

# Weekly Report of machine learning

**Haoyu Wang**

School of Software Engineering  
Shanghai Jiao Tong University  
gogowhy@sjtu.edu.cn

## Abstract

In the past few weeks, I finished the basic knowledge of machine learning, and I am going to give its details here.

## 1 Current State

1. learned the basic classification, hypothesis representation and the decision boundary
2. basically grasped the gradient descent, logistic regression and neural network
3. learned how to debug and evaluate the hypothesis
4. learned the division of the dataset and the learning curves (underfitting and overfitting, precision and recall)
5. learned the "support vector machine", known as SVM and the large margin classifier
6. learned the kernel, especially the Gaussian kernel
7. learned the unsupervised learning, (K-means, principle component analysis, known as PCA)
8. learned the batch, stochastic and mini-batch
9. learned the concept of online learning, including map-reduce and data parallelism, pipeline

## 2 basic classification, hypothesis representation and the decision boundary

The function  $g(z)$ , shown here, maps any real number to the  $(0, 1)$  interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification.  $h(x)$  will give us the probability that our output is 1. The "Sigmoid Function" is as follows:

$$h_{\theta}(x) = g(\theta^T x)$$

$$z = \theta^T x$$

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

The decision boundary is the line that separates the area where  $y = 0$  and where  $y = 1$ . It is created by our hypothesis function. The way our logistic function  $g$  behaves is that when its input is greater

than or equal to zero, its output is greater than or equal to 0.5.

### 3 the gradient descent ,logistic regression and neural network

#### 3.1 the gradient descent

The gradient descent equation itself is generally the same form; we just have to repeat it for our 'n' features:

repeat until convergence: {

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

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)}$$

...

}

repeat until convergence: {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \quad \text{for } j := 0 \dots n$$

}

#### 3.2 logistic regression

The cost function of logistic regression is as follows:

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

$$\text{Cost}(h_{\theta}(x), y) = -\log(h_{\theta}(x)) \quad \text{if } y = 1$$

$$\text{Cost}(h_{\theta}(x), y) = -\log(1 - h_{\theta}(x)) \quad \text{if } y = 0$$

We can compress our cost function's two conditional cases into one case, We can fully write out our entire cost function as follows:

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

#### 3.3 neural network

Our input nodes (layer 1), also known as the "input layer", go into another node (layer 2), which finally outputs the hypothesis function, known as the "output layer". We can have intermediate layers of nodes between the input and output layers called the "hidden layers." The values for each of the "activation" nodes is obtained as follows:

$$\begin{aligned}
a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\
a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\
a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \\
h_{\Theta}(x) &= a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})
\end{aligned}$$

when adding 1, it means we add a bias node, but there is no bias node in the output layer. The cost function of the neural network is as follows:

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

We have added a few nested summations to account for our multiple output nodes. In the first part of the equation, before the square brackets, we have an additional nested summation that loops through the number of output nodes.

In the regularization part, after the square brackets, we must account for multiple theta matrices. The number of columns in our current theta matrix is equal to the number of nodes in our current layer (including the bias unit). The number of rows in our current theta matrix is equal to the number of nodes in the next layer (excluding the bias unit). As before with logistic regression, we square every term.

The Back propagation Algorithm is actually a process that reconstruct the network. If we consider simple non-multiclass classification ( $k = 1$ ) and disregard regularization, the cost is computed with:

$$cost(t) = y^{(t)} \log(h_{\Theta}(x^{(t)})) + (1 - y^{(t)}) \log(1 - h_{\Theta}(x^{(t)}))$$

## 4 debug and evaluate the hypothesis

$$J_{test}(\Theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_{\Theta}(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

The test set error, for linear regression:

$$err(h_{\Theta}(x), y) = \begin{cases} 1 & \text{if } h_{\Theta}(x) \geq 0.5 \text{ and } y = 0 \text{ or } h_{\Theta}(x) < 0.5 \text{ and } y = 1 \\ 0 & \text{otherwise} \end{cases}$$

For classification:

This gives us a binary 0 or 1 error result based on a misclassification. The average test error for the test set is:

$$\text{Test Error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_{\Theta}(x_{test}^{(i)}), y_{test}^{(i)})$$

## 5 learning curves (underfitting and overfitting, precision and recall)

### 5.1 bias and variance

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  $J_{train}(\Theta)$ .

## 5.2 learning curves

Low training set size: causes  $J_{train}(\Theta)$  to be low and  $J_{CV}(\Theta)$  to be high.

Large training set size: causes both  $J_{train}(\Theta)$  and  $J_{CV}(\Theta)$  to be high with  $J_{train}(\Theta) \approx J_{CV}(\Theta)$ .

