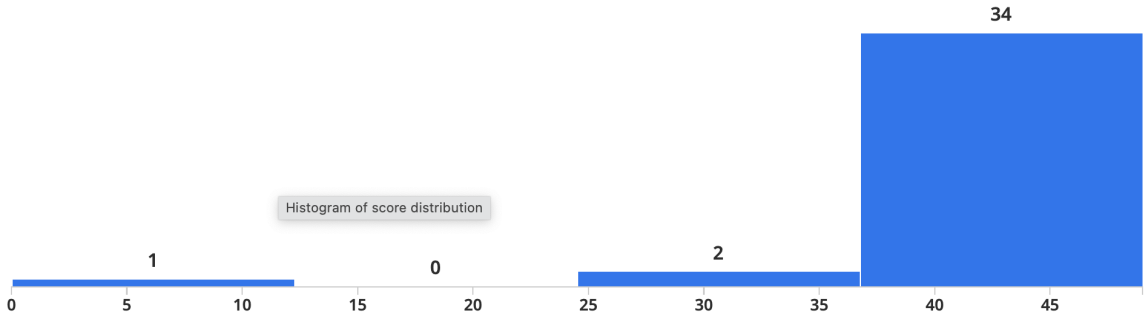
HW2 GRADES POSTED

Review Grades for HW2-Written

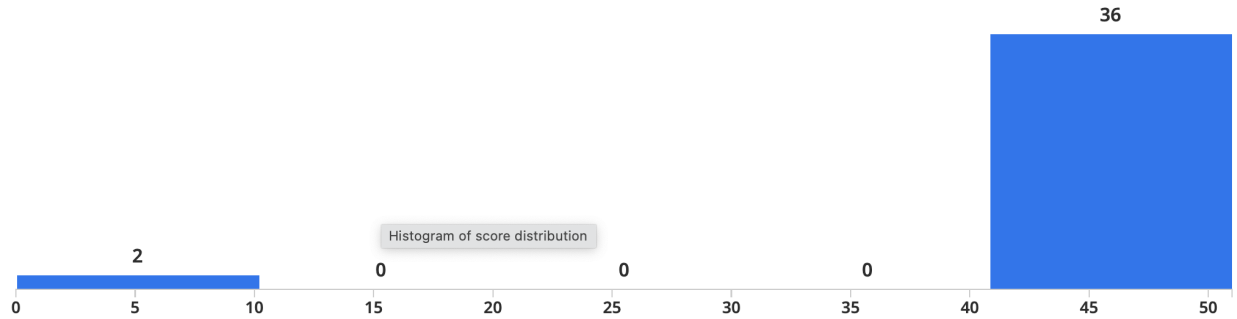
● Regrade Requests Open ● Grades Published



Minimum	Median	Maximum	Mean	Std Dev
0.0	45.0	49.0	42.58	8.71

Review Grades for HW2-Code

● Grades Published



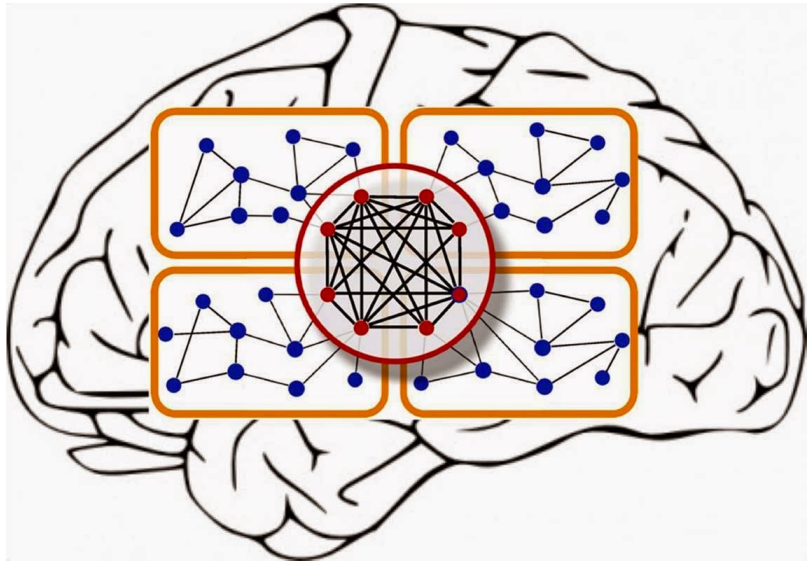
Minimum	Median	Maximum	Mean	Std Dev
0.0	50.0	51.0	47.0	10.85

NEURAL NETWORKS

CS 334: Machine Learning

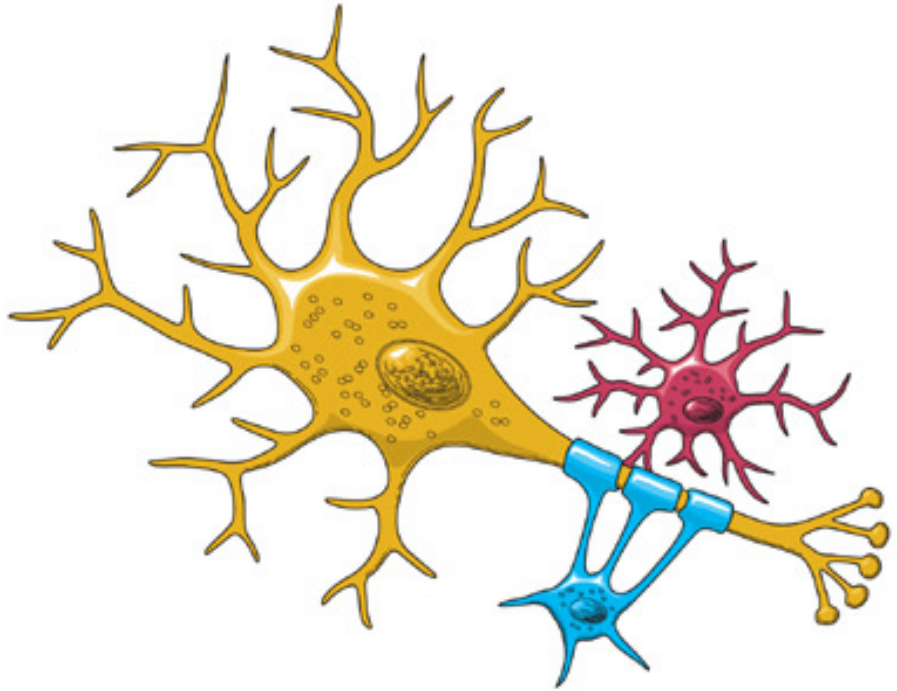
MOTIVATION: HUMAN BRAIN

- Contains 10^{11} neurons, each with up to 10^5 connections
- Learning in the brain happens by neurons becoming connected to other neurons, and the strengths of connections adapting over time



MOTIVATION: NEURON

- Electrically excitable cell that processes and transmits information
- Information comes in on the dendrites (input)
- If neuron excited/activated, send a spike of electrical activity to axon (output)

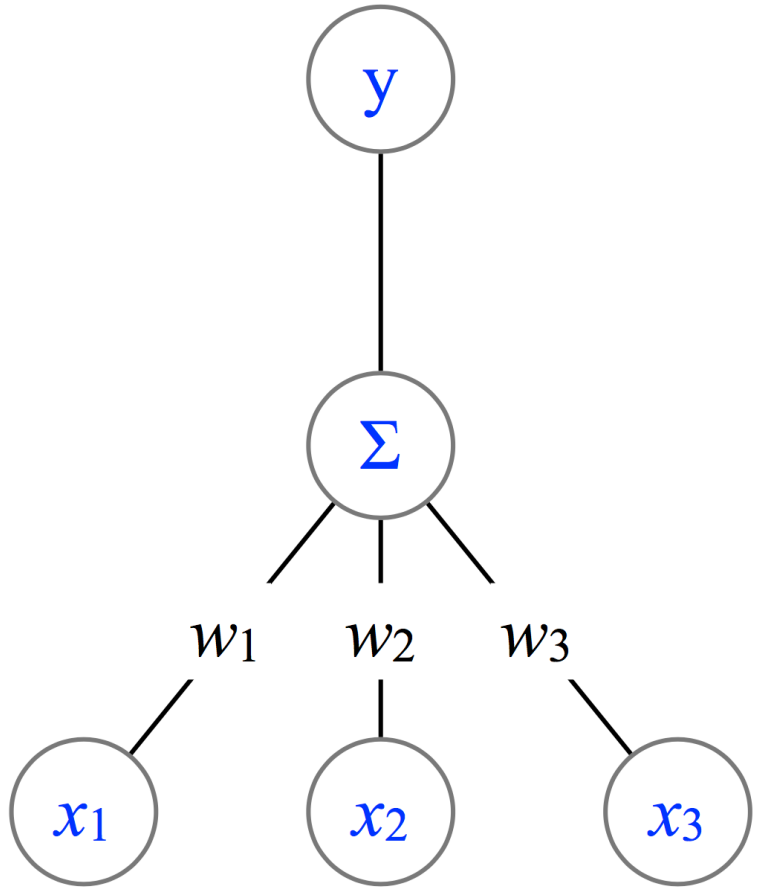


REVIEW: PERCEPTRON

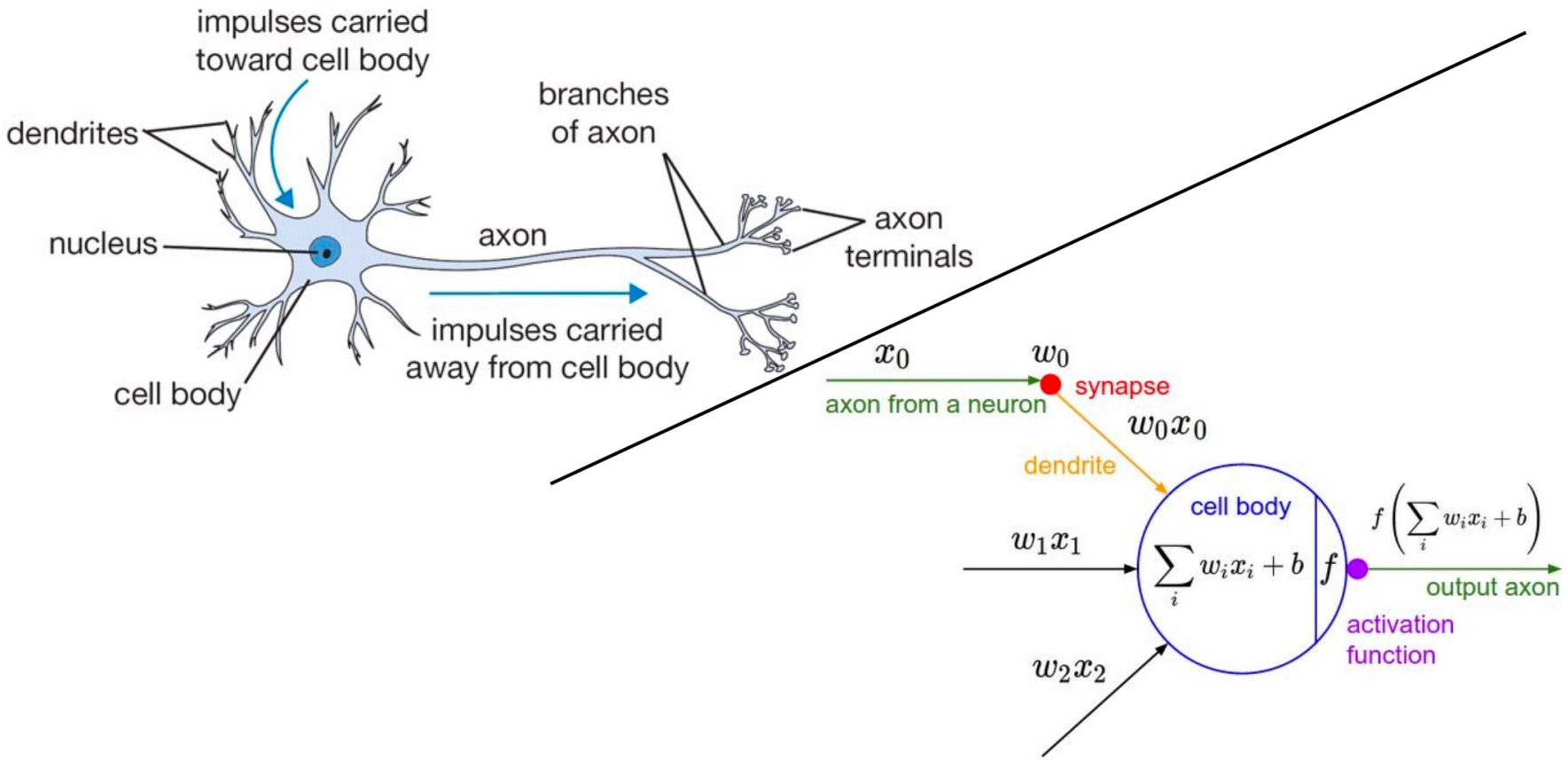
- Uses hyperplane classifier to map input to binary output
- Compute linear combination of the inputs and threshold it

$$f_{\mathbf{w}}(\mathbf{x}) = \text{sign}(\mathbf{x} \cdot \mathbf{w})$$

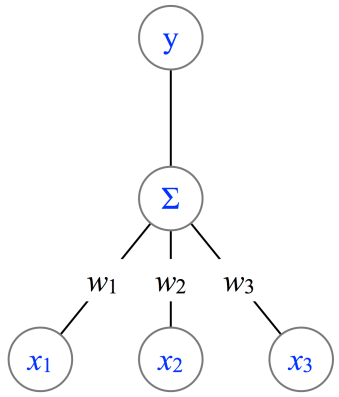
$$= \begin{cases} +1 & \text{if } \mathbf{x} \cdot \mathbf{w} > 0 \\ -1 & \text{otherwise} \end{cases}$$

