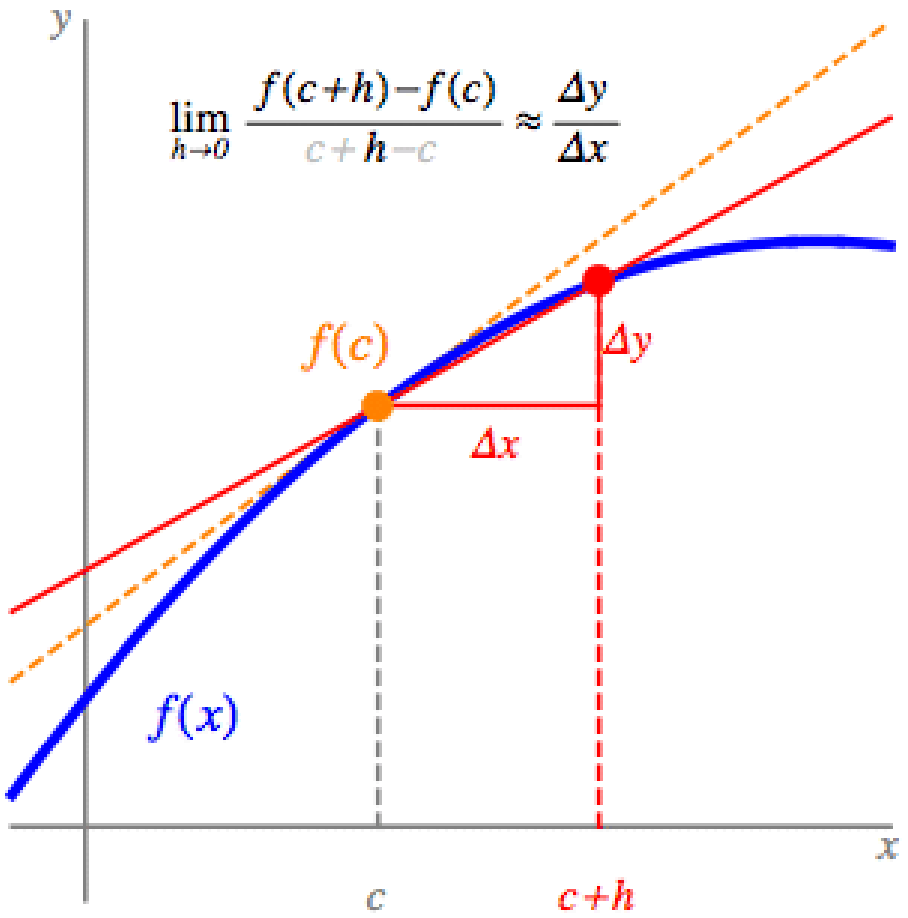
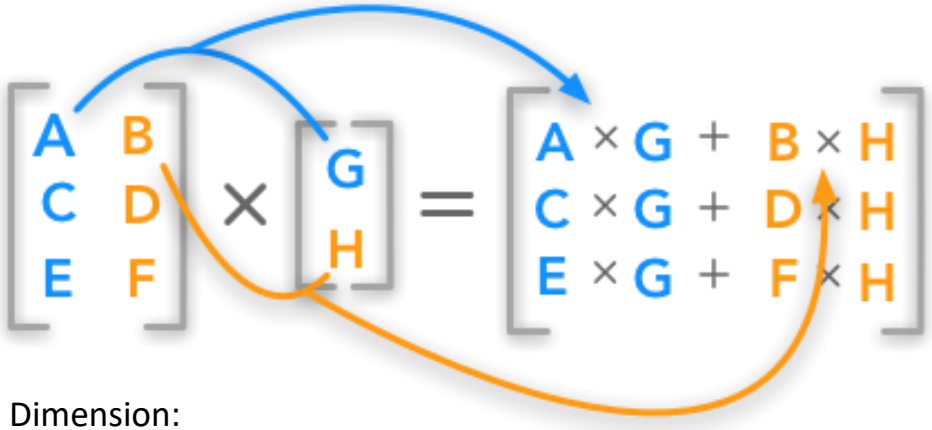
Limit Definition of the Derivative $f'(c)$



https://github.com/LambdaSchool/DS-Unit-4-Sprint-2-Neural-Networks/blob/main/module2-Train/LS_DS_422_Train_Lecture.ipynb

Review of Matrix Multiplication (linear algebra)



Dimension:

m*n n*t m*t

$$= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

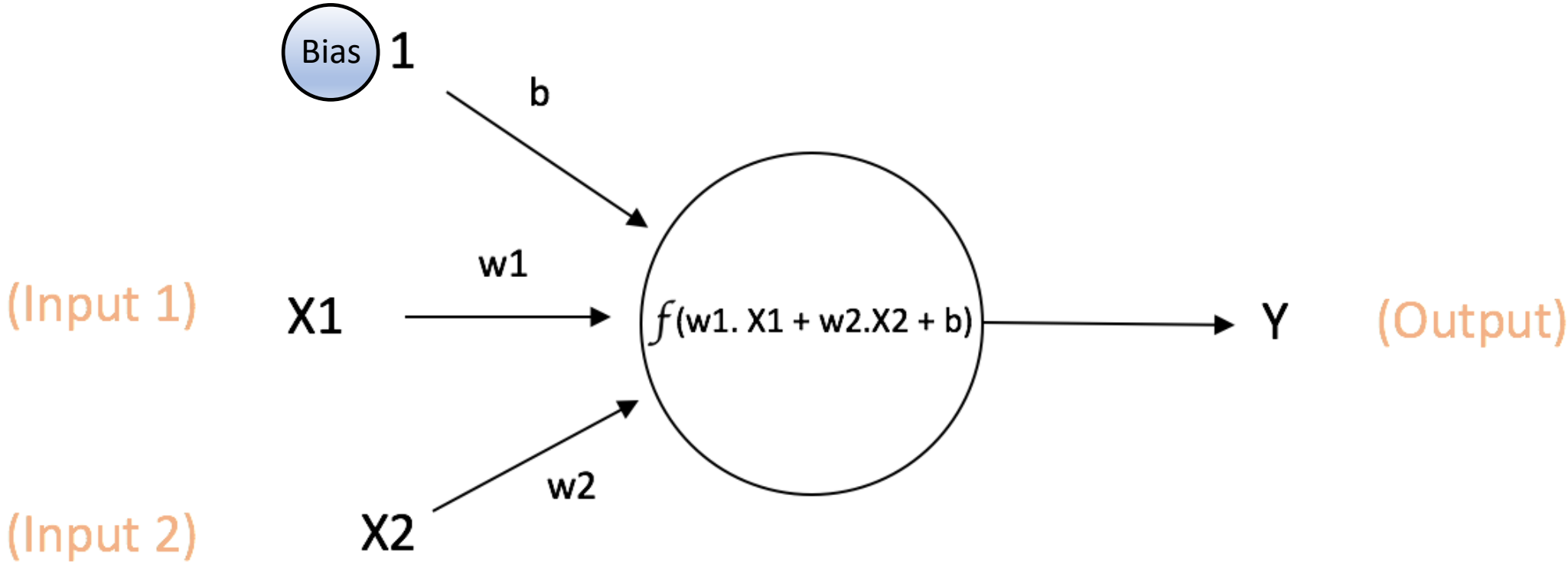
$$= \begin{bmatrix} 1(5) + 2(7) & 1(6) + 2(8) \\ 3(5) + 4(7) & 3(6) + 4(8) \end{bmatrix}$$

$$= \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

$$\underset{3\times 2}{\begin{bmatrix} 0 & 1 \\ 4 & 3 \\ 5 & -1 \end{bmatrix}} \underset{2\times 3}{\begin{bmatrix} 2 & -1 & 0 \\ 3 & 4 & 1 \end{bmatrix}}$$

$$\begin{bmatrix} 0+3 & 0+4 & 0+1 \\ 8+9 & -4+12 & 0+3 \\ 10-3 & -5-4 & 0-1 \end{bmatrix} = \begin{bmatrix} 3 & 4 & 1 \\ 17 & 8 & 3 \\ 7 & -9 & -1 \end{bmatrix}$$

A Good References



Output of neuron = Y= f(w1. X1 + w2.X2 + b)

Network Weight Initialization¶

how we initalize our model weights can determine the difference between Gradient Descent converging towards a local minimum or a global minimum!

Keras has documentation on all these options

```
init_mode = ['uniform', 'lecun_uniform', 'normal', 'zero', 'glorot_normal', 'glorot_uniform', 'he_normal', 'he_uniform']
```

If you wish to dive into a deep analysis of varying weight initializers and their affect on model performance **check this article.**(optional).

Take Away: Include a few different weight initalizers in your gridsearch.

Weight Initialization

Equation 11-1. Xavier initialization (when using the logistic activation function)

Normal distribution with mean 0 and standard deviation $\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

Or a uniform distribution between -r and +r, with $r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$

Table 11-1. Initialization parameters for each type of activation function

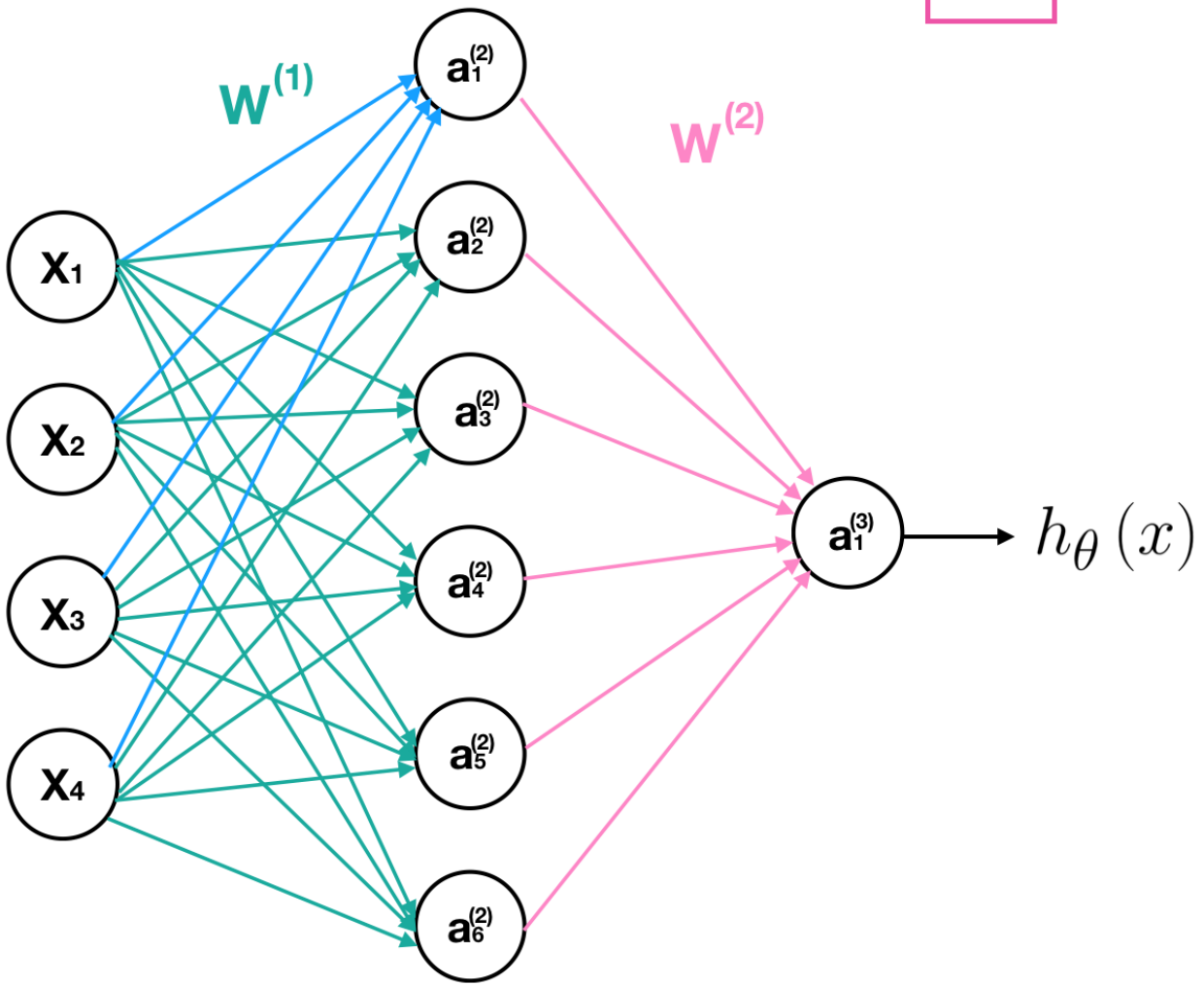
Activation function	Uniform distribution [–r, r]	Normal distribution
Logistic	$r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
Hyperbolic tangent	$r = 4\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = 4\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
ReLU (and its variants)	$r = \sqrt{2}\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

