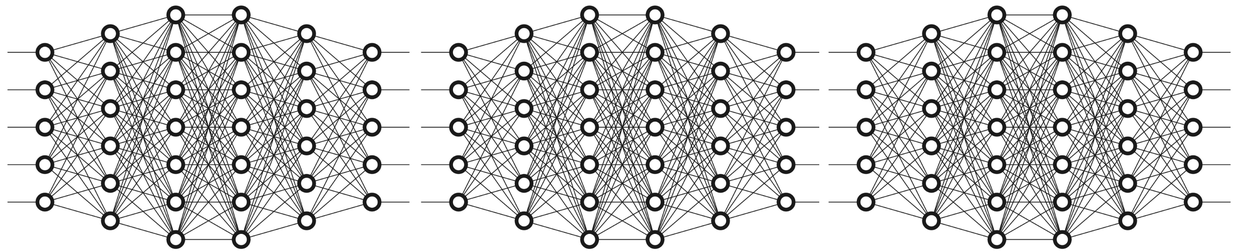
THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision



Lecture 19: Neural Networks 1

Menu for Today

Topics:

- **Neural Networks** introduction
- **Activation functions** softmax, relu
- **2-layer** fully connected net
- **Backprop** intro

Readings:

- **Today's** Lecture: Szeliski 5.1.3, 5.3-5.4, Justin Johnson Michigan EECS 498/598

Reminders:

- **Assignment 5**: due today
- **Assignment 6**: Deep Learning is now available

Recall: Linear Classifier

Defines a score function:

$$f(\mathbf{x}_i, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x}_i + \mathbf{b}$$

image features

weights (parameters)

bias vector

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Recall: Linear Classifier

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

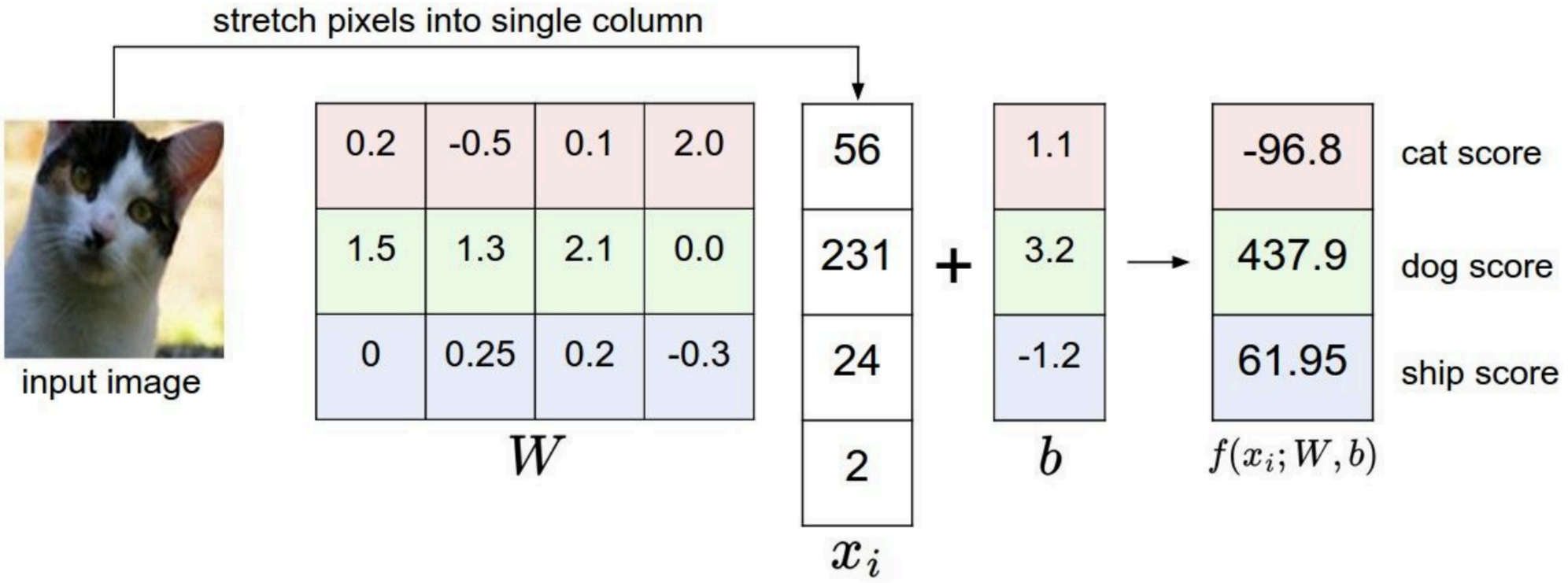


Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

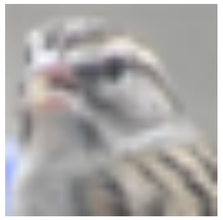
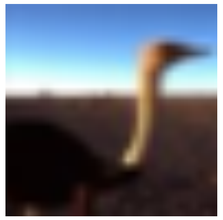
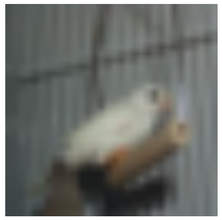
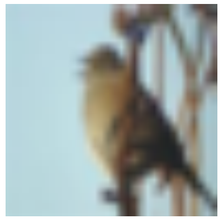
Linear Classification

- Let's start by using 2 classes, e.g., bird and plane
- Apply labels (y) to training set:

y = +1




y = -1

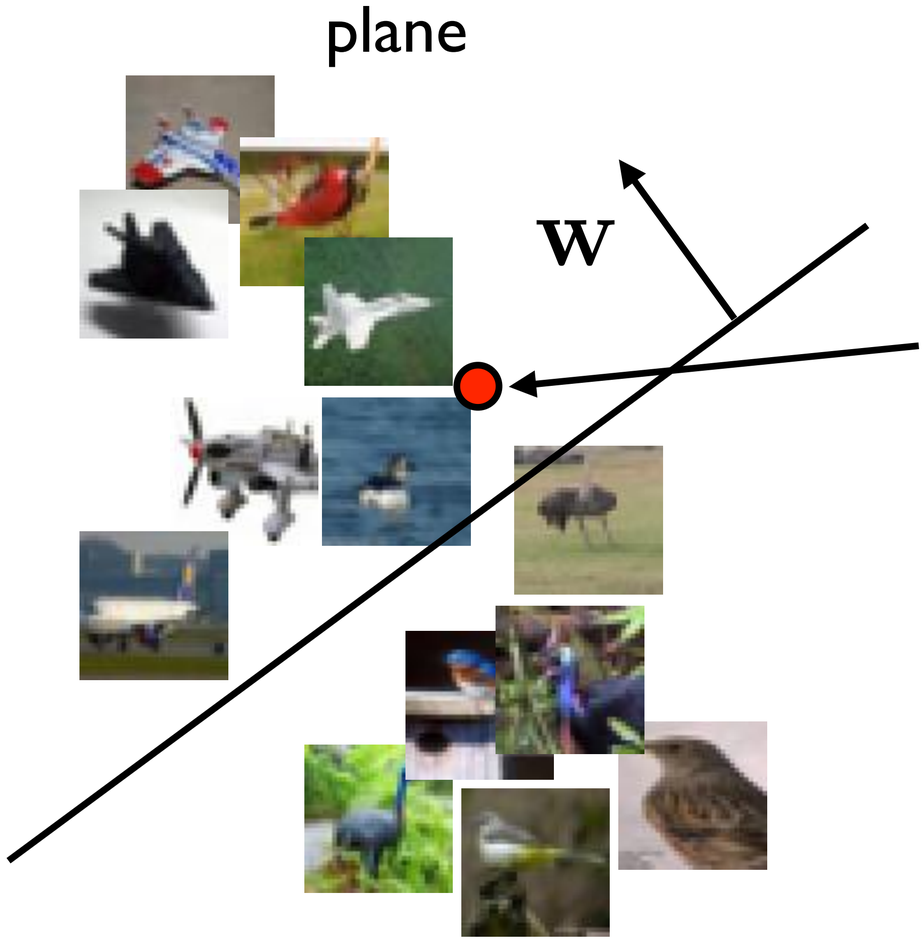





- Use a linear model to regress y from x

$$\hat{y} = \text{sign } h = \text{sign } \mathbf{w}^T \mathbf{x}_q$$

2-class Linear Classification

- Separating hyperplane, projection to a line defined by **w**



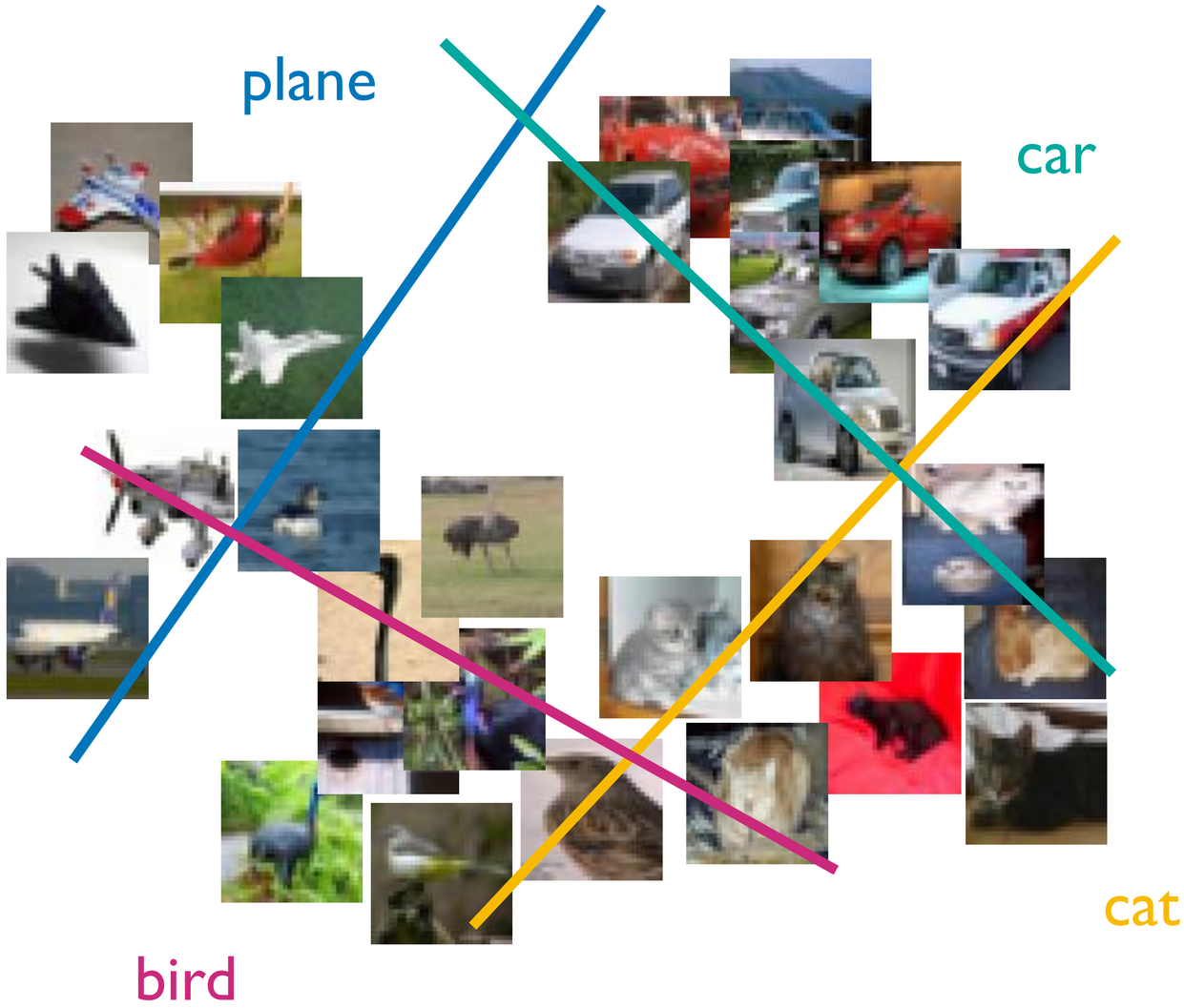
Query:

$$\mathbf{x}_q =$$ 

$$\hat{y} = \text{sign } h = \text{sign } \mathbf{w}^T \mathbf{x}_q$$

N-class Linear Classification

- One hot regression = 1 vs all classifiers



One-Hot Regression

- A better solution is to regress to one-hot targets = 1 vs all classifiers

$$\begin{bmatrix} \\ \mathbf{W}^T \\ \\ \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \dots \end{bmatrix}$$

class 2 = 'automobile'

$$\begin{bmatrix} \\ \mathbf{W}^T \\ \\ \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ \dots \end{bmatrix}$$

class 4 = 'cat'

One-Hot Regression

- Transpose (to match Project 3 notebook)


$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots \\ x_{21} & x_{22} & x_{23} & \cdots \\ x_{31} & x_{32} & x_{33} & \cdots \\ & \cdots & & \end{bmatrix} \begin{bmatrix} \\ \mathbf{W} \\ \\ \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 1 & \cdots \\ & .. & .. & & \end{bmatrix}$$

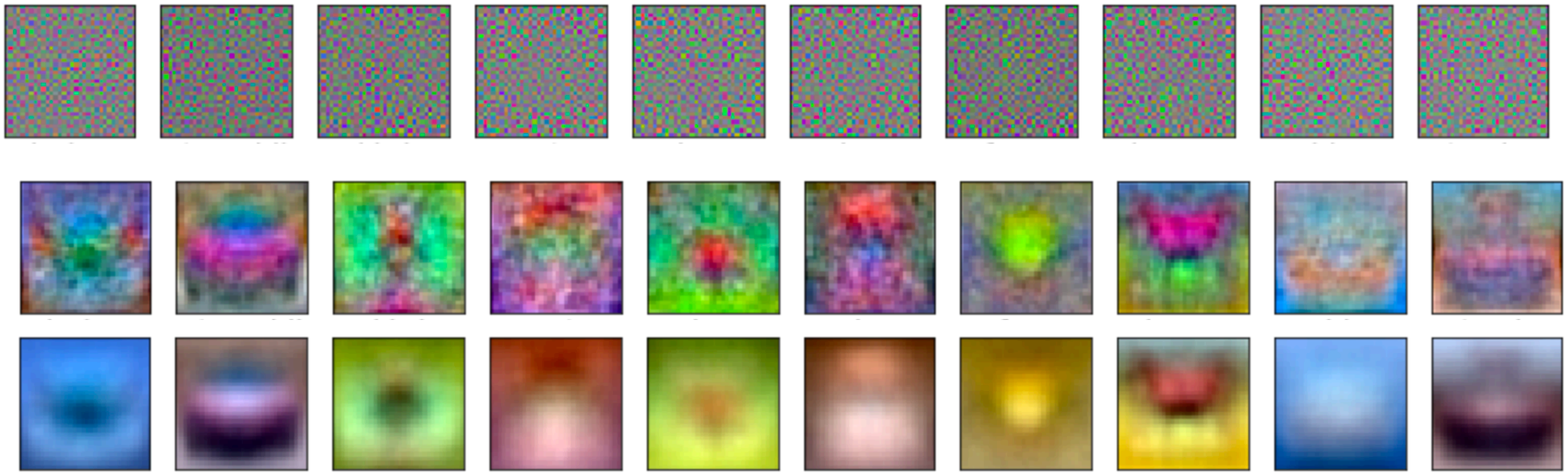
auto
cat

$$\mathbf{XW} = \mathbf{T}$$

- Solve regression problem by Least Squares

Regularized Classification

- Add regularization to CIFAR10 linear classifier



- Row 1 = overfitting, Row 3 = oversmoothing?

$$e = |\mathbf{XW} - \mathbf{T}|^2 + \lambda|\mathbf{W}|^2$$

Softmax + Logistic Outputs

- Linear regression to one-hot targets is a bit strange..
- Output could be very large, and scores >>1 are penalised even for the correct class, ditto scores << 1 for incorrect
- How about restricting output scores to 0-1?

Softmax + Cross Entropy

- What is the gradient of the softmax linear classifier?
- We could use L2 loss, but we'll use cross entropy instead
- This has a sound motivation — it is a measure of the difference between probability distributions
- It also leads to a simple update rule

Linear + Softmax Regression

- We found the following gradient descent update rule

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \alpha(\mathbf{h} - \mathbf{t})\mathbf{x}^T$$

prediction targets data

- This applies to:

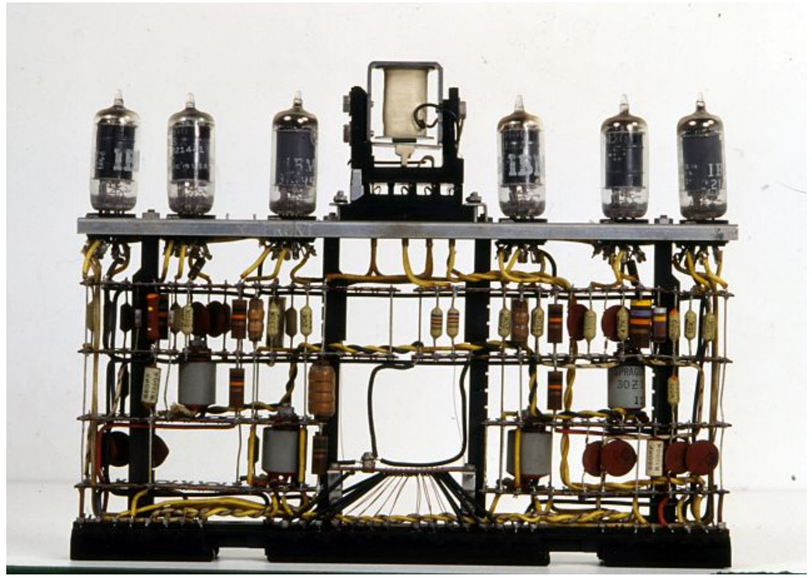
Linear regression	$\mathbf{h} = \mathbf{W}^T\mathbf{x}$	L2 loss
Softmax regression	$\mathbf{h} = \sigma(\mathbf{W}^T\mathbf{x})$	cross-entropy loss

- The same update rule with a binary prediction function

$$\mathbf{h} = \mathbb{1}_{\max}(\mathbf{W}^T\mathbf{x})$$

implements the multiclass Perceptron learning rule

History of the Perceptron



[I.B.M. Italia]

- This machine (IBM 704) was used by Frank Rosenblatt to implement the perceptron in 1958
- Based on his statements, the New York Times reported it as: "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

2-class Perceptron Classifier

- Classification function is

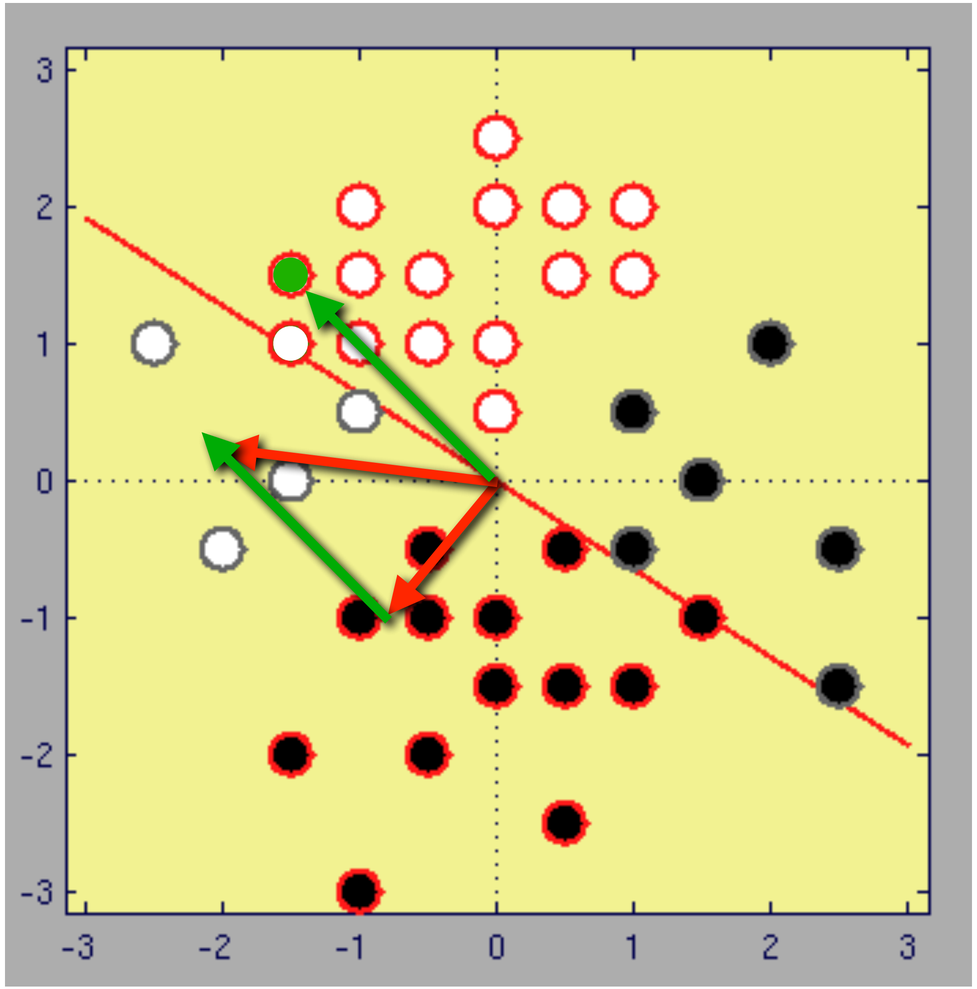
$$\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x})$$