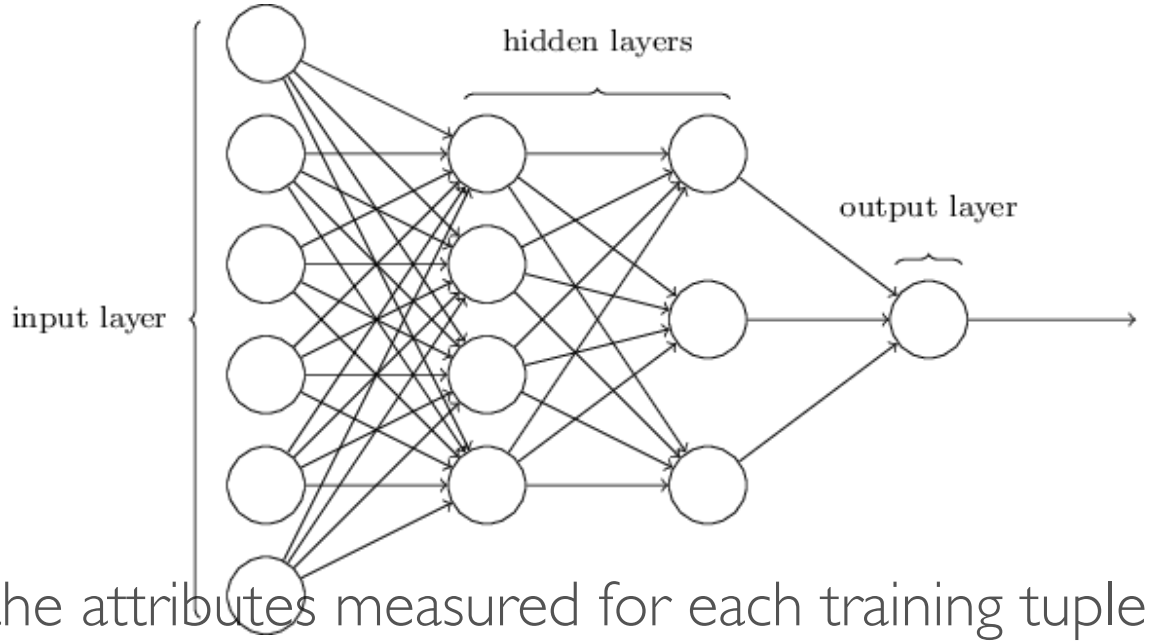


NEURON —> PERCEPTRON



FROM NEURON TO NEURAL NETWORK





The **input layer** correspond to the attributes measured for each training tuple

They are then weighted and fed simultaneously to **hidden layers**

The weighted outputs of the last hidden layer are input to **output layer**, which emits the network's prediction

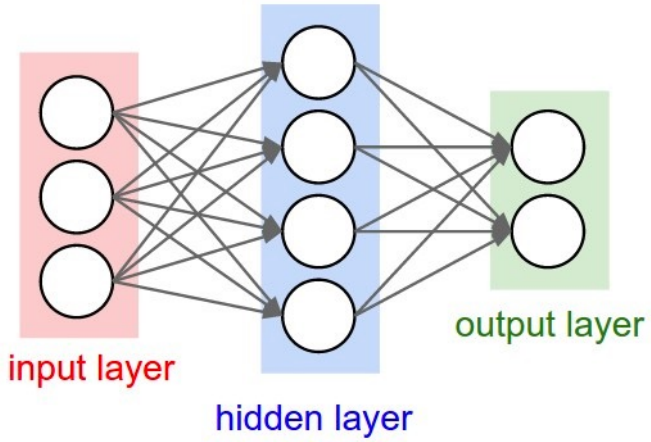
Perform **nonlinear regression**: Given enough hidden units and training samples, can closely approximate any function

NEURAL NETWORKS

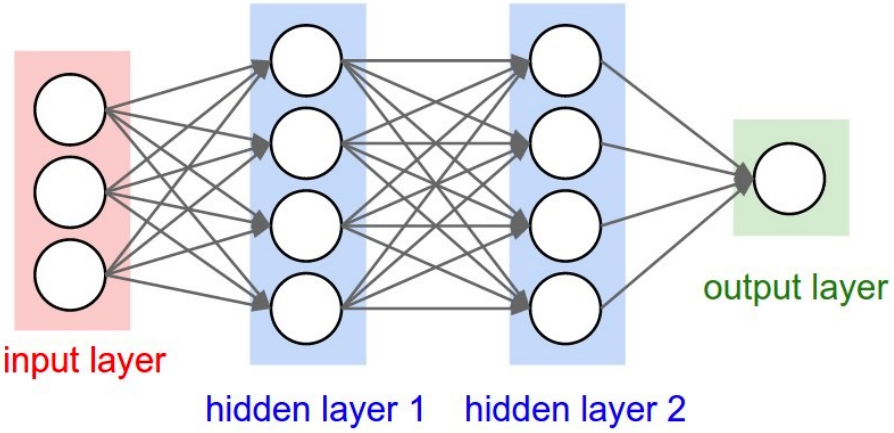
- Collection of neurons that are connected in an acyclic graph
  - Outputs of some neurons become inputs to other neurons
  - Compute non-linear decision boundaries
- Often organized into distinct layers of neurons
- Artificial Neural Networks (ANN) or Multi-Layer Perceptrons (MLP)

NEURAL NETWORKS: ARCHITECTURES

2-layer neural network

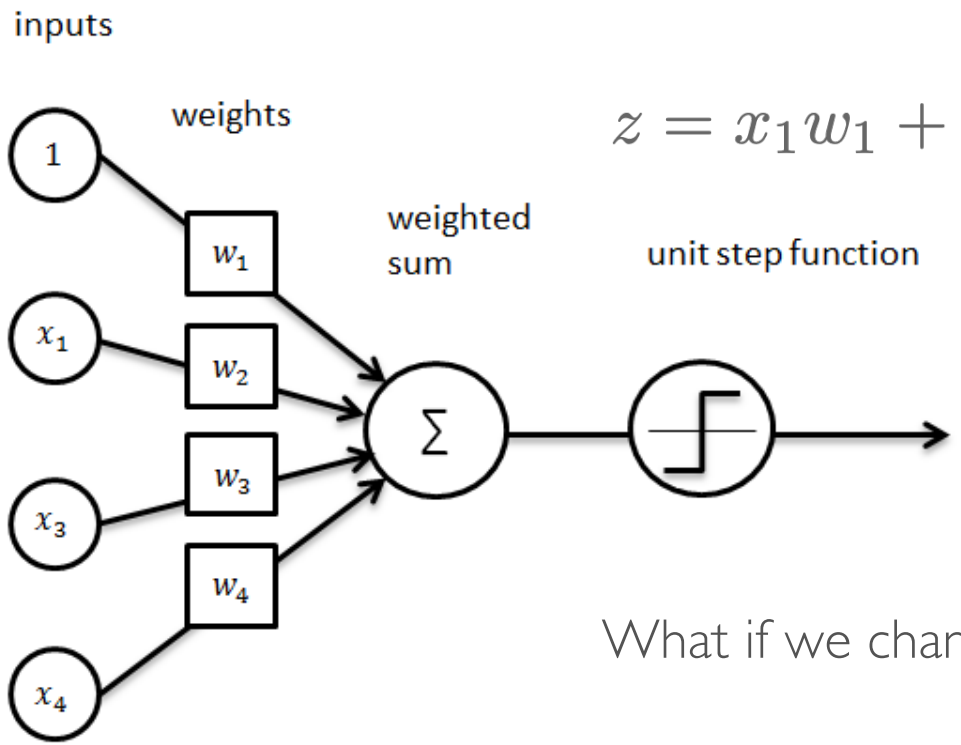


3-layer neural network



Naming convention doesn't count input layer

PERCEPTRON: REVISITED

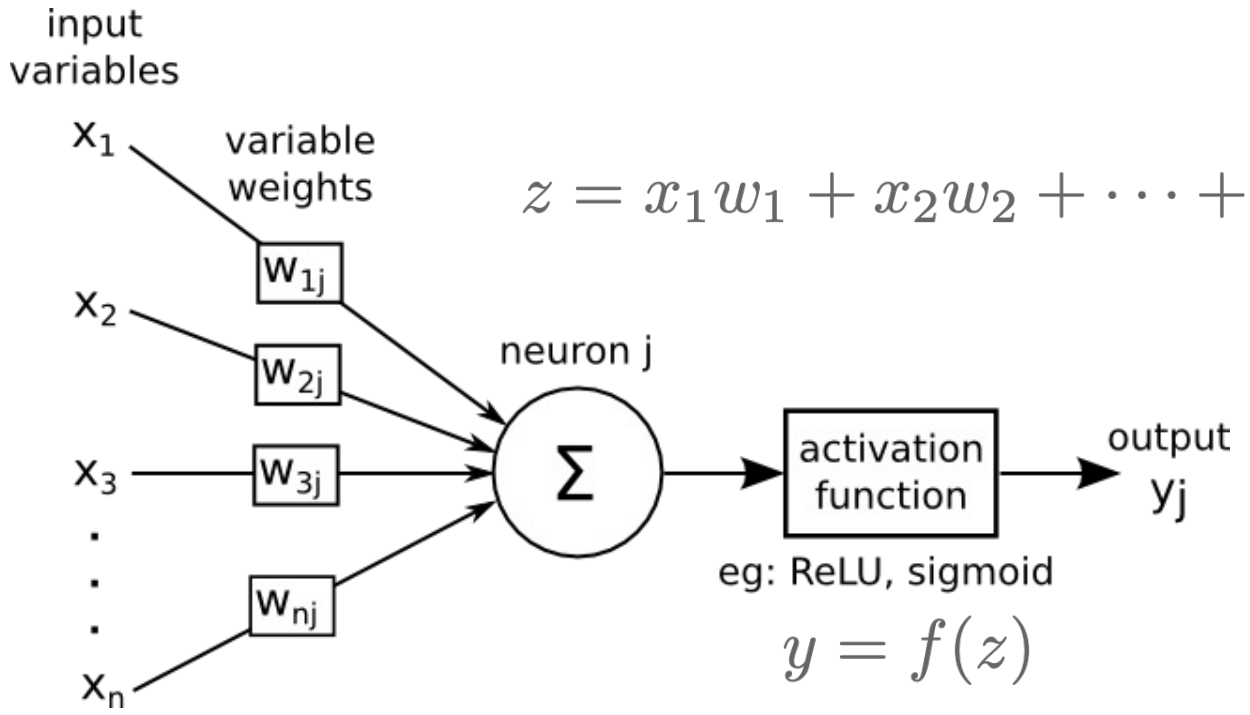


$$z = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$$

What if we change the activation function?

http://ataspinar.com/2016/12/22/the-perceptron/

GENERALIZED PERCEPTRON



$$z = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$$

$$y = f(z)$$

http://dataminingtheworld.blogspot.com/

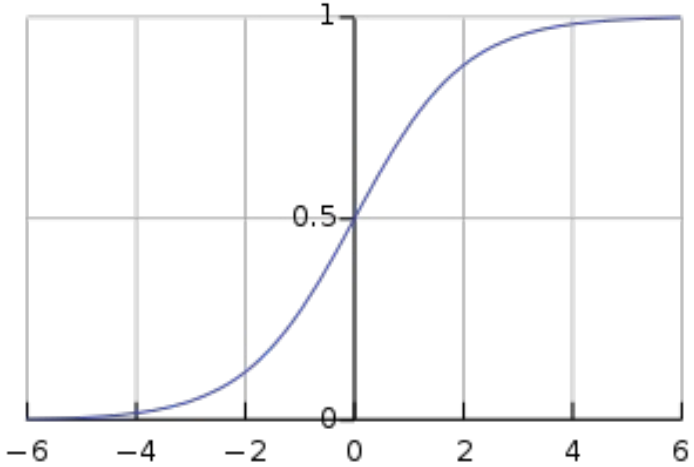
NEURON: SIGMOID UNIT

- Activation function is sigmoid function

$$\sigma(\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{x})}$$

- Nice property of sigmoid

$$\frac{\partial \sigma(\mathbf{x})}{\partial x} = \sigma(\mathbf{x})(1 - \sigma(\mathbf{x}))$$



What does this look like?!

