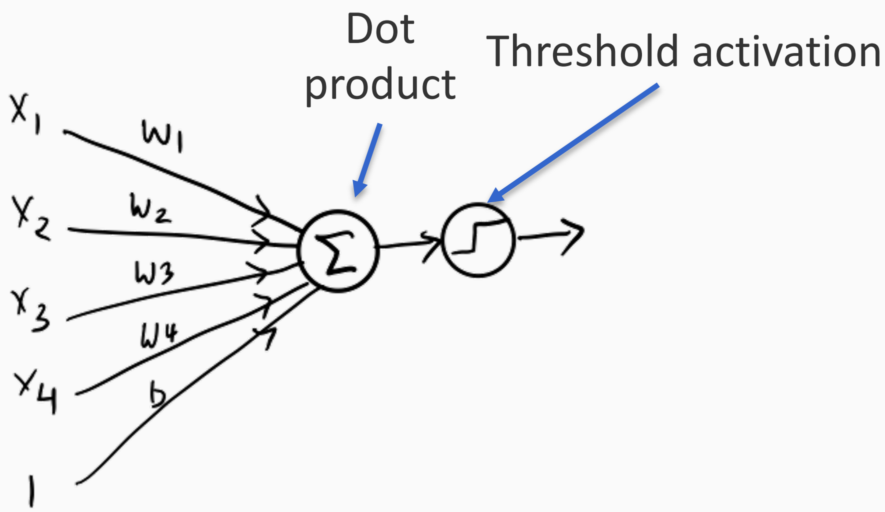
Artificial neurons

*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***

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



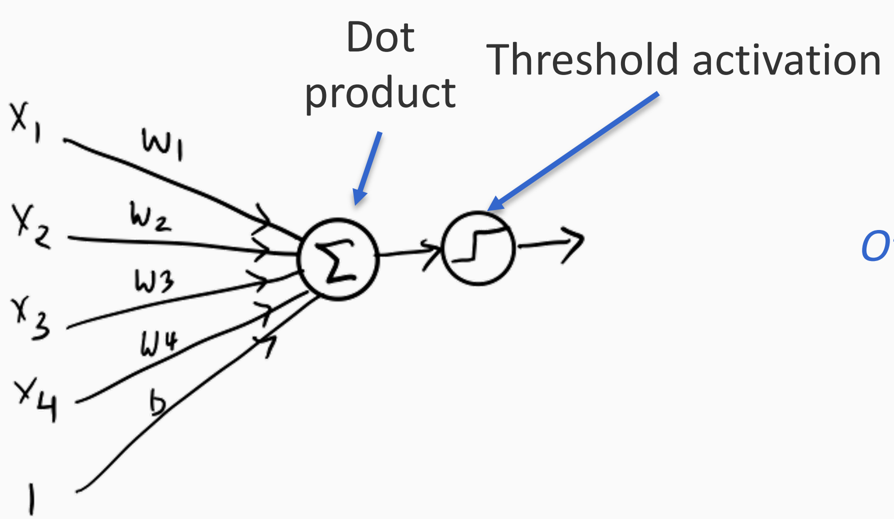
Artificial neurons

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***

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



Other activations are possible

Activation functions

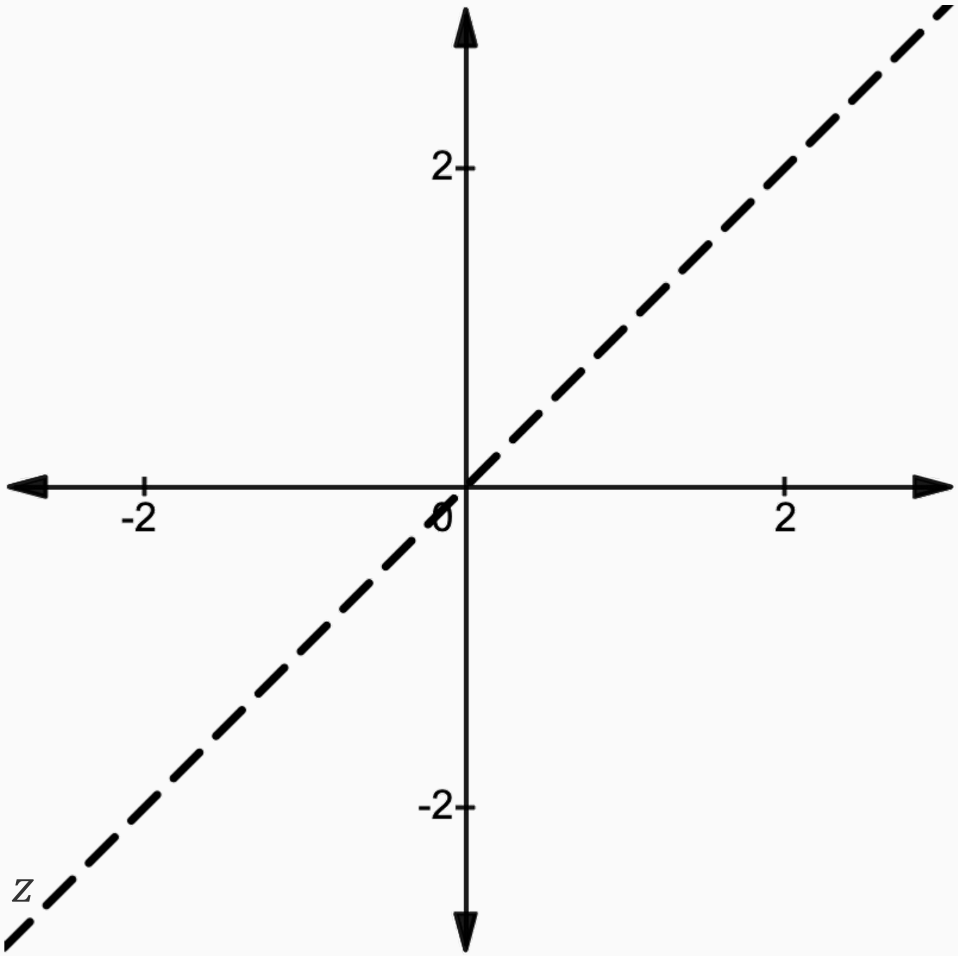
