Neural Networks Basics

CSE 4237 - Soft Computing

Mir Tafseer Nayeem
Faculty Member, CSE AUST
tafseer.nayeem@gmail.com

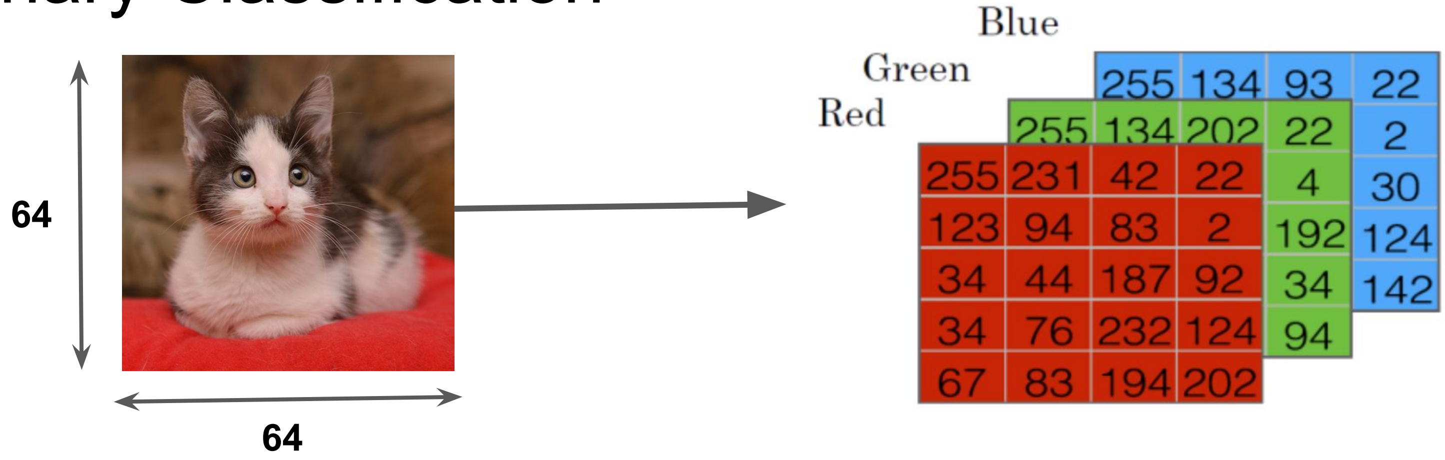
Binary Classification

1 (Cat) vs 0 (non-Cat)

Example: Cat vs Non-Cat

- The goal is to **train a classifier** with an input image represented by a **feature vector** x.
- To predict whether the corresponding label y is 1 or 0.
- In this case, whether this is a cat image (1) or a non-cat image (0).

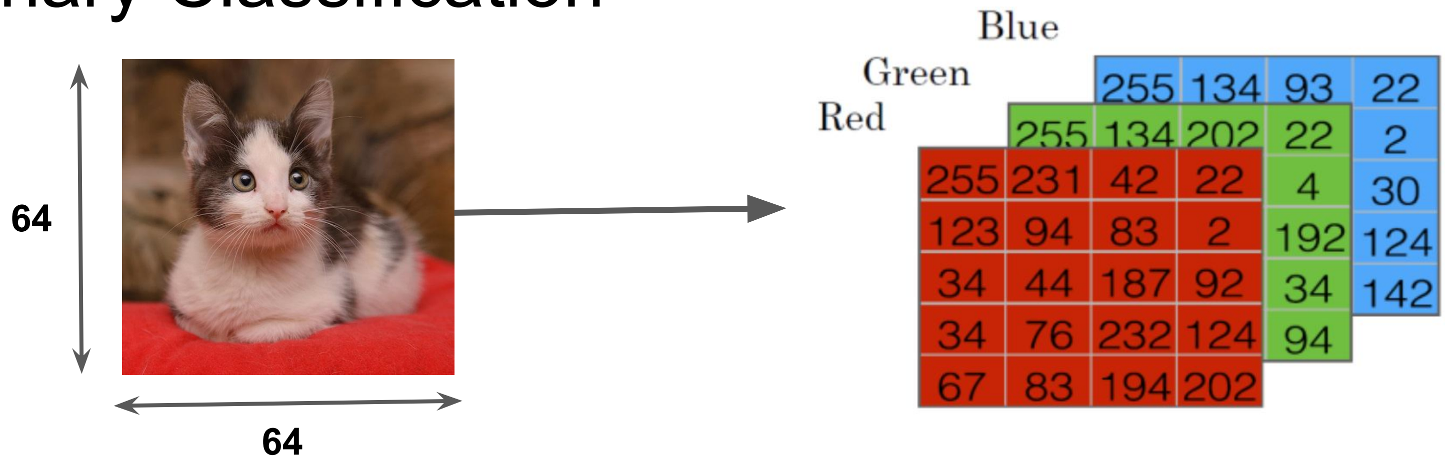
Binary Classification



- An image is stored in the computer in three separate matrices corresponding to the **Red, Green, and Blue** color channels of the image.
- The three matrices have the **same size as the image**, for example, the resolution of the cat image is **64 pixels X 64 pixels**, the **three matrices (RGB) are 64 X 64 each.**

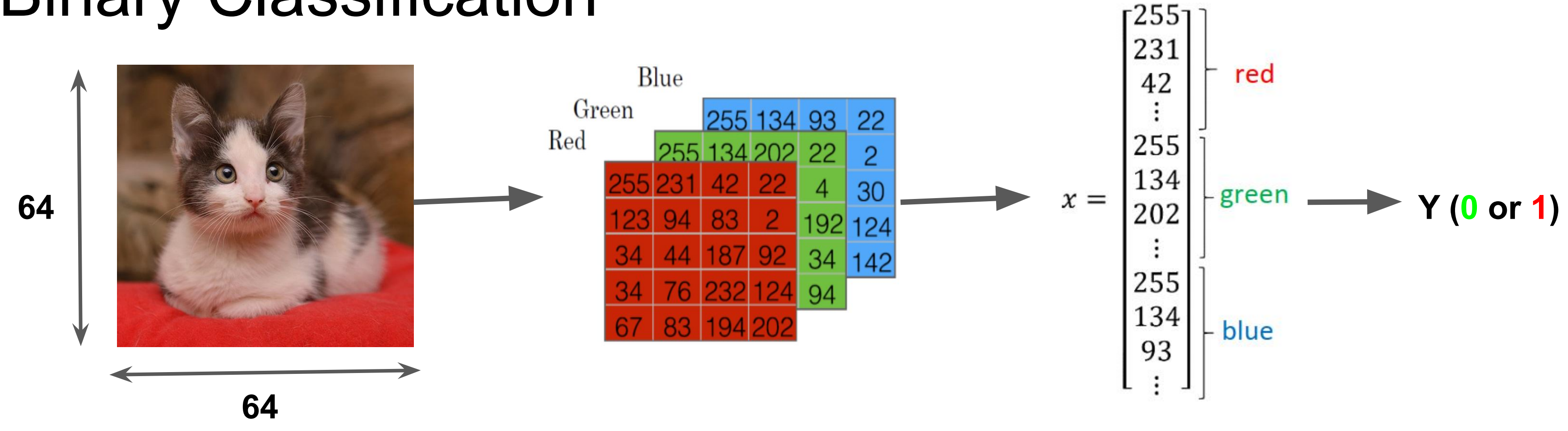
Content Credit: Andrew Ng

Binary Classification



- The value in **a cell represents the pixel intensity** which will be used to create **a feature vector of n dimension**. In pattern recognition and machine learning, a feature vector represents an object, in this case, a cat or no cat.

