# Neural Networks

Kevin Wang

Colaboratory notebook:

https://colab.research.google.com/drive/1I-GsUE-Cgx3T9vvdPoGukHTyEuJ2-EjW

# You should know the following things

- Material from any one of the previous lessons
  - How to represent inputs *X*, *y*; weights *w* and bias *b*; output *y_hat* a.k.a *a*
  - Forward propagation: using *w*, *b* to calculate *a*
  - Backward propagation: using gradient descent to adjust *w*, *b*

# “Deep learning” deals with neural networks

- Deep learning is a subset of machine learning
- Most machine learning research today is focused in deep learning
- “Fathers of the Deep Learning Revolution”:

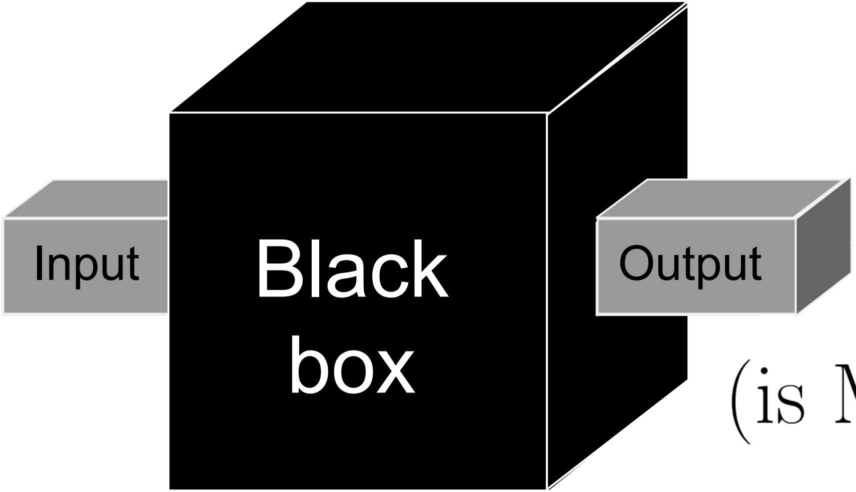
Yoshua Bengio
*Speech recognition, natural language processing (NLP)*

Geoffrey Hinton
*Backpropagation, Boltzmann Machines*

Yann LeCun
*Convolutional neural networks, optical character recognition (OCR)*

# Neural networks can do a wide range of things

- For today, we'll keep it on classification task
  - Output 1 if input class is 1
  - Output 0 if input class is 0