"This initialization strategy is often called *Xavier initialization* (after the author's first name), or sometimes *Glorot initialization*." *Hands-On Machine Learning with Scikit-Learn and TensorFlow* Aurélien Géron *2017*

Forward Propagation

$$W^{(1)} = \begin{bmatrix} \theta_{11}^{(1)} & \theta_{21}^{(1)} & \theta_{31}^{(1)} & \theta_{41}^{(1)} & \theta_{51}^{(1)} & \theta_{61}^{(1)} \\ \theta_{12}^{(1)} & \theta_{22}^{(1)} & \theta_{32}^{(1)} & \theta_{42}^{(1)} & \theta_{52}^{(1)} & \theta_{62}^{(1)} \\ \theta_{13}^{(1)} & \theta_{23}^{(1)} & \theta_{33}^{(1)} & \theta_{43}^{(1)} & \theta_{53}^{(1)} & \theta_{63}^{(1)} \\ \theta_{14}^{(1)} & \theta_{24}^{(1)} & \theta_{34}^{(1)} & \theta_{44}^{(1)} & \theta_{54}^{(1)} & \theta_{64}^{(1)} \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} \theta_{11}^{(2)} \\ \theta_{12}^{(2)} \\ \theta_{13}^{(2)} \\ \theta_{14}^{(2)} \\ \theta_{15}^{(2)} \\ \theta_{16}^{(2)} \end{bmatrix}$$



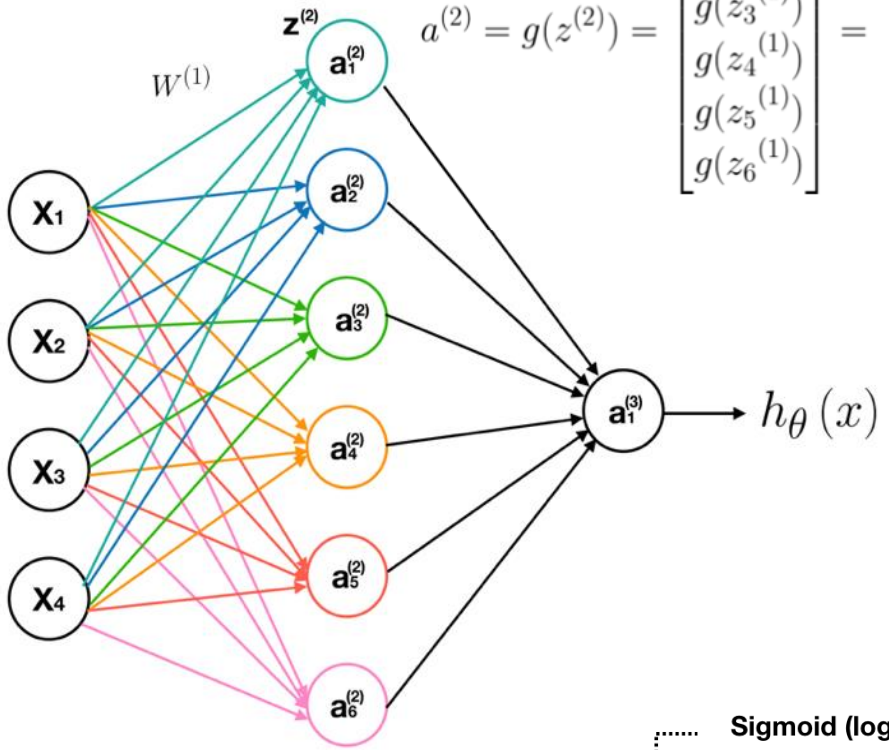
Forward Propagation

only calculate value by sigmoid function ar

$$W^T X = \begin{bmatrix} \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} & \theta_{14}^{(1)} \\ \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} & \theta_{24}^{(1)} \\ \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)} & \theta_{34}^{(1)} \\ \theta_{41}^{(1)} & \theta_{42}^{(1)} & \theta_{43}^{(1)} & \theta_{44}^{(1)} \\ \theta_{51}^{(1)} & \theta_{52}^{(1)} & \theta_{53}^{(1)} & \theta_{54}^{(1)} \\ \theta_{61}^{(1)} & \theta_{62}^{(1)} & \theta_{63}^{(1)} & \theta_{64}^{(1)} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \\ z_4^{(1)} \\ z_5^{(1)} \\ z_6^{(1)} \end{bmatrix} = Z^{(2)}$$

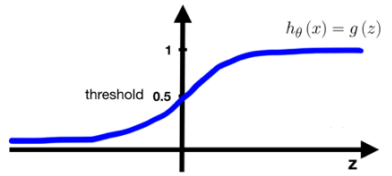
$$z_1^{(2)} = \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3 + \theta_{14}^{(1)} x_4$$
$$z_2^{(2)} = \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3 + \theta_{24}^{(1)} x_4$$
$$z_3^{(2)} = \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3 + \theta_{34}^{(1)} x_4$$
$$z_4^{(2)} = \theta_{41}^{(1)} x_1 + \theta_{42}^{(1)} x_2 + \theta_{43}^{(1)} x_3 + \theta_{44}^{(1)} x_4$$
$$z_5^{(2)} = \theta_{51}^{(1)} x_1 + \theta_{52}^{(1)} x_2 + \theta_{53}^{(1)} x_3 + \theta_{54}^{(1)} x_4$$
$$z_6^{(2)} = \theta_{61}^{(1)} x_1 + \theta_{62}^{(1)} x_2 + \theta_{63}^{(1)} x_3 + \theta_{64}^{(1)} x_4$$

$$a^{(2)} = g(z^{(2)}) = \begin{bmatrix} g(z_1^{(1)}) \\ g(z_2^{(1)}) \\ g(z_3^{(1)}) \\ g(z_4^{(1)}) \\ g(z_5^{(1)}) \\ g(z_6^{(1)}) \end{bmatrix} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \\ a_4^{(2)} \\ a_5^{(2)} \\ a_6^{(2)} \end{bmatrix}$$



Sigmoid (logistic) Activation Function

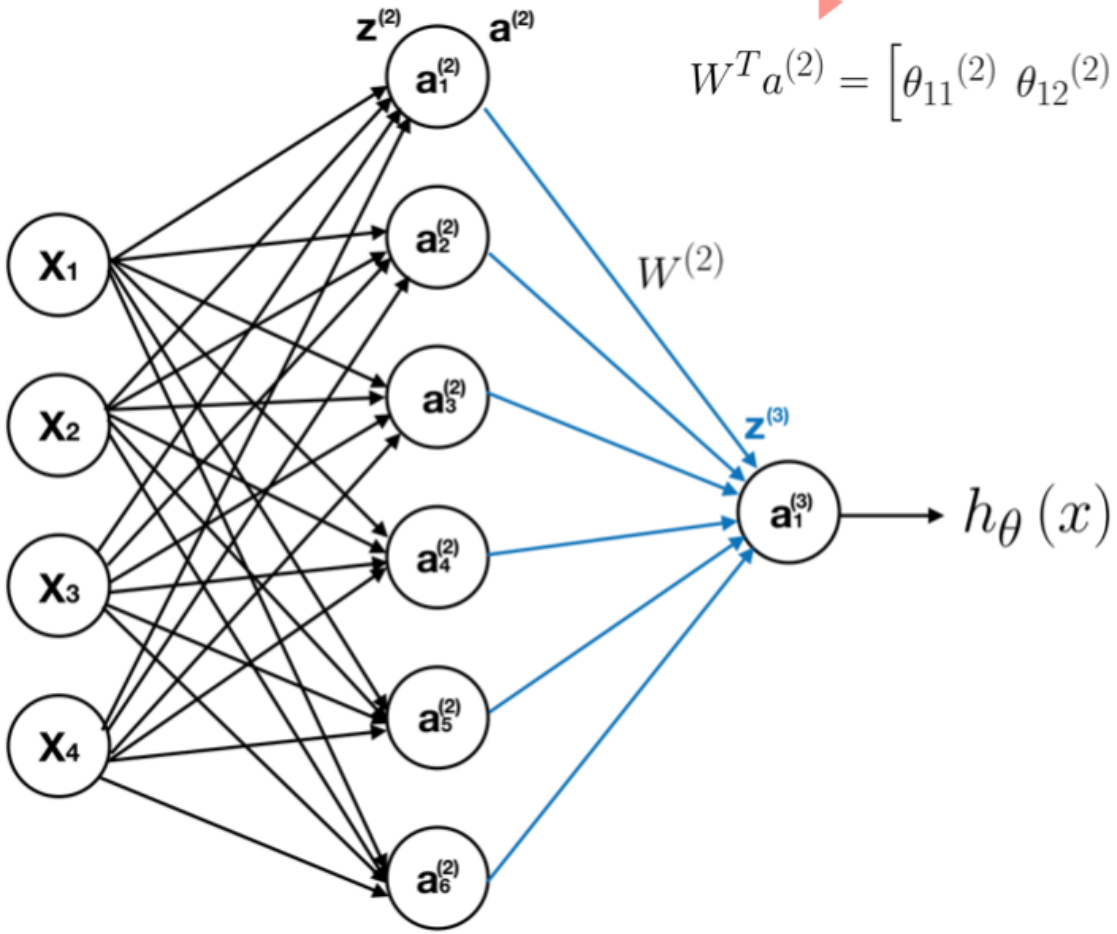
$$h_\theta(x) = g(-\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

