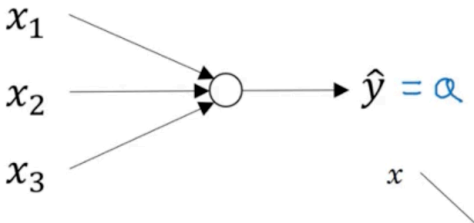
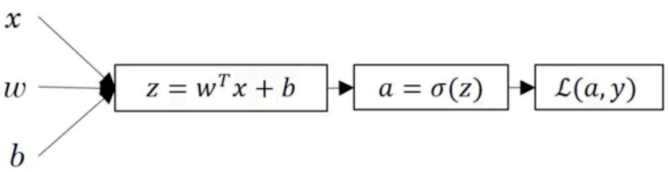
Overview of Neural Networks

What is a Neural Network?

What is a Neural Network?

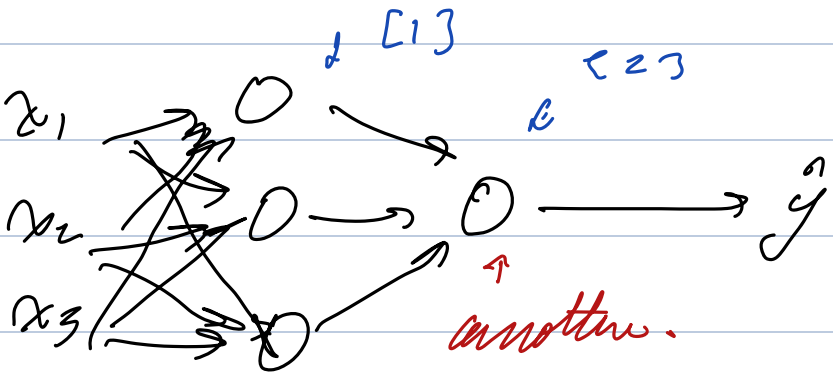


what we did in wk2



Features (inputs) x, Parameters: w and b

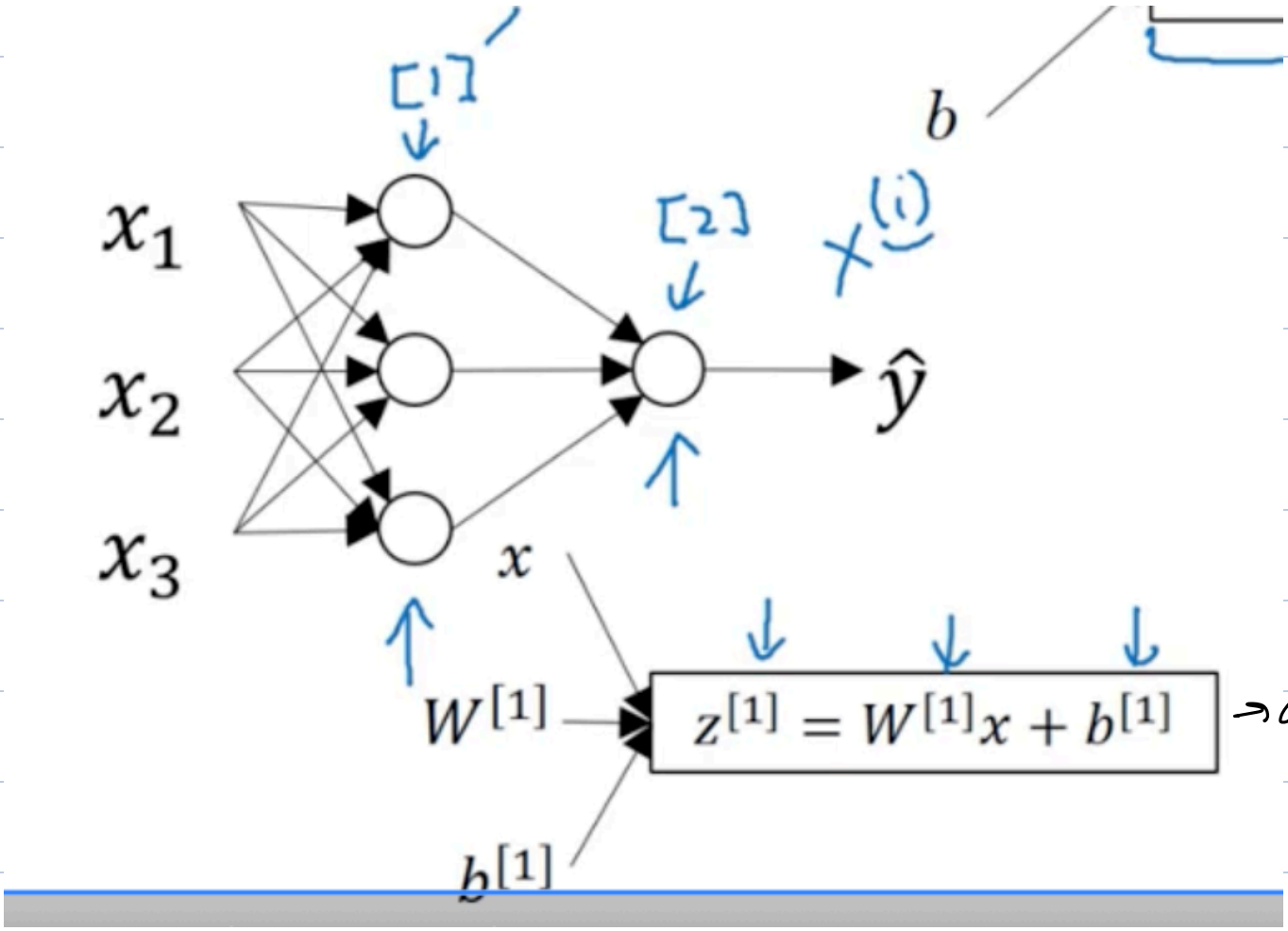
Neural Network:



"z" like calculation and "a" like calculation

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

$b^{[1]}$ →

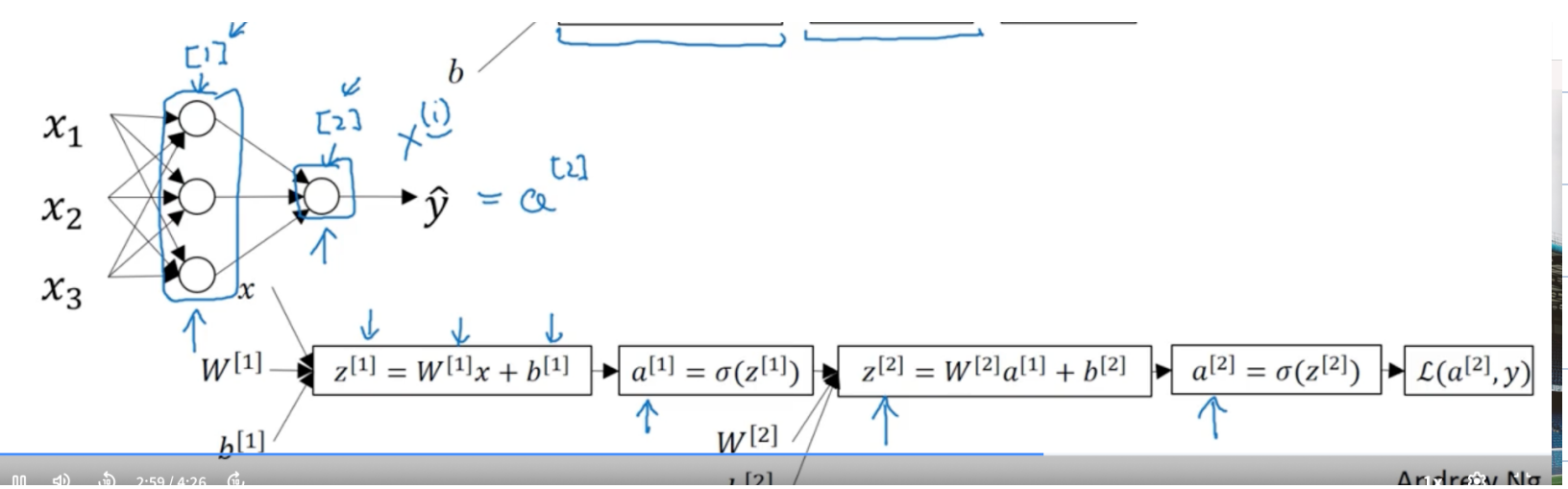


[1] → quantities associated to layer 1

[2] → layer 2

(i) → individual training example. $x^{(i)}$

$W^{[1]}$ → layer 1.