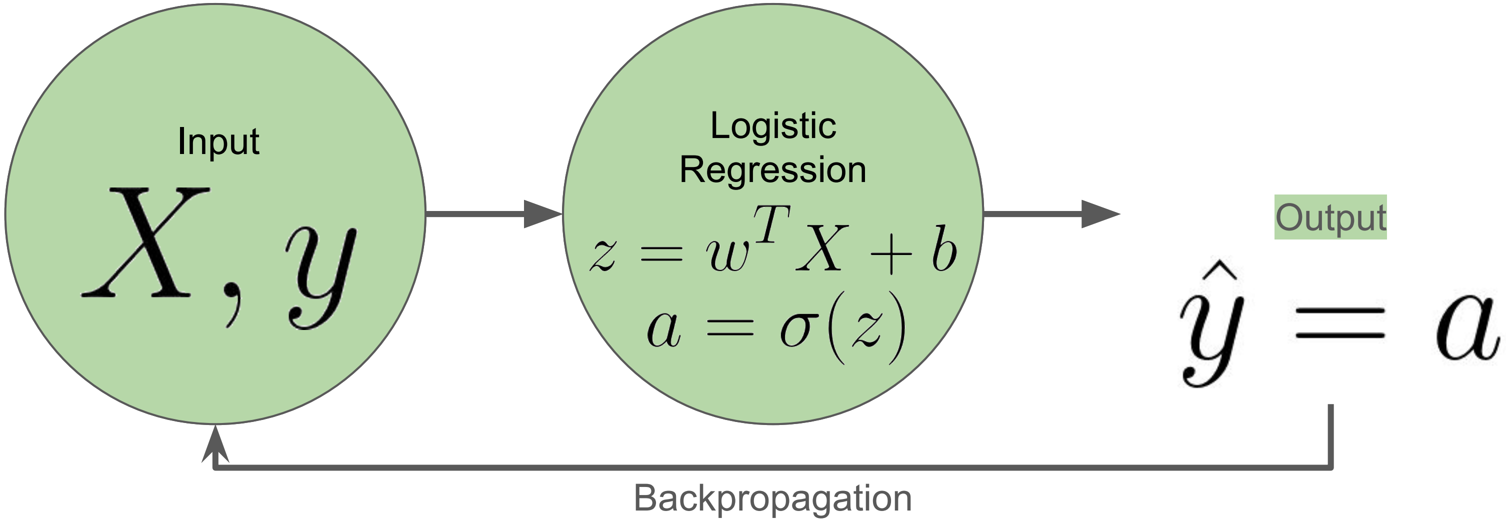




$1$

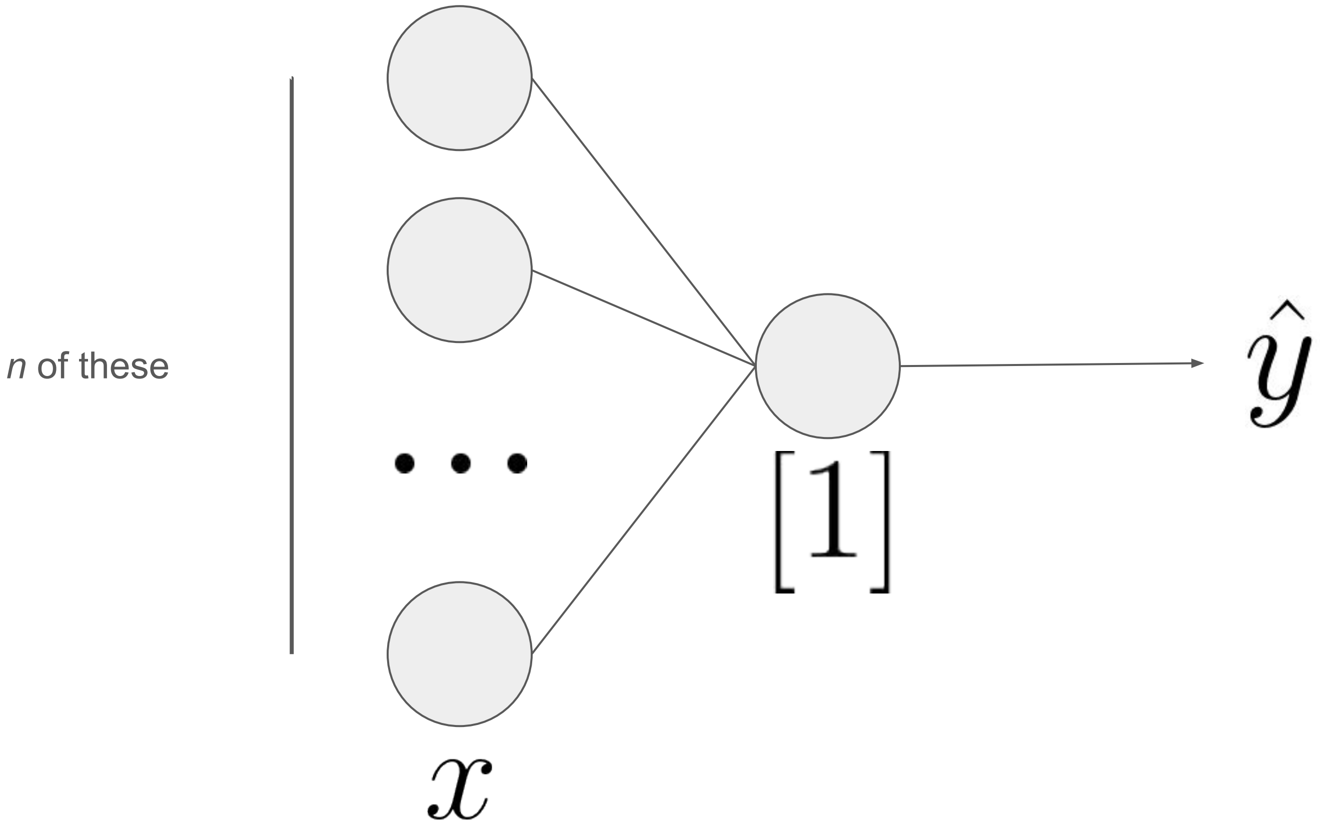
(is Michigan logo)

# But what is a neural network?

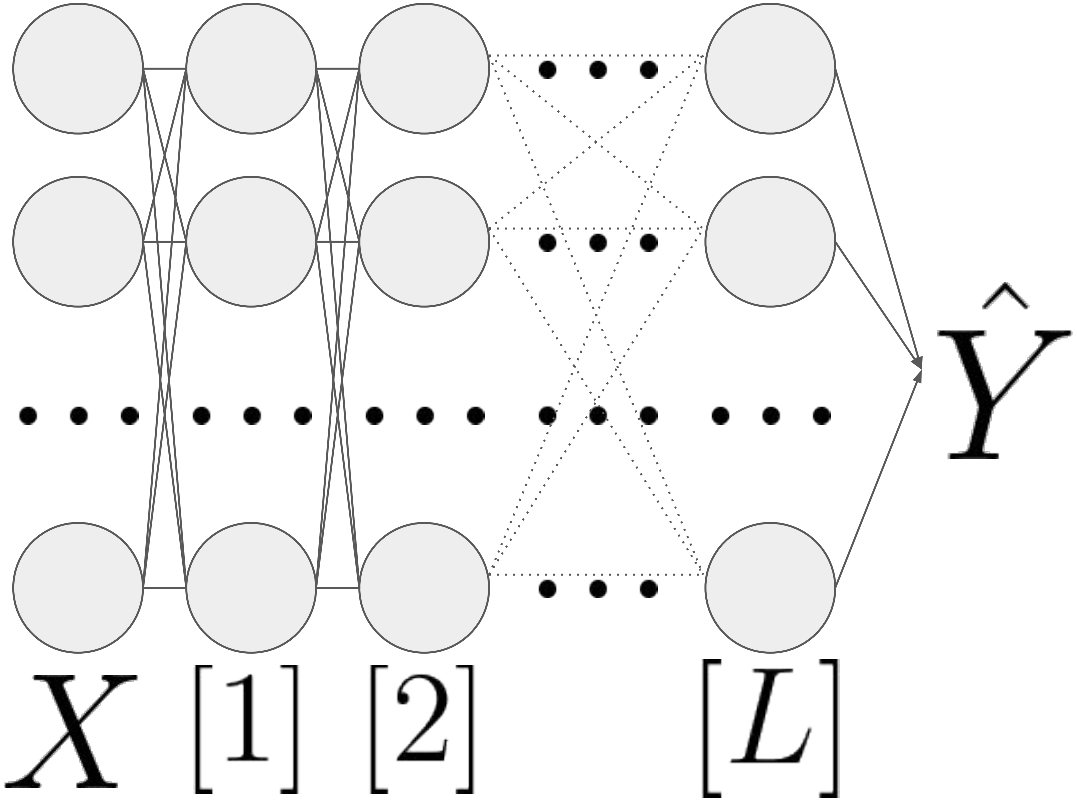
- Logistic regression can actually be considered a very simple neural network