- Linear function of the data (x) followed by 0/1 activation

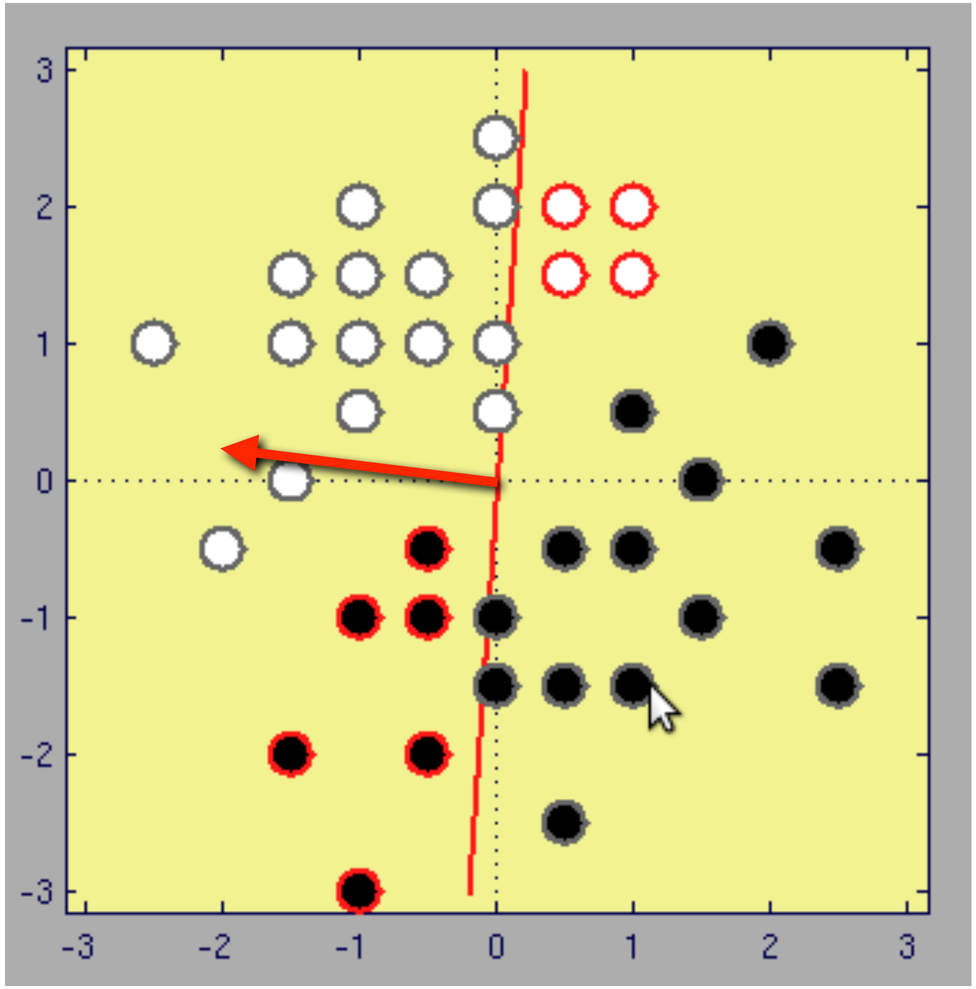
- Update rule: present data x
  - if correctly classified, do nothing
  - if incorrectly classified, update the weight vector

$$\mathbf{w}_{n+1} = \mathbf{w}_n + y_i \mathbf{x}_i$$

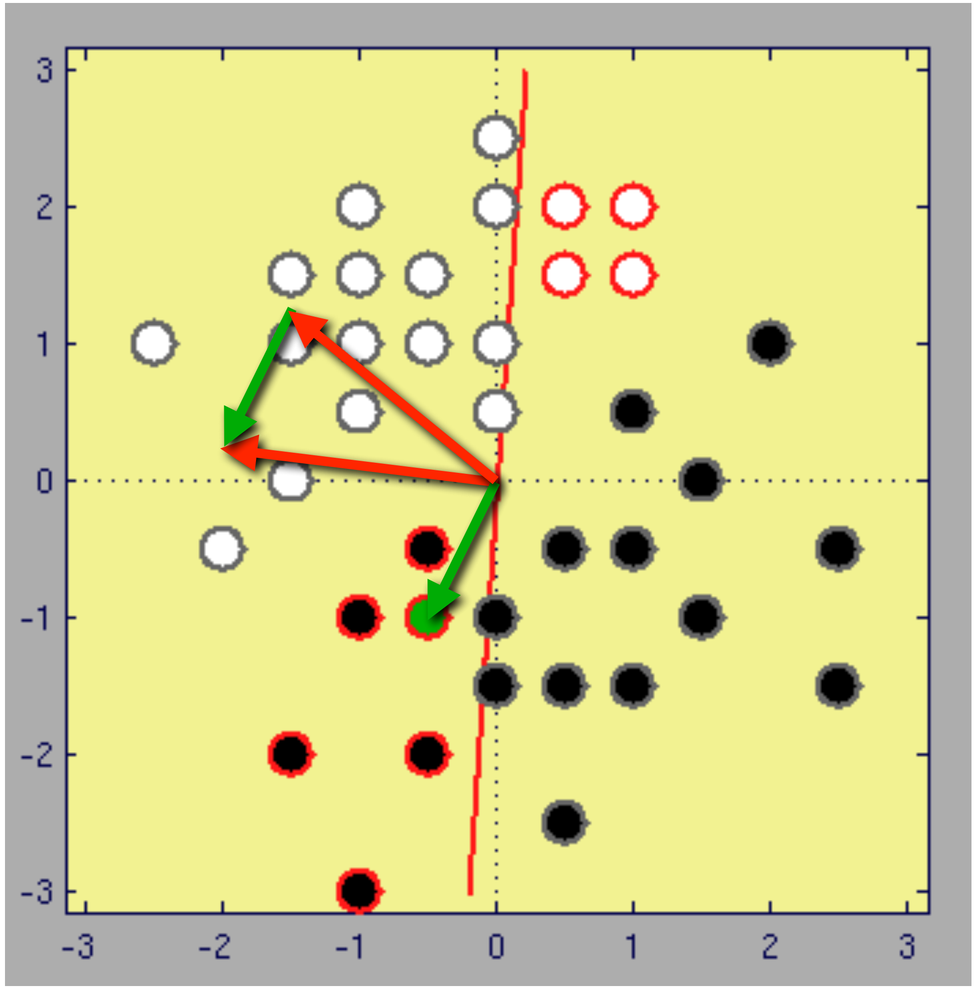
Example of Perceptron Learning



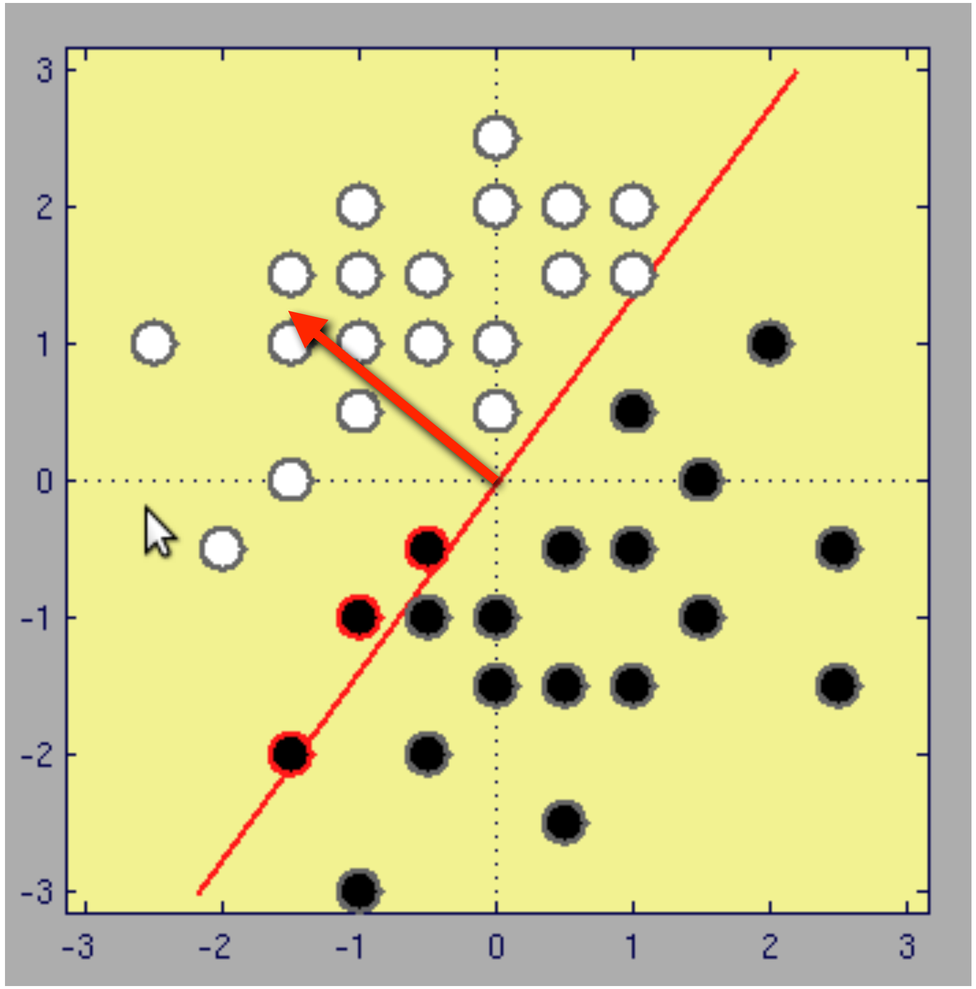
Example of Perceptron Learning



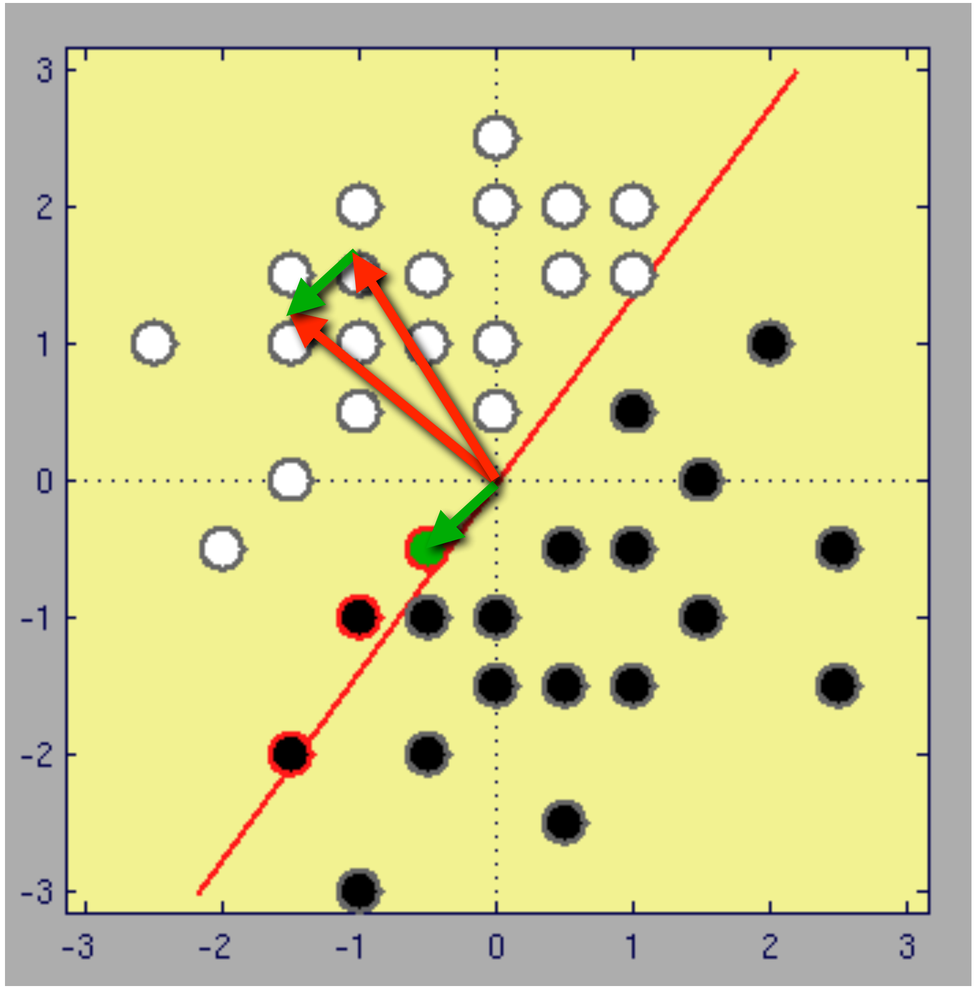
Example of Perceptron Learning



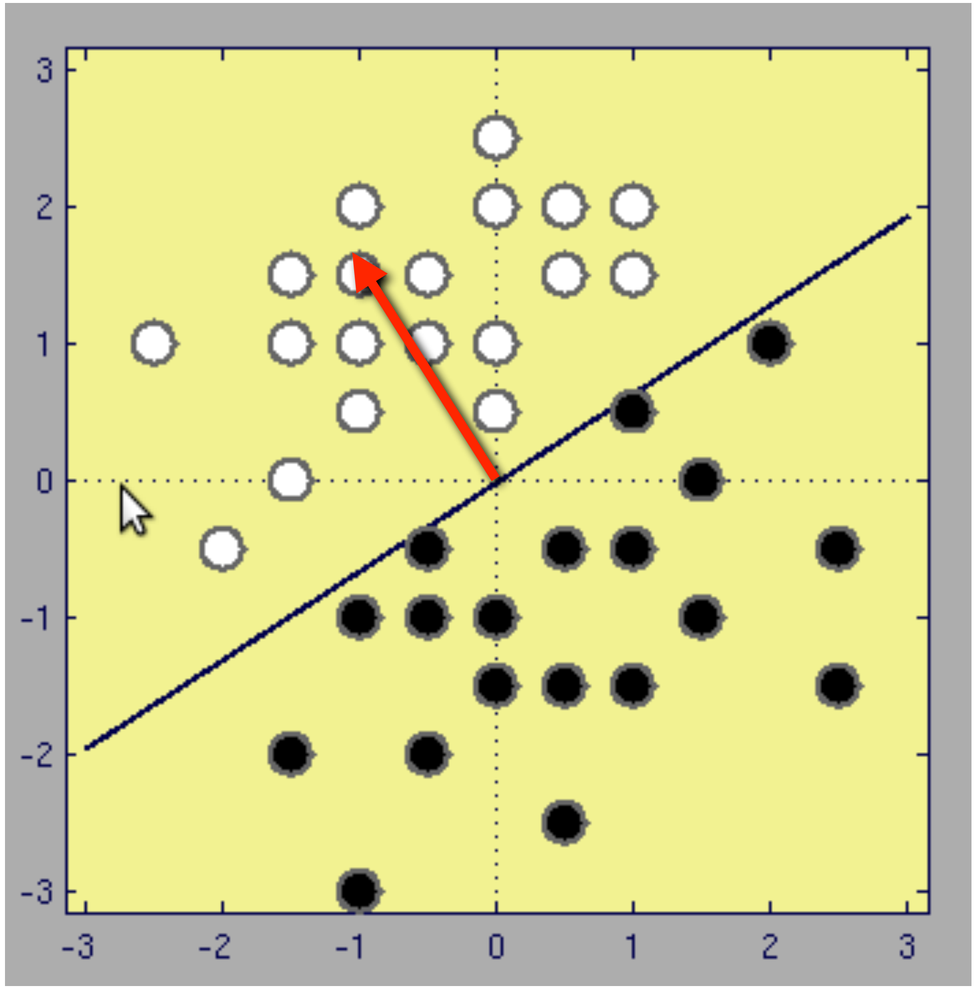
Example of Perceptron Learning



Example of Perceptron Learning

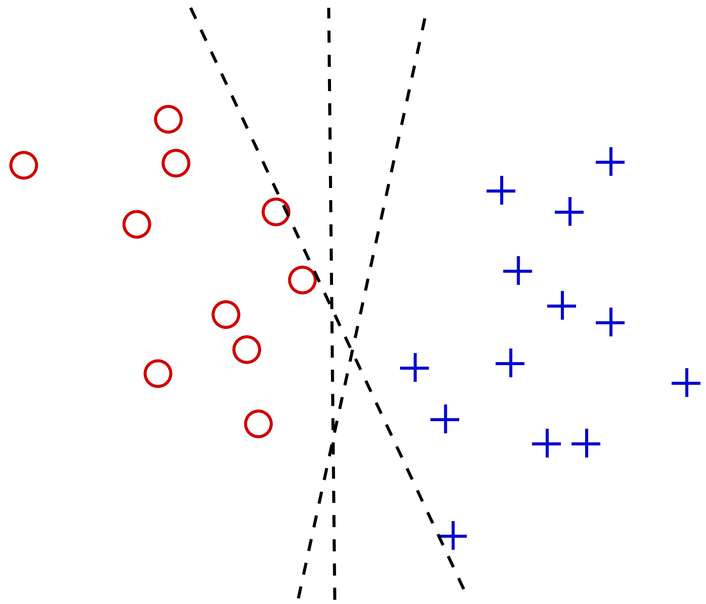


Example of Perceptron Learning

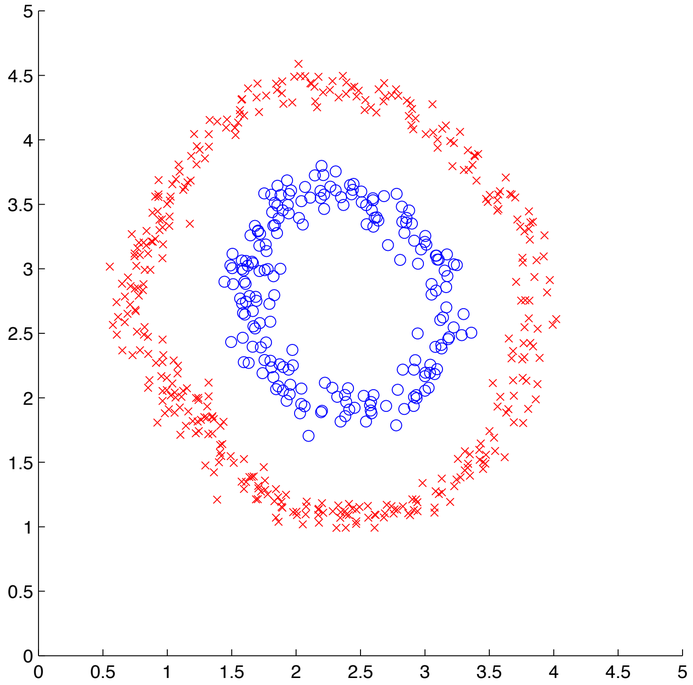


Perceptron Limitations

- Perceptrons + linear + softmax regressors are limited to data that are linearly separable, e.g.,



Linearly separable

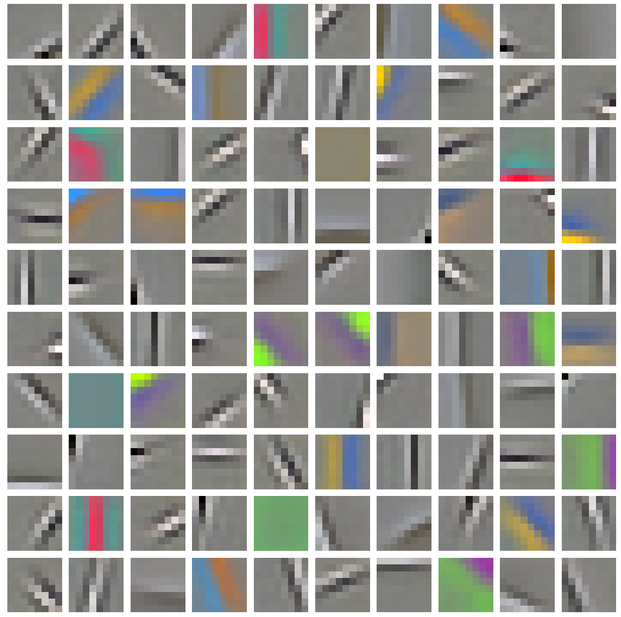


Not linearly separable

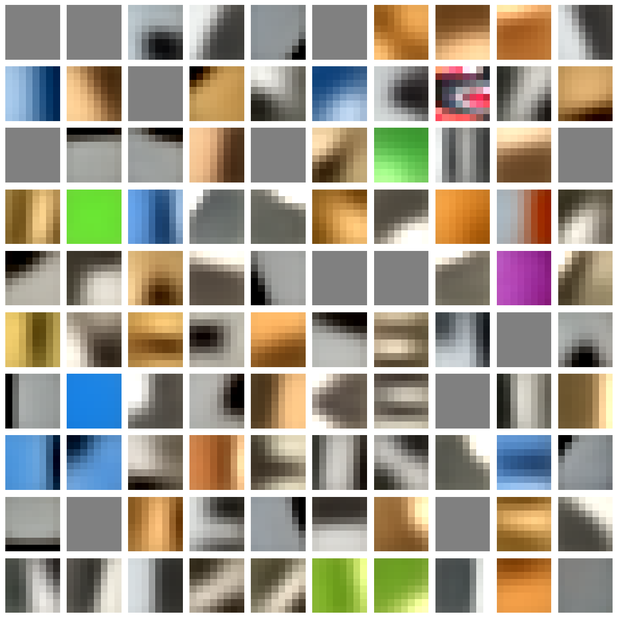
Could we extract features to make the data linearly separable?

CIFAR10 Feature Extraction

- So far, we used RGB pixels as the input to our classifier
- Feature extraction can improve results by a lot
- e.g., Coates et al. achieve 79.6% accuracy on CIFAR10 with a features based on k-means of whitened image patches



k-means, whitened

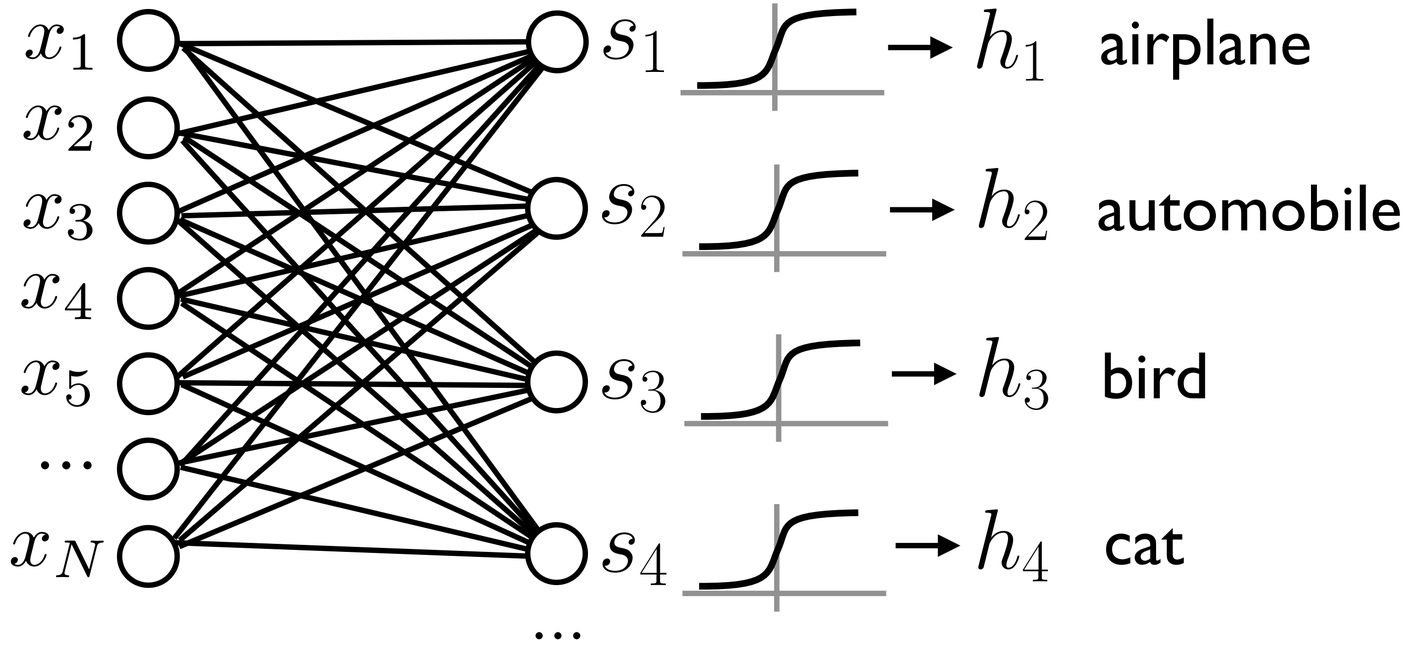


k-means, raw RGB

[Coates et al. 2011]

Linear = Fully Connected Layer

- Note that our linear matrix multiplication classifier is equivalent to a fully connected layer in a neural network

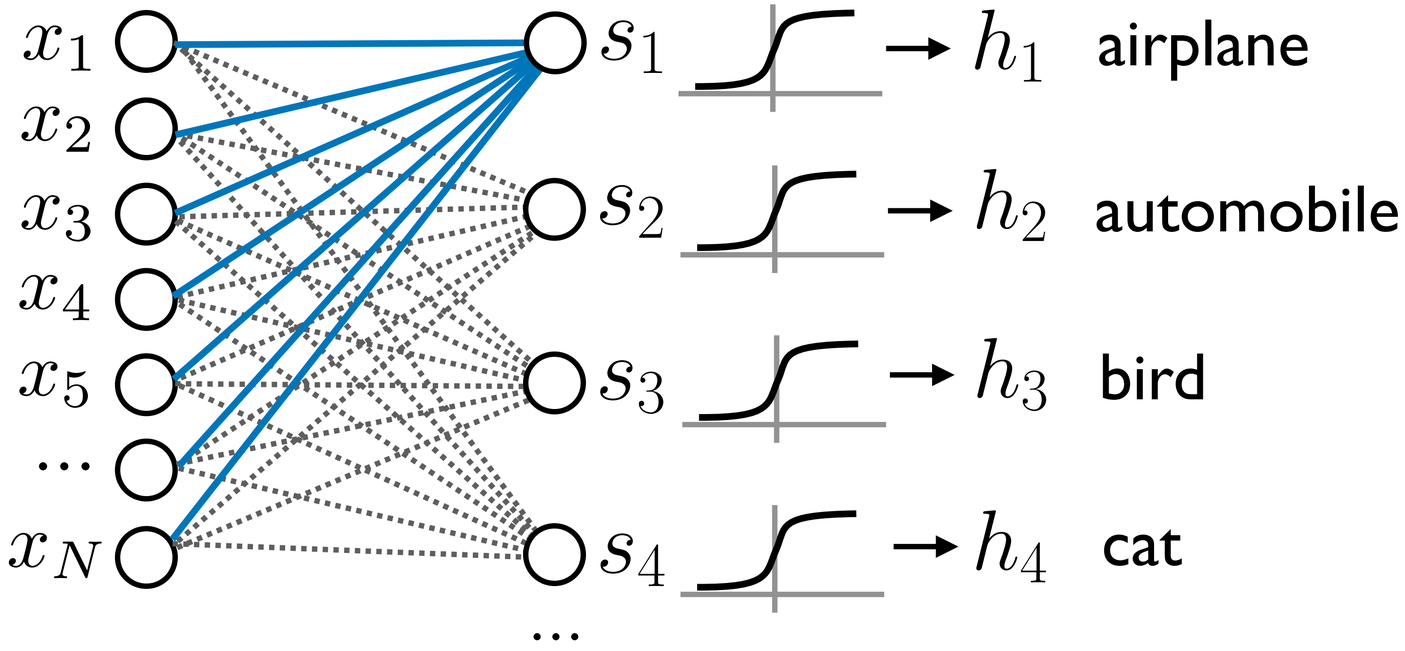


- Typically, we'll also add a bias term b

$$\mathbf{h} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b})$$

Linear = Fully Connected Layer

- Note that our linear matrix multiplication classifier is equivalent to a fully connected layer in a neural network