In logistic regression ⇒ 2 calculations ⇒ a couple

In neural networks it's a lot of these repeat calculations

Also lots of backwards calculations here as well for the gradient

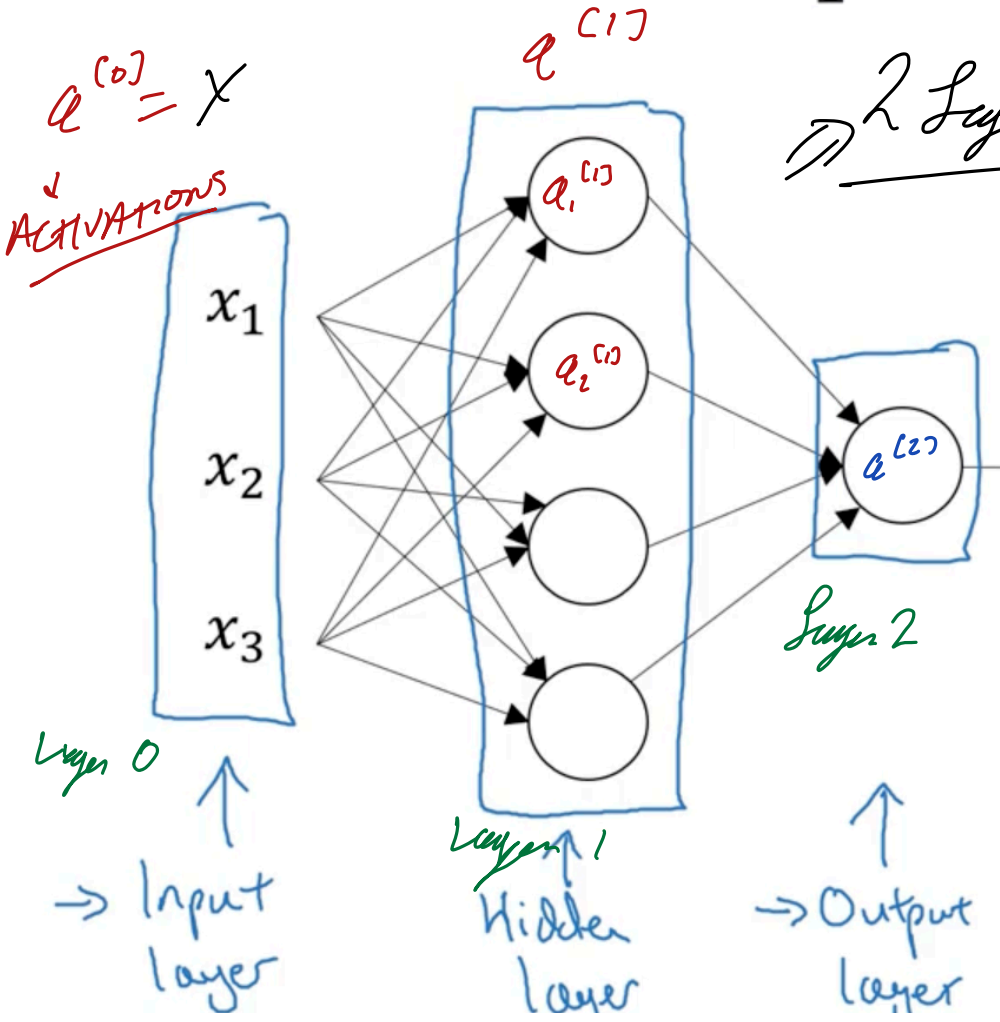
$dw^{[2]}$ $da^{[2]}$

Think of logistic regression model and repeating it one or more times

---

Neural Network Representations and what they mean.

SINGLE HIDDEN LAYER NN



2 Layer NN (DON'T COUNT THE INPUT LAYER.)

"$\hat{y} = a$"
y but we only need one layer (output)

Supervised learning: has x's and target output $\hat{y}$.
So hidden layer, we don't know what they should be in the training set.

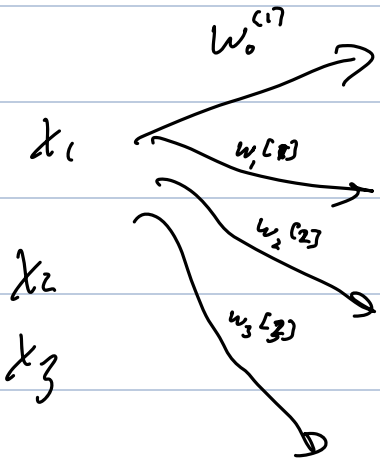
INPUT LAYER passes on X to hidden layer as the ACTIVATIONS

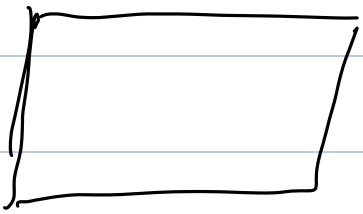
$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}$$

Hidden layer will have $w^{[1]}, b^{[1]}$ associated with each layer.

4 nodes / 4 hidden units

$w^{[1]} = \langle 4, 3 \rangle$ ← 3 input features





$b^{[1]} = \langle 4, 1 \rangle$.

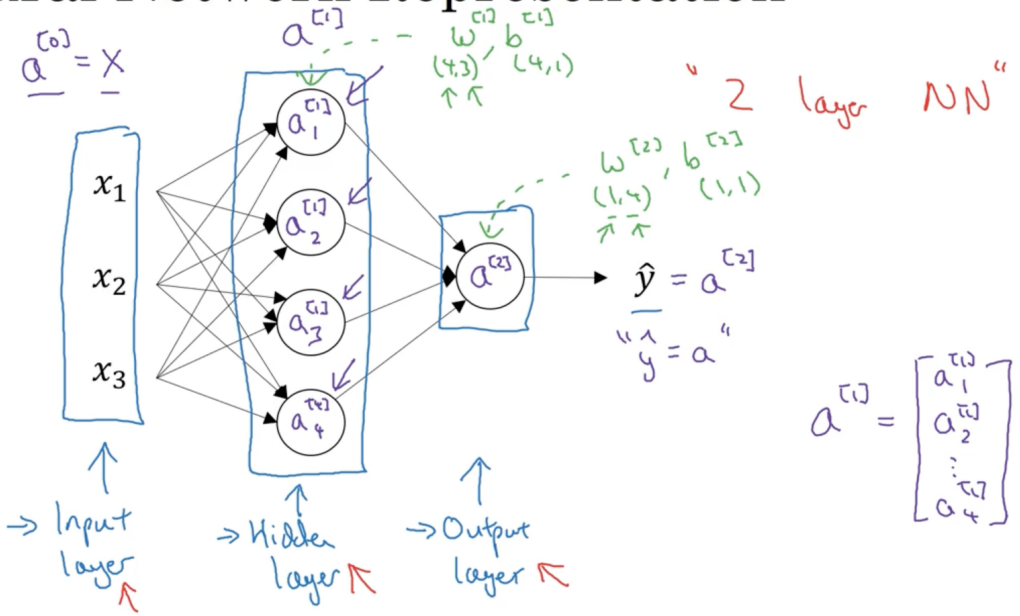
→ connected to the specific layer.

$w^{[2]}, b^{[2]} \Rightarrow w^{[2]} \langle 1, 4 \rangle$ ← 2 hidden layers ← 4 layers

output ← 1

$b^{[2]} \Rightarrow \langle 1, 1 \rangle$.

Neural Network Representation

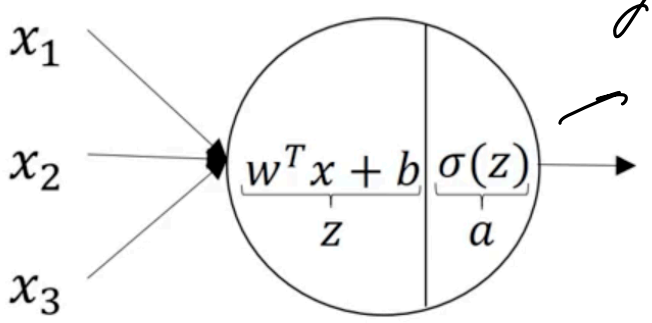


$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}$$

Andrew Ng

Computing A Neural Network's Output

In Log. Reg.



