Computer Vision – Programming Project 2

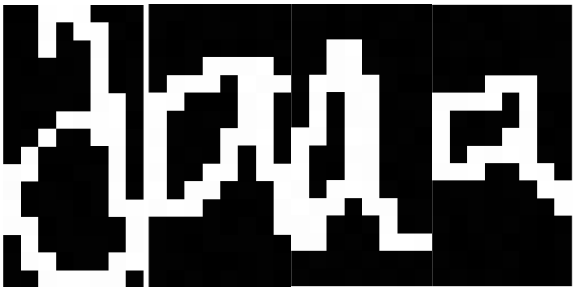
王順興　0210184

Introduction

In this project, I will be dealing with multi-class letter recognition problem. I start by implement Linear Softmax Regression, then Sigmoid based Neural Network with one hidden layer, and lastly, Dual Softmax Regression.

i. The Dataset

The dataset I use can be found here(http://ai.stanford.edu/~btaskar/ocr/). Each sample is a 16x8 binary image (0 and 1) of an English alphabet (letter) taken from a written word (as opposed to test subjects writing individual letters), this makes the recognition more difficult.



Some sample of 'a'

Method

i. Linear Softmax Regression

For Linear Softmax Regression I followed Stanford university's UFLDL tutorial. The likelihood function, or hypothesis, is the Softmax function:

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix}$$

The cost function, or loss function, is the cross entropy:

$$J(\theta) = -\left[\sum_{i=1}^{m} \sum_{k=1}^{K} 1\left\{ y^{(i)} = k \right\} \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})} \right]$$

And so the resulting gradient for the gradient descent and the implementation of gradient descent are the same as the ones mentioned in textbook and algorithm handbook:

$$\frac{\partial L}{\partial \phi_n} = -\sum_{i=1}^{I} (y_{in} - \delta[w_i - n]) \mathbf{x}_i$$

To avoid over-fitting, I've also tried adding weight decay:

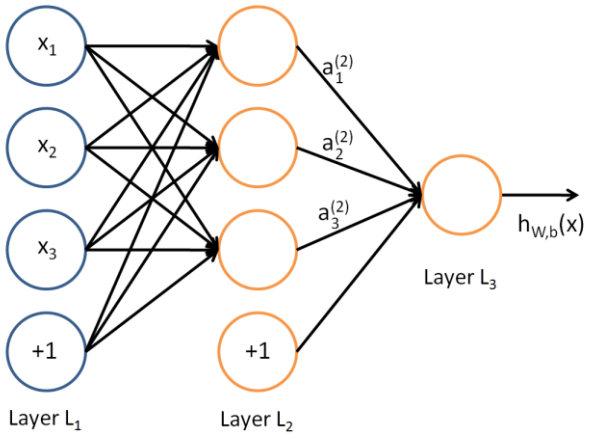
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\left\{y^{(i)} = j\right\} \log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}\right] + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2$$

But as it turns out, over-fitting isn't really an issue with this dataset.

ii. Sigmoid Neural Network

I implemented Sigmoid Neural Network following the same Stanford university's UFLDL tutorial. The goal is to have a Network with one hidden layer, then replace the output layer with the Softmax Regression before, to see the difference between multiple Sigmoid neuron and a single Softmax Neuron.

A simplified version of the network:



For each neuron in each layer with ***x*** as input, there are 2 parameters regarding input:

$Weight$: $\boldsymbol{W}$ - a vector with $\dim(\boldsymbol{x})$ elements

$Bias$: b - a scalar

The job of a neuron is to produce the $Activation$: a - a scalar,

where

$$a = sigmoid(\boldsymbol{W}\boldsymbol{x} + b)$$

and all the activations ***a*** of a layer become the input ***x*** of the next.

To learn the parameters ***W*** and *b*:

- Randomly initialize ***W*** and *b* - **Symmetry Breaking**
- Feed a sample into the network and gather the output – **Forward Propagation**
- From the last layer, calculate the error for output layer and the weighted error for hidden layers of each neuron – **Backpropagation I**
- Use the error/weighted error for each neuron to update the ***W*** and *b* - **Backpropagation II**

iii. Dual Softmax Regression

The difference between Dual Softmax Regression and Linear Softmax Regression is that, instead of making up parameters with numerous interations, we use the existing dataset to construct the parameters.

However, there is a huge drawback when learning, each sample is associated with a weight, so the more sample you have, the more parameter you have. I am using 200 samples with 26 classes as my training set, so I have a total of 5200 parameters, not too far away from Linear Softmax Regression, which has 128x26 = 3328 parameters. For predicting, the drawback is that you'll need to have the training dataset.

