

Coursera-Stanford-ML-Notes

Quentin Truong

20 June 2017 - ? July 2017

Contents

1	Week 1: Introduction	2
1.1	Overview	2
2	Week 2: Linear Regression with Multiple Variables	3
2.1	Overview	3
2.2	Notation	3
2.3	Gradient Descent	3
2.4	Normal Equation	3
3	Week 3: Logistic Regression	4
3.1	Overview	4
3.2	Logistic Regression Hypothesis Function	4
3.3	Logistic Regression Cost Function	4
3.4	Proof of Logistic Regression Cost Function Derivative	5
3.5	Regularization	6
4	Week 4: Artificial Neural Networks Representation	7
4.1	Overview	7
4.2	Notation	7
4.3	Equations	7
4.4	Sample Three Layer System	7
5	Week 5: Artificial Neural Network Learning	8
5.1	Notation	8
5.2	Cost Function	8
5.3	Backpropagation Algorithm	8
5.4	Backpropagation Derivation - Base Case	9
5.5	Backpropagation Derivation - Recursive Case	9
5.6	Backpropagation Intuition	10
5.7	Gradient Checking	10
5.8	Random Initialization	10
6	Sources	11

1 Week 1: Introduction

1.1 Overview

- Machine Learning: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .”
- Supervised Learning: know what our correct output looks like
 - Regression: want continuous output
 - Classification: want discrete output
- Unsupervised Learning: little or no idea what our results should look like
 - Clustering: find groups according to similarity in various variables
 - Nonclustering: find structure in chaos

2 Week 2: Linear Regression with Multiple Variables

2.1 Overview

- Use linear regression for continuous output
- Choose gradient descent if many features (million+) because the inverse matrix required for the normal equation can become expensive to compute
- Normal equation will directly compute theta
- Normalize features if using gradient descent

2.2 Notation

$m = \text{number of samples}$

$n = \text{number of feature}$

$x = (n \times 1)$

$X = (m \times n)$

$X_j = (m \times 1)$

$\theta = (n \times 1)$

$\theta_j = (1 \times 1)$

2.3 Gradient Descent

Hypothesis Function	$h_{\theta}(x) = \theta^T \times x$
Vectorized Hypothesis Function	$h_{\theta}(X) = X \cdot \theta$
Linear Regression Cost Function	$J(\theta) = \frac{1}{2m} \sum (h_{\theta}(X) - y)^2$
Derivative of Linear Regression CF wrt θ_j	$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum (h_{\theta}(X) - y) \cdot * X_j$
Change in θ_j	$\begin{aligned} \theta_j &= \theta_j - \alpha \frac{\partial}{\partial \theta_j} \\ &= \theta_j - \alpha \frac{1}{m} \sum (h_{\theta}(X) - y) \cdot * X_j \end{aligned}$
Vectorized Change in θ	$\theta = \theta - \alpha \frac{1}{m} X^T (X \cdot \theta - y)$

2.4 Normal Equation

$$\theta = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$

3 Week 3: Logistic Regression

3.1 Overview

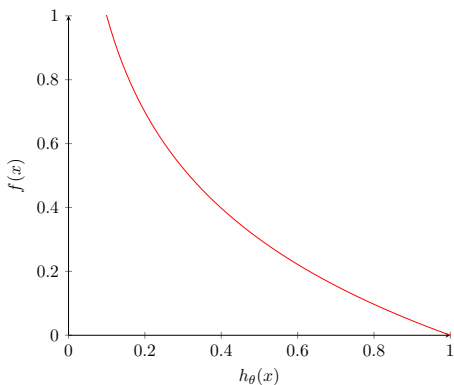
- Use logistic regression for discrete output (classification)
 - $h_{\theta}(x) = (y = 1|x; \theta)$; gives probability that the output is 1 given x
 - Sigmoid/Logistic function maps any real number to $(0, 1)$
 - Logarithm turns sum into product, allowing easier differentiation without altering search space
- For multi-class classification, use one-vs-all
 - Pick class i that maximizes $h_{\theta}^i(x)$
- Overfitting is when learned hypothesis fits training data well but fails to generalize; underfitting is when doesn't fit training data
- Address overfitting by reducing number of features, model selection, and regularization
 - Regularization results in simpler hypothesis and less overfitting
 - Extremely large λ will result in underfitting and gradient descent will fail to converge
 - Do not regularize λ_0
- Use other prewritten optimization algorithms (conjugate gradient, BFGS, L-BFGS) because they are faster