Also called transfer functions

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

Name of the neuron	**Activation function:** $\text{activation}(z)$
Linear unit	z (i.e. no change to the input)
Threshold/sign unit	$\text{sgn}(z)$
Sigmoid unit	$\dfrac{1}{1+\exp(-z)}$
Rectified linear unit (ReLU)	$\max(0, z)$
Tanh unit	$\tanh(z)$

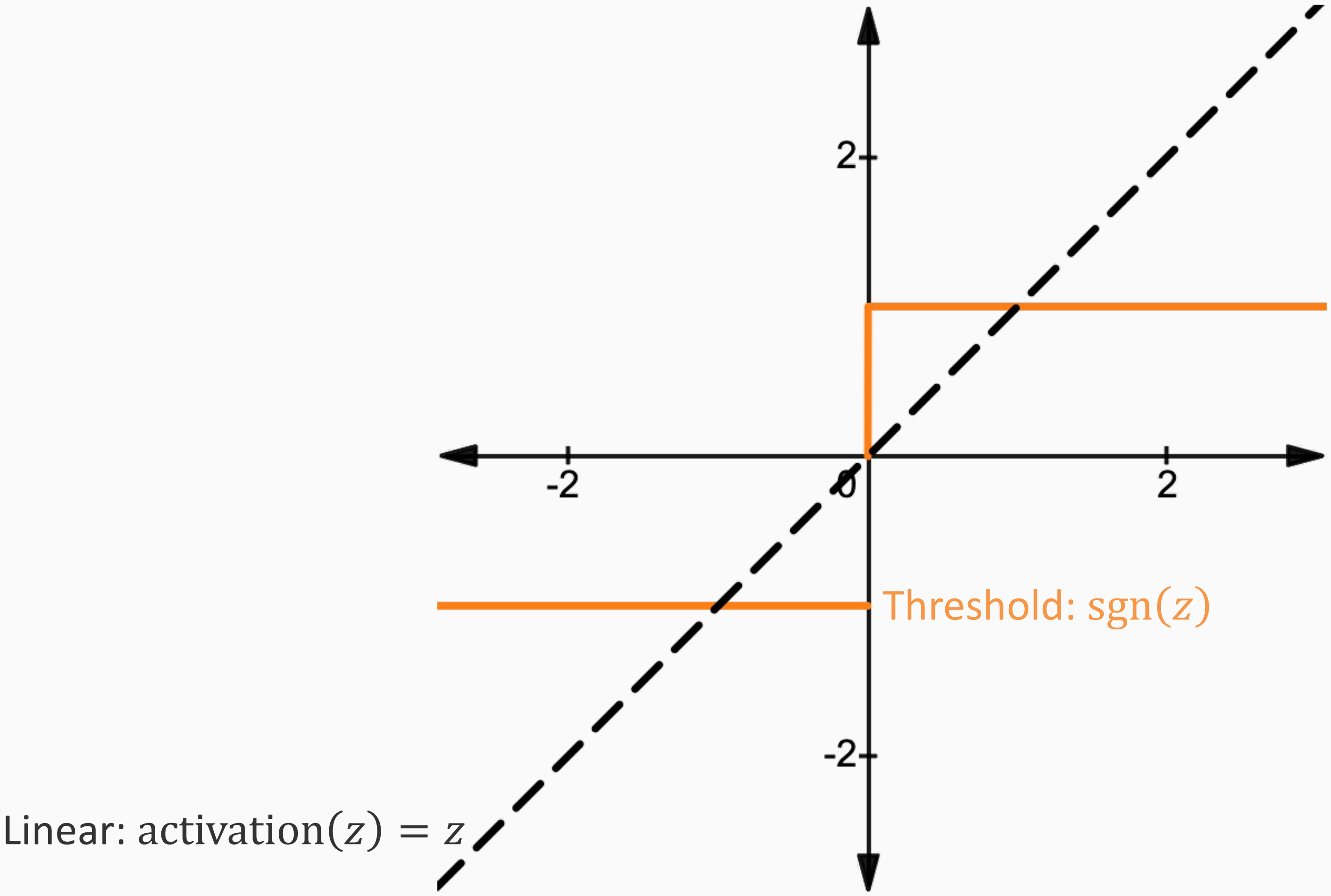
Many more activation functions exist (sinusoid, sinc, gaussian, polynomial…)

