CS129 - Review Session

Younes Bensouda Mourri
Algorithms Covered

- Linear regression
- Logistic Regression
- Softmax
- Neural Networks
- Support Vector Machines
- Principal Component Analysis
- Bias vs. Variance Trade off
- K-Means Clustering
- Make sure you are familiar with the problem sets
Biggest Tips

- Know the formulas well enough and be able to derive them
- Know how to write the vectorized implementations
- The gradient of a weight matrix $W$ is the same dimension as $W$.
- Understand the Bias vs. Variance tradeoff.
Cheatsheet

Assume that $\theta \in \mathbb{R}^n$, $a \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times n}$, and $A$ is symmetric. Then,

$$\frac{\partial a^T \theta}{\partial \theta} = a \quad \quad (V1)$$
$$\frac{\partial a^T A \theta}{\partial \theta} = 2A \theta \quad \quad (V2)$$

Assume that $x \in \mathbb{R}^n$, $W \in \mathbb{R}^{d \times n}$, $f: \mathbb{R} \rightarrow \mathbb{R}$ element-wise function, and $\delta \in \mathbb{R}^d$. Then,

$$\frac{\partial x}{\partial x} = I_{n \times n} \text{ with } I_{n \times n} \text{ the identity matrix} \quad \quad (B1)$$
$$\frac{\partial W x}{\partial x} = W \quad \quad (B2)$$
$$\frac{\partial W f}{\partial W} = \delta x^T \quad \quad (B3)$$
$$\frac{\partial f(x_1), \ldots, f(x_n)}{\partial x} = \text{Diag}[f'(x_1), \ldots, f'(x_n)] \quad \quad (B4)$$

Assume that $y \in \mathbb{R}^K$, is one-hot vector, and $z \in \mathbb{R}^N$. If

$$\begin{cases} 
\hat{y} = \text{Softmax}(z) \\
J = \text{CE}(y, \hat{y}) \end{cases}$$

with

$$\text{Softmax}(z) = \left( \frac{1}{Z} e^{z_{k}}, \frac{1}{Z} e^{z_{k}}, \ldots, \frac{1}{Z} e^{z_{k}} \right) \quad (Z \text{ the normalization constant})$$

and

$$\text{CE}(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)$$

Then,

$$\frac{\partial J}{\partial z} = (\hat{y} - y)^T \quad \quad (B5)$$

Cost function of linear regression (with regularization):

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Cost function of logistic regression (with regularization):

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left( y^{(i)} \log \left( h_{\theta}(x^{(i)}) \right) + (1 - y^{(i)}) \log \left( 1 - h_{\theta}(x^{(i)}) \right) \right) + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Sigmoid function:

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

- Understand how the cost function works
- Understand how the gradient descent works
- Understand how the normal equations work
Logistic Regression

\[ J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] \]

Repeat {
\[
\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} \\
\theta_j := \theta_j - \alpha \left[ \left( \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \right] \quad j \in \{1,2,...n\}
\]}

Vectorized Implementation of the cost function:

\[ h = g(X\theta) \]
\[ J(\theta) = \frac{1}{m} \cdot (-y^T \log(h) - (1 - y)^T \log(1 - h)) \]
\[ \theta := \theta - \frac{\alpha}{m} X^T(g(X\theta) - \bar{y}) \]
Neural Networks

- Know how to use your identities
- Understand how Backpropagation works
- Be familiar with the softmax/sigmoid functions
Diagnosing Bias vs. Variance

In this section we examine the relationship between the degree of the polynomial $d$ and the underfitting or overfitting of our hypothesis.

- We need to distinguish whether bias or variance is the problem contributing to bad predictions.
- High bias is underfitting and high variance is overfitting. We need to find a golden mean between these two.

The training error will tend to decrease as we increase the degree $d$ of the polynomial.

At the same time, the cross validation error will tend to decrease as we increase $d$ up to a point, and then it will increase as $d$ is increased, forming a convex curve.
Bias Vs. Variance

Choosing M the order of polynomials. How can we tell which parameters $\Theta$ to leave in the model (known as "model selection")?

There are several ways to solve this problem:

- Get more data (very difficult).
- Choose the model which best fits the data without overfitting (very difficult).
- Reduce the opportunity for overfitting through regularization.