# We can redraw and label logistic regression



# This is a more general form for any neural network



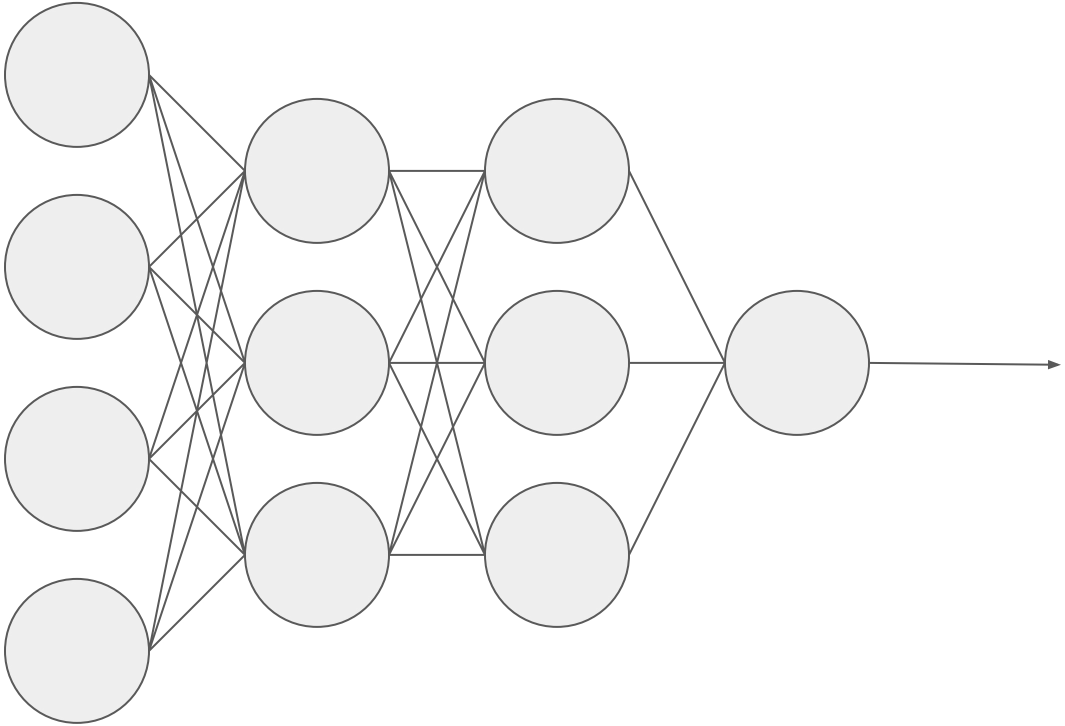

- We'll use entirety of input at once
- Note input layer is not layer 1

# We indicate the *l*th hidden layer w/ square brackets

- e.g. [2] indicates the 2nd hidden layer

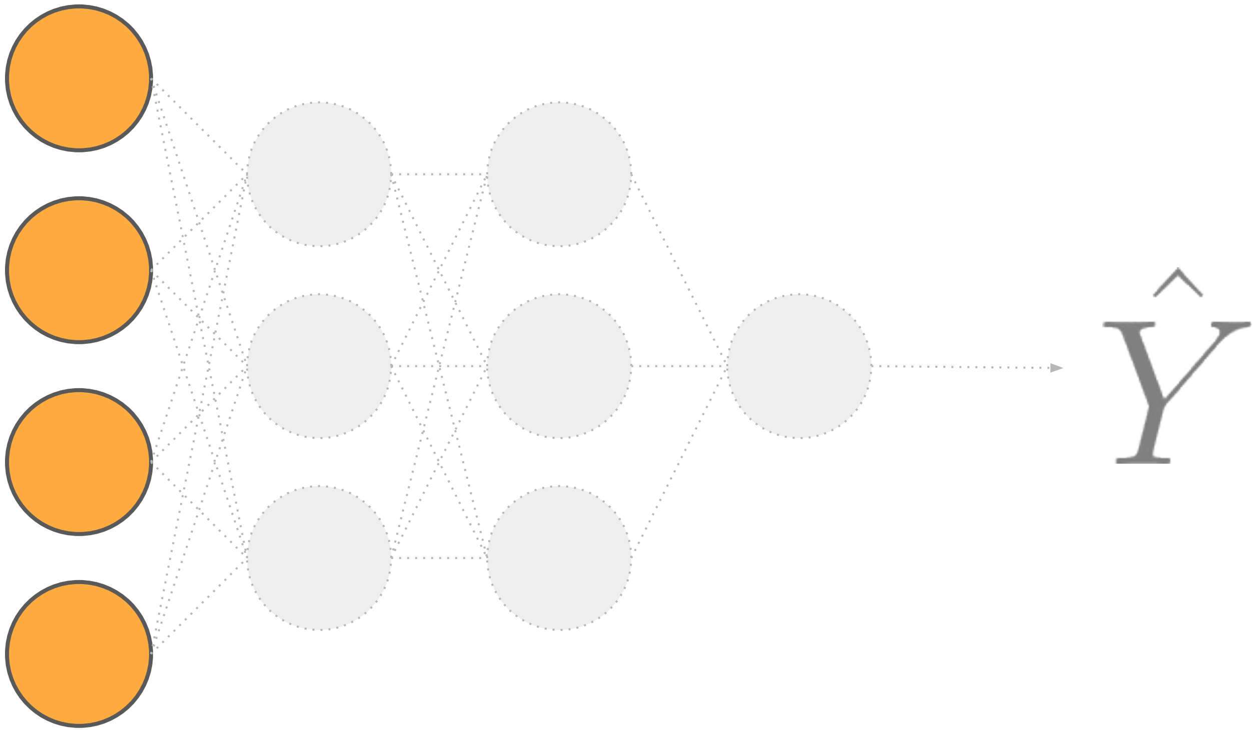
- *L* hidden layers total
- Use [l] for a generic hidden layer
    - This is the letter, not the number :)

# We will use a smaller model so that it fits on screen



$\hat{Y}$

# What's different about input $\mathcal{X}$?



# Nothing, actually

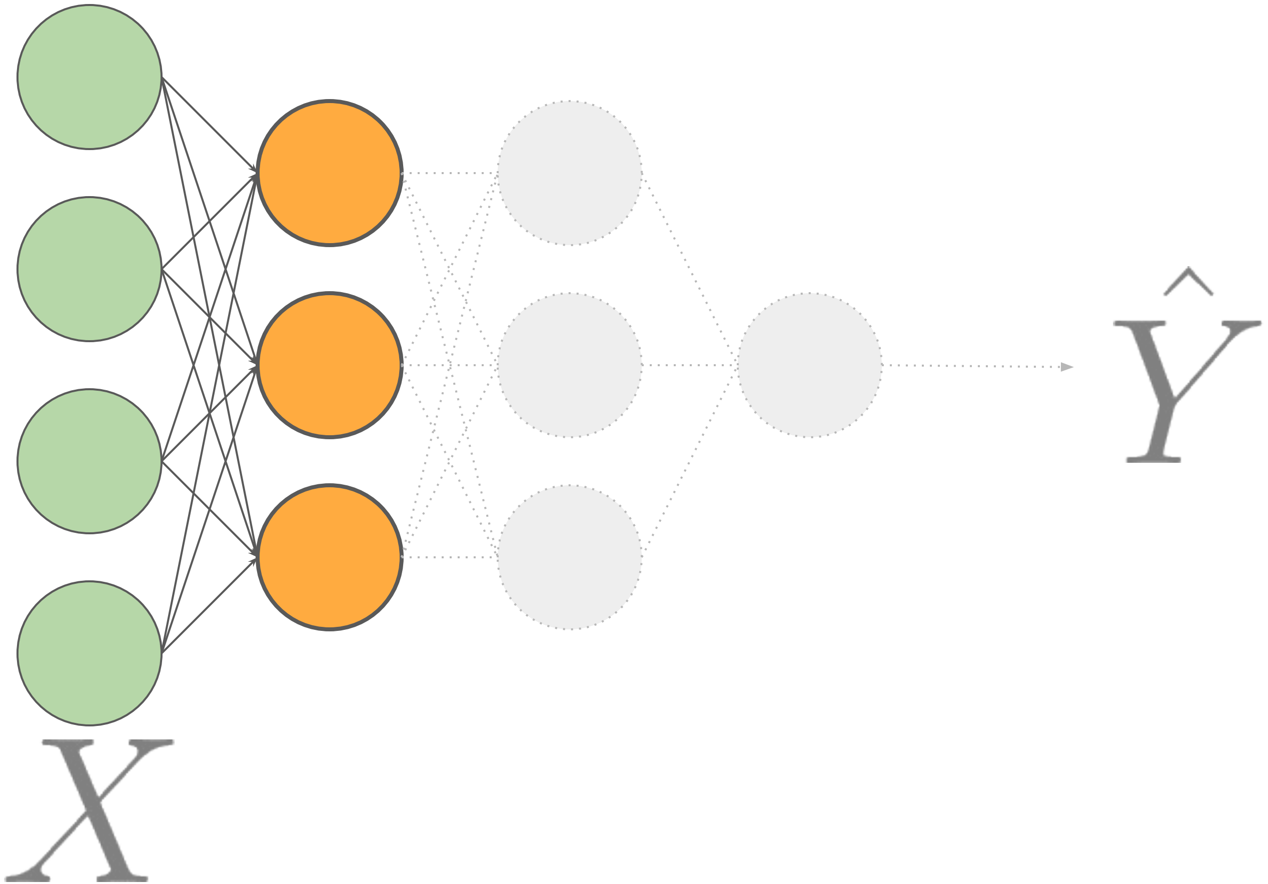
- *x* is still a column vector
  - Each entry is a feature
- *X* is still the matrix of examples *x*