RELATION TO LOGISTIC REGRESSION

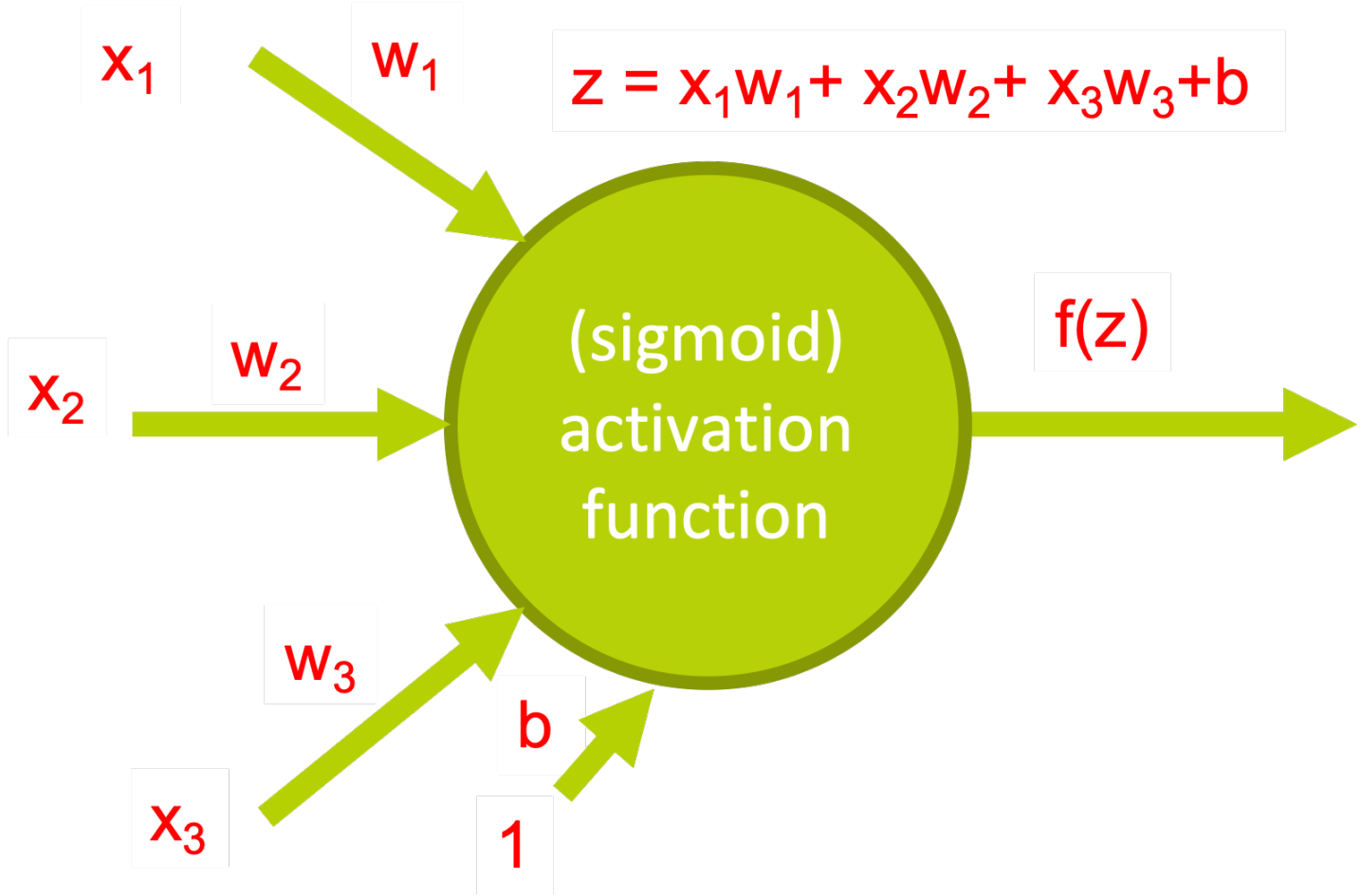
$$z = b + \sum_{i=1}^{m} x_i w_i = x_1 w_1 + x_2 w_2 + \cdots + x_m w_m + b$$

Then a neuron is simply a ”unit” of logistic regression!

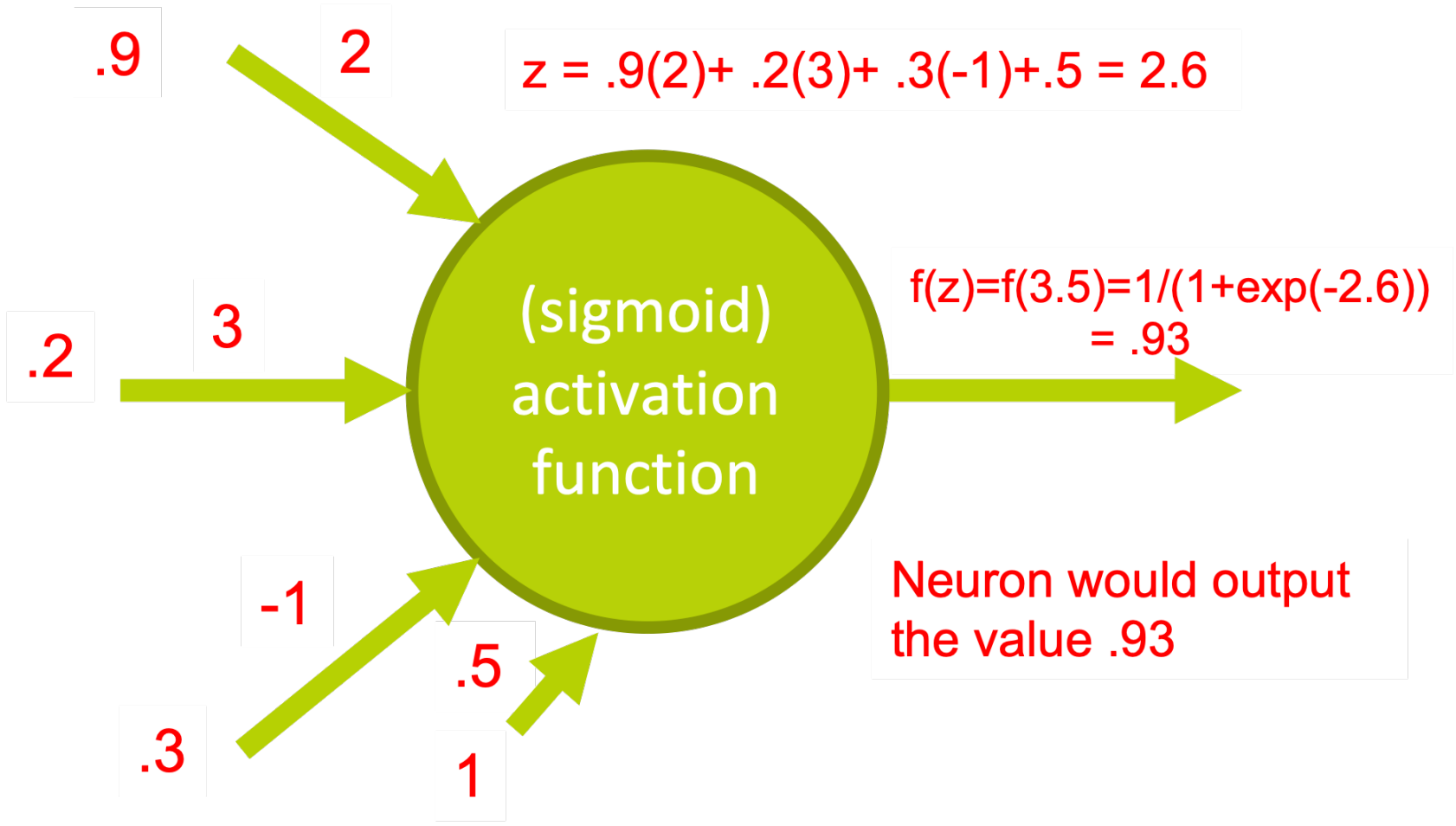
weights ⇔ coefficients inputs ⇔ variables

bias term ⇔ constant term

SIGMOID UNIT COMPUTATION



EXAMPLE: COMPUTATION

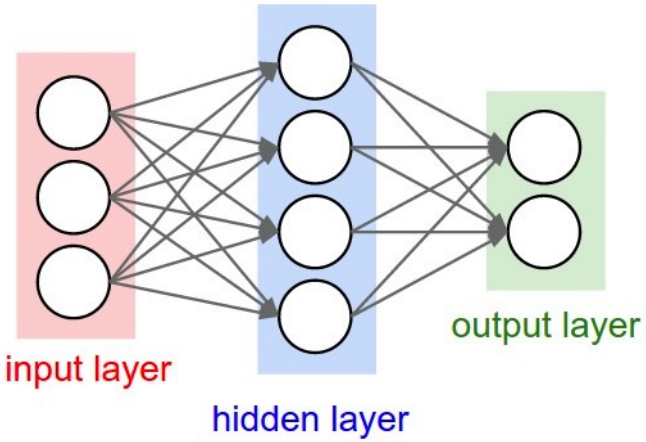


SIGMOID UNITS VS PERCEPTRON

- Sigmoid units provide "soft" threshold
- Perceptrons provide "hard" threshold
- Expressive power is the same: limited to linearly separable instances

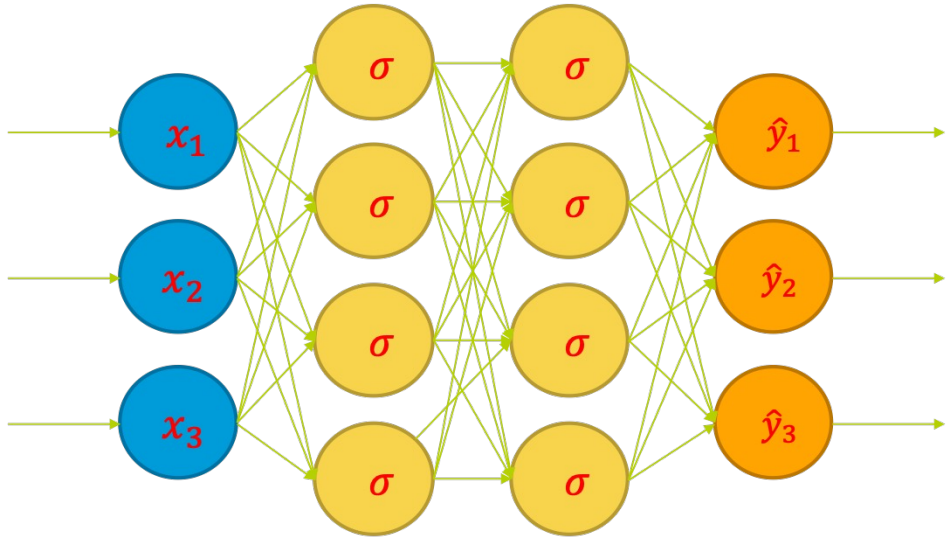
NN: KEY IDEAS

- Prediction: Feed-forward computation
- Training: Learning the weights by backpropogation