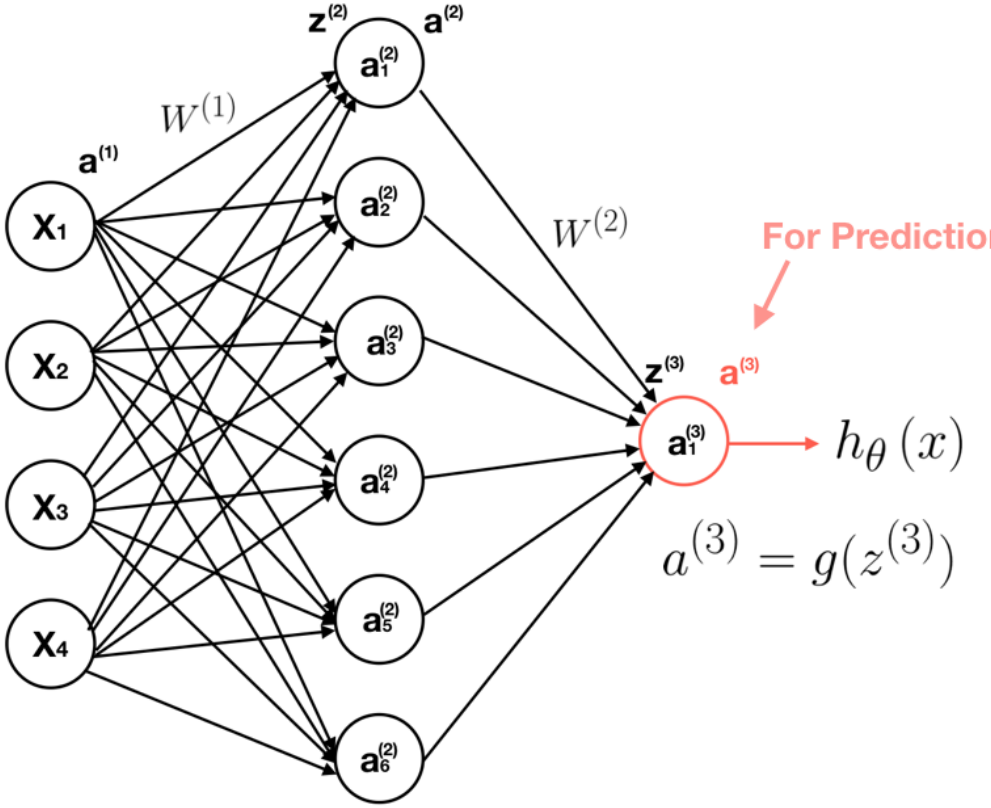


Forward Propagation

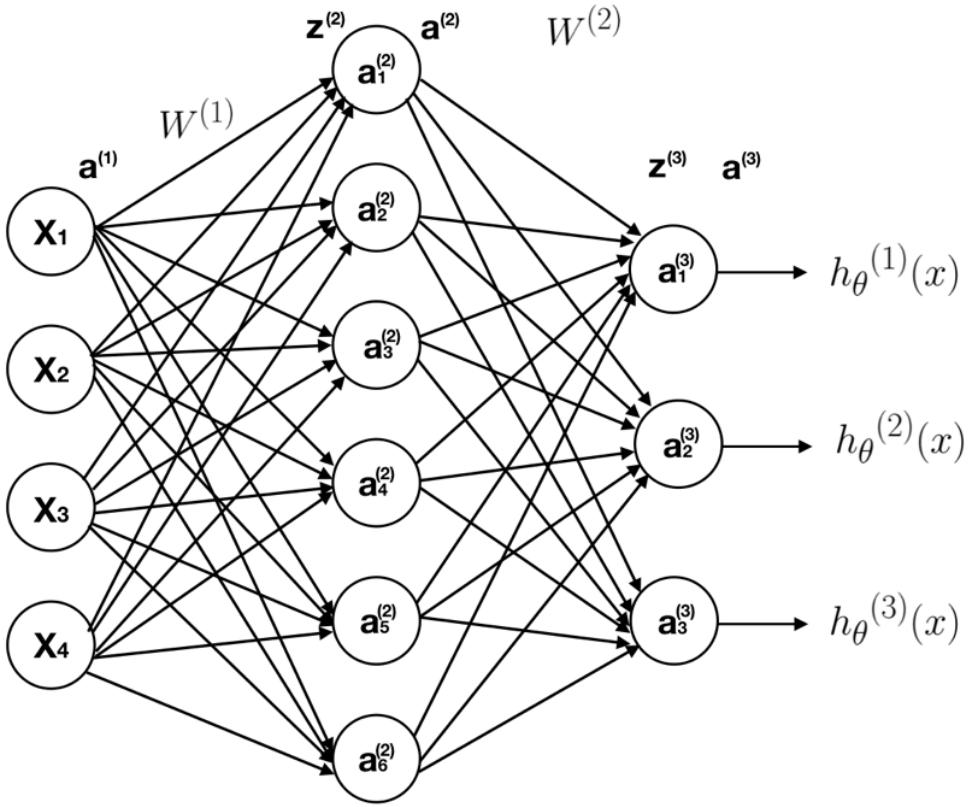
Passing Value

$$W^T a^{(2)} = \begin{bmatrix} \theta_{11}^{(2)} & \theta_{12}^{(2)} & \theta_{13}^{(2)} & \theta_{14}^{(2)} & \theta_{15}^{(2)} & \theta_{16}^{(2)} \end{bmatrix} \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \\ a_4^{(2)} \\ a_5^{(2)} \\ a_6^{(2)} \end{bmatrix} = z^{(3)}$$





Forward Propagation



Forward Propagation

$$W^{(1)^T} X = z^{(2)}$$

$$a^{(2)} = g(z^{(2)})$$

$$W^{(2)^T} a^{(2)} = z^{(3)}$$

$$a^{(3)} = g(z^{(3)})$$

For Prediction

$$a^{(3)} = \begin{bmatrix} a_1^{(3)} \\ a_2^{(3)} \\ a_3^{(3)} \end{bmatrix} \begin{matrix} \rightarrow P(x \in Class1) \\ \rightarrow P(x \in Class2) \\ \rightarrow P(x \in Class3) \end{matrix}$$

For Prediction

Multi-Classification Problem

Forward Propagation: Gradient Descent

data number

Regularization Term

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

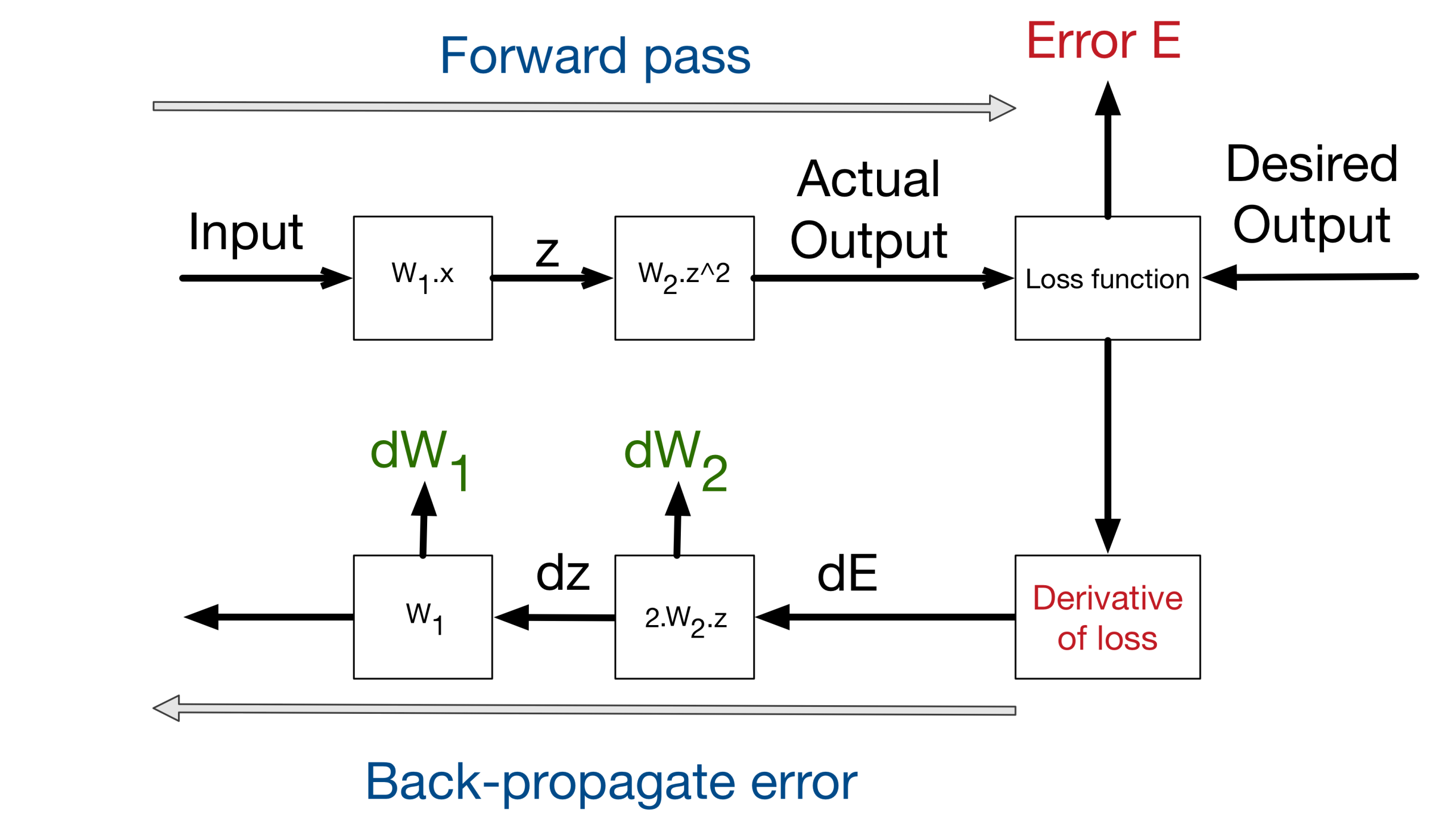
label number

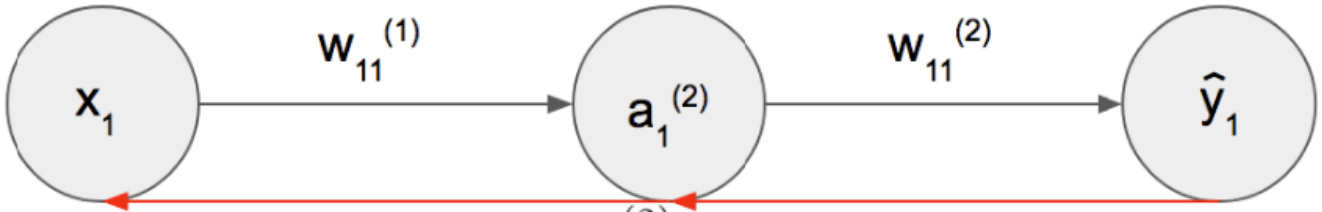
Repeat until converge {

$$\Theta_{ij}{}^{(l)} = \Theta_{ij}{}^{(l)} - \alpha\frac{\partial J(\Theta)}{\partial \Theta_{ij}{}^{(l)}}$$