Low training set size:  $J_{train}(\Theta)$  will be low and  $J_{CV}(\Theta)$  will be high.

Large training set size:  $J_{train}(\Theta)$  increases with training set size and  $J_{CV}(\Theta)$  continues to decrease without level.

If a learning algorithm is suffering from high variance, getting more training data is likely to help.

## 5.3 what we should do

Getting more training examples: Fixes high variance

Trying smaller sets of features: Fixes high variance

Adding features: Fixes high bias

Adding polynomial features: Fixes high bias

## 6 SVM and Kernels

### 6.1 SVM

The cost function is as follows:

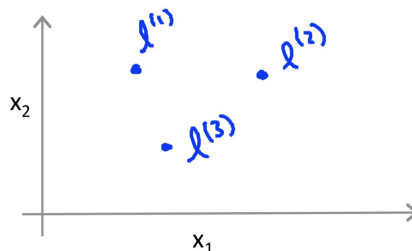
$$\min_{\theta} \frac{1}{m} \left[ \sum_{i=1}^m y^{(i)} \left( -\log h_{\theta}(x^{(i)}) \right) + (1 - y^{(i)}) \left( -\log(1 - h_{\theta}(x^{(i)})) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

The hypothesis is as follows:

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

### 6.2 Kernel

Kernel



Predict "1" when


$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

## 7 unsupervised learning, (K-means, PCA)

the unsupervised learning has no label  $y$ , we just do the learning by cluster centroid assignment and moving the centroid. The k-means algorithm is as follows:

### K-means algorithm

Input:

- $K$  (number of clusters) 
- Training set  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

$x^{(i)} \in \mathbb{R}^n$  (drop  $x_0 = 1$  convention)

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {

for  $i = 1$  to  $m$

$c^{(i)} :=$  index (from 1 to  $K$ ) of cluster centroid closest to  $x^{(i)}$

for  $k = 1$  to  $K$

$\mu_k :=$  average (mean) of points assigned to cluster  $k$

Random initialization

For  $i = 1$  to 100 {

Randomly initialize K-means.

Run K-means. Get  $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$ .

Compute cost function (distortion)

$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

}

Pick clustering that gave lowest cost  $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

## 8 the batch, stochastic and mini-batch

### Batch gradient descent

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Repeat {

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

(for every  $j = 0, \dots, n$ )

}

### Stochastic gradient descent

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$J_{train}(\theta) = \frac{1}{m} \sum_{i=1}^m cost(\theta, (x^{(i)}, y^{(i)}))$$

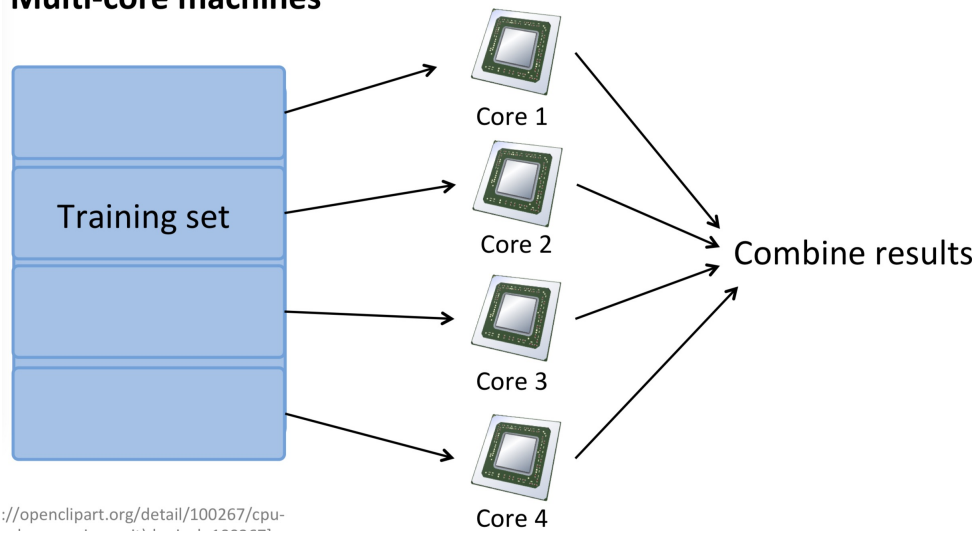
Batch gradient descent: Use all  $m$  examples in each iteration

Stochastic gradient descent: Use 1 example in each iteration

Mini-batch gradient descent: Use  $b$  examples in each iteration

## 9 map-reduce and data parrallelism,pipeline

### Multi-core machines



## 10 Plan for the Next period

1. basically grasp the idea of deep learning