Statistics and Artificial Intelligence

Lecture 10: Shallow Neural Networks

Yixin Wang

JiTTs and Some Important Changes

- JiTTs are due too often.
  - JiTTs now only due on Tue. We still drop 4 of these. Graded by completion.
  - The new JiTTs will contain practice quiz problems, and we will go through these in Tue class.
- How to prepare for quizzes?
  - Quiz will be similar to JiTTs. (We will go through all the quizzes and discuss the practice quiz on Tuesday)
- Lectures are about theory and homework is about coding. What's going on?
  - We'll do more coding demo.
  - Please consider the (new) JiTTs as the theory component of the course, and homework as the coding component.
- The different components of the class feels disconnected.
  - We attempted to improve as above. Please continue to give us feedback (via JiTTs) on how to do things better.

Goals for Today

- How is it connected to the implementation in homework? A quick demo
- What's going on under the hood
  - Forward pass in a two-layer neural network
  - Backward pass in a two-layer neural network

Neural Network Demo

- https://colab.research.google.com/drive/1bHmtSdDQ_zJeLJhNtoGmT2bhPbDa_Iyp?usp=sharing
- We will discuss what's going on under the hood in class.
- You will also have the opportunity to peak into and implement what's under the hood in the next two homework.

NOTES FROM DEMO

Need to specify input dimensions on 1st hidden layer

params each layer

3 inputs * 3 weights + 3 biases = 12 parameters

$$53 \cdot 25 + 25$$

$$\underbrace{\text{inputs} \cdot \text{outputs}}_{\text{weights}} + \underbrace{\#\ \text{outputs}}_{\text{Biases}}$$

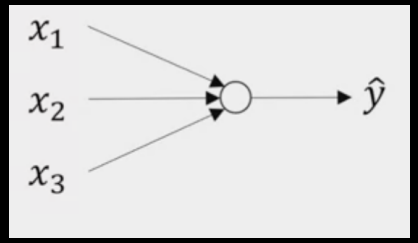
Logistic regression is a special case of a neural network

Early stopping - end gradient descent before it converges

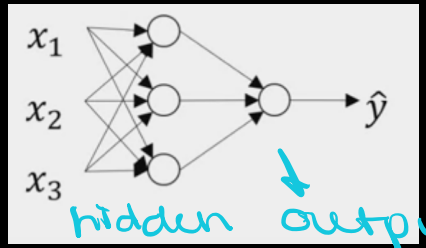
What is a Neural Network?

