Lecture 18: Neural Networks Forward Pass

COMP 343, Spring 2022

Victoria Manfredi

WESLEYAN UNIVERSITY

Acknowledgements: These slides are based primarily on content from the book “Machine Learning” by Tom Mitchell and slides created by Vivek Srikumar (Utah) and Dan Roth (Penn) and Dan Klein and Pieter Abbeel (UC Berkeley)

Today's Topics

Homework 6

- due today by 5p

Project proposal

- due April 20 by 5p
- No homework this week, so you have time to think about this carefully!

Recap

Neural networks

- Structure, expressiveness
- Prediction using a neural network
- Training neural networks
- Practical concerns

Miscellaneous

Homework 6

- Possible to see large weights/instability?
  - Why? learning rate is too large. With larger batches want a smaller learning rate.
- Working with long running simulations
  - Set CPU to not go to sleep
  - Run process in background or detach process (screen)

Homework 7/8

- Pushed back one week, would like to go through math more carefully monday before you implement anything

Homework 9

- is no more … will give you a bit more time to focus on project and you will get a bit of tensorflow in homework 8

Tensorflow

- My plan for lectures 22/23: a bit of me lecturing and a lot of you working through tensor flow setup and tutorial (so bring your laptops)

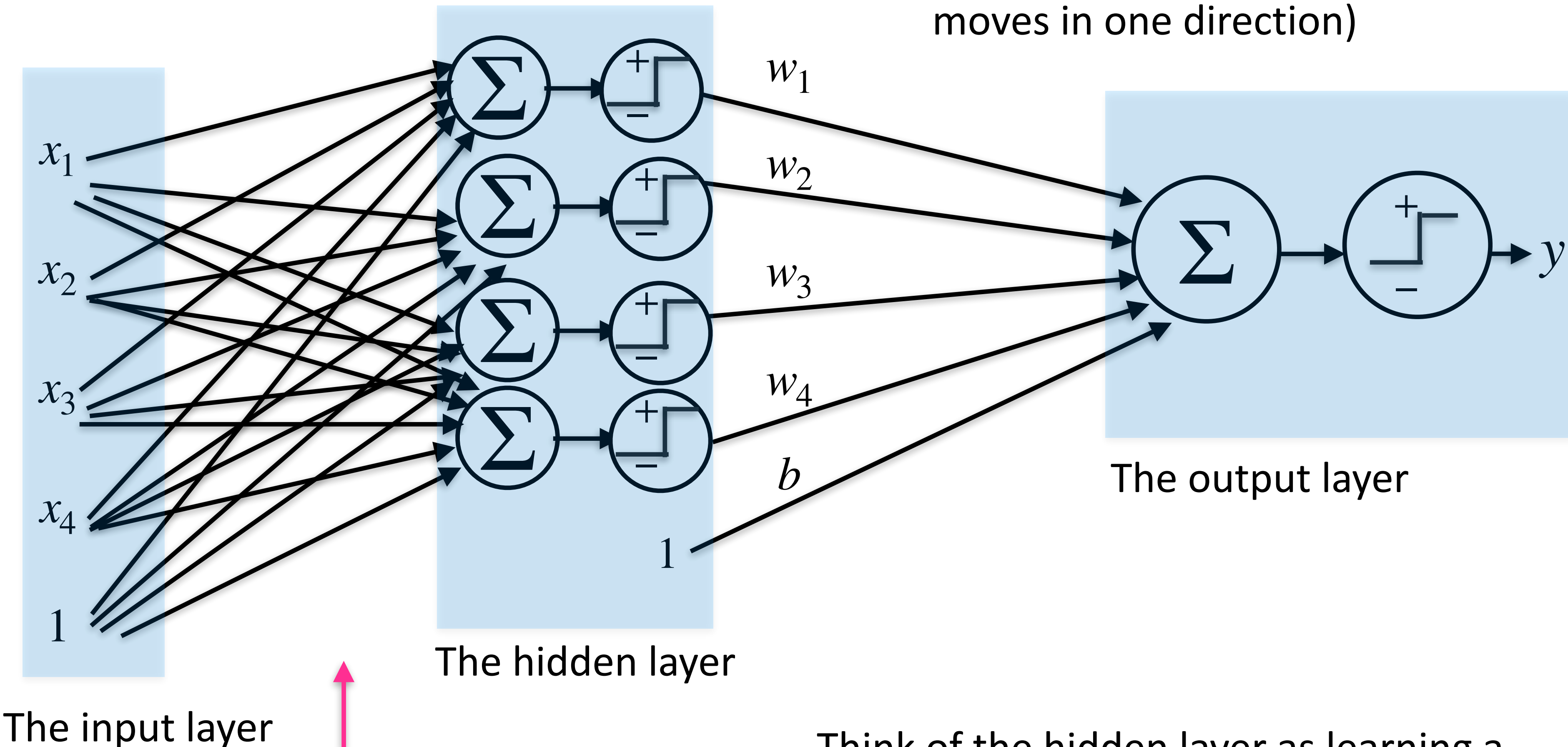
Project

- Webpage with handout is posted:
  - http://vumanfredi.wescreates.wesleyan.edu/teaching/comp343-s22/homework/project/
- I will update handout further in next week or so

Recap

Features from classifiers

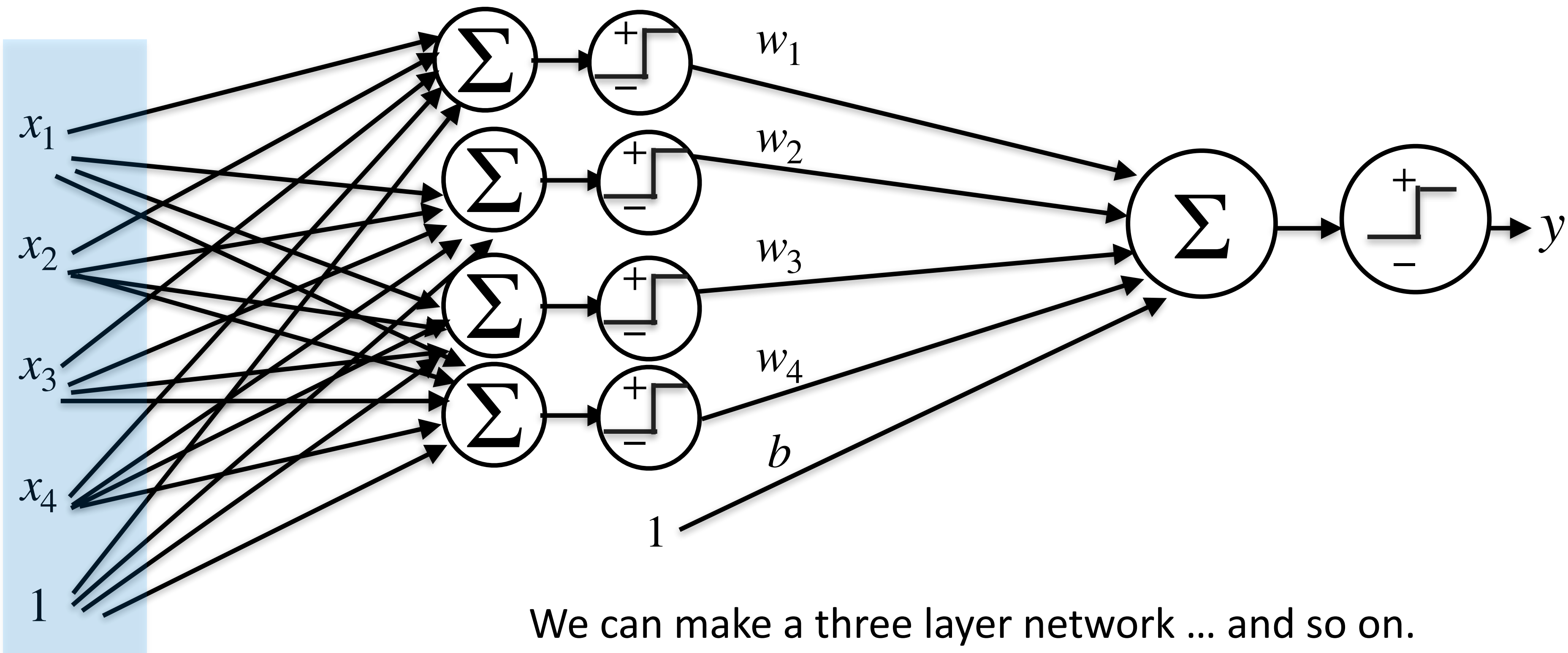
This is a two-layer feed forward neural network (i.e., information moves in one direction)



