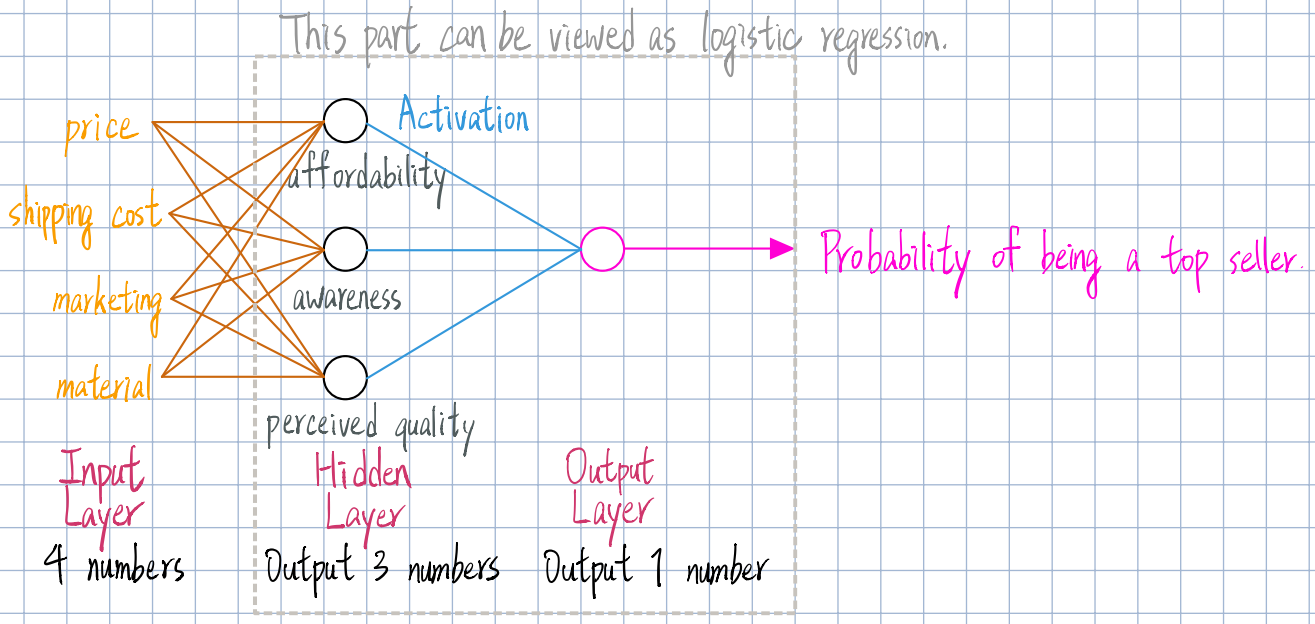
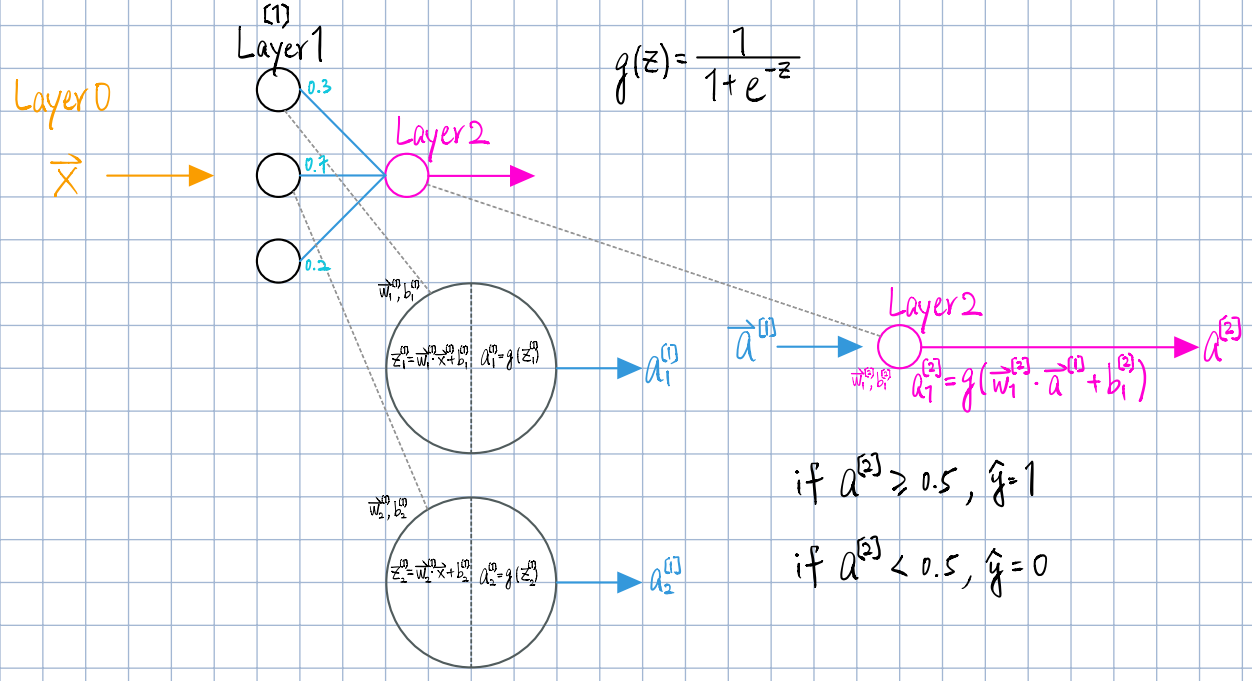
Start from a naive example. Given features of a T-shirt, we want to know if this T-shirt will become a top seller.