3.2 Logistic Regression Hypothesis Function

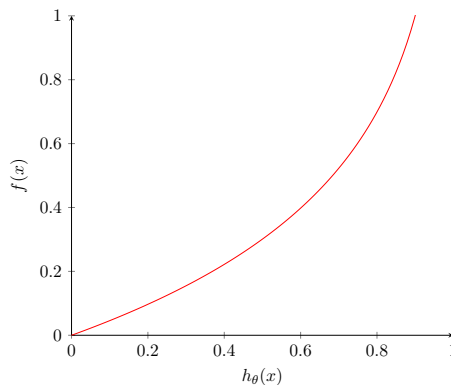
Sigmoid/Logistic Function	$g(z) = \frac{1}{1 + e^{-z}}$
Hypothesis Function	$h_{\theta}(x) = g(\theta^T x)$ $= \frac{1}{1 + e^{-\theta^T x}}$

3.3 Logistic Regression Cost Function

$$\begin{aligned} \text{Cost}(h_{\theta}(x), y) &= \begin{cases} -\log(h_{\theta}(x)) & \text{if } y = 1 \\ -\log(1 - h_{\theta}(x)) & \text{if } y = 0 \end{cases} \\ &= -y \log(h_{\theta}(x)) - (1 - y) \log(1 - h_{\theta}(x)) \\ J(\theta) &= \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^i), y^i) \\ J(\theta) &= \frac{-1}{m} \sum_{i=1}^m [y^i \log(h_{\theta}(x^i)) + (1 - y^i) \log(1 - h_{\theta}(x^i))] \end{aligned}$$



(a) if $y=1$



(b) if $y=0$

3.4 Proof of Logistic Regression Cost Function Derivative

$$\begin{aligned}
J(\theta) &= \frac{-1}{m} \sum_{i=1}^m [y^i \log(h_\theta(x^i)) + (1 - y^i) \log(1 - h_\theta(x^i))] \\
\log(h_\theta(x^i)) &= \log\left(\frac{1}{1 + e^{-\theta x^i}}\right) = -\log(1 + e^{-\theta x^i}) \\
\log(1 - h_\theta(x^i)) &= \log\left(1 - \frac{1}{1 + e^{-\theta x^i}}\right) = \log(e^{-\theta x^i}) - \log(1 + e^{-\theta x^i}) = -\theta x^i - \log(1 + e^{-\theta x^i}) \\
J(\theta) &= -\frac{1}{m} \sum_{i=1}^m \left[-y^i (\log(1 + e^{-\theta x^i})) + (1 - y^i) (-\theta x^i - \log(1 + e^{-\theta x^i})) \right] \\
&= -\frac{1}{m} \sum_{i=1}^m \left[y^i \theta x^i - \theta x^i - \log(1 + e^{-\theta x^i}) \right] \\
&= -\frac{1}{m} \sum_{i=1}^m \left[y^i \theta x^i - \log(e^{\theta x^i}) - \log(1 + e^{-\theta x^i}) \right] \\
&= -\frac{1}{m} \sum_{i=1}^m \left[y^i \theta x^i - \log(1 + e^{\theta x^i}) \right] \\
\frac{\partial}{\partial \theta_j} y^i \theta x^i &= y^i x_j^i \\
\frac{\partial}{\partial \theta_j} \log(1 + e^{\theta x^i}) &= \frac{x_j^i e^{\theta x^i}}{1 + e^{\theta x^i}} \\
&= \frac{x_j^i}{1 + e^{-\theta x^i}} \\
&= x_j^i h_\theta(x^i) \\
\frac{\partial}{\partial \theta_j} J(\theta) &= -\frac{1}{m} \sum_{i=1}^m [y^i x_j^i - x_j^i h_\theta(x^i)] \\
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{1}{m} \sum_{i=1}^m [h_\theta(x^i) - y^i] x_j^i
\end{aligned}$$