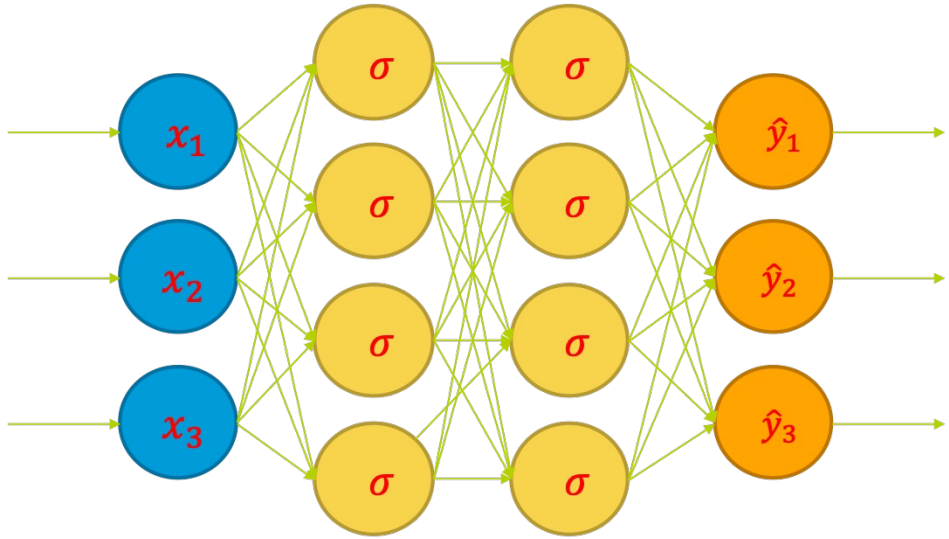
FEEDFORWARD NN

- Each unit of layer t is connected to every unit of the previous layer t - 1 only
- No cross-connections between units in the same layer

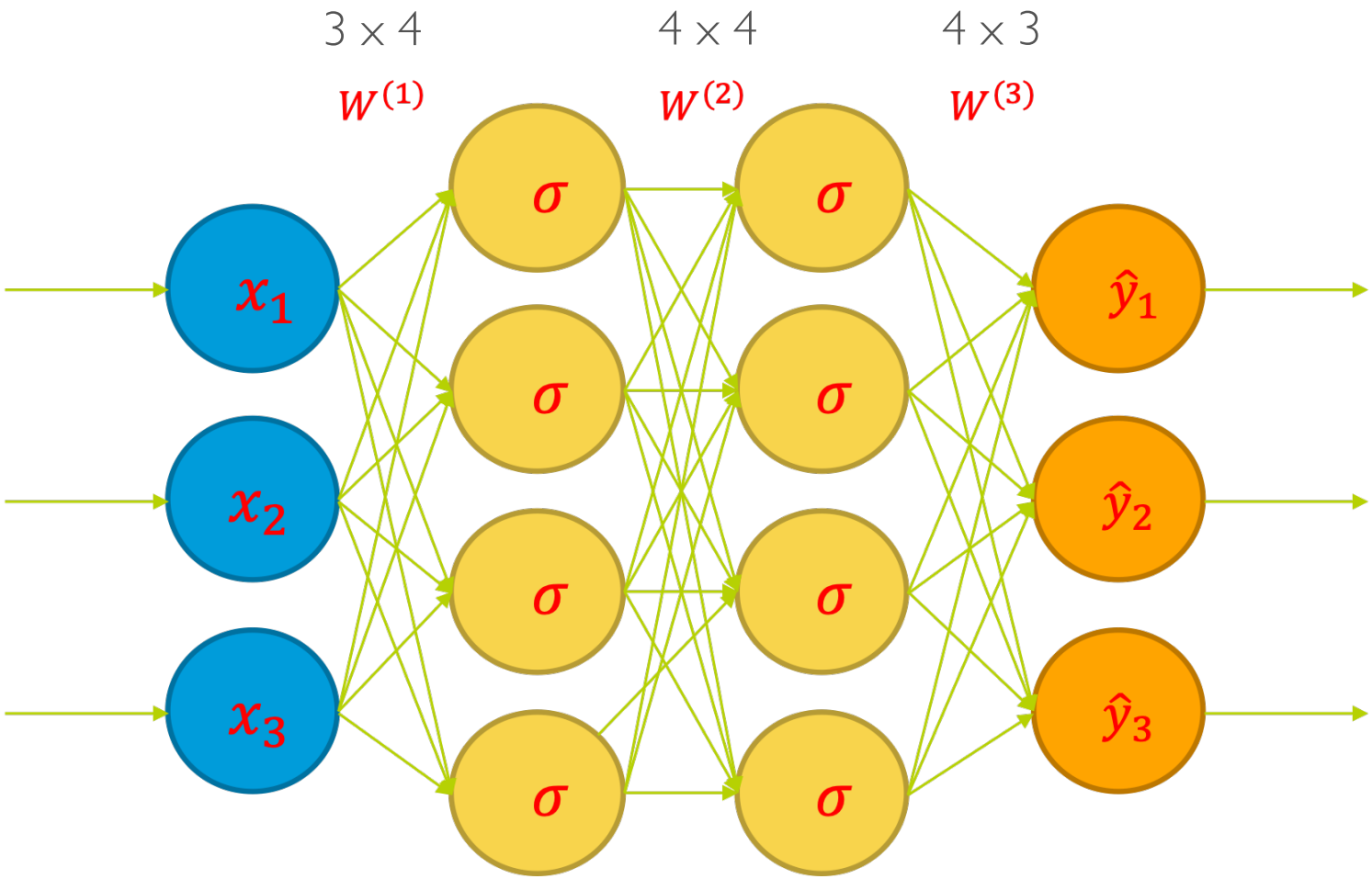


NN: FEED-FORWARD COMPUTATION

- Single forward pass to predict for a new sample
- For each layer
  - Compute the output of all neurons in the layer
  - Copy this output as inputs to the next layer and repeat until at the output layer

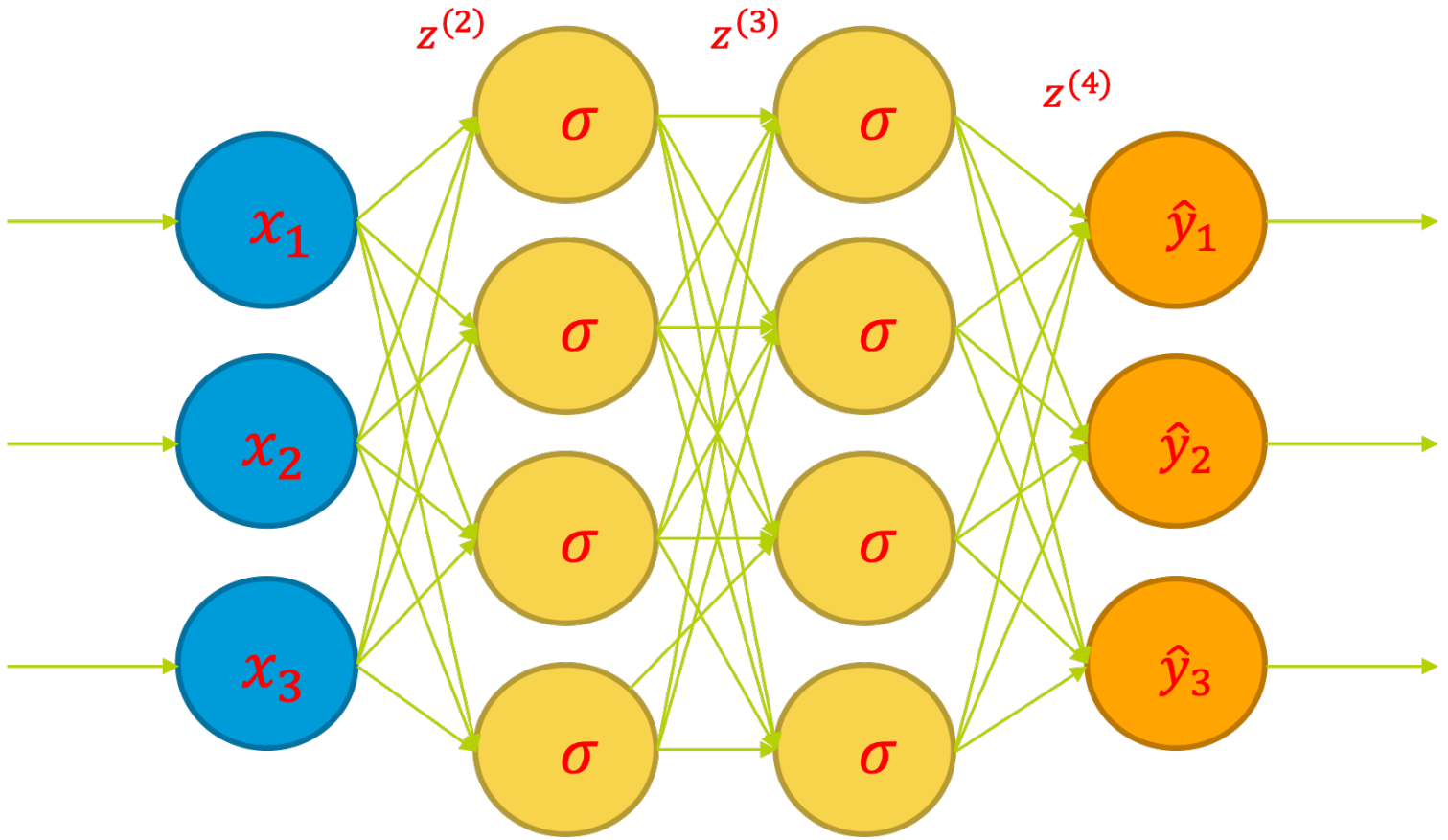


WEIGHTS: MATRIX REPRESENTATION

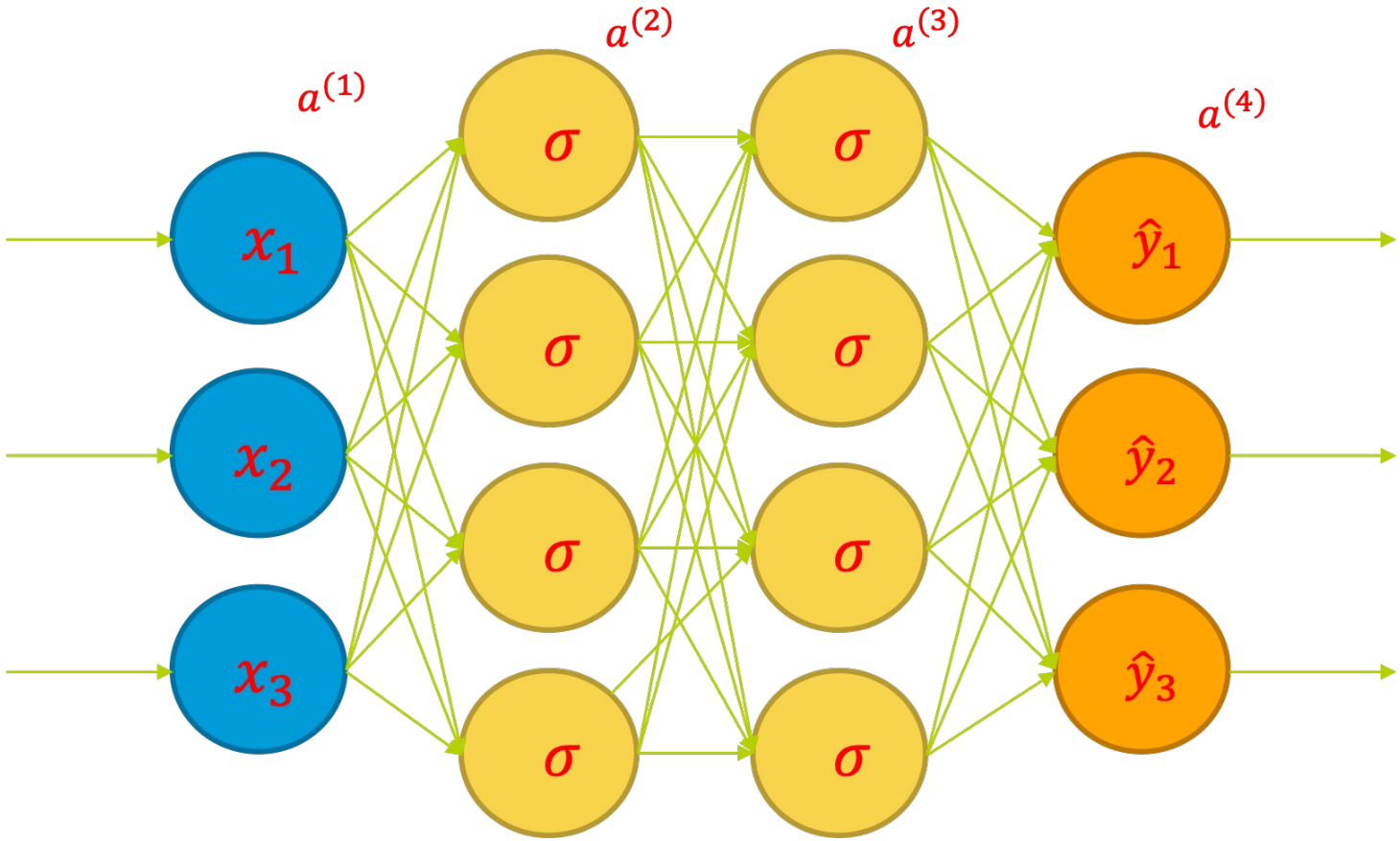


NET INPUT (SUM OF WEIGHTED INPUTS)