These connections are computing feature transformations

Think of the hidden layer as learning a good representation of the inputs for this particular classification or regression task. For different tasks, hidden layer might be different

But where do the inputs come from?



The input layer

Question: how do we know these input features are the right ones? Maybe we need another layer

We can make a three layer network … and so on.
Idea: let inputs be as close as possible to raw data and let neural network learn representation of features

OpenAI GPT-3: Neural network with 175 billion parameters

https://arxiv.org/pdf/2005.14165.pdf

Two takeaways about neural networks

- Neural networks can learn feature representations

- Learned feature representations outperform hand-coded representations

… It’s all about the features!

Neural Networks

A BIT MORE FORMALLY

Neural networks

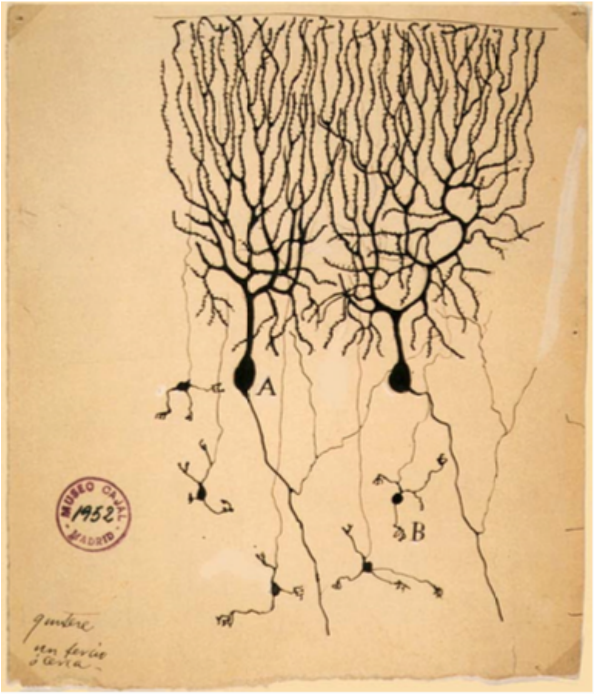
- Rich history, starting in early forties (McCulloch/Pitts 1943)

- A robust approach for approximating real-valued, discrete-valued, or vector valued functions. Neural networks are a family of functions, not just one.

- 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 like speech and pixels

- The Backpropagation algorithm for neural networks has been shown to be 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 in electrical signal

Modern artificial neurons are “inspired” by biological neurons, but there are many, many fundamental differences. Don’t take the similarity too seriously

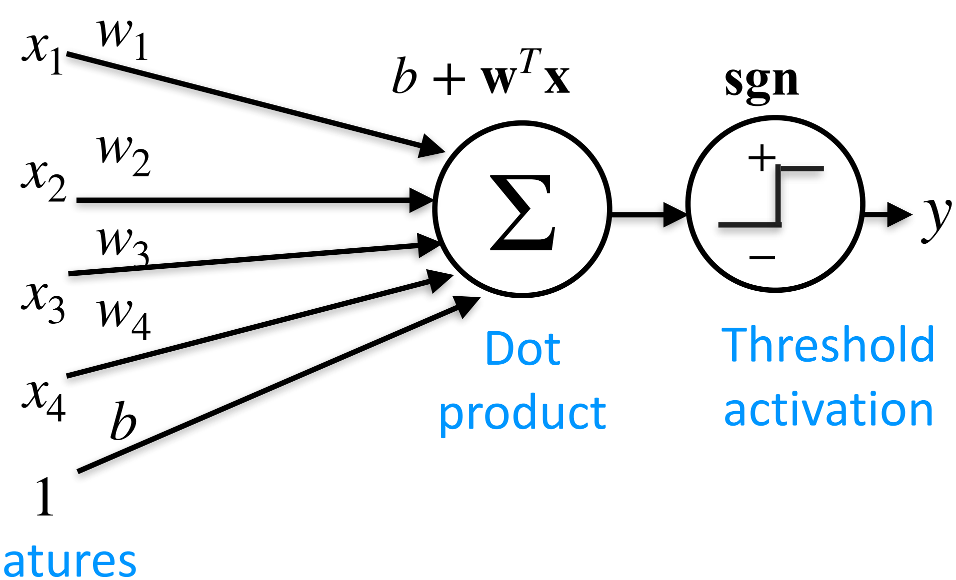
Artificial neurons

Artificial neurons are functions that very loosely mimic a biological neuron

A neuron/function accepts collection of inputs (typically a vector $\mathbf{x}$ but could be a matrix or something more complicated) and produces output by:

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

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



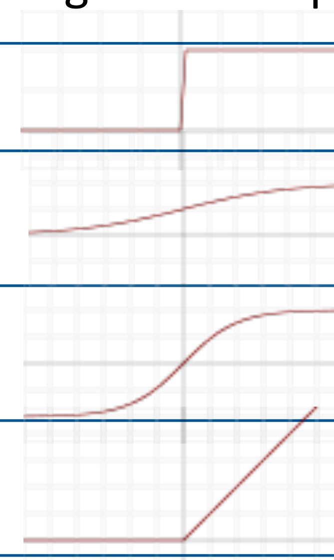
Other activation functions are possible

Common activation functions

Also called transfer functions

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

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



https://en.wikipedia.org/wiki/Activation_function

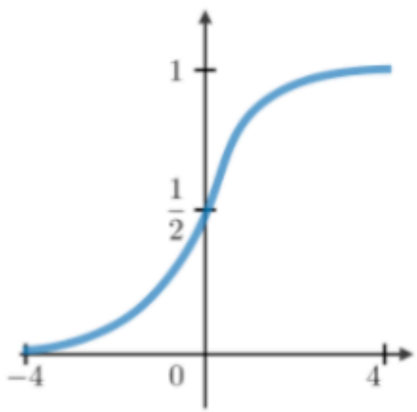
Notice that many of these functions are nonlinear! Allows neural network to represent nonlinear functions

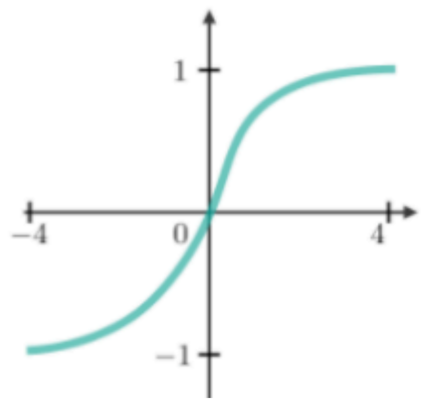
Common activation functions

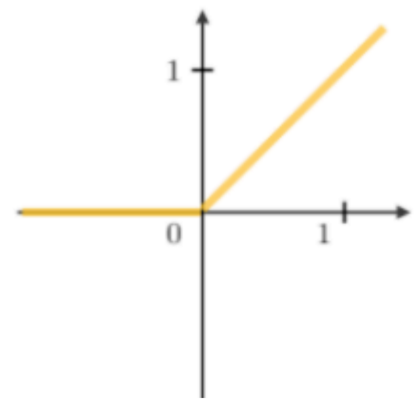
Different activation functions give different outputs

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

Sigmoid	Tanh	ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$







From slides by Nandita Bhaskhar and others at Stanford

3min:

For a classification task: which activation functions would you use at output layer?