$$X = \begin{bmatrix} | & | & | & & | \\ | & | & | & & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \dots & x^{(m)} \\ | & | & | & & | \\ | & | & | & & | \end{bmatrix}$$

- As well, Y is still a vector
  - Labels for every example

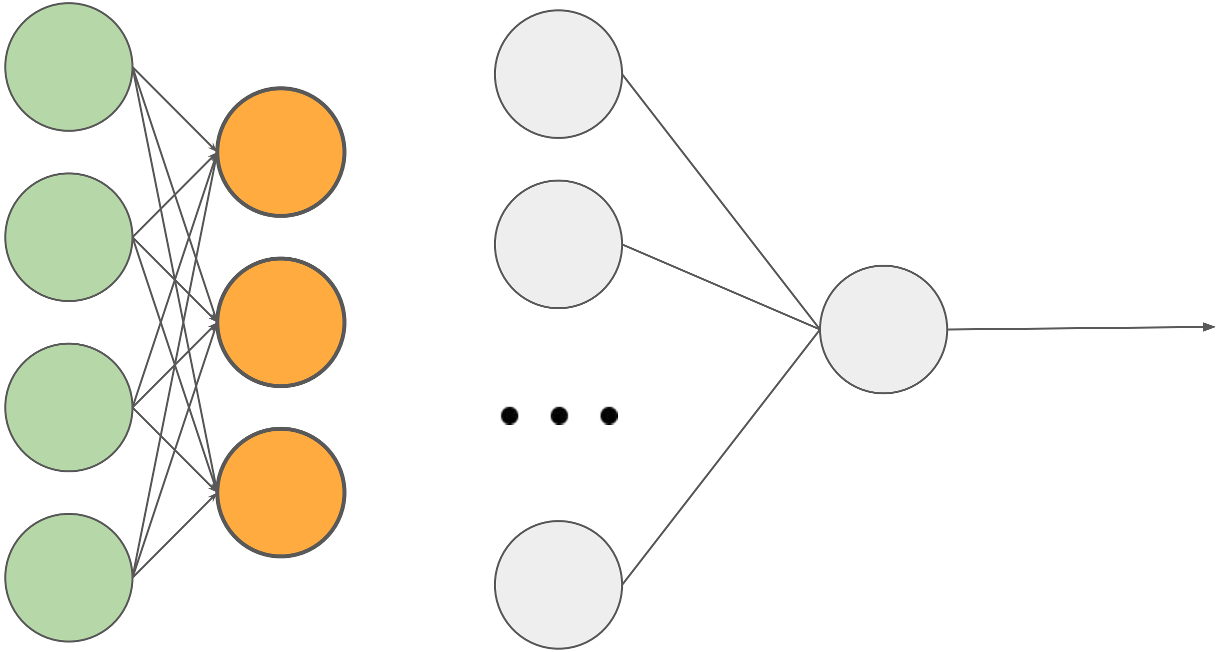
$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \dots \\ y^{(m)} \end{bmatrix}$$

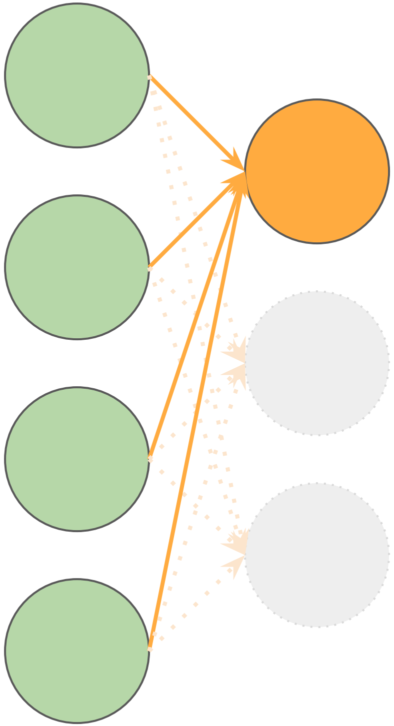
# What about the weights $w$ and biases $b$?



# $W$ is now a matrix and $b$ is now a vector

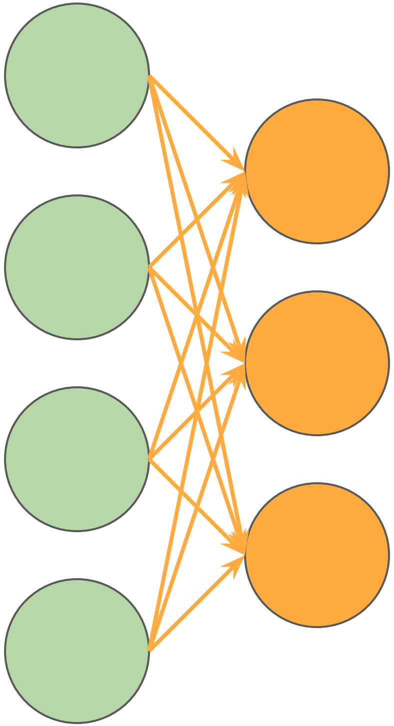
- This is because we have >1 nodes in the second layer
- Each layer has its own weights $W^{[l]}$ and bias $b^{[l]}$





$$W^{[1]} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

- This is what weights in logistic regression would look like
  - One row indicating weights from previous nodes to current node



$$W^{[1]} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix}$$

- With more nodes, simply add more rows to your matrix
    - Row ⇒ second layer node, column ⇒ first layer node