can have multiple output layers
↳ ie multiclassification problems

- Logistic regression



- Neural networks

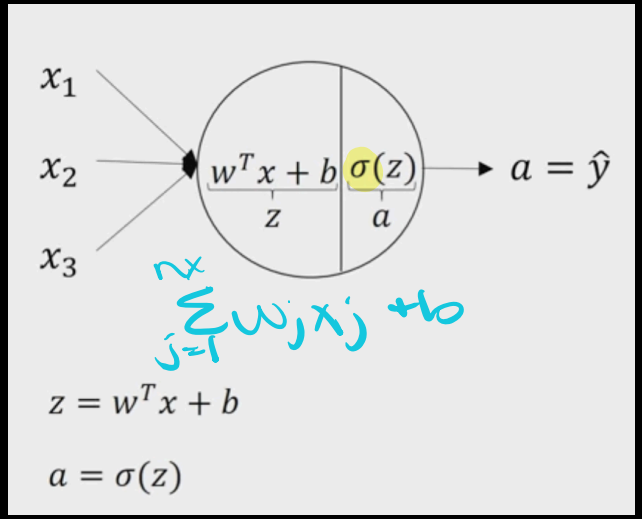


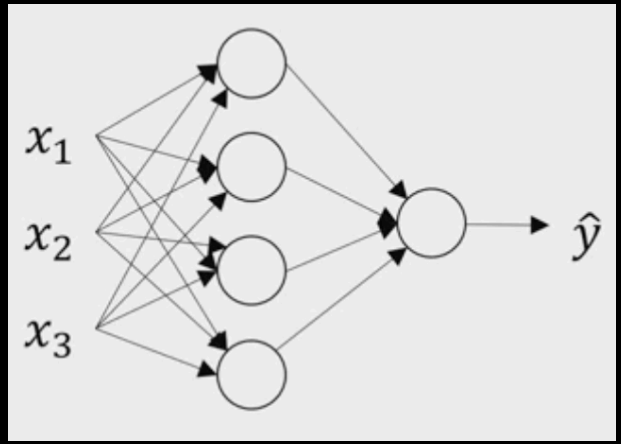
3 input
→ hidden layer has 3 neurons

Computing a Neural Network's Output

repeat computation in every neuron + stack together

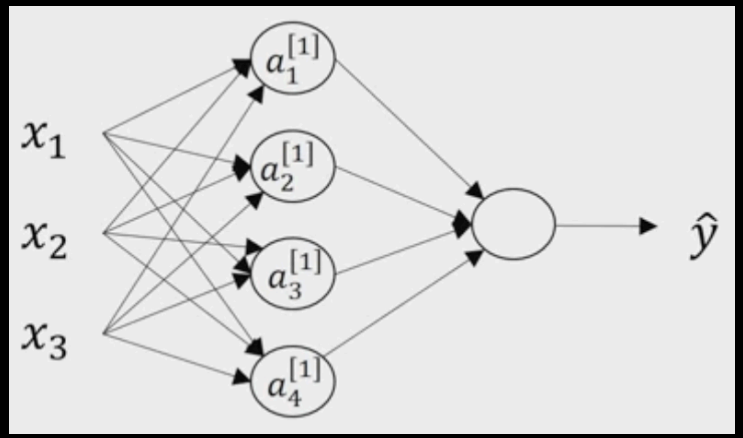
- Logistic Regression vs Neural Network





$z^{[1]} = w^{[1]} x + b^{[1]} \Rightarrow a^{[1]} = \sigma(z^{[1]})$

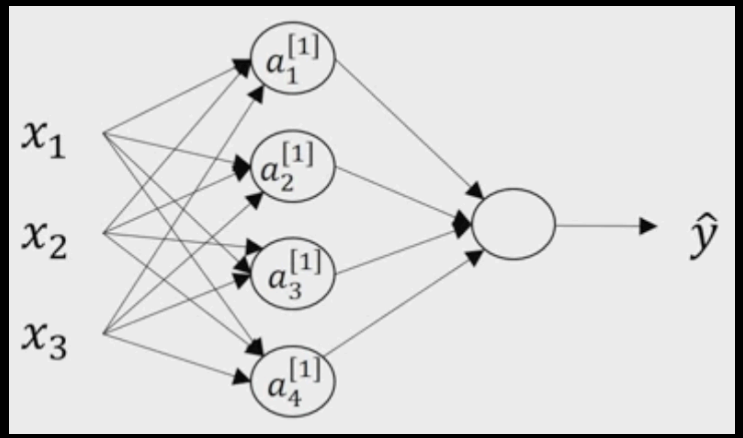
Computing a Neural Network's Output



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \; a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \; a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \; a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \; a_4^{[1]} = \sigma(z_4^{[1]})$$

$z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$ $\quad w^{[1]} = \begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}$ $\quad b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$

