ML Assignment 1

Abhinav Moudgil
201331039

Q1: ***Can you make the confusion matrix for the above network.***

```
from sklearn.metrics import confusion_matrix y_pred = model.predict(X_test)
        y_pred = np.argmax(y_pred,axis=1)
         y_t = np.argmax(Y_test,axis=1)
print(confusion_matrix(y_t, y_pred))
```

```
[[ 973    2    0    0    1    0    2    1    1    0]
 [   0 1125    1    2    0    0    2    0    5    0]
 [   0    0 1019    1    1    0    2    5    3    1]
 [   0    0    4  987    0    5    0    5    5    4]
 [   1    2    4    0  966    0    3    1    1    4]
 [   2    0    0    6    0  872    5    0    6    1]
 [   2    3    0    1    2    5  943    0    2    0]
 [   2    5    7    1    1    0    0 1002    5    5]
 [   3    3    2    2    0    1    1    4  955    3]
 [   3    3    0    0   16    3    1    1    1  981]]
```

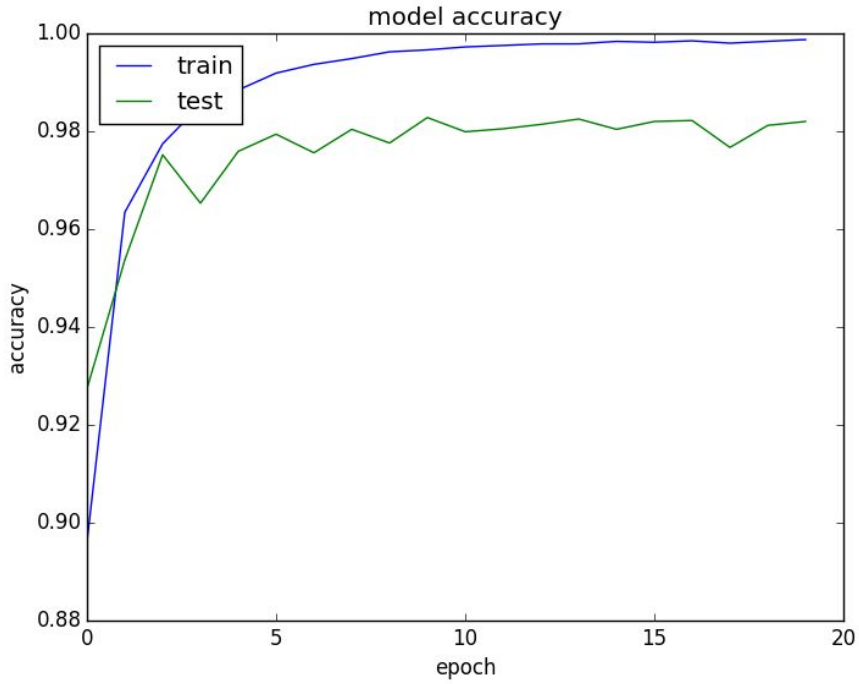
Q2: Try out the following variations and see how it affects loss and accuracy. Present your observations with plots and suitable explanation: - Batch size - Size of hidden layer - choice of activation function ('relU','sigmoid','tanh')

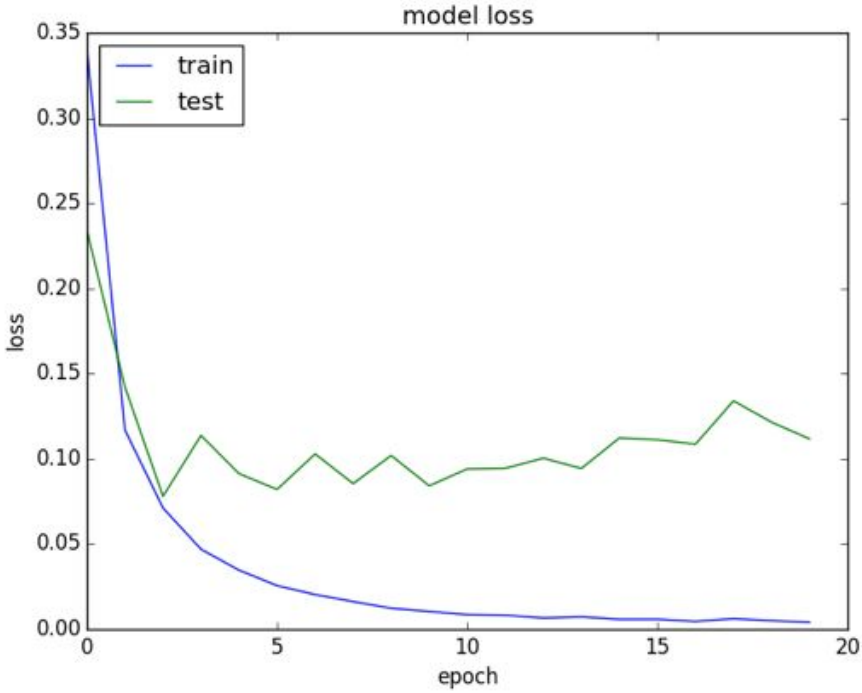
```
import numpy as np
import os
os.environ["THEANO_FLAGS"] = "mode=FAST_RUN,device=cpu,floatX=float32"
np.random.seed(1337) # for reproducibility
#Load the dataset
from keras.datasets import mnist
# Import the sequential module from keras
from keras.models import Sequential
# Import the layers you wish to use in your net
from keras.layers.core import Dense, Dropout, Activation
# Import the optimization algorithms that you wish to use
from keras.optimizers import SGD, Adam, RMSprop
# Import other utilities that help in data formatting etc. from keras.utils import np_utils

batch_size = 512
nb_classes = 10
```

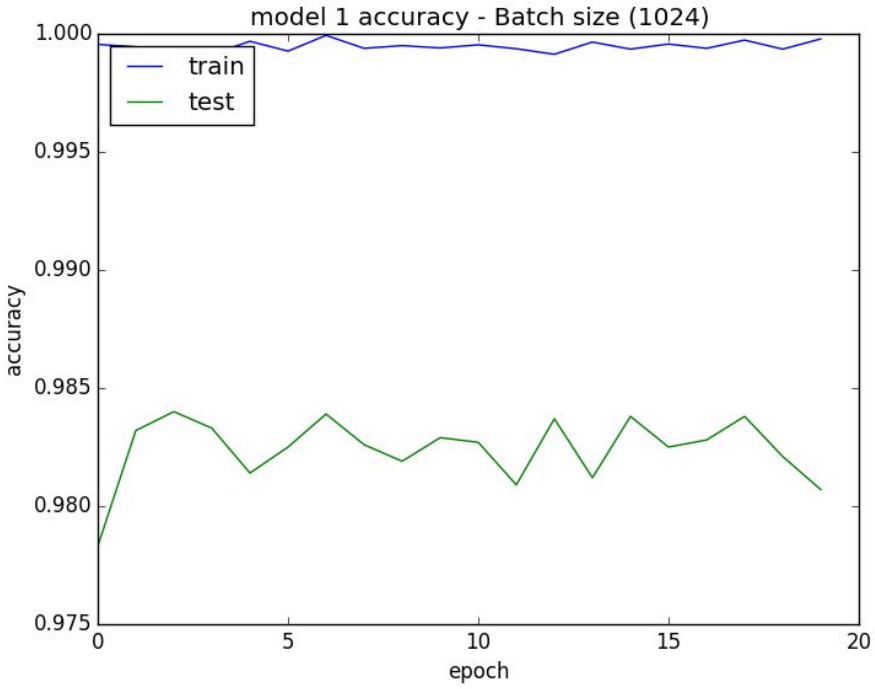
```
nb_epoch = 20

# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Preprocess the data
# We need to shape the data into a shape that network accepts.
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)

# Here we convert the data type of data to 'float32'
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# We normalize the data to lie in the range 0 to 1.
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)
# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)

model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dense(512))
model.add(Dense(10))
model.add(Activation('softmax'))
model1 = Sequential() #change batch size
model1.add(Dense(512, input_shape=(784,)))
model1.add(Activation('relu'))
model1.add(Dense(512))
model1.add(Activation('relu'))
model1.add(Dense(10))
model1.add(Activation('softmax'))
model2 = Sequential() #change hidden layers
model2.add(Dense(1024, input_shape=(784,)))
model2.add(Activation('relu'))
model2.add(Dense(1024))
model2.add(Activation('relu'))
model2.add(Dense(10))
model2.add(Activation('softmax'))
model3 = Sequential() #change activation function
model3.add(Dense(512, input_shape=(784,)))
model3.add(Activation('sigmoid'))
```

```
model3.add(Dense(512))
model3.add(Activation('sigmoid'))
model3.add(Dense(10))
model3.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',optimizer=RMSprop(),metrics=['accuracy'])
model1.compile(loss='categorical_crossentropy',optimizer=RMSprop(),metrics=['accuracy'])
model2.compile(loss='categorical_crossentropy',optimizer=RMSprop(),metrics=['accuracy'])
model3.compile(loss='categorical_crossentropy',optimizer=RMSprop(),metrics=['accuracy'])

history = model.fit(X_train, Y_train,batch_size=batch_size, nb_epoch=nb_epoch, verbose=1,
validation_data=(X_test, Y_test))
history1 = model.fit(X_train, Y_train,batch_size=1024, nb_epoch=nb_epoch, verbose=1,
validation_data=(X_test, Y_test))
history2 = model.fit(X_train, Y_train,batch_size=batch_size, nb_epoch=nb_epoch, verbose=1,
validation_data=(X_test, Y_test))
history3 = model.fit(X_train, Y_train,batch_size=batch_size, nb_epoch=nnb_epoch, verbose=1,
validation_data=(X_test, Y_test))

import matplotlib.pyplot as plt
        # list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(history1.history['acc']) plt.plot(history1.history['val_acc'])
plt.title('model 1 accuracy - Batch size (x2)')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(history2.history['acc']) plt.plot(history2.history['val_acc'])
plt.title('model 2 accuracy - Hidden layers (x2)')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(history3.history['acc']) plt.plot(history3.history['val_acc'])
plt.title('model 3 accuracy - Activation function (sigmoid)')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
```