# Dimensions depend on current and previous layer

- Number of nodes in layer *l* is $n^{[l]}$
  - Note the square brackets to indicate layer!
- For $W^{[l]}$, there are $n^{[l]}$ rows and $n^{[l-1]}$ columns
  - This is to ensure *WX* is a valid matrix multiplication

$$\begin{array}{cc} & \xrightarrow{\qquad n^{[l-1]} \qquad} \\ W^{[1]} = & \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} \Big\downarrow n^{[l]} \end{array}$$

# General form

$$W^{[l]} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1(n^{[l-1]})} \\ w_{21} & w_{22} & \cdots & w_{2(n^{[l-1]})} \\ \cdots & \cdots & \cdots & \cdots \\ w_{(n^{[l]})1} & w_{(n^{[l]})2} & \cdots & w_{n^{[l]}n^{[l-1]}} \end{bmatrix}$$

# Dimensions depend on current and previous layer

- There must be a bias value for every node in current layer
  - *b* is a vector of dimensions ($n^{[l]}$, 1)

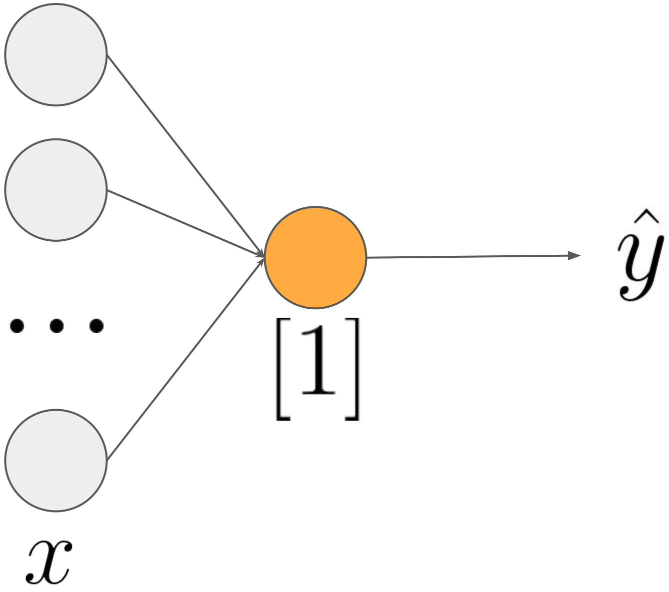
$$b^{[1]} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \Big\downarrow n^{[l]}$$

General form:

$$b^{[l]} = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_{n^{[l]}} \end{bmatrix}$$

# Calculating $\mathcal{Z}$

- Previously, we had a single scalar $z^{(i)}$ for any *i*th instance
  - This is because we had one node performing logistic regression

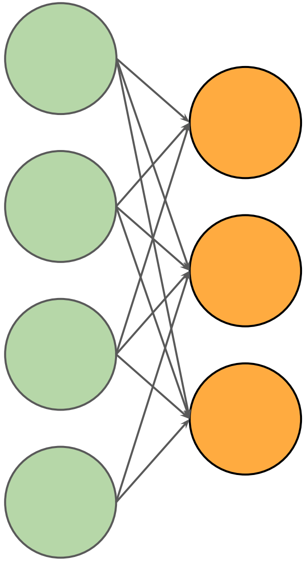



$$Z = \begin{bmatrix} z^{(1)} & z^{(2)} & z^{(3)} & \dots & z^{(m)} \end{bmatrix}$$

Each column corresponds to one example

# Calculating $\mathcal{Z}$

- Now, we have $n^{[l]}$ values of z for each instance
  - Each z is thus a vector
  - Z is thus a matrix



$$Z^{[1]} = \begin{array}{cc} m & \\ \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1m} \\ z_{21} & z_{22} & \cdots & z_{2m} \\ z_{31} & z_{32} & \cdots & z_{3m} \end{bmatrix} & n^{[l]} \end{array}$$

Each column corresponds to one example

# Equation barely changes from logistic regression

- Only difference: no need to take the transpose
  - Dimensions already line up!
- Remember that we take advantage of broadcasting

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

- Compare:

$$Z = w^T X + b$$

# General form

$$Z^{[l]} = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1m} \\ z_{21} & z_{22} & \cdots & z_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ z_{(n^{[l]})1} & z_{(n^{[l]})2} & \cdots & z_{(n^{[l]})m} \end{bmatrix}$$

# Calculating $A$

- For logistic regression, we input $z^{(i)}$ to predict class of *i*th example
  - We called this $a^{(i)}$, activation of $z^{(i)}$
- We can do the same, just with all values in $Z^{[l]}$
  - $A^{[l]}$ is a matrix with same dimensions of $Z^{[l]}$

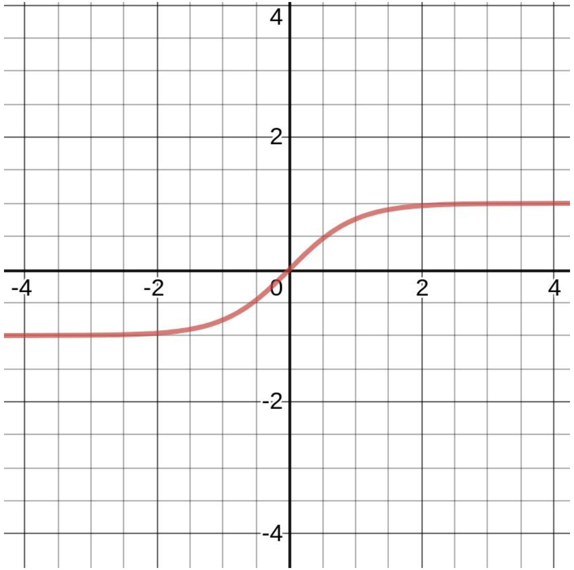
$$A^{[1]} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ a_{31} & a_{32} & \dots & a_{3m} \end{bmatrix}$$

General form:

$$A^{[l]} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{(n^{[l]})1} & a_{(n^{[l]})2} & \dots & a_{(n^{[l]})m} \end{bmatrix}$$