$$z = xw$$



ACTIVATIONS: OUTPUT OF NEURONS



MATRIX REPRESENTATION OF COMPUTATION

For a single data point (instance)

$$x = [x_1, x_2, x_3]$$

$$(x = a^{(1)})$$

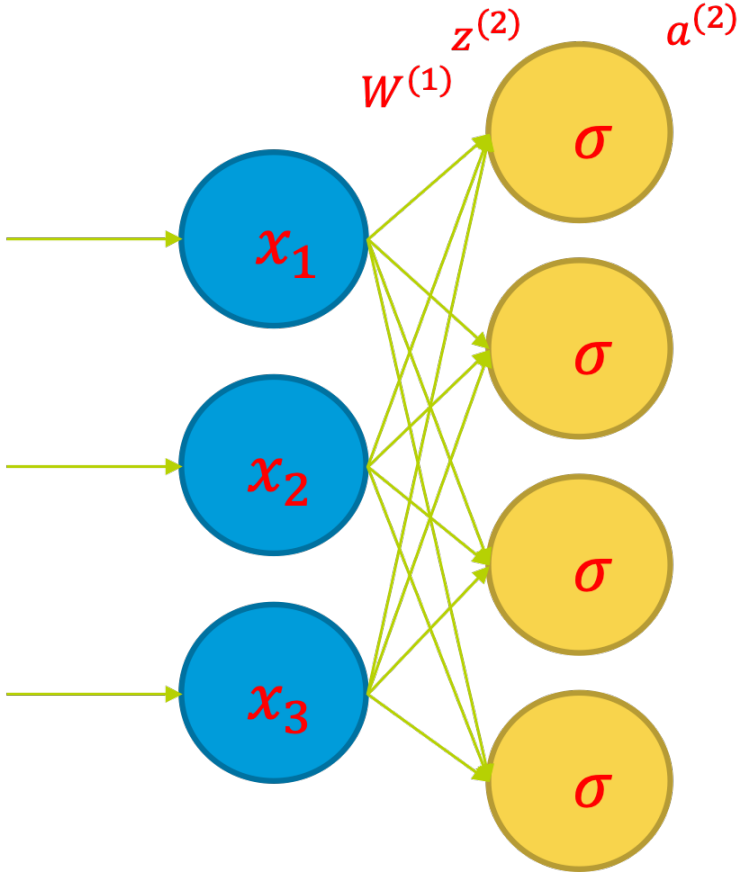
$$z^{(2)} = xW^{(1)}$$

$$a^{(2)} = \sigma(z^{(2)})$$

$W^{(1)}$ is a 3x4 matrix

$z^{(2)}$ is a 4-vector

$a^{(2)}$ is a 4-vector

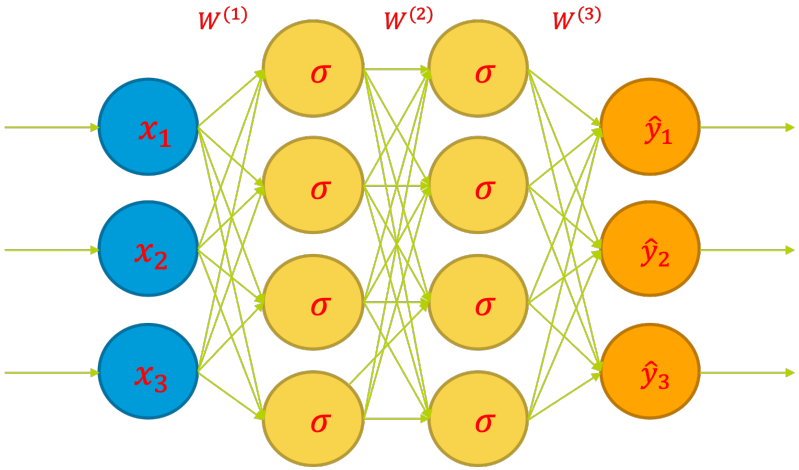


FORWARD COMPUTATION

$$z^{(2)} = xW^{(1)} \qquad a^{(2)} = \sigma(z^{(2)})$$

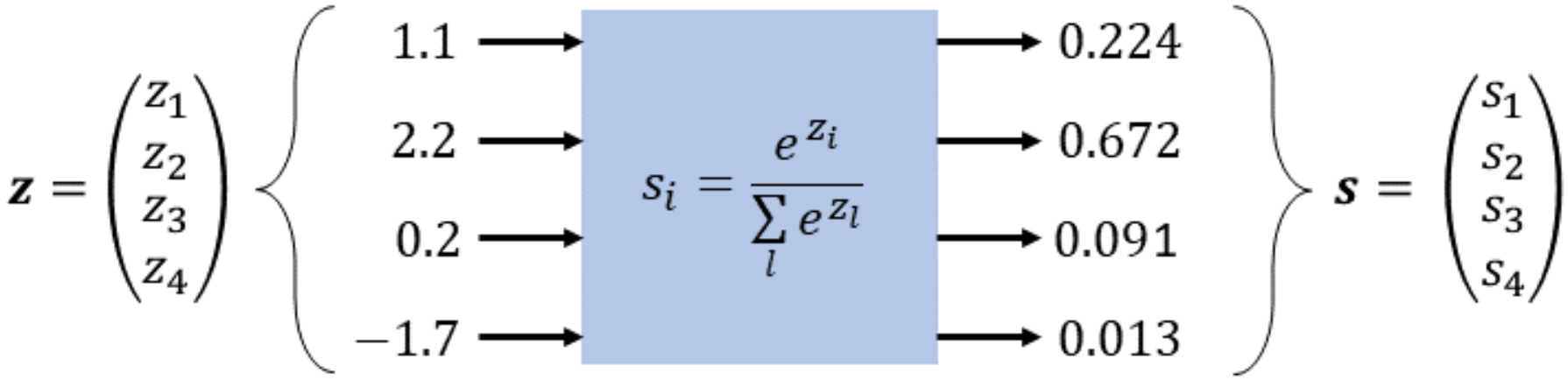
$$z^{(3)} = a^{(2)}W^{(2)} \qquad a^{(3)} = \sigma(z^{(3)})$$

$$z^{(4)} = a^{(3)}W^{(3)} \qquad \hat{y} = softmax(z^{(4)})$$



generalization of logistic function to multi-class problem

SOFTMAX FUNCTION

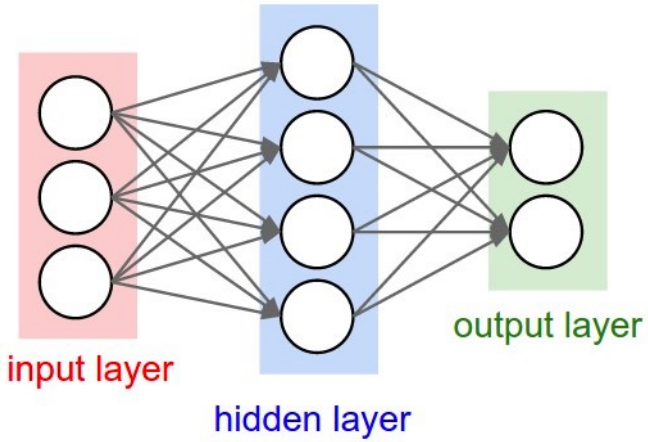


Gradient of cross entropy loss: $\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}} = \boldsymbol{s} - \boldsymbol{y}$

https://towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1

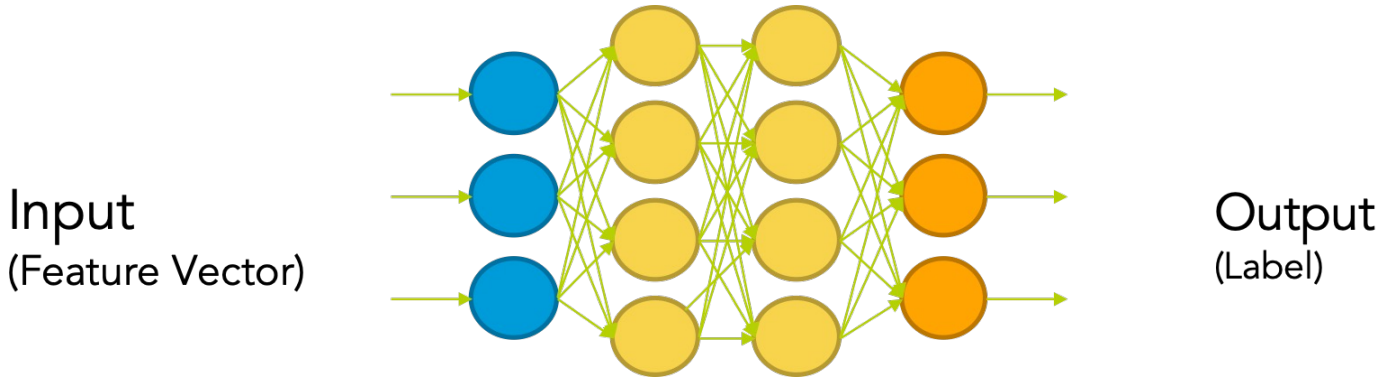
NN: KEY IDEAS

- Prediction: Feed-forward computation
- Training: Backpropogation



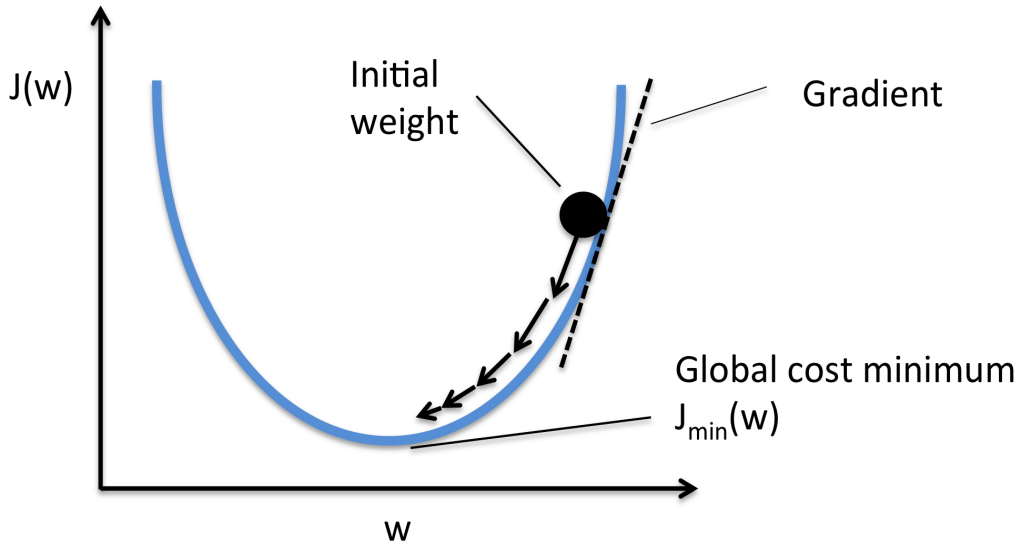
NN: LEARNING WEIGHTS

- Assume the network structure (units and connections) is given
- Learning problem is finding good set of weights

