Problem Set 4

Weifang Zheng 50248714

1 Problem 1

Demonstrate that a neural network to maximize the log likelihood of observing the training data is one that has softmax output nodes and minimizes the criterion function of the negative log probability of training data set:

t_n is the label of image x_n.

y is the prediction of x_n according to function f, y = $f(x_n) = \underset{m}{arg\max} P(\boldsymbol{t_n} = m|\boldsymbol{x_n}; \mathbf{W})$

(m is the index of labels (0,1,2,3,4,5,6,7,8,9), n is the index of input images)

We can define the loss function Loss = $\begin{cases} 1 & t_n = y \\ 0 & t_n \neq y \end{cases}$

Total loss equals to the number of images which are classified into the right classes. To maximum the Loss, we need to determine the W to make each image can be assigned to the right classes.

$$\mathcal{L}(W) = \prod_n \prod_{m=0}^{9} p(t_n = m|x_n; W)$$

$$\max_w \mathcal{L}(W) = \max \prod_n \prod_{m=0}^{9} p(t_n = m|x_n; W)$$

$$\max_w \log\mathcal{L}(W) = \max \log\left(\prod_n \prod_{m=0}^{9} p(t_n = m|x_n; W)\right)$$

$$= \max \sum_n \sum_{m=0}^{9} \log p(t_n = m|x_n; W)$$

softmax function:

$S(z_n) = \frac{e^{z_n^j}}{\sum_{k=1}^{K} e^{z_n^k}}$ the output of output layer is a [1,n] vector.

$y = \underset{j}{\operatorname{argmax}} \mathbf{S}(\boldsymbol{z_n})$. The index of max element in the vector will be chosen as the prediction.

To maximum the log likelihood can be transferred to minimum the negative log likelihood.

$\min \sum_n \sum_{m=0}^{9} -\log p(t_n = m|x_n; W) = \min \sum_n \sum_{m=0}^{9} -\log p(t_n = m|x_n; W)$

p(x_n) is evidence which are 1/n

$$\min \sum_n \sum_{m=0}^{9} -\log p(t_n = m|x_n; W) \to -\log p(\{(x_n, t_n): n = 1, 2, \cdots, \}; w)$$

$$J_0(w) = -\log p(\{(x_n, t_n): n = 1, 2, \cdots, \}; w) = -\log\left(\prod_n \prod_{m=0}^{9} p(t_n = m|x_n; W)\right)$$

Demonstrate that a neural network to maximize the posterior likelihood of observing the training data given a Gaussian prior of the weight distribution is one that minimizes the criterion function with L2 regularization

$(x_1, y_1),\ (x_2, y_2),\ \ldots(x_n, y_n)$

$y_n = \beta x_n + \epsilon$

ϵ is the Gaussian noise with mean 0 and variance σ^2

Gaussian likelihood: $\prod_{n-1}^{N} N(y_n|\beta x_n, \sigma^2)$

Regularize parameter W by imposing the Gaussian prior $N(\beta|0, \lambda^{-1})$

Where λ is a strictly positive scalar. Hence, combining the likelihood and the prior we have

$\prod_{n-1}^{N} N(y_n|\beta x_n, \sigma^2)\ N(\beta|0, \lambda^{-1})$

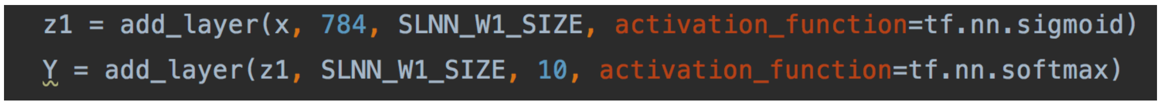
Logarithm and dropping some constants:$\sum_{n=1}^{N} -\frac{1}{\sigma^2}(y_n - \beta x_n)^2 - \lambda\beta^2$+const

If we maximize the above expression with respect to β we get the so called maximum a-posteriori estimate for β. In this expression it becomes apparent why the Gaussian prior can be interpreted as a L2 regularisation term.

reference: https://stats.stackexchange.com/questions/163388/l2-regularization-is-equivalent-to-gaussian-prior