# But there are other activation functions

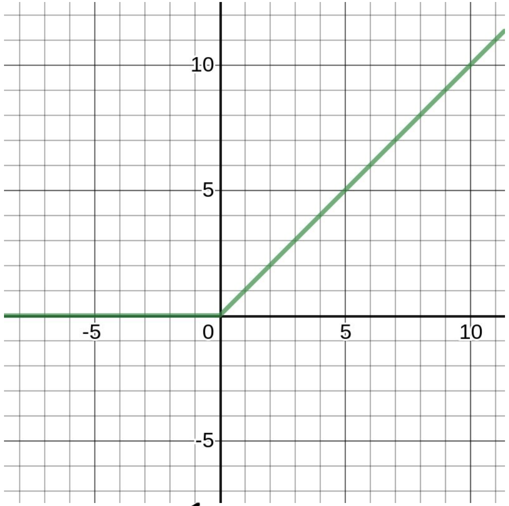
- tanh(z) works better if it's not the last (output) layer
    - Average value is 0 instead of 0.5 – helps w/ learning better weights




$$y = \tanh(x)$$

# ReLU is the choice activation for neural networks

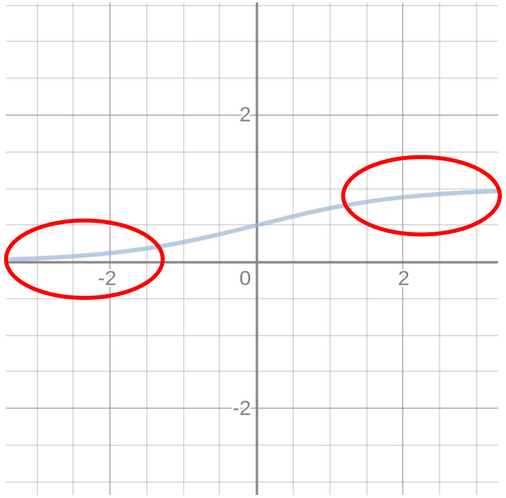
- Rectified exponential linear unit

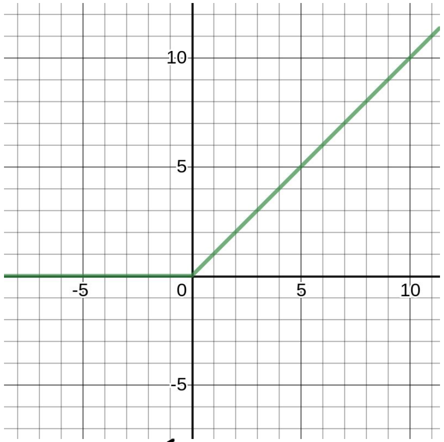



$$y = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

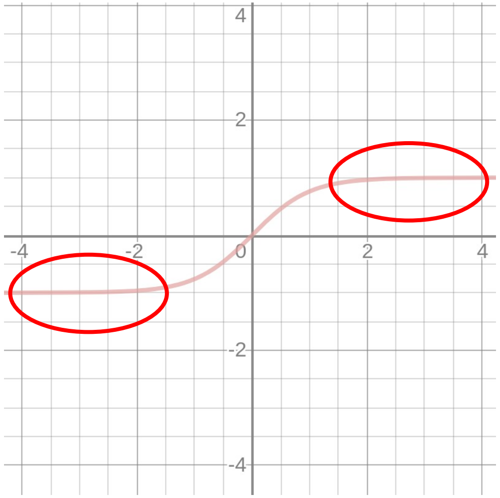
# ReLU is the choice activation for neural networks

- Rectified exponential linear unit
- Reason: sigmoid and tanh have “vanishing gradients”



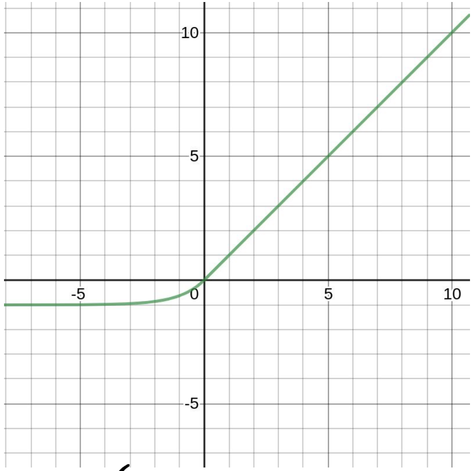

$$y = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$



# ReLU has variants that change x < 0

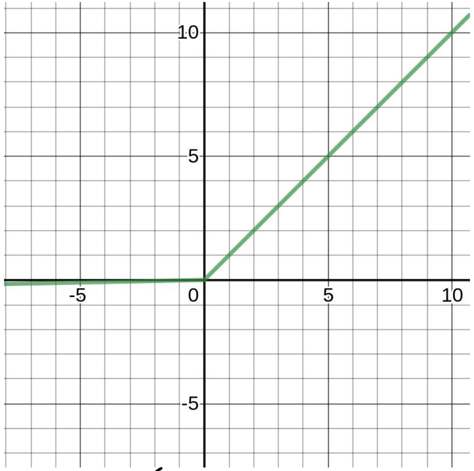
- Otherwise, some nodes start to “die” and become useless
- Variants have higher computational (time) cost