Credit¹

¹<https://math.stackexchange.com/questions/477207/derivative-of-cost-function-for-logistic-regression>

3.5 Regularization

Regularizing Term	$\lambda \sum_{j=1}^n \theta_j^2$
Regularized Linear Regression CF	$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^i) - y^i)^2 + \lambda \sum_{j=1}^n \theta_j^2$
Regularized Logistic Regression CF	$J(\theta) = \frac{-1}{m} \sum_{i=1}^m [y^i \log(h_{\theta}(x^i)) + (1 - y^i) \log(1 - h_{\theta}(x^i))] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$
Regularized GD (Lin/Log Regression)	$\begin{cases} \theta_0 = \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^i) - y^i) x_0^i \right] \\ \theta_j = \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^i) - y^i) x_j^i + \frac{\lambda}{m} \theta_j \right] \quad (j=1,2,\dots,n) \end{cases}$
Regularized Normal Equation	$\theta = (X^T X + \lambda \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}_{n+1,n+1})^{-1} X^T y$

4 Week 4: Artificial Neural Networks Representation

4.1 Overview

- Neural networks allow for non-linear classification in situations with many features
 - Necessary b/c 100 features at 3rd level polynomials generates 170k features, which quickly becomes intractable
 - “One learning algorithm” hypothesis; you can see with your tongue : brain learns using one algorithm, not thousands of different programs
 - Can have multiple hidden layers
 - Can have multiple outputs (one-vs-all for multi-class classification)
 - If network has s_j units in layer j and s_{j+1} units in layer $j+1$, then Θ^j will be of dimension $s_{j+1} \times s_j + 1$
 - The +1 comes from the addition in $\Theta^{(j)}$ of the bias node, x_0 and $\Theta_0^{(j)}$
- Forward Propagation is used to predict based on learned parameters
- Bias node gives each node a trainable constant value
 - Allows bias weight to shift the activation curve left/right
 - Other weights affect steepness

4.2 Notation

- $g(x)$: sigmoid function
- $\Theta^{(j)}$: matrix of weights controlling function mapping from layer j to layer $j+1$; each layer gets own Θ^j
- $\Theta_{j,0}, \dots, \Theta_{j,n}$ weights corresponding to the inputs a_0, \dots, a_n going into z_j
- $w_{0,j}, \dots, w_{n,j}$ weights corresponding to the inputs a_0, \dots, a_n going into z_j
- $z_k^{(j)}$: encompasses parameters inside of g function
- $a_i^{(j)}$: “activation” of unit i in layer j

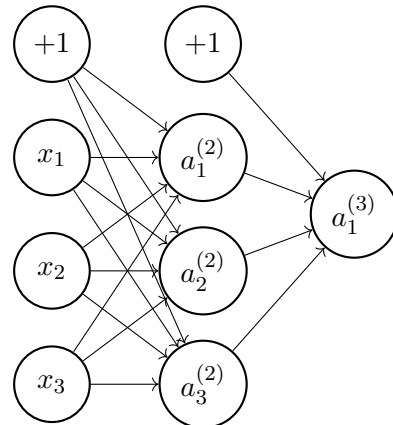
4.3 Equations

$$\begin{aligned}
 z_k^{(j)} &= \Theta_{k,0}^{(j-1)}x_0 + \Theta_{k,1}^{(j-1)}x_1 + \dots + \Theta_{k,n}^{(j-1)}x_n \\
 &= \Theta_{k,0}^{(j-1)}a_0^{(j-1)} + \Theta_{k,1}^{(j-1)}a_1^{(j-1)} + \dots + \Theta_{k,n}^{(j-1)}a_n^{(j-1)} \\
 z^{(j)} &= \Theta^{(j-1)}a^{(j-1)} \\
 a^{(j)} &= g(z^{(j)})
 \end{aligned}$$