Computing a Neural Network's Output



0 input $a^{[0]}$

1 hidden $a^{[1]}$

2 output $a^{(2)}$

⇒ 2 layer neural net

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$

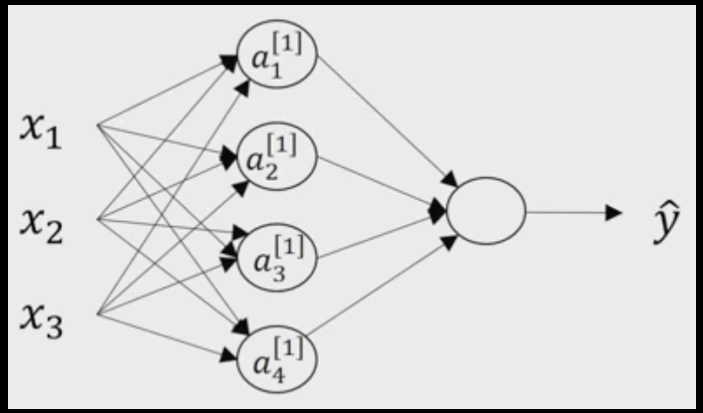
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

Neural Network Representation

$a^{[2](i)}$ → example i, layer 2



repeat for every point

X

$x^{(1)}$
$x^{(2)}$
$\vdots$
$x^{(n)}$

$\rightarrow a^{[2](1)} = \hat{y}^{(1)}$

$a^{[2](2)} = \hat{y}^{(2)}$

$\vdots$

$a^{[2](m)} = \hat{y}^{(m)}$

Given input x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

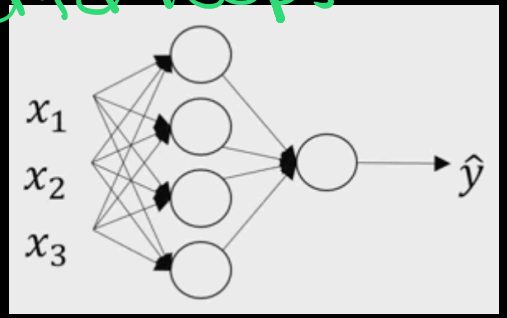
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

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

can vectorize operation

Vectorizing across multiple examples

organize into matrices to avoid for loops



$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

Do operation for all data points all at once

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

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

1st a' neuron, 1st datapoint

Vectorizing across multiple examples

```
for i = 1 to m:
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$

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

Justification for vectorized implementation

$z^{[1](1)} = w^{[1]} x^{(1)} + b^{[1]}$

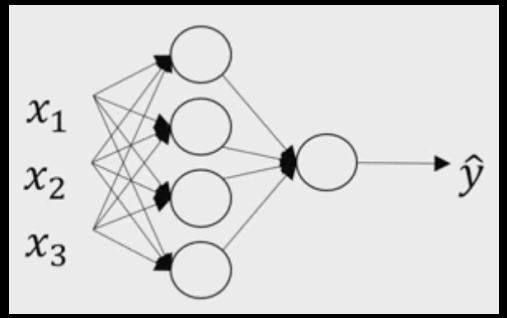
$z^{[1](2)} = w^{[1]} x^{(2)} + b^{[1]}$

$z^{[1](3)} = w^{[1]} x^{(3)} + b^{[1]}$

Datapoints are changing but weights are same

↳ organize into matrix

Recap of vectorizing across multiple examples




```
for i = 1 to m:
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

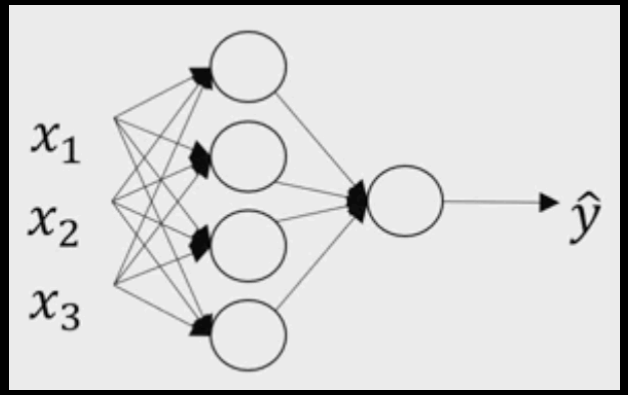
$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