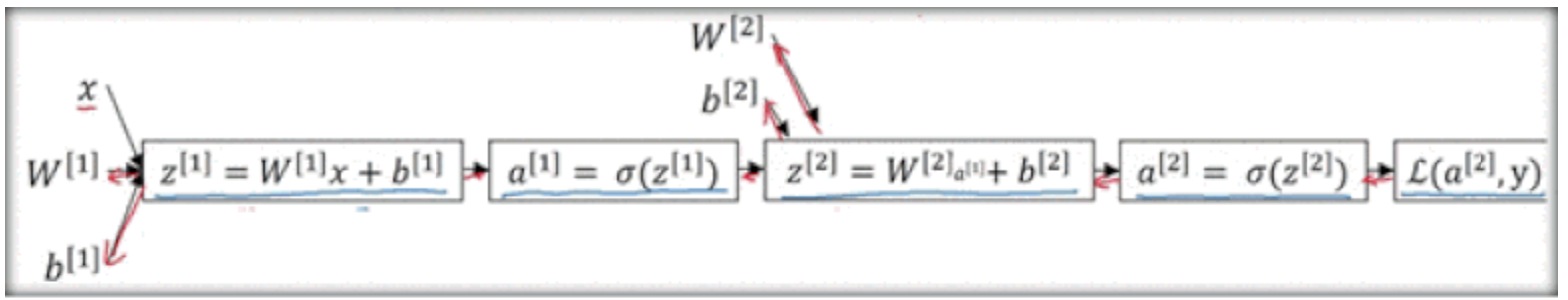
}

learning rate


Forward pass
Error E
Input
$W_1.x$
z
$W_2.z^2$
Actual Output
Loss function
Desired Output
dW_1
dW_2
W_1
dz
$2.W_2.z$
dE
Derivative of loss
Back-propagate error



$$\frac{\partial E}{\partial w_{11}^{(1)}} = \frac{\partial E}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial a_1^{(2)}}\frac{\partial a_1^{(2)}}{\partial w_{11}^{(1)}} \qquad \frac{\partial E}{\partial w_{11}^{(2)}} = \frac{\partial E}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial w_{11}^{(2)}}$$



$$\frac{\partial L}{\partial a}$$

All

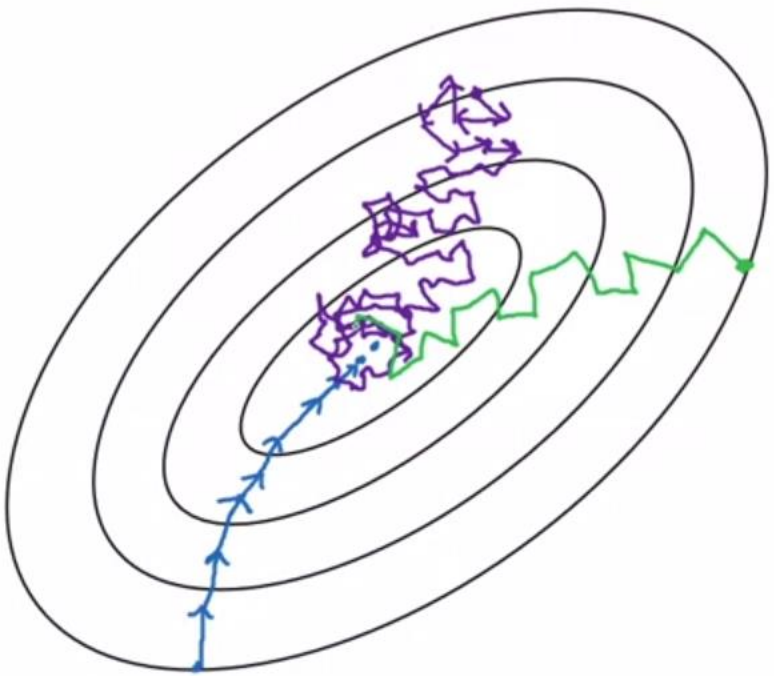
Batch size

Choosing your mini-batch size

→ If mini-batch size = m : Batch gradient descent. $(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$.

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is it own $(X^{\{1\}}, Y^{\{1\}}) = (x^{(1)}, y^{(1)})$ ... $(x^{(2)}, y^{(1)})$ mini-batch.

In practice: Somewhere in-between 1 and m



Stochastic gradient descent

{

Lose speedup from vectorization

In-between (mini-batch size not too big/small)

Fastest learning.

- Vectorization. (n 1000)
- Make progress without processing entire training set.

Batch gradient descent (mini-batch size = m)

Too long per iteration

Andrew Ng

Choosing your mini-batch size

If small train set: Use batch gradient descent.
($m \leq 2000$)

Typical mini-batch sizes:

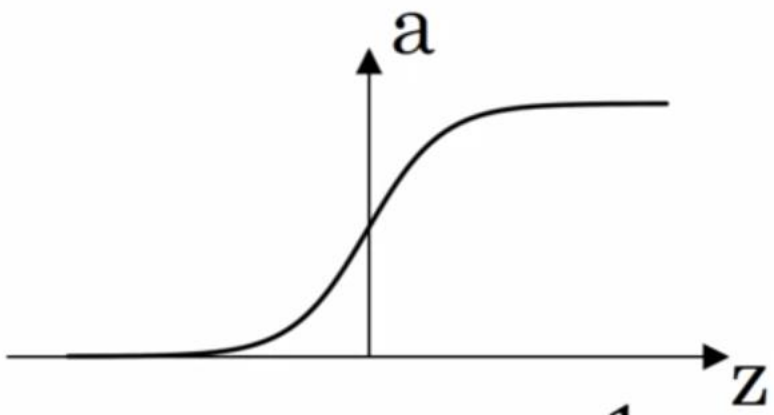
64, 128, 256, 512 1024

2^6 2^7 2^8 2^9 2^{10}

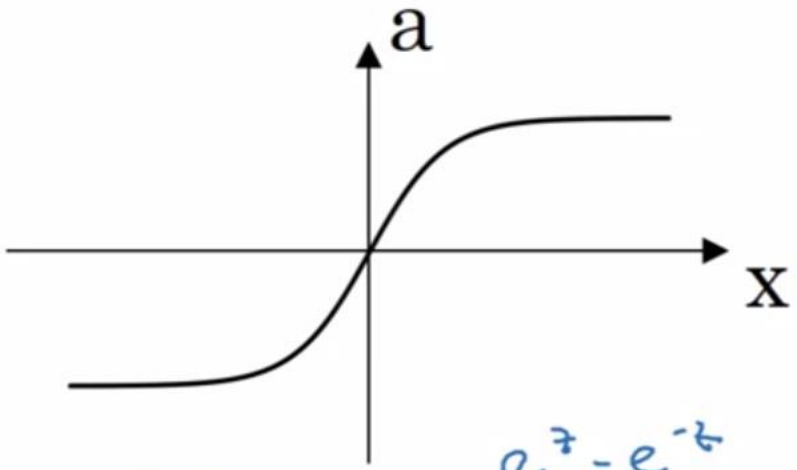
Make sure mini-batch fit in CPU/GPU memory.

$X^{\{t\}}, Y^{\{t\}}$.

