Machine Learning by Stanford University

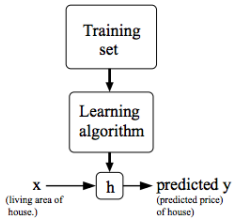
Vocabulary

Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed [1].

Supervised Learning: In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output. Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output.

Unsupervised Learning: Unsupervised learning allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables. We can derive this structure by clustering the data based on relationships among the variables in the data.

Linear regression



- n: number of features
- m: number of training set
- α: Learning Rate

c

Hypothesis:$h_\theta(x) = \theta^T x$ with $x_0 = 1$

Parameters:$\theta_0, \theta_1, ..., \theta_n$

Cost function:$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$

Gradient descent:Repeat

$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$

simultaneously update for every j={0,...,n}

Feature Scaling: for fast convergence, Get every feature into approximately a $-1 \leq x_i \leq 1$ range

Normal Equation

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

There is no need to do feature scaling with the normal equation.

Comparaison

Gradient Descent	Normal Equation
Need to choose α	No need to choose α
Needs many iterations	No need to iterate
$O(kn^2)$	$O(n^3)$ for $X^T X$
Works well when n is large	Slow if n is very large

Classification

Sigmoid function

$h_\theta(x) = p(y = 1/x; \theta) = \frac{1}{1+\exp -\theta^T x}$

Decision Boundary

$h_\theta(x) \geq 0.5 \Rightarrow y = 1 \quad h_\theta(x) < 0.5 \Rightarrow y = 0$

Logistic regression cost fonction

$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\text{Cost}(h_\theta(x^{(i)}), y^{(i)})$

$$\begin{cases} \text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) & \text{if y = 1} \\ \text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) & \text{if y = 0} \end{cases}$$

$$\begin{cases} \text{Cost}(h_\theta(x), y) = 0 & \text{if } h_\theta(x) = y \\ \text{Cost}(h_\theta(x), y) \to \infty & \text{if } y = 0 \text{ and } h_\theta(x) \to 1 \\ \text{Cost}(h_\theta(x), y) \to \infty & \text{if } y = 1 \text{ and } h_\theta(x) \to 0 \end{cases}$$

$$J(\theta) = \frac{1}{m}\left[\sum_{i=1}^{m} y_i \log h_\theta(x_i) + (1 - y_i)\log(1 - h_\theta(x_i))\right]$$

Gradient Descent Repeat

$\theta_j := \theta_j - \alpha\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$

simultaneously update for every j={0,...,n}

Advanced Optimization Matlab

Listing 1: Function

```
function [jVal, gradient] = costFunction(
    theta)
  jVal = [...code to compute J(theta)
      ...];
  gradient = [...code to compute
      derivative of J(theta)...];
end
```

Listing 2: Fminunc

```
options = optimset('GradObj', 'on', '
    MaxIter', 100);
initialTheta = zeros(2,1);
   [optTheta, functionVal, exitFlag] =
       fminunc(@costFunction,
       initialTheta, options);
```

Multiclass Classification

One-vs-all

Train a logistic regression classifier $h_\theta(x)$ for each class i to predict the probability that $y = i$.

$h^i(\theta) = P(y = i/x; \theta)$

To make a prediction on a new x, pick the class i that maximizes $h_\theta(x)$.

Overfitting

Underfit ==> high bias

Overfit ==> high variance

There are two main options to address the issue of overfitting:

- Reduce the number of features.
- Regularization: Keep all the features, but reduce the magnitude of parameters θ_j

Regularization Cost Function

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

Gradient Descent

$$\text{Repeat } \{ \quad \theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha\left[\left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j\right]$$

$$j \in \{1, 2...n\}\}$$

Normal Equation

$$\theta = \left(X^T X + \lambda \cdot L\right)^{-1} X^T y$$

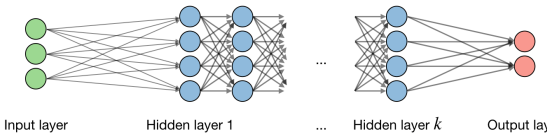
$$\text{where } L = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$$

If $m < n$, then $X^T X$ is non-invertible. However, when we add the term λL, then $X^T X + \lambda L$ becomes invertible.

Neural Networks I

Neural networks are a class of models that are built with layers.Commonly used types of neural networks include convolutional and recurrent neural networks.

Architecture



By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$z_j^{[i]} = {w_j^{[i]}}^T x + b_j^{[i]}$$

where we note w, b, z the weight, bias and output respectively.

$$h_\theta(x) = a^{(j+1)} = g(z^{(j+1)})$$

where $z^{(j+1)} = \Theta^{(j)} a^{(j)}$
If network has s_j units in layer j and s_{j+1} units in layer j+1, then $\Theta^{(j)}$ will be of dimension $s_{j+1} * s_j + 1$.

Cost Function

Let's first define a few variables that we will need to use:

- 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

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[y_k^{(i)} \log((h_\Theta(x^{(i)}))_k) + (1 - y_k^{(i)})$$

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

Back propagation Algorithm

Given training set $\{(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})\}$
Set $\Delta_{i,j}^{(l)} := 0$ for all (l,i,j), (hence you end up having a matrix full of zeros)

Neural Networks II

For training example t =1 to m:

1. Set $a^{(1)} := x^{(t)}$
2. Perform forward propagation to compute $a^{(l)}$ for l=2,3,...,L
3. Using $y^{(t)}$, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$
   Where L is our total number of layers and $a^{(L)}$ is the vector of outputs of the activation units for the last layer. So our "error values" for the last layer are simply the differences of our actual results in the last layer and the correct outputs in y. To get the delta values of the layers before the last layer, we can use an equation that steps us back from right to left:
4. Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$ using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \; .* \; a^{(l)} \; .* \; (1 - a^{(l)})$
   The delta values of layer l are calculated by multiplying the delta values in the next layer with the theta matrix of layer l. We then element-wise multiply that with a function called g', or g-prime, which is the derivative of the activation function g evaluated with the input values given by $z^{(l)}$.
5. $\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ or with vectorization, $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$