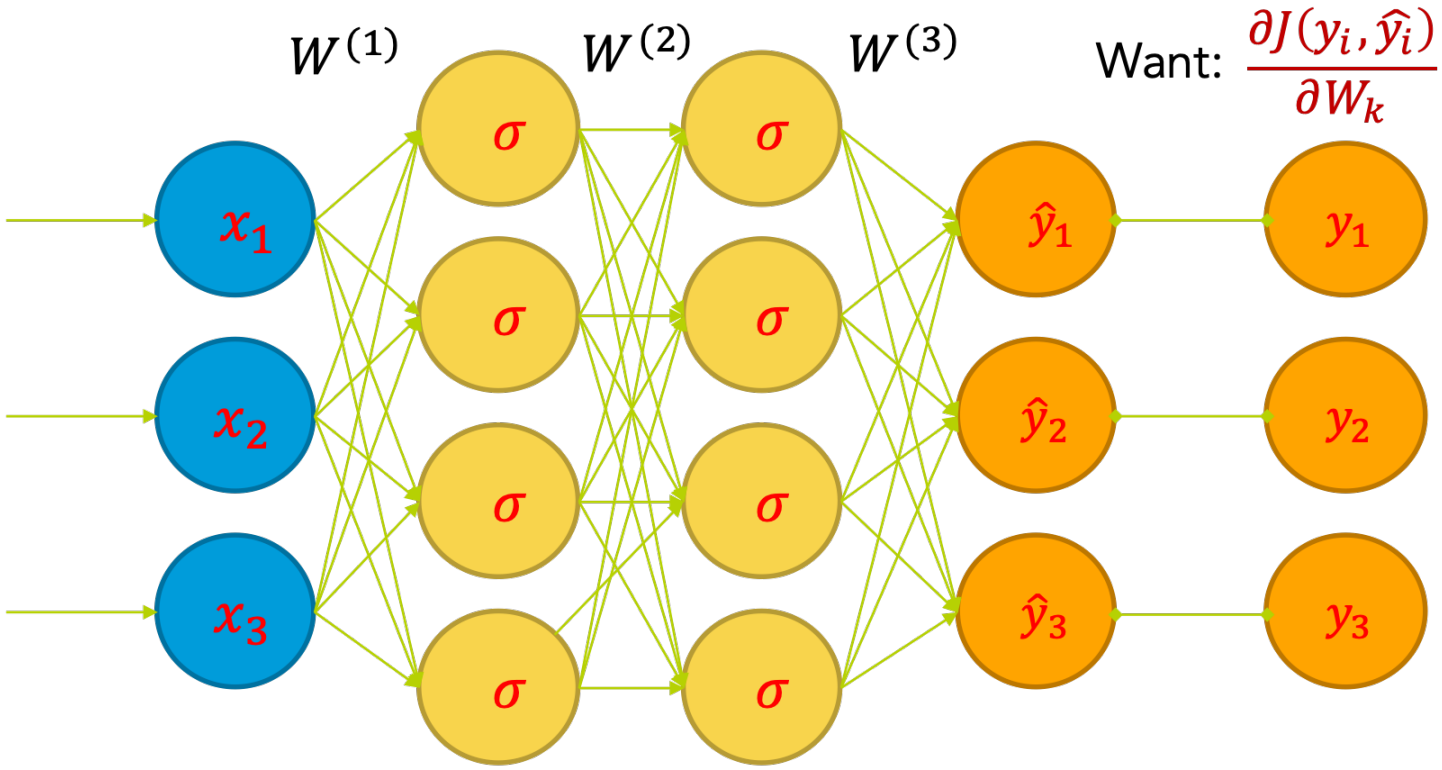


REVIEW: GRADIENT DESCENT

- Initialize weights/parameters
- Make prediction
- Calculate loss
- Calculate gradient of loss function with respect to parameters
- Gradient update by taking a step in the opposite direction
- Iterate

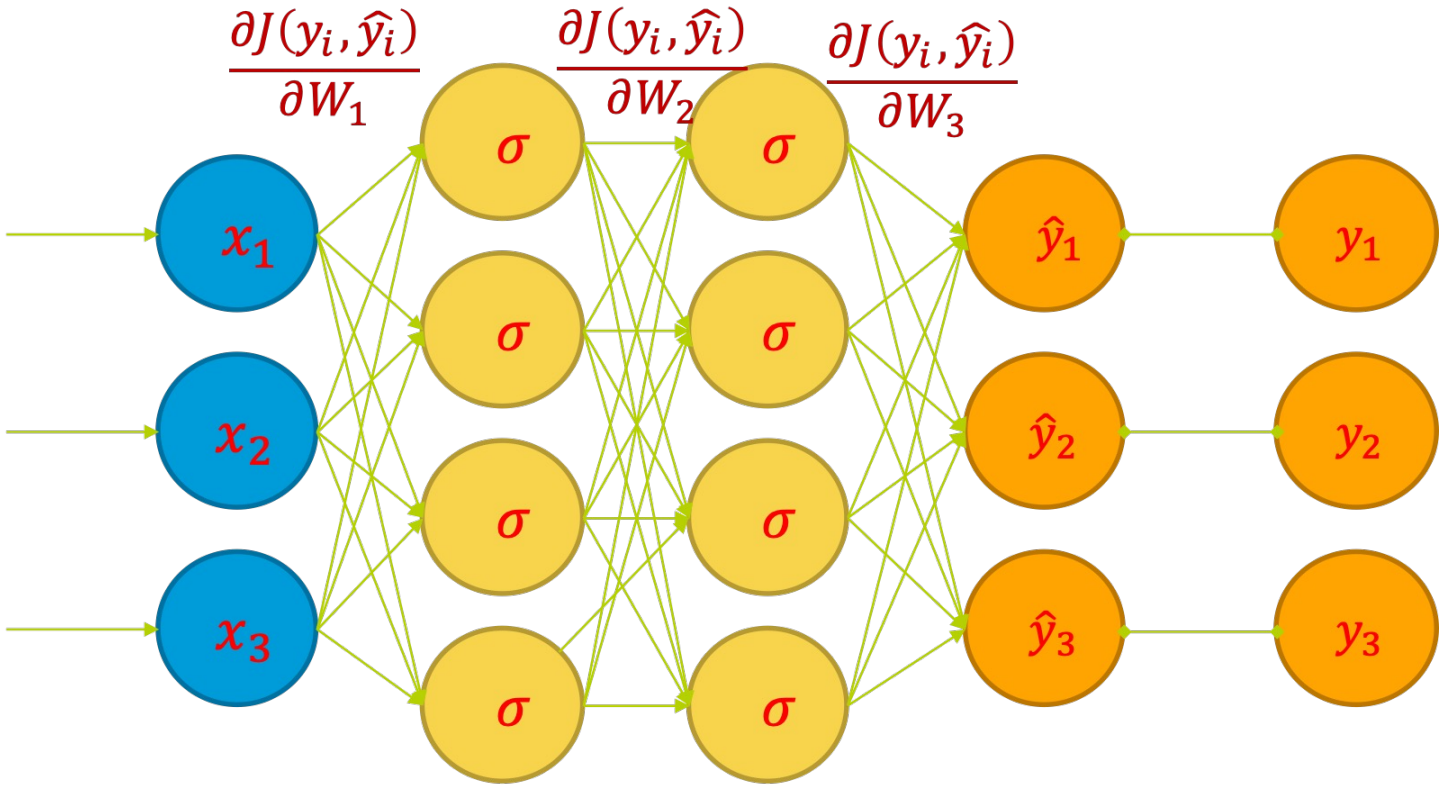


LEARNING THE WEIGHTS



BACKPROPAGATION ALGORITHM

- Backpropagation = gradient descent + chain rule
- Calculate error at output layer for each training example
- Propagate errors backward through the network and update the weights accordingly



https://openi.nlm.nih.gov/imgs/512/121/2716495/PMC2716495_bcr2257-1.png

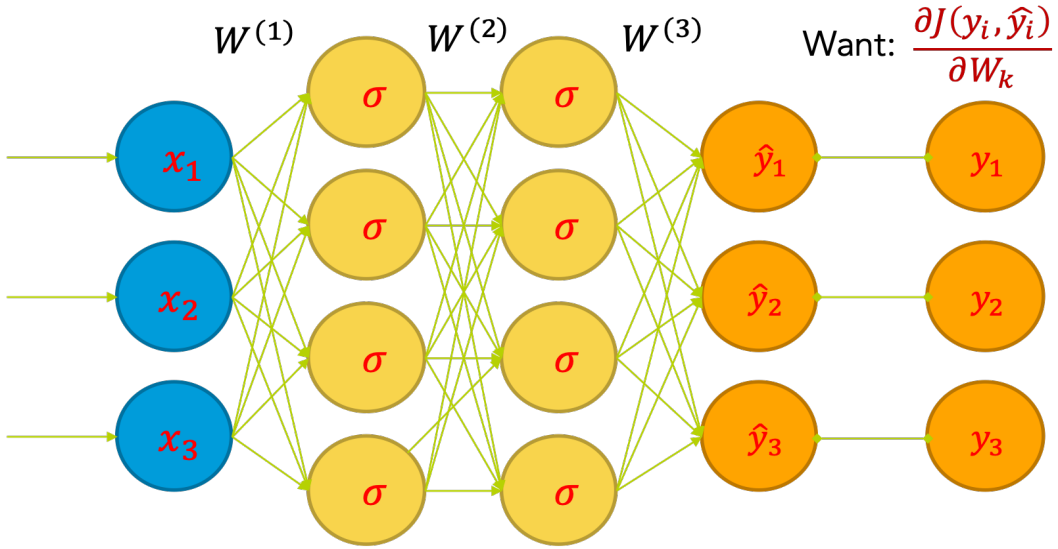
BACKPROPAGATION: EXAMPLE

Review of Feed Forward:

$$z^{(2)} = xW^{(1)} \qquad a^{(2)} = \sigma(z^{(2)})$$

$$z^{(3)} = a^{(2)}W^{(2)} \qquad a^{(3)} = \sigma(z^{(3)})$$

$$z^{(4)} = a^{(3)}W^{(3)} \qquad \hat{y} = softmax(z^{(4)})$$



Using cross entropy loss, what is the derivative with respect to W_3?

$$\frac{\partial J}{\partial z^{(4)}} = (\hat{y} - y)$$

$$\frac{\partial J}{\partial W^{(3)}} = (\hat{y} - y) \cdot a^{(3)}$$

https://towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1

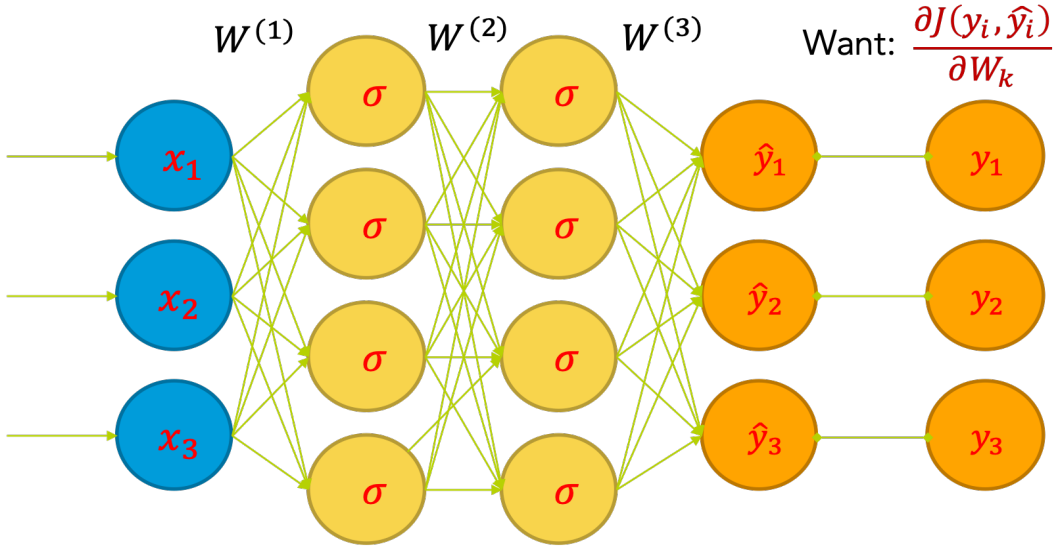
BACKPROPAGATION: EXAMPLE

Review of Feed Forward:

$$z^{(2)} = xW^{(1)} \qquad a^{(2)} = \sigma(z^{(2)})$$

$$z^{(3)} = a^{(2)}W^{(2)} \qquad a^{(3)} = \sigma(z^{(3)})$$

$$z^{(4)} = a^{(3)}W^{(3)} \qquad \hat{y} = softmax(z^{(4)})$$



What is the derivative with respect to W_2?

$$\frac{\partial J}{\partial W^{(2)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot a^{(2)}$$

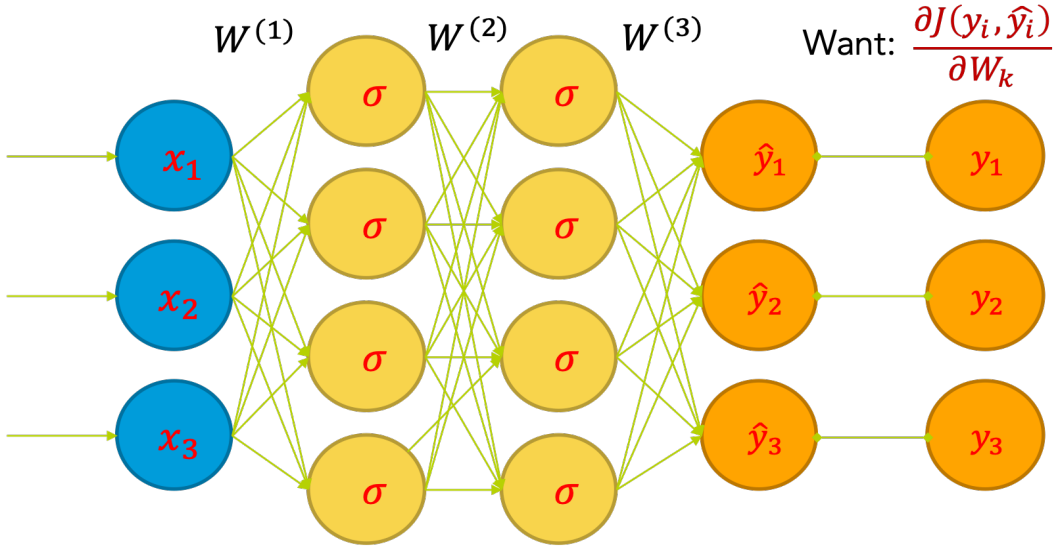
BACKPROPAGATION: EXAMPLE

Review of Feed Forward:

$$z^{(2)} = xW^{(1)} \qquad a^{(2)} = \sigma(z^{(2)})$$

$$z^{(3)} = a^{(2)}W^{(2)} \qquad a^{(3)} = \sigma(z^{(3)})$$

$$z^{(4)} = a^{(3)}W^{(3)} \qquad \hat{y} = softmax(z^{(4)})$$



What is the derivative with respect to W_1?