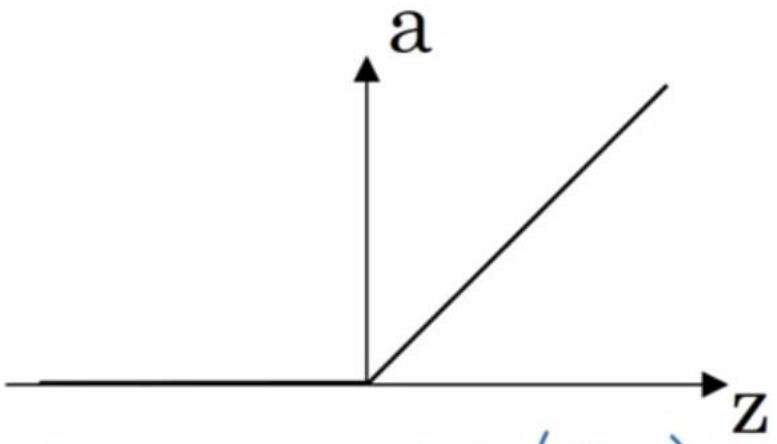
Pros and cons of activation functions



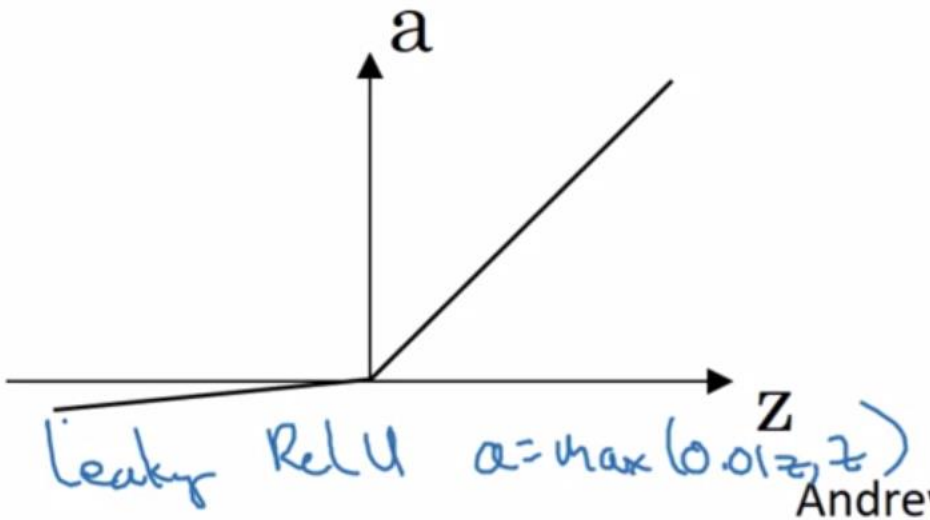
sigmoid: $a = \frac{1}{1+e^{-z}}$



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



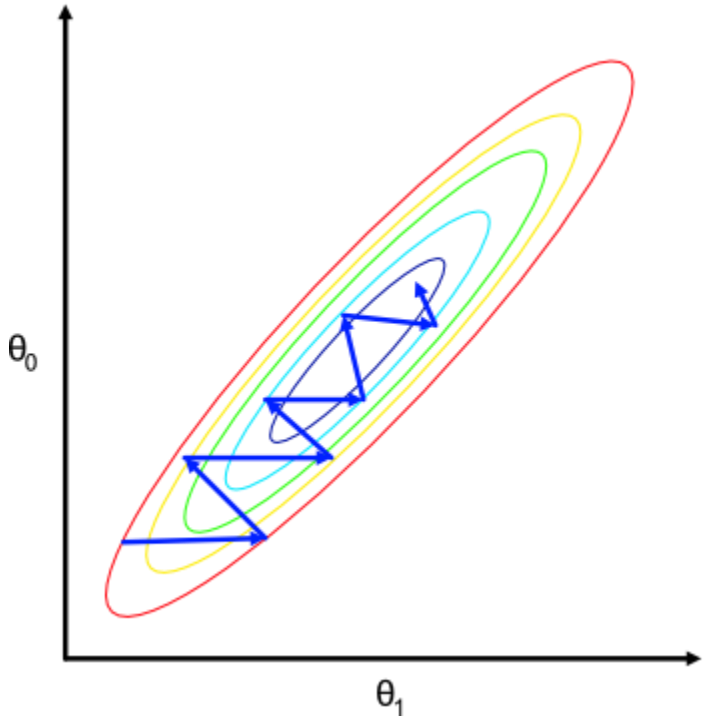
ReLU $a = \max(0, z)$

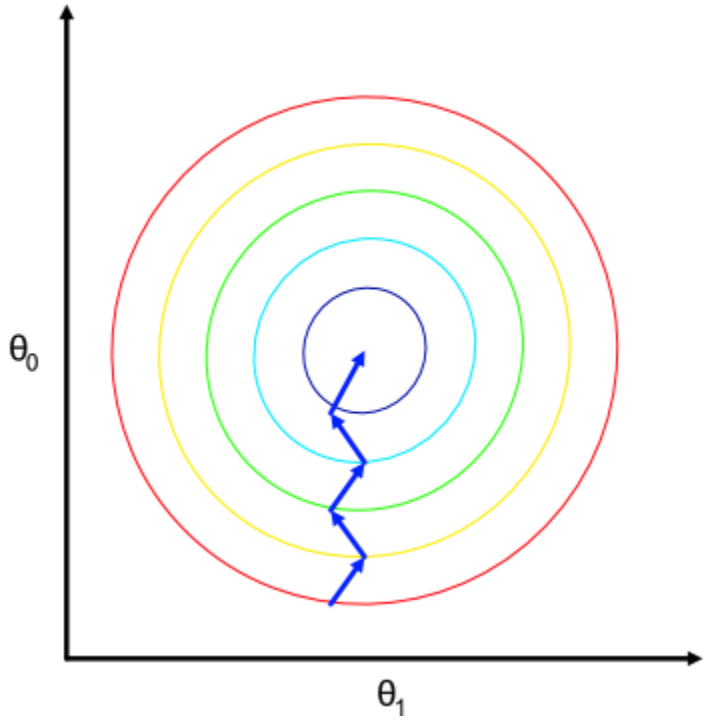


Leaky ReLU $a = \max(0.01z, z)$

```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.fit(X, y, epochs=5)
```

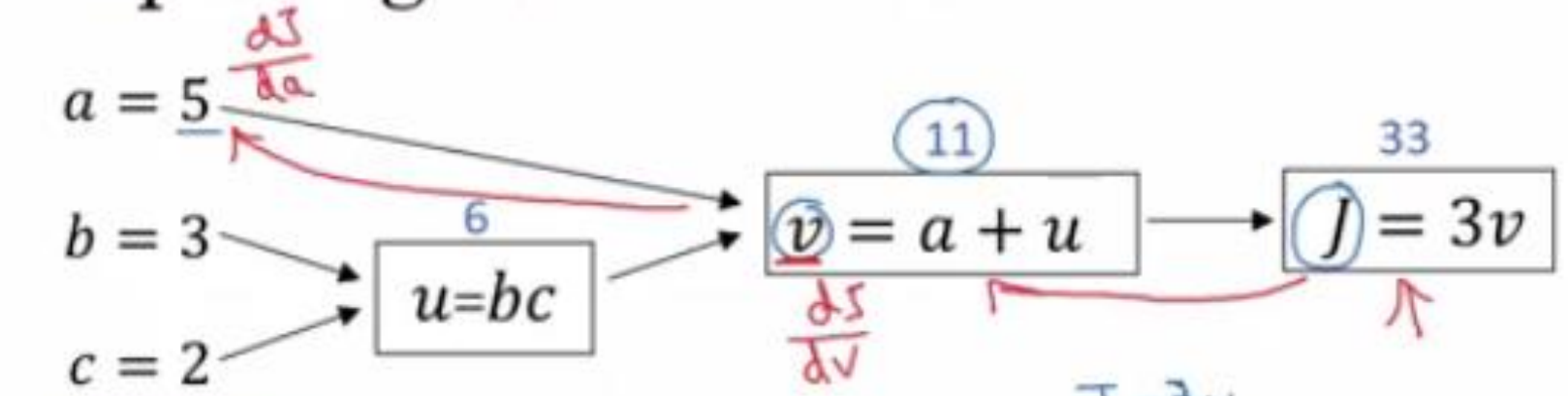
(Inputs) Normalization





Appendix

Computing derivatives



$\frac{dJ}{dv} = ? = 3$

$a \to v \to J$

$J = 3v$
$v = 11 \to 11.001$
$J = 33 \to 33.003$

$a = 5 \to 5.001$
$\to v = 11 \to 11.001$
$J = 33 \to 33.003$

$f(a) = 3a$
$\frac{df(a)}{da} = \frac{df}{da} = 3$
$J = 3v$
$\frac{dJ}{dv} = 3$

$\frac{dJ}{da} = 3 = \frac{dJ}{dv}\frac{dv}{da}$

3×1

$\frac{dv}{da} = 1$

Computation Graph

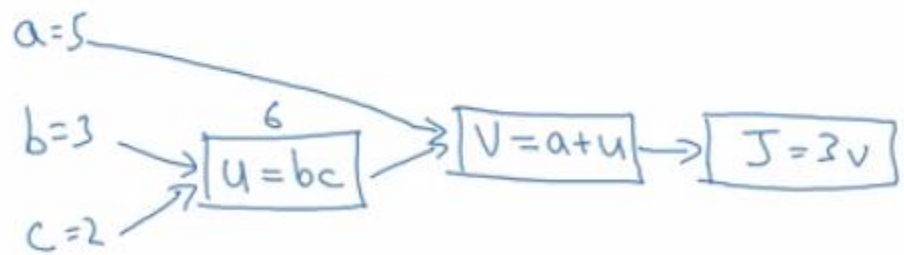
$J(a,b,c) = 3(a + \underbrace{bc}_{u})$

$J = \underbrace{3(\underbrace{a + u}_{v})}_{J}$

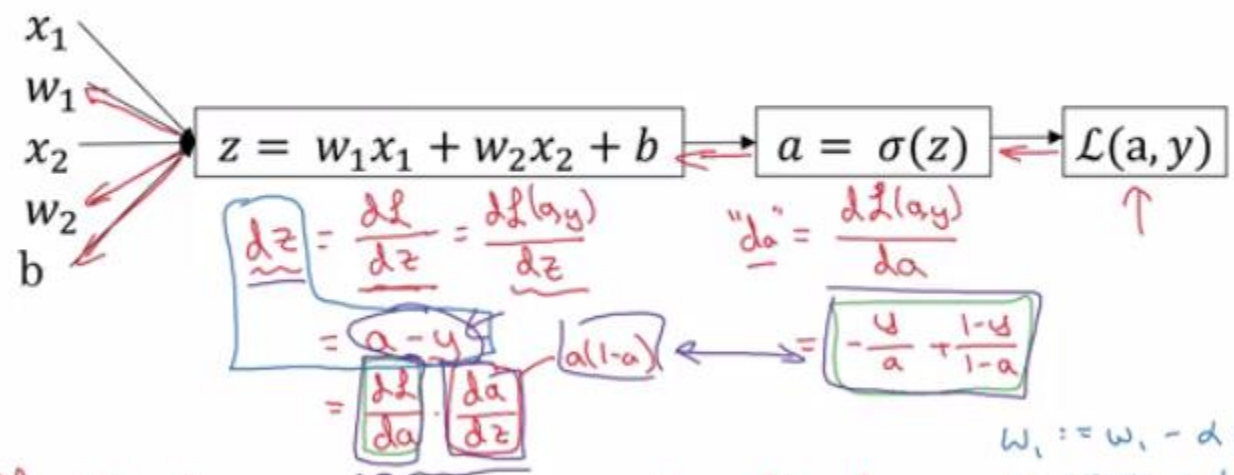
$u = bc$

$v = a + u$

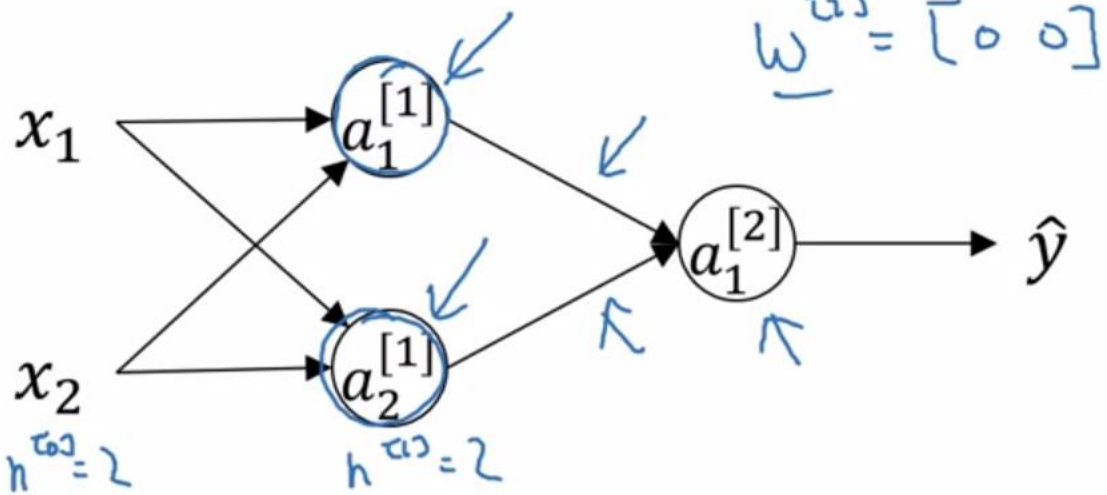
$J = 3v$



Logistic regression derivatives



What happens if you initialize weights to zero?