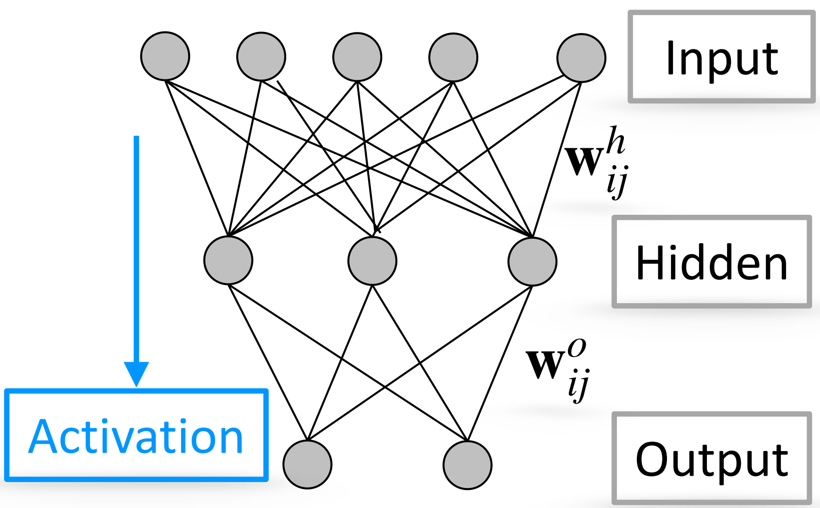
For a regression task: which activation functions would you use at output layer?

Would normalizing your y values (or output values) be important? Why?

What is a neural network?

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

- Nodes are organized in layers, correspond to neurons
- Edges carry output of one neuron to another neuron (i.e., provide input features)
- Edges are associated with weights



How do we define a neural network?

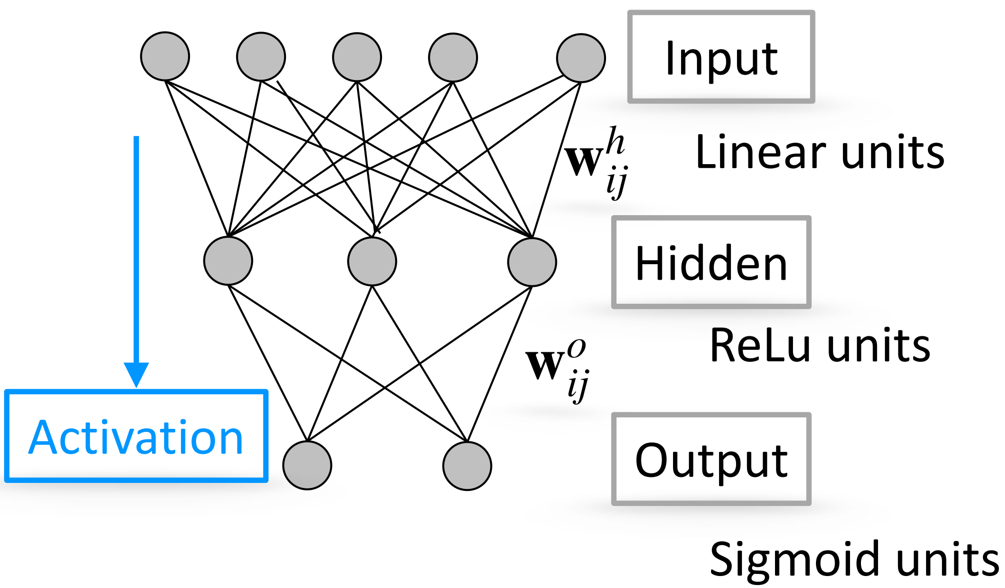
To define a neural network, we need to specify

- The structure of the graph: how many nodes (what is the type of each node) and how are they organized (what is the connectivity among nodes)
- The activation function on each node
- The edge weights

Learned from data, assuming structure of graph is fixed

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

Specialized architectures for different problems like vision or text



A very brief history of neural nets

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 Perceptron algorithm for a single threshold neuron

1969: Minsky and Papert studied the neuron from a geometrical perspective

1970s, 80s: Convolutional neural networks (Fukushima, LeCun), the back propagation algorithm (various)

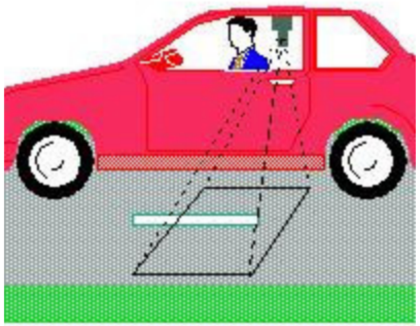
Early 2000s-today: More compute, more data, deeper networks

See also https://people.idsia.ch//~juergen/deep-learning-overview.html

Applications of neural networks

Trained to drive

- No-hands across America (Pomerleau)
- ARPA Challenge (Thrun)



Trained to recognize handwritten digits

- more than 99% accuracy

Trained to play games: Google DeepMind

- AlphaZero chess engine, AlphaGo for Go, …

And many more …

Neural Networks

WHAT FUNCTIONS DO NEURAL NETWORKS EXPRESS?

