BACK-PROPAGATION EQUATIONS

BINARY CLASSIFICATION WITH CROSS-ENTROPY (DEVIANCE) LOSS FUNCTION AND RELU (MAX(0, x)) ACTIVATION FUNCTION

$$\hat{y}(x,w) = \sigma\left(\sum_{j=1}^{M} w_j^{(2)}\, h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_n^{i} + w_{j0}^{(1)}\right) + w_0^{(2)}\right)$$

$$\mathrm{Err}(w) = -\sum_{n=1}^{N}\left\{ y_n \log \hat{y}(x_n,w) + (1-y_n)\log\left(1-\hat{y}(x_n,w)\right)\right\}$$

$$w^{t+1} = w^{t} - \eta \nabla \mathrm{Err}(w)$$

where $\eta \geq 0$ is the learning rate and can also be decreasing as a function of t.

EQUATIONS FOR THE DERIVATIVES FOR ONE TRAINING POINT (x, y)

FIRST, LET US COMPUTE THE DERIVATIVE OF $\sigma(x)$

$$\frac{d}{dx}\left(\sigma(x)\right) = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right)$$

$$= \frac{-1}{(1+e^{-x})^2} \times \left(-e^{-x}\right) = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{e^{x}}{(1+e^{x})^2}$$

$$\frac{\partial Err}{\partial w_{ji}^{(1)}} = -\left\{ \frac{y}{\hat{y}(x, w)} \cdot \frac{e^{a_j^{(2)}}}{(1+e^{a^{(2)}})^2} \cdot w_j^{(2)} h'(a_j^{(1)}) x^i \right.$$

$$\left. + \frac{(1-y)}{1-\hat{y}(x, w)} \cdot \frac{(-1) \cdot e^{a^{(2)}}}{(1+e^{a^{(2)}})^2} \cdot w_j^{(2)} h'(a_j^{(1)}) x^i \right\}$$

Here, $h(a_j^{(1)}) = \max(0, a) = \begin{cases} a_j^{(1)} & \text{if } a_j^{(1)} > 0 \\ 0 & \text{otherwise} \end{cases}$

$$\boxed{\begin{aligned} &\Rightarrow h'(a_j^{(1)}) = \begin{cases} 1 & \text{if } a_j^{(1)} > 0 \\ 0 & \text{if } a_j^{(1)} < 0 \end{cases} \\ &\text{and } a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} x^i + w_{j0}^{(1)} \\ &a^{(2)} = \sum_{j=1}^{M} w_j^{(2)} h(a_j^{(1)}) + w_0^{(2)} \end{aligned}}$$

SIMILARLY,

$$\frac{\partial Err}{\partial w_{j0}^{(1)}} = -\left\{ \frac{y}{\hat{y}(x,w)} \cdot \frac{e^{a^{(2)}}}{(1+e^{a^{(2)}})^2} \cdot w_j^{(2)} h'(a_j^{(1)}) \right.$$

$$\left. + \frac{(1-y)}{1-\hat{y}(x,w)} \cdot \frac{(-1)\, e^{a^{(2)}}}{(1+e^{a^{(2)}})^2} \cdot w_j^{(2)} h'(a_j^{(1)}) \right\}$$

$$\frac{\partial Err}{\partial w_j^{(2)}} = -\left\{ \frac{y}{\hat{y}(x,w)} \cdot \frac{e^{a^{(2)}}}{(1+e^{a^{(2)}})^2} \cdot h(a_j^{(1)}) \right.$$

$$\left. + \frac{(1-y)}{1-\hat{y}(x,w)} \cdot \frac{(-1)\, e^{a^{(2)}}}{(1+e^{a^{(2)}})^2} \cdot h(a_j^{(1)}) \right\}$$

$$\frac{\partial Err}{\partial w_0^{(2)}} = -\left\{ \frac{y}{\hat{y}(x,w)} \cdot \frac{e^{a^{(2)}}}{(1+e^{a^{(2)}})^2} \right.$$

$$+ \frac{(1-y)}{1-\hat{y}(x,w)} \cdot \frac{(-1)\, e^{a^{(2)}}}{(1+e^{a^{(2)}})^2}$$