$w^{[1]} = [0\ 0]$

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

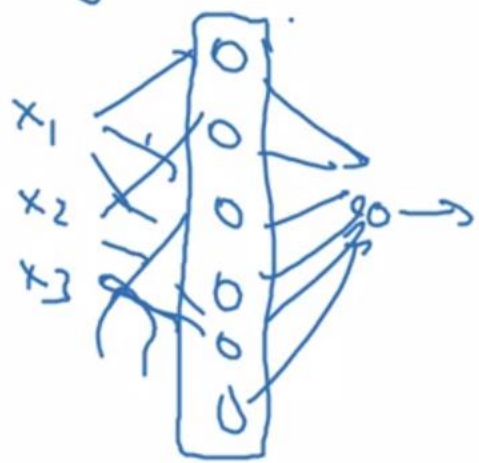
$$w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$a_1^{[1]} = a_2^{[1]}$ $\qquad dz_1^{[1]} = dz_2^{[1]}$

$$dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$$

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

Symmetric

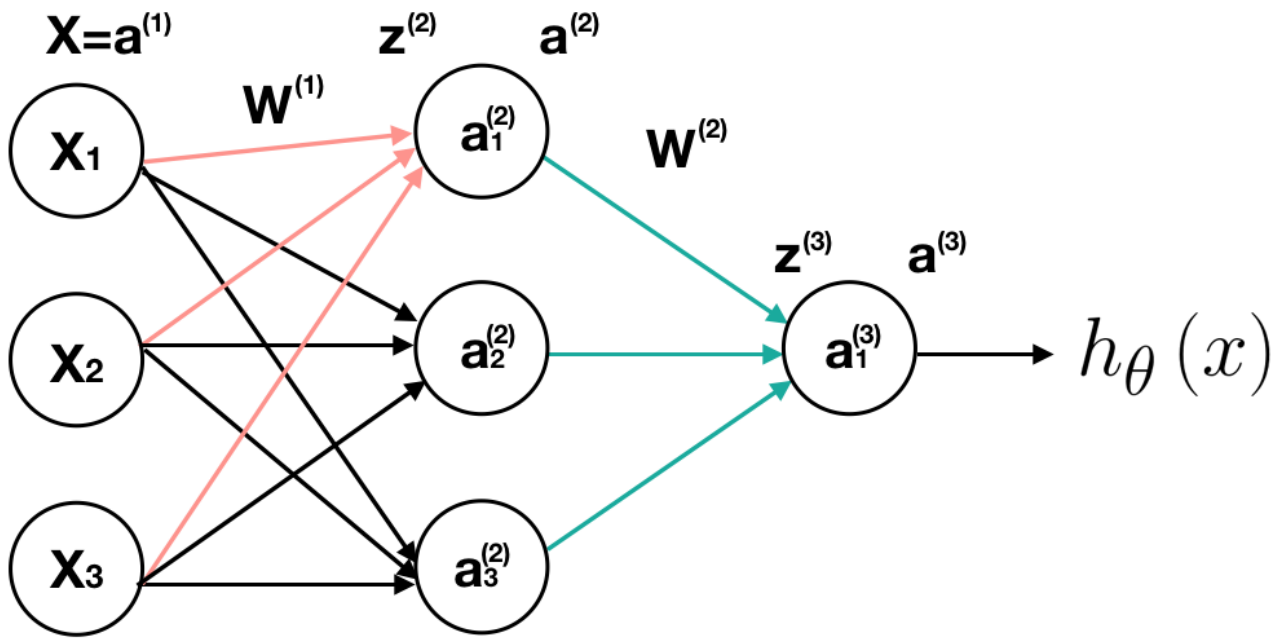


$$w^{[1]} = \begin{bmatrix} \cdots & \cdots \\ \cdots & \cdots \end{bmatrix}$$

Andrew Ng

$$\Theta^{(1)} = W^{(1)} = \begin{bmatrix} \Theta_{11}^{(1)} & \Theta_{21}^{(1)} & \Theta_{31}^{(1)} \\ \Theta_{12}^{(1)} & \Theta_{22}^{(1)} & \Theta_{32}^{(1)} \\ \Theta_{13}^{(1)} & \Theta_{23}^{(1)} & \Theta_{33}^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} & w_{33}^{(1)} \end{bmatrix}$$

$$\Theta^{(2)} = W^{(2)} = \begin{bmatrix} \Theta_{11}^{(2)} \\ \Theta_{12}^{(2)} \\ \Theta_{13}^{(2)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(2)} \\ w_{12}^{(2)} \\ w_{13}^{(2)} \end{bmatrix}$$



$$\frac{\partial J(\Theta)}{\Theta^{(2)}} = \delta^{(3)} * a^{(2)}$$

$$\frac{\partial J(\Theta)}{\Theta^{(1)}} = \delta^{(2)} * a^{(1)}$$

$$\delta^{(3)} = (a^{(3)} - y) * g'(z^{(3)})$$

$$\delta^{(2)} = \delta^{(3)}(W^{(2)})^T g'(z^{(2)})$$

Hint

$$\delta^{(3)} = (a^{(3)} - y) * g(z^{(3)})$$

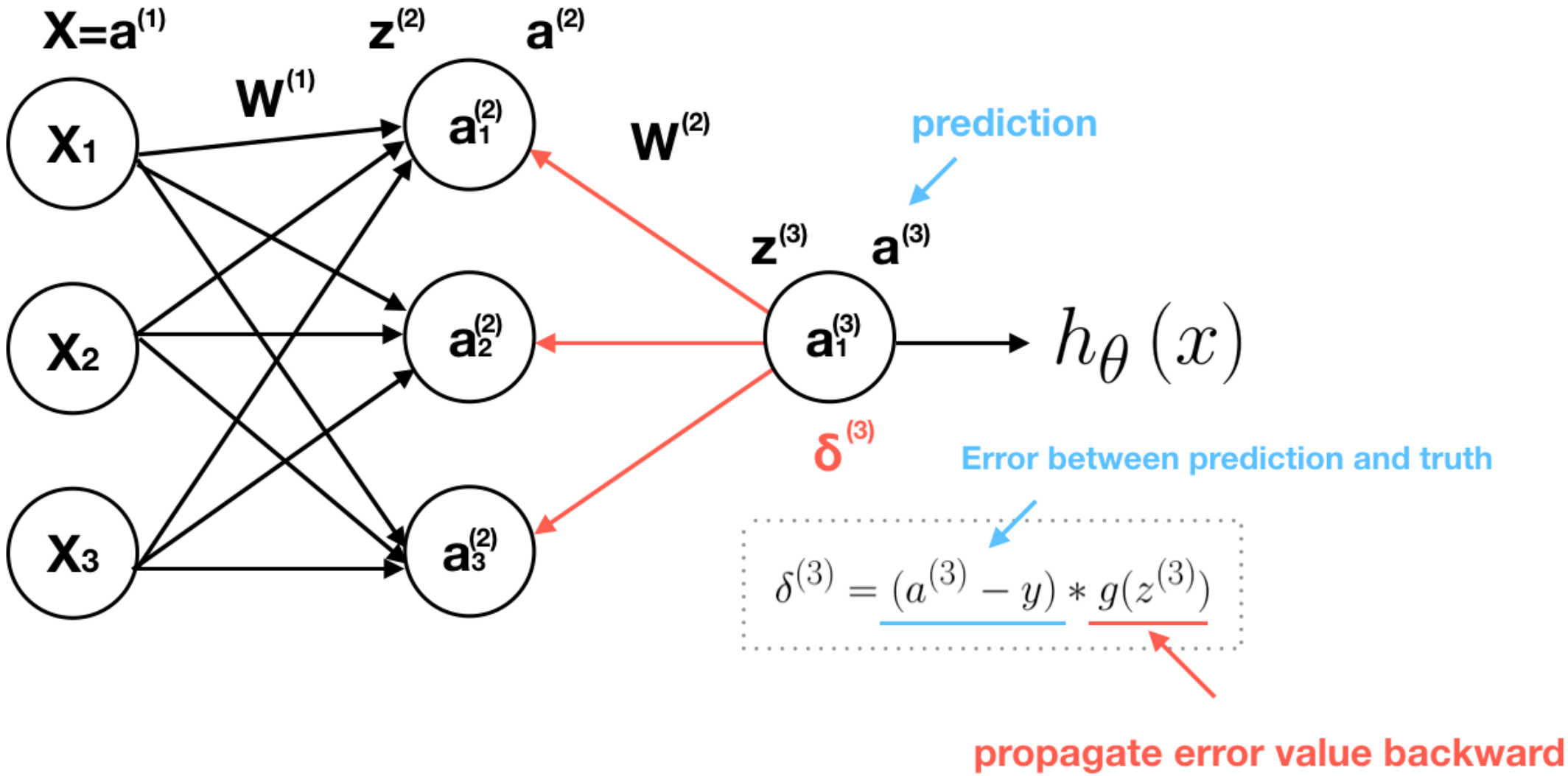
$$\begin{bmatrix} \delta_1^{(3)} \\ \delta_2^{(3)} \\ \delta_3^{(3)} \end{bmatrix} = \begin{bmatrix} (a_1^{(3)} - y_1) \\ (a_2^{(3)} - y_2) \\ (a_3^{(3)} - y_3) \end{bmatrix} * \begin{bmatrix} g'(z_1^{(3)}) \\ g'(z_2^{(3)}) \\ g'(z_3^{(3)}) \end{bmatrix} = \begin{bmatrix} (a_1^{(3)} - y_1) * g'(z_1^{(3)}) \\ (a_2^{(3)} - y_2) * g'(z_2^{(3)}) \\ (a_3^{(3)} - y_3) * g'(z_3^{(3)}) \end{bmatrix}$$

Hint

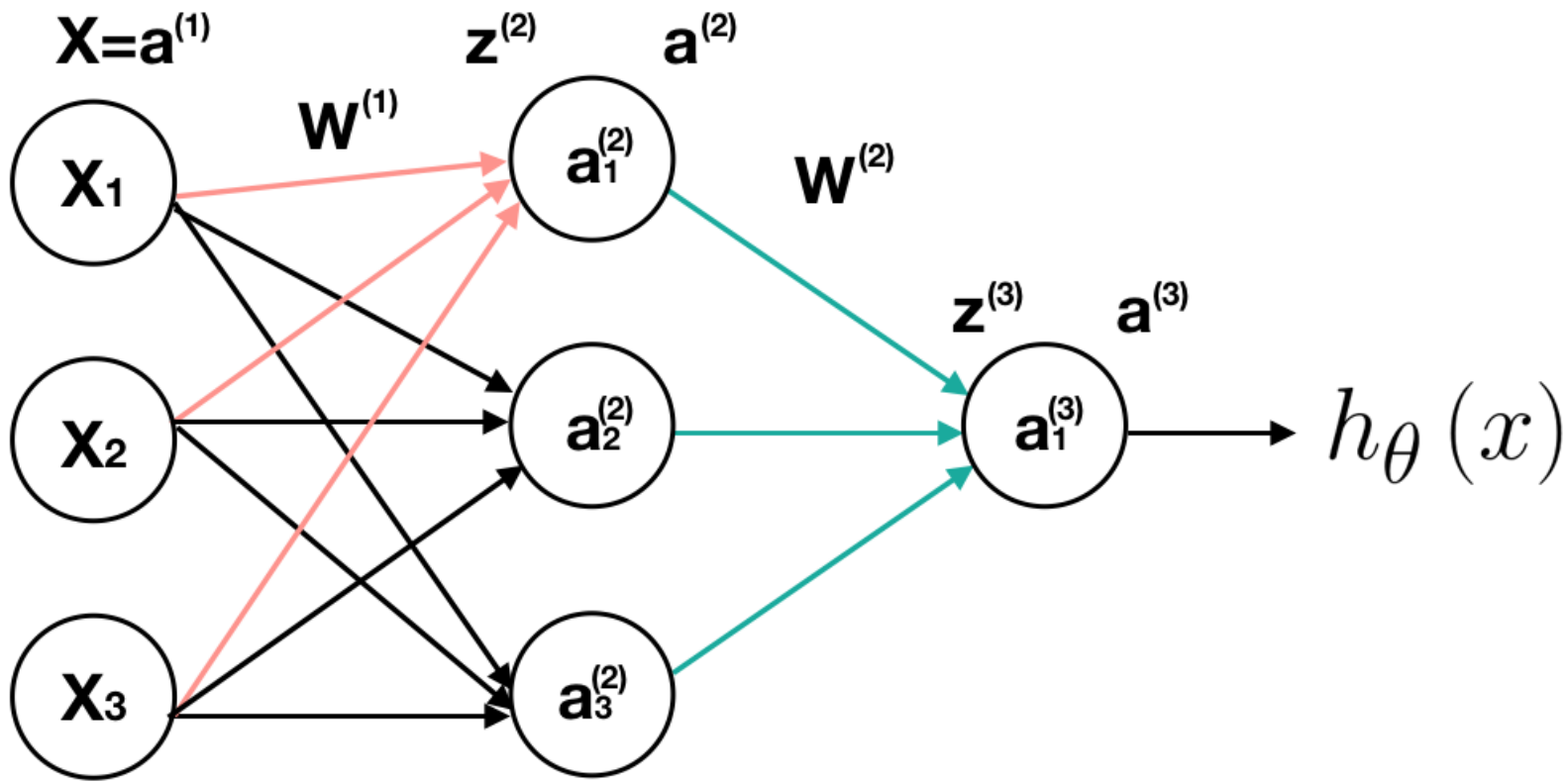
$$\delta^{(2)} = \delta^{(3)}(W^{(2)})^T g'(z^{(2)})$$

$$= \left(\begin{bmatrix} \delta_1^{(3)} \\ \delta_2^{(3)} \\ \delta_3^{(3)} \end{bmatrix} \begin{bmatrix} w_1^{(2)} & w_2^{(2)} * w_3^{(2)} \end{bmatrix} \right) * \begin{bmatrix} g'(z_1^{(2)}) \\ g'(z_2^{(2)}) \\ g'(z_3^{(2)}) \end{bmatrix}$$