Binary Classification



- To create a feature vector, x, the pixel intensity values will be “unroll” or “reshape” for each color. The dimension of the input feature vector x is $\boldsymbol{n\_x}$ **= 64 $\boldsymbol{x}$ 64 $\boldsymbol{x}$ 3 = 12 288**.

Logistic Regression

- Logistic regression is a learning algorithm used in a supervised learning problem when the output y are all either zero or one.
- The goal of logistic regression is to minimize the error between its predictions and training data.
- Given an image represented by a feature vector x, the algorithm will evaluate the probability of a cat being in that image.

$$Given\ x\ ,\ \ \hat{y} = P(y = 1|x), \text{where}\ 0\ \ \leq \hat{y} \leq 1$$

Logistic Regression

The parameters used in Logistic regression are:

- The input features vector: $x \in \mathbb{R}^{n_x}$, where n_x is the number of features
- The training label: $y \in 0,1$
- The weights: $w \in \mathbb{R}^{n_x}$, where n_x is the number of features
- The bias $b \in \mathbb{R}$
- The output: $\hat{y} = \sigma(w^T x + b)$
- Sigmoid function: s = $\sigma(w^T x + b)$ = $\sigma(z)$= $\frac{1}{1+ e^{-z}}$

Parameters

Content Credit: Andrew Ng

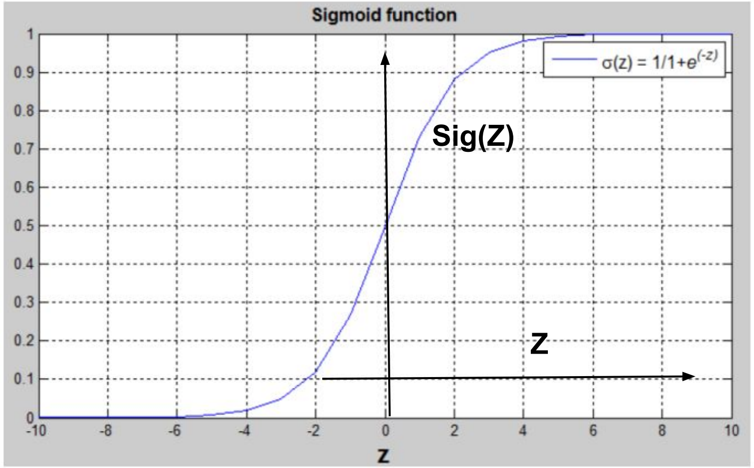
Logistic Regression : Role of bias (b)

up & down

- The bias value **allows the activation function to be shifted to the left or right**, to better fit the data.
- Changes to the weights **alter the steepness of the sigmoid curve**, whilst the bias offsets it, shifting the entire curve so it fits better.
- **Bias only influences the output values**, it doesn't interact with the actual input data. That's why it is called bias.
- You can think of the bias as a measure of **how easy it is to get a node to fire**.
    - **For a node with a large bias**, the output will tend to be intrinsically high, with small positive weights and inputs producing large positive outputs (near to 1).
    - **Biases can be also negative**, leading to sigmoid outputs near to 0.
    - **If the bias is very small (or 0)**, the output will be decided by the values of weights and inputs alone.

Logistic Regression

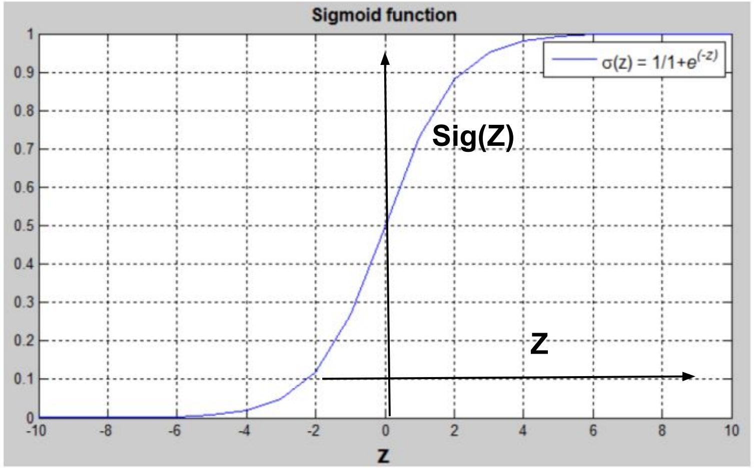
- $(wTx + b)$ is a linear function like $(ax + b)$, but since we are looking for a probability constraint between [0,1], the sigmoid function is used.
- The function is bounded between [0,1] as shown in the graph below.



Logistic Regression

Some observations from the graph:

- If z is a large positive number, then $\sigma(z)$ = 1
- If z is small or large negative number, then $\sigma(z)$ = 0
- If $z = 0$, then $\sigma(z)$ = 0.5



Logistic Regression: Cost Function

- To train the parameters w and b, we need to define a cost function.

$$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b), \text{ where } \sigma(z^{(i)}) = \frac{1}{1 + e^{-z^{(i)}}}$$

$$Given \{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\}, \; we\ want\ \hat{y}^{(i)} \approx y^{(i)}$$

$x^{(i)}$ the i-th training example

Loss (error) function:

- Loss function measures the discrepancy between the prediction ($\hat{y}(i)$) and the desired output ($y(i)$).
- In other words, the loss function computes the error for a single training example.

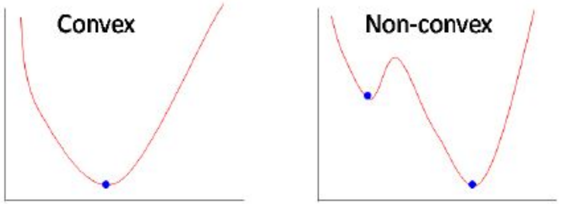
Error / Loss Function

Squared Error Function: $L\left(\hat{y}^{(i)}, y^{(i)}\right) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2$

- We can see an extra (1/2) in the right side of the equation. Does it matter?
- It is because when you take the derivative of the cost function, that is used in updating the parameters during gradient descent, that 2 in the power get cancelled with the (1/2) multiplier.
- These techniques are or somewhat similar are widely used in math in order **"To make the derivations mathematically more convenient"**.

Is squared error function a good choice?