4.4 Sample Three Layer System

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} \rightarrow h_{\Theta}(x)$$

$$\begin{aligned}
 a_1^{(2)} &= g(\Theta_{1,0}^{(1)}x_0 + \Theta_{1,1}^{(1)}x_1 + \Theta_{1,2}^{(1)}x_2 + \Theta_{1,3}^{(1)}x_3) \\
 a_2^{(2)} &= g(\Theta_{2,0}^{(1)}x_0 + \Theta_{2,1}^{(1)}x_1 + \Theta_{2,2}^{(1)}x_2 + \Theta_{2,3}^{(1)}x_3) \\
 a_3^{(2)} &= g(\Theta_{3,0}^{(1)}x_0 + \Theta_{3,1}^{(1)}x_1 + \Theta_{3,2}^{(1)}x_2 + \Theta_{3,3}^{(1)}x_3) \\
 h_{\Theta}(x) &= g(\Theta_{1,0}^{(2)}a_0^{(2)} + \Theta_{1,1}^{(2)}a_1^{(2)} + \Theta_{1,2}^{(2)}a_2^{(2)} + \Theta_{1,3}^{(2)}a_3^{(2)}) \\
 &= g(z^3) \\
 &= a_1^{(3)}
 \end{aligned}$$



5 Week 5: Artificial Neural Network Learning

5.1 Notation

- L : total number of layers in the network
- s_l : number of units (not counting bias unit) in layer l
- K : number of output units/classes
- $h_{\Theta}(x)_k$: hypothesis that results in the k th output
- δ_k : Error signal

5.2 Cost Function

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

Picking training example

Picking output node

Picking layer

Picking node

Picking Θ

$$\sum_{i=1}^m \sum_{k=1}^K \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}}$$

5.3 Backpropagation Algorithm

1. Set $a(1) := x(t)$
2. Perform forward propagation to compute $a(l)$ for $l = 2, 3, \dots, L$
3. Using $y^{(t)}$, compute $\delta^L = a^{(L)} - y^{(t)}$
4. Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$ using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \cdot * a^{(l)} \cdot * (1 - a^{(l)})$
5. $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ or with vectorization $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$
6. $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$ where $\begin{cases} D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)} & (j = 0) \\ D_{i,j}^{(l)} := \frac{1}{m} \left(\Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)} \right) & (j \neq 0) \end{cases}$

5.4 Backpropagation Derivation - Base Case

$$\begin{aligned}
\frac{\partial C}{\partial w_{ij}} &= \frac{\partial C}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} && \text{Cost } C \text{ varies wrt input accumulator } z_j, z_j \text{ varies wrt } w_{ij} \text{ (Chain Rule)} \\
\frac{\partial C}{\partial z_j} &= \frac{\partial}{\partial z_j} (y_j - a_j)^2 && \text{Cost defined as } (y_j - a_j)^2 \\
&= \frac{\partial}{\partial z_j} (y_j - g(z_j))^2 \\
&= -2(y_j - g(z_j))g'(z_j) \\
g'(z_j) &= \frac{d}{dz_j} \frac{1}{1 + e^{-z_j}} && \text{Derivative of logistic function} \\
&= \frac{1}{1 + e^{-z_j}} \frac{e^{-z_j}}{1 + e^{-z_j}} \\
&= g(z_j)(1 - g(z_j)) \\
\frac{\partial C}{\partial z_j} &= -2(y_j - g(z_j))g(z_j)(1 - g(z_j)) \\
&= -2(y_j - a_j)a_j(1 - a_j) \\
\frac{\partial z_j}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \sum_q a_q w_{q,j} && \text{Definition of } z_j \text{ as sum of previous node's inputs and their weights} \\
&= a_i \\
\frac{\partial C}{\partial w_{ij}} &= -2(y_j - a_j)a_j(1 - a_j)(a_i) \\
&= -\Delta_j a_i
\end{aligned}$$