Activation functions illustrated

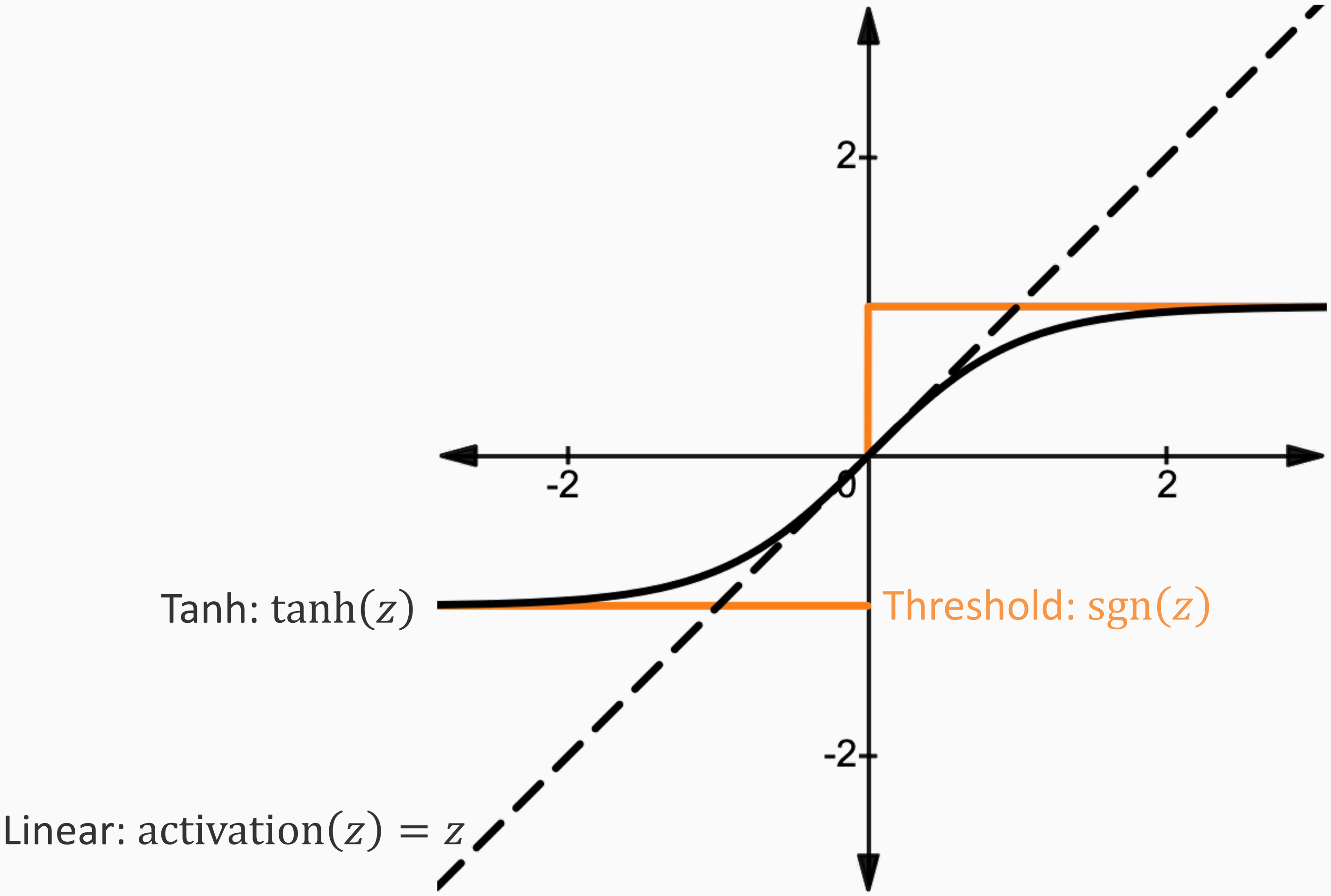


Linear: $\text{activation}(z) = z$

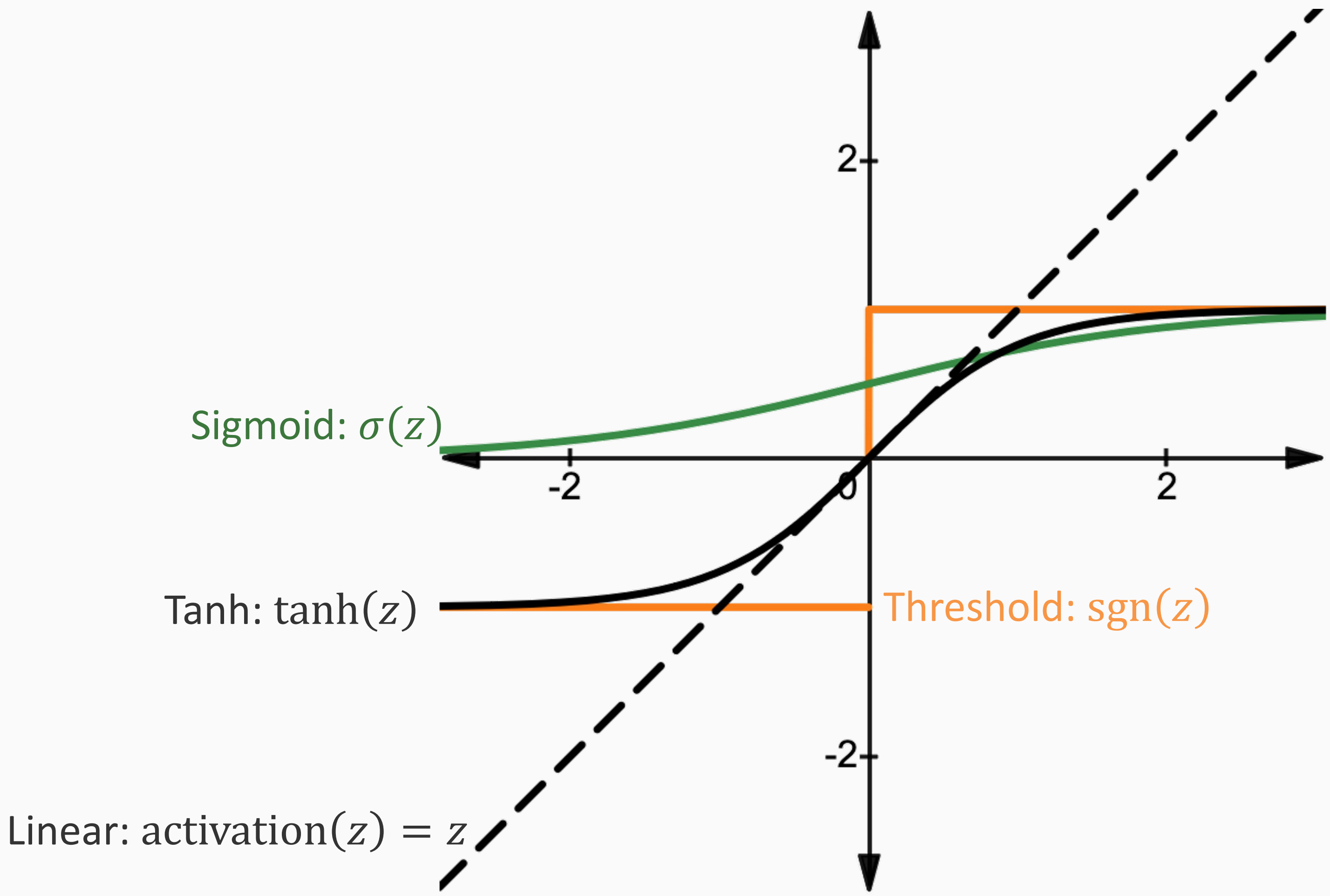
Activation functions illustrated



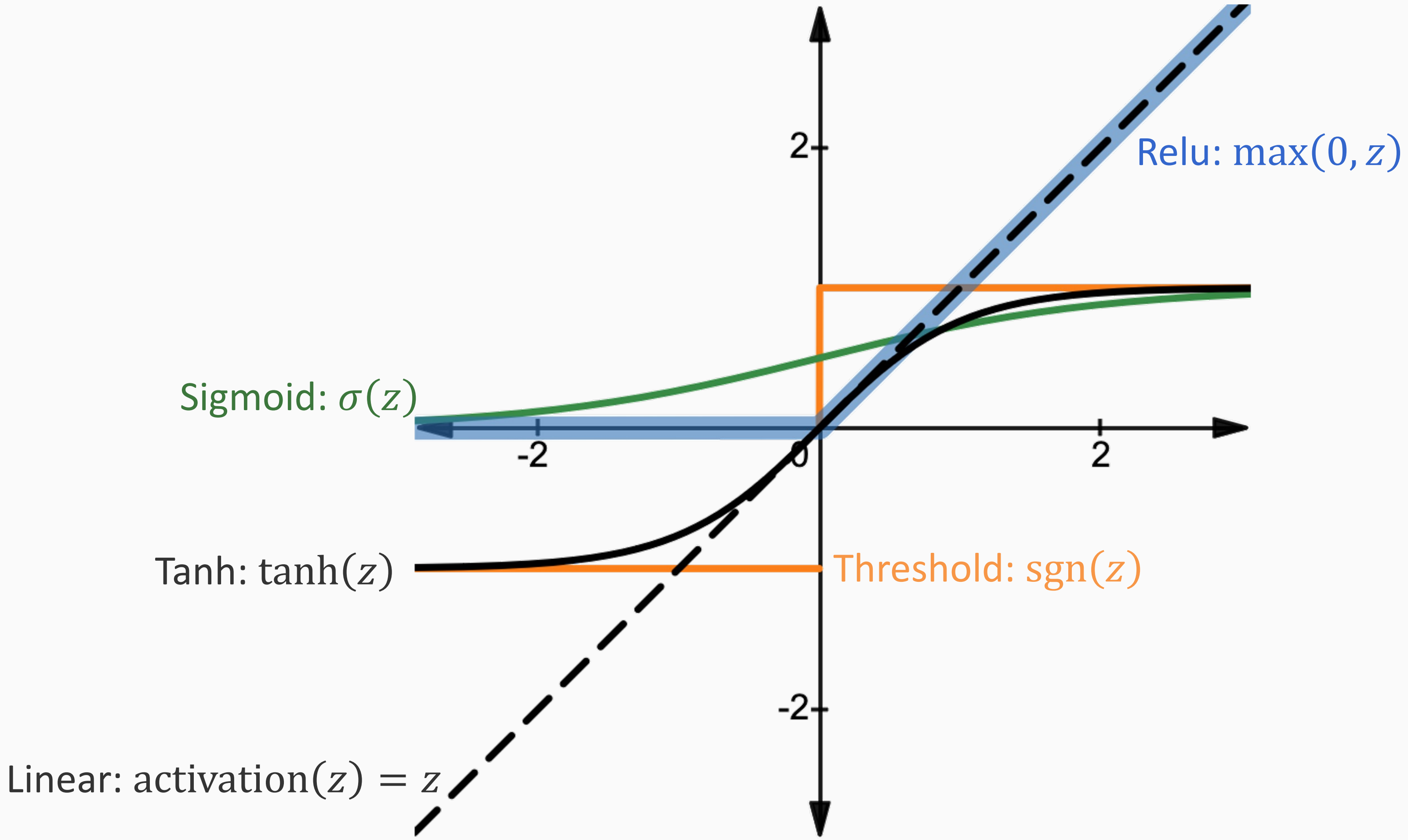
Activation functions illustrated



Activation functions illustrated



Activation functions illustrated