Neural networks are just functions

Can view neural network as systematic way of writing down a function

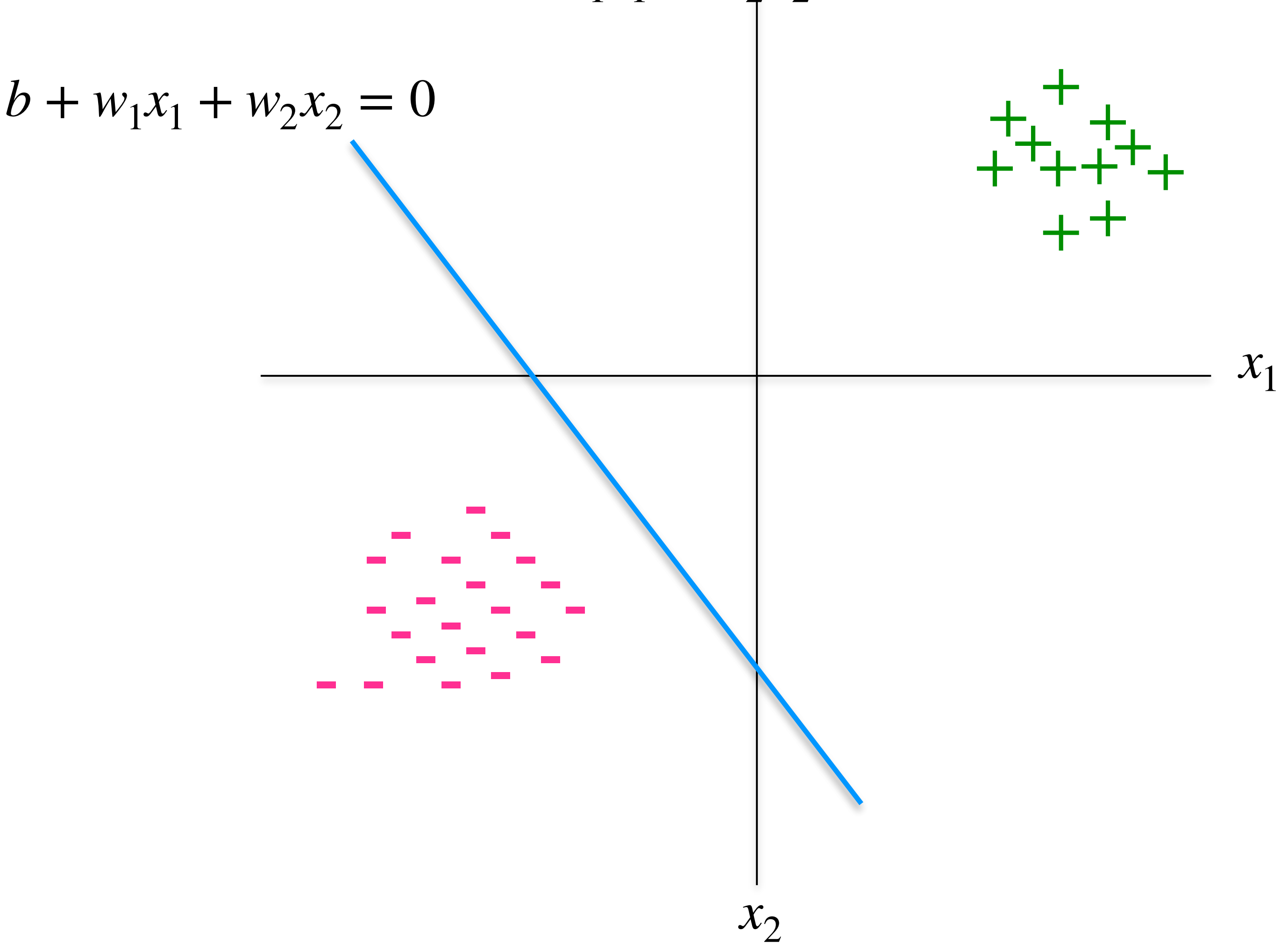
Any neural network architecture you have (graph of nodes, activation functions, connectivity) gives you a hypothesis class!

What is expressivity of this hypothesis class?

A single neuron with threshold activation

Prediction = sgn$(b + w_1x_1 + w_2x_2)$

aka one-layer neural network



Single neuron with threshold activation: represents line that separates positives from negatives

Two layers, with threshold activations

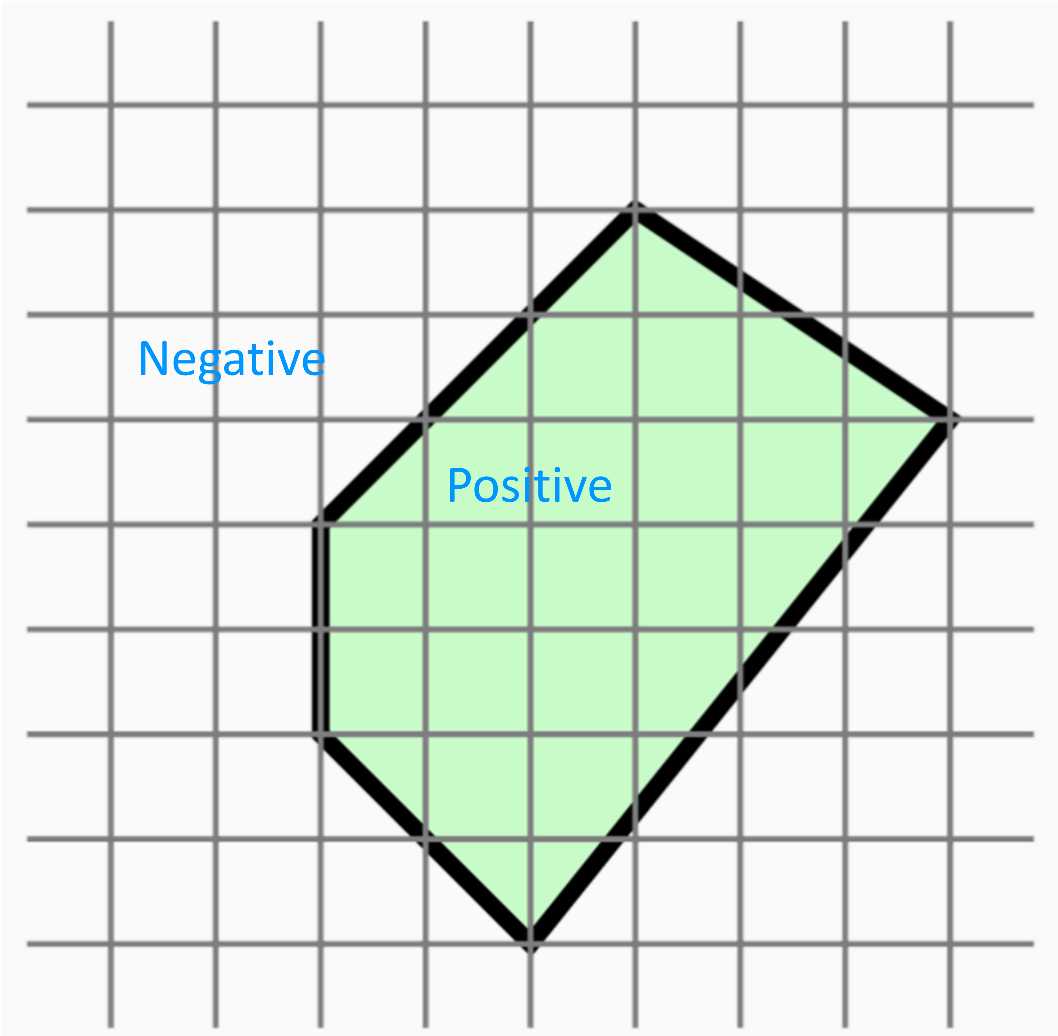


Figure from Understanding Machine Learning and Algorithms
by Shai Shalev-Shwartz and Shai Ben-David, 2014