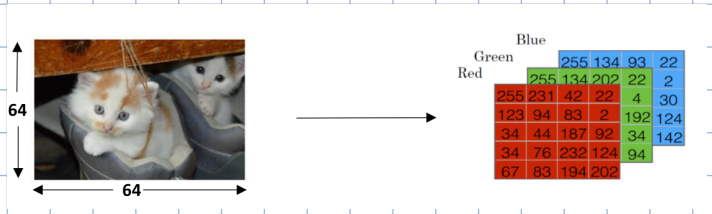
Here, we apply every connections between input layer and hidden layer so that the hidden features can be automatically learned. We don't need to hand-engineer features manually.

The above architecture is also called Multi Layer Perceptron (MLP).

The calculation is shown below. The graph can be simplified as:



Binary Classification



$$x = \begin{bmatrix} 255 \\ 231 \\ 42 \\ \vdots \\ 255 \\ 134 \\ 202 \\ \vdots \\ 255 \\ 134 \\ 93 \\ \vdots \end{bmatrix} \begin{matrix} \text{red} \\ \\ \\ \\ \text{green} \\ \\ \\ \\ \text{blue} \\ \\ \\ \end{matrix}$$

$64 \times 64 \times 3 = 12288$

n_x

the dimension of the input feature vector

Notation:

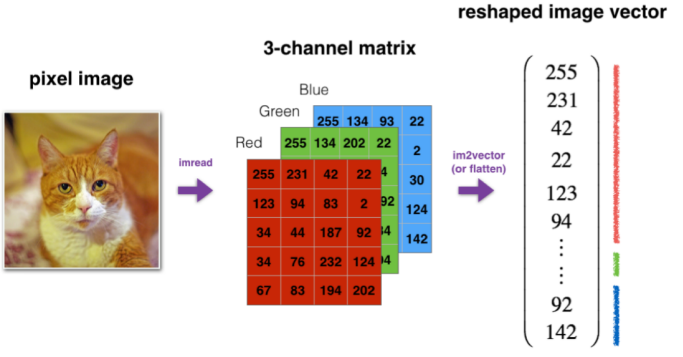
A single training example: (x, y), $x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