Activation functions

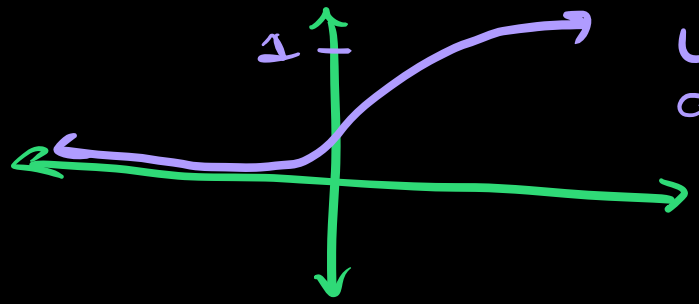
Logistic regression-
always have to
return #s between 0 + 1
-always used sigmoid



- now can work w/ more general activation functions

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

Sigmoid

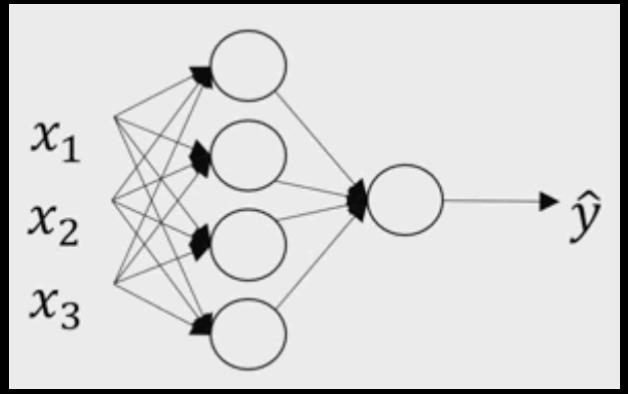


Small # go to 0

Large # go to 1

$$a = \frac{1}{1+e^{-z}}$$

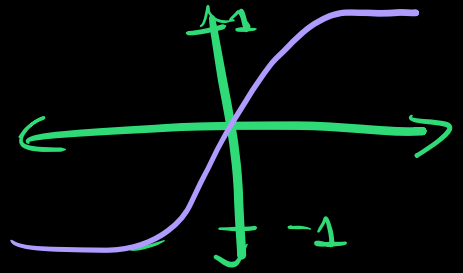
Activation functions



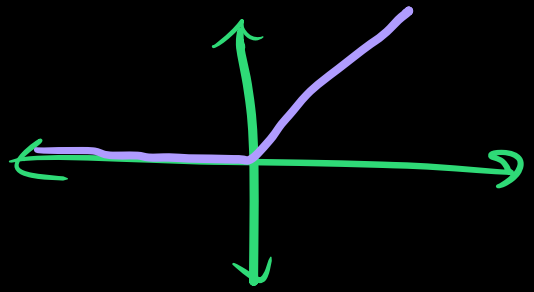
$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

tanh

$$a = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



Relu



$$a = \max(0, z)$$

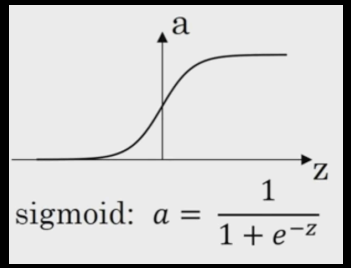
Relu = rectified linear unit

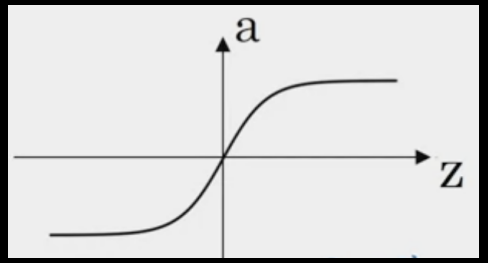
Pros and Cons of Activation Functions

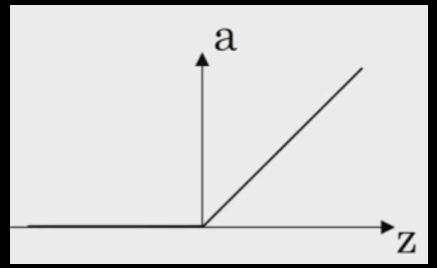
everything is non-negative so it really limits the neuron