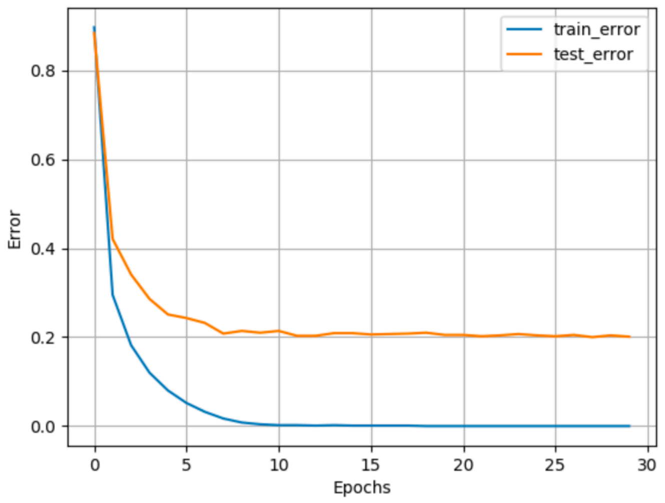
2 Problem 2

a. Single hidden layer NN

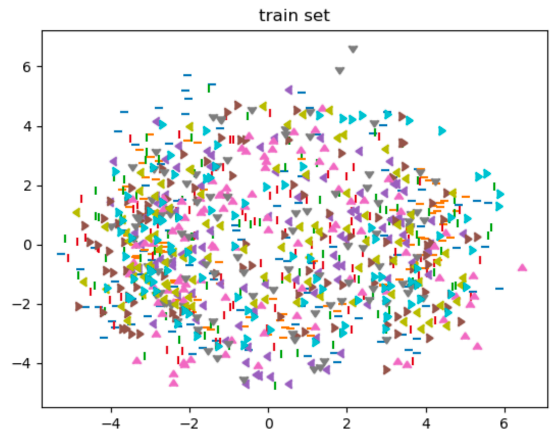


```
z1 = add_layer(x, 784, SLNN_W1_SIZE, activation_function=tf.nn.sigmoid)
Y = add_layer(z1, SLNN_W1_SIZE, 10, activation_function=tf.nn.softmax)
```

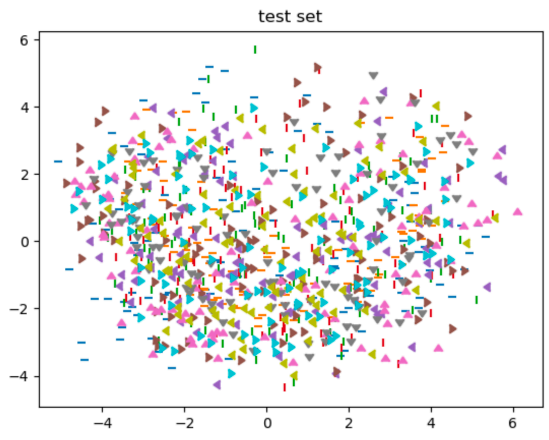
Plot:



criterion function on training data set（PCA）:



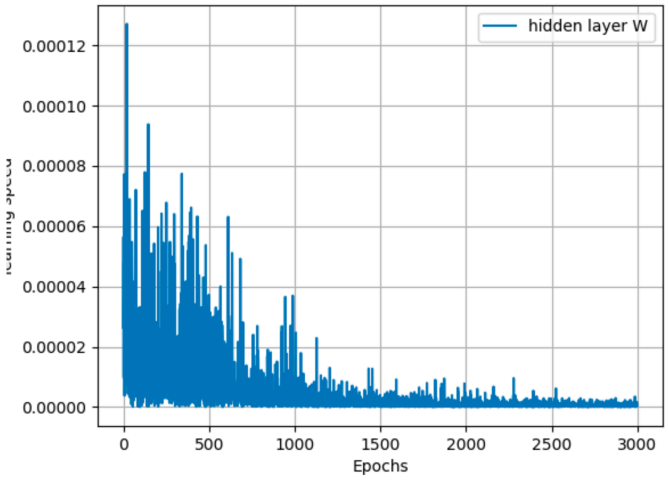
criterion function on testing data（PCA）:



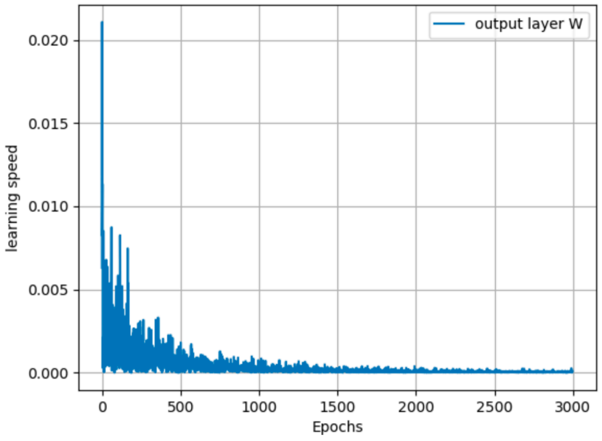
Learning speed:

Use L1 regularizer to estimate the learning speed.

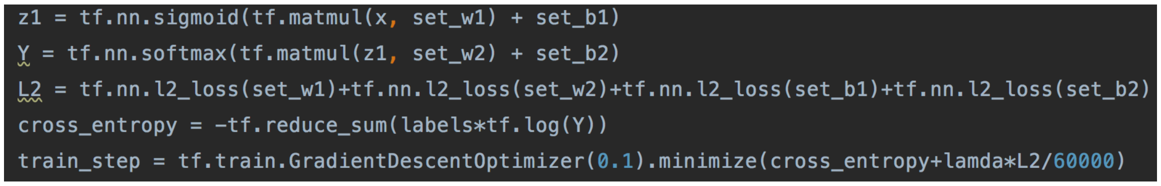
learning speed of W in hidden layer



learning speed of W in output layer

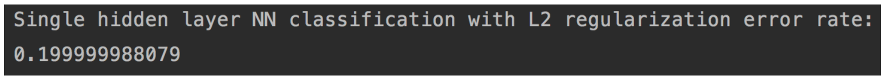


b. Single hidden layer NN with L2 regularizer



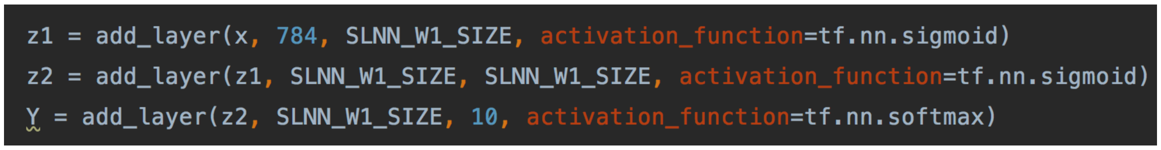
```
z1 = tf.nn.sigmoid(tf.matmul(x, set_w1) + set_b1)
Y = tf.nn.softmax(tf.matmul(z1, set_w2) + set_b2)
L2 = tf.nn.l2_loss(set_w1)+tf.nn.l2_loss(set_w2)+tf.nn.l2_loss(set_b1)+tf.nn.l2_loss(set_b2)
cross_entropy = -tf.reduce_sum(labels*tf.log(Y))
train_step = tf.train.GradientDescentOptimizer(0.1).minimize(cross_entropy+lamda*L2/60000)
```

$$\frac{\lambda}{2n} \| w \|^2$$

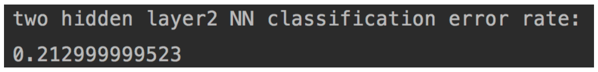


```
Single hidden layer NN classification with L2 regularization error rate:
0.199999988079
```