- The squared error function (commonly used function for linear regression) is not very suitable for logistic regression.
    - In case of logistic regression, the hypothesis / prediction is non-linear (sigmoid function), which makes the square error function to be non-convex.
    - On the other hand, logarithmic function is a convex function for which there is no local optima, so gradient descent works well.
- If you are doing binary classification, squared error function generally also penalize examples that **are correctly classified but are still near the decision boundary**, thus creating a **"margin."**
- Gradient descent waste a lot of time getting predictions very close to {0, 1}



Logistic Regression: Cross Entropy Loss

$$L\left(\hat{y}^{(i)}, y^{(i)}\right) = -\left(y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log(1 - \hat{y}^{(i)}) \right)$$

- If $y^{(i)} = 1$: $L\left(\hat{y}^{(i)}, y^{(i)}\right) = -\log\left(\hat{y}^{(i)}\right)$ where $\log\left(\hat{y}^{(i)}\right)$ and $\hat{y}^{(i)}$ should be close to 1
- If $y^{(i)} = 0$: $L\left(\hat{y}^{(i)}, y^{(i)}\right) = -\log\left(1 - \hat{y}^{(i)}\right)$ where $\log\left(1 - \hat{y}^{(i)}\right)$ and $\hat{y}^{(i)}$ should be close to 0

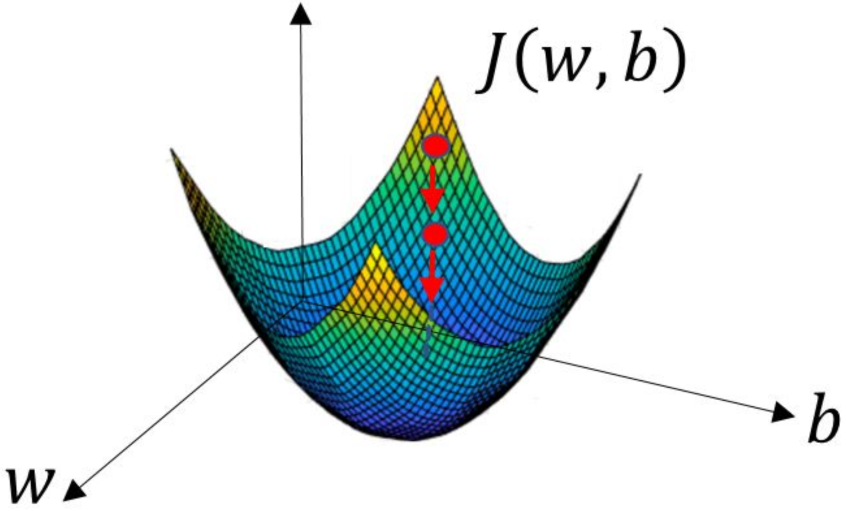
- **Cost function**
  - The cost function is the average of the loss function of the entire training set. We are going to find the parameters w *and* b that minimize the overall cost function.

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right) = -\frac{1}{m} \sum_{i=1}^{m} \left[\left(y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log(1 - \hat{y}^{(i)}\right)\right]$$

Gradient Descent

$$\text{Recap: } \hat{y} = \sigma(w^T x + b), \ \sigma(z) = \frac{1}{1+e^{-z}}$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$



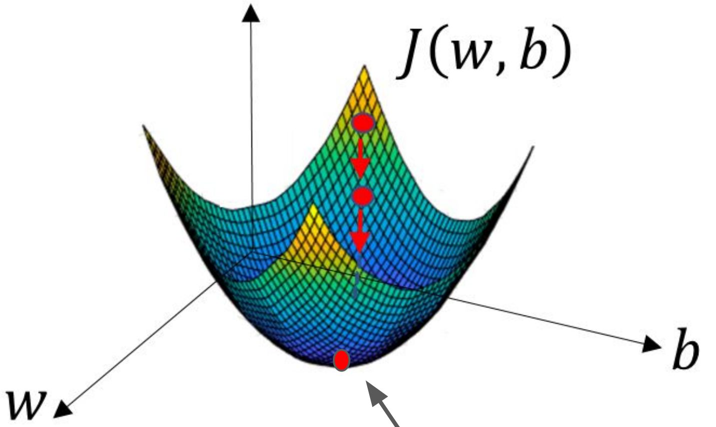
We want to find parameters **W, b** that minimize **J(W, b)**

Content Credit: Andrew Ng

Gradient Descent

- Our cost function is convex.
- First we initialize **w and b to 0,0** or initialize them to a **random value** in the convex function and then try to improve the values the reach minimum value.
- In Logistic regression people **always use 0,0 instead of random**.
- **This function is convex**, no matter where you initialize you should get to the **global optimal point or roughly close the global optimal point**.

We want to parameters **W, b** that minimize **J(W, b)**

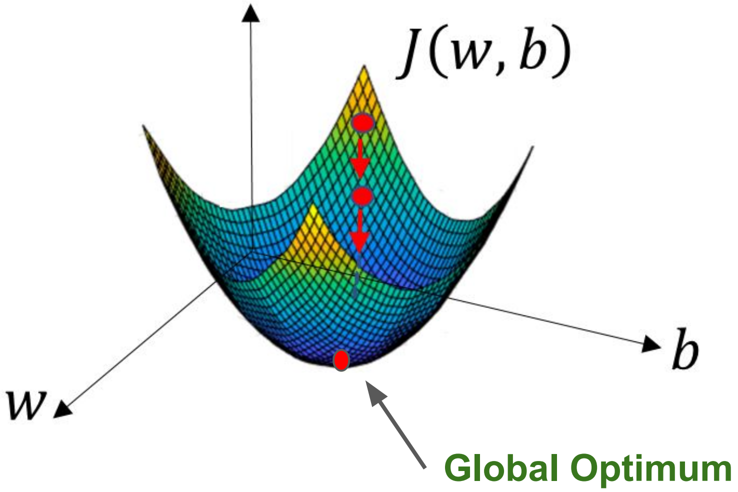


Global Optimum

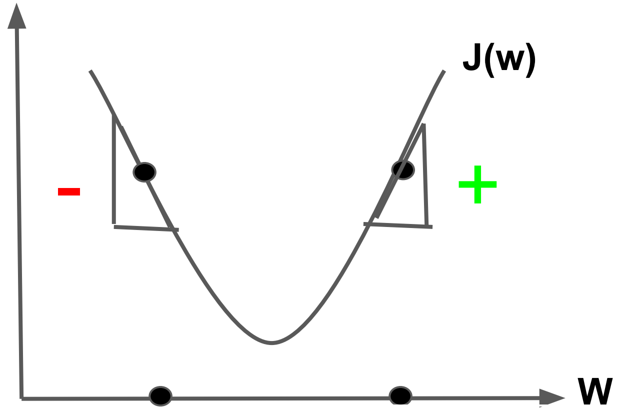
Gradient Descent

- Gradient starts at the initial point and take a step in the steepest downhill direction after each iteration.
- It will try to reach to the global optimum or somewhere near to the global optimum.