With two layers, get a convex polygon

Three layers, with threshold activations

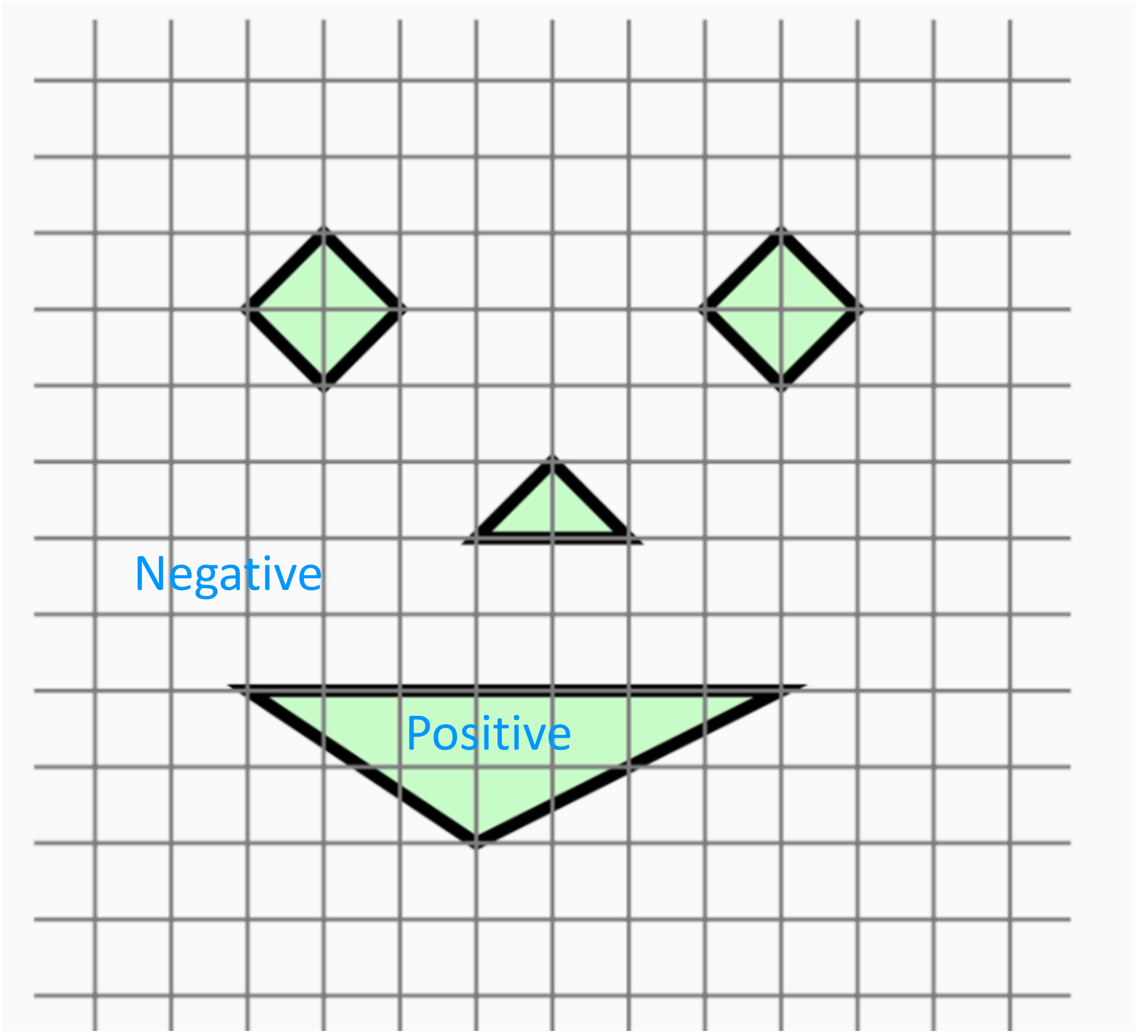


Figure from Understanding Machine Learning and Algorithms
by Shai Shalev-Shwartz and Shai Ben-David, 2014

In general, get union of convex polygons

Neural networks are universal function approximations

- Any continuous function can be approximated to arbitrary accuracy using one layer of sigmoid units [Cybenko 1989]
- At least theoretically, approximation error is insensitive to the choice of activation functions [DasGupta et al 1993], but empirically which activation function used has an impact
- Two layer threshold networks can express any Boolean function
- 3 min: if only linear activation functions, does adding multiple layers change expressiveness? No

Universal function approximation theorem

Hornik theorem 1: Whenever the activation function is *bounded and nonconstant*, then, for any finite measure μ, standard multilayer feedforward networks can approximate any function in $L^p(\mu)$ (the space of all functions on R^k such that $\int_{R^k} |f(x)|^p d\mu(x) < \infty$) arbitrarily well, provided that sufficiently many hidden units are available.

Hornik theorem 2: Whenever the activation function is *continuous, bounded and nonconstant*, then, for arbitrary compact subsets $X \subseteq R^k$, standard multilayer feedforward networks can approximate any continuous function on X arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are available.

In words: Given any continuous function $f(x)$, if a 2-layer neural network has enough hidden units, then there is a choice of weights that allow it to closely approximate $f(x)$

Practical considerations

- Can be seen as learning the features
- Large number of neurons (danger for overfitting, use early stopping)

Further reading

Cybenko (1989) “Approximations by superpositions of sigmoidal functions”

Hornik (1991) “Approximation Capabilities of Multilayer Feedforward Networks”

Leshno and Schocken (1991) ”Multilayer Feedforward Networks with Non-Polynomial Activation Functions Can Approximate Any Function”

https://mcneela.github.io/machine_learning/2017/03/21/Universal-Approximation-Theorem.html

http://neuralnetworksanddeeplearning.com/chap4.html

Neural Networks

HOW DO WE COMPUTE OUTPUT OF NEURAL NETWORK?

Questions to ask about any ML algorithm …

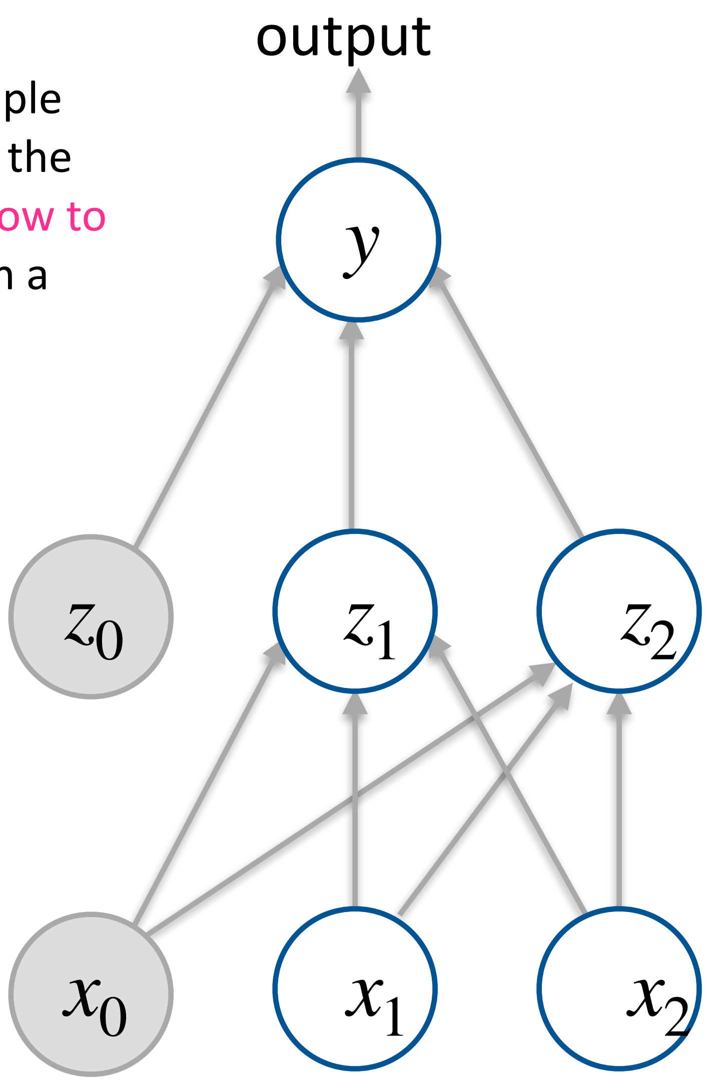
How do we make predictions (compute output)?

How do we learn or train?