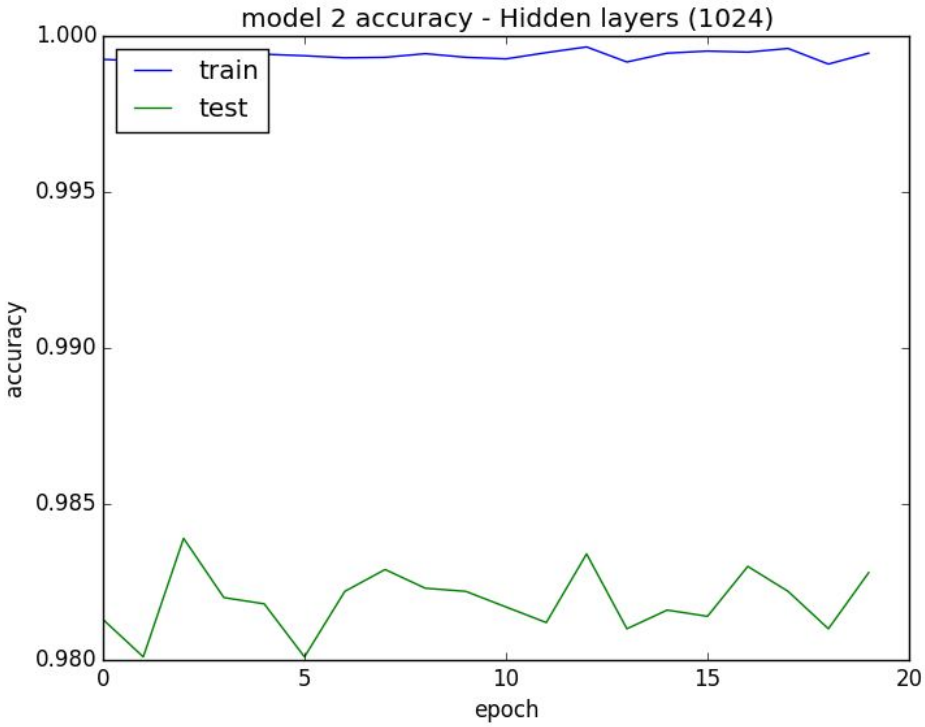
```
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(history1.history['loss']) plt.plot(history1.history['val_loss'])
plt.title('model 1 loss - Batch size (x2)')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(history2.history['loss']) plt.plot(history2.history['val_loss'])
plt.title('model 2 loss - Hidden layers (x2)')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.plot(history3.history['loss']) plt.plot(history3.history['val_loss'])
plt.title('model 3 loss - Activation function (sigmoid)')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

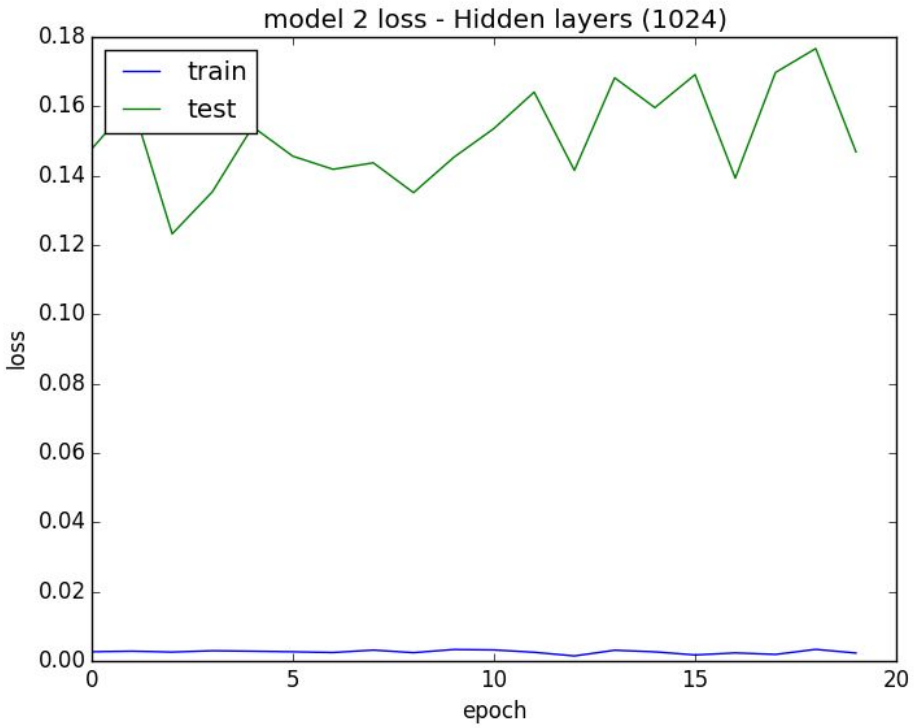

model accuracy
train
test
accuracy
epoch
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0
5
10
15
20

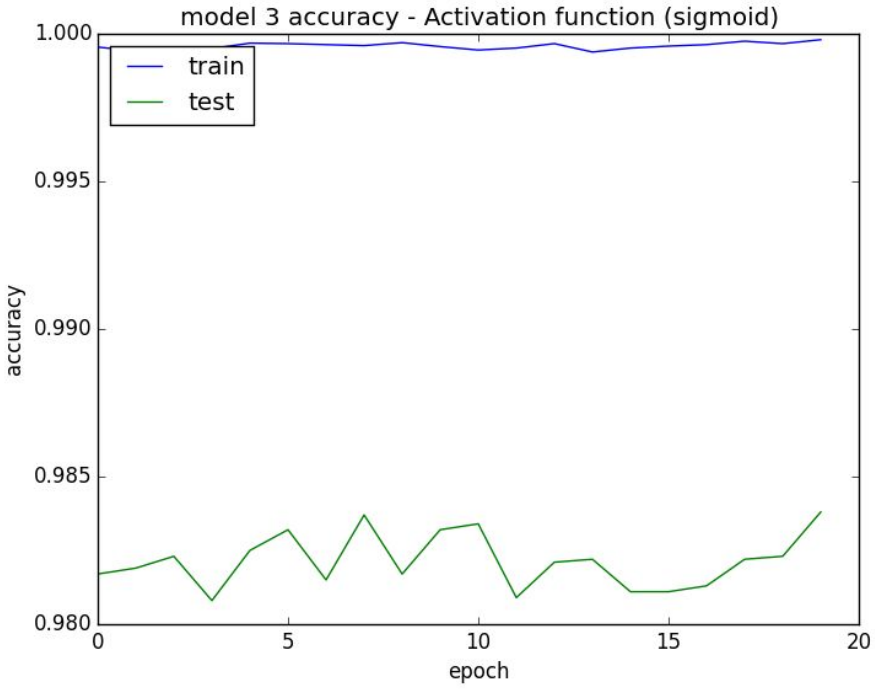

model loss
train
test
loss
epoch
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20

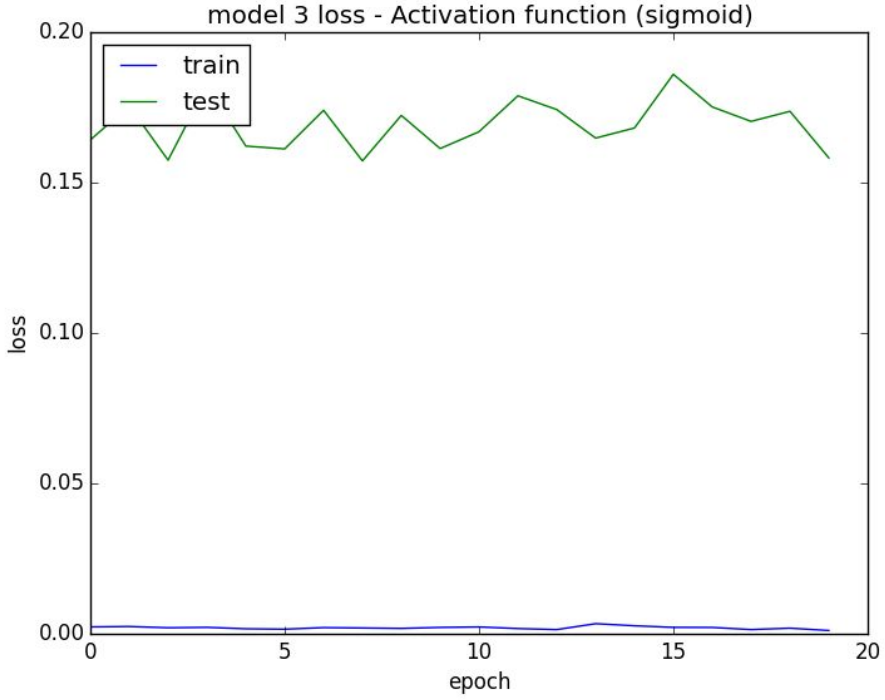

model 1 accuracy - Batch size (1024)
train
test
accuracy
epoch
1.000
0.995
0.990
0.985
0.980
0.975
0
5
10
15
20


model 1 loss - Batch size (1024)
train
test
loss
epoch
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
5
10
15
20


model 2 accuracy - Hidden layers (1024)
train
test
accuracy
epoch
1.000
0.995
0.990
0.985
0.980
0
5
10
15
20


model 2 loss - Hidden layers (1024)
train
test
loss
epoch
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
5
10
15
20


model 3 accuracy - Activation function (sigmoid)
train
test
accuracy
epoch
1.000
0.995
0.990
0.985
0.980
0
5
10
15
20