SIMPLIFIED BACK-PROPAGATION EQUATIONS:

i)

$$\frac{\partial Err}{\partial w_{ji}^{(1)}} = -\sum_{n=1}^{N} \left\{ \left(Y_n(1-\hat{Y}) - (1-Y_n)\hat{Y} \right) w_j^{(2)} h'(a_j^{(1)}) x_n^i \right\}$$

$$= \begin{cases} -\sum_{n=1}^{N} \left\{ \left(Y_n(1-\hat{Y}) - (1-Y_n)\hat{Y} \right) w_j^{(2)} x_n^i \right\} & \text{if } a_j^{(1)} > 0 \\ 0 & \text{if } a_j^{(1)} < 0 \end{cases}$$

ii)

$$\frac{\partial Err}{\partial w_j^{(2)}} = -\sum_{n=1}^{N} \left\{ \left(Y_n(1-\hat{Y}) - (1-Y_n)\hat{Y} \right) . h(a_j^{(1)}) \right\}$$

$$= -\sum_{n=1}^{N} \left\{ \left(Y_n(1-\hat{Y}) - (1-Y_n)\hat{Y} \right) . \max(0, a_j^{(1)}) \right\}$$

Assignment 3 - Neural Networks

Shivakanth Thudi

2/15/2017

Part 2

1) Best Learning Rate

We look at the following learning rates: 1, 0.1, 0.01, 0.001, 0.0001, 0.00001.

Learning Rate	Validation Accuracy
1	0.0892
0.1	0.2134
0.01	0.9692
0.001	0.9798
0.0001	0.9555
0.00001	0.8966

We find that the best two learning rates among the list we considered are 0.01 and 0.001, which had validation accuracies of 0.9692 and 0.9798, respectively. We can interpolate between these values to find which learning rate is the best; however, we find that the validation accuracy drops off from using a learning rate of 0.001 onwards. So it is better to interpolate between 0.0001 and 0.001 instead:

Learning Rate	Validation Accuracy
0.0001	0.9557
0.0002	0.9692
0.0003	0.9751
0.0004	0.9765
0.0005	0.978
0.0006	0.9774
0.0007	**0.9802**
0.0008	0.98
0.0009	0.9787

We see that the best learning rate was 0.0007 which gave us a validation accuracy of 0.9802. We used 10 epochs for all learning rates.

2) Best Hidden Layer Size

We look at the following hidden layer sizes: 10, 50, 100, 300, 1000, 2000. We use a learning rate of 0.01 and train for 10 epochs. We shall report the validation accuracies.

Hidden Layer Size	Validation Accuracy
10	0.9139
50	0.9646
100	0.9712
300	**0.9794**
1000	0.9773
2000	0.9741

We find that the best hidden layer size was 300, which achieved a validation accuracy of 0.9794. Apart from the hidden layer size of 10, we see that all the hidden layer sizes achieve a validation accuracy higher than 0.96. However, the gains in validation accuracy are small as we increase the hidden layer sizes - after 300, the validation accuracy goes down.

When we look at both the training accuracies and the validation accuracies across epochs for all the hidden layer sizes, we see that they both increase, however, after around epoch 5 the validation accuracy either flatlines or improves marginally.

For the hidden layer sizes of 1000 and 2000, the validation accuracy decreases but is very close to that attained by the hidden layer size of 300. **We cannot explicitly state that overfitting has occurred because the validation accuracy does not decrease monotonically across epochs, but instead tends to fluctuate.**

3) Neural Network with L2 Weight Decay

Model	Learning Rate	Hidden Layer Size	L2 Weight Decay	Number of Epochs	Validation Accuracy
1	0.01	300	None	10	0.9693
2	**0.001**	**300**	**None**	**10**	**0.9801**
3	0.001	300	0.002	30	0.9761

We see that a smaller learning rate of 0.001 in model 2 improves the validation accuracy - this is the model that performs the best among the three.