We want to parameters **W, b** that minimize **J(W, b)**



Gradient Descent



*Ignore **b** for now to make it a one dimensional plot rather than a higher dimensional plot.*

- α = **Learning Rate**: How bigger step we choose at each iteration of gradient descent.
- Definition of a derivative:
  - Slope of a function at a point.

// Repeatedly do that until the algorithm converges.

Repeat {

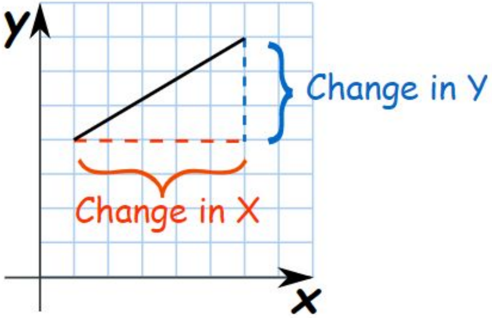
Learning Rate

$$w := w - \alpha \frac{dJ(w)}{dw}$$

Update or change you want to make to the parameter w

}

$$\text{Slope} = \frac{\text{Change in Y}}{\text{Change in X}} = \frac{\Delta y}{\Delta x}$$



Gradient Descent : Actual Update Rule

We want to parameters
W, b that minimize **J(W, b)**

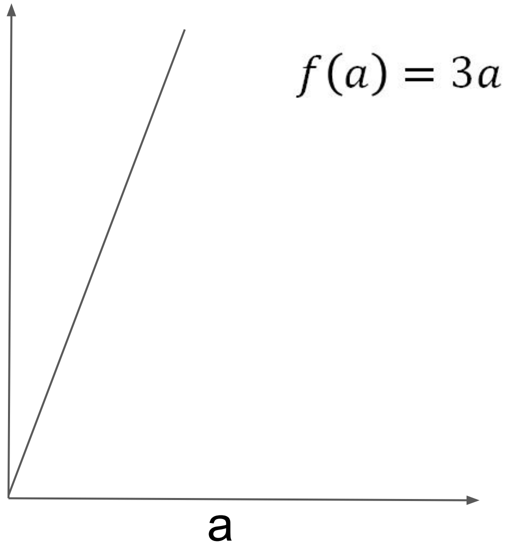
Partial Derivative

J(w,b)

$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

Derivatives : Intuition



- a = 2 f (a) = 6
  a = 2.001 f (a) = 6.003

  Slope (derivative) of f (a) at a = 2 is 3

- a = 5 f (a) = 15
  a = 5.001 f (a) = 15.003

  Slope (derivative) of f (a) at a = 5 is also 3

If we shift **a** by 0.001 then **f (a)** shift by 3 times 0.001.

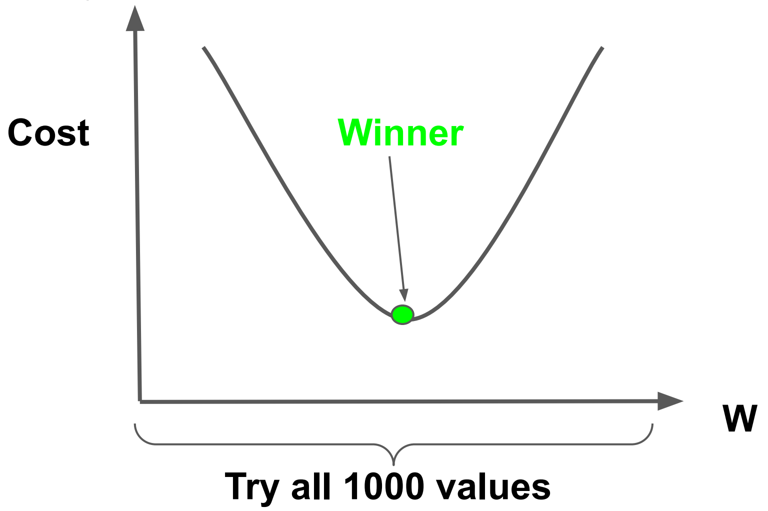
$$\boxed{\frac{d\,f(a)}{da} = 3}$$

What does $\frac{d}{dx}x^2 = 2x$ mean?

The slope or "rate of change" at any point is **2x**.

Do we actually need Gradient Descent?

- Let's pretend that we only have 1 weight. To find the ideal value of our weight that will minimize our cost, we need to try a bunch of values for W, let's say we test 1000 values. That doesn't seem so bad, after all, my computer is pretty fast.

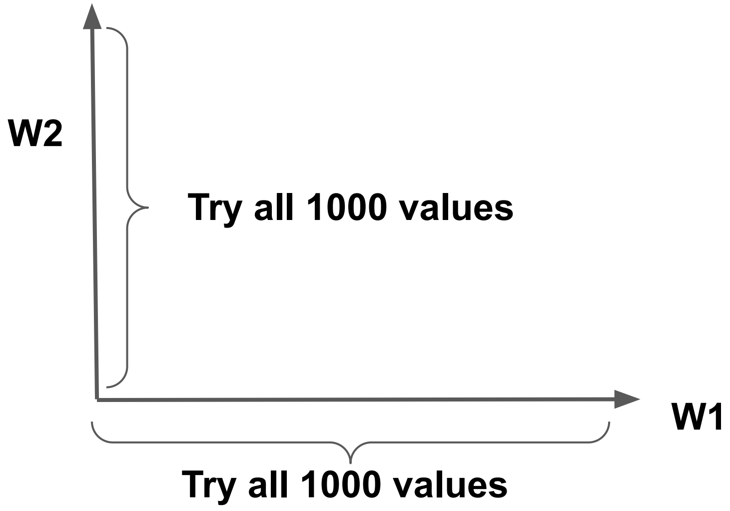


- It takes about **0.04 seconds** to check **1000 different weight** values for our neural network.

- Since we've computed the cost for a wide range values of W, we can **just pick the one with the smallest cost**.

Do we actually need Gradient Descent?