Two hidden layers NN

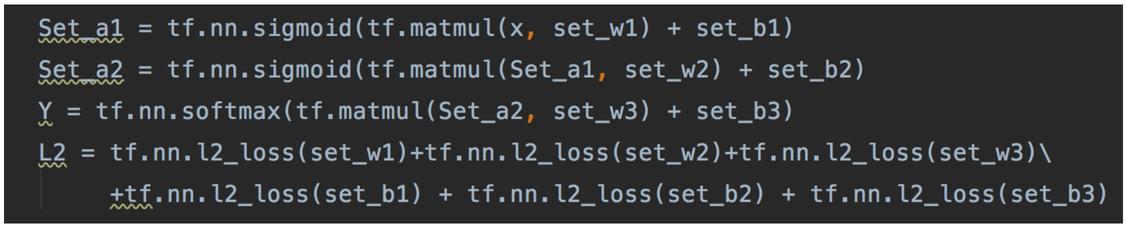


```
z1 = add_layer(x, 784, SLNN_W1_SIZE, activation_function=tf.nn.sigmoid)
z2 = add_layer(z1, SLNN_W1_SIZE, SLNN_W1_SIZE, activation_function=tf.nn.sigmoid)
Y = add_layer(z2, SLNN_W1_SIZE, 10, activation_function=tf.nn.softmax)
```

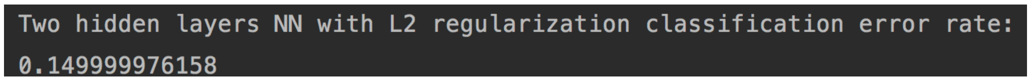


```
two hidden layer2 NN classification error rate:
0.212999999523
```

Two hidden layers NN with L2 regularizer

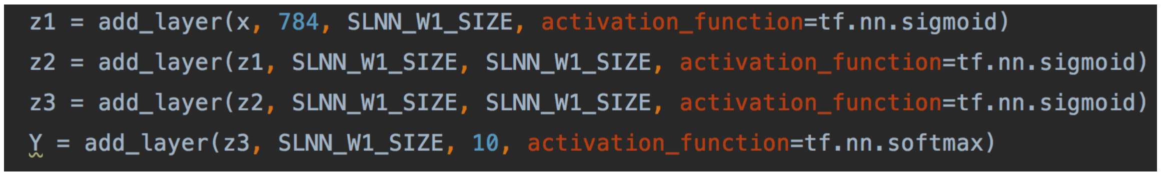


```
Set_a1 = tf.nn.sigmoid(tf.matmul(x, set_w1) + set_b1)
Set_a2 = tf.nn.sigmoid(tf.matmul(Set_a1, set_w2) + set_b2)
Y = tf.nn.softmax(tf.matmul(Set_a2, set_w3) + set_b3)
L2 = tf.nn.l2_loss(set_w1)+tf.nn.l2_loss(set_w2)+tf.nn.l2_loss(set_w3)\
     +tf.nn.l2_loss(set_b1) + tf.nn.l2_loss(set_b2) + tf.nn.l2_loss(set_b3)
```

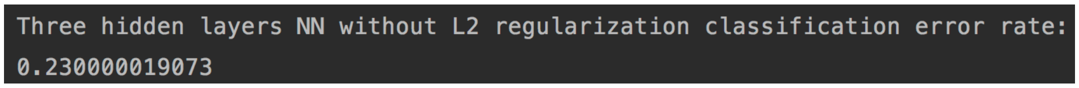


```
Two hidden layers NN with L2 regularization classification error rate:
0.149999976158
```

Three hidden layers NN

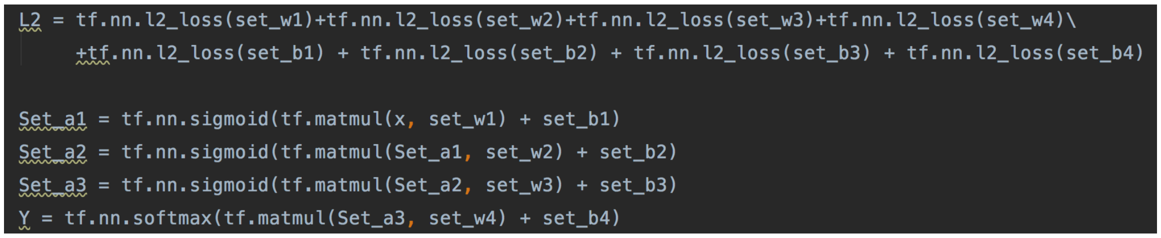


```
z1 = add_layer(x, 784, SLNN_W1_SIZE, activation_function=tf.nn.sigmoid)
z2 = add_layer(z1, SLNN_W1_SIZE, SLNN_W1_SIZE, activation_function=tf.nn.sigmoid)
z3 = add_layer(z2, SLNN_W1_SIZE, SLNN_W1_SIZE, activation_function=tf.nn.sigmoid)
Y = add_layer(z3, SLNN_W1_SIZE, 10, activation_function=tf.nn.softmax)
```