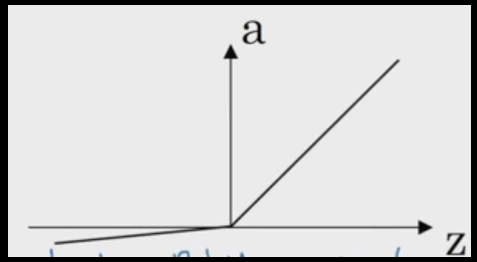
can be hard to set right α for leaky ReLU

- Almost never use sigmoid except in the output layer
- Tanh is almost always better map (-1, 1)
- Default to use: ReLU typically default to ReLU or Leaky ReLU
- Feel free to try leaky ReLU too.









$a = \max(\alpha z, z)$
Leaky ReLU

Why do we need nonlinear activation function?

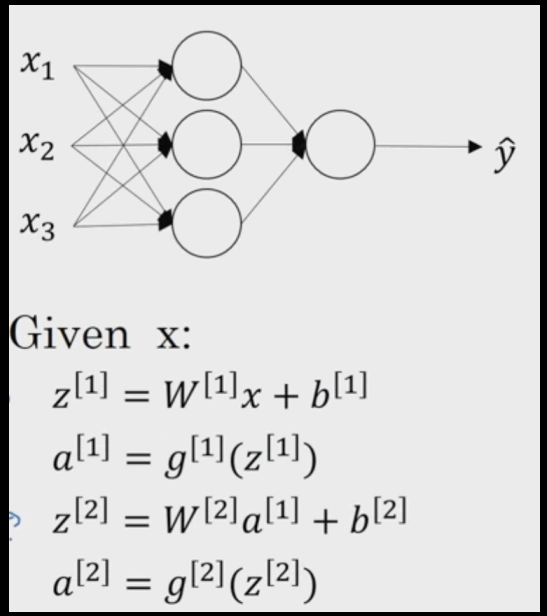
suppose linear Activation

$a^{[1]} = z^{[1]} = w^{[1]}x + b^{[1]}$

$a^{[2]} = z^{[2]} = w^{[2]}x + b^{[2]}$

$a^{[2]} = w^{[2]}(\underbrace{w^{[1]}x + b^{[1]}}_{a^{[1]}}) + b^{[2]}$

$(\underbrace{w^{[2]}w^{[1]}}_{w1})x + (\underbrace{w^{[2]}b^{[1]} + b^{[2]}}_{b1})$



Like we don't have hidden layers between input + output

Gradient descent for neural networks

Go from loss function back to parameters to optimize loss function

Params $w^{[1]}$, $b^{[1]}$, $w^{[2]}$, $b^{[2]}$

$(n^{[1]}, n^{[0]})$ $(n^{[1]}, 1)$ $(n^{[2]}, n^{[1]})$ $(n^{[2]}, 1)$

$n^{[1]}$: neurons layer 1; $n^{[0]}$: neurons layer 0

Cost Func: $J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}, y)$

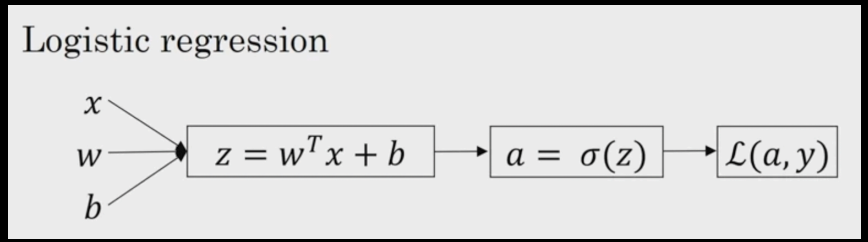
$dw^{[i]} = \frac{\partial J}{\partial w^{[i]}}$ $db^{[i]} = \frac{\partial J}{\partial b^{[i]}}$