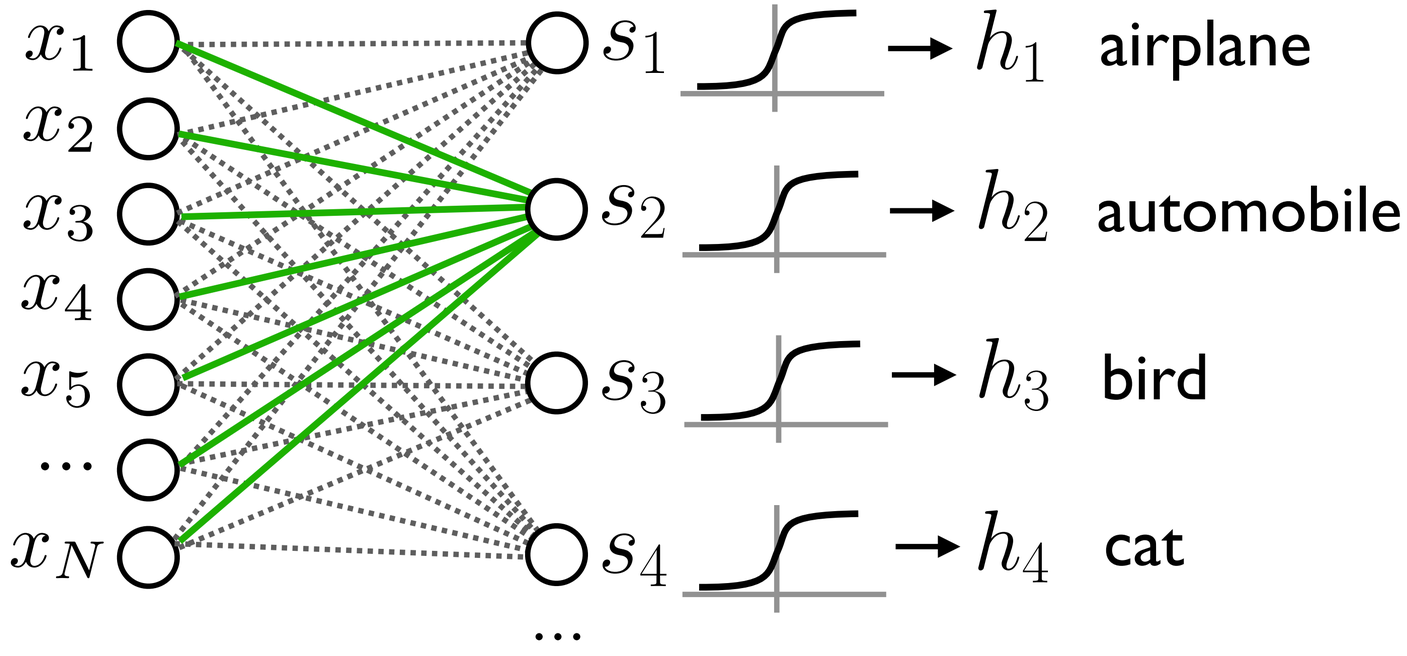


- Typically, we'll also add a bias term b

$$\mathbf{h} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b})$$

Linear = Fully Connected Layer

- Note that our linear matrix multiplication classifier is equivalent to a fully connected layer in a neural network

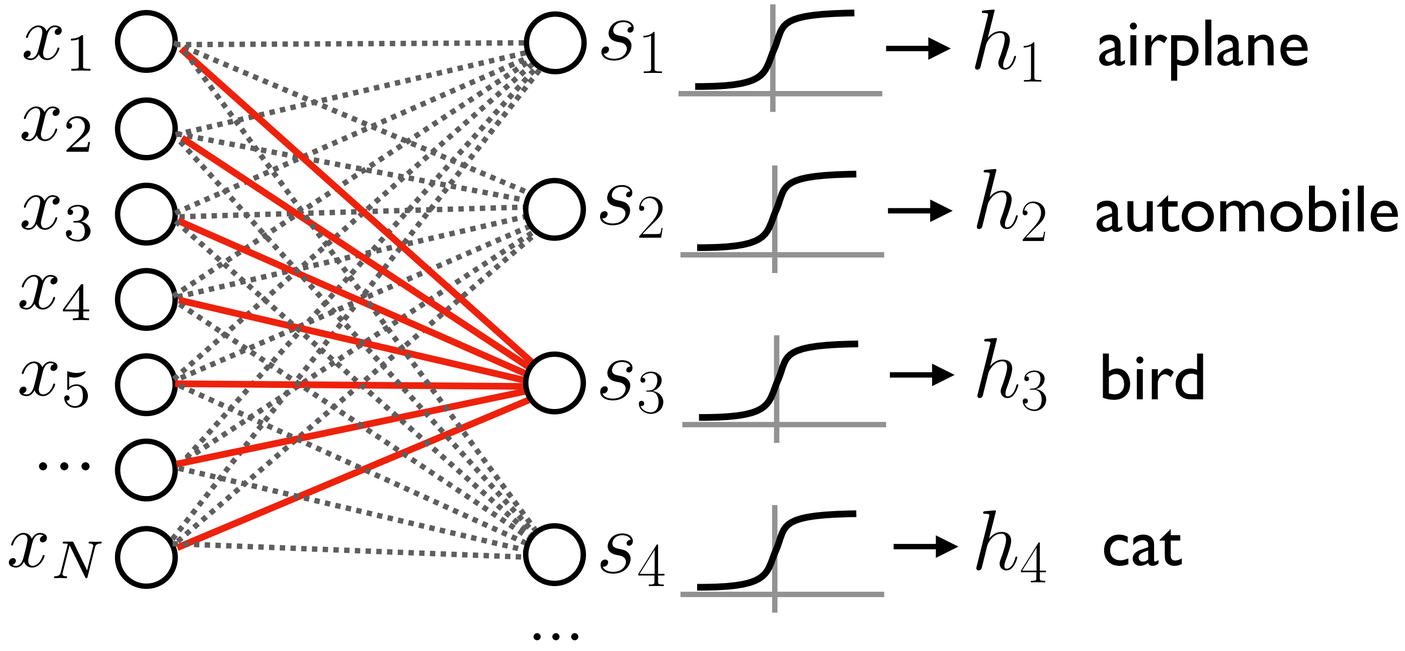


- Typically, we'll also add a bias term b

$$\mathbf{h} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b})$$

Linear = Fully Connected Layer

- Note that our linear matrix multiplication classifier is equivalent to a fully connected layer in a neural network

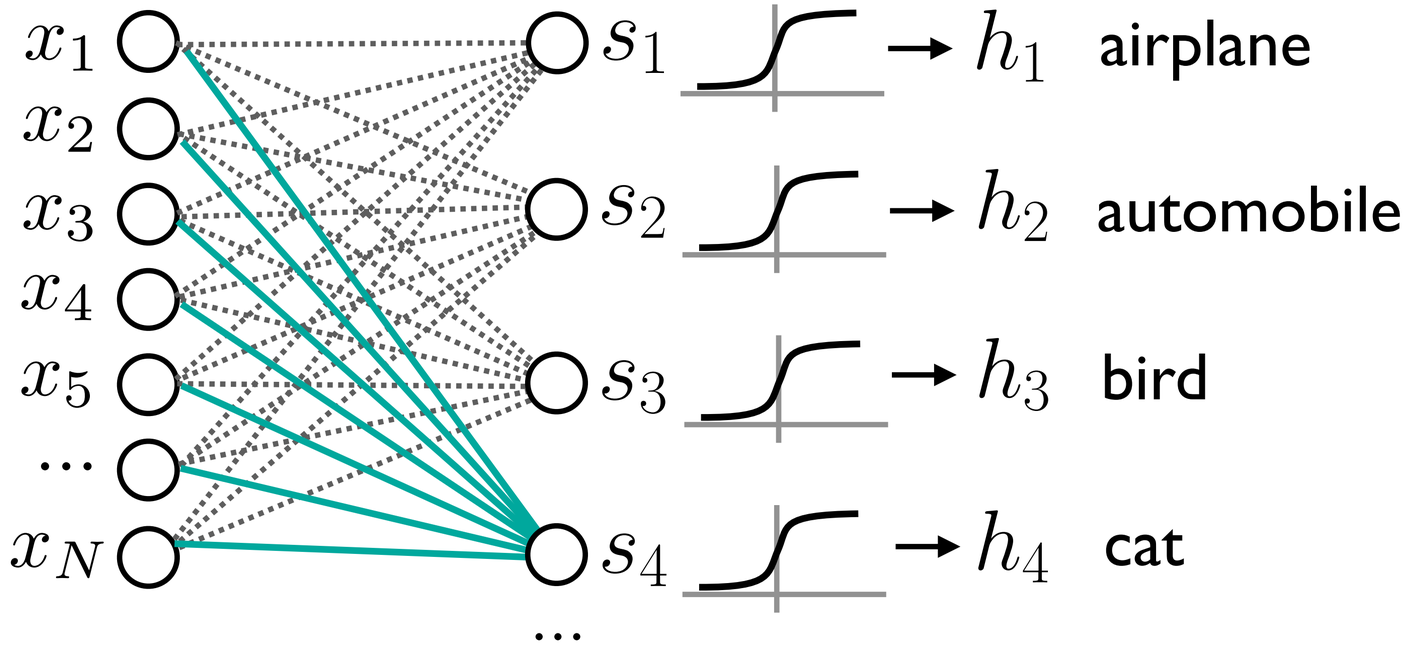


- Typically, we'll also add a bias term b

$$\mathbf{h} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b})$$

Linear = Fully Connected Layer

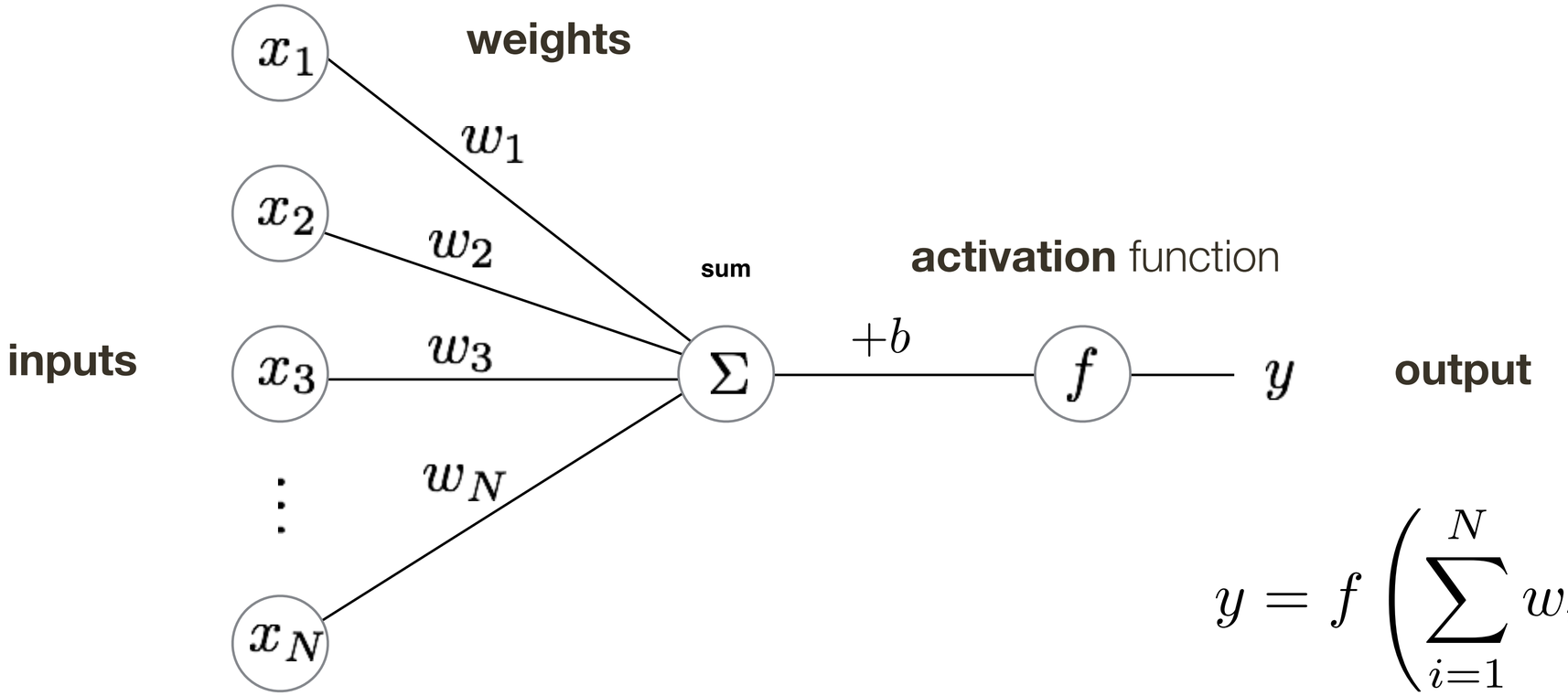
- Note that our linear matrix multiplication classifier is equivalent to a fully connected layer in a neural network



- Typically, we'll also add a bias term b

$$\mathbf{h} = \sigma(\mathbf{W}^T\mathbf{x} + \mathbf{b})$$

A **Neuron**



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

— The basic unit of computation in a neural network is a neuron.

— A neuron accepts some number of input signals, computes their weighted sum, and applies an **activation function** (or **non-linearity**) to the sum.

— Common activation functions include sigmoid and rectified linear unit (ReLU)

Activation Function: **Sigmoid**

$$f(x) = 1/(1 + e^{-x})$$

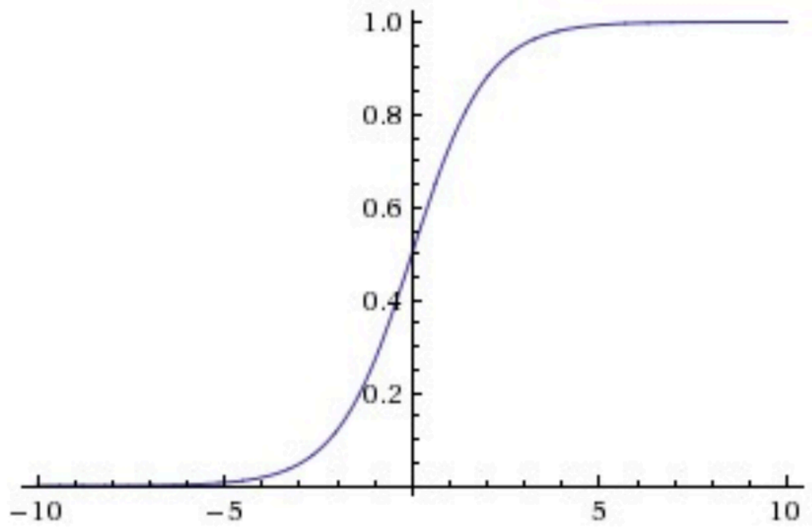


Figure credit: Fei-Fei and Karpathy

Common in many early neural networks

Biological analogy to saturated firing rate of neurons

Maps the input to the range [0,1]

Activation Function: **ReLU** (Rectified Linear Unit)

$$f(x) = \max(0, x)$$

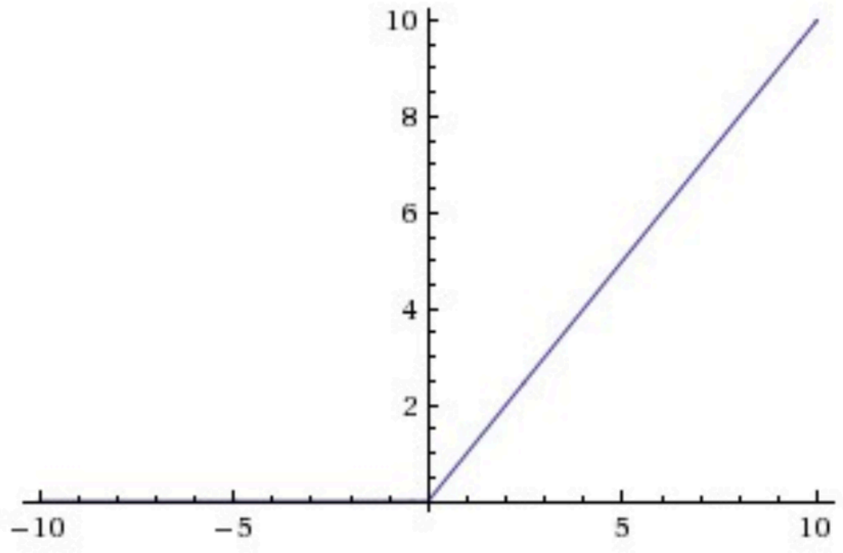


Figure credit: Fei-Fei and Karpathy

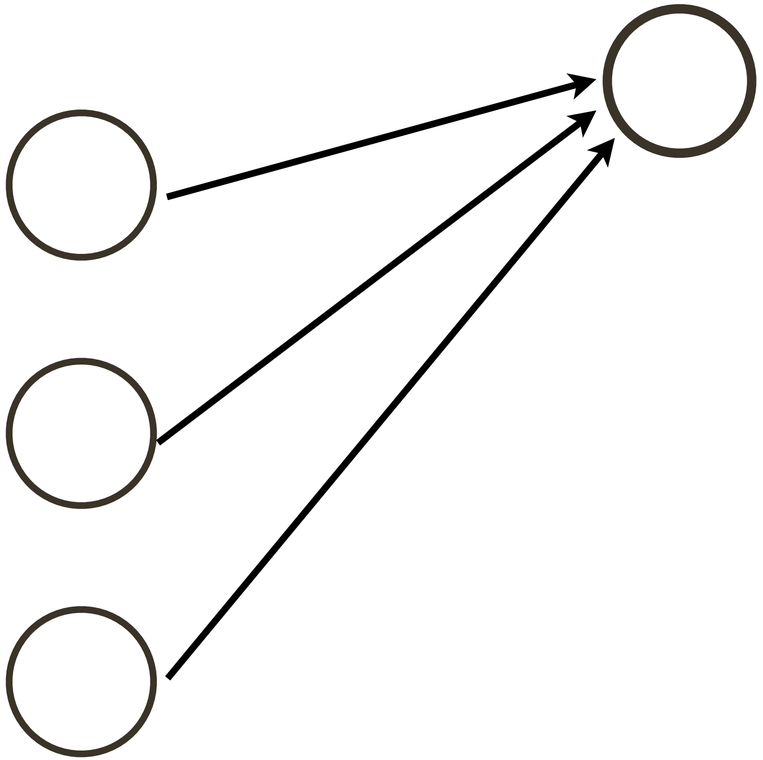
Maintains good gradient flow in networks, prevents vanishing gradient problem

Very commonly used in interior (hidden) layers of neural nets

Why can't we have **linear** activation functions?

Neural Network

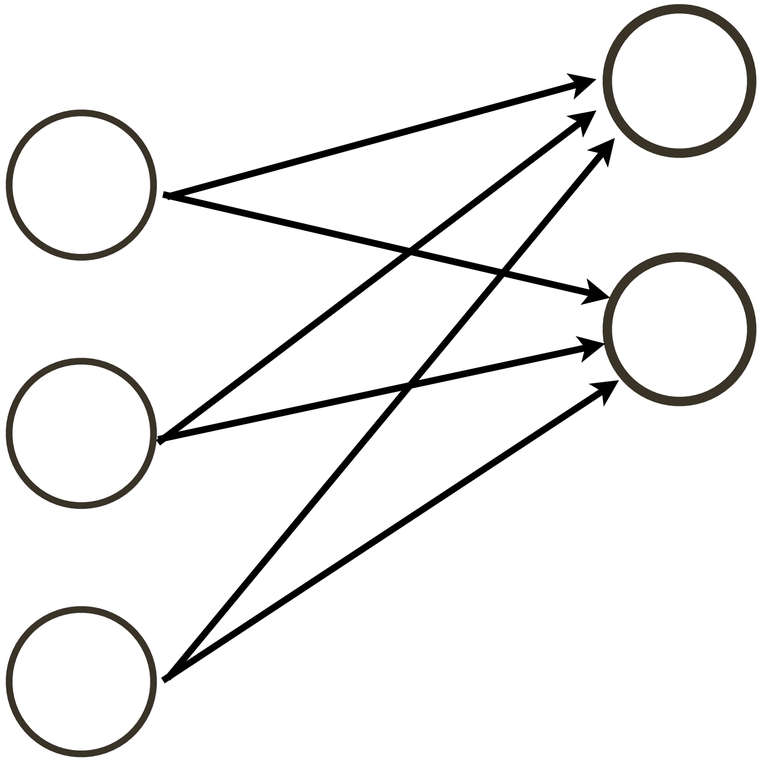
Connect a bunch of neurons together — a collection of connected neurons



‘one neuron’

Neural Network

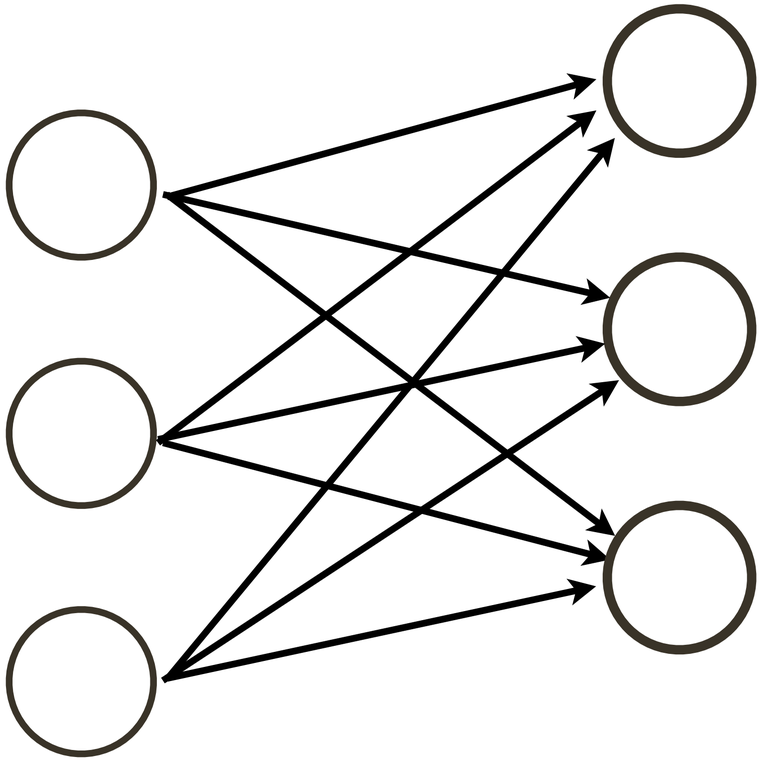
Connect a bunch of neurons together — a collection of connected neurons



‘two neurons’

Neural Network

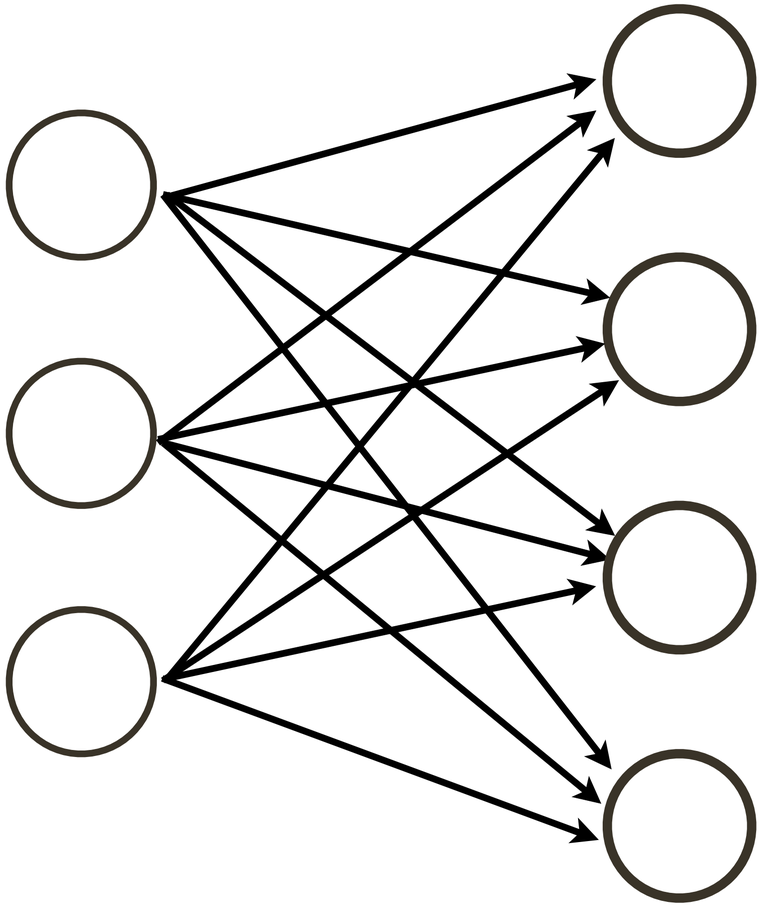
Connect a bunch of neurons together — a collection of connected neurons



‘three neurons’