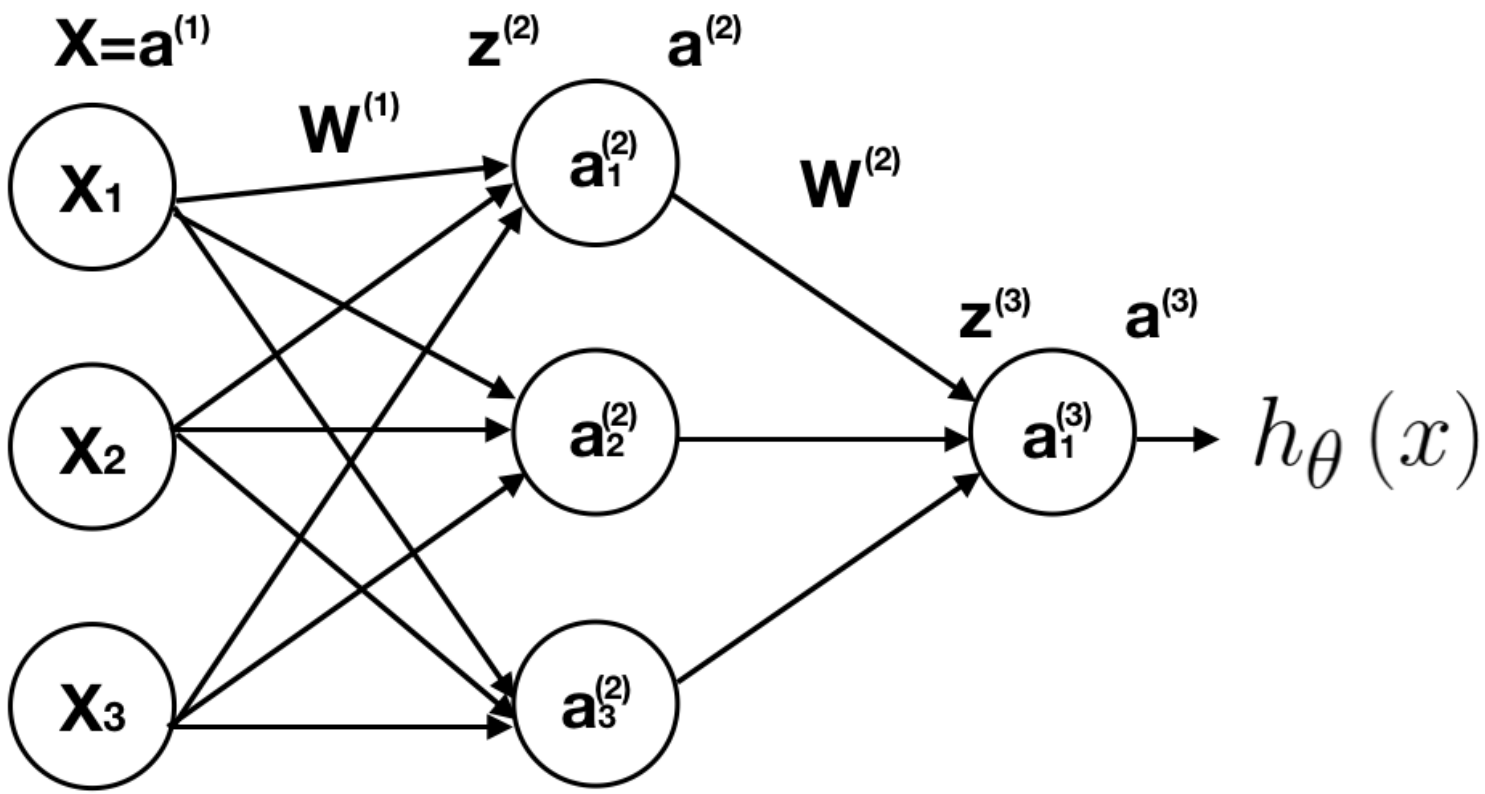
$$= \begin{bmatrix} \delta_1^{(3)} * w_1^{(2)} & \delta_1^{(3)} * w_2^{(2)} & \delta_1^{(3)} * w_3^{(2)} \\ \delta_2^{(3)} * w_2^{(2)} & \delta_2^{(3)} * w_2^{(2)} & \delta_2^{(3)} * w_3^{(2)} \\ \delta_3^{(3)} * w_3^{(2)} & \delta_3^{(3)} * w_2^{(2)} & \delta_3^{(3)} * w_3^{(2)} \end{bmatrix} * \begin{bmatrix} g'(z_1^{(2)}) \\ g'(z_2^{(2)}) \\ g'(z_3^{(2)}) \end{bmatrix}$$


X=a(1)
z(2)
a(2)
W(1)
W(2)
x1
x2
x3
a1(2)
a2(2)
a3(2)
prediction
z(3)
a(3)
a1(3)
hθ (x)
δ(3)
Error between prediction and truth
δ(3) = (a(3) − y) ∗ g(z(3))
propagate error value backward

$$\Theta^{(1)} = W^{(1)} = \begin{bmatrix} \Theta_{11}^{(1)} & \Theta_{21}^{(1)} & \Theta_{31}^{(1)} \\ \Theta_{12}^{(1)} & \Theta_{22}^{(1)} & \Theta_{32}^{(1)} \\ \Theta_{13}^{(1)} & \Theta_{23}^{(1)} & \Theta_{33}^{(1)} \end{bmatrix} \qquad \Theta^{(2)} = W^{(2)} = \begin{bmatrix} \Theta_{11}^{(2)} \\ \Theta_{12}^{(2)} \\ \Theta_{13}^{(2)} \end{bmatrix}$$



$$J(\Theta) = -[ylog(h_\Theta(x))+(1-y)log(1-h_\Theta(x))]$$



Forward Propagation

$$(\Theta^{(1)})^T X = z^{(2)}$$

$$a^{(2)} = g(z^{(2)})$$

$$(\Theta^{(2)})^T a^{(2)} = z^{(3)}$$

$$a^{(3)} = g(z^{(3)})$$

Hint

$$a^{(3)} = g(z^{(3)})$$

Hint

$$\Theta^{(2)} a^{(2)} = z^{(3)}$$

$a^{(3)} - y$ $\quad$ $g'(z^{(3)})$ $\quad$ ${a_1}^{(2)}$

$$\frac{\partial J(\Theta)}{{\Theta_{11}}^{(2)}} = \frac{\partial J(\Theta)}{\partial {a_1}^{(3)}} * \frac{\partial {a_1}^{(3)}}{\partial {z_1}^{(3)}} * \frac{\partial {z_1}^{(3)}}{{\Theta_{11}}^{(2)}} = {\delta_1}^{(3)} * {a_1}^{(2)}$$

Hint

$$\frac{\partial J(\Theta)}{{a_1}^{(3)}} = -[y * \frac{1}{g(z)} * g'(z) + (1-y) * \frac{1}{1-g(z)} * (-g'(z))]$$
$$= -[y * (1-g(z)) + (y-1) * g(z)] = g(z) - y$$

Hint-The Property of Sigmoid Function

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

$\delta_1^{(3)}$ $g'(z_1^{(2)})$ $a_1^{(1)}$ or x_1

$$\frac{\partial J(\Theta)}{\Theta_{11}^{(1)}} = \frac{\partial J(\Theta)}{\partial a_1^{(3)}} * \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} * \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} * \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} * \frac{\partial z_1^{(2)}}{\Theta_{11}^{(1)}} = \delta_1^{(2)} * a_1^{(1)}$$