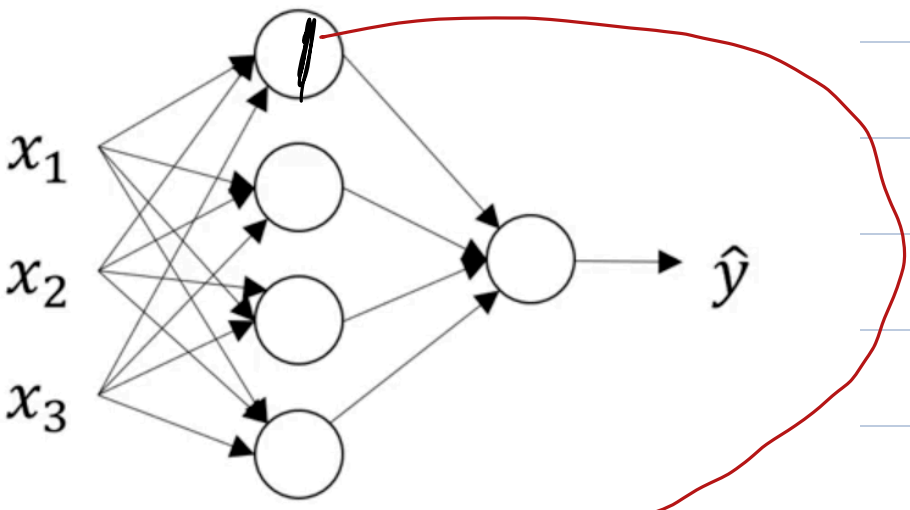
→ two computations.

→ first and then the ACTIVATION

$$z = w^T x + b$$

$$a = \sigma(z)$$

∴ A neural network just does this a bunch of times.



Zoom into one node of hidden layer

Like Logistic Regression, does 2 steps of computation

first layer

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

↓ first node

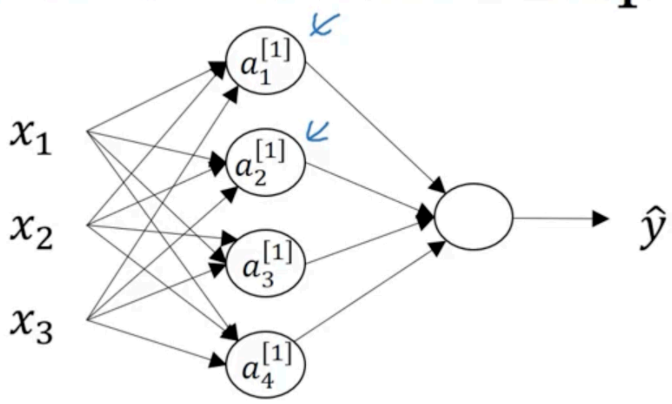
$a_i^{[l]}$ → layer; i → node in layer.

2nd node in hidden layer.

$$z_2^{[1]} = w_2^{[1]} x^T + b_2^{[1]}$$

$$a_2^{[1]} = \sigma(z_2^{[1]})$$

Neural Network Representation



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

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

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

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

Vectorize the equations.

W was : <3, 4>

now <4, 3>

$W^{[1]}$

↑ 4x3.

$b^{[1]}$

$$
4\left\{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}\right. \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T}x + b \\ w_2^{[1]T}x + b \\ w_3^{[1]T}x + b \\ w_4^{[1]T}x + b \end{bmatrix}
$$

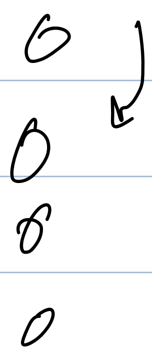
(matrix: 4 × 3; x: 3×1; b: 4×1; result: 4×1)

→ for each hidden node

$\langle 4, 3 \rangle$ → each of these nodes are affected by previous activations differently.

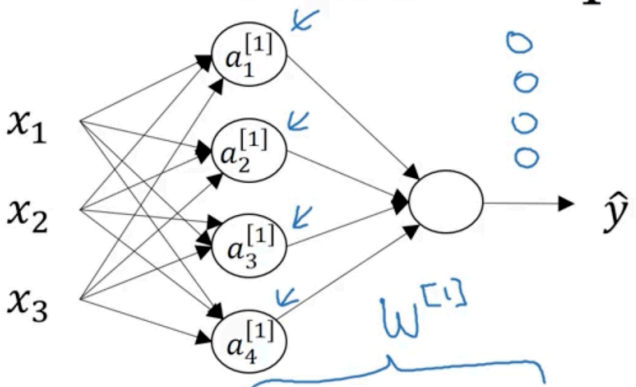
$$
Z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}
$$

apply $\sigma(z)$ to get ACTIVATIONS



nodes stack vertically & get $z^{[1]}$.

