Machine Learning HW2

b03901056 孫凡耕 (kaggle : b03901056_daikon)

Q1：Logistic Regression Function：

- Amount of data: I used 400 data to update parameters once (i.e. batch size = 400)
- Method of Gradient Descent: I tried simple gradient descent, adagrad and adadelta. I found that adadelta performs the best overall, since simple gradient descent and adagrad improves too slowly. So the code I pasted here represents my implementation of adadelta.

```
#w[:-1]: the weight of corresponding feature, w[-1] is the bias term
w = np.zeros((train_dat.shape[1], 1))  #initialize w and b with zeros
#adding a column of 1s represents the bias term b
train_dat.append([ 1. for i in range(train_len) ])    #bias term
#btch_cnt: number of batches, btch_sz: size of one batch
#head: start index of this batch, tail: end index of this batch
#res[i]: the value of one particular i for (ŷ^i − σ(∑ w_j x_j^i)), note that w includes the bias term
#gra[i]: the gradient of w[i]
for j in range(btch_cnt):
    head = j*btch_sz
    tail = (j+1)*btch_sz
    res = np.transpose(train_ans[head:tail,:]-(1/(1+np.exp(-np.dot(train_dat[head:tail,:], w)))))
    gra = -np.transpose(np.dot(res, train_dat[head:tail,:]))
#adad_g: the sum of squares of previous gradients with an attenuation coefficient ρ(rho)
#adad_d: the sum of squares of previous △w with an attenuation coefficient ρ(rho)
#△w : the changes of w for every update
#epsilon: to avoid division by zero while calculating △w
    ada_g = ada_g * rho + (1. - rho) * (gra ** 2)
    dw = - gra * (ada_d + eps) ** 0.5 / (ada_g + eps) ** 0.5
    ada_d = rho * ada_d + (1. - rho) * (dw ** 2)
    w += dw
```

Q2：My second method -> Neural Network：

- Network Structure: Fully-Connected. Input dimension: between 1 and 57. Output dimension: 2. Any amount of hidden layers with any amount of neurons in each layers.
- Data Preprocessing: Preprocess the data of each feature to zero-mean and unit variance
- Weight Initialize: Random Gaussian with zero mean and unit variance
- Loss Function: Cross Entropy
- Gradient Descent : backpropagation
- Optimizer: adadelta, adagrad, RMSprop, adam, nesterov momentum
- Activation Function: sigmoid, arctan, tanh, relu, leaky relu
- Regularization Method: L2 regularization, dropout
- Hyper-parameters: features to train; nb_epochs; batch size; structure of network; hyper-parameters in gradient descent optimizer; alpha in leaky relu; lamba in L2 regularization; dropout rate in dropout

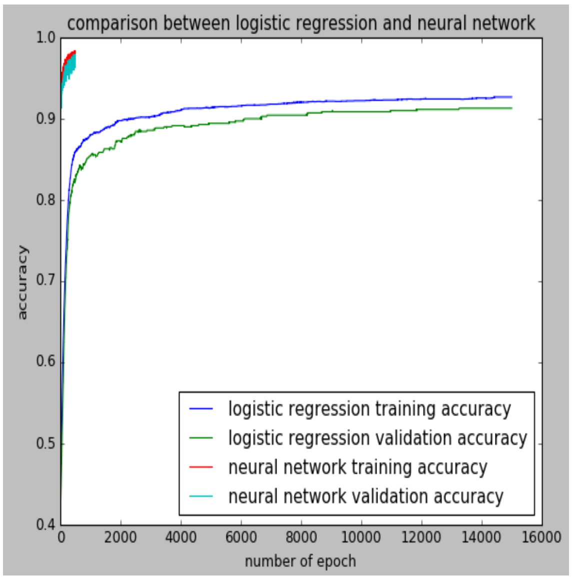
✧ **For later discussion:**

training set: random 3301 data, validation set: the rest 700 (for one discussion, the sets are identical)

features: always all 57 features

batch size: always 400

Q3：Which one is best：



number of epochs: 15000 for logistic regression, 500 for neural network

optimizer:

both adadelta with rho = 0.95, but epsilon for logistic regression is 1e-14, for neural network is 1e-6 (for logistic regression, epsilon = 1e-14 is trainable, but 1e-6 is not)