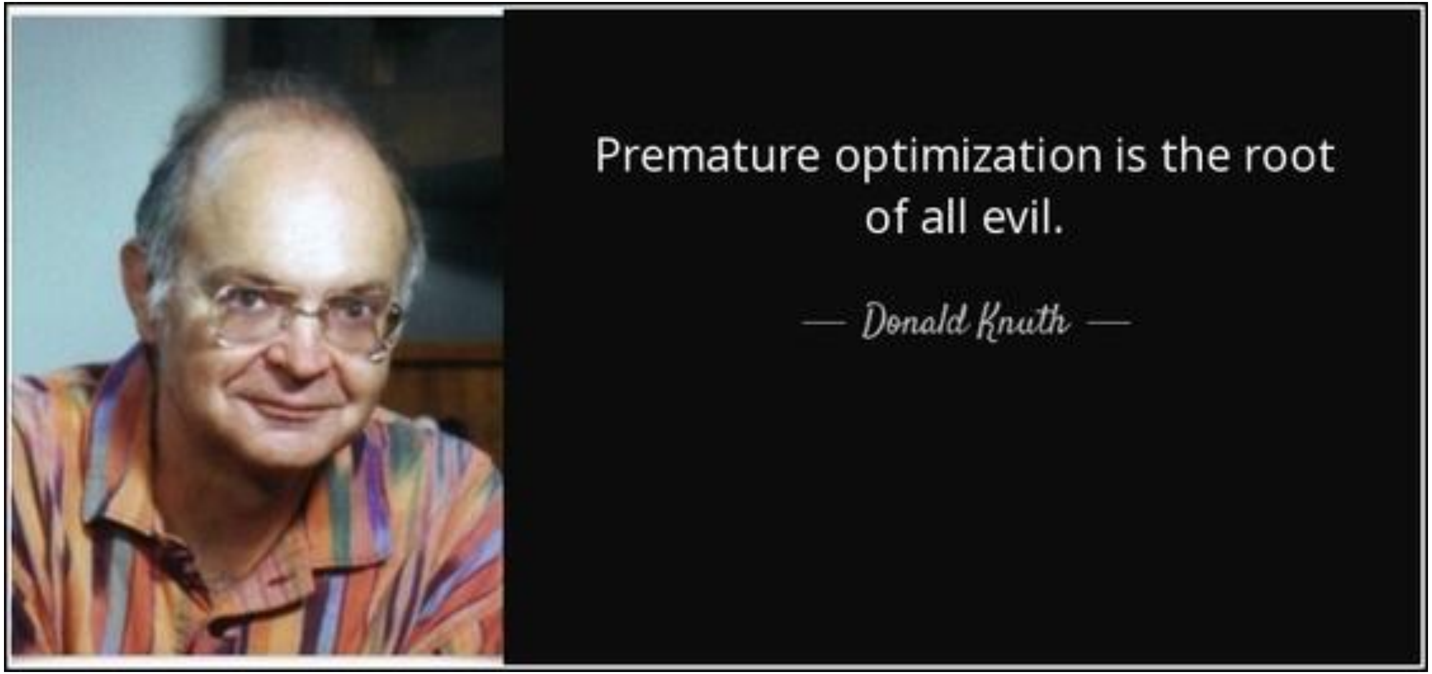
- Let's next consider 2 weights for a moment. To maintain the same precision we now need to check **1000 times 1000, or one million values**. This is a lot of work, even for a fast computer.



- After our 1 million evaluations we've found our solution, but it took an **agonizing 40 seconds!** Searching through **three weights would take a billion evaluations, or 11 hours!**

- Searching through all 9 weights we need for our simple network would take **1,268,391,679,350,583.5 years. (Over a quadrillion years)**. So for that reason, the ***"just try everything"*** or **brute force optimization method** is clearly not going to work.

A Famous Quote



Computation Graph

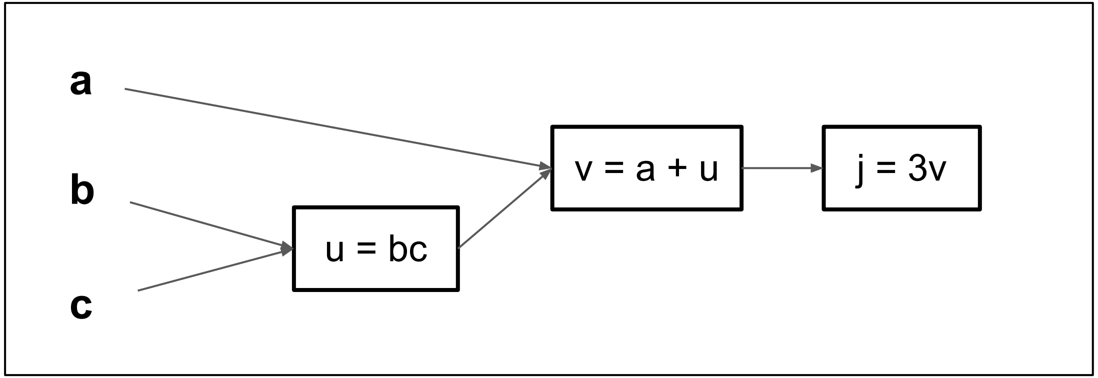
- Neural Networks are organized in terms of a forward pass or backward pass.
- Forward Pass / Propagation
  - Which we compute the output of the neural network
- Backward Pass / Propagation
  - Which we use to compute gradients / derivatives
- **Computation Graph**
  - Explains why it is organized in this way.

Computation Graph

$J(a, b, c) = 3(a + bc)$

3 steps of computation:

1. $u = bc$
2. $v = a + u$
3. $j = 3v$



Logistic regression : Forward Propagation

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

$$\hat{y} = a = \sigma(z)$$

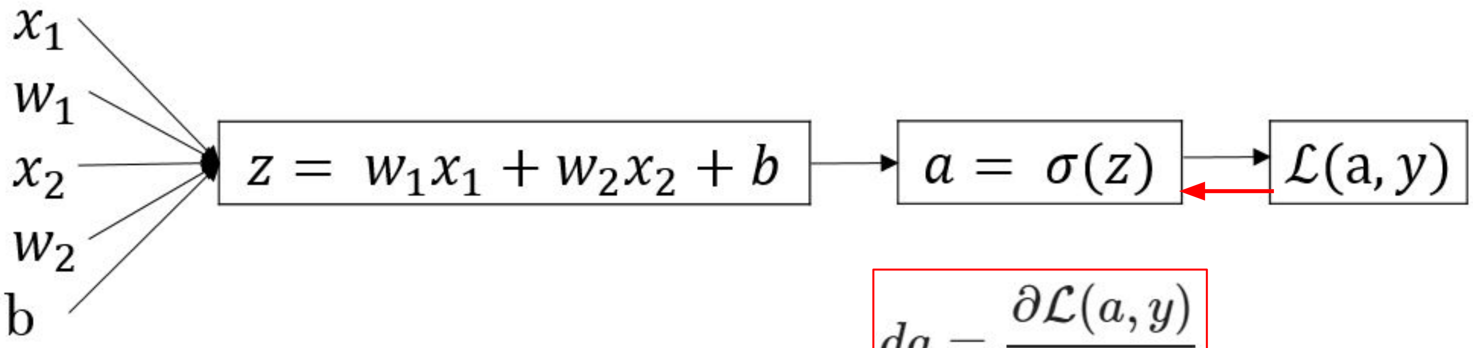
$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

Computing loss of a single training example

$$x_1,\ w_1,\ x_2,\ w_2,\ b \rightarrow \boxed{z = w_1 x_1 + w_2 x_2 + b} \rightarrow \boxed{a = \sigma(z)} \rightarrow \boxed{\mathcal{L}(a, y)}$$

Modify the parameters w1, w2 and b in order to minimize the loss

Logistic regression : Backward Propagation



$$\boxed{da = \frac{\partial \mathcal{L}(a,y)}{\partial a}}$$

Rules for derivatives of logarithmic expressions

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

$$\frac{\delta}{\delta x} \log(\text{expression}) = \frac{1}{\text{expression}} \cdot \frac{\delta}{\delta x} \text{expression}$$

Examples:

$$\frac{\delta}{\delta x} \log(x) = \frac{1}{x} \cdot \frac{\delta}{\delta x} x = \frac{1}{x} \cdot 1 = \frac{1}{x}$$

If you are unsure about your derivative check this link to generate the derivation steps.