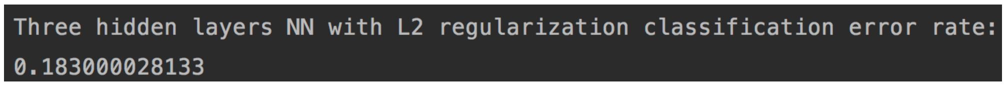


```
Three hidden layers NN without L2 regularization classification error rate:
0.230000019073
```

Three hidden layers NN with L2 regularizer



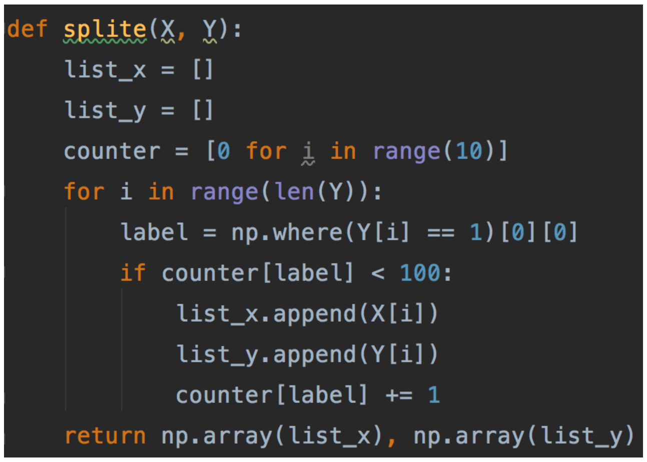
```
L2 = tf.nn.l2_loss(set_w1)+tf.nn.l2_loss(set_w2)+tf.nn.l2_loss(set_w3)+tf.nn.l2_loss(set_w4)\
     +tf.nn.l2_loss(set_b1) + tf.nn.l2_loss(set_b2) + tf.nn.l2_loss(set_b3) + tf.nn.l2_loss(set_b4)

Set_a1 = tf.nn.sigmoid(tf.matmul(x, set_w1) + set_b1)
Set_a2 = tf.nn.sigmoid(tf.matmul(Set_a1, set_w2) + set_b2)
Set_a3 = tf.nn.sigmoid(tf.matmul(Set_a2, set_w3) + set_b3)
Y = tf.nn.softmax(tf.matmul(Set_a3, set_w4) + set_b4)
```



```
Three hidden layers NN with L2 regularization classification error rate:
0.183000028133
```

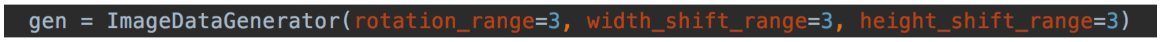
c. CNN

1000 training images (100 images per digit)

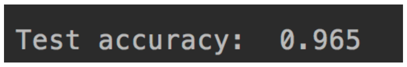


```
def splite(X, Y):
    list_x = []
    list_y = []
    counter = [0 for i in range(10)]
    for i in range(len(Y)):
        label = np.where(Y[i] == 1)[0][0]
        if counter[label] < 100:
            list_x.append(X[i])
            list_y.append(Y[i])
            counter[label] += 1
    return np.array(list_x), np.array(list_y)
```

Regularize the training of neural network through augment your selection of 1000 images by rotating them for 1-3 degrees clockwise and counter clockwise, and shifting them for 3 pixels in 8 different directions.



```
gen = ImageDataGenerator(rotation_range=3, width_shift_range=3, height_shift_range=3)
```



```
Test accuracy:  0.965
```