Hence we update our new Δ matrix.

$$D_{i,j}^{(l)} := \frac{1}{m}\left(\Delta_{i,j}^{(l)} + \lambda\Theta_{i,j}^{(l)}\right), \text{ If }] \neq 0.$$

$$D_{i,j}^{(l)} := \frac{1}{m}\Delta_{i,j}^{(l)} \text{ If } j = 0$$

D is used as an "accumulator" to add up our values as we go along and eventually compute our partial derivative. Thus we get $\frac{\partial}{\partial \Theta_{ij}^{(i)}} J(\Theta) = D_{ij}^{(l)}$

Implementation Note

In order to use optimizing functions such as "fminunc()" with NN, we will want to "unroll" all the elements and put them into one long vector:

```
thetaVector = [ Theta1(:); Theta2(:);
    Theta3(:); ]
deltaVector = [ D1(:); D2(:); D3(:) ]
```

Gradient Checking

Gradient checking will assure that our backpropagation works as intended. We can approximate the derivative of our cost function $\frac{\partial}{\partial \Theta_j} J(\Theta)$ with:

$$\frac{J(\Theta_1, \ldots, \Theta_j + \epsilon, \ldots, \Theta_n) - J(\Theta_1, \ldots, \Theta_j - \epsilon, \ldots, \Theta_n)}{2\epsilon}$$

A small value for ϵ (epsilon) such as $\epsilon = 10^{-4}$, guarantees that the math works out properly. If the value for ϵ is too small, we can end up with numerical problems.
Once you have verified once that your backpropagation algorithm is correct, you don't need to compute gradApprox again. The code to compute gradApprox can be very slow.

Gradient Checking

Initializing all theta weights to zero does not work with neural networks. When we backpropagate, all nodes will update to the same value repeatedly. Instead we can randomly initialize our weights for our Θ matrices using the following method:
Random initialization: Symmetry breaking

Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$
(i.e. $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)

Neural network learning resume

First, pick a network architecture; choose the layout of your neural network, including how many hidden units in each layer and how many layers in total you want to have.

- Number of input units = dimension of features $x^{(i)}$
- Number of output units = number of classes
- Number of hidden units per layer = usually more the better (must balance with cost of computation as it increases with more hidden units)
- Defaults: 1 hidden layer. If you have more than 1 hidden layer, then it is recommended that you have the same number of units in every hidden layer.

Neural network learning resume

Training a Neural Network

1. Randomly initialize the weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement the cost function
4. Implement backpropagation to compute partial derivatives
5. Use gradient checking to confirm that your backpropagation works. Then disable gradient checking.
6. Use gradient descent or a built-in optimization function to minimize the cost function with the weights in theta.

When we perform forward and back propagation, we loop on every training example:

```
for i = 1:m,
   Perform forward propagation and
       backpropagation using example (x(i
       ),y(i))
   (Get activations a(l) and delta terms
       d(l) for l = 2,...,L
```

Evaluating a Learning Algorithm

Evaluating a Hypothesis:
Once we have done some troubleshooting for errors in our predictions by:

- Getting more training examples
- Trying smaller sets of features
- Trying additional features
- Trying polynomial features
- Increasing or decreasing λ

We can move on to evaluate our new hypothesis. A hypothesis may have a low error for the training examples, but still be inaccurate (overfitting). Thus, to evaluate a hypothesis, given a dataset, we can split up the data into two sets: a training set and a test set. Typically, the training set consists of 70% of the data and the test set is the remaining 30 %.

Evaluating a Learning Algorithm

The new procedure using these two sets is then:

- Learn Θ and minimize $J_{train}(\Theta)$ using the training set.
- Compute the test set error $J_{test}(\Theta)$.

Model Selection

Given many models with different polynomial degrees, we can use a systematic approach to identify the 'best' function. In order to choose the model of your hypothesis, you can test each degree of polynomial and look at the error result. One way to break down our dataset into the three sets is:

- Training set: 60
- Cross validation set: 20
- Test set: 20

We can now calculate three separate error values for the three different sets using the following method:

1. Optimize the parameters in Θ using the training set for each polynomial degree.
2. Find the polynomial degree d with the least error using the cross validation set.
3. Estimate the generalization error using the test set with $J_{test}(\Theta^{(d)})$, (d = theta from polynomial with lower error);

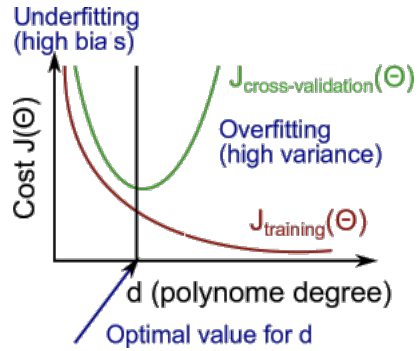
Diagnosing Bias vs. Variance

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. Ideally, we need to find a golden mean between these two.

-**High bias (underfitting)**: both $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ will be high. Also, $J_{CV}(\Theta) \approx J_{train}(\Theta)$.
- **High variance (overfitting)**: $J_{train}(\Theta)$ will be low and $J_{CV}(\Theta)$ will be much greater than $J_{train}(\Theta)$.

Diagnosing Bias vs. Variance



Reference

References

[1] Machine Learning andrew ng. https://www.coursera.org/learn/machine-learning.