Logistic regression : Backward Propagation

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

Ignoring the (-) sign for now.

$$\frac{d}{da}[y\ln(a) + (1-y)\ln(1-a)]$$

$$= y \cdot \frac{d}{da}[\ln(a)] + (1-y) \cdot \frac{d}{da}[\ln(1-a)]$$

$$= y \cdot \frac{1}{a} + (1-y) \cdot \frac{1}{1-a} \cdot \frac{d}{da}[1-a]$$

$$= \frac{y}{a} + \frac{(1-y)\left(\frac{d}{da}[1] - \frac{d}{da}[a]\right)}{1-a}$$

log (x) refers to **e base log or the natural logarithm (ln(x))** in mathematical analysis, physics, chemistry, statistics, economics, and some engineering fields.

Logistic regression : Backward Propagation

$$= \frac{y}{a} + \frac{(1-y)\left(\frac{\mathrm{d}}{\mathrm{d}a}[1] - \frac{\mathrm{d}}{\mathrm{d}a}[a]\right)}{1-a}$$

$$= \frac{y}{a} + \frac{(1-y)(0-1)}{1-a}$$

$$= \frac{y}{a} + \frac{y-1}{1-a}$$

$$= \frac{y}{a} - \frac{1-y}{1-a}$$

Finally, adding the (-) sign.

$$\boxed{= -\frac{y}{a} + \frac{1-y}{1-a}}$$

Logistic regression : Backward Propagation

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

$$\hat{y} = a = \sigma(z)$$

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

x_1, w_1, x_2, w_2, b

$$z = w_1 x_1 + w_2 x_2 + b \rightarrow a = \sigma(z) \rightarrow \mathcal{L}(a, y)$$

$$dz = \frac{\partial \mathcal{L}(a,y)}{\partial z} = \frac{\partial \mathcal{L}}{\partial z}$$

$$da = \frac{\partial \mathcal{L}(a,y)}{\partial a}$$

Applying Chain Rule

$$= \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial z}$$

$$= -\frac{y}{a} + \frac{1-y}{1-a}$$

Logistic regression : Backward Propagation

$$\boxed{a = \sigma(z) = \frac{1}{1+e^{-z}}}$$

$$\begin{aligned}
\frac{\partial a}{\partial z} &= \frac{\partial}{\partial z}\sigma(z) \\
&= \frac{\partial}{\partial z}\left[\frac{1}{1+e^{-z}}\right] \\
&= \frac{\partial}{\partial z}\left(1+\mathrm{e}^{-z}\right)^{-1} \\
&= -(1+e^{-z})^{-2}(-e^{-z}) \\
&= \frac{e^{-z}}{(1+e^{-z})^2} \\
&= \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}}
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{1+e^{-z}} \cdot \frac{(1+e^{-z})-1}{1+e^{-z}} \\
&= \frac{1}{1+e^{-z}} \cdot \left(\frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}\right) \\
&= \frac{1}{1+e^{-z}} \cdot \left(1 - \frac{1}{1+e^{-z}}\right) \\
&= \sigma(z) \cdot (1-\sigma(z)) \\
&\boxed{= a \cdot (1-a)}
\end{aligned}$$

Logistic regression : Backward Propagation

$$\begin{tikzcd} x_1, w_1, x_2, w_2, b \arrow[r] & \boxed{z = w_1x_1 + w_2x_2 + b} \arrow[r, shift left] & \boxed{a = \sigma(z)} \arrow[l, shift left] \arrow[r, shift left] & \boxed{\mathcal{L}(a, y)} \arrow[l, shift left] \end{tikzcd}$$

$$dz = \frac{\partial \mathcal{L}(a,y)}{\partial z} = \frac{\partial \mathcal{L}}{\partial z}$$

$$da = \frac{\partial \mathcal{L}(a,y)}{\partial a}$$

$$= -\frac{y}{a} + \frac{1-y}{1-a}$$

Applying Chain Rule

$$= \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial z}$$

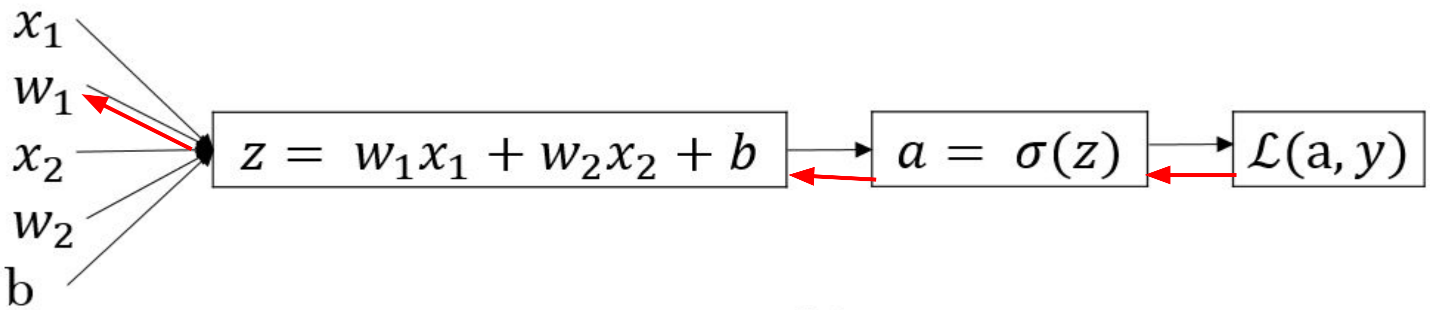
$$= \left(-\frac{y}{a} + \frac{1-y}{1-a}\right) * (a \cdot (1-a))$$

$$= -y * (1-a) + a * (1-y)$$

$$= -y + ay + a - ay$$

$$\boxed{= a - y}$$

Logistic regression : Backward Propagation



$$dw_1 = \frac{\partial \mathcal{L}(a,y)}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} . \frac{\partial a}{\partial z}$$