Neural Network Representation



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

$$z^{[1]} = \begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

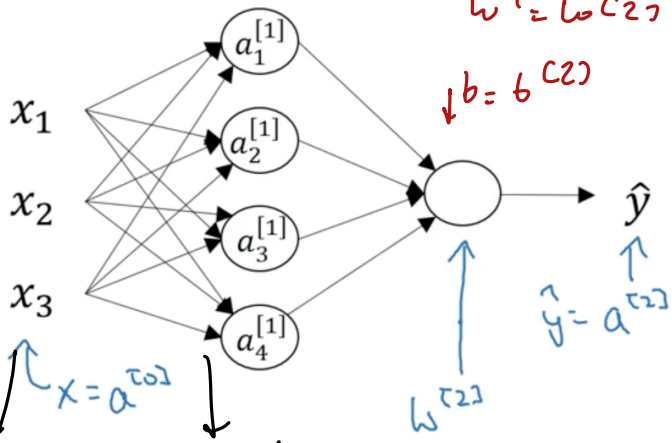
$W^{[1]}$ (4, 3) $b^{[1]}$ (4, 1)

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

sigmoid element wise

Andrew Ng

Neural Network Representation learning



input activation

weights on the edges.

Given input x:

$$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]} \quad (4,1)\ (4,3)\ (3,1)\ (4,1)$$
$$a^{[1]} = \sigma(z^{[1]}) \quad (4,1)\ (4,1)$$
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]} \quad (1,1)\ (1,4)\ (4,1)\ (1,1)$$
$$a^{[2]} = \sigma(z^{[2]}) \quad (1,1)\ (1,1)$$

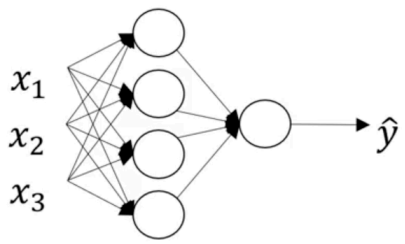
vectorized

Andrew Ng

Really just a bunch of logistic regression units with different activations and weights.

Vectorize Across Multiple Examples

Vectorizing across multiple examples



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

→ for $i = 1$ to m

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

⋮

$x \longrightarrow a^{[2]} = \hat{y}$.

input feature vector x

m examples.

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

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

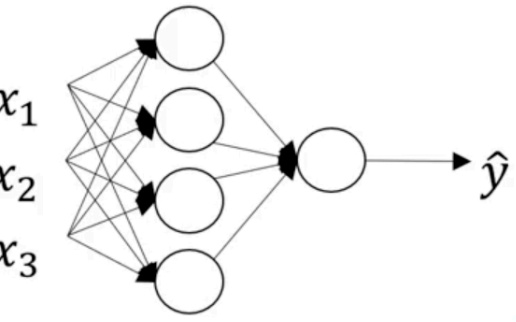
⋮

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

Y

$a^{[2](i)}$ ← example i

↑ layer 2



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

$x \rightarrow a^{[2]} = \hat{y}$

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

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

$\vdots$

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

$a^{[2](i)}$: example i, layer 2

for $i = 1$ to m,

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

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

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

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

Andrew

I want to vectorize all this

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

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

recall

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

(n_x, m) $\rightarrow$ $X = A^{[0]}$

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

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

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

neurons ↓ — vertically across different nodes of activations / neurons

horizontally across different examples →

← → TRAINING EXAMPLES

↕ HIDDEN LAYER UNITS OR NEURONS

JUSTIFICATION OF THE VECTORIZED IMPLEMENTATION

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

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

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

$w^{[1]} = \begin{bmatrix} — \\ — \\ — \end{bmatrix}$

$w^{[1]} x^{(1)} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}$

$w^{[1]} x^{(2)} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}$

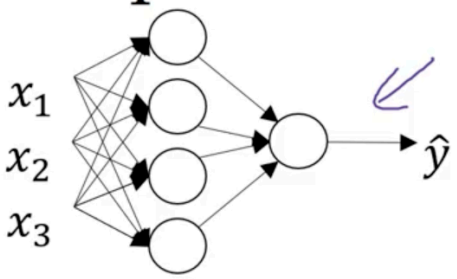
$w^{[1]} x^{(3)} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}$

$$w^{[1]} \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \\ | & | & | \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \\ | & | & | \end{bmatrix} = Z^{[1]}$$

X

$+b^{[1]} \quad +b^{[1]} \quad +b^{[1]}$

Recap of vectorizing across multiple examples



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

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

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

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

$A^{[0]}$

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

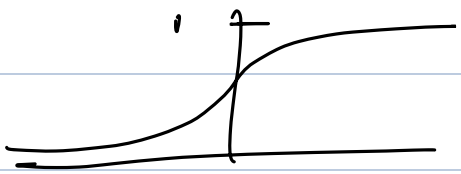
Vectorized

$Z^{[1]} = W^{[1]}A^{[0]} + b^{[1]}$ → so the diff layers are basically doing the same thing to do.

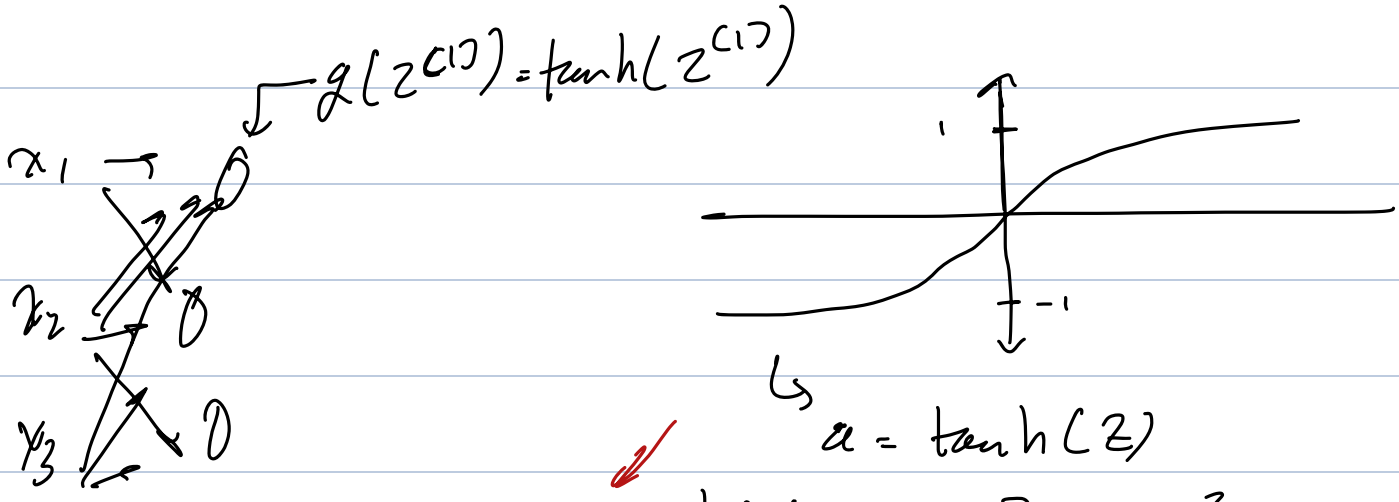
ACTIVATION FUNCTIONS → for hidden and output layers.