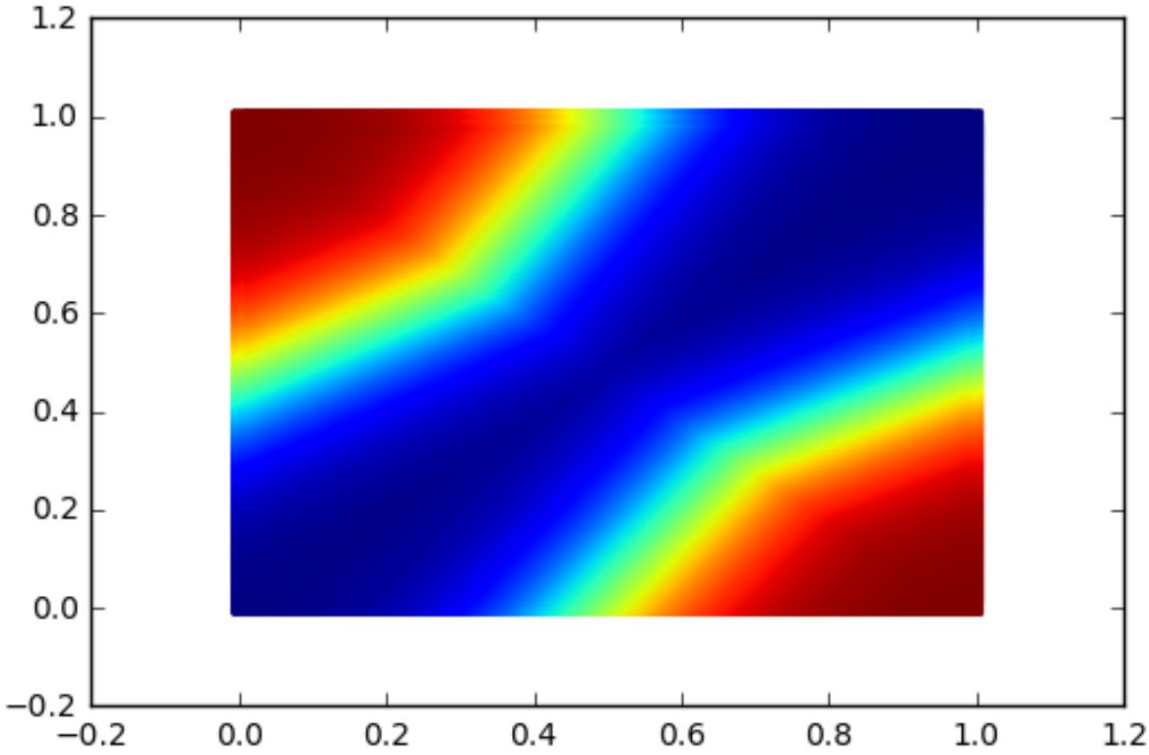

model 3 loss - Activation function (sigmoid)
train
test
loss
epoch
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20

Q3: Try classification on XOR data using similar MLP.
The problem can be stated as follows. Build a neural network that will produce the following truth table, called the 'exclusive or' or 'XOR' (either A or B but not both):
Here (X,Y) will be the input and the label would be X xor Y.
Plot the decision boundary

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation from keras.optimizers import SGD
import numpy as np
         np.random.seed(100)
         model4 = Sequential()
         model4.add(Dense(8,input_shape=(2,)))
         model4.add(Activation('relu'))
         model4.add(Dense(1))
         model4.add(Activation('sigmoid'))

         X = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
         y = np.array([[0],[1],[1],[0]], "float32")
         sgd = SGD(lr=0.1)
         model4.compile(loss='binary_crossentropy', optimizer=sgd)
         model4.fit(X, y, nb_epoch=500, batch_size=1)
print(model4.predict([[0.5,0.3],[0.0,0.8],[0.7,0.7],[0.7,0.2]]))
```

```
[[ 0.14591138]
 [ 0.97089398]
 [ 0.01678502]
 [ 0.74476397]]
```

```
h=0.001;
x_min=0;
x_max=1;
y_min=0;
y_max=1;

xx, yy = np.meshgrid(np.arange(x_min, x_max, h),np.arange(y_min, y_max, h)
Z = model4.predict(np.c_[xx.ravel(), yy.ravel()])
from matplotlib import pyplot as plt
from matplotlib import cm

plt.scatter(xx,yy,s=20,c=Z, marker = 'x', cmap = cm.jet )
plt.show()
```