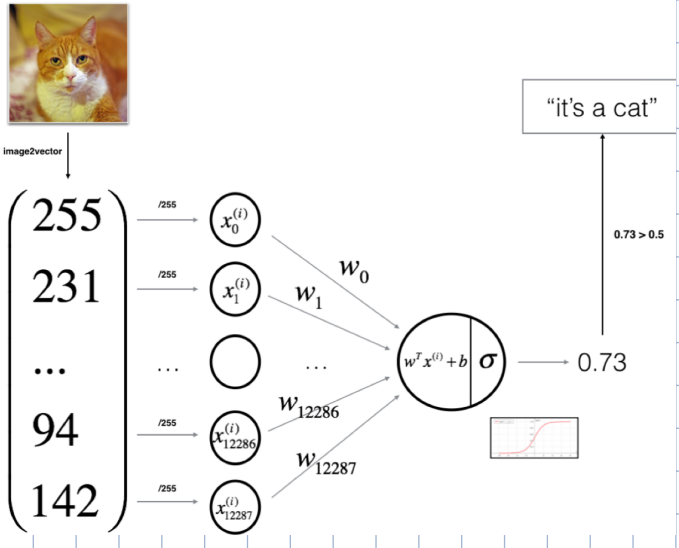
There are m training samples: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots\ldots, (x^{(m)}, y^{(m)})$

m_{train}

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n_x \times m}$$

$$Y = \begin{bmatrix} y^{(1)}, y^{(2)}, \cdots, y^{(m)} \end{bmatrix} \in \mathbb{R}^{1 \times m}$$





Logistic Regression: probability of the picture being a cat

Given x, we want $\hat{y} = P(y=1|x)$, $0 \le \hat{y} \le 1$

x: A picture of cat or non-cat

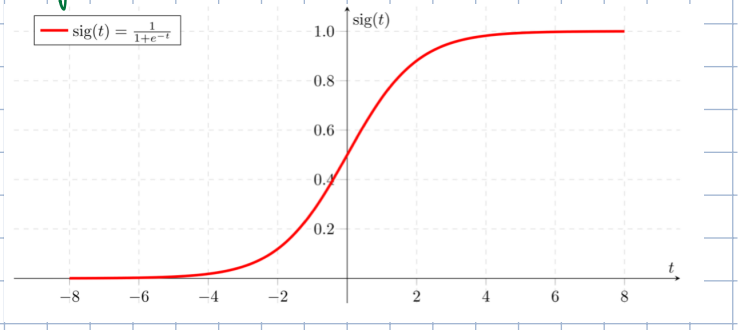
$\hat{y}$: estimated output

$x \in \mathbb{R}^{n_x}$

The parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

Output $\hat{y} = w^T x + b$ → This is a bad algorithm. We need $\hat{y}$ to be $0 \le \hat{y} \le 1$, but $w^T x + b$ can be larger than 1 or negative.

This is why we need sigmoid function.



Apply sigmoid function σ to $\hat{y}$

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

Thus, $0 \le \hat{y} \le 1$

In logistic regression, we want to train w and b so that we have a good estimate of $\hat{y}$.

Another notation which combines w and b in θ.

$x_0 = 1$, $x \in \mathbb{R}^{n_x+1}$

$\hat{y} = \sigma(\theta^T x)$

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix}$ (θ_0 = b; $\theta_1 \dots \theta_{n_x}$ = w)

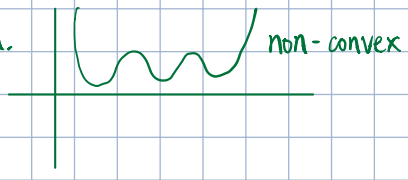
Cost function in Logistic Regression

$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$, we use $x^{(i)}$ to denote the i-th sample.

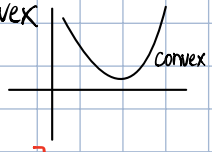
Given training examples $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$, we want $\hat{y}^{(i)} \approx y^{(i)}$

Loss (error) function: $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$, In logistic regression, people usually dont do this, because the optimization problem becomes non-convex.
(Only apply to single sample)

In gradient descent, it cant find the global optima.



Thus, we need a different loss function in logistic regression which produces convex in optimization problem.



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

Intuition: We want $\mathcal{L}(\hat{y}, y)$ as small as possible.