Better than Sigmoid

$A^{[1]} = \sigma(Z^{[1]})$



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



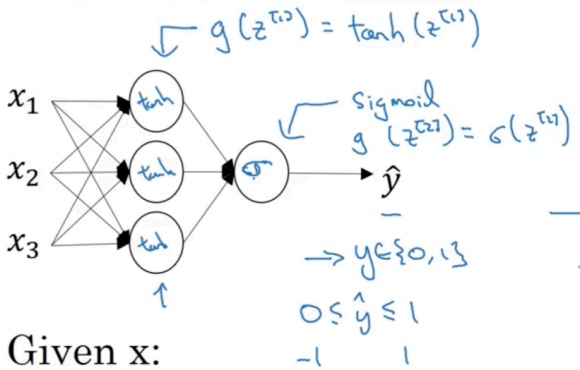
$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Mean to prev of 0 which is what we do to standardize to have 0 mean of effect.

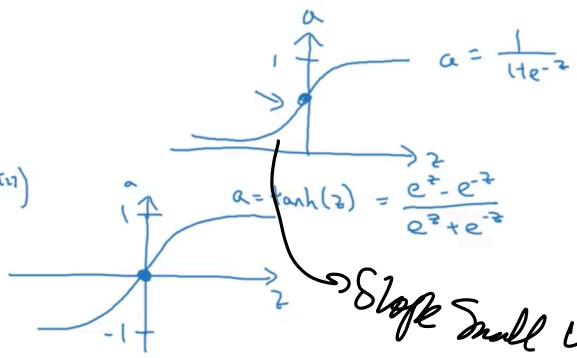
Sigmoid could be better for binary classification where $y \in \{0, 1\}$. FOR THE OUTPUT LAYER

$0 \leq \hat{y} \leq 1$

Activation functions



Given x:

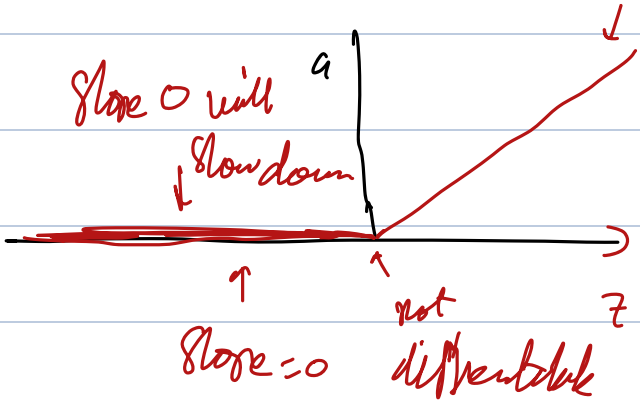


Slope small when z is large or small, So slows down gradient descent

Activation fun can be diff at diff layers.

Really popular.

ReLU For Rectified Lein Unit

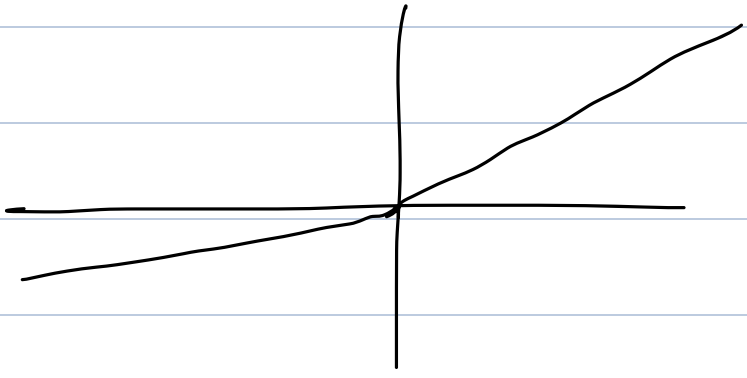


$a = \max(0, z)$.

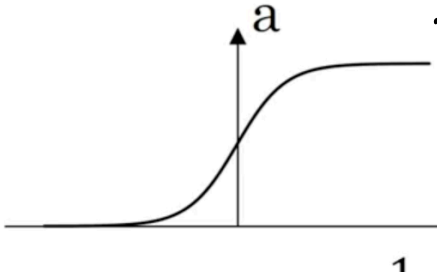
but can take eg 0 or 1 at $z = 1$.

if Binary classification → Sigmoid,
Else use ReLU is a safe choice for hidden layer.