$g'(z_1^{(2)})$

Hint

$$(\Theta^{(1)})^T X = z^{(2)}$$

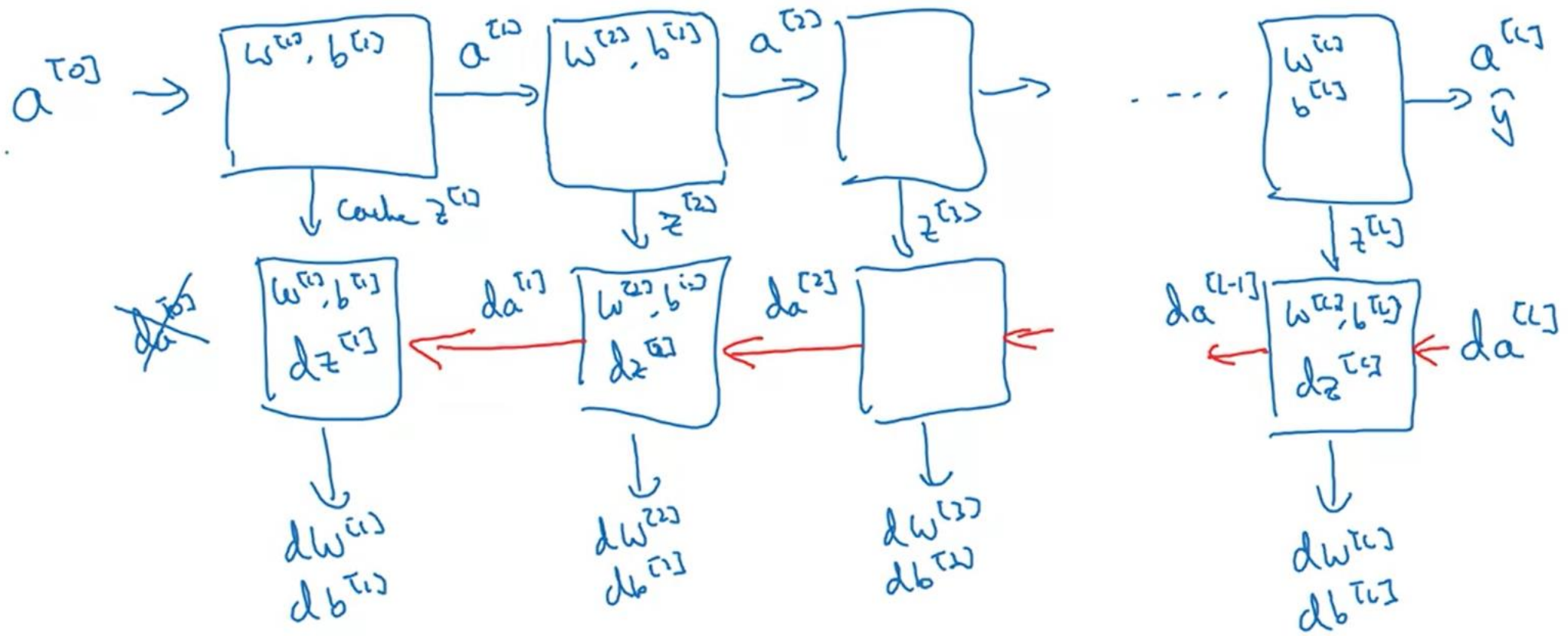
Hint

$$a^{(2)} = g(z^{(2)})$$

Hint

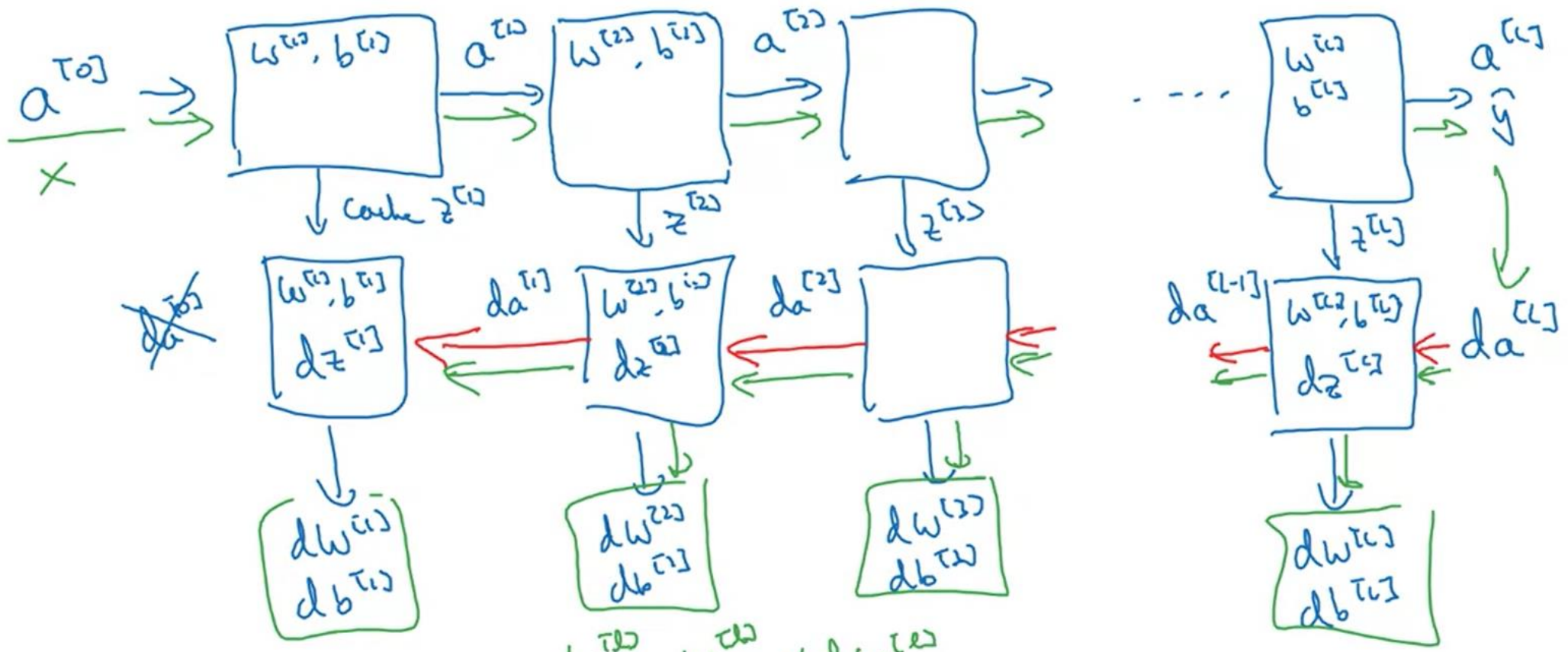
$$\Theta^{(2)} a^{(2)} = z^{(3)}$$

Forward and backward functions



Andrew Ng

Forward and backward functions



Andrew Ng

Forward propagation for layer l

→ Input $a^{[l-1]}$

→ Output $a^{[l]}$, cache ($z^{[l]}$)

$W^{[l]}, b^{[l]}$

$z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]}$

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

Vectorized:

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

$A^{[l]} = g^{[l]}(Z^{[l]})$

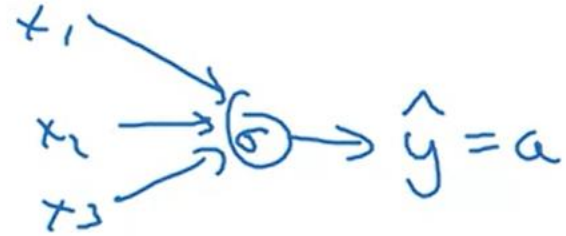
$a^{[0]}$

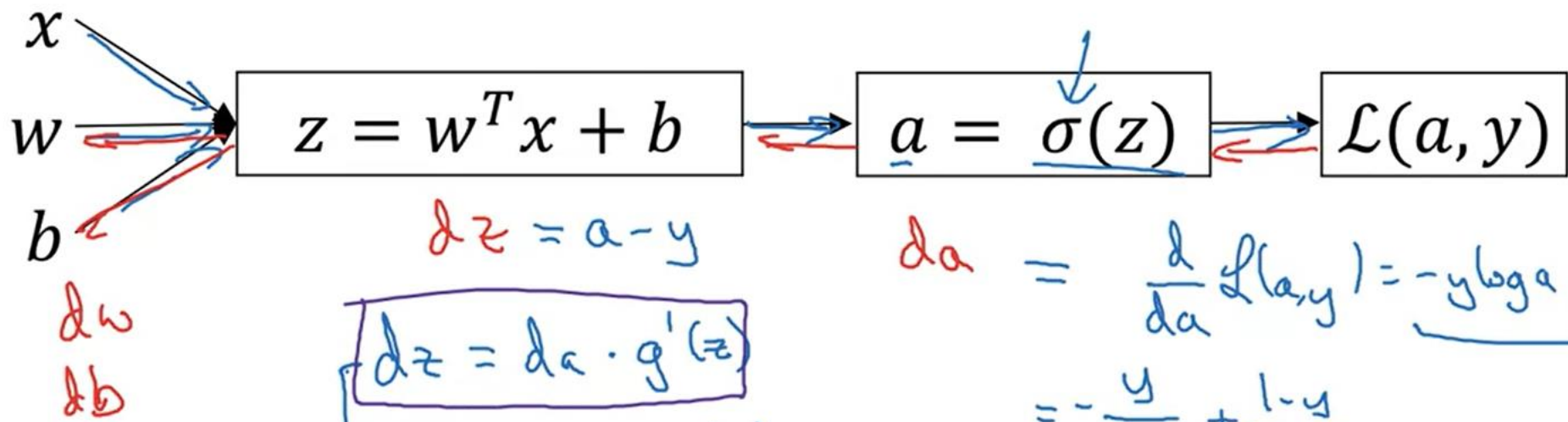
$A^{[0]}$

$X = A^{[0]}$ → [ ] → [ ] → [ ] →

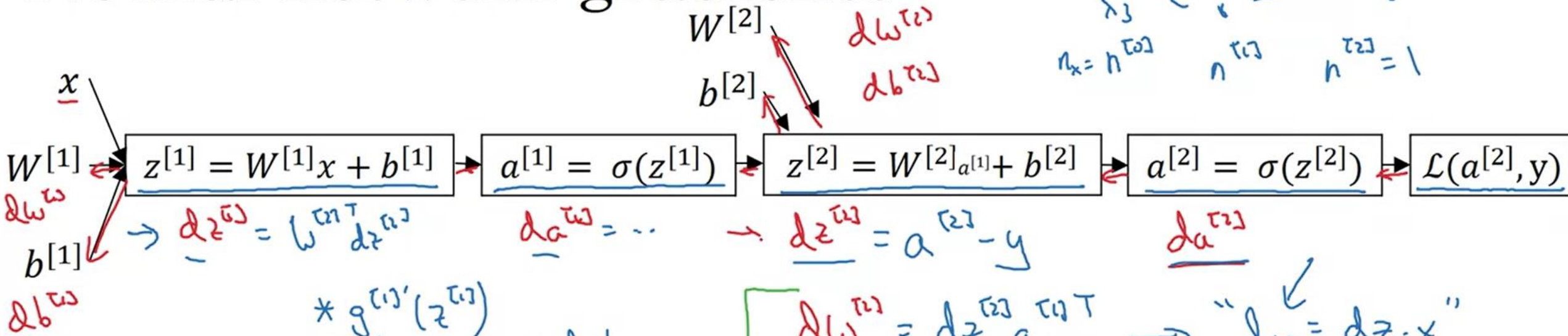
Computing gradients

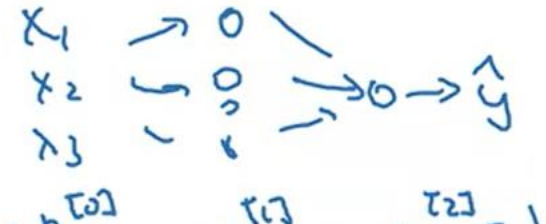
Logistic regression





Neural network gradients





$\rightarrow dz^{[1]} = W^{[2]T} dz^{[2]}$

$da^{[1]} = \ldots$

$\rightarrow dz^{[2]} = a^{[2]} - y$

$da^{[2]}$

$* g^{[1]\prime}(z^{[1]})$

↳ element-wise product

$$\begin{boxed} dW^{[2]} = dz^{[2]} a^{[1]T} \\ db^{[2]} = dz^{[2]} \end{boxed}$$

$\rightarrow$ "$dW = dz \cdot x$"

$W^{[1]}$: [——]

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

$\rightarrow z^{[2]}, dz^{[2]}$ $(n^{[2]}, 1)$ $-(1,1)$

$\rightarrow z^{[1]}, dz^{[1]}$ $(n^{[1]}, 1)$

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

$(n^{[1]}, 1) = (n^{[1]}, n^{[2]})\ (n^{[2]}, 1) * (n^{[1]}, 1)$

foo $W^{[1]}$

dfoo $dW^{[2]}$

$$\begin{boxed} dW^{[1]} = dz^{[1]} \cdot x^T \\ db^{[1]} = dz^{[1]} \end{boxed}$$

$x^T = a^{[0]T}$

Andrew Ng

Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

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

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

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$

$$J(\cdot) = \frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\hat{y}, y)$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

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

$(n^{[1]}, m)$ $(n^{[1]}, m)$ $(n^{[1]}, m)$

element-wise product

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

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

Andrew Ng