5.5 Backpropagation Derivation - Recursive Case

$$\begin{aligned}
\frac{\partial C}{\partial w_{i,j}} &= \sum_k \left(\frac{\partial C}{\partial z_k} \frac{\partial z_k}{\partial a_j} \frac{\partial a_j}{\partial z_j} \frac{\partial z_j}{\partial w_{i,j}} \right) && C \text{ depends on } z_k, z_k \text{ depends on } a_j, a_j \text{ depends on } z_j, z_j \text{ depends on } w_{i,j} \\
\frac{\partial C}{\partial z_k} &= -\Delta_k \\
\frac{\partial z_k}{\partial a_j} &= \frac{\partial}{\partial a_j} \sum_s a_s w_{s,k} \\
&= w_{j,k} \\
\frac{\partial a_j}{\partial z_j} &= \frac{\partial}{\partial z_j} g(z_j) \\
&= g(z_j)(1 - g(z_j)) \\
&= a_j(1 - a_j) \\
\frac{\partial z_j}{\partial w_{i,j}} &= \frac{\partial}{\partial w_{i,j}} \sum_q a_q w_{q,j} \\
&= a_i \\
\frac{\partial C}{\partial w_{i,j}} &= \sum_k (-\Delta_k w_{j,k} a_j(1 - a_j) a_i) \\
&= \sum_k (-\Delta_k w_{j,k}) a_j(1 - a_j) a_i \\
&= -\Delta_j a_i
\end{aligned}$$

5.6 Backpropagation Intuition

δ_k is the error signal from the output

$$\delta_k = (a_k - t_k)g'_k(z_k)$$

so, error wrt output weights is

$$\frac{\partial E}{\partial w_{j,k}} = \delta_k a_j$$

similarly, error wrt internal weights is

$$\frac{\partial E}{\partial w_{i,j}} = \delta_j a_i$$

the error is determined by following layer's error, so it may also be understood as

$$\frac{\partial E}{\partial w_{i,j}} = g'_j(z_j) \sum_k (\delta_k w_{j,k}) a_i$$

5.7 Gradient Checking

- Gradient checking ensures that backpropagation is actually working
 - Turn off gradient checking once backpropagation is verified to work
- Approximate the derivative of cost function using slope
 - Pick $\epsilon = 10^{-4}$
 - Check all Θ_j

$$\frac{\partial}{\partial \Theta_j} J(\Theta) \approx \frac{J(\Theta_1, \dots, \Theta_j + \epsilon, \dots, \Theta_n) - J(\Theta_1, \dots, \Theta_j - \epsilon, \dots, \Theta_n)}{2\epsilon}$$

5.8 Random Initialization

- Need Symmetry Breaking
 - All hidden units would receive the same signal and the same updates
 - Results in finding only one feature, redundantly copied many times over
- Fixed by randomly initializing each $\Theta_{ij}^{(l)}$ to a value in $[-\epsilon, \epsilon]$
 - A good choice of ϵ init is $\epsilon = \frac{\sqrt{6}}{\sqrt{L_{in} + L_{out}}}$, where $L_{in} = s_l$ and $L_{out} = s_{l+1}$, the number of units in the layers adjacent to $\Theta^{(l)}$

²<https://pdfs.semanticscholar.org/69c2/814c7f8ca16aa836fd32e3c4975f8208e63f.pdf>

³<https://stats.stackexchange.com/questions/94387/how-to-derive-errors-in-neural-network-with-the-backpropagation-algorithm>

⁴<https://theclevermachine.wordpress.com/2014/09/06/derivation-error-backpropagation-gradient-descent-for-neural-networks/>

6 Sources