A neural network

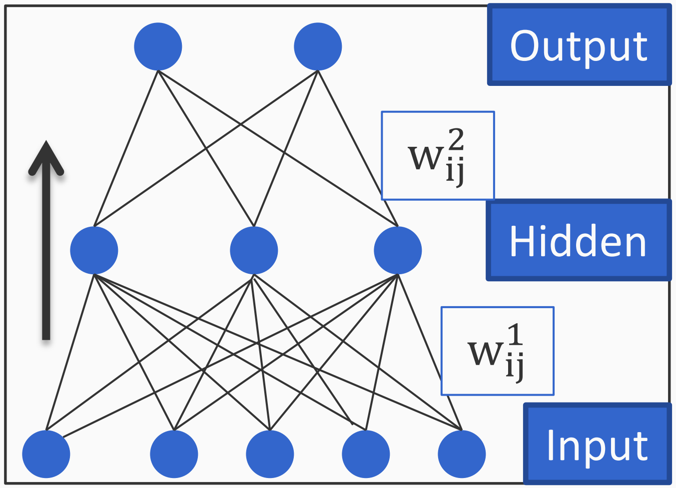
A function that converts inputs to outputs defined by a directed acyclic graph

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

A neural network

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

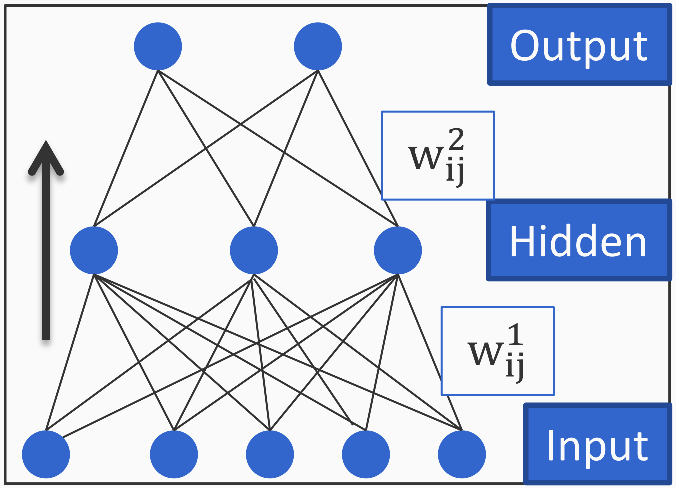
- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights



A 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

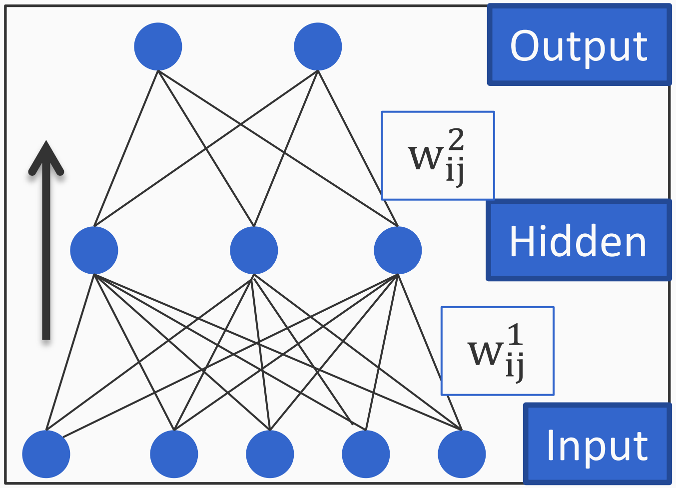


- To define a neural network, we need to specify:
  - The structure of the graph
    - How many nodes, the connectivity
  - The activation function on each node
  - The edge weights