$w^{[i]} \leftarrow w^{[i]} - \alpha dw^{[i]}$ $b^{[i]} \leftarrow b^{[i]} - \alpha db^{[i]}$

The gradients take complicated forms…

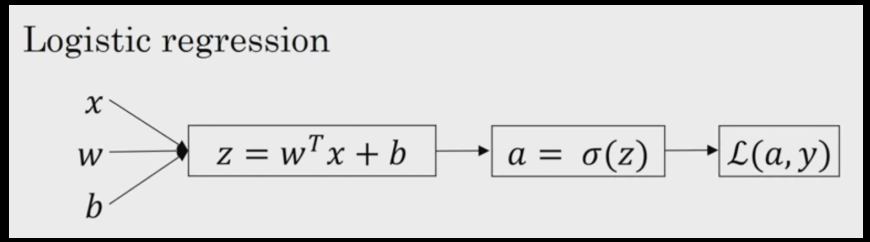
Computational graph (a key construct underlying Tensorflow) is here to help

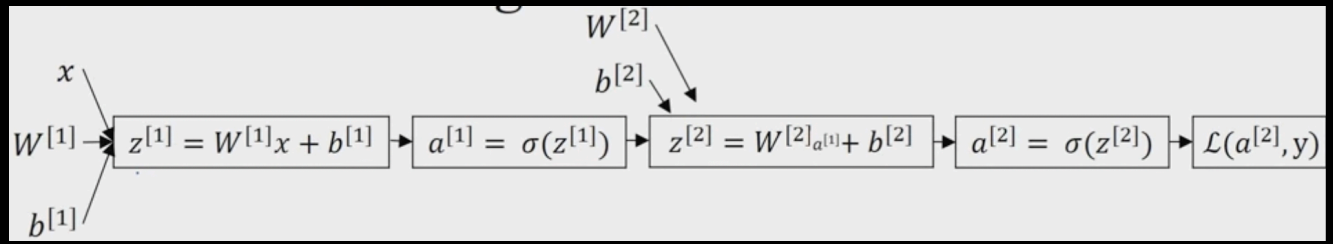
- Computational graph holds computational operations;
- They treat computation as data, so we can derive the gradients of these computation (all composed together) automatically.
- It is also known as backpropagation.



Neural network gradients

- Logistic regression vs neural network

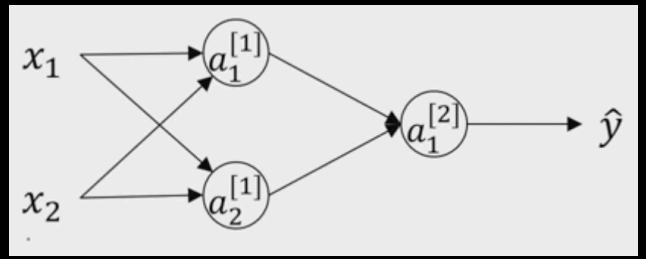




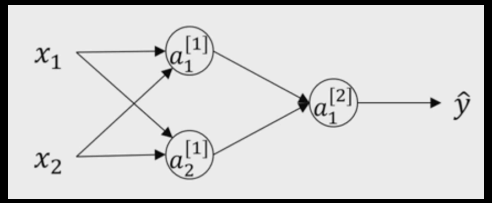
Random Initialization

What happens if you initialize weights to zero

- All hidden units will be computing the same function due to symmetry.
- There will be no point to have multiple hidden units.



How do we perform random initialization then?



- Numpy vs Tensorflow:
  - Tensorflow tensors are not assignable. Only the tensorflow variables are assignable.
  - Numpy arrays are always assignable.

The materials in this course are adapted from materials created by Alexander Amini, Alfredo Canziani, Justin Johnson, Andrew Ng, Bhiksha Raj, Grant Sanderson and the 3blue1brown channel, Rita Singh, Ava Soleimany, and Ambuj Tewari.