Lowering the learning rate while keeping the hidden layer size and the number of epochs the same improved the model. This was without L2 regularization.

Using L2 regularization with weight = 0.002 and a higher number of epochs (30) did NOT improve the model when the learning rate was the same.

4) Best Dropout

We use a learning rate of 0.001, 30 epochs, and a hidden layer size of 300.

Dropout	Training Accuracy	Validation Accuracy
0.1	0.9986	0.9828
0.2	0.9976	0.9836
0.3	**0.9962**	**0.9841**
0.4	0.9939	0.9839
0.5	0.9892	0.9838
0.6	0.9841	0.9823
0.7	0.9752	0.9813
0.8	0.9552	0.9772
0.9	0.8998	0.9641

We observe that a dropout of 0.3 achieves the best performance here, since the model with this dropout value has the highest validation accuracy of 0.9841. Dropout prevents overfitting and all the dropout values ranging from 0.1 to 0.6 have very good performance on the training and testing set. Both the training and testing accuracies increase when using dropout.

Yes, the dropout helps to reduce testing error compared to models without a dropout. Also, dropout performs better than L2 regularization. The model with L2 regularization had a validation accuracy of 0.9761, while models with dropout ranging from 0.1 to 0.8 all have higher validation accuracies.

5) Three Layer Network

For our three layer network, we consider the following parameters:

- First Hidden Layer Size = [300, 500]
- Second Hidden Layer Size= [100, 150, 300]
- Learning Rate = [0.001, 0.005]
- Dropout on First Hidden Layer = [0.1, 0.2, 0.3]
- Dropout on Second Hidden Layer = [0.1, 0.2, 0.3]

We use the Relu activation on both hidden layers, with a softmax transformation in the end for converting to our 10 categorical responses.

We observe that the learning rate of 0.001 is better than the learning rate of 0.005 in all the cases.

We look at some of the top models below:

Using 10 Epochs with Relu Activations and Dropout in both hidden layers

Model	Learning Rate	First Hidden Layer Size	Second Hidden Layer Size	Dropout in First Hidden Layer	Dropout in Second Hidden Layer	Validation Accuracy
1	0.001	300	100	0.2	0.2	0.9832
2	0.001	300	150	0.1	0.2	0.9836
3	0.001	300	300	0.1	0.3	0.9832
4	0.001	500	100	0.3	0.1	0.9830
5	**0.001**	**500**	**100**	**0.3**	**0.3**	**0.9842**
6	0.001	500	150	0.2	0.3	0.9837
7	0.001	500	300	0.2	0.2	0.9838

Using 30 Epochs with Relu activations in both hidden layers

Model	Learning Rate	First Hidden Layer Size	Second Hidden Layer Size	Dropout in First Hidden Layer	Dropout in Second Hidden Layer	Validation Accuracy
1	0.001	500	150	0.2	0.2	0.9855
2	0.001	500	300	0.2	0.2	0.9822
3	0.001	500	300	0.2	No Dropout	0.984
4	**0.001**	**300**	**150**	**0.1**	**0.2**	**0.9856**