Neural Network

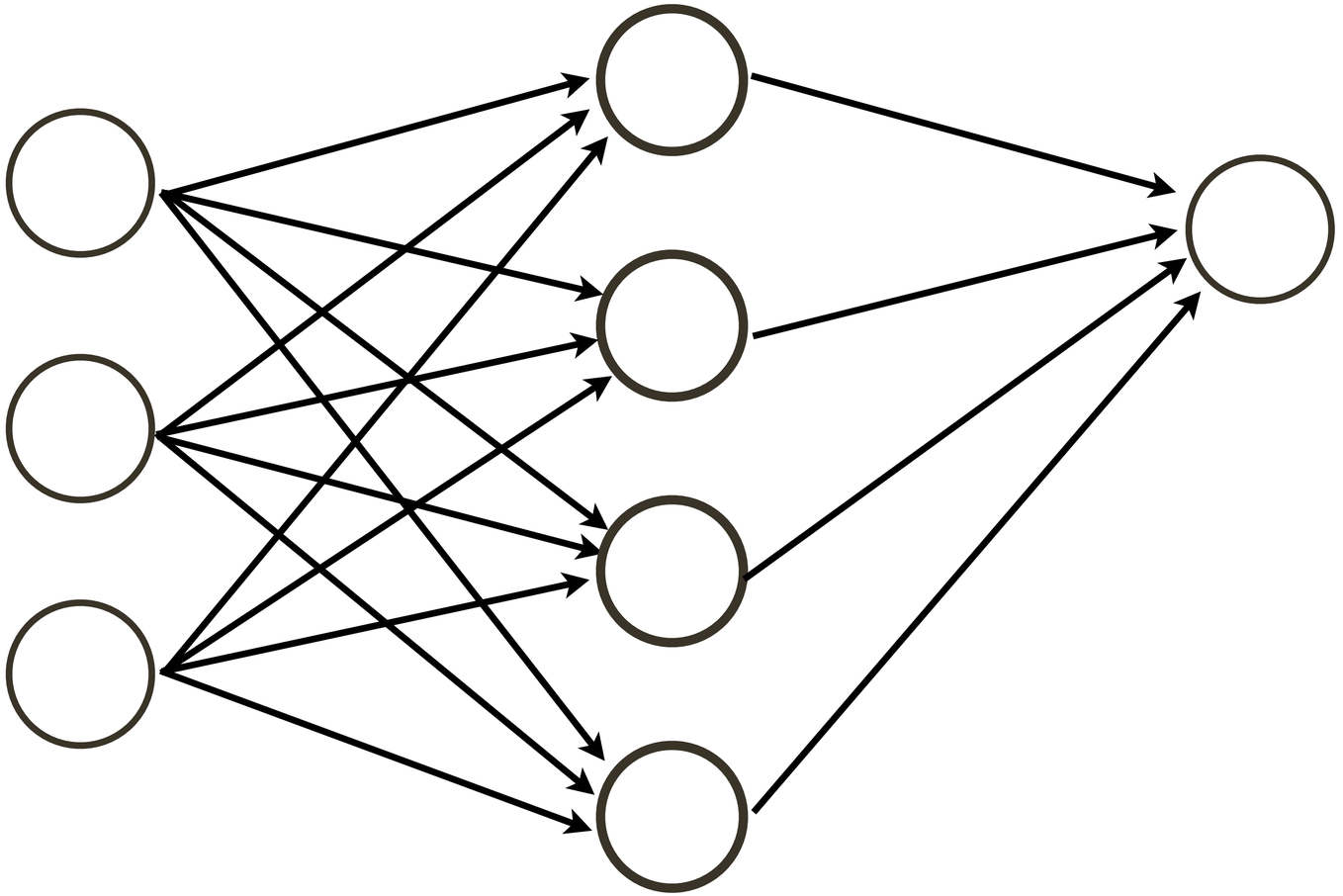
Connect a bunch of neurons together — a collection of connected neurons



‘four neurons’

Neural Network

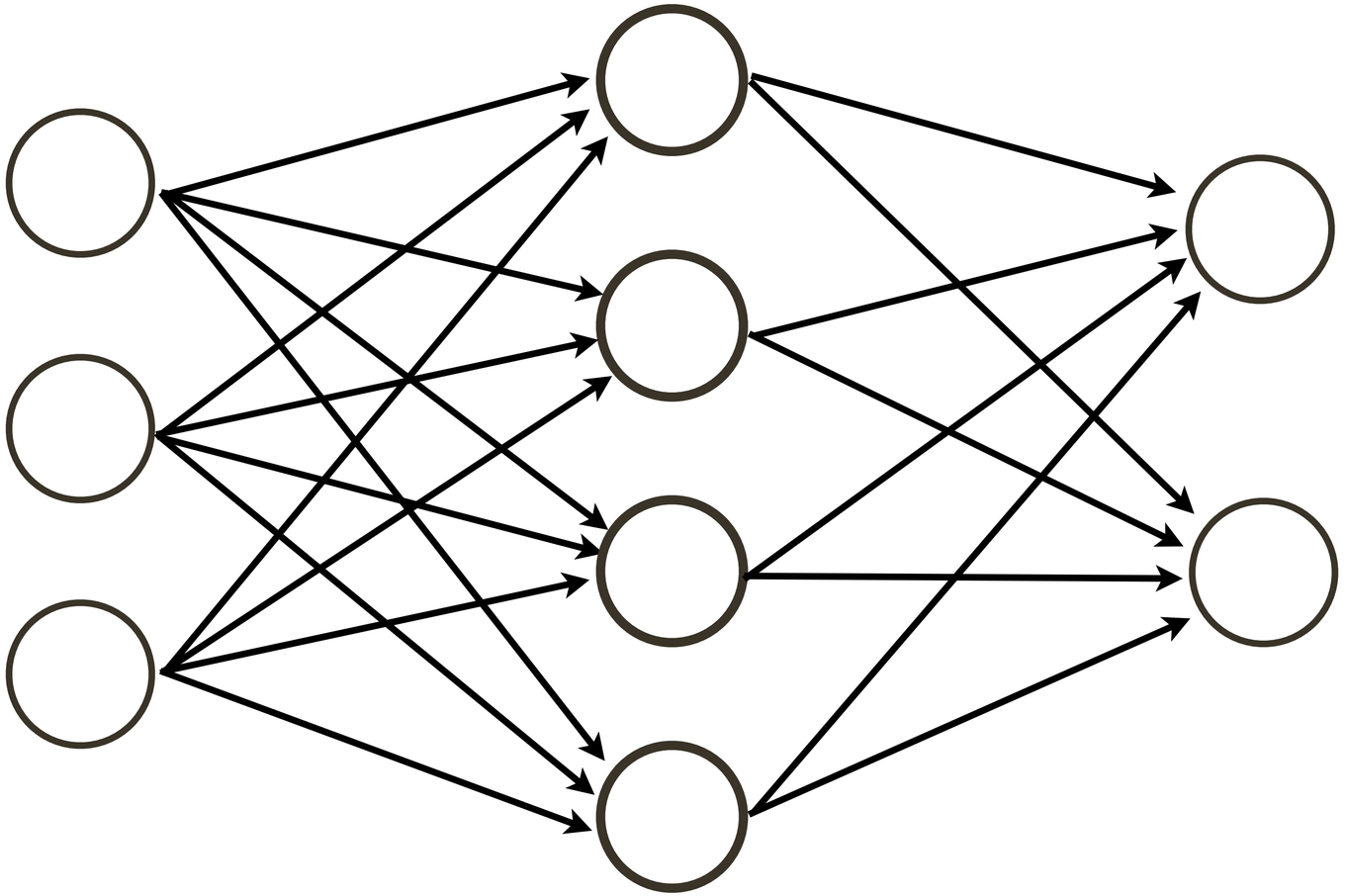
Connect a bunch of neurons together — a collection of connected neurons



‘five neurons’

Neural Network

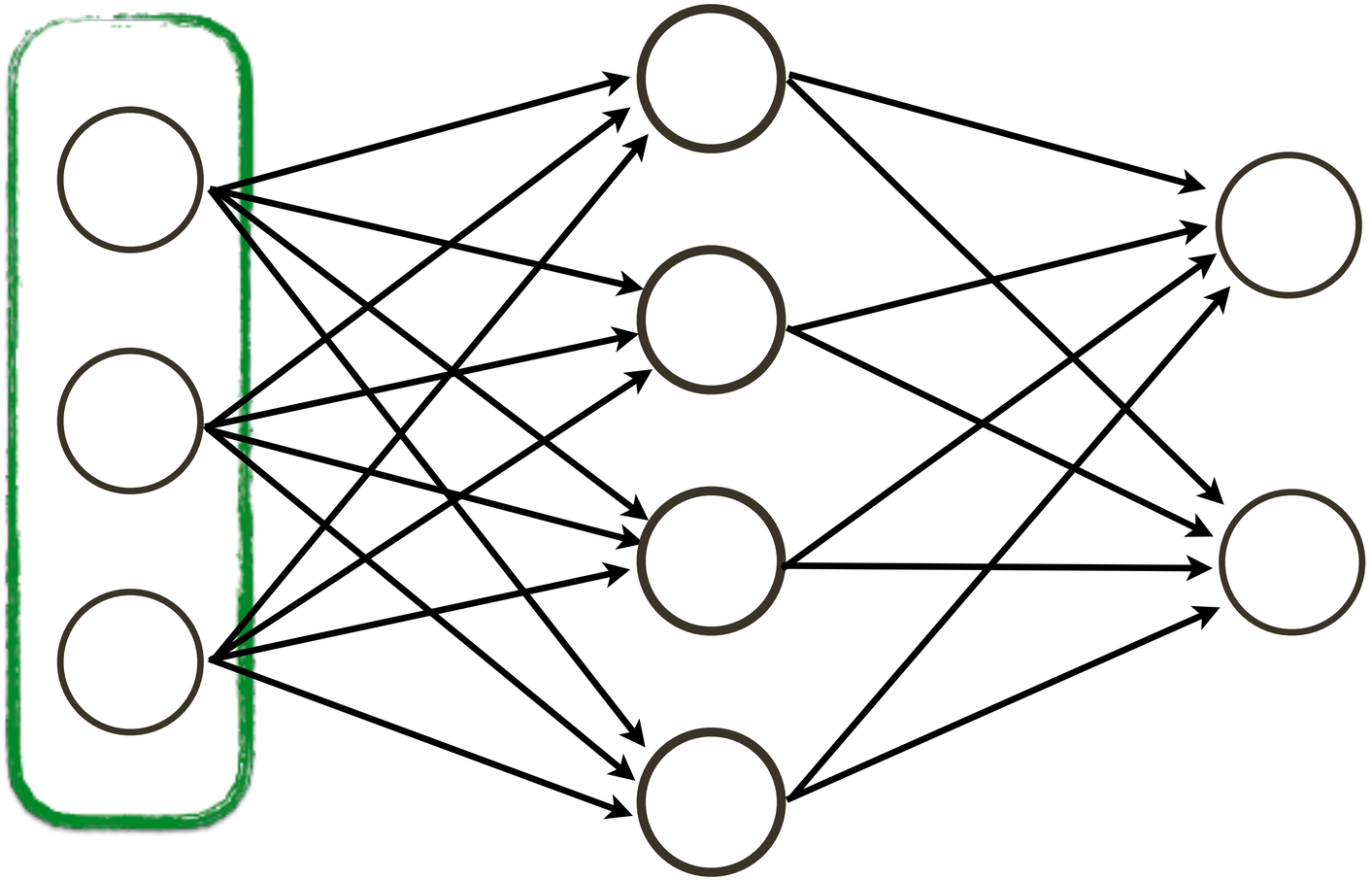
Connect a bunch of neurons together — a collection of connected neurons



‘six neurons’

Neural Network: **Terminology**

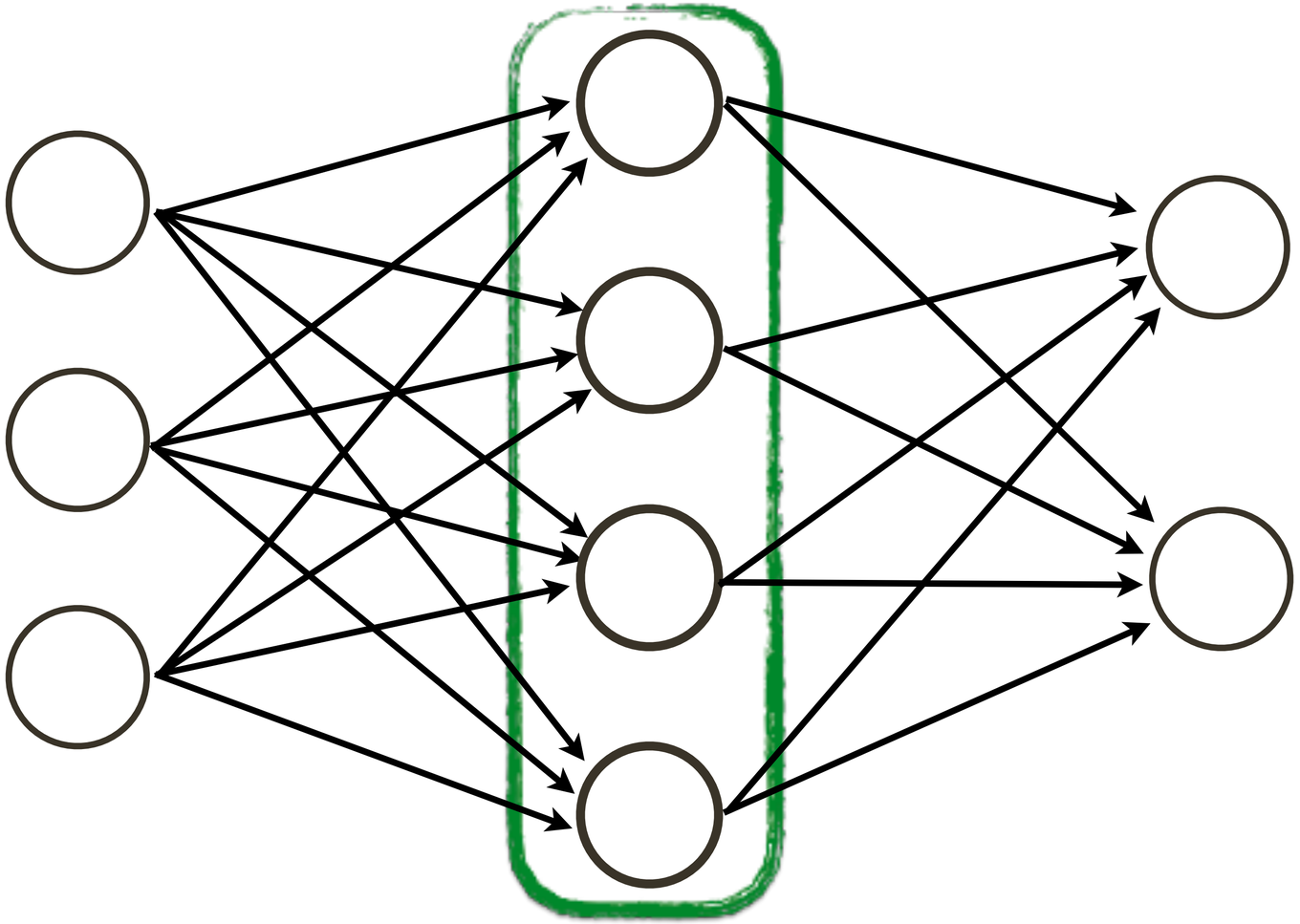
‘**input**’ layer



Neural Network: **Terminology**

‘**hidden**’ layer

‘**input**’ layer

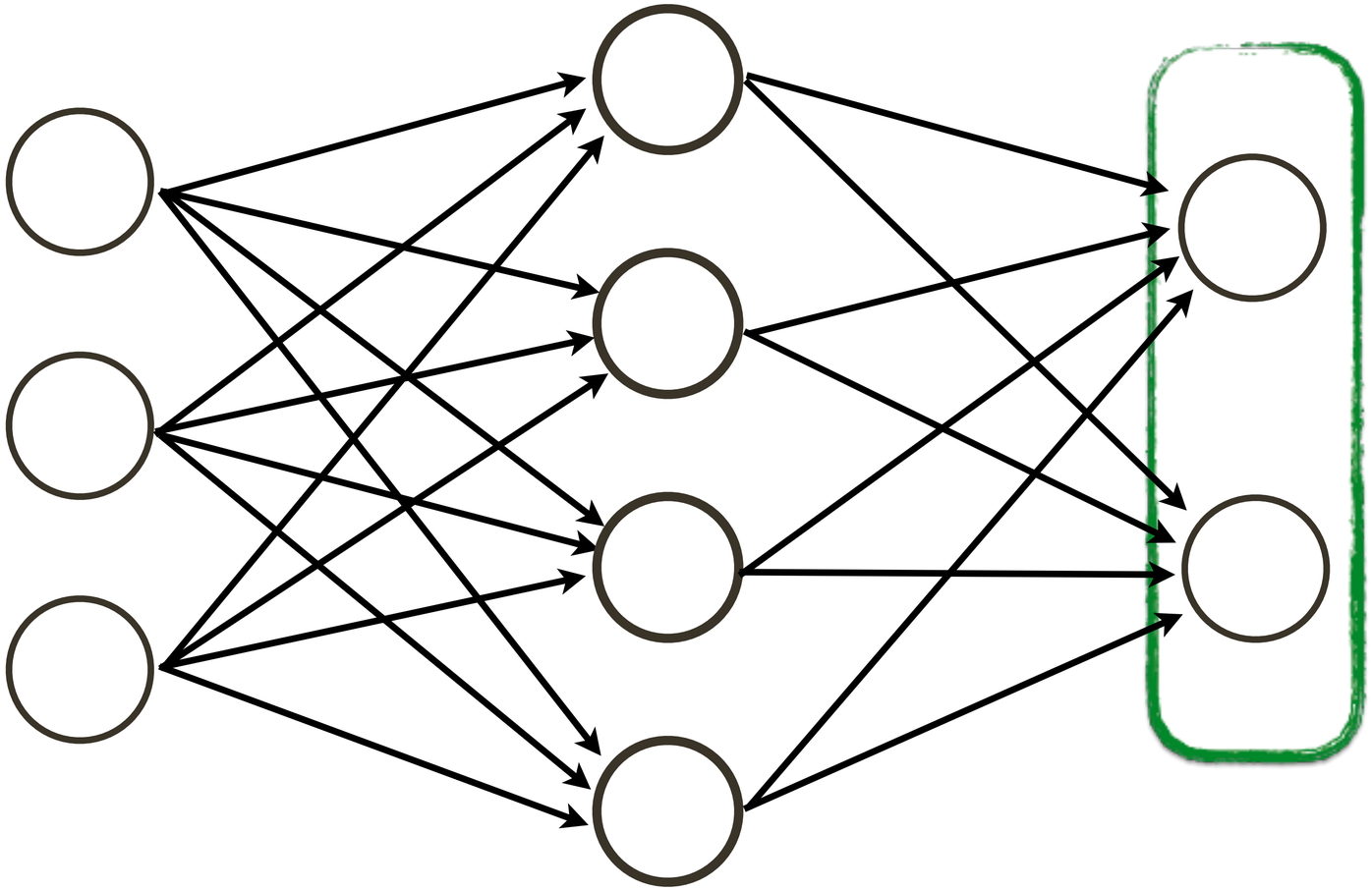


Neural Network: **Terminology**

‘**hidden**’ layer

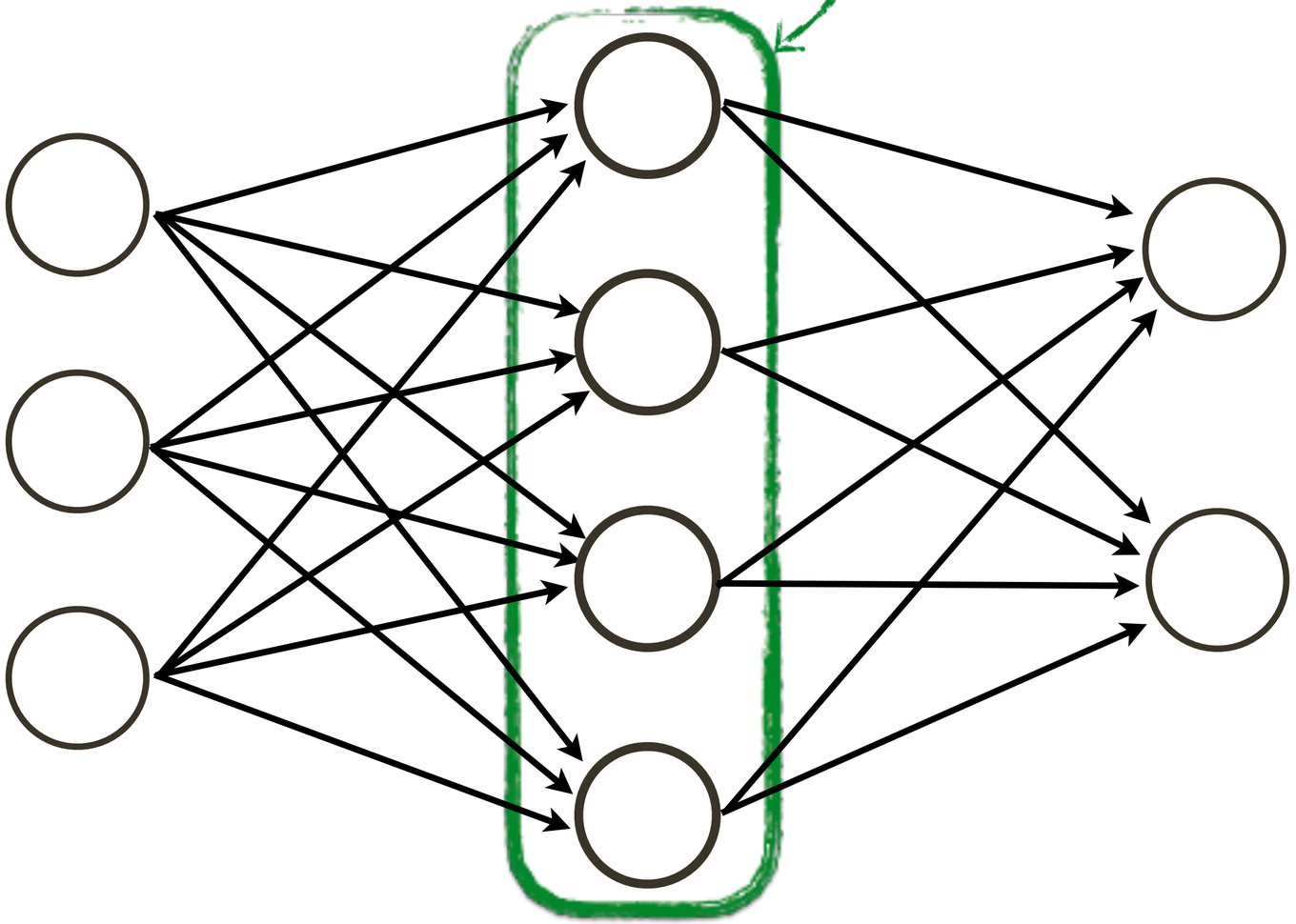
‘**input**’ layer

‘**output**’ layer

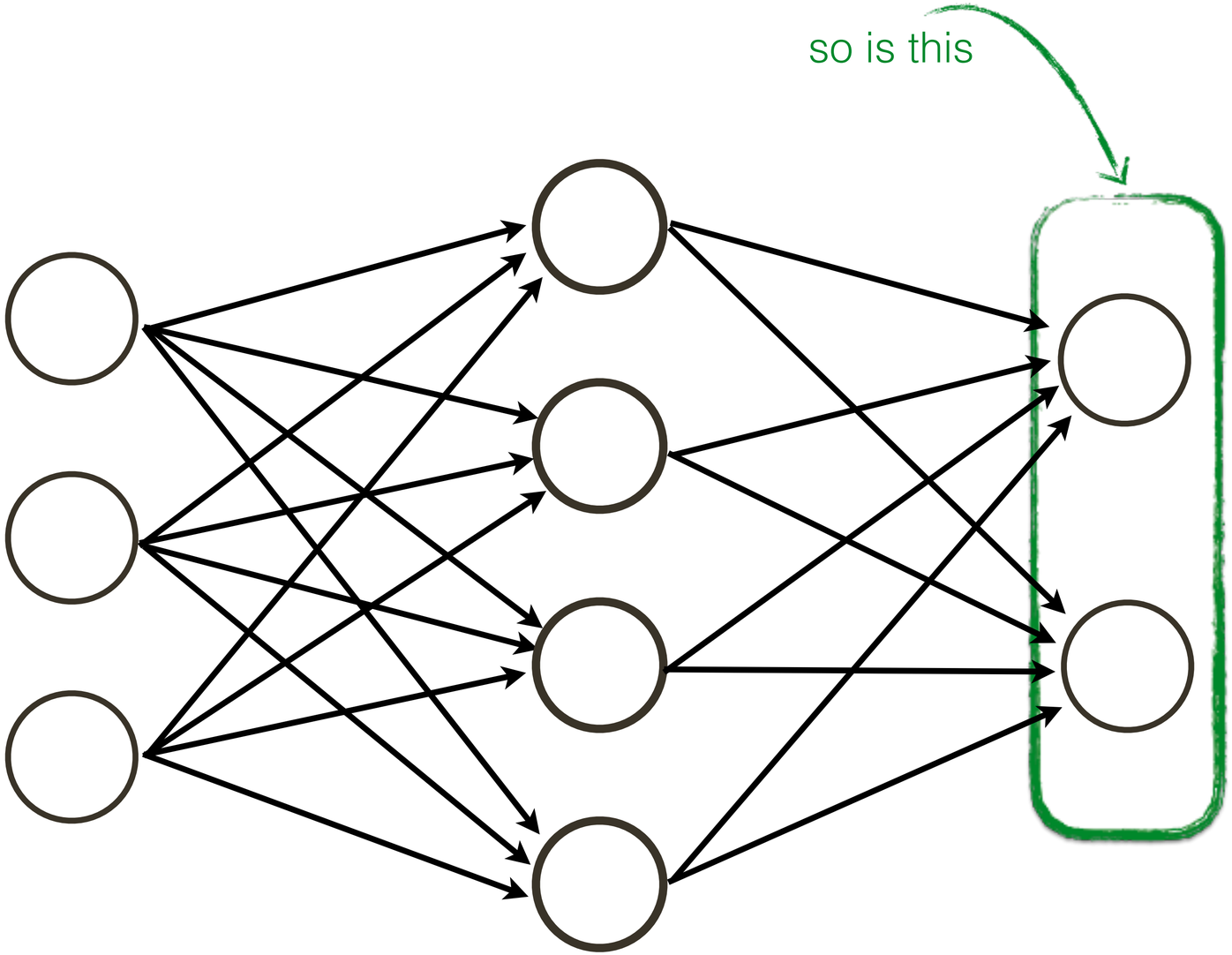


Neural Network: **Terminology**

this layer is a
‘**fully connected layer**’