If $y=1$, $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$ so $\hat{y}$ as large as possible

If $y=0$, $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$ so $\hat{y}$ as small as possible.

Now, let define a Cost function which measures the loss function on the entire training set.

Cost function: $J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = \frac{-1}{m} \sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})$
(the cost for all samples)

In training, we want to find w and b which minimize the cost function $J(w, b)$ in our Logistic Regression model.

Justification of $\mathcal{L}(\hat{y}, y)$: $\hat{y} = \sigma(z)$, where $\sigma(z) = 1/(1+e^{-z})$

If $y=1$, $P(y|x) = \hat{y}$
If $y=0$, $P(y|x) = 1-\hat{y}$

$P(y|x)$ can be written as

$\hat{y}^{y}(1-\hat{y})^{(1-y)}$

Now, maximizing $P(y|x)$ is the same as maximizing $\log P(y|x)$.

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

$= -\mathcal{L}(\hat{y}, y)$

Now, cost on m samples.

$P(\text{labels in training set}) = \prod_{i=1}^{m} P(y^{(i)}|x^{(i)})$

$\rightarrow \log P(\text{labels in training set}) = \log \prod_{i=1}^{m} P(y^{(i)}|x^{(i)})$

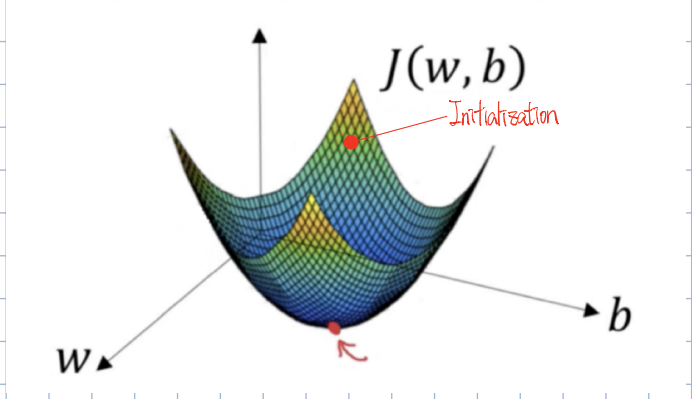
$\rightarrow = \sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)})$

$= \sum_{i=1}^{m} -\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

Maximum Likelihood Estimation: Find parameters to maximize: $\sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)})$

It turns out that logistic regression can be viewed as a very small neural network.

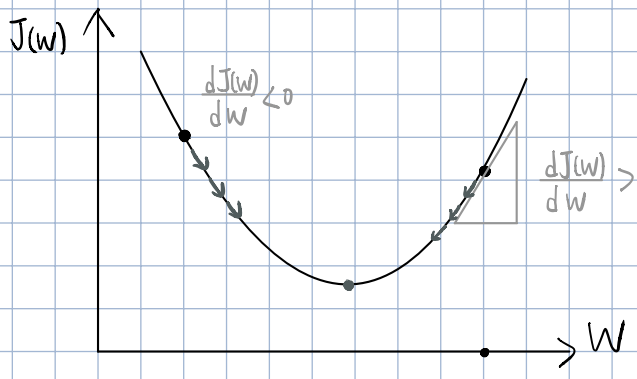
How to use Gradient Descent to train our logistic regression model.



For the case of $J(w, b)$

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

$$b := b - \alpha \frac{dJ(w,b)}{db}$$



Repeat { α: learning rate

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

}

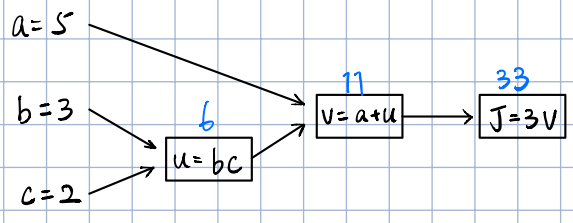
Case 1: $\frac{dJ(w)}{dw} > 0$, w decrease toward the bottom.

Case 2: $\frac{dJ(w)}{dw} < 0$, w increase toward the bottom.