Intuition for the bias-variance trade-off:

- Complex model $\Rightarrow$ sensitive to data $\Rightarrow$ much affected by changes in X $\Rightarrow$ high variance, low bias.
- Simple model $\Rightarrow$ more rigid $\Rightarrow$ does not change as much with changes in X $\Rightarrow$ low variance, high bias.

One of the most important goals in learning: finding a model that is just right in the bias-variance trade-off.

Regularization Effects:

- Small values of $\lambda$ allow model to become finely tuned to noise leading to large variance $\Rightarrow$ overfitting.
- Large values of $\lambda$ pull weight parameters to zero leading to large bias $\Rightarrow$ underfitting.
Bias Vs. Variance

Model Complexity Effects:

- Lower-order polynomials (low model complexity) have high bias and low variance. In this case, the model fits poorly consistently.
- Higher-order polynomials (high model complexity) fit the training data extremely well and the test data extremely poorly. These have low bias on the training data, but very high variance.
- In reality, we would want to choose a model somewhere in between, that can generalize well but also fits the data reasonably well.

A typical rule of thumb when running diagnostics is:

- More training examples fixes high variance but not high bias.
- Fewer features fixes high variance but not high bias.
- Additional features fixes high bias but not high variance.
- The addition of polynomial and interaction features fixes high bias but not high variance.
- When using gradient descent, decreasing lambda can fix high bias and increasing lambda can fix high variance (lambda is the regularization parameter).
- When using neural networks, small neural networks are more prone to under-fitting and big neural networks are prone to over-fitting. Cross-validation of network size is a way to choose alternatives.
Support Vector Machines

\[ J(\theta) = C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \Theta_j^2 \]

\[ \min_{\Theta} C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\Theta^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \Theta_j^2 \]

Choosing C (recall that C = 1/\lambda)

- If C is large, then we get higher variance/lower bias
- If C is small, then we get lower variance/higher bias
K-Means

The K-Means Algorithm is the most popular and widely used algorithm for automatically grouping data into coherent subsets.

1. Randomly initialize two points in the dataset called the cluster centroids.
2. Cluster assignment: assign all examples into one of two groups based on which cluster centroid the example is closest to.
3. Move centroid: compute the averages for all the points inside each of the two cluster centroid groups, then move the cluster centroid points to those averages.
4. Re-run (2) and (3) until we have found our clusters.

Clustering is good for:

- Market segmentation
- Social network analysis
- Organizing computer clusters
- Astronomical data analysis

\[ J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m} \sum_{i=1}^{m} \| x^{(i)} - \mu_{c^{(i)}} \|^2 \]
PCA

Given two features, \(x_1\) and \(x_2\), we want to find a single line that effectively describes both features at once. We then map our old features onto this new line to get a new single feature.

The same can be done with three features, where we map them to a plane.

PCA is not linear regression

- In linear regression, we are minimizing the squared error from every point to our predictor line. These are vertical distances.
- In PCA, we are minimizing the shortest distance, or shortest orthogonal distances, to our data points.

- Feature Normalize
- Compute Covariance matrix \(\Sigma\)
- Compute "eigenvectors" of covariance matrix \(\Sigma\)
- Take the first \(k\) columns of the \(U\) matrix and compute \(z\)
The recommended approach to solving machine learning problems is:

- Start with a simple algorithm, implement it quickly, and test it early.
- Plot learning curves to decide if more data, more features, etc. will help
- Error analysis: manually examine the errors on examples in the cross validation set and try to spot a trend.

For example: In predicting a cancer diagnosis where 0.5% of the examples have cancer, we find our learning algorithm has a 1% error. However, if we were to simply classify every single example as a 0, then our error would reduce to 0.5% even though we did not improve the algorithm.

For this we can use Precision/Recall.

- Predicted: 1, Actual: 1 --- True positive
- Predicted: 0, Actual: 0 --- True negative
- Predicted: 0, Actual: 1 --- False negative
- Predicted: 1, Actual: 0 --- False positive

**Precision**: of all patients we predicted where $y=1$, what fraction actually has cancer?

\[
\frac{\text{True Positives}}{\text{Total number of predicted positives}} = \frac{\text{True Positives}}{\text{True Positives} + \text{False positives}}
\]

**Recall**: Of all the patients that actually have cancer, what fraction did we correctly detect as having cancer?

\[
\frac{\text{True Positives}}{\text{Total number of actual positives}} = \frac{\text{True Positives}}{\text{True Positives} + \text{False negatives}}
\]