eLU




$$y = \begin{cases} a(e^x - 1) & x \leq 0 \\ x & x > 0 \end{cases}$$

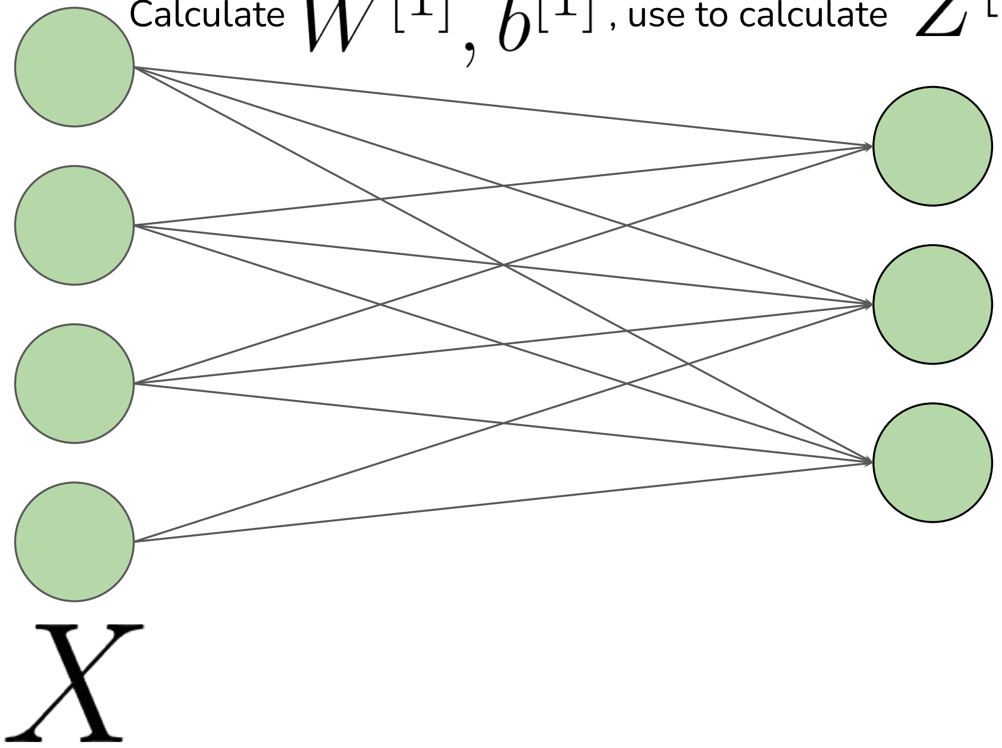
Leaky ReLU




$$y = \begin{cases} 0.01x & x \leq 0 \\ x & x > 0 \end{cases}$$

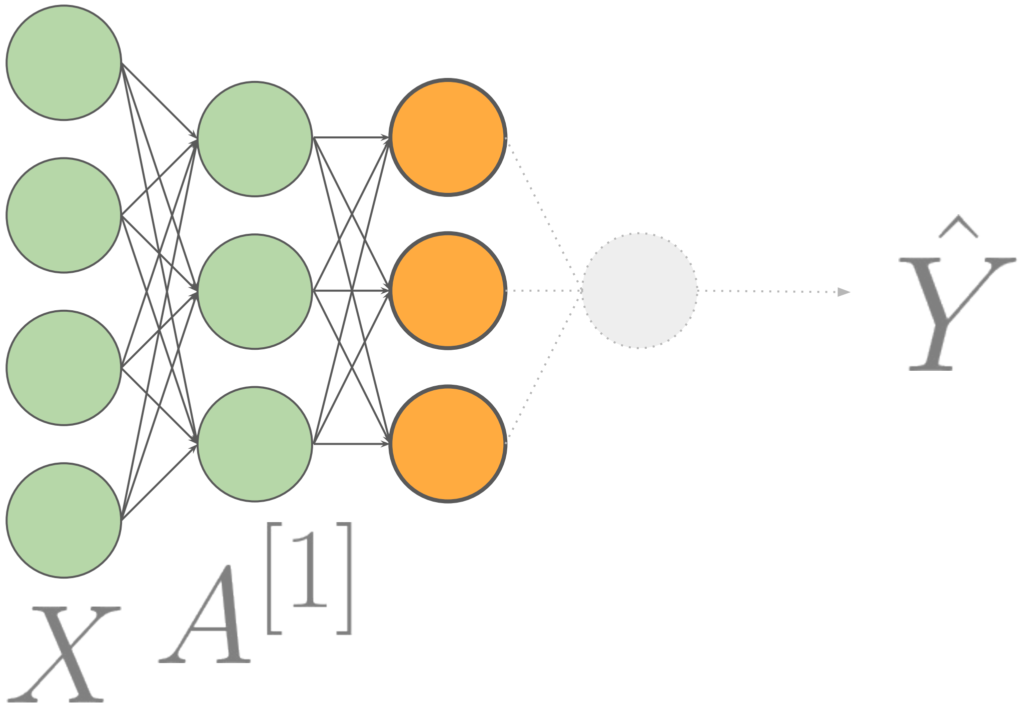
# This is what calculating $\mathcal{Z}$, $A$ for one layer looks like

Calculate $W^{[1]}, b^{[1]}$, use to calculate $Z^{[1]}$



Pass $Z^{[1]}$ through activation function to get $A^{[1]}$

# Forward propagation simply continues for all layers



# $Z^{[l]}$ takes in $A^{[l-1]}$

- $Z^{[1]}$ is the only exception
  - It takes X

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

# $Z^{[l]}$ takes in $A^{[l-1]}$

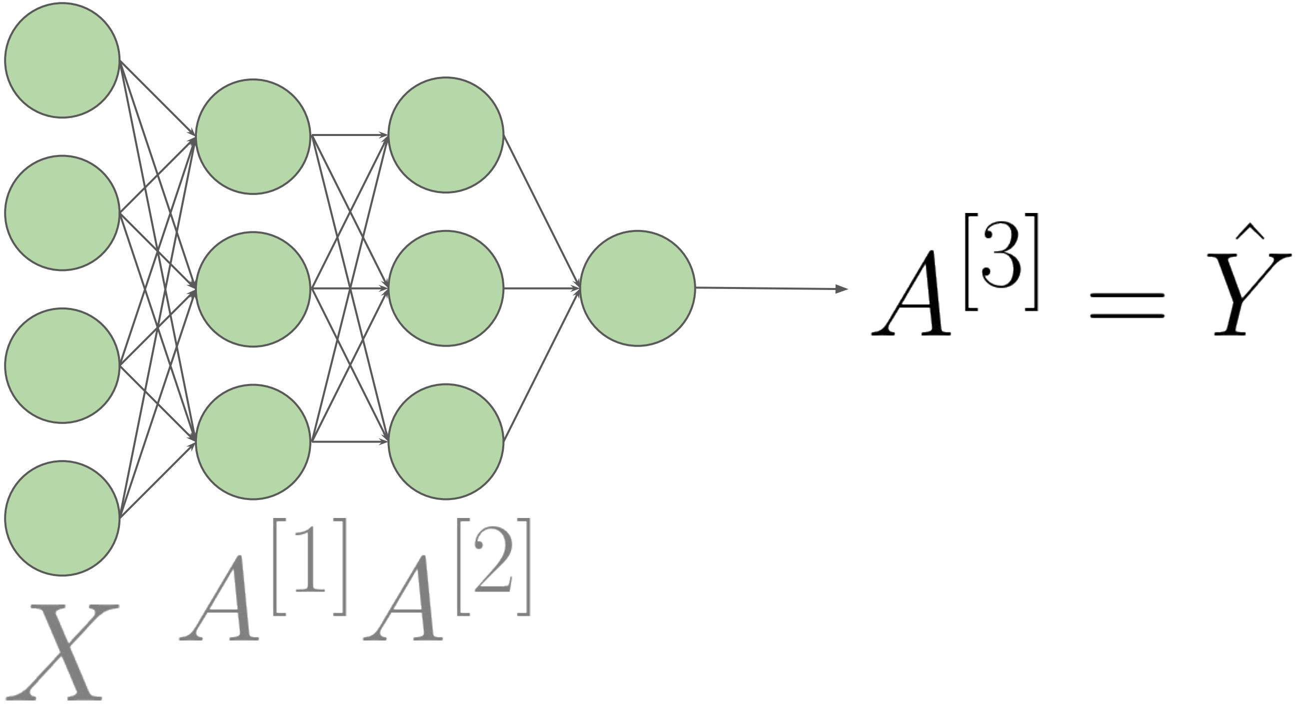
- $Z^{[1]}$ is the only exception
  - It takes X