Computation Graph: Forward

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

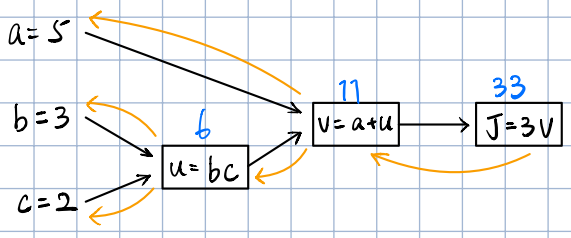
$u = bc$
$v = a + u$
$J = 3v$



Why computation graph? Given a cost function J, we want to know how J varies when its parameters W, b vary. We can use forward propagation, but it's slow. Instead, backpropagation is faster.

The time difference is N x P versus N + P where there are N nodes and P parameters.
forward backward

Derivatives with Computation Graph: Backward



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

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

$$\frac{dJ}{du} = ?$$

We need to figure out $\frac{dJ}{da}, \frac{dJ}{db}, \frac{dJ}{dc}$. Hint: Use chain rule.

Logistic Regression Gradient Descent: On one sample.

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

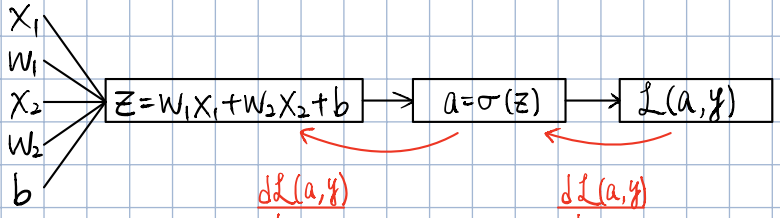
estimated output — sigmoid fun.

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

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

E.g.: With two features x_1 and x_2

Thus, the computation graph is:



$$\frac{d\mathcal{L}(a,y)}{dz} = \frac{d\mathcal{L}}{da}\cdot\frac{da}{dz} \quad \left(\frac{da}{dz} = a(1-a)\right)$$

$$= a - y$$

$$\frac{d\mathcal{L}(a,y)}{da} = \frac{-y}{a} + \frac{1-y}{1-a}$$

Now, we can calculate $\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \frac{\partial \mathcal{L}}{\partial b}$.

$$\frac{\partial \mathcal{L}}{\partial w_1} = "dw_1" = \frac{\partial \mathcal{L}}{\partial z}\cdot\frac{\partial z}{\partial w_1} = x_1 \cdot \frac{d\mathcal{L}}{dz}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = "dw_2" = x_2 \frac{d\mathcal{L}}{dz}$$

$$\frac{\partial \mathcal{L}}{\partial b} = "db" = \frac{d\mathcal{L}}{dz} \rightarrow a - y$$

Once dw_1, dw_2, db are known, we can update w_1, w_2 and b.

$$w_1 := w_1 - \alpha\, dw_1$$
$$w_2 := w_2 - \alpha\, dw_2$$
$$b := b - \alpha\, db$$

Logistic Regression Gradient Descent: On m samples.

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

$$\hat{y}^{(i)} = a^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

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

Notice $dw_1 = \frac{\partial J}{\partial w_1}$, $dw_2 = \frac{\partial J}{\partial w_2}$, $db = \frac{\partial J}{\partial b}$

for i = 1 to m: {

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

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

$J \mathrel{+}= -\left[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})\right]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

$n=2$ features. This could be a for loop if there are more features.

$db \mathrel{+}= dz^{(i)}$

}

$J \mathrel{/}= m$

$dw_1 \mathrel{/}= m$

$dw_2 \mathrel{/}= m$

$db \mathrel{/}= m$

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

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

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

There are two for loops here. It's too slow. Don't even think about it. Use Vectorization.

Avoid explicit for-loop whenever possible.

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

$$a = \sigma(z)$$

$$dz = a - y$$

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

$$db = \frac{1}{m} \text{sum}(dz)$$

$$w \mathrel{-}= \alpha\, dw$$

$$b \mathrel{-}= \alpha\, db$$