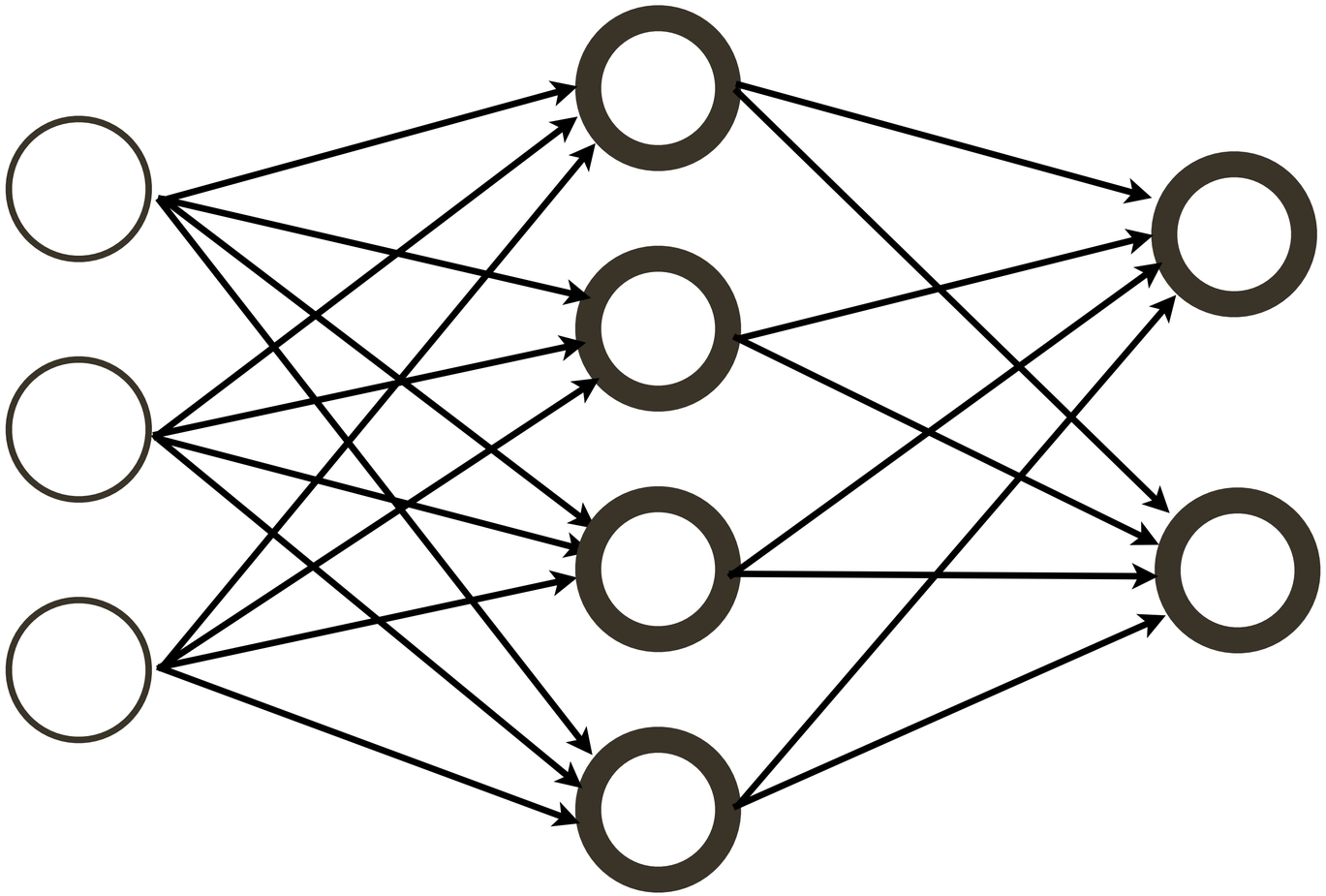


Neural Network: **Terminology**



Neural Network

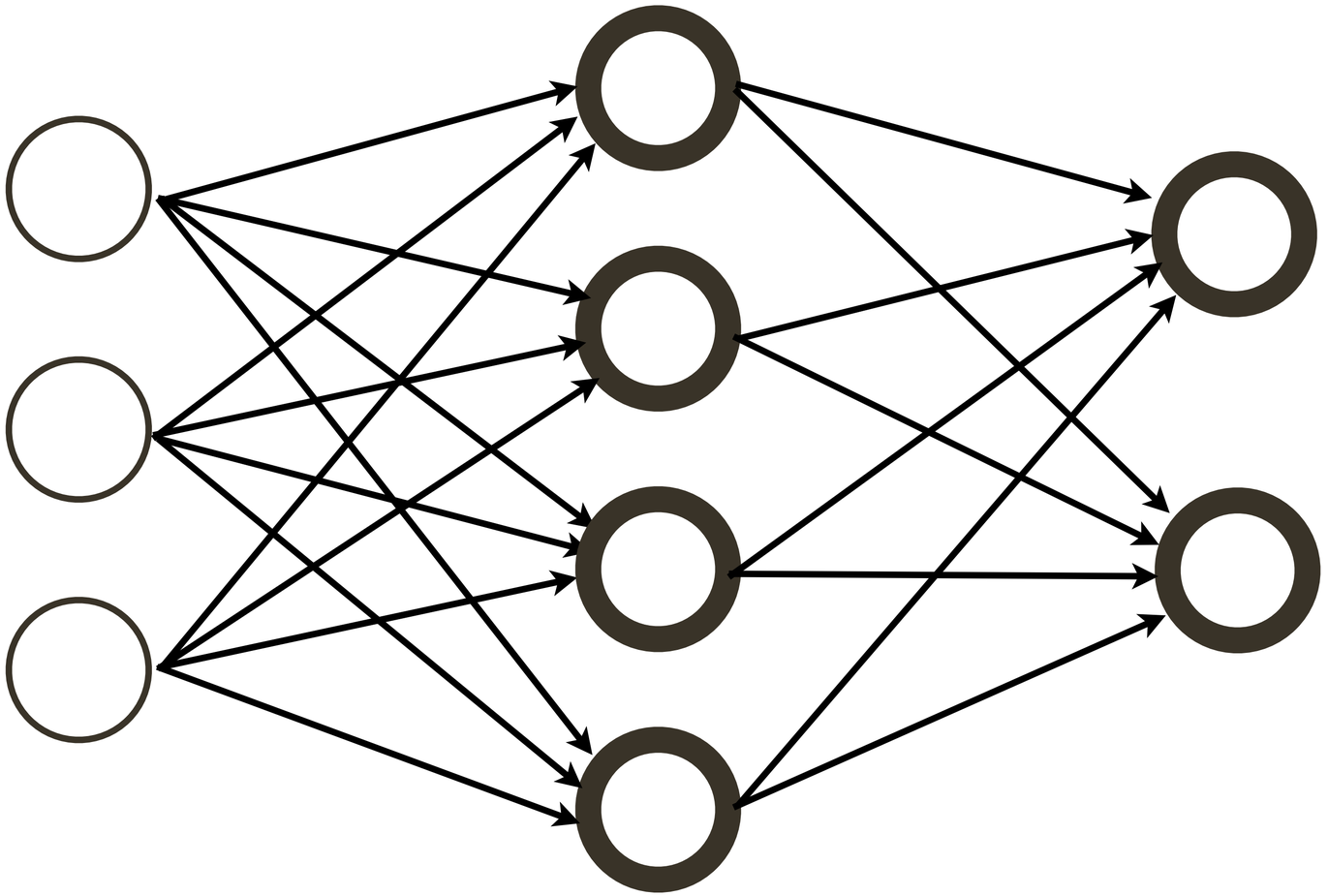
How many neurons? 4+2 = 6



Neural Network

How many neurons? 4+2 = 6

How many weights?

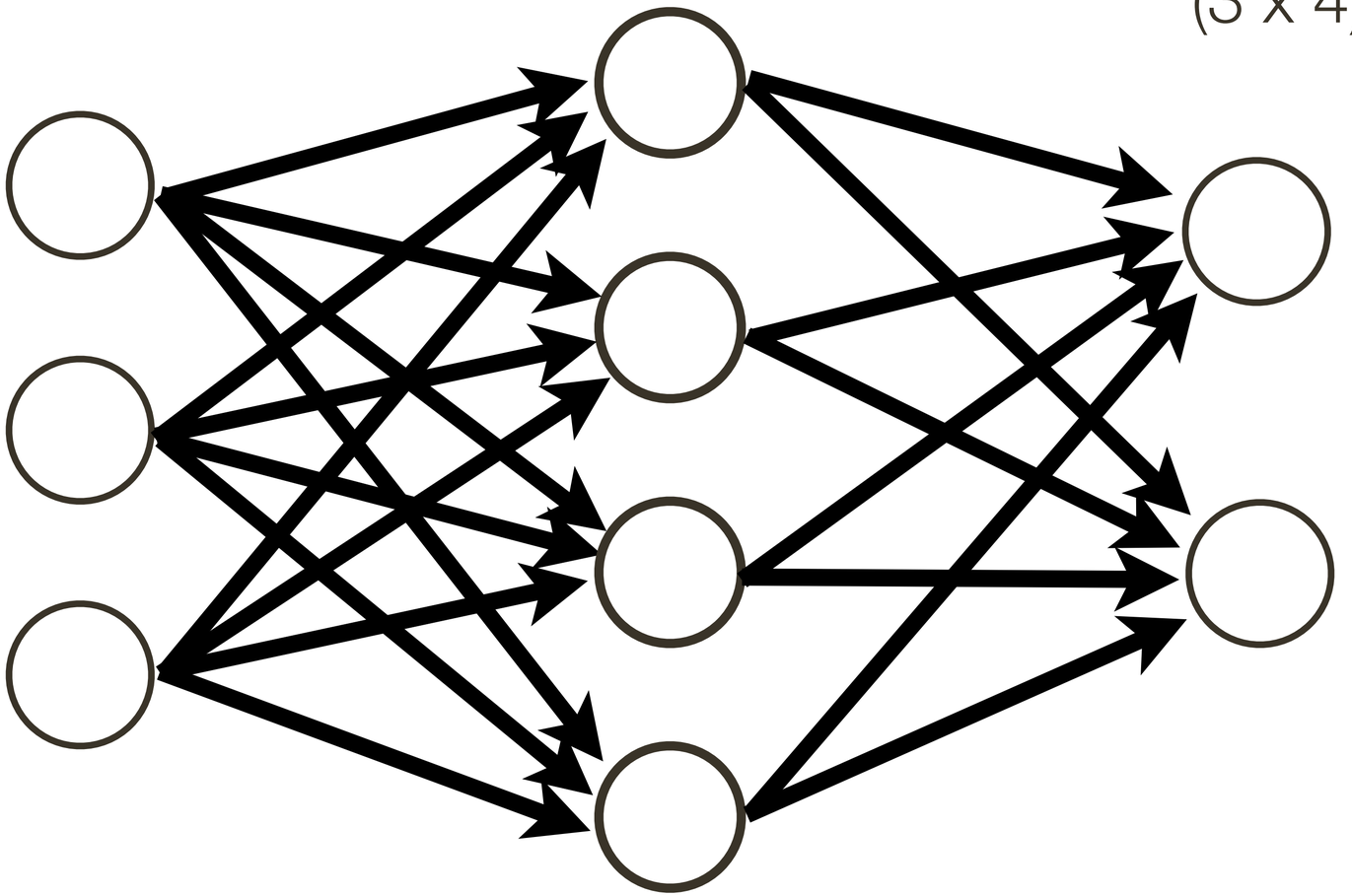


Neural Network

How many neurons? 4+2 = 6

How many weights?

(3 x 4) + (4 x 2) = 20

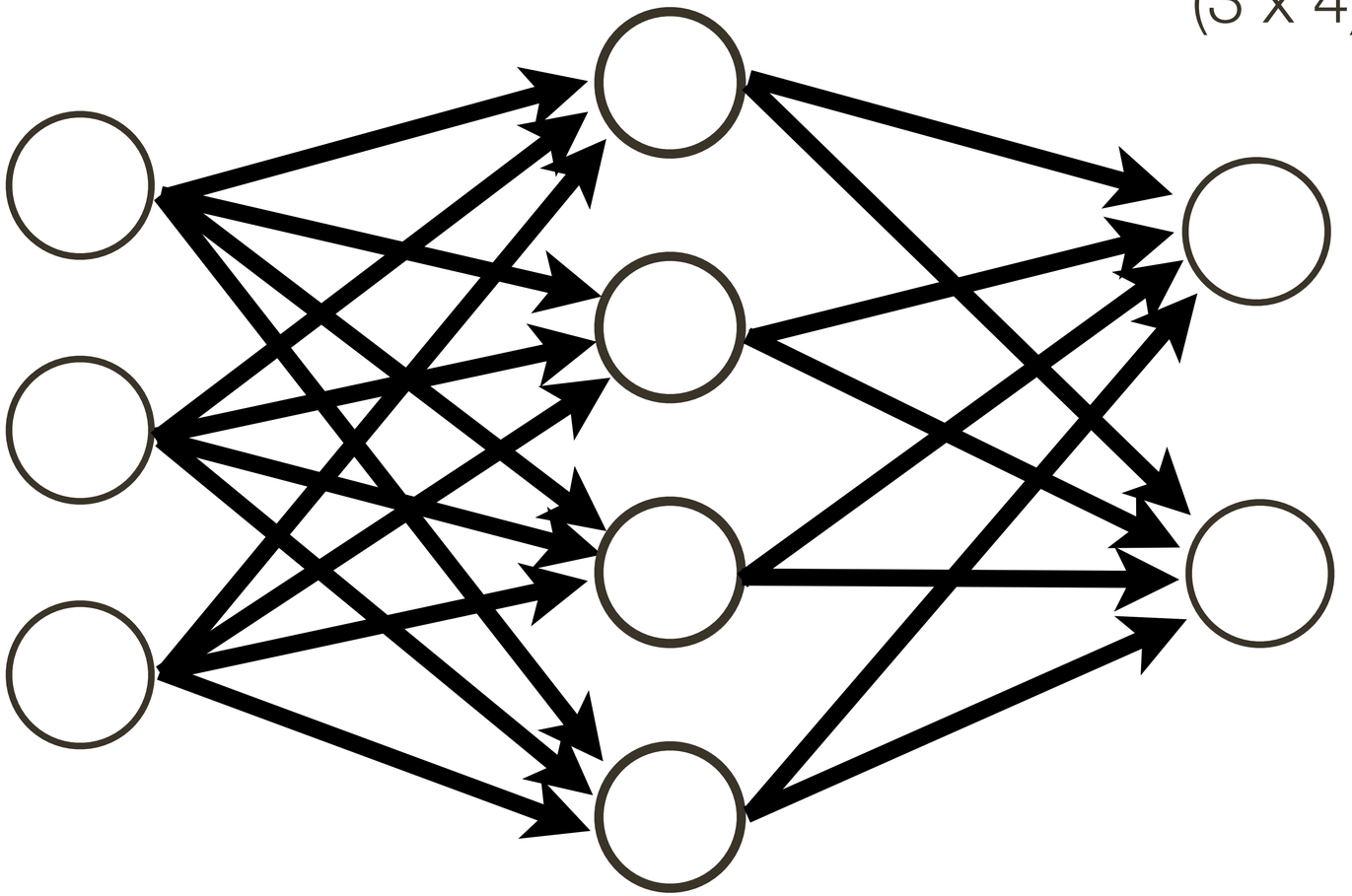


Neural Network

How many neurons? 4+2 = 6

How many weights?

(3 x 4) + (4 x 2) = 20



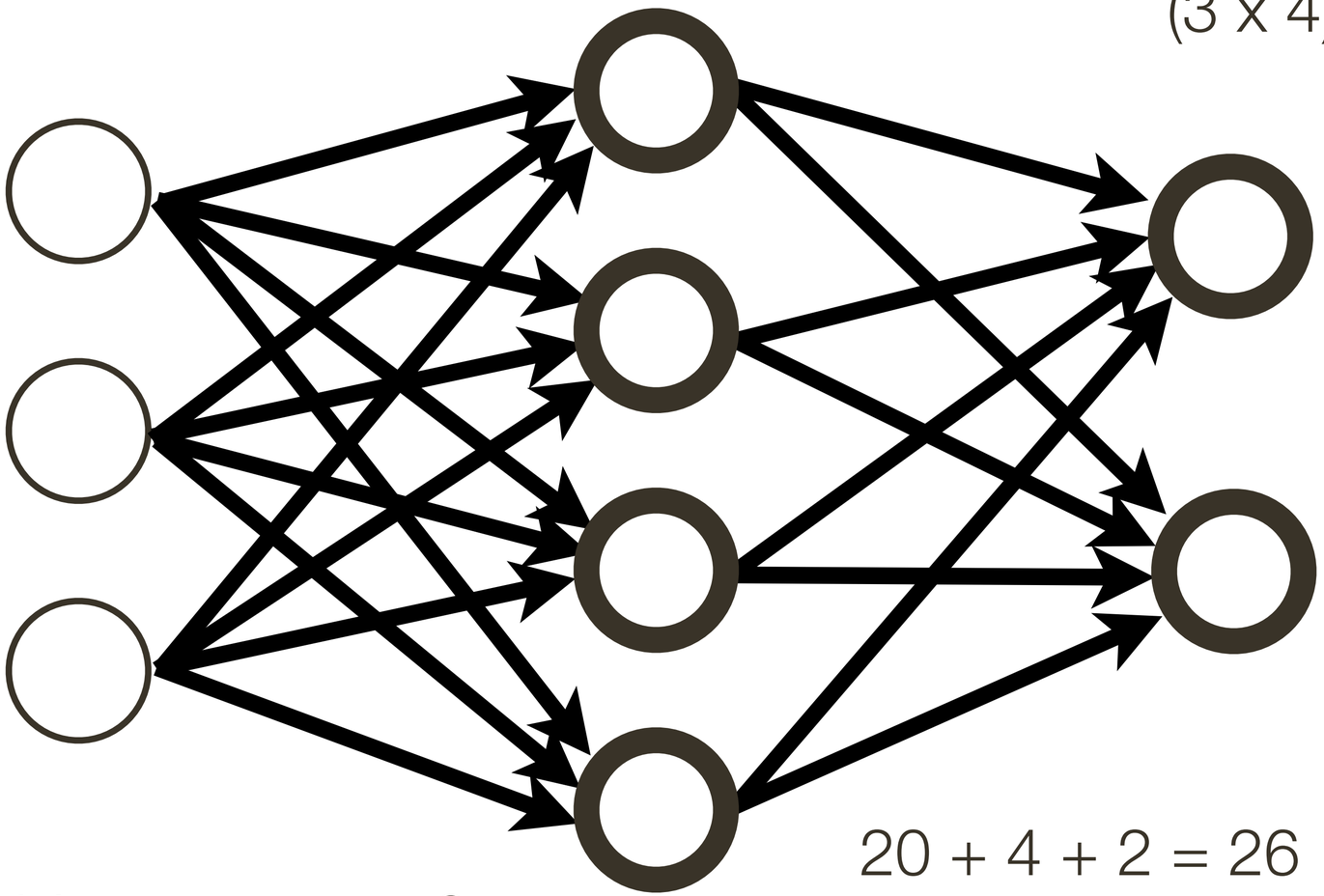
How many learnable parameters?

Neural Network

How many neurons? 4+2 = 6

How many weights?

(3 x 4) + (4 x 2) = 20



20 + 4 + 2 = 26

bias terms

How many learnable parameters?

Neural Network **Intuition**

Question: What is a Neural Network?

Answer: Complex mapping from an input (vector) to an output (vector)

Question: What class of functions should be considered for this mapping?

Answer: Compositions of simpler functions (a.k.a. layers)? We will talk more about what specific functions next …

Question: What does a hidden unit do?

Answer: It can be thought of as classifier or a feature.

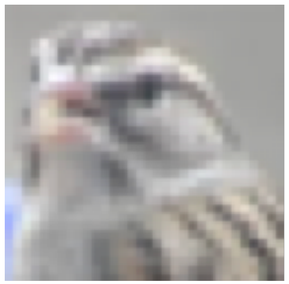
Question: Why have many layers?