When we use 30 epochs rather than 10, we see improved validation accuracy. **The best 3-layer network we observed was the model with learning rate as 0.001, hidden layer sizes of 300 and 150 with dropout of 0.1 and 0.2 respectively.** The validation accuracy was 0.9856; this model used Relu activations in both layers. We also experimented with not using dropout in the second layer, but this did not improve validation accuracy over models which did use dropout in both layers.

```
# coding: utf-8

# # Neural Networks for MNIST dataset

# ## Loading data

# In[1]:

import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.utils import np_utils
from keras.datasets import mnist


# In[2]:

seed = 7
np.random.seed(seed)


# In[3]:

(X_train, Y_train), (X_test, Y_test) = mnist.load_data()


# In[4]:

X_train.shape


# In[5]:

get_ipython().magic(u'matplotlib inline')


# In[6]:

# flatten 28*28 images to a 784 vector for each image
num_pixels = X_train.shape[1] * X_train.shape[2]
X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32')
X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')


# In[7]:

# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_test = X_test / 255


# In[8]:
```

```
# one hot encode outputs
Y_train = np_utils.to_categorical(Y_train)
Y_test = np_utils.to_categorical(Y_test)
num_classes = Y_test.shape[1]


# ## Part 2.1 - Tuning Learning Rates

# In[9]:

from keras.models import Sequential
from keras.layers import Dense, Activation


# In[25]:

M = 300

learning_rates = [1, 0.1, 0.01, 0.001, 0.0001, 0.00001]

for index, learning_rate in enumerate(learning_rates):
    model1 = Sequential()
    model1.add(Dense(M, input_dim=num_pixels, init='normal', activation='relu'))
    model1.add(Dense(10, activation='softmax'))
    model1.compile(optimizer='adam',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])

    model1.optimizer.lr.set_value(learning_rate)

    model1.fit(X_train, Y_train, validation_data=(X_test, Y_test), nb_epoch=10,
batch_size=200, verbose=0)

    scores = model1.evaluate(X_test, Y_test, verbose=0)

    print "Learning Rate:", learning_rate, ", Validation Accuracy:", scores[1]


# ### 0.01 and 0.001 were the best learning rates. We will interpolate between
these values and find the best learning rate to use:

# In[29]:

new_learning_rates = np.arange(0.001, 0.01, 0.001).astype('float32')

for index, learning_rate in enumerate(new_learning_rates):
    model1 = Sequential()
    model1.add(Dense(M, input_dim=num_pixels, init='normal', activation='relu'))
    model1.add(Dense(10, activation='softmax'))
    model1.compile(optimizer='adam',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])
```

```
    model1.optimizer.lr.set_value(learning_rate)

    model1.fit(X_train, Y_train, validation_data=(X_test, Y_test), nb_epoch=10,
batch_size=200, verbose=0)

    scores = model1.evaluate(X_test, Y_test, verbose=0)

    print "Learning Rate:", learning_rate, ", Validation Accuracy:", scores[1]


# ### Since the validation accuracy drops off from learning rate = 0.001
onwards, we will interpolate between 0.0001 and 0.001 instead:

# In[30]:

new_learning_rates = np.arange(0.0001, 0.001, 0.0001).astype('float32')

for index, learning_rate in enumerate(new_learning_rates):
    model1 = Sequential()
    model1.add(Dense(M, input_dim=num_pixels, init='normal', activation='relu'))
    model1.add(Dense(10, activation='softmax'))
    model1.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])

    model1.optimizer.lr.set_value(learning_rate)

    model1.fit(X_train, Y_train, validation_data=(X_test, Y_test), nb_epoch=10,
batch_size=200, verbose=0)

    scores = model1.evaluate(X_test, Y_test, verbose=0)

    print "Learning Rate:", learning_rate, ", Validation Accuracy:", scores[1]


# ## Part 2.2 - Tuning Hidden Layer Size

# In[33]:

def get_model(lr=0.001, M=300):
    model = Sequential()
    model.add(Dense(M, input_dim=num_pixels, init='normal', activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])
    model.optimizer.lr.set_value(lr)
    return model


# In[34]:

hidden_sizes = [10, 50, 100, 300, 1000, 2000]
```

```
# In[37]:

for hidden_layer_size in hidden_sizes:

    model = get_model(lr=0.01, M = hidden_layer_size)
    model.fit(X_train, Y_train, validation_data=(X_test, Y_test), nb_epoch=10,
batch_size=200, verbose=2)

    scores = model.evaluate(X_test, Y_test, verbose=0)

    print "Hidden Layer Size:", hidden_layer_size, ", Validation Accuracy:",
scores[1]



# In[ ]:




# ## Part 2.3 - L2 Weight Decay

# In[38]:

from keras.regularizers import l2

def get_reg_model(lr=0.001, M=300, w=0.1):
    model = Sequential()
    model.add(Dense(M, input_dim=num_pixels, init='normal', activation='relu',
W_regularizer=l2(w)))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    model.optimizer.lr.set_value(lr)
    return model


# In[39]:

models = [get_model(lr = 0.01, M = 300), get_model(lr = 0.001, M = 300),
get_reg_model(lr=0.001, M = 300, w = 0.002)]


# ## Model 1 - Learning Rate = 0.01, M = 300

# In[41]:

model = models[0]
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), nb_epoch=10,
batch_size=200, verbose=2)

scores = model.evaluate(X_test, Y_test, verbose=0)
```

```
print  "Validation Accuracy:", scores[1]


# ## Model 2 - Learning Rate = 0.001, M = 300

# In[42]:

model = models[1]
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), nb_epoch=10,
batch_size=200, verbose=2)

scores = model.evaluate(X_test, Y_test, verbose=0)

print  "Validation Accuracy:", scores[1]


# ## Model 3 - Learning Rate = 0.001, M = 300, w = 0.002, and epochs = 30
#
#

# In[44]:

model = models[2]
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), nb_epoch=30,
batch_size=200, verbose=2)

scores = model.evaluate(X_test, Y_test, verbose=0)

print  "Validation Accuracy:", scores[1]


# In[ ]:




# In[ ]:




# ## Part 2.4 - Models with Dropout

# In[63]:

def get_dropout_model(lr=0.001, M=300, w=0.2):
    model = Sequential()
    model.add(Dense(M, input_dim=num_pixels, init='normal', activation='relu'))
    model.add(Dropout(w))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