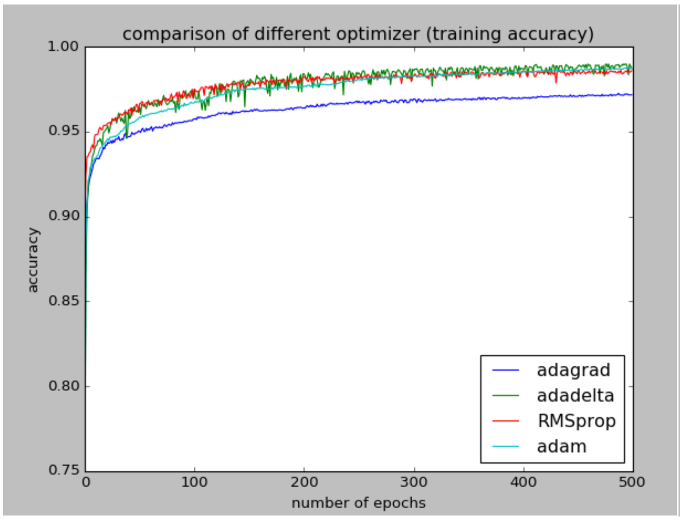
neural network hyper-parameters:

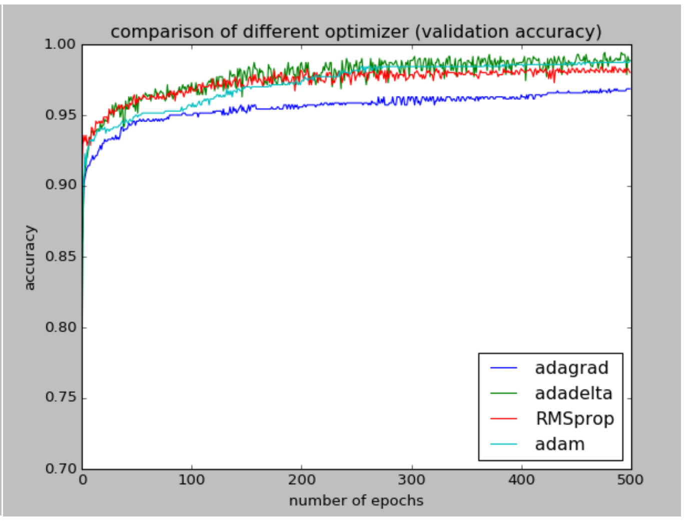
1 hidden layer with 57 neurons, drop out rate = 0.05, lambda = 0.001,

activation function: leaky relu with alpha = 0.01

➢ We can see that the results of neural network is way better than logistic regression even spending much lesser epochs. For actual experiment, the training accuracy of neural network can go up to 0.995 but only 0.96 for logistic regression, and the validation accuracy of neural network can reach 0.96 but only 0.93 for logistic regression.