$$\boxed{dz = a - y}$$

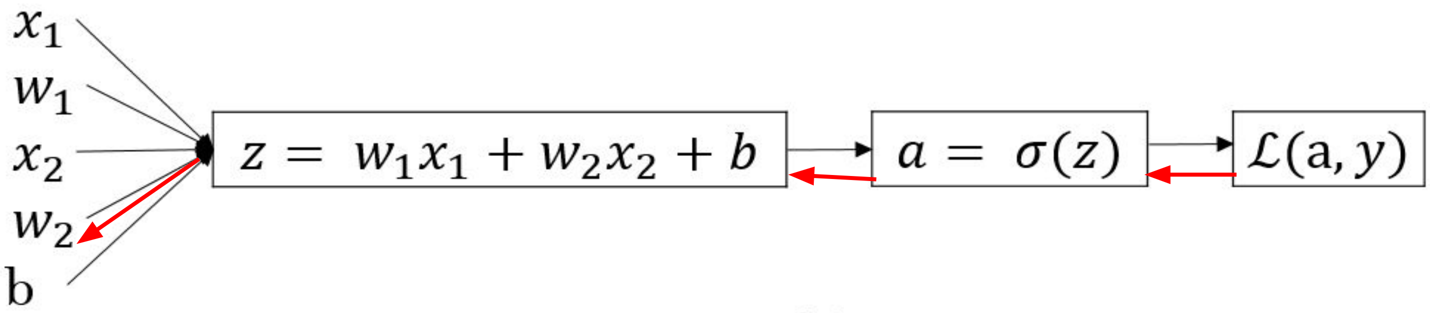
Applying Chain Rule

$$= \frac{\partial \mathcal{L}}{\partial z} . \frac{\partial z}{\partial w_1}$$

$$= dz * \frac{\partial}{\partial w_1}(w_1 x_1 + w_2 x_2 + b)$$

$$\boxed{dw_1 = dz * x_1}$$

Logistic regression : Backward Propagation



$$dw_2 = \frac{\partial \mathcal{L}(a,y)}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial w_2}$$

Applying Chain Rule

$$= \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial w_2}$$

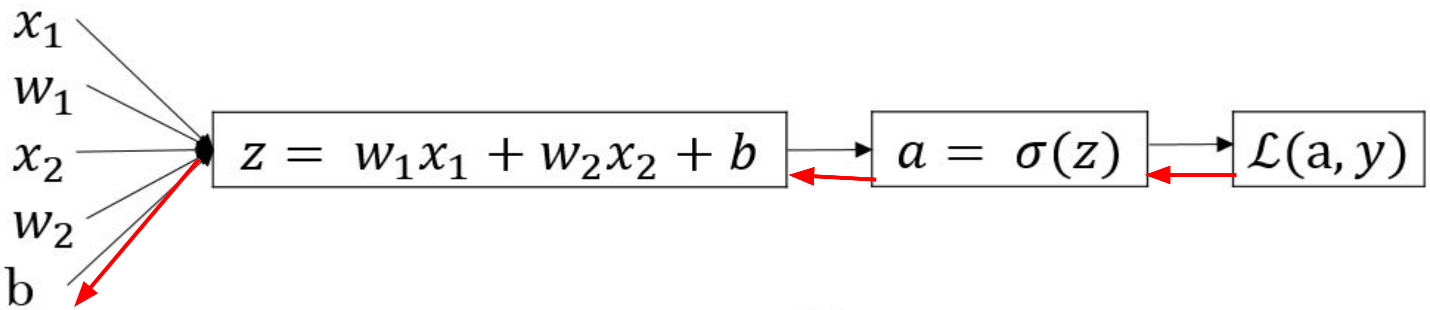
$$= dz * \frac{\partial}{\partial w_2}(w_1x_1 + w_2x_2 + b)$$

$$\boxed{dw_2 = dz * x_2}$$

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial z}$$

$$\boxed{dz = a - y}$$

Logistic regression : Backward Propagation



$$db = \frac{\partial \mathcal{L}(a,y)}{\partial b} = \frac{\partial \mathcal{L}}{\partial b}$$

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} . \frac{\partial a}{\partial z}$$

$$\boxed{dz = a - y}$$

Applying Chain Rule

$$= \frac{\partial \mathcal{L}}{\partial z} . \frac{\partial z}{\partial b}$$

$$= dz * \frac{\partial}{\partial b}(w_1x_1 + w_2x_2 + b)$$

$$\boxed{db = dz}$$

Updating the Parameters: w1, w2 and b

This is one step of Gradient Descent on a single example.

$$w_1 := w_1 - \alpha * dw_1$$

$$w_2 := w_2 - \alpha * dw_2$$

$$b := b - \alpha * db$$

Learning Rate

Logistic regression Gradient descent on m examples

Basic Parameters

x1	Feature
x2	Feature
w1	Weight of the first feature.
w2	Weight of the second feature.
b	Logistic Regression parameter (Bias).
m	Number of training examples
y(i)	Expected output of i

Logistic regression Gradient descent on m examples

$$z^{(i)} = w^T x^{(i)} + b$$

$$\hat{y}^{(i)} = a^{(i)} = sigmoid(z^{(i)})$$

$$\mathcal{L}(a^{(i)}, y^{(i)}) = -y^{(i)} \log(a^{(i)}) - (1 - y^{(i)}) \log(1 - a^{(i)})$$

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

$$\frac{\partial}{\partial w_1} J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)})}_{dw_1^{(i)}}$$

For the example $(x^{(i)}, y^{(i)})$

Logistic regression Gradient descent on m examples

Derivatives: All it turned out as simple arithmetic operations

d(a)	- (y/a) + ((1-y) / (1-a))
d(z)	a - y
d(w1)	x1 * d(z)
d(w2)	x2 * d(z)
d(b)	d(z)

Logistic regression Gradient descent on m examples

```
J = 0; dw1 = 0; dw2 =0; db = 0;
w1 = 0; w2 = 0; b=0;

for i = 1 to m
        # Forward pass
        z(i) = w1*x1(i) + w2*x2(i) + b
        a(i) = sigmoid(z(i))
        J += (y(i)*log(a(i)) + (1-y(i))*log(1-a(i)))

        # Backward pass
        dz(i) = a(i) - y(i)
        dw1 += dz(i) * x1(i)
        dw2 += dz(i) * x2(i)
        db  += dz(i)
```

n = 2

```
J /= m
dw1 /= m
dw2 /= m
db /= m

# Gradient descent
w1 = w1 - alpha * dw1
w2 = w2 - alpha * dw2
b = b - alpha * db
```

w1, w2, b are the accumulators and single instances for the all **m** training examples.

One iteration of gradient descent

Logistic regression Gradient descent on m examples

- Previous slide is **just one step of Gradient Descent**, we need to repeat it **multiple times** in order to take multiple steps of gradient descent.
- There are **weaknesses** in the previous implementation. In order to implement we need to **write two for loops**.
- Having explicit for loops in your code make your **code less efficient**.
- Solution:- **vectorization techniques**
- To train with larger datasets we need to take the help from vectorization techniques **without using for loops**.