```
    model.optimizer.lr.set_value(lr)
    return model


# In[69]:

dropouts = [ i/10.0 for i in range(1,11,1)]


# In[70]:

dropouts


# In[71]:

for dropout in dropouts:
    model = get_dropout_model(lr = 0.001, M = 300, w = dropout)

    model.fit(X_train, Y_train, validation_data=(X_test, Y_test), nb_epoch=30,
batch_size=200, verbose=2)
    scores = model.evaluate(X_test, Y_test, verbose=0)

    print "Dropout:", dropout, ", Validation Accuracy:", scores[1]



# In[ ]:




# ## Part 2.5 - 3-layer Network

# In[ ]:




# In[17]:

hl_1 = [300, 500]
hl_2 = [150, 300]
lrs = [0.001]
dropout_1 = [0.0, 0.1, 0.2]
dropout_2 = [0.0, 0.1, 0.2]


# In[ ]:




# In[18]:
```

```
def get_three_layer_model(lr=0.001, M=500, N =150, w1=0.2, w2=0.2):
    model = Sequential()
    model.add(Dense(M, input_dim=num_pixels, init='normal', activation='relu'))
    model.add(Dropout(w1))

    model.add(Dense(N, input_dim=M, init='normal', activation='relu'))
    model.add(Dropout(w2))

    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])
    model.optimizer.lr.set_value(lr)
    return model


# In[20]:

for M in hl_1:
    for N in hl_2:
        for lr in lrs:
            for w1 in dropout_1:
                for w2 in dropout_2:

                    model = get_three_layer_model(lr=lr, M=M, N =N, w1=w1,
w2=w2)

                    model.fit(X_train, Y_train, validation_data=(X_test,
Y_test), nb_epoch=30, batch_size=200, verbose=0)
                    scores = model.evaluate(X_test, Y_test, verbose=0)

                    print "Hidden Layer 1:", M, ", Hidden Layer 2:", N, ",
Learning Rate:", lr, ", Dropout 1:", w1, ", Dropout 2:", w2, ", Validation
Accuracy:", scores[1]






# In[ ]:
```