Answer: 1) More layers = more complex functional mapping

2) More efficient due to distributed representation

* slide from Marc'Aurelio Renzato

2-Layer **Neural** Network



activations

input data

x_0

x_1

x_2

$w_{00}^{(1)}$

$\cdots$

a_0

a_1

a_2

a_3

$w_{00}^{(2)}$

$\cdots$

“bird”

h_0

h_1

“plane”

$e \leftarrow \begin{bmatrix} t_0 \\ t_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

targets

e.g.,

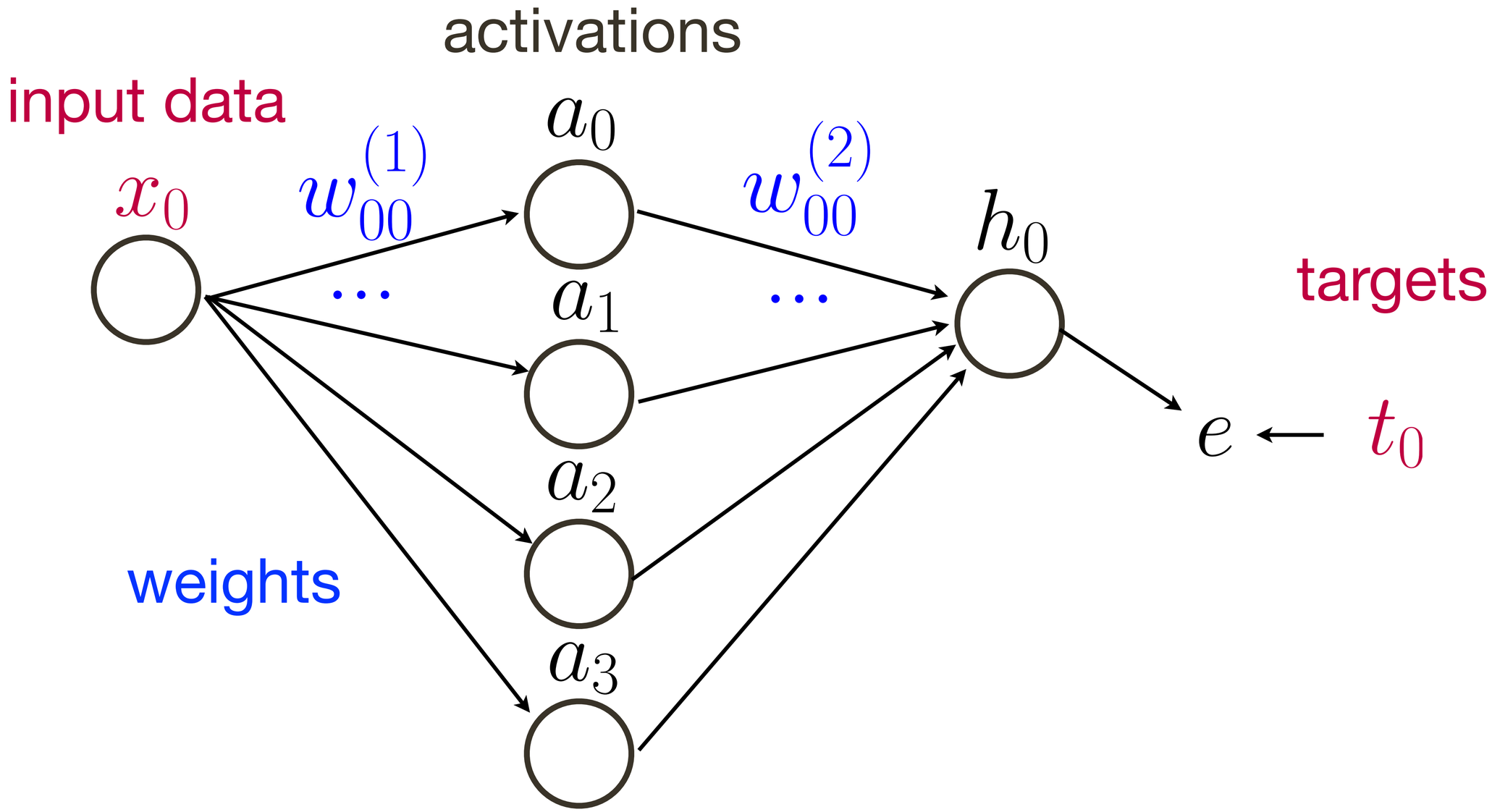
weights

$w_{23}^{(1)}$

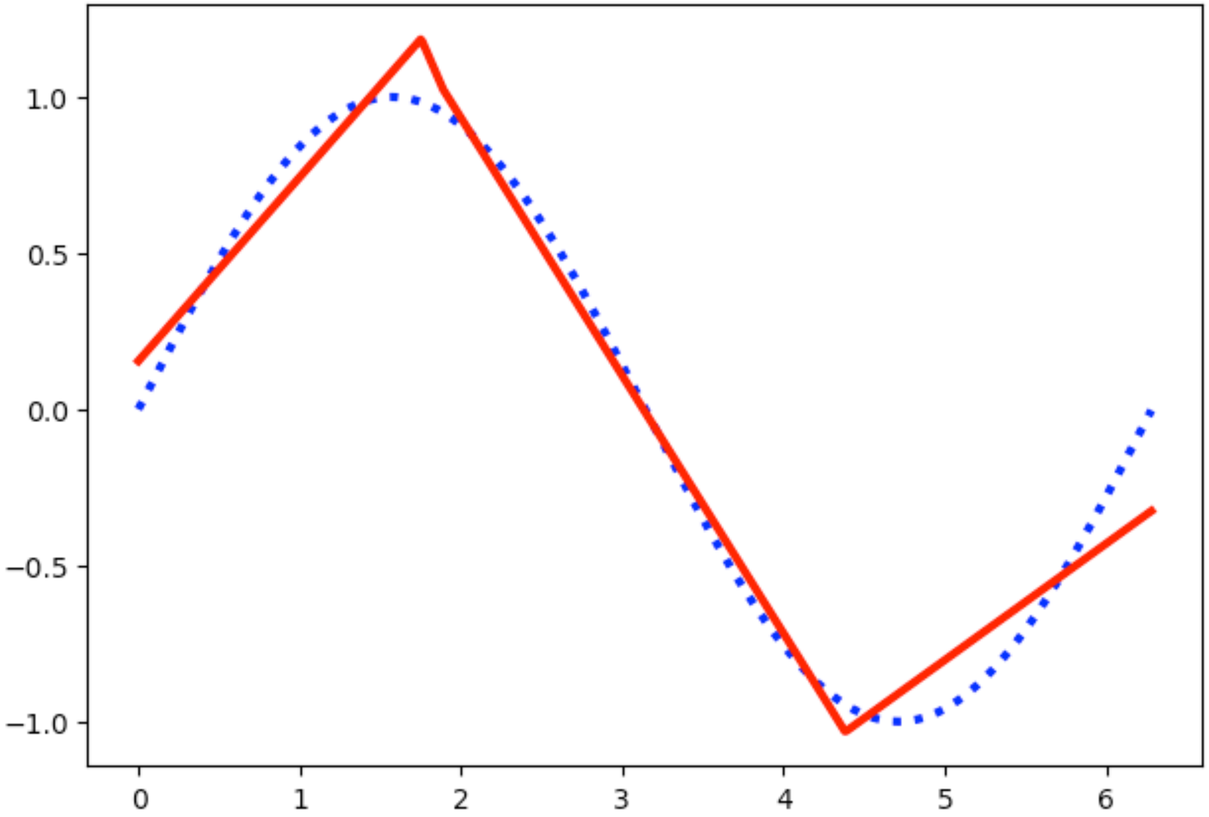
$w_{31}^{(2)}$

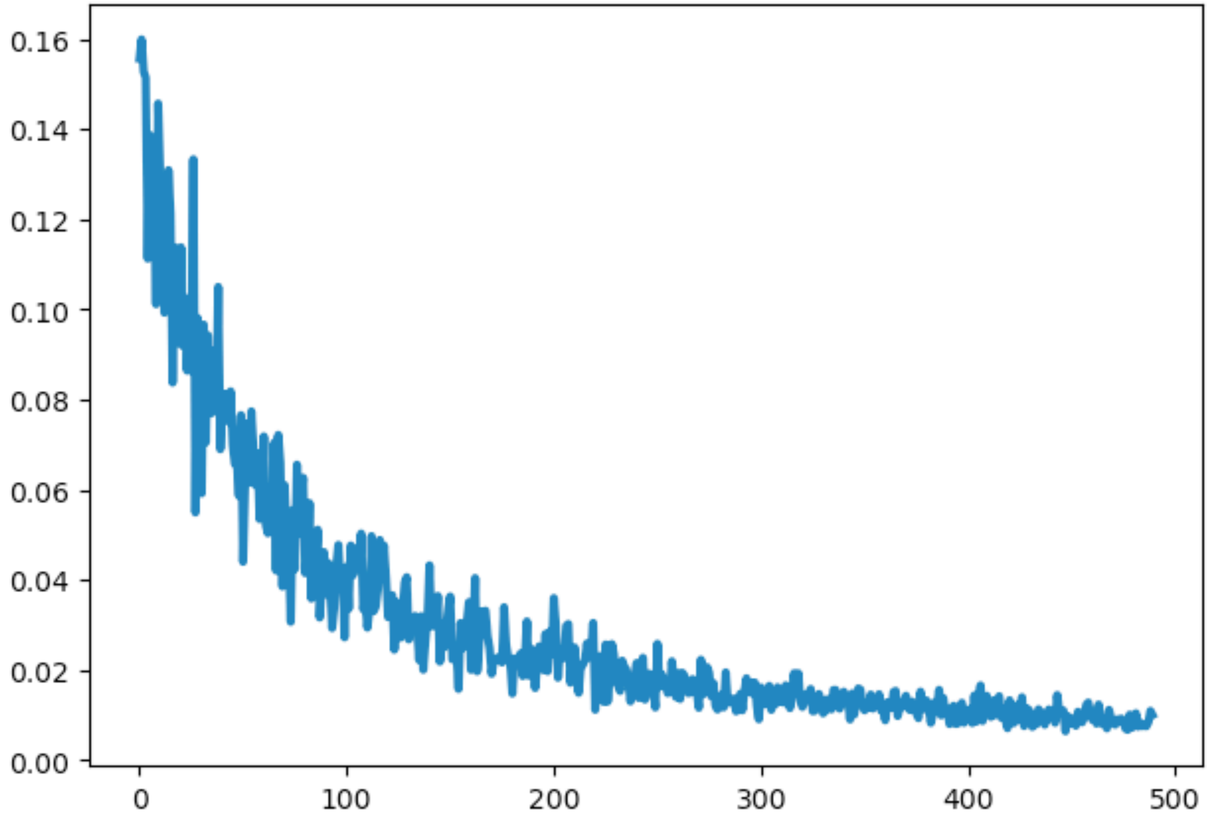
19.4

2-Layer **Neural** Network — n hidden, 1 input/output



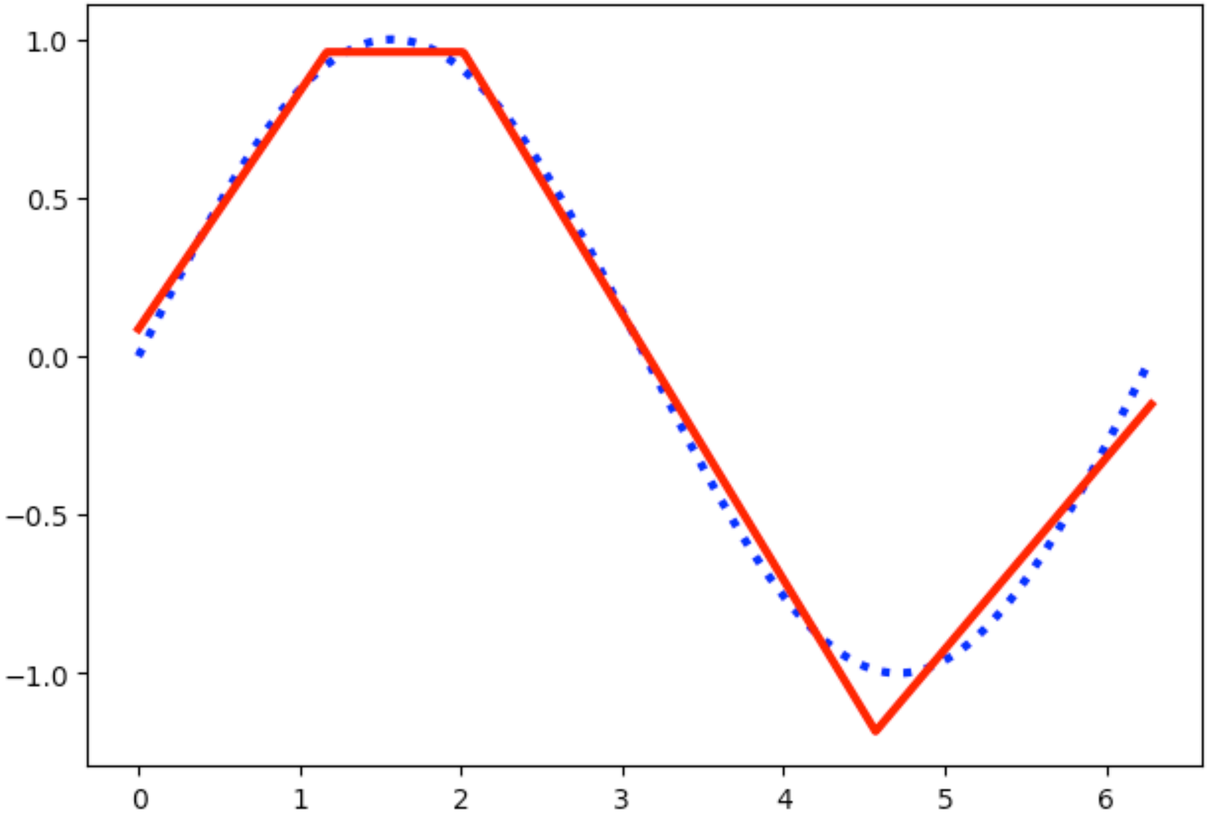
2-Layer **Neural** Network — n hidden, 1 input/output

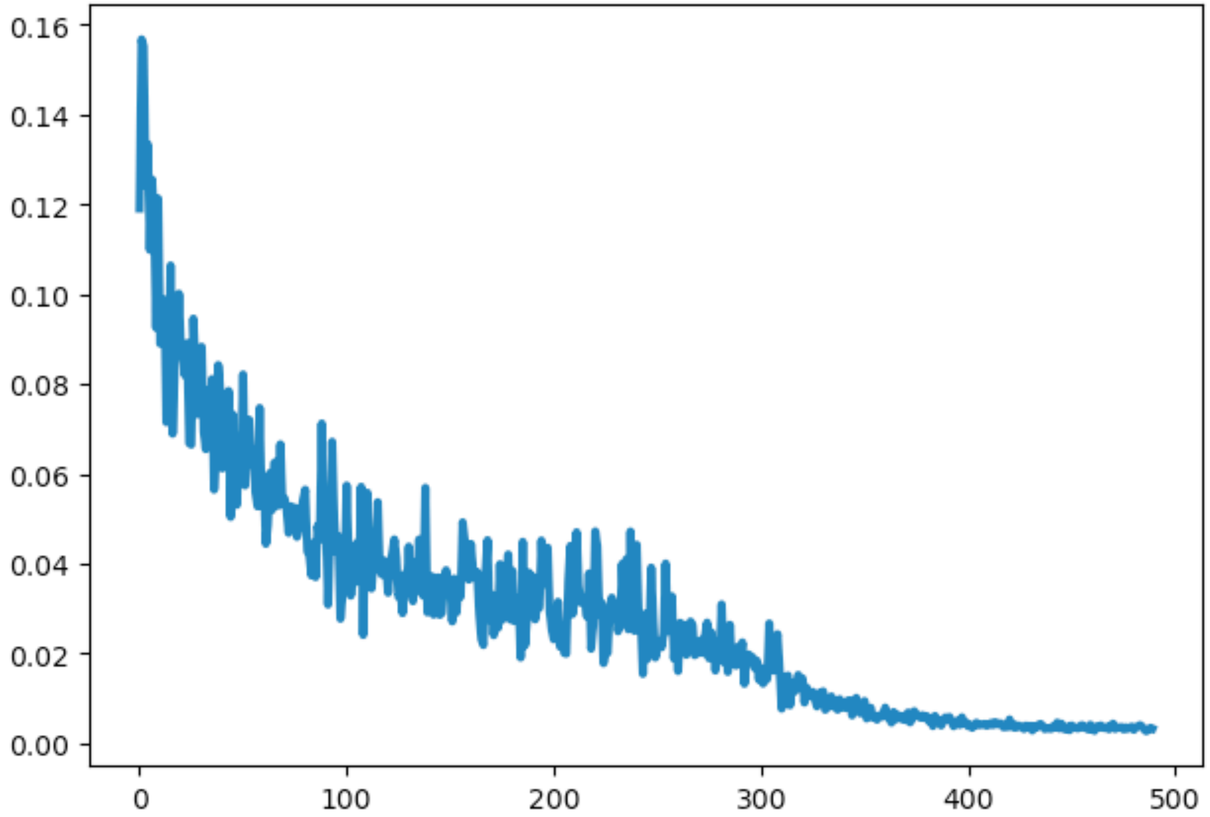




3 hidden units

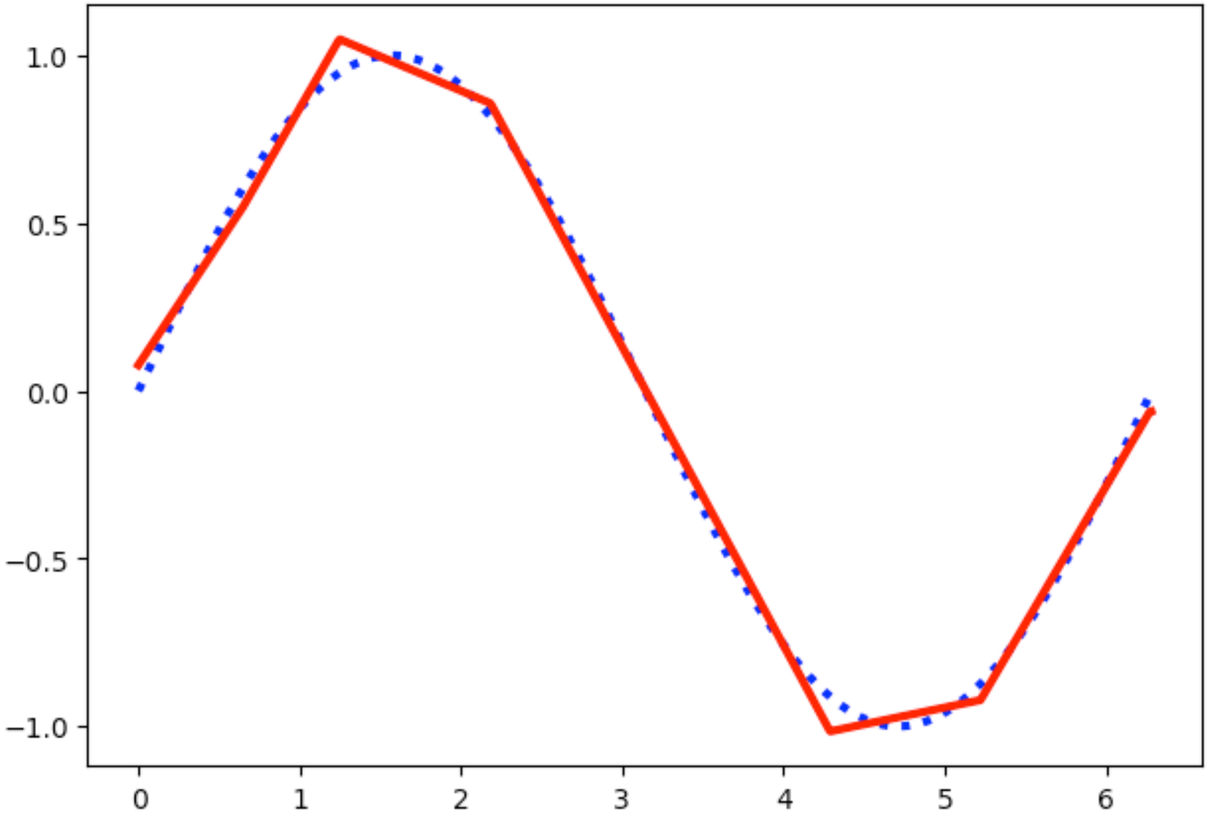
2-Layer **Neural** Network — n hidden, 1 input/output

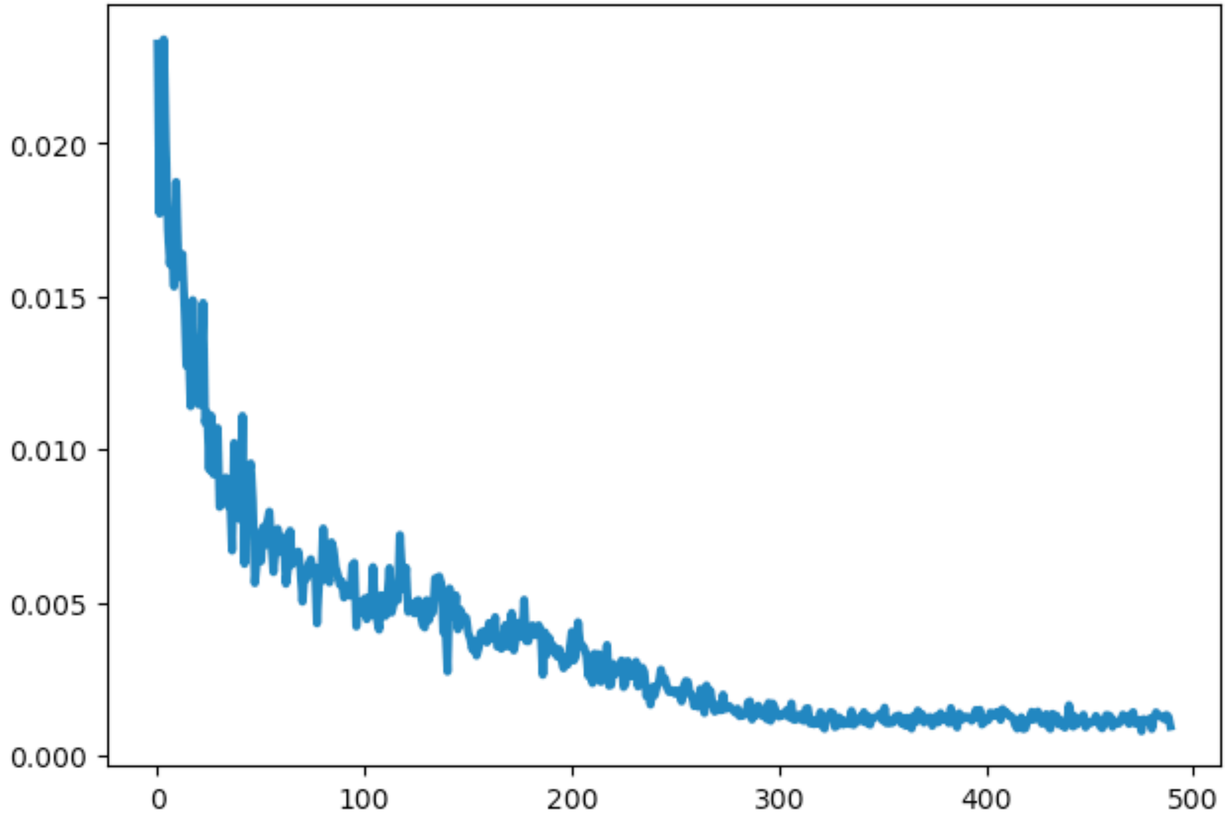




4 hidden units

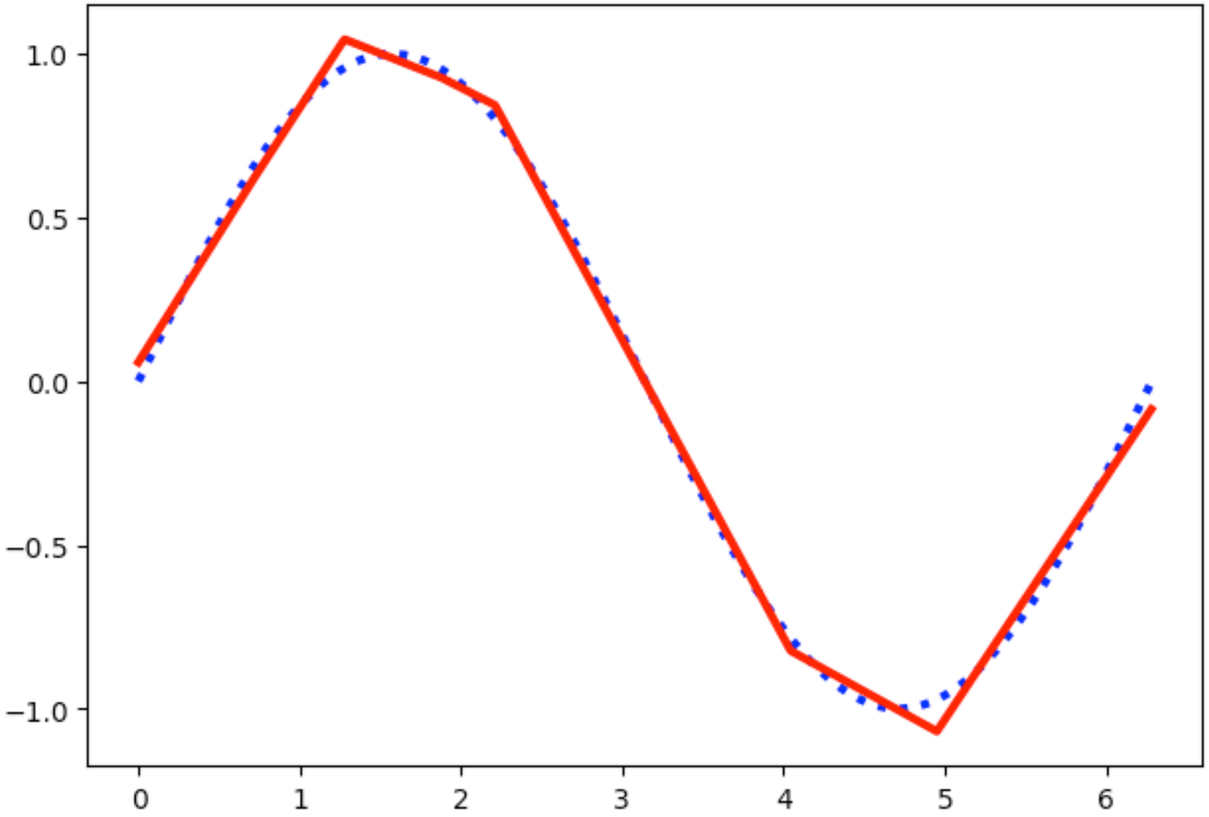
2-Layer **Neural** Network — n hidden, 1 input/output