LR Gradient descent on m examples (modified)

```
J = 0; dw1 = 0; dw2 = 0; db = 0;
w1 = 0; w2 = 0; b = 0;

for i = 1 to m
        # Forward pass
        z(i) = w1*x1(i) + w2*x2(i) + b
        a(i) = sigmoid(z(i))
        J += - (y(i)*log(a(i)) + (1-y(i))*log(1-a(i)))

        # Backward pass
        dz(i) = a(i) - y(i)
        dw1 += dz(i) * x1(i)
        dw2 += dz(i) * x2(i)
        db  += dz(i)
```

```
dw = np.zeros ((nx, 1))
```

n = 2

```
dw += x(i) * dz(i)
```

Logistic regression Gradient descent on m examples

J = J / m

$\boxed{\begin{array}{l}\text{dw1 = dw1 / m}\\ \text{dw2 = dw2 / m}\end{array}} \longrightarrow \boxed{\text{dw = dw / m}}$

db = db / m

 # Gradient descent

w1 = w1 - *alpha* * dw1

w2 = w2 - *alpha* * dw2

b = b - *alpha* * db

w1, w2, b are the accumulators and single instances for the all **m** training examples.

We have gone from 2 for loops to 1 for loop, we still have one for loop that loops over individual training examples.

Vectorizing Logistic Regression (Forward)

$$z^{(1)} = w^T x^{(1)} + b \qquad z^{(2)} = w^T x^{(2)} + b \qquad z^{(3)} = w^T x^{(3)} + b$$

$$a^{(1)} = \sigma(z^{(1)}) \qquad a^{(2)} = \sigma(z^{(2)}) \qquad a^{(3)} = \sigma(z^{(3)})$$

1st training example | 2nd training example | 3rd training example

We need to do it m times if you have **m training examples**.

$$\mathbf{X} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix} \text{ while } \mathbb{R}^{n_x \times m}$$

$$\boxed{\begin{bmatrix} & w^T & \end{bmatrix} \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}}$$

$$[z^{(1)}\ z^{(2)}\ z^{(3)} \dots z^{(m)}] = w^T X + [\, b\ b\ \dots b] = [w^T x^{(1)} + b \quad w^T x^{(2)} + b \quad \dots \quad w^T x^{(m)} + b]$$

(1 x m) dimension

(1 x m)

Vectorizing Logistic Regression (Forward)

$$[z^{(1)}\ z^{(2)}\ z^{(3)} \ldots z^{(m)}] = w^T X + [\ b\ b\ \ldots b] = [w^T x^{(1)} + b \quad w^T x^{(2)} + b \ \ldots \quad w^T x^{(m)} + b]$$

Broadcasting

$$Z = [z^{(1)}\ z^{(2)}\ z^{(3)} \ldots z^{(m)}]$$

$$Z = np.dot(w.T, X) + b$$

(1, 1) dimension

(1 x m) dimension

$$A = [a^{(1)}\ a^{(2)}\ a^{(3)} \ldots a^{(m)}] = \sigma\,(Z)$$

Gradient Computation

$$dz^{(1)} = a^{(1)} - y^{(1)} \qquad dz^{(2)} = a^{(2)} - y^{(2)} \qquad dz^{(3)} = a^{(3)} - y^{(3)}$$

$$dZ = [dz^{(1)} dz^{(2)} \dots dz^{(m)}]$$

$$A = [a^{(1)}\ a^{(2)}\ a^{(3)} \dots a^{(m)}] \qquad Y = [y^{(1)}\ y^{(2)}\ y^{(3)} \dots y^{(m)}]$$

$$dZ = A - Y = [(a^{(1)} - y^{(1)})(a^{(2)} - y^{(2)})(a^{(3)} - y^{(3)}) \dots (a^{(m)} - y^{(m)})]$$

Gradient Computation

$$dw = 0$$
$$dw += X^{(1)} dz^{(1)}$$
$$dw += X^{(2)} dz^{(2)}$$
$$\vdots$$
$$dw += X^{(m)} dz^{(m)}$$
$$dw /= m$$

$$db = 0$$
$$db += dz^{(1)}$$
$$db += dz^{(2)}$$
$$\vdots$$
$$db += dz^{(m)}$$
$$db /= m$$

$$db = \frac{1}{m} \sum_{i=1}^{m} dz(i))$$

$$db = \frac{1}{m} np.\, sum(dZ)$$

$$dw = \frac{1}{m} X.\, dZ^T$$

$$dw = \frac{1}{m} \begin{bmatrix} \vdots & \vdots & & \vdots \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ \vdots & \vdots & & \vdots \end{bmatrix} \begin{pmatrix} dz^{(1)} \\ dz^{(2)} \\ \cdot \\ dz^{(m)} \end{pmatrix}$$

$$dw = \frac{1}{m} [X^{(1)} dz^{(1)} + X^{(2)} dz^{(2)} + \dots + X^{(m)} dz^{(m)}]$$

Implementing Logistic Regression

$$J = 0,\ dw_1 = 0,\ dw_2 = 0,\ db = 0$$

for i = 1 to m:

$$z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$
$$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$$
$$dz^{(i)} = a^{(i)} - y^{(i)}$$
$$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$$
$$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$$
$$db \mathrel{+}= dz^{(i)}$$

$$J = J/m,\ dw_1 = dw_1/m,\ dw_2 = dw_2/m$$
$$db = db/m$$

Single Iteration of Gradient Descent

$$Z = w^T X + b$$
$$Z = np.dot(w.T, X) + b$$
$$A = \sigma(Z)$$

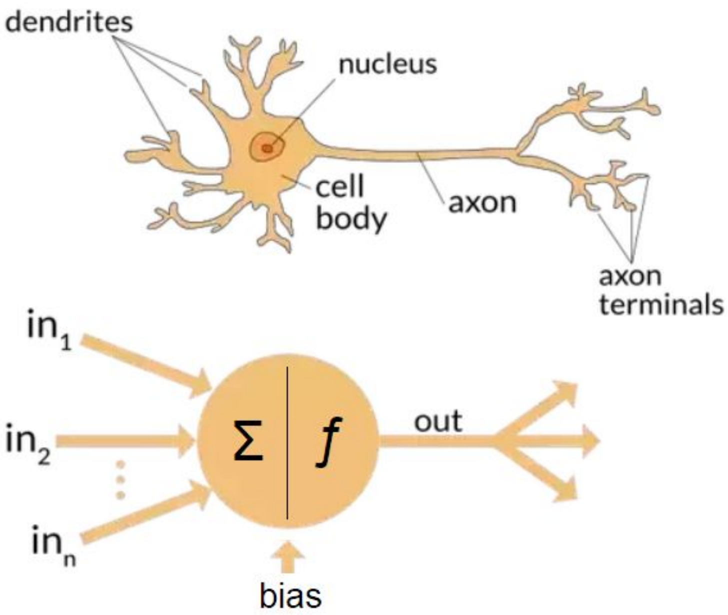
$$dZ = A - Y$$

$$dw = \frac{1}{m} X.dZ^T$$
$$db = \frac{1}{m} \sum_{i=1}^{m} dz(i))$$
$$db = \frac{1}{m} np.sum(dZ)$$

$$w := w - \alpha * dw$$
$$b := b - \alpha * db$$

Gradient Update

What does this have to do with the brain?



END