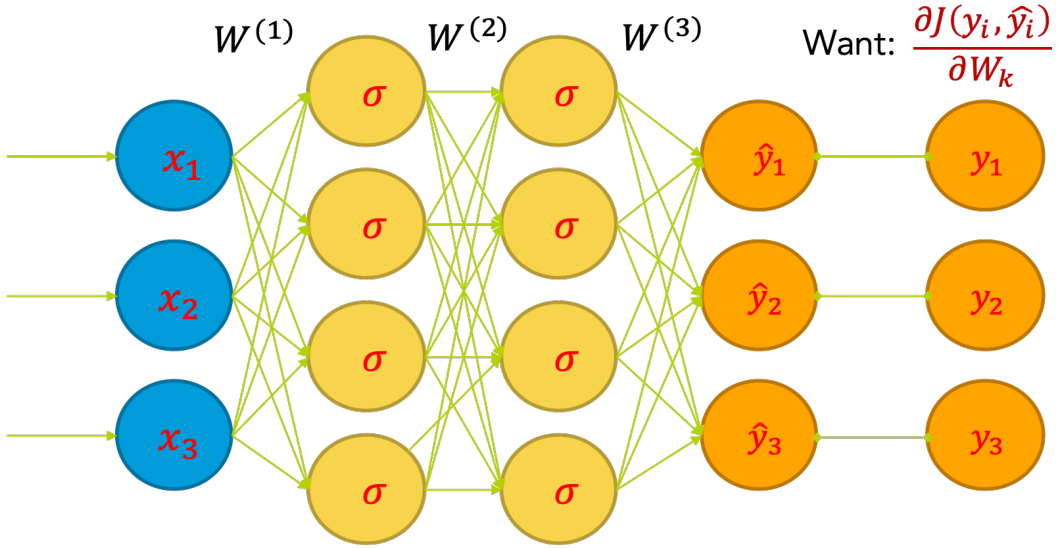
$$\frac{\partial J}{\partial W^{(1)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot W^{(2)} \cdot \sigma'\left(z^{(2)}\right) \cdot X$$

BACKPROPAGATION: EXAMPLE

$$\frac{\partial J}{\partial W^{(3)}} = (\hat{y} - y) \cdot a^{(3)}$$

$$\frac{\partial J}{\partial W^{(2)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot a^{(2)}$$

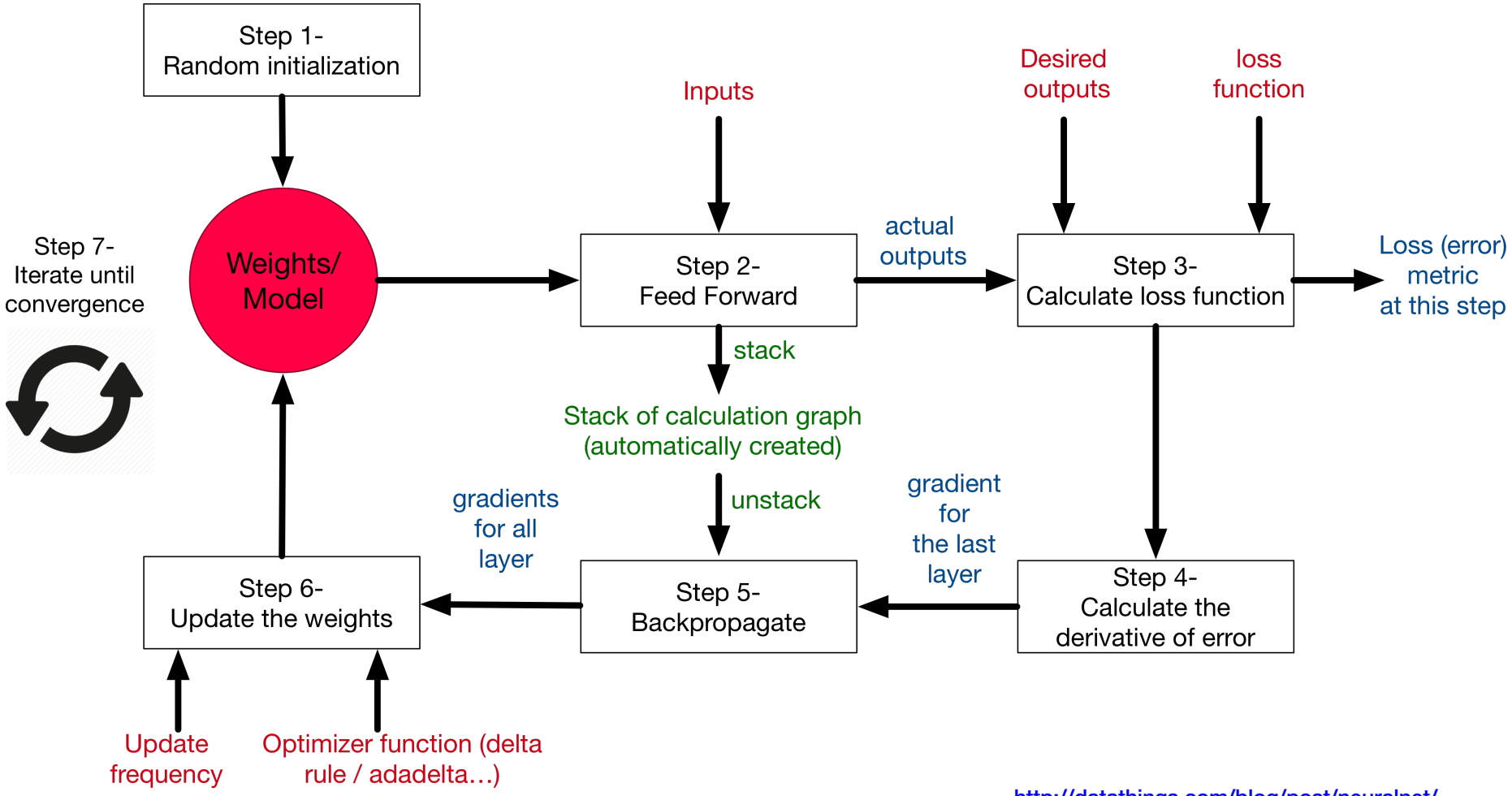
$$\frac{\partial J}{\partial W^{(1)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot W^{(2)} \cdot \sigma'\left(z^{(2)}\right) \cdot X$$



vanishing gradient

- Recall that: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- Though they appear complex, above are easy to compute!