Q4: What are the number of parameters in convolution layers with K filters each of size 3*w*h.

Number of parameters = (3 * w * h + 1) * K

Q5: What are the number of parameters in a max pooling operation?

There are 0 parameters in max pooling operation because it is just a function which is applied on given activation map.

Q6: Which of the operations contain most number of parameters?
(a) conv (b) pool (c) Fully connected layer (FC) (d) Relu

Fully connected layer always contain maximum number of parameters because

- in convolution layer, same weights are shared across the whole activation map. Filters size are generally small to capture more local information and thus, we get number of parameters = filter size * depth * number of filters
- in pool layer, number of parameters is 0 because we just do a simple operation of max, average or min.
- in relu layer, number of parameters is 0 because we apply a simple function and no memory is consumed
- in fully connected layer, number of parameters = number of hidden units * number of output units. In case of images, number of hidden is very large (for example, 4k in alex net) and thus, number of parameters shoot up easily

Q7: Which operation consume most amount of memory?
(a) initial convolution layers (b) fully connected layers at the end

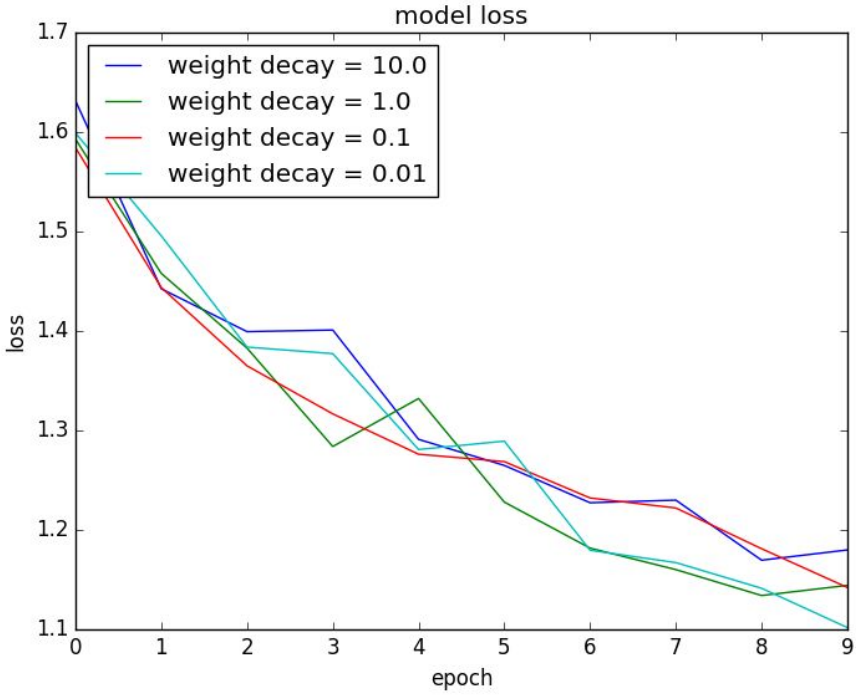
Most of the memory is consumed in initial convolution layers because we generally convolve with small filter sizes so that local information is preserved but due to more number of filters in early convolution layers, we get activation maps of generally (size of image x number of filters) which consume most of the memory whereas in fully connected layers, we've only a few thousand hidden units, which is much small number as compared to initial layer activation map size.

Q8: Experiment with learning rate and notice the behaviour of the learning process. Plot your observations in a graph with brief explanation.

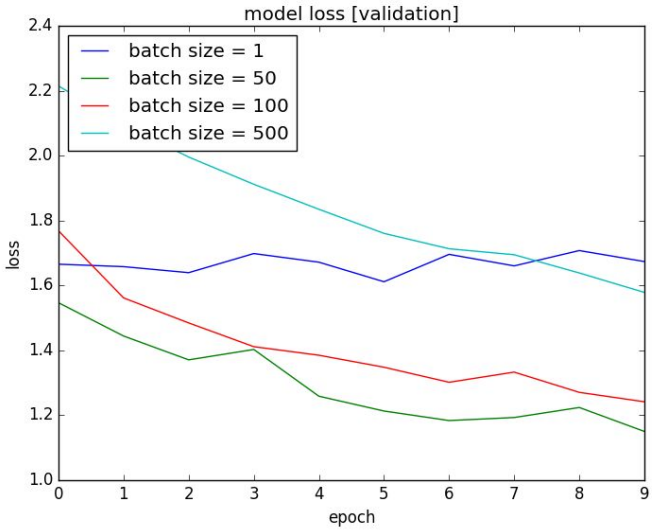
Initial learning rate is kept 0.001 but weight decay factor is changed. In Keras SGD, learning rate is changed by the rule

$$lr(t) = lr(0)/(1 + weight\_decay * t)$$

Too less may settle to local optima whereas large learning rate may lead to gradient to wrong direction.



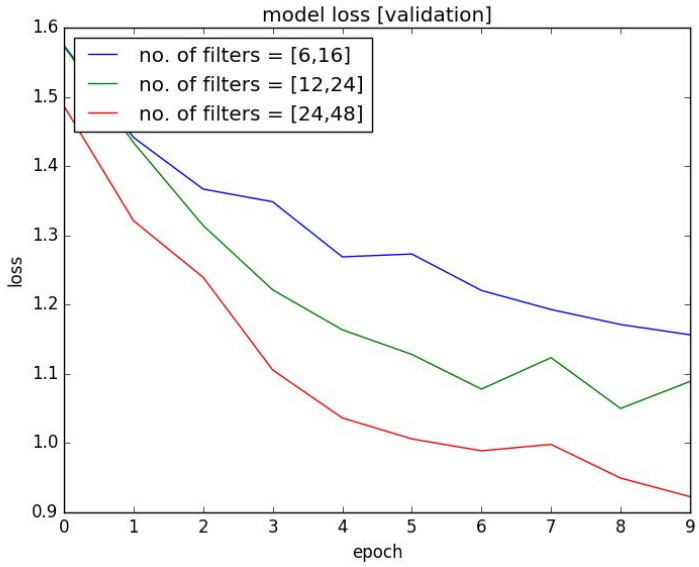
Q9: Currently, the batch-size is 50. Notice the training loss curve if batch size is changed to 1. Is it smooth or fluctuating? Show the effect of batch-size on the learning curves in a plot.



Setting batch_size = 1 may not be fruitful as gradient of a single point of may be positive too depending on the point. So we may not moving in the direction of gradient. Similar is the case when batch size is increased a lot, it leads to more loss and less accuracy.

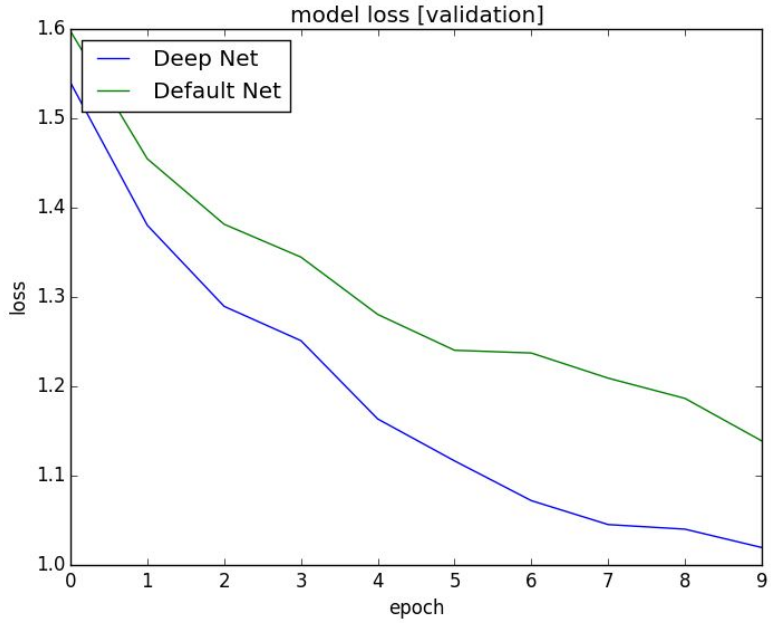
Q10: Increase the number of convolution filters and experiment. Present your observations using plots and brief explanations.

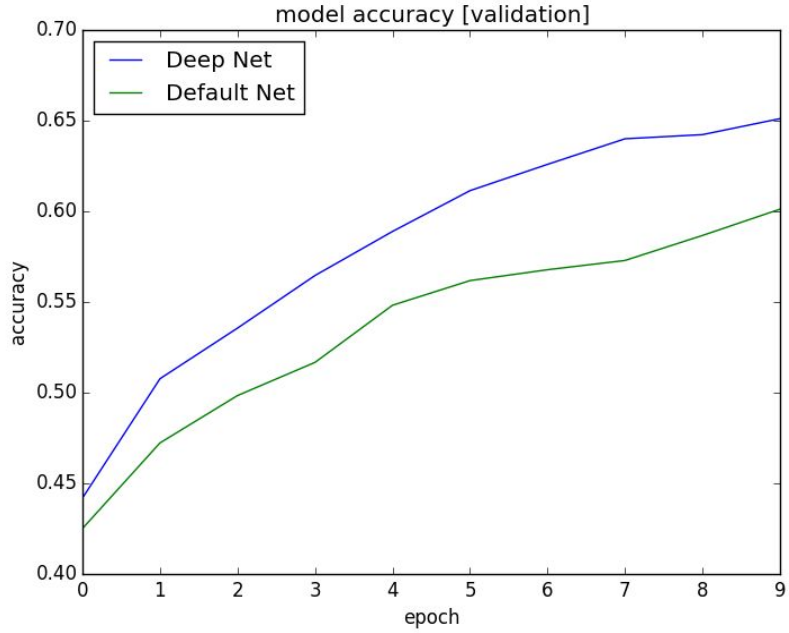
Increasing the number of filters causes means we're essential increasing the activation units which leads to better learning of embedding space and hence there is increasing in accuracy.