Leaky ReLU also works better



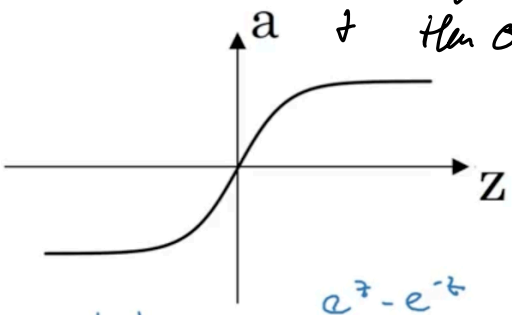
Pros and cons of activation functions



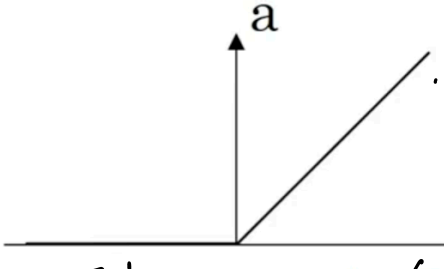
→ Never use EXCEPT for binary (0/1) to do binary classif. (when?)

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

strictly superior a ∈ (-1, 1)

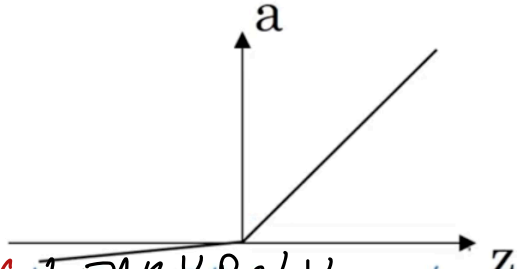


tanh: $a = \frac{e^z - e^{-z}}{e^z + e^{-z}}$



DEFAULT. Not sure, use this

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



Leaky ReLU

↳ $a = \max(0.01z, z)$

could do.

0.01

Maybe say this is also a parameter.

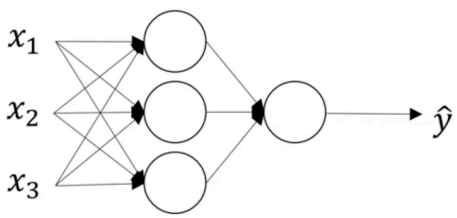
WHY DOES NEURAL NETWORK NEED ACTIVATION FCN

---

why not just use the ReLU

WHY DOES IT NEED A NON-LINEAR TRANSFER FCN

Activation function



why not do this?

Given x:

→ $z^{[1]} = W^{[1]}x + b^{[1]}$

→ $a^{[1]} = \cancel{g^{[1]}(z^{[1]})}\ z^{[1]}$ $\quad g(z) = z$ "linear activation function" identity?

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

→ $a^{[2]} = g^{[2]}(z^{[2]})$

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

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

Well then $a^{[2]} = W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]}$

$a^{[2]} = \underbrace{W^{[2]}W^{[1]}}_{W'}x + \underbrace{W^{[2]}b^{[1]} + b^{[2]}}_{b'}$

$= W'x + b'$

& ∴ no matter how many hidden layers, it's essentially one big linear fcn

since composition of 2 lin fcn = lin fcn.

So you need non-lin fcn.

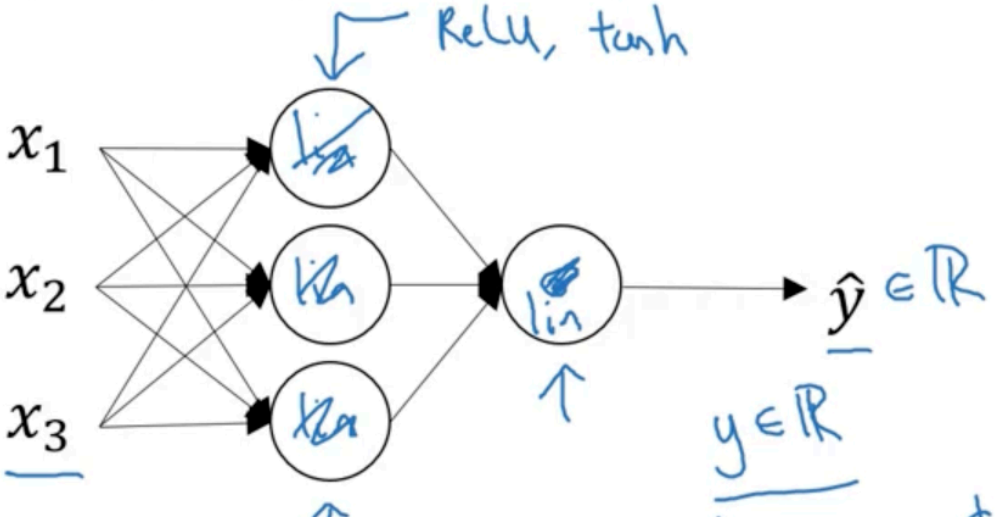
where would you use lin fcn?

Ex: a REGRESSION PROBLEM, where y is a real #

$y \in \mathbb{R}$ → house price $0 \to \$1,000,000$.

∴ it might be possible to have lin act the output layer

Activation function



So lin fcn activation fcn is not that good.

Derivatives of Activation Fons
for Gradient Descent

Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$ $\quad g'(z) = g(z)(1-g(z))$

$g(z) = a$. $\quad g'(z) = a(1-a)$

TANH $g(z) = \tanh(z)$

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

$g'(z) = 1 - (\tanh(z))^2$

$g'(z) = 1 - (g(z))^2$.

$g'(z) = 1 - a^2$

$g(z) = a$.

