Weekly Report(July.9,2018-July.15,2018)

Zhang Yuandi
15826138027@163.com

Abstract

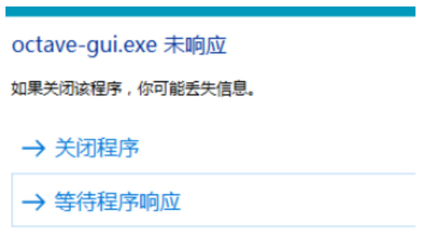
In the last week, I learned week2, week3, week4 of **Machine Learning**. To know more about js, I learned **HTML, CSS, and Javascript for Web Developers** on coursera and implemented the basic function of *wordladder.js*. About data structure, I learned B-Tree. What's more, I have my initial time with Object Detection Networks.

1 Work done in this week

1.1 Machine Learning

1.1.1 Environment Setup

Thanks for free access to MATLAB, I can use it to do powerful works. However, I tried Octave Professor Ng recommended, but got stuck when graphing. When I use the function *hist()* in Octave GUI, the outcome is just like this:

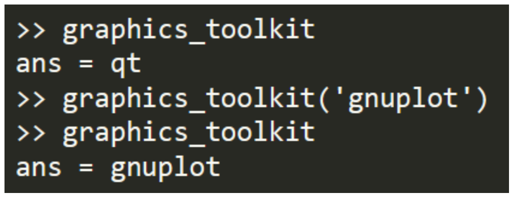

octave-gui.exe 未响应

如果关闭该程序，你可能丢失信息。

→ 关闭程序

→ 等待程序响应

Then I searched for some methods trying to fix it, like downloading *gnuplot*, which runs well on my computer, like changing *graphics_toolkit*:


```
>> graphics_toolkit
ans = qt
>> graphics_toolkit('gnuplot')
>> graphics_toolkit
ans = gnuplot
```

But this problem of graphing still exists, which makes me feel dizzy. Luckily, MATLAB works well.

1.1.2 Multivariate Linear Regression

It's a common sense that numerous objects have multiple features, so we ought to facilitate our algorithm. We get these:

- Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

- Parameters: $\theta_0, \theta_1, \ldots, \theta_n$
- Cost Function: $J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)})^2)$
- Gradient Descent($n \geq 1$): Repeat{

$$\theta_j := \theta_j - \alpha\frac{1}{m}J(\theta_0, \theta_1, \ldots, \theta_n)x_j^{(i)}$$

}(simultaneously update for every $j = 0, \ldots, n$)

Feature Scaling
To make sure features are on the same scale, we can get every feature into approximately a $-1 \leq x_i \leq 1$ range by mean normalization:

Replace x_i with $\frac{x_i - \mu_i}{s_i}$ (μ_i is the average of x_i and s_i = max - min)

Learning Rate
How to make sure gradient descent is working correctly?
We declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration.

- If α is too small, the convergence will be very slow.
- If α is too large, $J(\theta)$ may not decrease on every iteration; may not convergence.

When the outcome of our convergence looks strange, then we need to choose a better α.

Normal Equation
Normal Equation is a method to solve for θ analytically. We declare:

$$X = \begin{pmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{pmatrix}$$

Then we have

$$\theta = (X^TX)^{-1}X^Ty$$

We can make a simple comparison between Gradient Descent and Normal Equation.

Gradient Descent	Normal Equation
Need to choose α. Need many iterations. Works well even when n is large.	Don't need to choose α. Don't need to iterate. Need to compute $(X^TX)^{-1}$ Slow if n is very large.

1.1.3 Logistic Regression

- Classification: y = 0 or 1, $h_\theta(x)$ can be > 1 or < 0
- Logistic Regression: $0 \leq h_\theta(x) \leq 1$

We make our Logistic Regression:

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

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

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

Gradient Descent

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

Want $\min_\theta$ $J(\theta)$:
Repeat{

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

}

1.1.4 Overfitting

If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) \approx 0$), but fail to generalize to new examples. What can we do to fix the problem?