Q11: What do you observe if you increase the number of layers (depth of the network) ? Present your observations using plots and brief explanations.

Increasing number of layers causes CNN to learn more high level features and hence there is increasing in accuracy. It is demonstrated in the plots below:



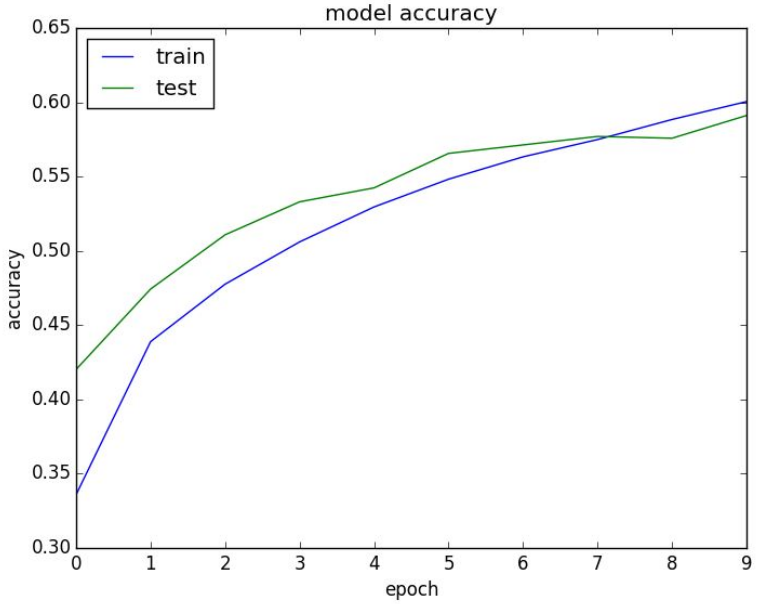


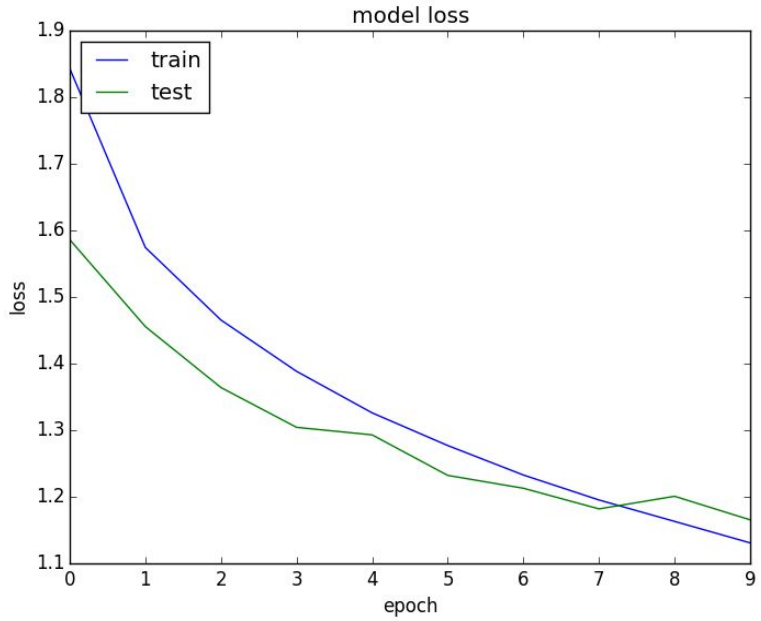
Q12: CNN training requires lot of training data. In the absence of large training data, a common practice is to use synthetic data using operations such as flipping, scaling, etc. Can you think of any other two operations techniques that can help to increase the training set? Plot the graphs which demonstrate these effects with sufficient explanation.

Two additional operations are done:

1. Rotation by 90 degrees
2. Addition of salt and pepper noise

10,000 random samples are rotated and 10,000 random samples are bursted with salt and pepper noise. Thus we have total 70,000 training samples and 10,000 test samples. With the addition of this synthetic data, CNN is learning more robust embedding space and there is increase in accuracy because we're essentially using more data.
Below are the plots using same default network but with using synthetic data:





Appendix

This section contains all the source code files which produced above results.

q8.py

```
import numpy as np
import os
os.environ["THEANO_FLAGS"] = "mode=FAST_RUN,device=cpu,floatX=float32"
import theano
import keras

from keras.datasets import cifar10
from keras.models  import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.optimizers import SGD
from keras.utils import np_utils
import matplotlib.pyplot as plt
#%matplotlib inline

# load cifar dataset
cifar10 = np.load('../../../data/lab2/lab2_data/cifar10_data.npz')
X_train = cifar10['X_train']
y_train = cifar10['y_train']
X_test = cifar10['X_test']
y_test = cifar10['y_test']

print "Training data:"
print "Number of examples: ", X_train.shape[0]
print "Number of channels:",X_train.shape[1]
print "Image size:", X_train.shape[2], X_train.shape[3]
print
print "Test data:"
print "Number of examples:", X_test.shape[0]
print "Number of channels:", X_test.shape[1]
print "Image size:",X_test.shape[2], X_test.shape[3]


# normalise data
print "mean before normalization:", np.mean(X_train)
print "std before normalization:", np.std(X_train)

mean=[0,0,0]
std=[0,0,0]
newX_train = np.ones(X_train.shape)
newX_test = np.ones(X_test.shape)
for i in xrange(3):
```

```
    mean[i] = np.mean(X_train[:,i,:,:])
    std[i] = np.std(X_train[:,i,:,:])

for i in xrange(3):
    newX_train[:,i,:,:] = X_train[:,i,:,:] - mean[i]
    newX_train[:,i,:,:] = newX_train[:,i,:,:] / std[i]
    newX_test[:,i,:,:] = X_test[:,i,:,:] - mean[i]
    newX_test[:,i,:,:] = newX_test[:,i,:,:] / std[i]


X_train = newX_train
X_test = newX_test

print "mean after normalization:", np.mean(X_train)
print "std after normalization:", np.std(X_train)

learningRate= 0.001               #-- Learning rate for the network
lr_weight_decay = np.array([10, 1, 0.1, 0.01])            #-- Learning weight decay. Reduce
the learn rate by 0.95 after epoch

print "learning rate array size:", lr_weight_decay.size

class SGDLearningRateTracker(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        optimizer = self.model.optimizer
        lr = optimizer.lr * (1. / (1. + optimizer.decay * optimizer.iterations))
        print('\nLR: {:.6f}\n'.format(lr))

for i in xrange(lr_weight_decay.size):
    img_rows, img_cols = 32, 32       #-- input image dimensions
    num_classes = 10                  #-- Number of classes in CIFAR-10 dataset

    model = Sequential()                                                  #-- Sequential
container.
    model.add(Convolution2D(6, 5, 5,                                      #-- 6 outputs (6
filters), 5x5 convolution kernel
                        border_mode='valid',
                        input_shape=(3, img_rows, img_cols),dim_ordering="th"))       #-- 3
input depth (RGB)
    model.add(Activation('relu'))                                         #-- ReLU non-linearity
    model.add(MaxPooling2D(pool_size=(2, 2),  dim_ordering="th"))
#-- A max-pooling on 2x2 windows
    model.add(Convolution2D(16, 5, 5, dim_ordering="th"))                                  #--
16 outputs (16 filters), 5x5 convolution kernel
    model.add(Activation('relu'))                                         #-- ReLU non-linearity
    model.add(MaxPooling2D(pool_size=(2, 2),  dim_ordering="th"))
#-- A max-pooling on 2x2 windows

    model.add(Flatten())                                                  #-- eshapes a 3D
tensor of 16x5x5 into 1D tensor of 16*5*5
```

```
    model.add(Dense(120))                                                #-- 120 outputs fully
connected layer
    model.add(Activation('relu'))                                        #-- ReLU non-linearity
    model.add(Dense(84))                                                 #-- 84 outputs fully
connected layer
    model.add(Activation('relu'))                                        #-- ReLU non-linearity
    model.add(Dense(num_classes))                                        #-- 10 outputs fully
connected layer (one for each class)
    model.add(Activation('softmax'))                                     #-- converts the
output to a log-probability. Useful for classification problems
    print model.summary()

# specify training parameters
    batchSize = 50                    #-- Training Batch Size
    num_epochs = 10                   #-- Number of epochs for training

    Y_train = np_utils.to_categorical(y_train, num_classes)
    Y_test = np_utils.to_categorical(y_test, num_classes)
    sgd = SGD(lr=learningRate, decay = lr_weight_decay[i])
    model.compile(loss='categorical_crossentropy',
                          optimizer='sgd',
                                       metrics=['accuracy'])

#    print(model.lr.get_value())
    #-- switch verbose=0 if you get error "I/O operation from closed file"
    history = model.fit(X_train, Y_train, batch_size=batchSize, nb_epoch=num_epochs,
                        verbose=1, shuffle=True, validation_data=(X_test, Y_test),
callbacks=[SGDLearningRateTracker()])
#    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'], label='weight decay = ' + str(lr_weight_decay[i]))
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(loc='upper left')
    plt.hold(True)

plt.legend(loc='upper left')
plt.show()
```