LEARNING PROCESS

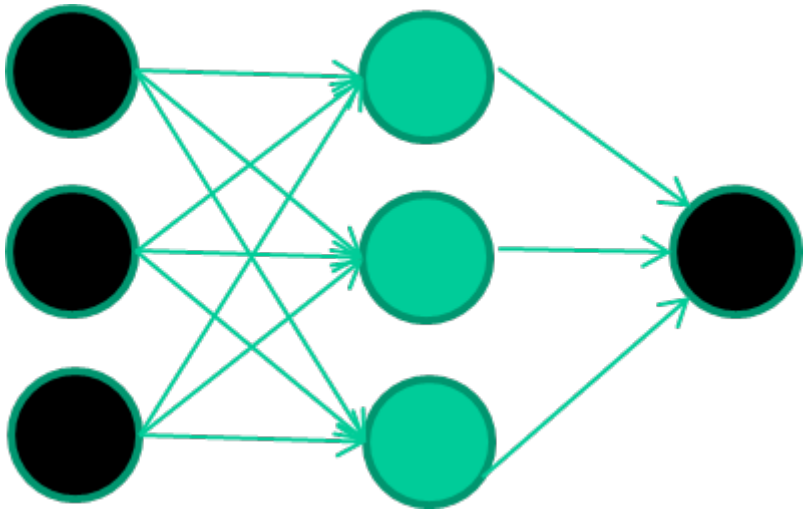


http://datathings.com/blog/post/neuralnet/

Training the neural network

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

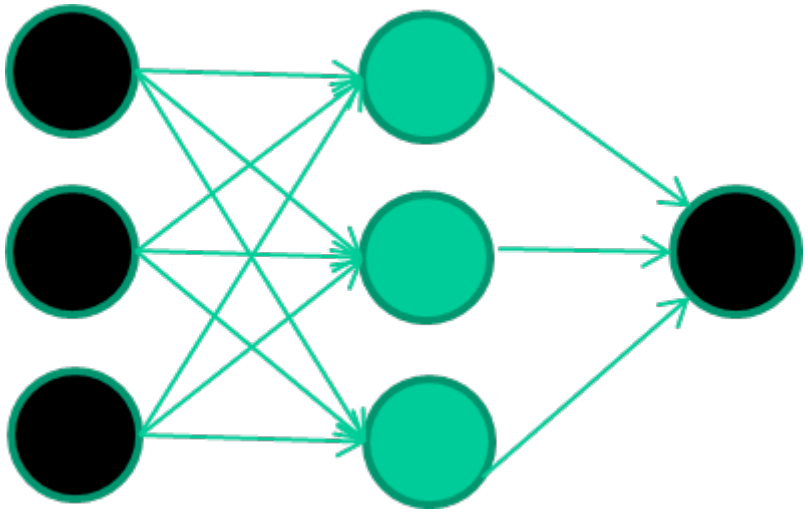


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

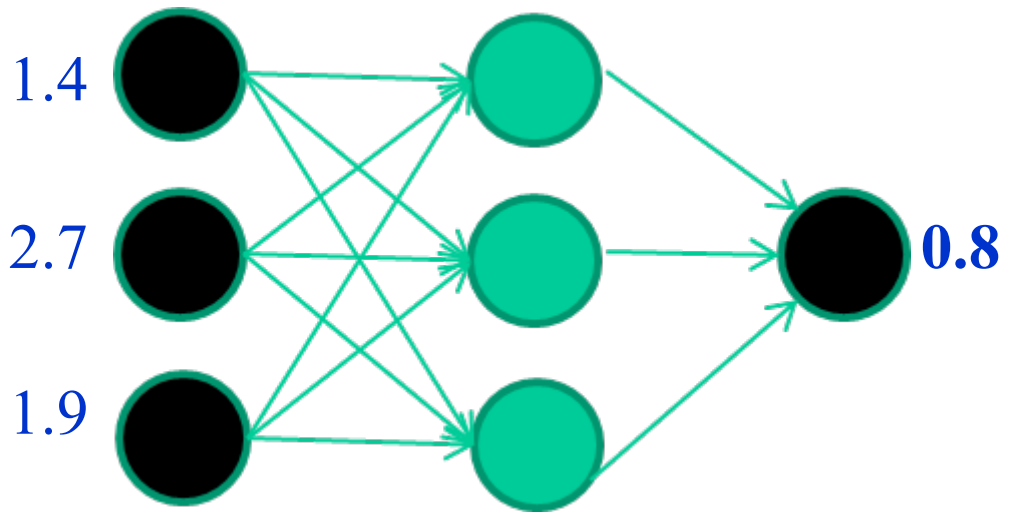
Initialise with random weights



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

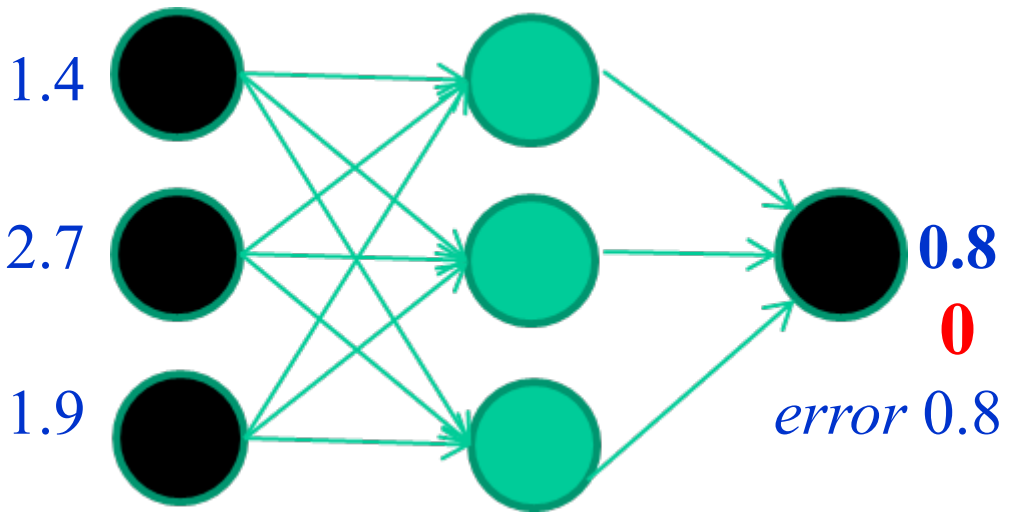
Feed it through to get output



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

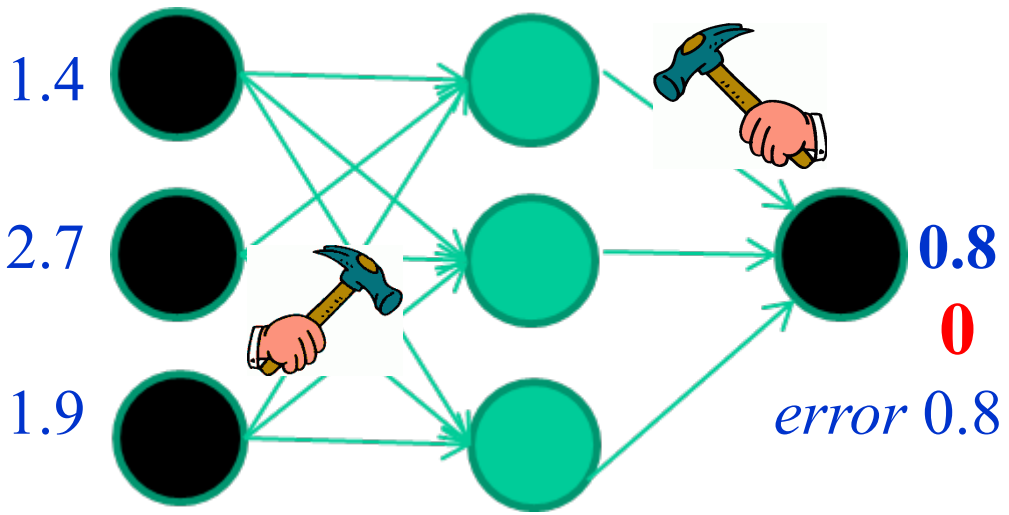
Compare with target output



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Adjust weights based on error

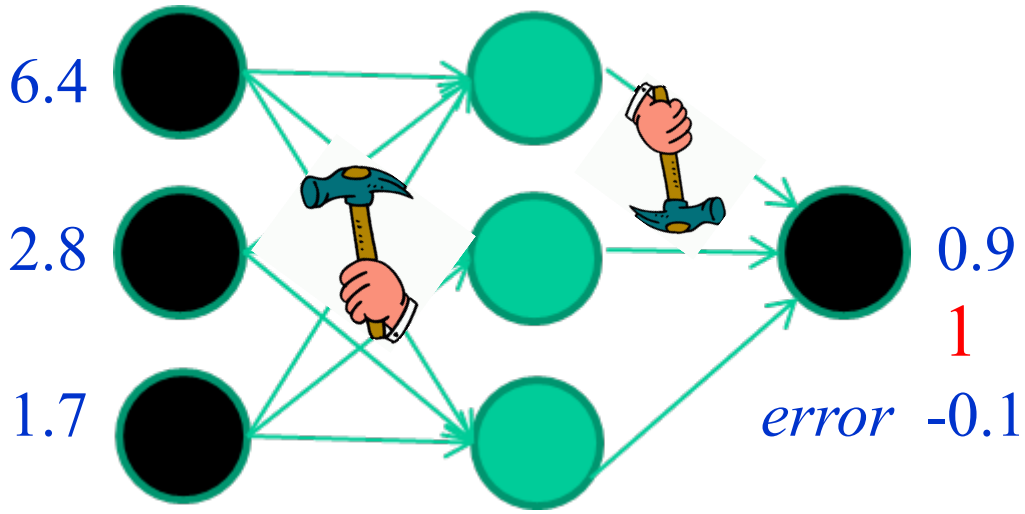


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

And so on ….



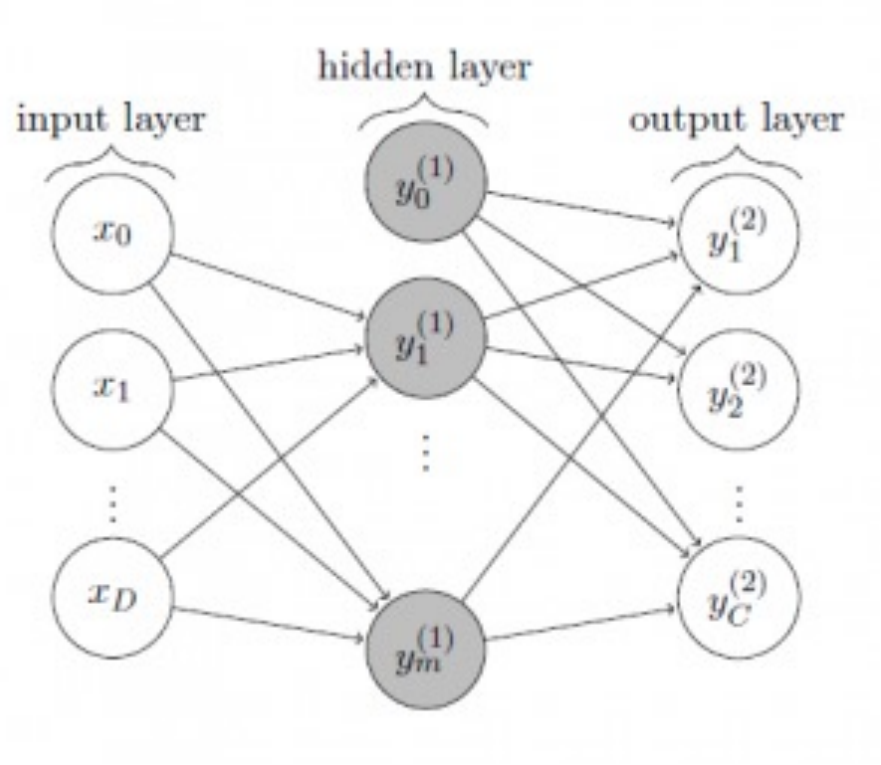
Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments *(stochastic gradient descent)*

MNIST DATASET

- Scanned 28 x 28 greyscale images of handwritten digits
- Training data
  - 60,000 images
  - 250 people
- Test Data
  - 10,000 images
  - Different 250 people

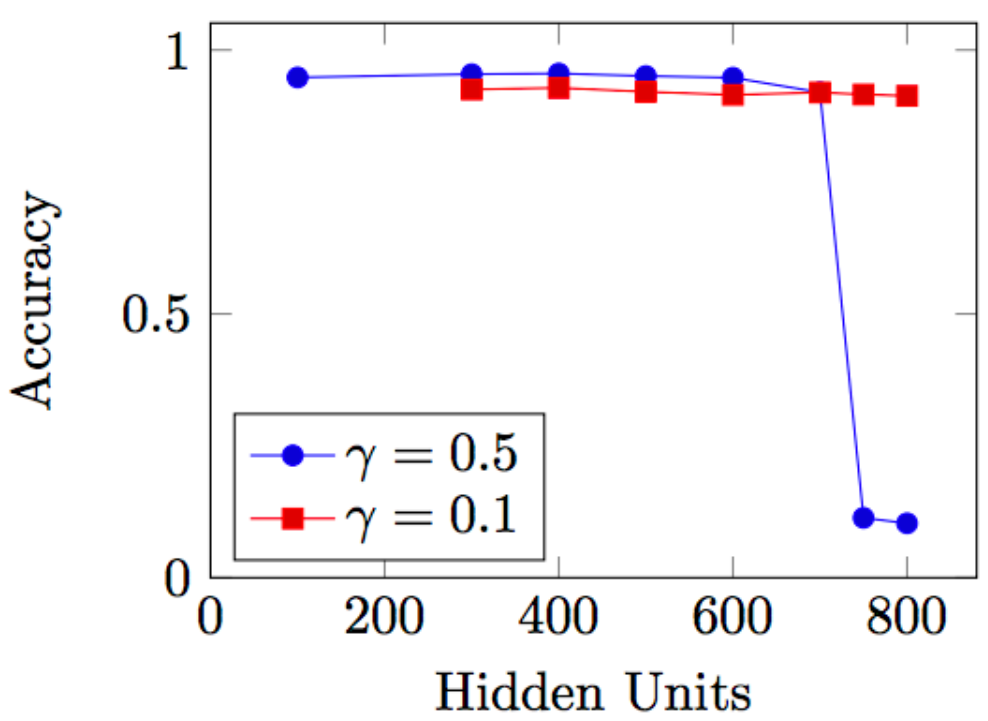
EXPERIMENT: 2 LAYER PERCEPTRON

- 784 (28*28) input units, variable number of hidden units, and 10 output units
- Ground truth labels use one-hot encoding
- Activation function = logistic sigmoid
- Sum of squared error function
- Stochastic variant of mini-batch training

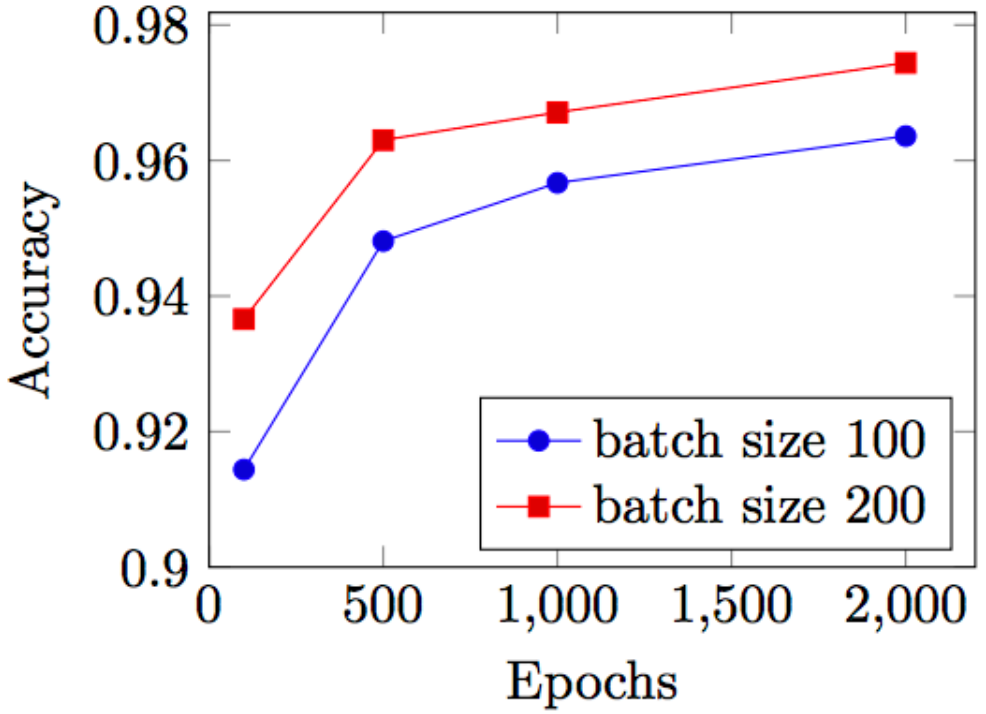


http://davidstutz.de/recognizing-handwritten-digits-mnist-dataset-twolayer-perceptron

EXPERIMENT: 2 LAYER PERCEPTRON



(a) 500 epochs with batch size 100.



(b) 500 epochs with learning rate $\gamma = 0.5$.

http://davidstutz.de/wordpress/wp-content/uploads/2014/03/seminar.pdf