1. Reduce number of features.
2. Regularization.

1.1.5 Regularization Cost function

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

What if λ is set to an extremely large value?

- Algorithm works fine; setting λ to be very large cant hurt it.
- Algorithm fails to eliminate overfitting.
- Algorithm results in underfitting. (Fails to fit even training data well).
- Gradient descent will fail to converge.

Gradient Descent
Repeat{

$$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 x_j^{(i)}$$

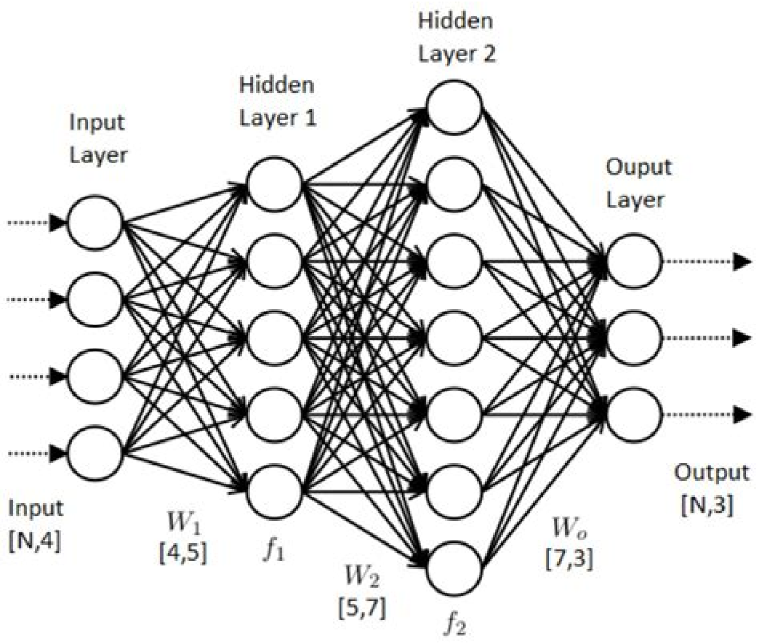
}

Normal Equation

$$\theta = (X^T X + \lambda \begin{pmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix})^{-1} X^T y$$

1.1.6 Neural Networks

Actually it is a little hard to understand fully, since I have't tried it yet and think it's somehow abstract. This picture shows clearly how neural networks work:

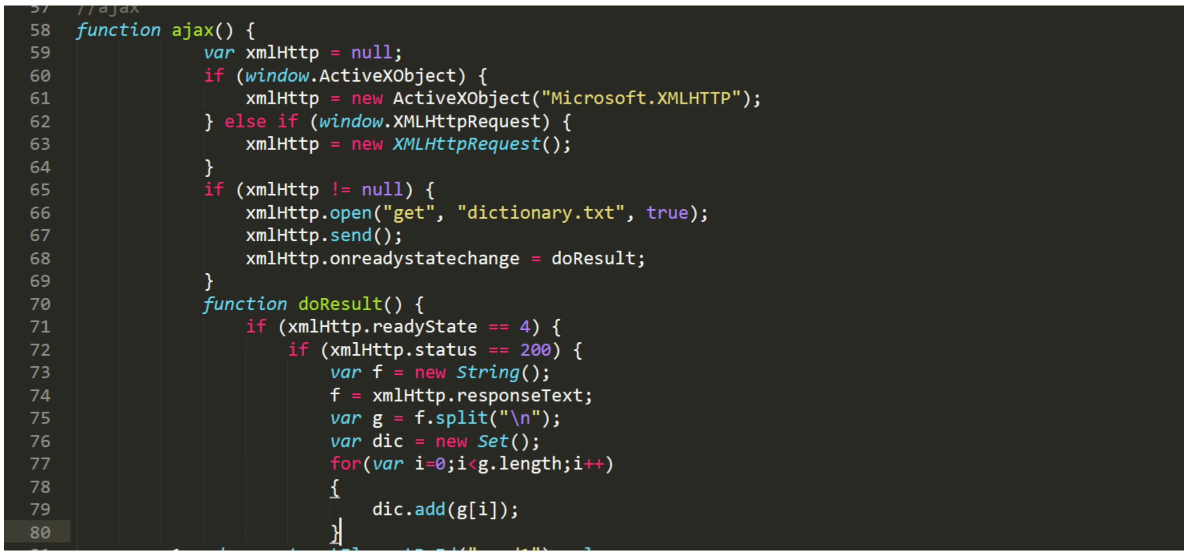


We declare that
$a_i^{(j)}$ = "activation" of unit i in layer j
$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer j + 1
If network has s_j units in layer j, s_{j+1} units in layer j + 1 , then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$. The steps look like this:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ \cdots \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(3)} \\ a_1^{(3)} \\ a_2^{(3)} \\ \cdots \end{bmatrix} \rightarrow \cdots \rightarrow \begin{bmatrix} h_\Theta(x)_1 \\ h_\Theta(x)_2 \\ h_\Theta(x)_3 \\ h_\Theta(x)_4 \end{bmatrix}$$

1.2 Js

The biggest problem I met is reading files. I solved it with the help of Ajax. Here is part of the function:



```
//ajax
function ajax() {
            var xmlHttp = null;
            if (window.ActiveXObject) {
                xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
            } else if (window.XMLHttpRequest) {
                xmlHttp = new XMLHttpRequest();
            }
            if (xmlHttp != null) {
                xmlHttp.open("get", "dictionary.txt", true);
                xmlHttp.send();
                xmlHttp.onreadystatechange = doResult;
            }
            function doResult() {
                if (xmlHttp.readyState == 4) {
                    if (xmlHttp.status == 200) {
                        var f = new String();
                        f = xmlHttp.responseText;
                        var g = f.split("\n");
                        var dic = new Set();
                        for(var i=0;i<g.length;i++)
                        {
                            dic.add(g[i]);
                        }
```

And the function *replace()* of a string doesn't change the original string itself. Also, the shallow copy of an object should be highly paid attention.

1.3 Data Structure

The B−tree is a generalization of a binary search tree in that a node can have more than two children. It is very efficient for storage systems that read and write relatively large blocks of data, such as discs. The basic rule for a B−tree are as follows:

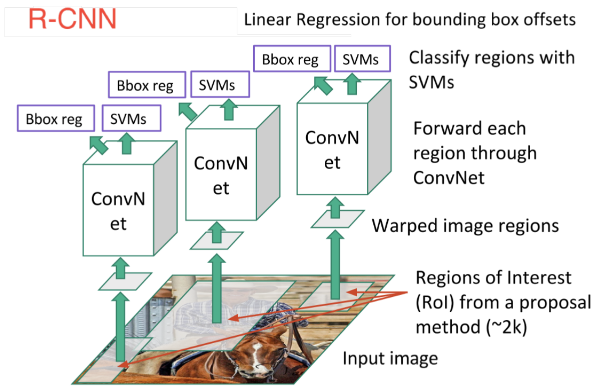
1. Every node has at most m children.
2. Every non-leaf node (except root) has at least [m/2] children.
3. The root has at least two children if it is not a leaf node.
4. A non-leaf node with k children contains k−1 keys.
5. All leaves appear in the same level.

1.4 Detection Networks

This is quite hard to understand at the beginning, but I'm trying to ignore some details and focus on the procedure. I learned some basic steps of some important detection model.

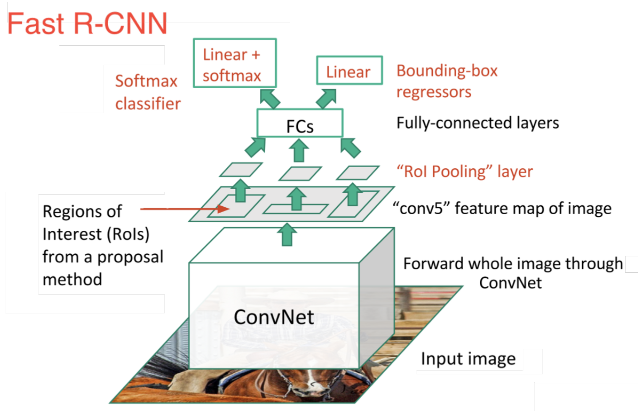
1.4.1 R−CNN

Region-based Convolutional Neural Network(R−CNN), consisted of 3 simple steps, which can be illustrated like this:


R-CNN
Linear Regression for bounding box offsets
Bbox reg
SVMs
Classify regions with SVMs
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

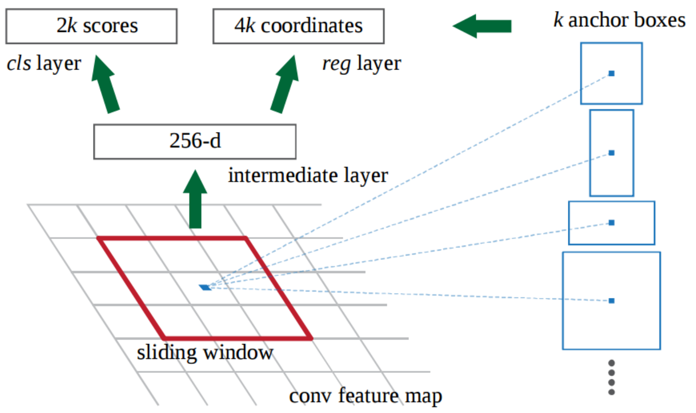
1.4.2 Fast R−CNN

The reason why it is fast is that we can train just one CNN for the entire image. The steps can be illustrated like this:



1.4.3 Faster R−CNN

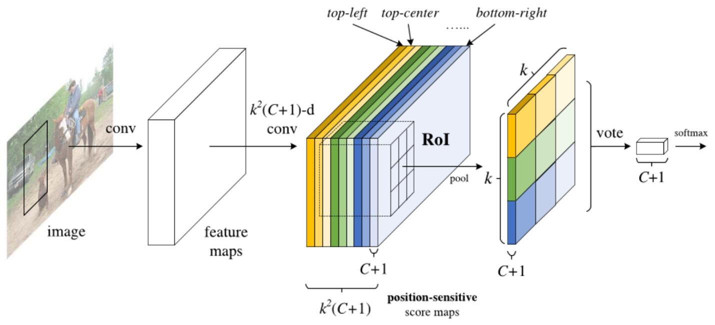
Specifically, it introduced the region proposal network (RPN). Here is how RPN works:



But to be honest, this is somehow hard to understand, at least I haven't understand it fully. In a sense, **Faster R−CNN = RPN + Fast R−CNN**.

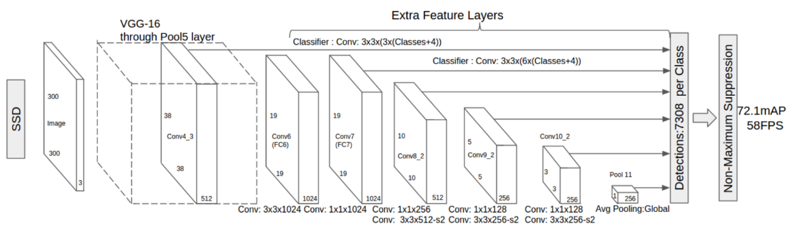
1.4.4 R−FCN

Region-based Fully Convolutional Net(R−FCN) uses the position−sensitive score maps, which are convolutional feature maps that have been trained to recognize certain parts of each object. to help us compromise between location and location variance. It can be illustrated like this:



1.4.5 SSD

Single−Shot Detector(SSD) provides enormous speed gains over Faster R−CNN in a markedly different manner. SSD generates regions of interest and classifies regions in a "single shot", that is, simultaneously predicting the bounding box and the class as it processes the image. It can be illustrated like this:



2 Plans for Next Week

1. Learn the course of week5, week6, week7 of **Machine Learning**.

2. Write a data base in C++ code.
3. Learn more about Detection Networks.