q9.py

```
import numpy as np
import os
os.environ["THEANO_FLAGS"] = "mode=FAST_RUN,device=cpu,floatX=float32"
import theano
import keras

from keras.datasets import cifar10
from keras.models  import Sequential
```

```
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.optimizers import SGD
from keras.utils import np_utils
import matplotlib.pyplot as plt
#%matplotlib inline

# load cifar dataset
cifar10 = np.load('../../../data/lab2/lab2_data/cifar10_data.npz')
X_train = cifar10['X_train']
y_train = cifar10['y_train']
X_test = cifar10['X_test']
y_test = cifar10['y_test']

print "Training data:"
print "Number of examples: ", X_train.shape[0]
print "Number of channels:",X_train.shape[1]
print "Image size:", X_train.shape[2], X_train.shape[3]
print
print "Test data:"
print "Number of examples:", X_test.shape[0]
print "Number of channels:", X_test.shape[1]
print "Image size:",X_test.shape[2], X_test.shape[3]

# visualise images
plot = []
for i in range(1,10):
    plot_image = X_train[100*i,:,:,:].transpose(1,2,0)
    for j in range(1,10):
        plot_image = np.concatenate((plot_image, X_train[100*i+j,:,:,:].transpose(1,2,0)),
axis=1)
    if i==1:
        plot = plot_image
    else:
        plot = np.append(plot, plot_image, axis=0)

plt.imshow(plot)
plt.axis('off')
plt.show()

# normalise data
print "mean before normalization:", np.mean(X_train)
print "std before normalization:", np.std(X_train)

mean=[0,0,0]
std=[0,0,0]
newX_train = np.ones(X_train.shape)
newX_test = np.ones(X_test.shape)
for i in xrange(3):
    mean[i] = np.mean(X_train[:,i,:,:])
```

```
    std[i] = np.std(X_train[:,i,:,:])

for i in xrange(3):
    newX_train[:,i,:,:] = X_train[:,i,:,:] - mean[i]
    newX_train[:,i,:,:] = newX_train[:,i,:,:] / std[i]
    newX_test[:,i,:,:] = X_test[:,i,:,:] - mean[i]
    newX_test[:,i,:,:] = newX_test[:,i,:,:] / std[i]


X_train = newX_train
X_test = newX_test

print "mean after normalization:", np.mean(X_train)
print "std after normalization:", np.std(X_train)

learningRate= 0.001               #-- Learning rate for the network
lr_weight_decay = 0.95
batchSize = np.array([1, 50, 100, 500])            #-- Learning weight decay. Reduce the learn
rate by 0.95 after epoch

print "batchSize array size: ", batchSize.size

for i in xrange(batchSize.size):
    img_rows, img_cols = 32, 32       #-- input image dimensions
    num_classes = 10                  #-- Number of classes in CIFAR-10 dataset

    model = Sequential()                                              #-- Sequential
container.
    model.add(Convolution2D(6, 5, 5,                                  #-- 6 outputs (6
filters), 5x5 convolution kernel
                        border_mode='valid',
                        input_shape=(3, img_rows, img_cols),dim_ordering="th"))       #-- 3
input depth (RGB)
    model.add(Activation('relu'))                                     #-- ReLU non-linearity
    model.add(MaxPooling2D(pool_size=(2, 2),  dim_ordering="th"))
#-- A max-pooling on 2x2 windows
    model.add(Convolution2D(16, 5, 5, dim_ordering="th"))                                  #--
16 outputs (16 filters), 5x5 convolution kernel
    model.add(Activation('relu'))                                     #-- ReLU non-linearity
    model.add(MaxPooling2D(pool_size=(2, 2),  dim_ordering="th"))
#-- A max-pooling on 2x2 windows

    model.add(Flatten())                                              #-- eshapes a 3D
tensor of 16x5x5 into 1D tensor of 16*5*5
    model.add(Dense(120))                                             #-- 120 outputs fully
connected layer
    model.add(Activation('relu'))                                     #-- ReLU non-linearity
    model.add(Dense(84))                                              #-- 84 outputs fully
connected layer
    model.add(Activation('relu'))                                     #-- ReLU non-linearity
```

```
    model.add(Dense(num_classes))                                    #-- 10 outputs fully
connected layer (one for each class)
    model.add(Activation('softmax'))                                 #-- converts the
output to a log-probability. Useful for classification problems
    print model.summary()

# specify training parameters
 #   batchSize = 50                    #-- Training Batch Size
    num_epochs = 10                   #-- Number of epochs for training

    Y_train = np_utils.to_categorical(y_train, num_classes)
    Y_test = np_utils.to_categorical(y_test, num_classes)
    sgd = SGD(lr=learningRate, decay = lr_weight_decay)
    model.compile(loss='categorical_crossentropy',
                          optimizer='sgd',
                                       metrics=['accuracy'])

    #-- switch verbose=0 if you get error "I/O operation from closed file"
    history = model.fit(X_train, Y_train, batch_size=batchSize[i], nb_epoch=num_epochs,
                        verbose=1, shuffle=True, validation_data=(X_test, Y_test))
#    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'], label='batch size = ' + str(batchSize[i]))
    plt.title('model loss [validation]')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(loc='upper left')
    plt.hold(True)

plt.legend(loc='upper left')
plt.show()
```