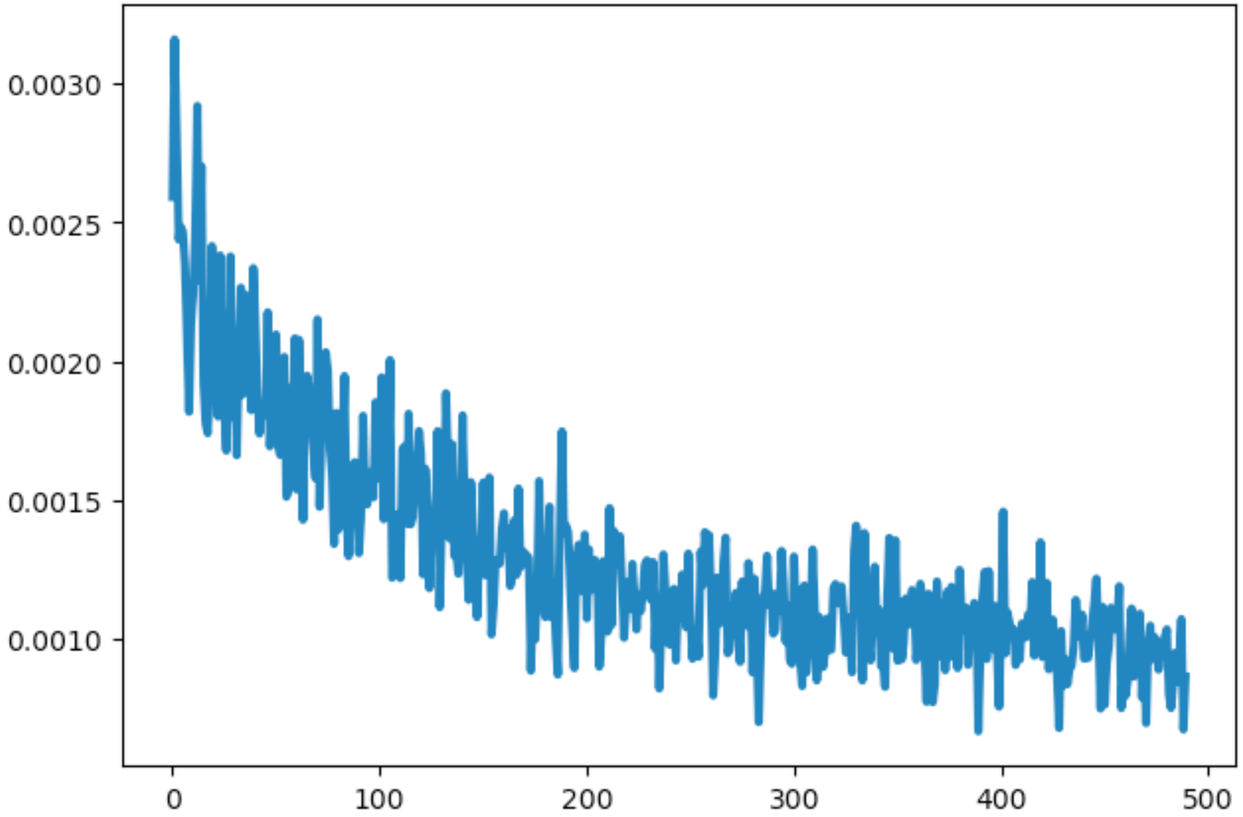




6 hidden units

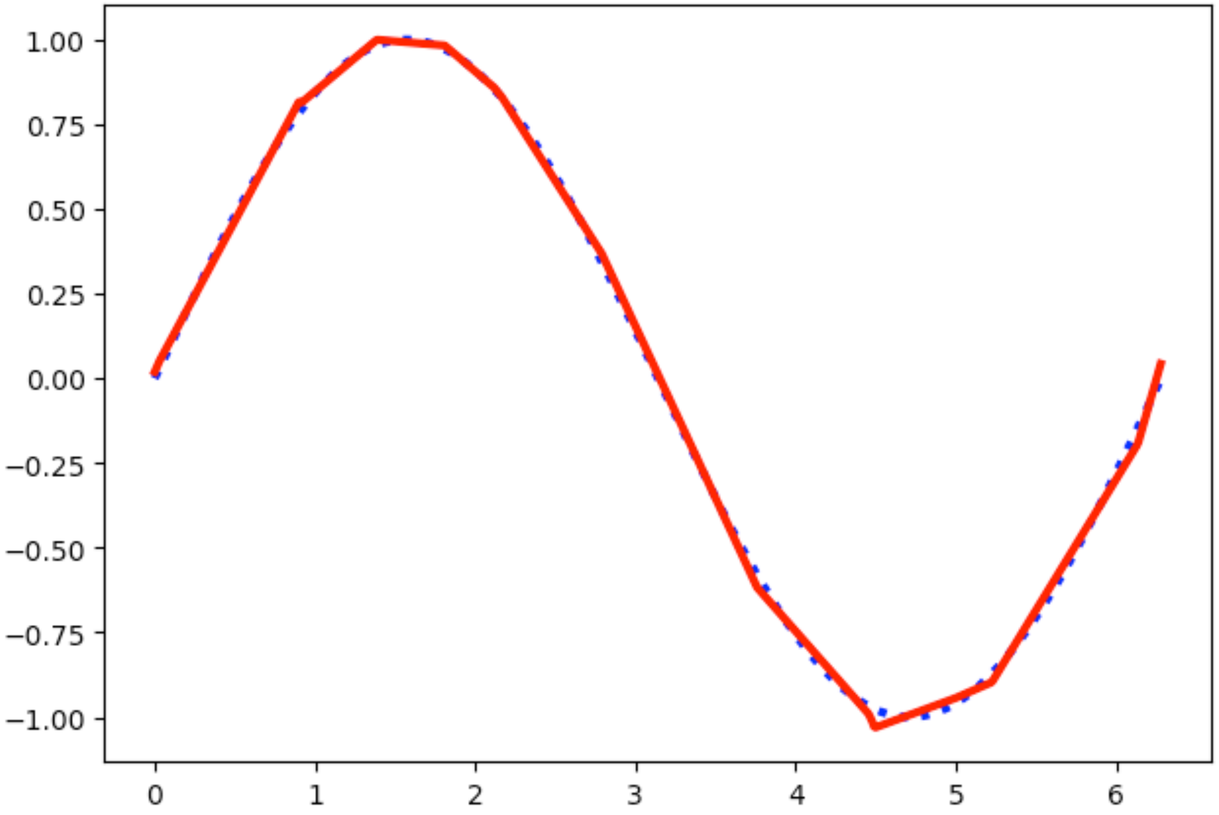
2-Layer **Neural** Network — n hidden, 1 input/output

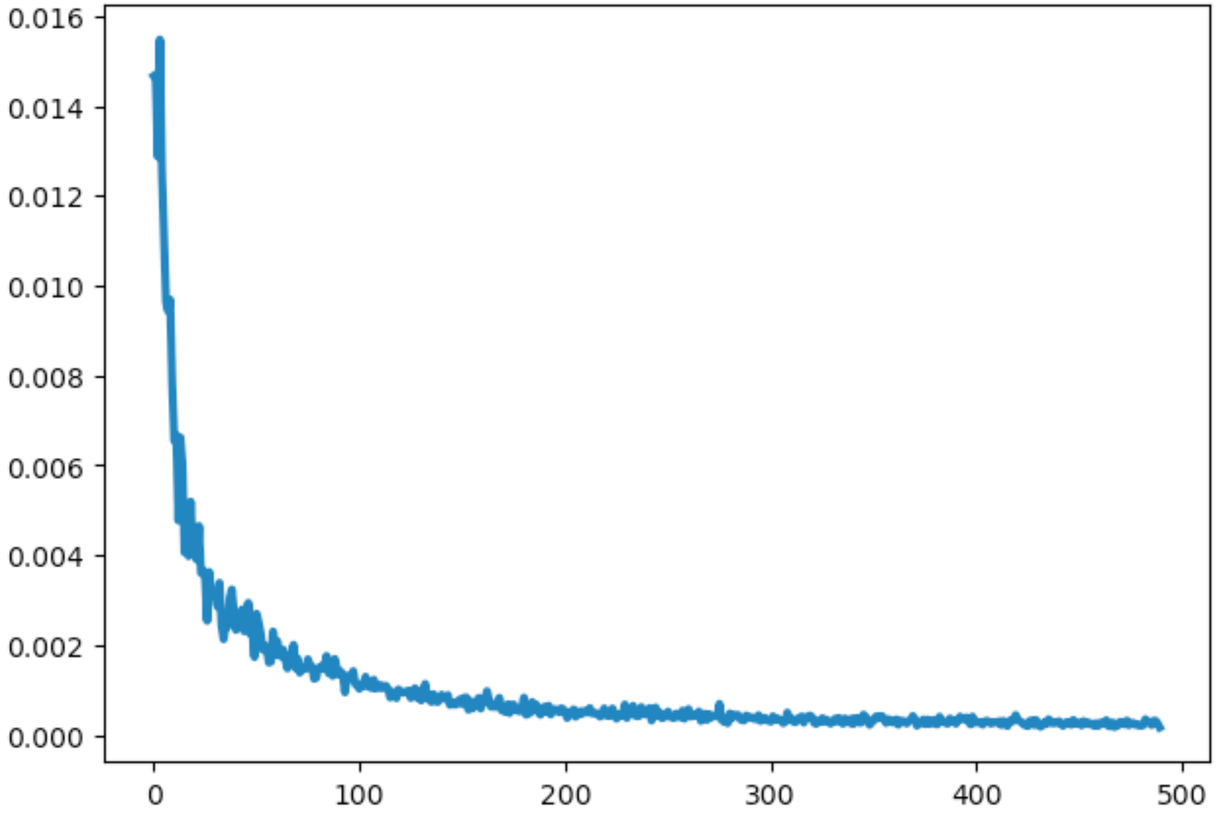




8 hidden units

2-Layer **Neural** Network — n hidden, 1 input/output





20 hidden units

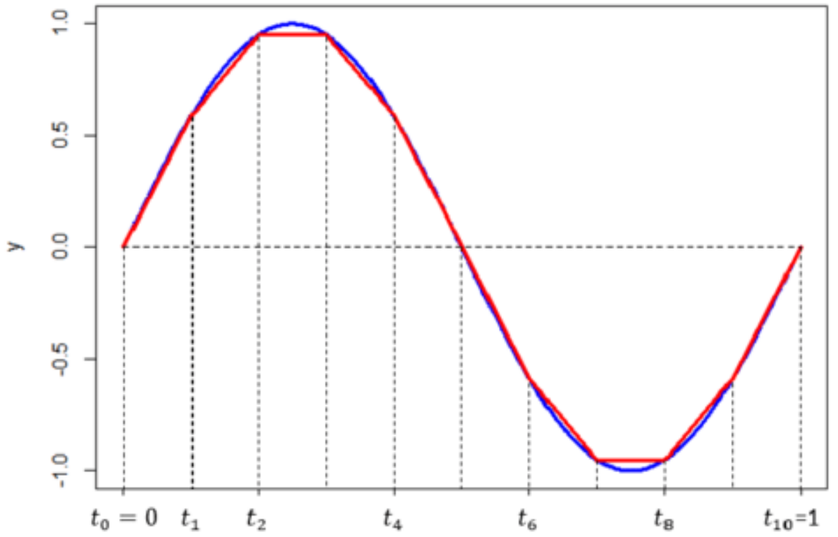
Neural Network as Universal Approximator

Non-linear activation is required to provably make the Neural Net a **universal function approximator**

Intuition: with ReLU activation, we effectively get a linear spline approximation to any function.

Optimization of neural net parameters = finding slops and transitions of linear pieces

The quality of approximation depends on the number of linear segments



Neural Network as Universal Approximator

Universal Approximation Theorem: Single hidden layer can approximate any continuous function with compact support to arbitrary accuracy, when the width goes to infinity.

[Hornik *et al*., 1989]

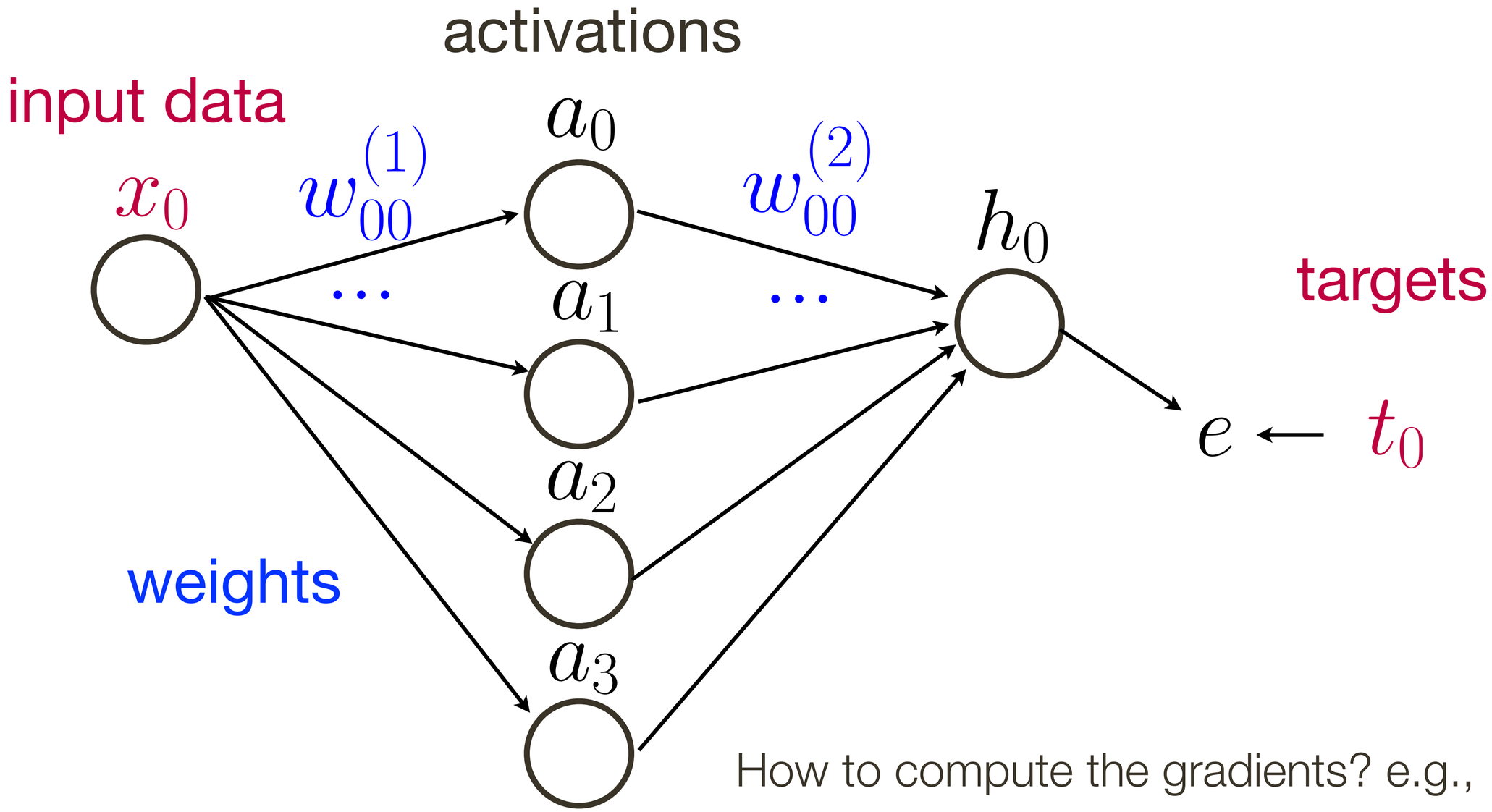
Universal Approximation Theorem (revised): A network of infinite depth with a hidden layer of size $d + 1$ neurons, where d is the dimension of the input space, can approximate any continuous function.

[Lu *et al*., NIPS 2017]

Universal Approximation Theorem (further revised): ResNet with a single hidden unit and infinite depth can approximate any continuous function.

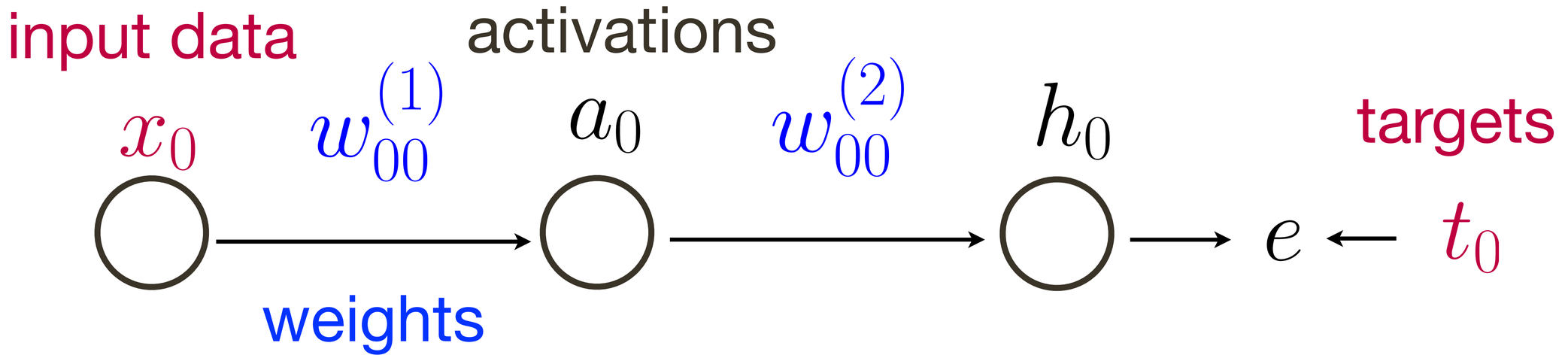
[Lin and Jegelka, NIPS 2018]

2-Layer **Neural** Network — n hidden, 1 input/output



How to compute the gradients? e.g., $\frac{\partial e}{\partial w_{00}^{(1)}}$

2-Layer **Neural** Network — 1 hidden, 1 input/output



2-Layer **Neural** Network — 1 hidden, 1 input/output

$$y = w_2(\max(0, w_1 x + b_1)) + b_2 \qquad L = (y - t)^2$$

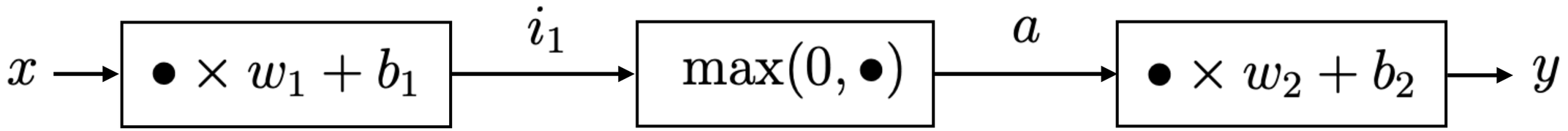
Optimise by **gradient descent**

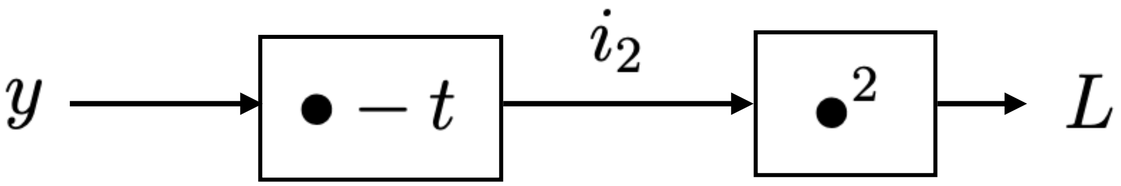
$$\begin{bmatrix} w_1 \\ b_1 \\ w_2 \\ b_2 \end{bmatrix} \rightarrow \begin{bmatrix} w_1 \\ b_1 \\ w_2 \\ b_2 \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial b_1} \\ \frac{\partial L}{\partial w_2} \\ \frac{\partial L}{\partial b_2} \end{bmatrix}$$

19.5 How to compute the gradients? e.g., $\frac{\partial L}{\partial w_1}$

2-Layer **Neural** Network — 1 hidden, 1 input/output

$$y = w_2(\max(0, w_1 x + b_1)) + b_2 \qquad L = (y - t)^2$$





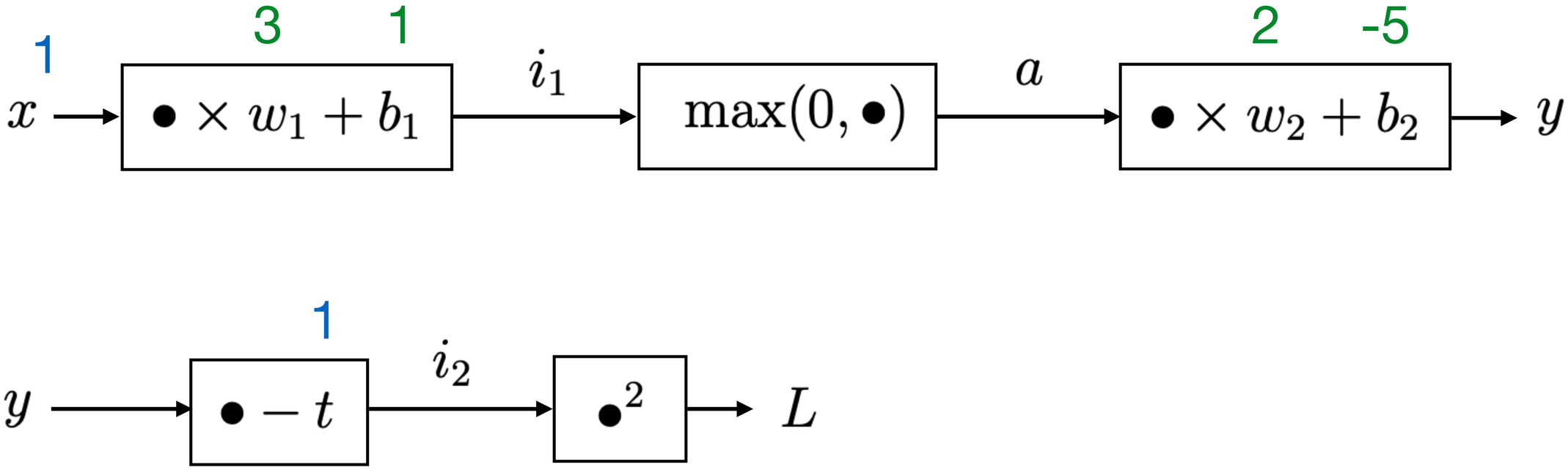
Alternative: build a **computational graph** to apply the **chain rule**

19.6

2-Layer **Neural** Network — 1 hidden, 1 input/output

Input + /target

Initial weights

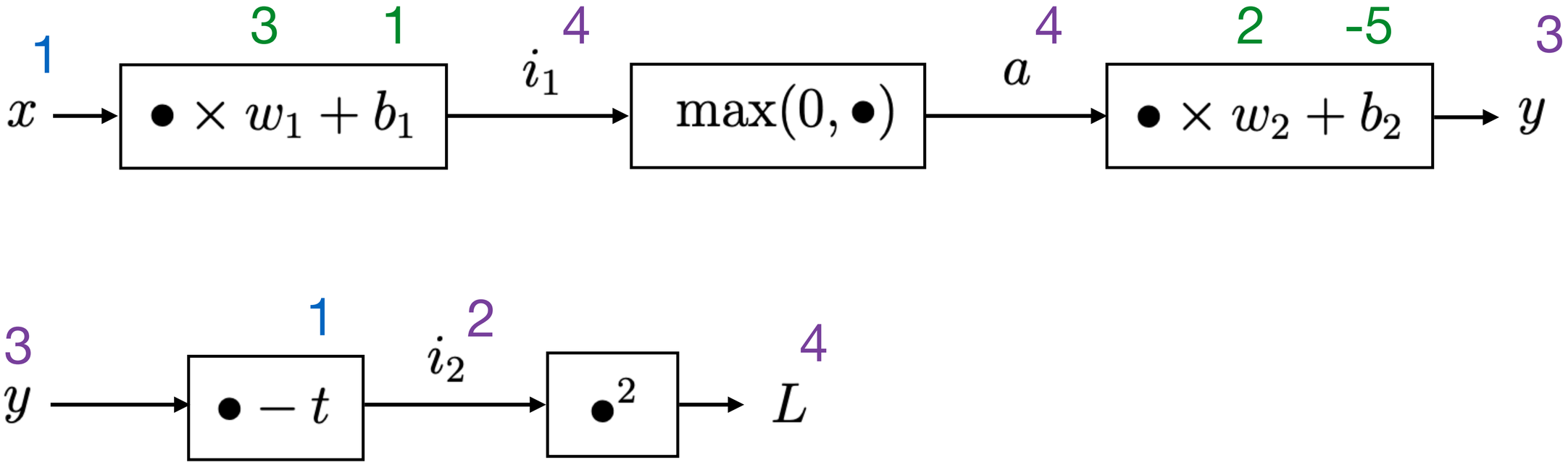


2-Layer **Neural** Network — 1 hidden, 1 input/output

Input + /target

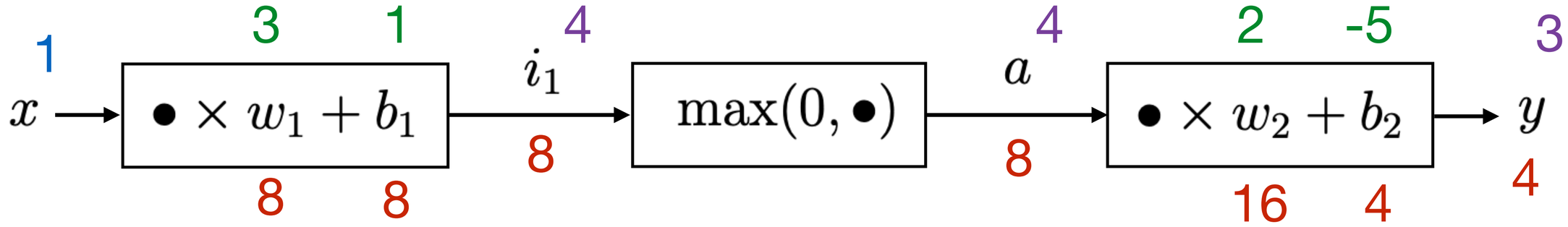
Initial weights

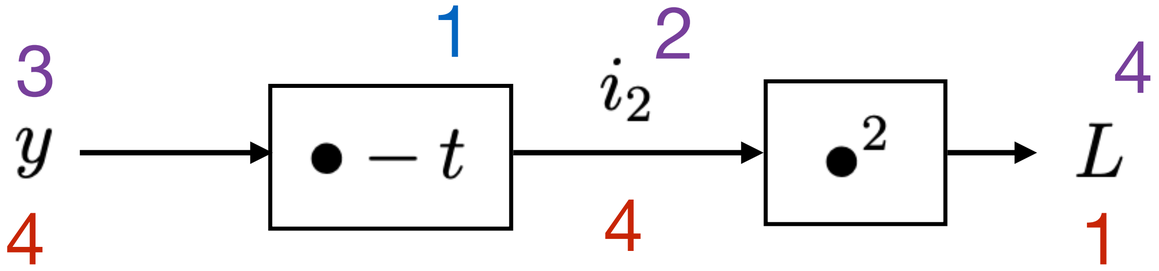
Forward pass



2-Layer **Neural** Network — 1 hidden, 1 input/output

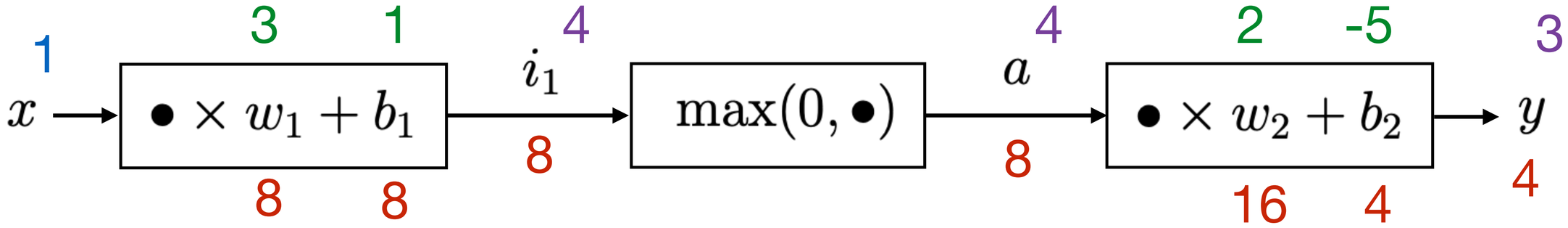
Input + /target | Initial weights | Forward pass | Backward pass $= \frac{\partial L}{\partial \bullet}$