First, let's look at how we make predictions with neural network …

Consider an example network

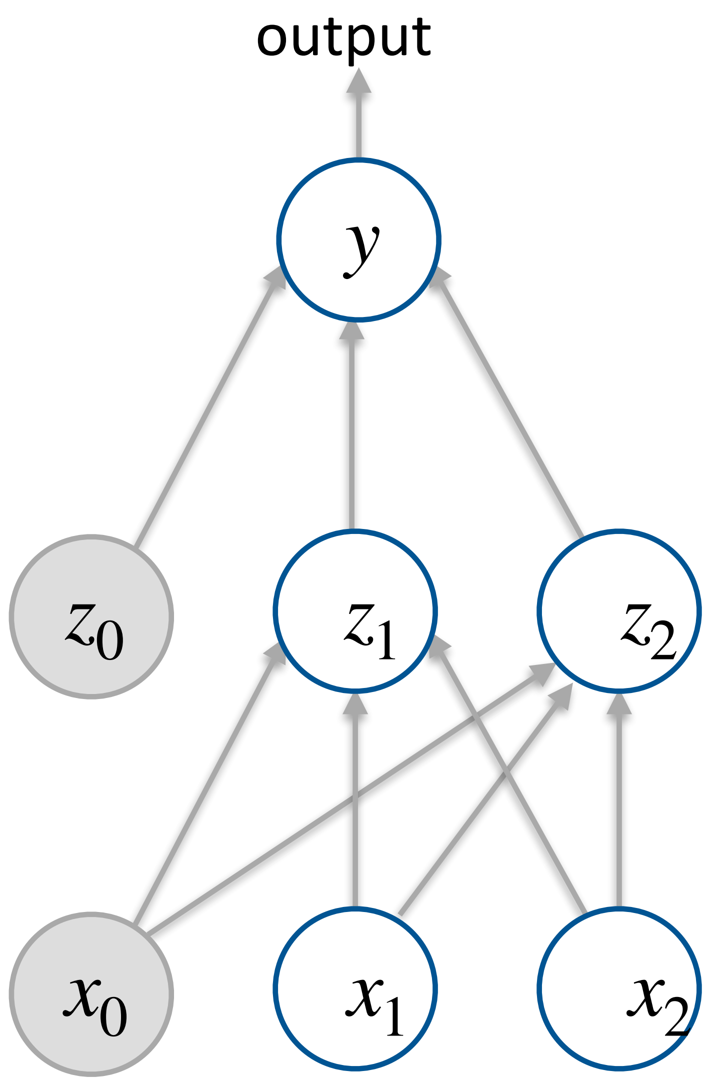
We will use this example network to introduce the general principle of how to make predictions with a neural network



How many layers does this neural network have?

2 layers

What is all this notation?



Naming conventions for this example

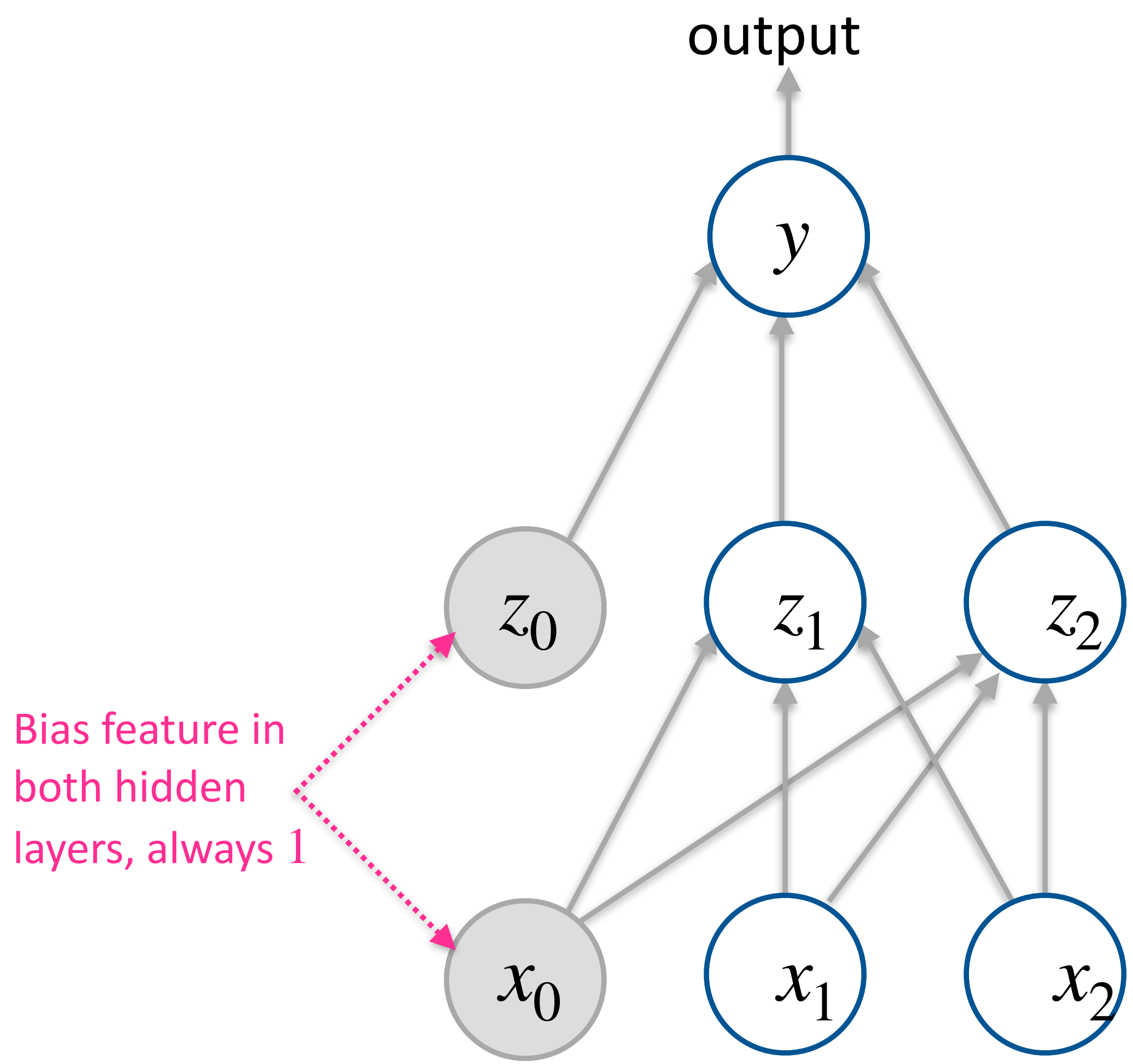
- inputs: x
- hidden: z
- output: y

What are some examples of x and y?

xs are features, could be any features we've used on homework

ys are what we are trying to predict, e.g., house price, spam or not spam, etc.