q10.py

```
import numpy as np
import os
os.environ["THEANO_FLAGS"] = "mode=FAST_RUN,device=cpu,floatX=float32"
import theano
import keras

from keras.datasets import cifar10
from keras.models  import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.optimizers import SGD
from keras.utils import np_utils
import matplotlib.pyplot as plt
#%matplotlib inline

# load cifar dataset
```

```
cifar10 = np.load('../../../data/lab2/lab2_data/cifar10_data.npz')
X_train = cifar10['X_train']
y_train = cifar10['y_train']
X_test = cifar10['X_test']
y_test = cifar10['y_test']

print "Training data:"
print "Number of examples: ", X_train.shape[0]
print "Number of channels:",X_train.shape[1]
print "Image size:", X_train.shape[2], X_train.shape[3]
print
print "Test data:"
print "Number of examples:", X_test.shape[0]
print "Number of channels:", X_test.shape[1]
print "Image size:",X_test.shape[2], X_test.shape[3]

# visualise images
plot = []
for i in range(1,10):
    plot_image = X_train[100*i,:,:,:].transpose(1,2,0)
    for j in range(1,10):
        plot_image = np.concatenate((plot_image, X_train[100*i+j,:,:,:].transpose(1,2,0)),
axis=1)
    if i==1:
        plot = plot_image
    else:
        plot = np.append(plot, plot_image, axis=0)

plt.imshow(plot)
plt.axis('off')
plt.show()

# normalise data
print "mean before normalization:", np.mean(X_train)
print "std before normalization:", np.std(X_train)

mean=[0,0,0]
std=[0,0,0]
newX_train = np.ones(X_train.shape)
newX_test = np.ones(X_test.shape)
for i in xrange(3):
    mean[i] = np.mean(X_train[:,i,:,:])
    std[i] = np.std(X_train[:,i,:,:])

for i in xrange(3):
    newX_train[:,i,:,:] = X_train[:,i,:,:] - mean[i]
    newX_train[:,i,:,:] = newX_train[:,i,:,:] / std[i]
    newX_test[:,i,:,:] = X_test[:,i,:,:] - mean[i]
    newX_test[:,i,:,:] = newX_test[:,i,:,:] / std[i]
```

```
X_train = newX_train
X_test = newX_test

print "mean after normalization:", np.mean(X_train)
print "std after normalization:", np.std(X_train)

learningRate= 0.001               #-- Learning rate for the network
lr_weight_decay = 0.95
batchSize = 50            #-- Learning weight decay. Reduce the learn rate by 0.95 after epoch

img_rows, img_cols = 32, 32       #-- input image dimensions
num_classes = 10                  #-- Number of classes in CIFAR-10 dataset

conv1_size = 6
conv2_size = 16

for i in xrange(3):

    model = Sequential()                                                #-- Sequential
container.
    model.add(Convolution2D(conv1_size, 5, 5,                                   #-- 6 outputs
(6 filters), 5x5 convolution kernel
                    border_mode='valid',
                    input_shape=(3, img_rows, img_cols),dim_ordering="th"))       #-- 3 input
depth (RGB)
    model.add(Activation('relu'))                                       #-- ReLU non-linearity
    model.add(MaxPooling2D(pool_size=(2, 2),  dim_ordering="th"))
#-- A max-pooling on 2x2 windows
    model.add(Convolution2D(conv2_size, 5, 5, dim_ordering="th"))
#-- 16 outputs (16 filters), 5x5 convolution kernel
    model.add(Activation('relu'))                                       #-- ReLU non-linearity
    model.add(MaxPooling2D(pool_size=(2, 2),  dim_ordering="th"))
#-- A max-pooling on 2x2 windows

    model.add(Flatten())                                                #-- eshapes a 3D
tensor of 16x5x5 into 1D tensor of 16*5*5
    model.add(Dense(120))                                               #-- 120 outputs fully
connected layer
    model.add(Activation('relu'))                                       #-- ReLU non-linearity
    model.add(Dense(84))                                                #-- 84 outputs fully
connected layer
    model.add(Activation('relu'))                                       #-- ReLU non-linearity
    model.add(Dense(num_classes))                                       #-- 10 outputs fully
connected layer (one for each class)
    model.add(Activation('softmax'))                                    #-- converts the
output to a log-probability. Useful for classification problems
    print model.summary()

    # specify training parameters
```

```
     #   batchSize = 50                    #-- Training Batch Size
    num_epochs = 10                   #-- Number of epochs for training

    Y_train = np_utils.to_categorical(y_train, num_classes)
    Y_test = np_utils.to_categorical(y_test, num_classes)
    sgd = SGD(lr=learningRate, decay = lr_weight_decay)
    model.compile(loss='categorical_crossentropy',
                        optimizer='sgd',
                                    metrics=['accuracy'])

    #-- switch verbose=0 if you get error "I/O operation from closed file"
    history = model.fit(X_train, Y_train, batch_size=batchSize, nb_epoch=num_epochs,
                      verbose=1, shuffle=True, validation_data=(X_test, Y_test))
    #plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'], label = 'no. of filters = [' + str(conv1_size) + ','
+ str(conv2_size) + ']')
    plt.title('model loss [validation]')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(loc='upper left')
    conv1_size = 2 * conv1_size
    conv2_size = 2 * conv1_size

plt.show()
```

q11.py

```
import numpy as np
import os
os.environ["THEANO_FLAGS"] = "mode=FAST_RUN,device=cpu,floatX=float32"
import theano
import keras

from keras.datasets import cifar10
from keras.models  import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.optimizers import SGD
from keras.utils import np_utils
import matplotlib.pyplot as plt

# process data
cifar10 = np.load('../../../data/lab2/lab2_data/cifar10_data.npz')
X_train = cifar10['X_train']
y_train = cifar10['y_train']
X_test = cifar10['X_test']
y_test = cifar10['y_test']

print "Training data:"
print "Number of examples: ", X_train.shape[0]
print "Number of channels:",X_train.shape[1]
```

```
print "Image size:", X_train.shape[2], X_train.shape[3]
print
print "Test data:"
print "Number of examples:", X_test.shape[0]
print "Number of channels:", X_test.shape[1]
print "Image size:",X_test.shape[2], X_test.shape[3]

# normalise data
print "mean before normalization:", np.mean(X_train)
print "std before normalization:", np.std(X_train)

mean=[0,0,0]
std=[0,0,0]
newX_train = np.ones(X_train.shape)
newX_test = np.ones(X_test.shape)
for i in xrange(3):
    mean[i] = np.mean(X_train[:,i,:,:])
    std[i] = np.std(X_train[:,i,:,:])

for i in xrange(3):
    newX_train[:,i,:,:] = X_train[:,i,:,:] - mean[i]
    newX_train[:,i,:,:] = newX_train[:,i,:,:] / std[i]
    newX_test[:,i,:,:] = X_test[:,i,:,:] - mean[i]
    newX_test[:,i,:,:] = newX_test[:,i,:,:] / std[i]


X_train = newX_train
X_test = newX_test

print "mean after normalization:", np.mean(X_train)
print "std after normalization:", np.std(X_train)

batchSize = 50                    #-- Training Batch Size
num_classes = 10                  #-- Number of classes in CIFAR-10 dataset
num_epochs = 10                   #-- Number of epochs for training
learningRate= 0.001               #-- Learning rate for the network
lr_weight_decay = 0.95            #-- Learning weight decay. Reduce the learn rate by 0.95
after epoch


img_rows, img_cols = 32, 32       #-- input image dimensions

Y_train = np_utils.to_categorical(y_train, num_classes)
Y_test = np_utils.to_categorical(y_test, num_classes)


# build model VGG Net
model = Sequential()
#-- layer 1
model.add(Convolution2D(6, 5, 5,
```

```
                        border_mode='valid',
                        input_shape=(3, img_rows, img_cols) ,dim_ordering="th"))
model.add(Convolution2D(16, 5, 5, dim_ordering="th"))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2),dim_ordering="th"))
#--layer 2
model.add(Convolution2D(26, 3, 3,dim_ordering="th"))
model.add(Convolution2D(36, 3, 3,dim_ordering="th"))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2),dim_ordering="th"))
#--layer 3
#model.add(Convolution2D(256, 3, 3,dim_ordering="th"))
#model.add(Dropout(0.4))
#model.add(Convolution2D(256, 3, 3,dim_ordering="th"))
#model.add(Dropout(0.4))
#model.add(Activation('relu'))
#model.add(MaxPooling2D(pool_size=(2, 2),dim_ordering="th"))
#-- layer 4
model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
#-- layer 5
model.add(Dense(256))
model.add(Activation('relu'))
#-- layer 6
model.add(Dense(num_classes))
#-- loss
model.add(Activation('softmax'))
print model.summary()

sgd = SGD(lr=learningRate, decay = lr_weight_decay)
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
#-- switch verbose=0 if you get error "I/O operation from closed file"
history = model.fit(X_train, Y_train, batch_size=batchSize, nb_epoch=num_epochs,
          verbose=1, shuffle=True, validation_data=(X_test, Y_test))

#-- summarize history for accuracy
plt.figure(1)
# plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'], label = 'Deep Net')
plt.title('model accuracy [validation]')
plt.ylabel('accuracy')
plt.xlabel('epoch')

plt.figure(2)
#-- summarize history for loss
# plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'], label = 'Deep Net')
```

```
plt.title('model loss [validation]')
plt.ylabel('loss')
plt.xlabel('epoch')

learningRate= 0.001              #-- Learning rate for the network

# build model Default Net
model = Sequential()                                          #-- Sequential container.
model.add(Convolution2D(6, 5, 5,                              #-- 6 outputs (6 filters),
5x5 convolution kernel
                        border_mode='valid',
                        input_shape=(3, img_rows, img_cols),dim_ordering="th"))       #-- 3
input depth (RGB)
model.add(Activation('relu'))                                 #-- ReLU non-linearity
model.add(MaxPooling2D(pool_size=(2, 2),  dim_ordering="th"))                          #-- A
max-pooling on 2x2 windows
model.add(Convolution2D(16, 5, 5, dim_ordering="th"))                                  #-- 16
outputs (16 filters), 5x5 convolution kernel
model.add(Activation('relu'))                                 #-- ReLU non-linearity
model.add(MaxPooling2D(pool_size=(2, 2),  dim_ordering="th"))                          #-- A
max-pooling on 2x2 windows

model.add(Flatten())                                          #-- eshapes a 3D tensor of
16x5x5 into 1D tensor of 16*5*5
model.add(Dense(120))                                         #-- 120 outputs fully
connected layer
model.add(Activation('relu'))                                 #-- ReLU non-linearity
model.add(Dense(84))                                          #-- 84 outputs fully
connected layer
model.add(Activation('relu'))                                 #-- ReLU non-linearity
model.add(Dense(num_classes))                                 #-- 10 outputs fully
connected layer (one for each class)
model.add(Activation('softmax'))                              #-- converts the output to
a log-probability. Useful for classification problems

print model.summary()


sgd = SGD(lr=learningRate, decay = lr_weight_decay)
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

#-- switch verbose=0 if you get error "I/O operation from closed file"
history = model.fit(X_train, Y_train, batch_size=batchSize, nb_epoch=num_epochs,
          verbose=1, shuffle=True, validation_data=(X_test, Y_test))

plt.figure(1)
# plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'], '-', label = 'Default Net')
```

```
plt.legend(loc='upper left')

plt.figure(2)
#-- summarize history for loss
# plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'], '-', label = 'Default Net')
plt.legend(loc='upper left')

plt.show()
```