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

- Activations are calculated the same way
  - You can use different activation functions in different layers
  - The only change between layers is which variables you use

# Forward propagation simply continues for all layers

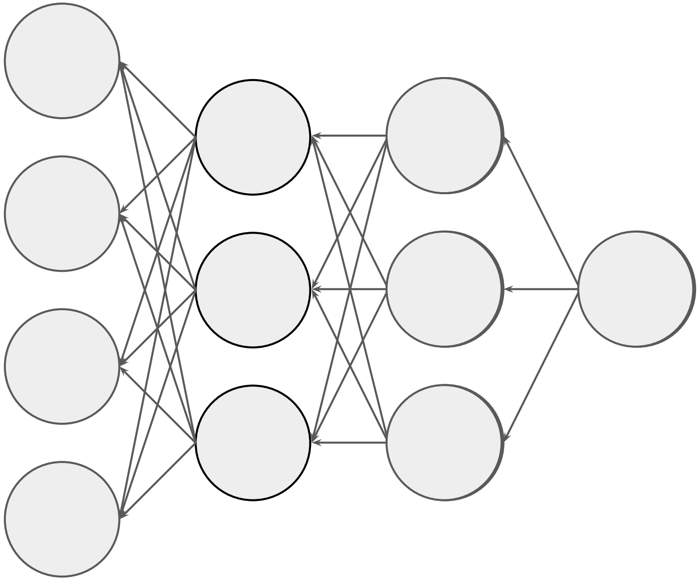
- And so on, until we get to the last layer which outputs y_hat



# Questions?

# Backprop becomes more nuanced

- We want to update all weights and biases in all layers



# Again, we'll stick to intuition over formal proofs

- Backprop is usually the most difficult part of any given model
  - We'll give some links to more detailed proofs!
- We still want to calculate gradients of $W$, $b$ and update $W$, $b$ with $dW$, $db$

# Again, we'll stick to intuition over formal proofs

- Backprop is usually the most difficult part of any given model
  - We'll give some links to more detailed proofs!
- We still want to calculate gradients of $W$, $b$ and update $W$, $b$ with $dW$, $db$

- First question: what changes from our cost function?

# Cost function doesn’t change in this context

- We can use different cost functions depending on what task our neural network is doing
  - Our cost function from last time works well for classification tasks
- Changing the cost function will change formula for dZ
  - The rest is unchanged!

Same cost function from previous lesson

$$L(\hat{y}, y) = \begin{cases} y = 1 & -\log(\hat{y}) \\ y = 0 & -\log(1 - \hat{y}) \end{cases}$$

# The formula for dZ changes to this

- Neural network version
  - Asterisk operator is element-wise multiplication
  - $g^{[l]\prime}$ indicates derivative of activation function *g*

$$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$$

- Logistic regression version
  - Recall that we only had one layer and one vector z

$$dz = a - y$$

# dZ depends on derivative of the activation function

- How much Z changes should depend on the activation value for Z
- ReLU derivative:

$$g'(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

- Sigmoid derivative:

$$g'(z) = g(z)(1 - g(z))$$

- These are the ones we’ll be using in notebook

# What formula do we use for dA?

- For the last layer, we can take derivative of

$$L(\hat{y}, y) = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

  with respect to y_hat

- We will get the following (division is element-wise)

$$dA^{[L]} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}} = \frac{1-y}{1-A^{[L]}} - \frac{y}{A^{[L]}}$$

# Why wouldn't this work for $l \neq L$?

# Why wouldn't this work for $l \neq L$?

- Cost function not in direct terms of $A^{[l]}$ for $l \neq L$
    - Chain rule

- For earlier layers, we use the equation on the right

$$dA^{[l-1]} = W^{[l]T} dZ^{[l]}$$

# dW

- Note $dZ^{[l]} \times A^{[l-1]T}$ has dimensions of $W^{[l]}$

$$dW^{[l]} = \frac{1}{m} dZ^{[l]} A^{[l-1]T}$$

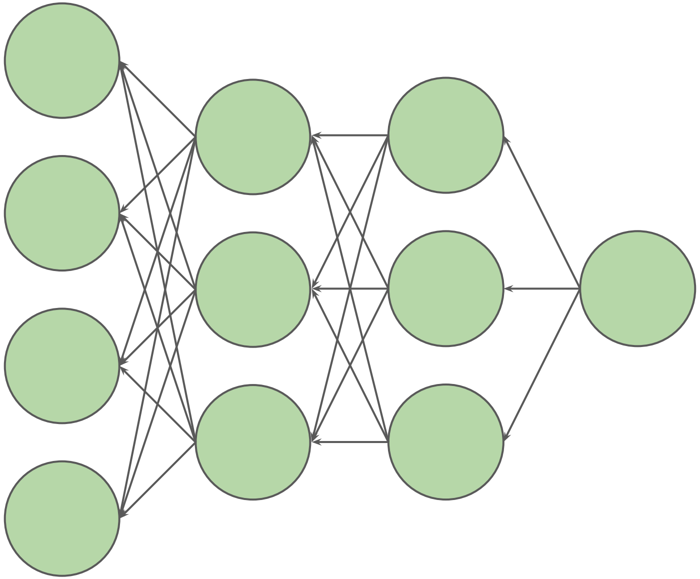
# db

- The notation is uglier than the math – just average each row of dZ

$$db_j^{[l]} = \frac{1}{m} \sum_{i=1}^{m} dZ_{ji}^{[l]}$$

# Do this for all layers and that's one backprop cycle

- Like with logistic regression, repeat forwardprop and backprop for many iterations



# Review

- Variables change from logistic regression
  - W, b, Z, A exist for every single layer except input layer
  - W is $(n^{[l]}, n^{[l-1]})$
  - b is $(n^{[l]}, 1)$
  - Z and A are $(n^{[l]}, m)$
- Forward propagation is very similar to logistic regression
  - Must be done on every single layer
- Backpropagation uses new equations

# Questions?