deeplearning.ai

Basics of Neural Network Programming

Logistic Regression

This is a learning algorithm that you use when the output label y in a supervised learning problem are all either zero or one, so for binary classification problem.

Logistic Regression

Given x, want $\hat{y} = P(y=1 \mid x)$

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

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

Parameters: $\boxed{w} \in \mathbb{R}^{n_x}$, $\boxed{b} \in \mathbb{R}$.

Output $\hat{y} = \sigma(\underbrace{w^T x + b}_{z})$

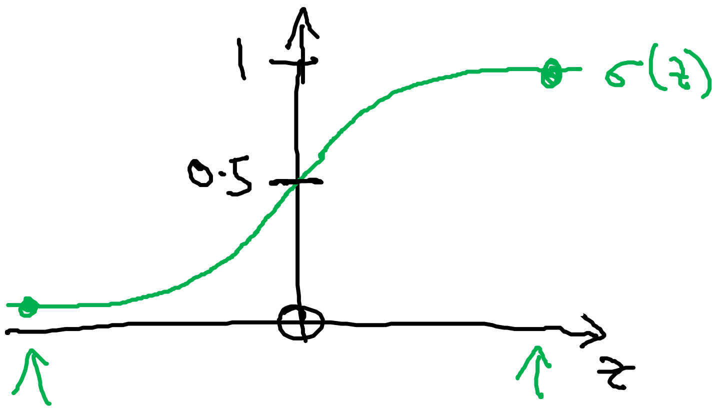
$x_0 = 1$, $x \in \mathbb{R}^{n_x+1}$

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

$$\underbrace{\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{array}{l} \} b \leftarrow \\ \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} w \leftarrow \end{array}}$$

When implement neural network, it will be easier to just keep B and W as separate parameters

So in this class, we will not use any of this notational convention



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

If z large $\sigma(z) \approx \frac{1}{1+0} = 1$

if z is very small or If z large negative number

$$\sigma(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{1+\text{Bignum}} \approx 0$$

So when you implement logistic regression, your job is to try to learn parameters W and B so that y hat becomes a good estimate of the chance of Y equal to one.

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Logistic Regression cost function

Logistic Regression cost function

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

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

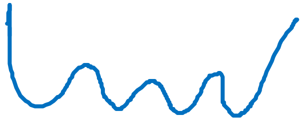
Given $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$.

$x^{(i)}$
$y^{(i)}$
$z^{(i)}$

i-th example.

$\boxed{\text{Loss}}$ (error) function: $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

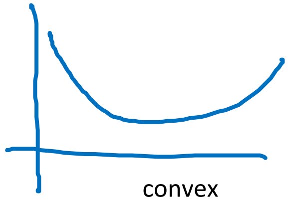
the function I will need to define the measure how good our output y-hat is when true labels is y. And square error seems like it might be a reasonable except it makes gradient descent not work well.



non-convex

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

optimization problem becomes non-convex
so you end up with optimization with multiple local optima
so gradient descent may not find the global optimum



convex

If $y = 1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y} \leftarrow$ Want $\log \hat{y}$ large, want $\hat{y}$ large.

If $y = 0$: $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y}) \leftarrow$ Want $\log 1-\hat{y}$ large .... want $\hat{y}$ small

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

Lost function was defined with respect to a single training example. It measures how well you're doing on a single training example.
Cost function, which measure how well you're doing on the entire training set, is the cost of your parameters.

Andrew Ng

So in training your logistic regression model, we're going to find parameters W and B that minimize the overall cost function J written at the bottom

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