RELU

$g(z) = \max(0, z)$ $\quad g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \\ \text{undef.} - \text{if } z = 0 - \end{cases}$

In programming if z set to 0 or 1

LEAKY RELU

$g(z) = \max(0.01z, z)$ $\quad g'(z) = \begin{cases} 0.01 & z < 0 \end{cases}$

$z \geq 0$

g for $z = 0$, same as ReLU, Pick 0 or 1

Gradient Descent for Neural Network with one Hidden Layer

Parameters: $w^{[1]}$, $b^{[1]}$, $w^{[2]}$, $b^{[2]}$. $n^x = n^{[0]}$, $n^{[1]}$, $n^{[2]} = 1$

$\langle n^{[1]}, n^{[0]} \rangle$ $\langle n^{[1]}, 1 \rangle$ $\langle n^{[2]}, n^{[1]} \rangle$ $\langle n^{[2]}, 1 \rangle$

input features. $n^{[1]}$: # hidden units. $n^{[2]}$: # output units

Cost for training big classifier

$$J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum \mathcal{L}(\hat{y}, y)$$

to get $w^{[2]}$

Gradient Descent: → Initialize parameters Randomly and not to 0

Repeat {

Compute $(\hat{y}^{(i)}, i = 1 \cdots m)$

$dw^{[1]} = \frac{dJ}{dw^{[1]}}$, $db^{[1]} = \frac{dJ}{db^{[1]}}$, ...

$w^{[1]} := w^{[1]} - \alpha\, dw^{[1]}$

FORMULAS FOR COMPUTING DERIVATIVES:

Forward Prop.

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

Backprop

$$dZ^{[2]} = A^{[2]} - Y \qquad Y = [y^{(1)}, y^{(2)} \dots y^{(m)}]$$
$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$
$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis=1, keepdims=True)$$

↳ Sum across axis=1 (across rows) (reduce)

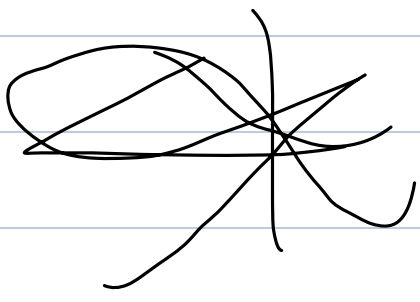
↳ keeps (n, 1)

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$

$(n^{[1]}, m)$ — element-wise — $(n^{[1]}, m)$

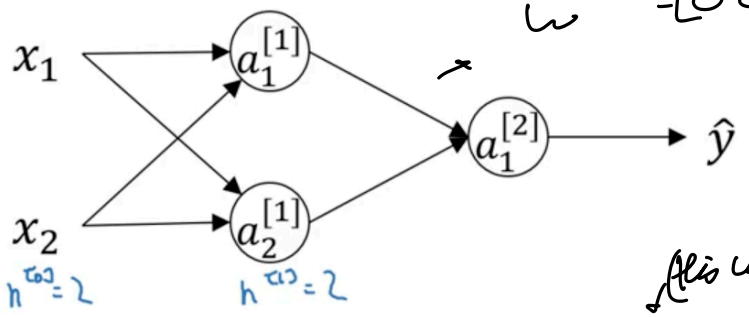
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$
$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis=1, keepdims=True)$$
↳ $(n^{[1]}, 1)$

WATCH WITH THESE VIDS TO SEE DERIVATION.



RANDOM INITIALIZATION

What happens if you initialize weights to zero?



$w^{[2]} = [0\ 0]$.

(this will be OK)

$n^{[0]} = 2$ $n^{[1]} = 2$

$w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ $b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Initially,

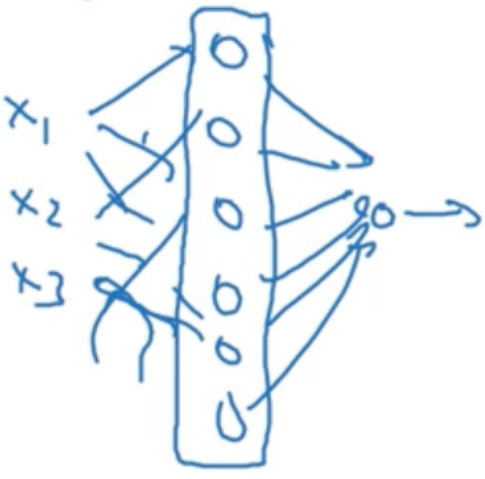
$a_1^{[1]} = a_2^{[1]}$ and then $dz_1^{[1]} = dz^{[1]}$.

So these two hidden units are symmetric.

$dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$ $\therefore w^{[1]} = w^{[1]} - \alpha\, dw$.

THE 2 HIDDEN UNITS ARE COMPUTING THE SAME THING

Symmetric



→ So they're doing the same thing.

Use [illegible] random.

$W^{[1]} = np.random.randn(2,2) * 0.01$ to a small random variable

$b^{[1]} = np.zeros((2,1))$

Don't have problem here.

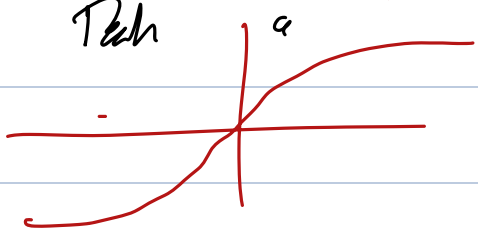
$W^{[2]} =$ random

$b^{[2]} = 0$

Can be a better for deep neural network

b/c

Prefer to start with small weights



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

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

large weights $\Rightarrow$ large a

So gradient descent (or learning) will be very slow.