Q4：Discussion on Different Method of Gradient Descent on Neural Network：





note: my implementation of nesterov momentum can't work on this hyper-parameters

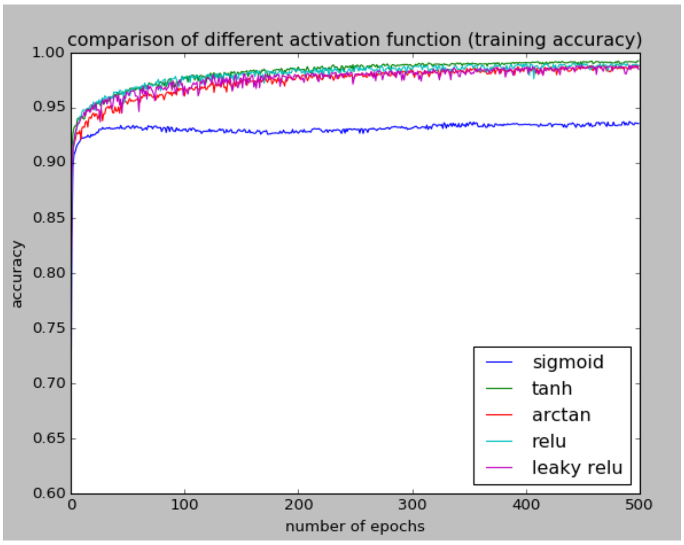
number of epochs: 500

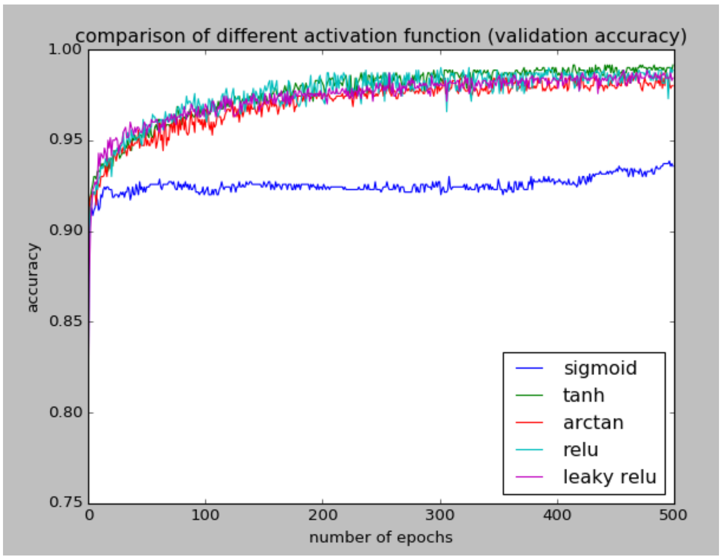
neural network hyper-parameters: 1 hidden layer with 57 neurons, drop out rate = 0, lambda = 0, leaky relu with alpha = 0.01

parameters for activation function: adagrad(learning rate=1e-2), adadelta(rho=0.95, epsilon=1e-6), RMSprop(learning rate=2e-3, rho=0.99, epsilon=1e-6), adam(learning rate=6e-4, beta1=0.9, beta2=0.999, epsilon=1e-8)

➢ We can see that adadelta, RMSprop, adam are all better than adagrad, because they all solved the monotonically decreasing learning rate of adagrad. In this case the three of them all perform similarly, except that adam is the most smooth one, whereas adadelta vibrates the most. We can also observe that the vibration on validation set is much more severe than on the training set, which is expected.

Q5：Discussion on Different Activation Function on Neural Network：





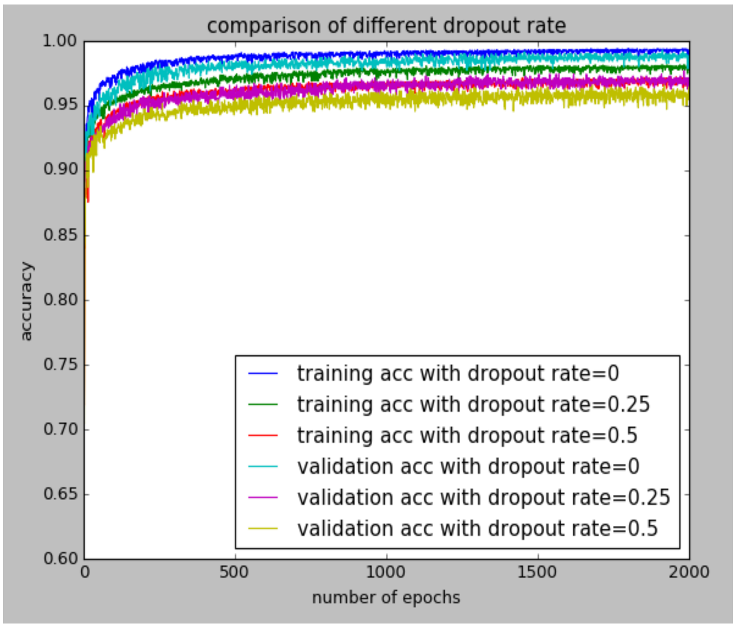
number of epochs: 500

neural network hyper-parameters: 1 hidden layer with 57 neurons, drop out rate = 0, lambda = 0, leaky relu with alpha = 0.01

optimizer: adadelta with rho = 0.95, epsilon = 1e-6

- The reason why using sigmoid in this case is not optimal is not known, but we can still see the problem of sigmoid function: the vanishing gradient. From the graph shown, sigmoid function learning slower even in the early stage.

Q6：Discussion on dropout on Neural Network：



number of epochs: 2000

neural network hyper-parameters:

1 hidden layer with 57 neurons, lambda = 0,

leaky relu with alpha = 0.01

optimizer: adadelta with rho = 0.95, epsilon = 1e-6

- It's obvious that the higher the drop rate, the worse the training accuracy is, which is expected. However, the validation accuracy is even worse, which is not expected since I originally thought that 0.5 dropout rate will be optimal. Actually I found out that dropout rate of about 0.01 will be optimal in this case, which is definitely not a typical dropout rate.