A 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



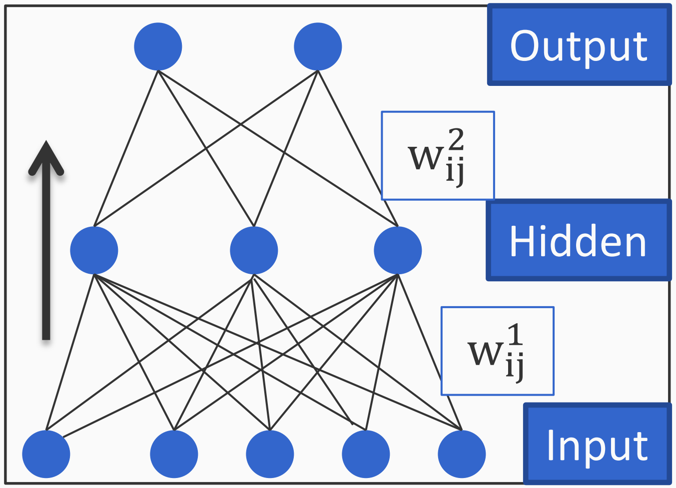
- To define a neural network, we need to specify:
  – The structure of the graph
    - How many nodes, the connectivity
  – The activation function on each node
  – The edge weights