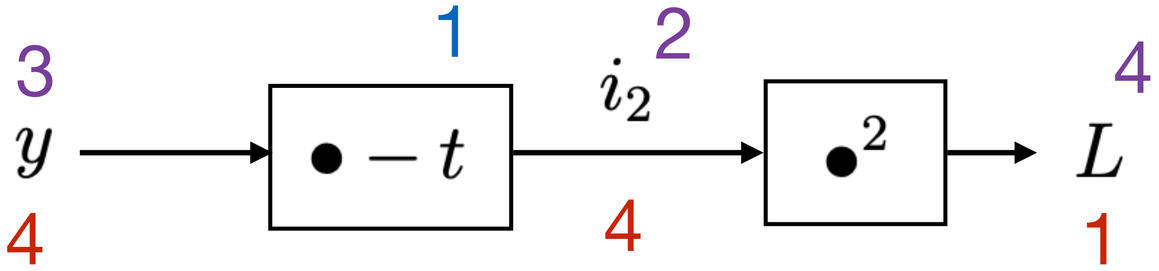




2-Layer **Neural** Network — 1 hidden, 1 input/output

Input + /target | Initial weights | Forward pass | Backward pass $= \frac{\partial L}{\partial \bullet}$

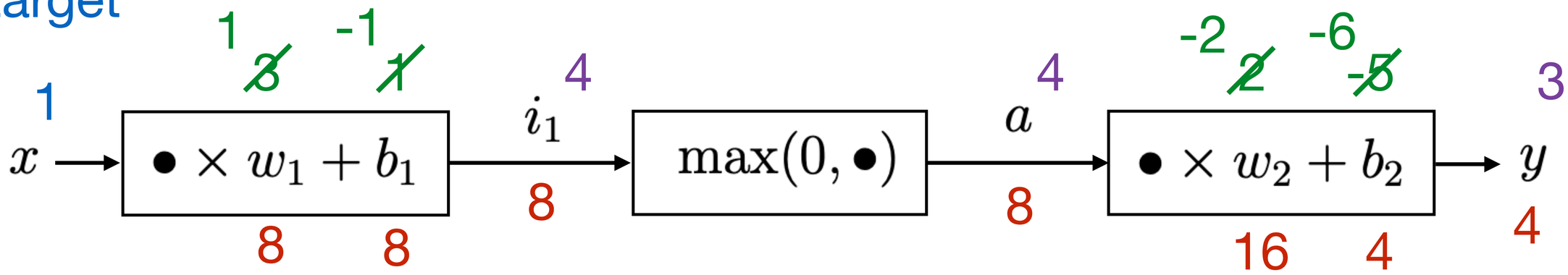


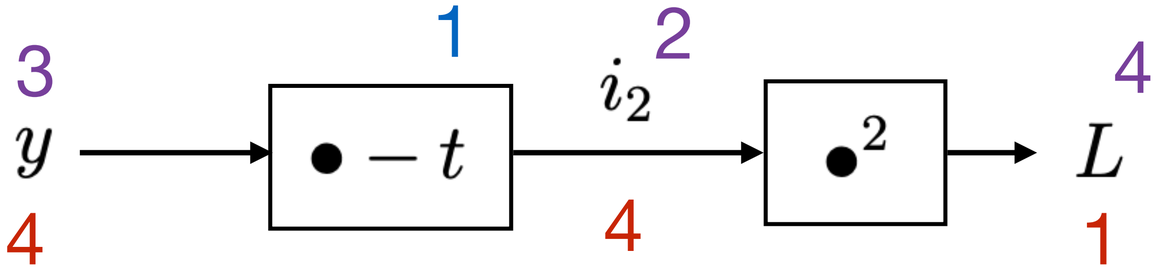


Gradient = $\begin{bmatrix} 8 \\ 8 \\ 16 \\ 4 \end{bmatrix}$

2-Layer **Neural** Network — 1 hidden, 1 input/output

Input + /target | Initial weights | Forward pass | Backward pass $= \frac{\partial L}{\partial \bullet}$





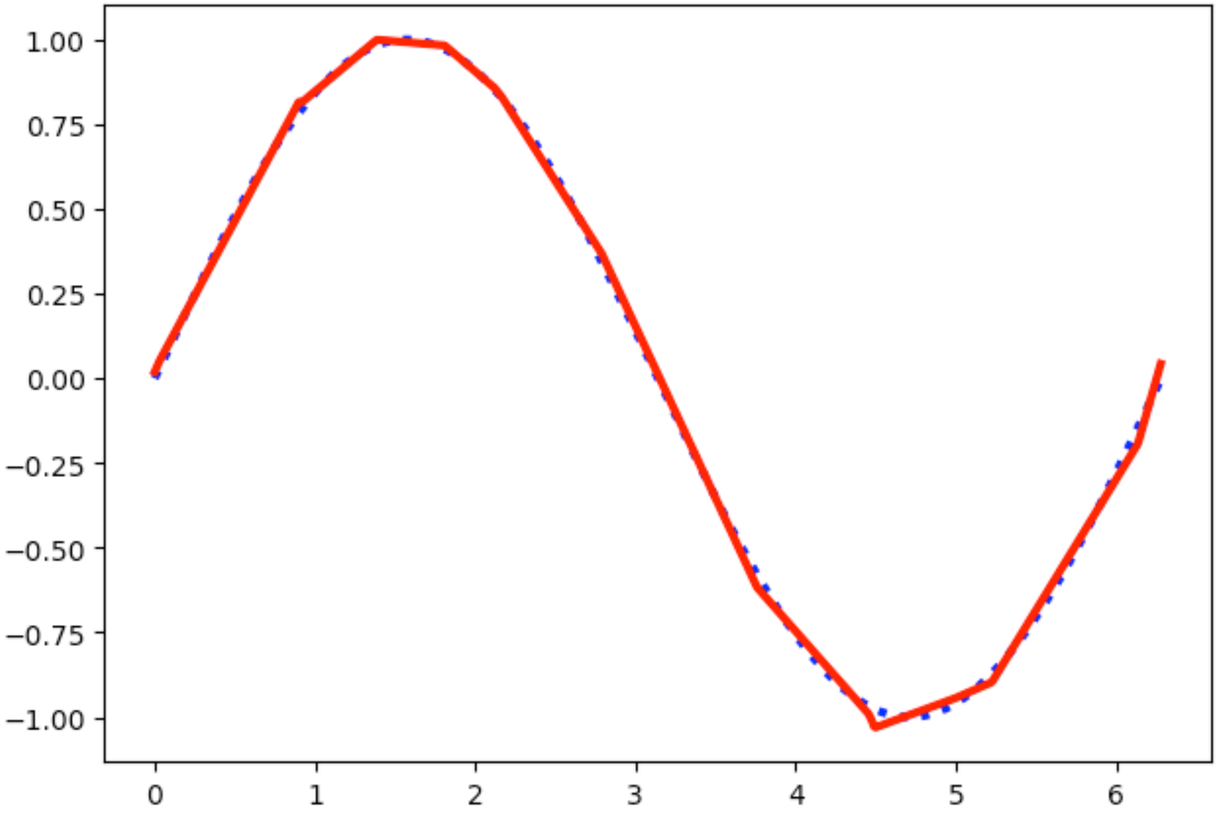
Gradient descent step

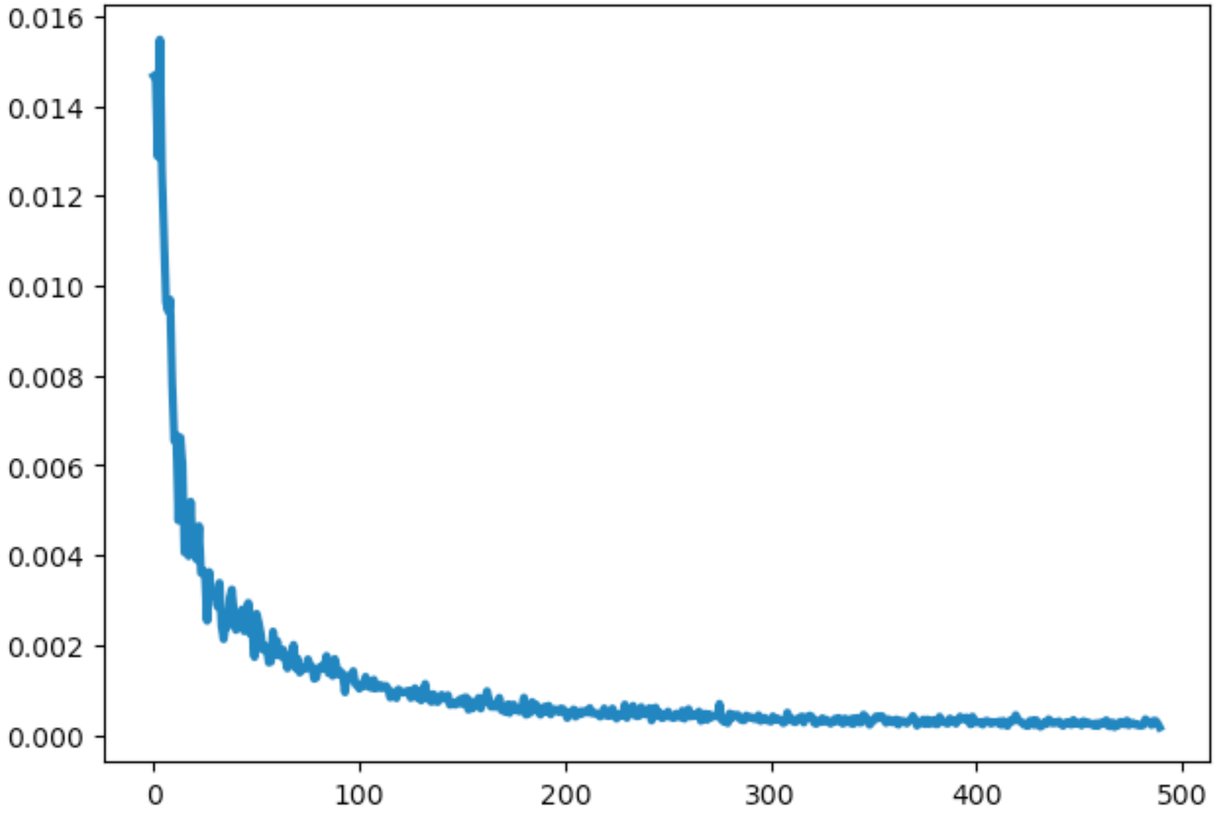
$$\begin{bmatrix} w_1 \\ b_1 \\ w_2 \\ b_2 \end{bmatrix} \rightarrow \begin{bmatrix} 3 \\ 1 \\ 2 \\ -5 \end{bmatrix} - 1/4\,\alpha \begin{bmatrix} 8 \\ 8 \\ 16 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -2 \\ -6 \end{bmatrix}$$

+ update weights

Repeat: +Input/target, Forward, Backward, Update until convergence!

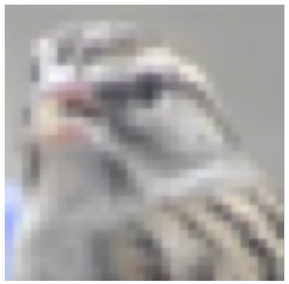
2-Layer **Neural** Network — n hidden, 1 input/output





20 hidden units

2-Layer **Neural** Network



activations

input data

x_0

x_1

x_2

$w_{00}^{(1)}$

$\cdots$

a_0

a_1

a_2

a_3

$w_{00}^{(2)}$

$\cdots$

“bird”

h_0

h_1

“plane”

$e \leftarrow \begin{bmatrix} t_0 \\ t_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

targets

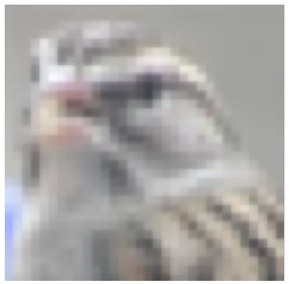
e.g.,

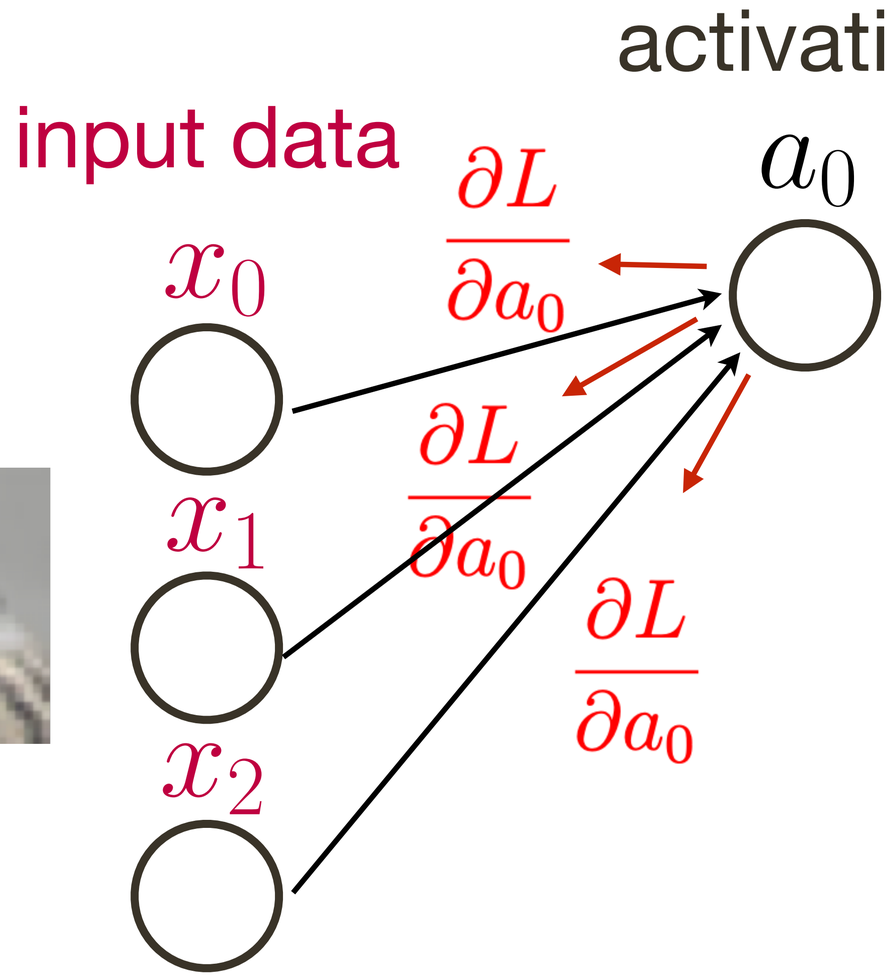
weights

$w_{23}^{(1)}$

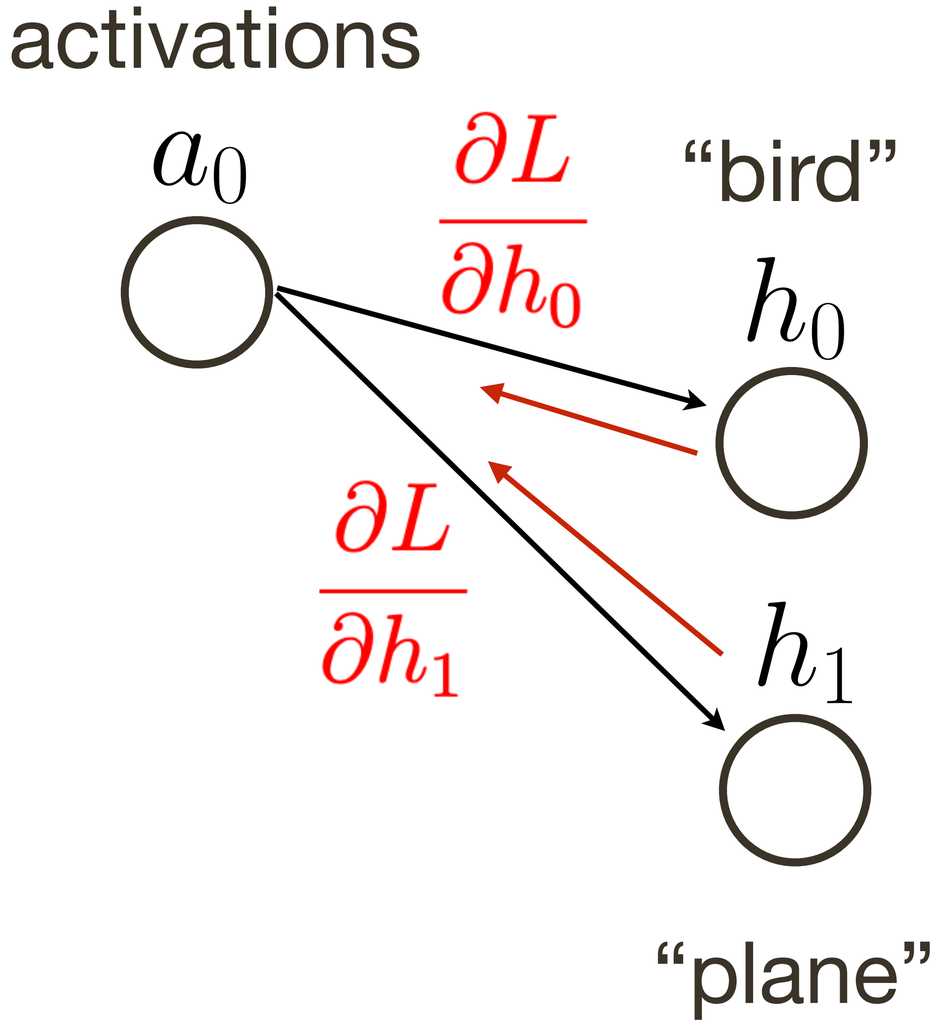
$w_{31}^{(2)}$

2-Layer **Neural** Network — multiple inputs





2-Layer **Neural** Network — multiple outputs



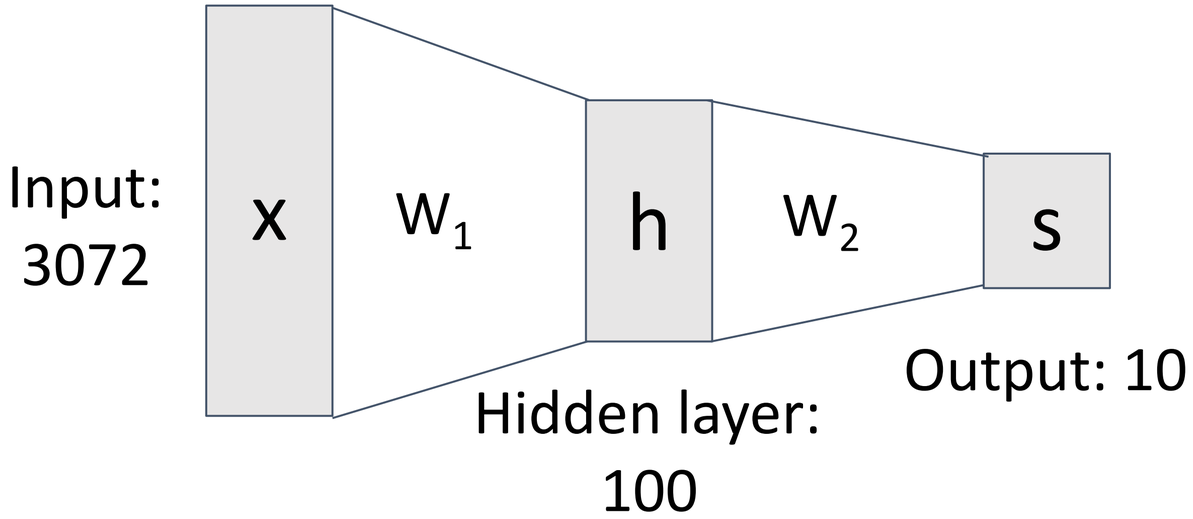
19.7

Neural Networks

Linear classifier: One template per class

(**Before**) Linear score function:

(**Now**) 2-layer Neural Network



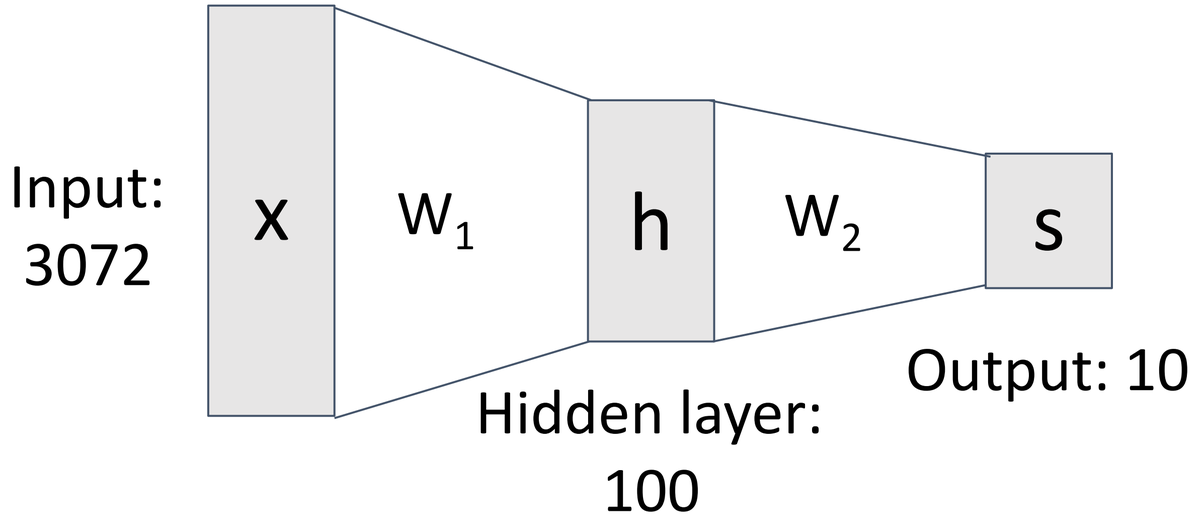
$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

Neural Networks

Neural net: first layer is bank of templates;
Second layer recombines templates

(**Before**) Linear score function:

(**Now**) 2-layer Neural Network



$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

Menu for Today

Topics:

- **Neural Networks** introduction
- **Activation functions** softmax, relu
- **2-layer** fully connected net
- **Backprop** intro

Readings:

- **Today's** Lecture: Szeliski 5.1.3, 5.3-5.4, Justin Johnson Michigan EECS 498/598

Reminders:

- **Assignment 5**: due today
- **Assignment 6**: Deep Learning is now available