Turning Linear Softmax Regression into Dual Softmax Regression is quite easy. In the former, the activation for Softmax function is

$$a_n = \boldsymbol{\phi}_n^T \mathbf{x}$$ so we turn it into $$a_n = \boldsymbol{\psi}^T \mathbf{X}^T \mathbf{x}$$

and the cost function's gradient follows:

$$\frac{\partial L}{\partial \boldsymbol{\phi}_n} = -\sum_{i=1}^{I} (y_{in} - \delta[w_i - n]) \mathbf{X}^T \mathbf{x}_i$$

Technical Details and Issues

When using Gradient Descent to find parameters in Linear Softmax Regression:

for i=1 **to** I **do**
  // Compute prediction y
  $\mathbf{y}_i = \mathbf{softmax}[\phi_1^T \mathbf{x}_i, \phi_2^T \mathbf{x}_i, \ldots \phi_N^T \mathbf{x}_i]$
  // Update log likelihood
  $L = L - \log[y_{i,w_i}]$
  // Update gradient and Hessian
  for n=1 **to** N **do**
  end
end

The pseudo code given by algorithm handbook iterates training samples one by one. However, because Softmax function can be easily modified to take a matrix of samples as input, the for-loop in the example is not necessary.

Similarly, in Neural Network, the computation on each layer can also be turn into matrix operation.

Neural Network not working:

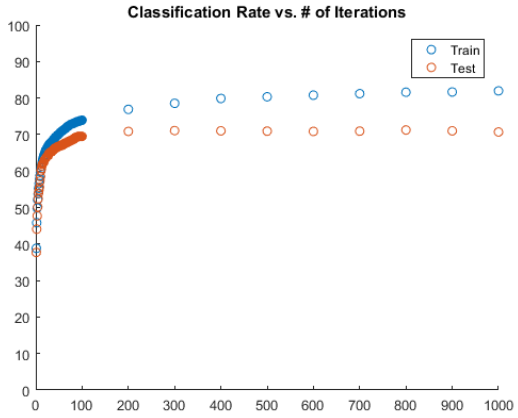
For the Neural Network I have 128/64/26 neurons at each layer. For wieght and bias initilization, at first I used Normal distributed random number, then switched to Uniform distributed random, because the former seems to kept falling into local minimum. However, even after many tweaking, my implementation of NN still wouldn't work. So I decided to left it until I have a formal understanding of it.

Slow iteration in Dual Softmax Regression:

In Softmax Regression, we need to evaluates the activation of each class, and unlike Linear Softmax Regression, the evaluation cannot be turned into matrix operation (atleast I can't figure it out), so iterations takes significant longer time.

Result

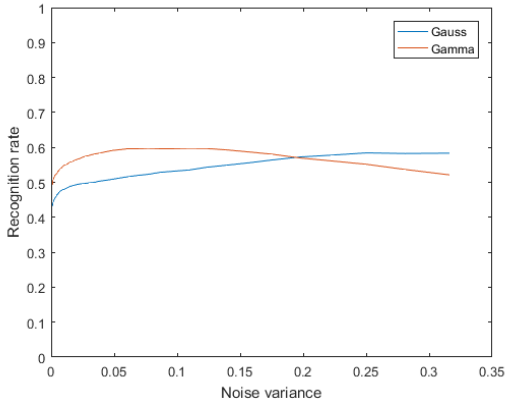
For Linear Softmax Regression:



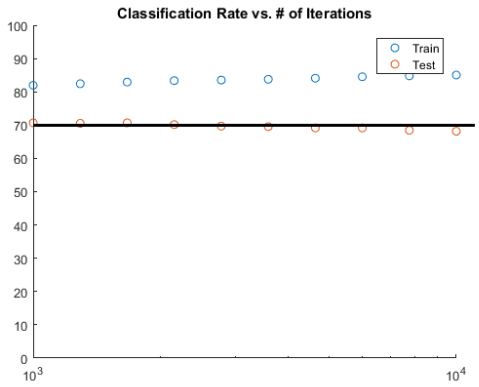
Since I'm using Gradient Desecent to find the parameters, the first thing I check is the Classification rate vs. # of Iterations.

After playing around with the learning rate, the parameters converges just after ~100 iterations.

Also, comparing the result to the last project, where I implemented Gaussian and Gamma Maximum Likelihood Generative model for Classificaion, using Linear Softmax Regression, the classfication rate increases by ~10%.

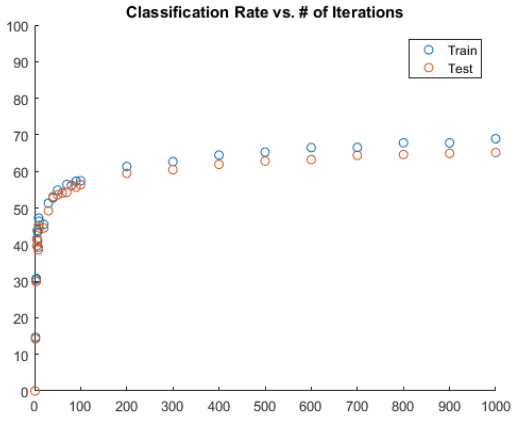


This is due to the fact that Linear Softmax Regression is a Discriminant model, and Generative models without proper prior is expeceted to perform no better.



Next, I want see when will the over-fitting occurs, and as you can see, the overfitting only occurs after ~3000 iterations. So the weight decay isn't really necessary, I would have run out of patience and stop it anyway.

For Dual Softmax Regression:



In Dual Softmax Regression, the learning rate has to be kept very small, otherwise the parameter update will overshoot every iteration. This combined with the fact that the iterations have to be done with for-loop, makes iteration super slow.

The parameter converges around several hundered iterations, however the classification rate are capped at ~65%, probably because of the high variance of samples in the dataset.