Called the *architecture* of the network

Typically predefined, part of the design of the classifier

A 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

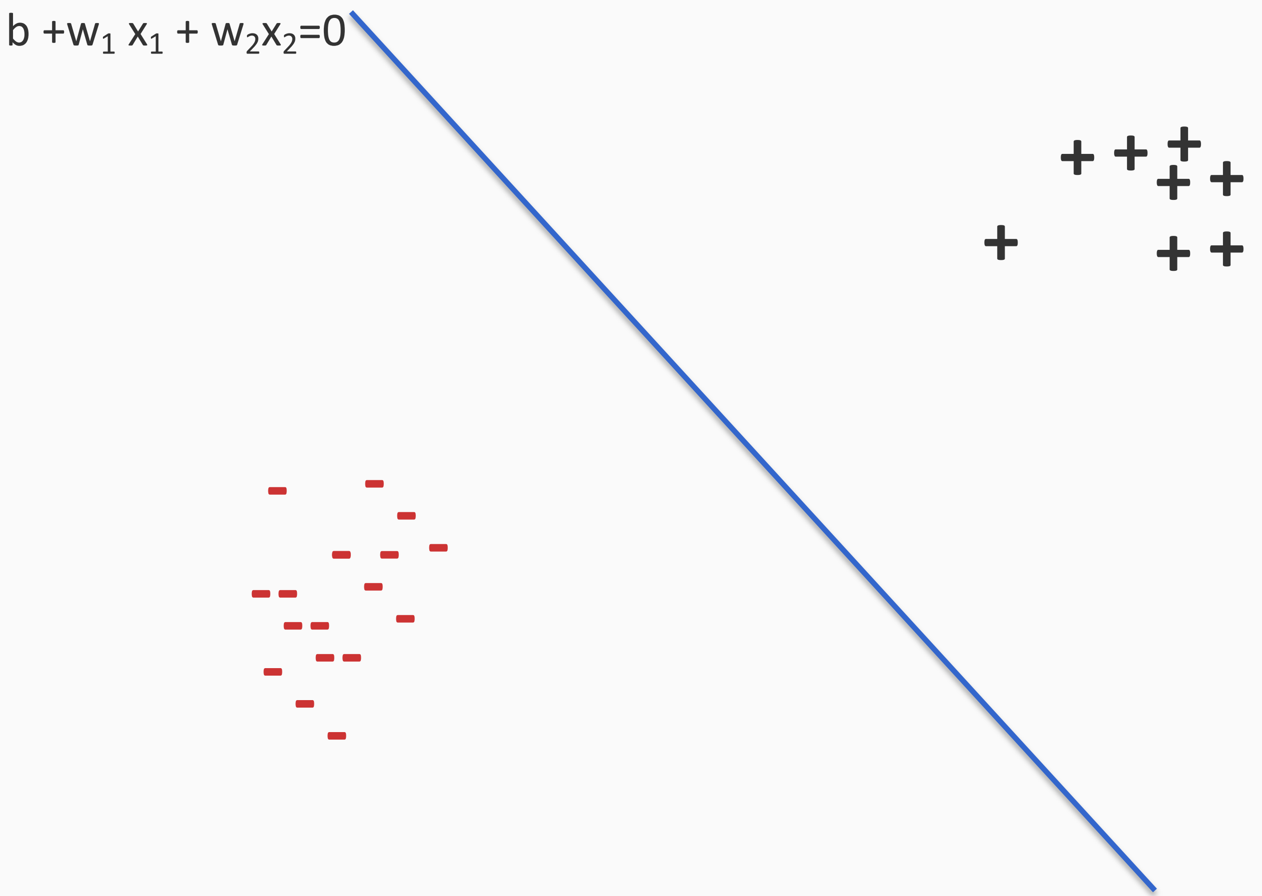
very
A brief history of neural networks

1943	McCullough and Pitts showed how linear threshold units can compute logical functions
1949	Hebb suggested a learning rule that has some physiological plausibility
1950	Rosenblatt, the Peceptron algorithm for a single threshold neuron
1969	Minsky and Papert studied the neuron from a geometrical perspective
1970s-90s	Convolutional neural networks (Fukushima, LeCun), the backpropagation algorithm (various), recurrent neural networks (various)
2000s-now	More compute, more data, more applications, deeper networks, better optimization, transformers, graph neural networks

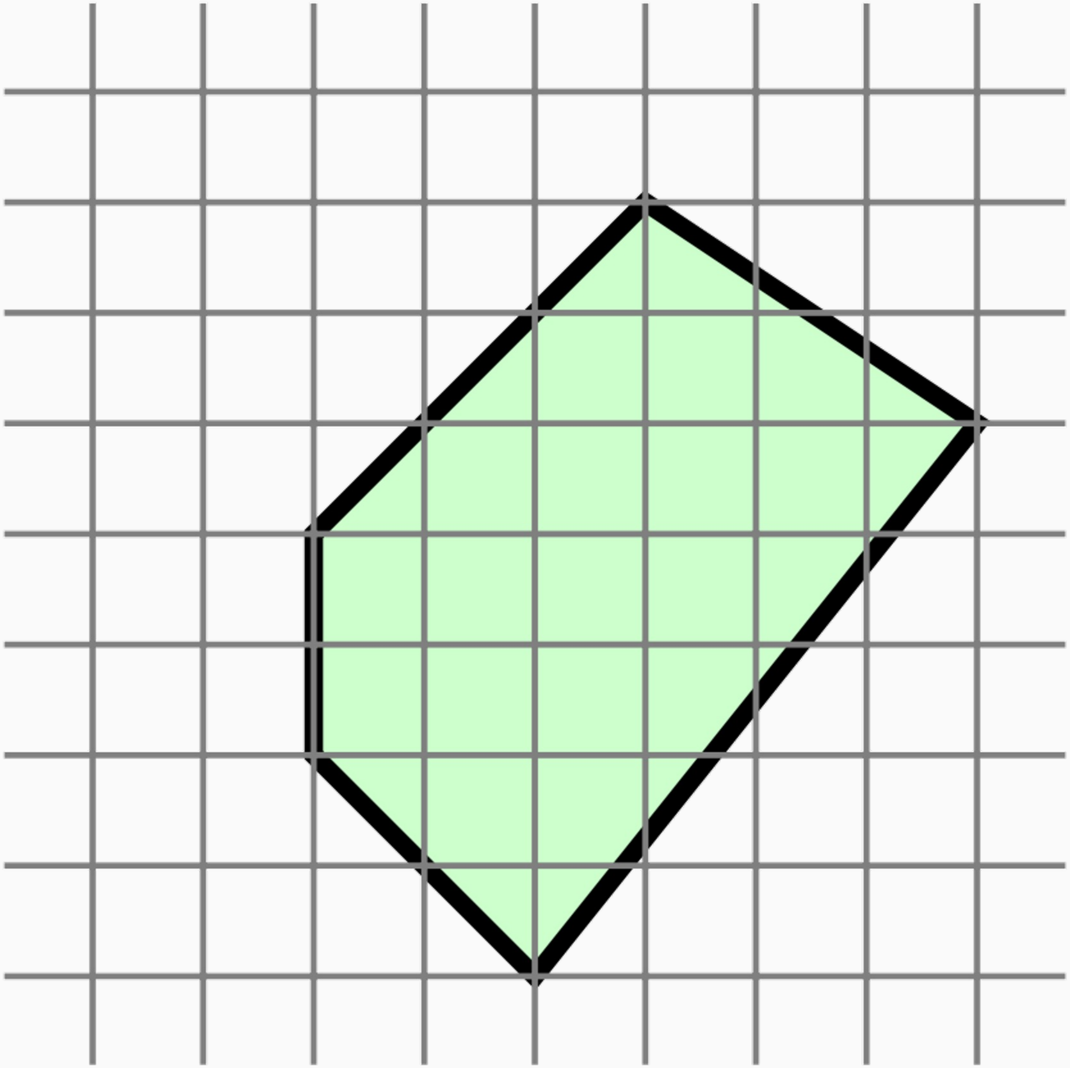
What functions do neural networks express?