Bias features are part of network



Naming conventions for this example

- inputs: x
- hidden: z
- output: y

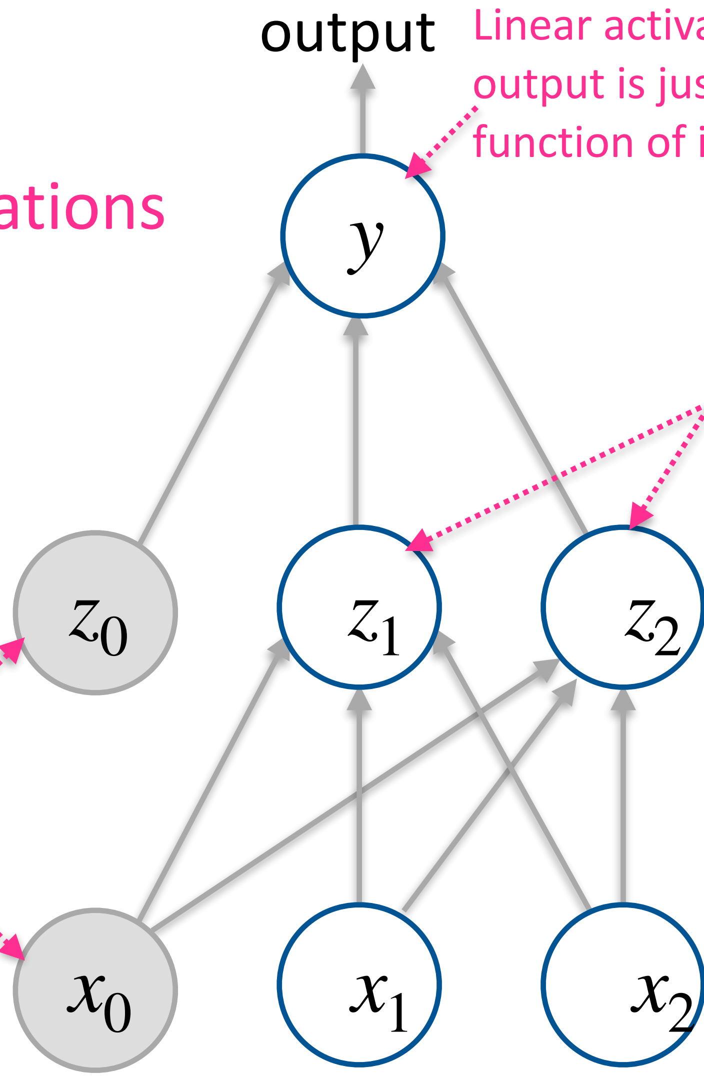
What activation functions to use?

Naming conventions for this example

- inputs: x
- hidden: z
- output: y

Need to also specify activations

Linear activation, so output is just linear function of inputs



Let's assume sigmoid activations for these nodes, $\frac{1}{1+e^{-z}}$

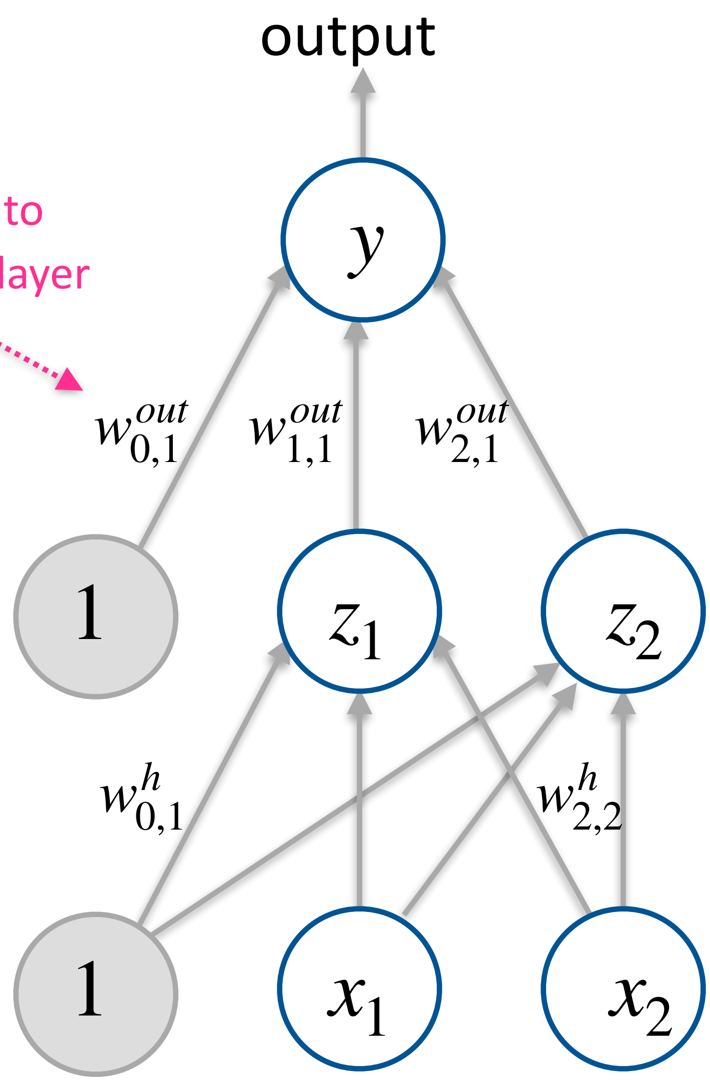
Bias feature, always 1

Notice no activation function on input or bias

Every edge has a weight

Naming conventions for weights: $w^{target-layer}_{from,to}$

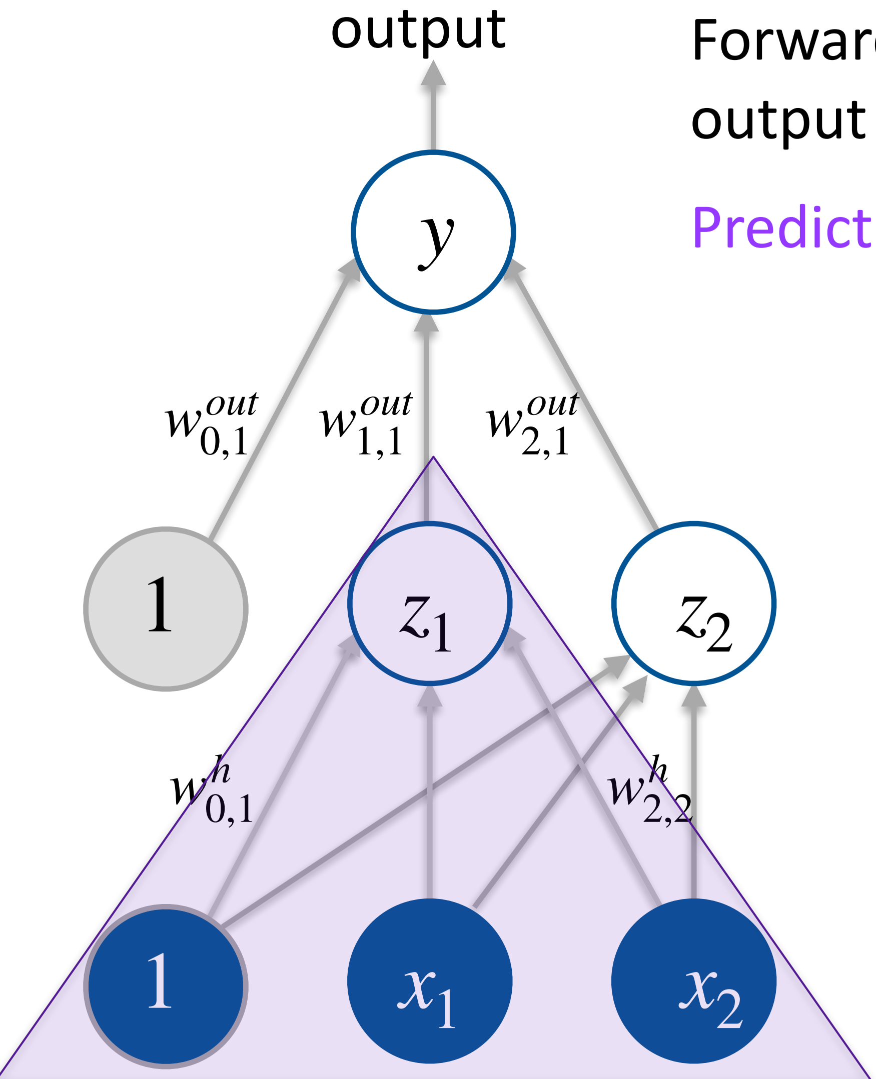
E.g., from neuron 0 to neuron 1 in output layer



Need to specify weights in order to make predictions

One weight for every edge in graph

How to do prediction?



Forward pass: given an input $\mathbf{x}$, how is the output predicted using a neural network?

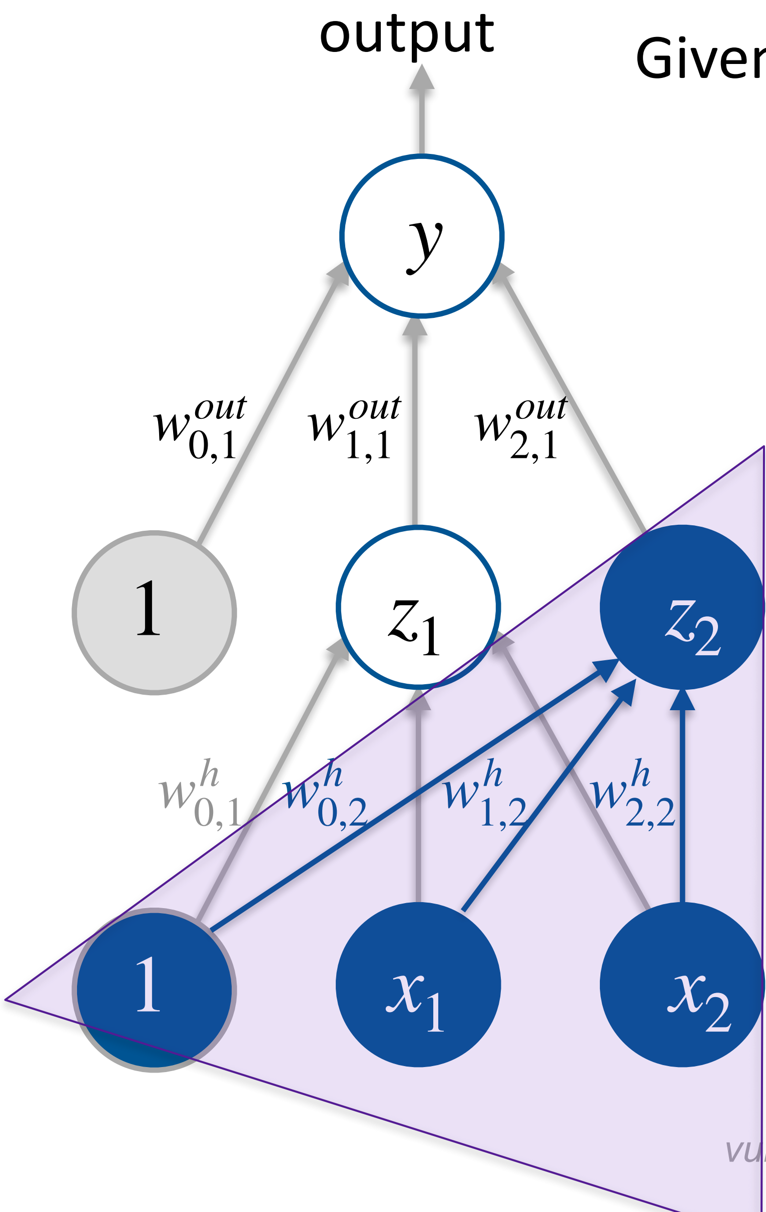
Predict from the bottom up …

$$z_1 = \sigma(w_{0,1}^h + w_{1,1}^h x_1 + w_{2,1}^h x_2)$$

To compute output need to know each input and associated weight

Question: what is activation at hidden nodes, z? Sigmoid

The forward pass for prediction

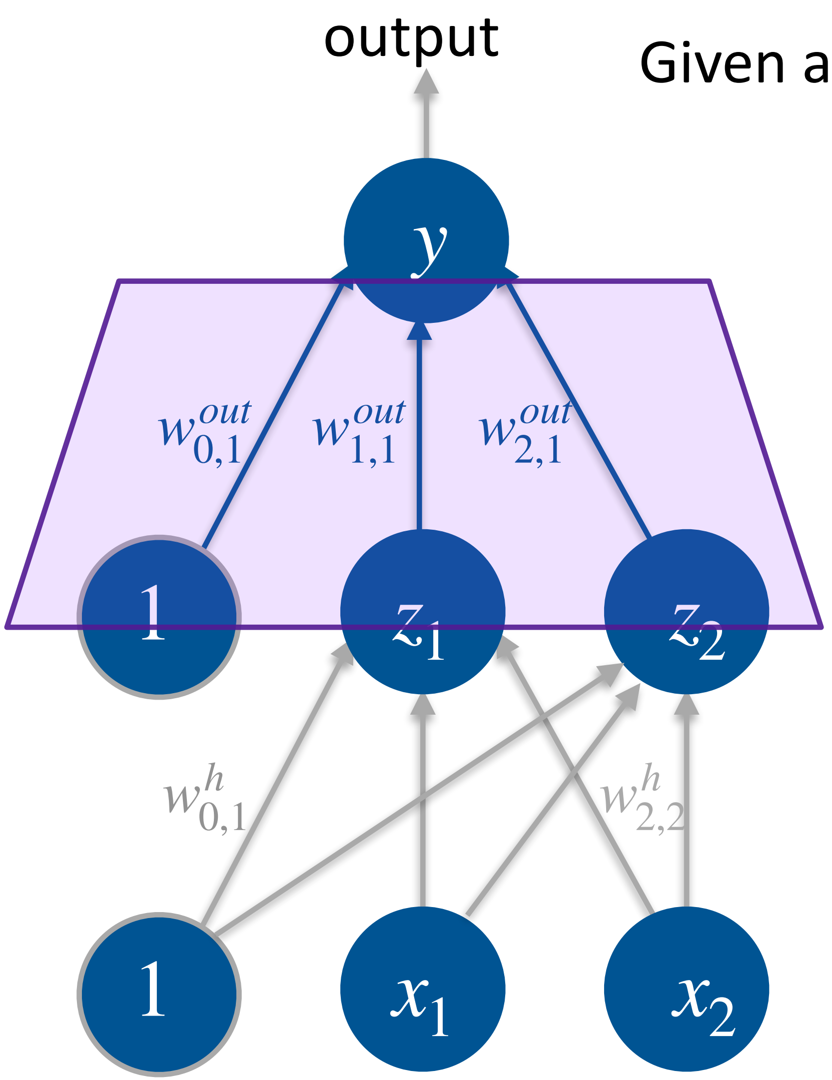


Given an input $\mathbf{x}$, how is the output predicted?

$$z_2 = \sigma(w^h_{0,2} + w^h_{1,2}x_1 + w^h_{2,2}x_2)$$

$$z_1 = \sigma(w^h_{0,1} + w^h_{1,1}x_1 + w^h_{2,1}x_2)$$

The forward pass for prediction



Given an input $\mathbf{x}$, how is the output predicted?

$$y = \sigma(w^o_{0,1} + w^o_{1,1} z_1 + w^o_{2,1} z_2)$$

$$z_2 = \sigma(w^h_{0,2} + w^h_{1,2} x_1 + w^h_{2,2} x_2)$$

$$z_1 = \sigma(w^h_{0,1} + w^h_{1,1} x_1 + w^h_{2,1} x_2)$$

In general, to predict output, just compute value. But before visiting (i.e., computing) the value of a node, need to visit all nodes that serve as inputs to it