A single neuron with threshold activation

Prediction = $\textcolor{red}{\mathit{sgn}}(b + w_1 x_1 + w_2 x_2)$



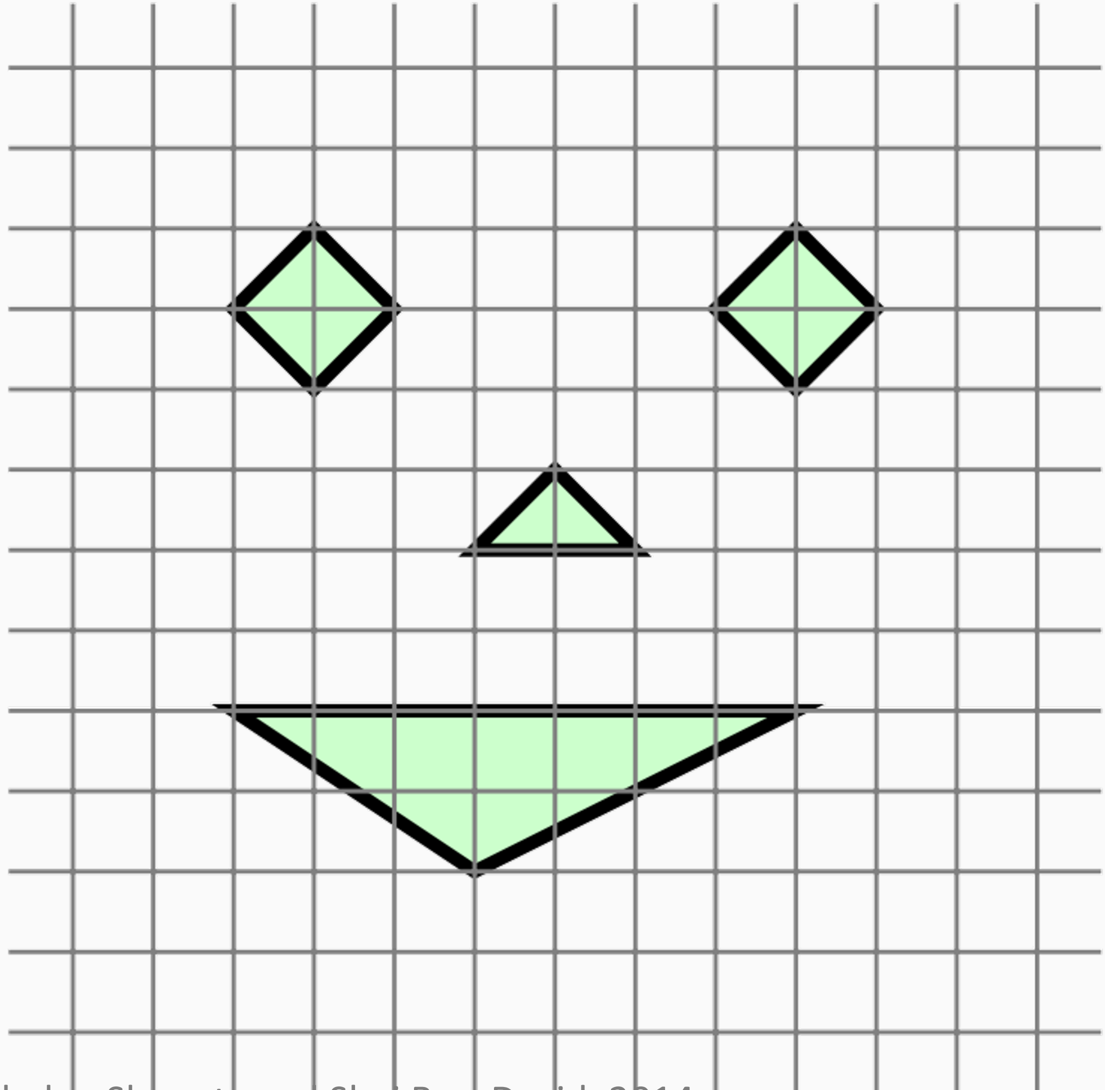
Two layers, with threshold activations



In general, convex polygons

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

Three layers with threshold activations



In general, unions of convex polygons

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

Neural networks 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]

- Two layer threshold networks can express *any* Boolean function
  - **Exercise**: Prove this

Neural networks 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]

- Two layer threshold networks can express *any* Boolean function
  - **Exercise**: Prove this

- VC dimension of threshold network with edges E: $VC = O(|E| \log |E|)$

- VC dimension of sigmoid networks with nodes V and edges E:
  - Upper bound: $O(|V|^2|E|^2)$
  - Lower bound: $\Omega(|E|^2)$

Neural networks 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]

- Two layer threshold networks can express *any* Boolean function
  - **Exercise**: Prove this

- VC dimension of threshold network with edges E: $VC = O(|E| \log |E|)$

- VC dimension of sigmoid networks with nodes V and edges E:
  - Upper bound: $O(|V|^2 |E|^2)$
  - Lower bound: $\Omega(|E|^2)$

Exercise: Show that if we have only linear units, then multiple layers does not change the expressiveness