Q12.py

```
import numpy as np
import os
os.environ["THEANO_FLAGS"] = "mode=FAST_RUN,device=cpu,floatX=float32"
import theano
import keras
import scipy
from math import *
from keras.datasets import cifar10
from keras.models  import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.optimizers import SGD
from keras.utils import np_utils
import matplotlib.pyplot as plt
from tempfile import TemporaryFile
#%matplotlib inline

# load cifar dataset
cifar10 = np.load('../../../data/lab2/lab2_data/cifar10_data.npz')
X_train = cifar10['X_train']
y_train = cifar10['y_train']
X_test = cifar10['X_test']
y_test = cifar10['y_test']
num_classes = 10


# adds salt and pepper noise to data
def noisy(image):
    row,col,ch = image.shape
    s_vs_p = 0.5
    amount = 0.004
    out = image
    # Salt mode
    num_salt = np.ceil(amount * image.size * s_vs_p)
    coordsy = [np.random.randint(0, image.shape[0] - 1, int(num_salt))]
    coordsx = [np.random.randint(0, image.shape[1] - 1, int(num_salt))]
    out[coordsx, coordsy, :] = 255
```

```
    # Pepper mode
    num_pepper = np.ceil(amount * image.size * (1. - s_vs_p))
    coordsy = [np.random.randint(0, image.shape[0] - 1, int(num_pepper))]
    coordsx = [np.random.randint(0, image.shape[1] - 1, int(num_pepper))]
    out[coordsx, coordsy, :] = 0
    return out

# def rotate_image(image):


def create_noisy_data(X_train, y_train):
    no_noisy_samples = 10000
    X_train_noisy = np.zeros((X_train.shape[0] + 2 * no_noisy_samples, X_train.shape[1],
X_train.shape[2], X_train.shape[3]))
    y_train_noisy = np.zeros((y_train.shape[0] + 2 * no_noisy_samples, 1), dtype=np.int)
    noisy_coords = np.random.randint(0, X_train.shape[0] - 1, no_noisy_samples)
    rotated_coords = np.random.randint(0, X_train.shape[0] - 1, no_noisy_samples)
    for i in xrange(no_noisy_samples):
        plot_image1 = X_train[noisy_coords[i],:,:,:].transpose(1,2,0)
        plot_image2 = X_train[rotated_coords[i],:,:,:].transpose(1,2,0)
        noisy_image = noisy(plot_image1)
        rotated_image = scipy.ndimage.interpolation.rotate(plot_image2, 90)
        X_train_noisy[2 * i,:,:,:] = noisy_image.transpose(2,0,1)
        X_train_noisy[2 * i + 1, :,:,:] = rotated_image.transpose(2,0,1)
        y_train_noisy[2 * i] = y_train[noisy_coords[i]]
        y_train_noisy[2 * i + 1] = y_train[rotated_coords[i]]

    X_train_noisy[2 * no_noisy_samples:,:,:,:] = X_train
    y_train_noisy[2 * no_noisy_samples:, :] = y_train
    permut = np.random.permutation(X_train_noisy.shape[0])
    X_train_noisy = X_train_noisy[permut,:,:,:]
    y_train_noisy = y_train_noisy[permut,:]
    cifar10_data_noisy = TemporaryFile()
    np.savez('../../../data/lab2/lab2_data/cifar10_data_noisy', X_train_noisy, y_train_noisy)

create_noisy_data(X_train, y_train)
cifar10 = np.load('../../../data/lab2/lab2_data/cifar10_data_noisy.npz')
X_train = cifar10['arr_0']
y_train = cifar10['arr_1']


print "Training data:"
print "Number of examples: ", X_train.shape[0]
print "Number of channels:",X_train.shape[1]
print "Image size:", X_train.shape[2], X_train.shape[3]
print
print "Test data:"
print "Number of examples:", X_test.shape[0]
print "Number of channels:", X_test.shape[1]
```

```
print "Image size:",X_test.shape[2], X_test.shape[3]


# # normalise data
print "mean before normalization:", np.mean(X_train)
print "std before normalization:", np.std(X_train)

mean=[0,0,0]
std=[0,0,0]
newX_train = np.ones(X_train.shape)
newX_test = np.ones(X_test.shape)
for i in xrange(3):
    mean[i] = np.mean(X_train[:,i,:,:])
    std[i] = np.std(X_train[:,i,:,:])

for i in xrange(3):
    newX_train[:,i,:,:] = X_train[:,i,:,:] - mean[i]
    newX_train[:,i,:,:] = newX_train[:,i,:,:] / std[i]
    newX_test[:,i,:,:] = X_test[:,i,:,:] - mean[i]
    newX_test[:,i,:,:] = newX_test[:,i,:,:] / std[i]


X_train = newX_train
X_test = newX_test

print "mean after normalization:", np.mean(X_train)
print "std after normalization:", np.std(X_train)

batchSize = 50                    #-- Training Batch Size                #-- Number of classes
in CIFAR-10 dataset
num_epochs = 10                   #-- Number of epochs for training
learningRate= 0.001               #-- Learning rate for the network
lr_weight_decay = 0.95            #-- Learning weight decay. Reduce the learn rate by 0.95
after epoch


img_rows, img_cols = 32, 32       #-- input image dimensions

Y_train = np_utils.to_categorical(y_train, num_classes)
Y_test = np_utils.to_categorical(y_test, num_classes)

model = Sequential()                                                #-- Sequential container.

model.add(Convolution2D(6, 5, 5,                                    #-- 6 outputs (6 filters),
5x5 convolution kernel
                        border_mode='valid',
                        input_shape=(3, img_rows, img_cols)))       #-- 3 input depth (RGB)
model.add(Activation('relu'))                                       #-- ReLU non-linearity
model.add(MaxPooling2D(pool_size=(2, 2)))                           #-- A max-pooling on 2x2
windows
```

```
model.add(Convolution2D(16, 5, 5))                                  #-- 16 outputs (16
filters), 5x5 convolution kernel
model.add(Activation('relu'))                                       #-- ReLU non-linearity
model.add(MaxPooling2D(pool_size=(2, 2)))                           #-- A max-pooling on 2x2
windows

model.add(Flatten())                                                #-- eshapes a 3D tensor of
16x5x5 into 1D tensor of 16*5*5
model.add(Dense(120))                                               #-- 120 outputs fully
connected layer
model.add(Activation('relu'))                                       #-- ReLU non-linearity
model.add(Dense(84))                                                #-- 84 outputs fully
connected layer
model.add(Activation('relu'))                                       #-- ReLU non-linearity
model.add(Dense(num_classes))                                       #-- 10 outputs fully
connected layer (one for each class)
model.add(Activation('softmax'))                                    #-- converts the output to
a log-probability. Useful for classification problems

print model.summary()

sgd = SGD(lr=learningRate, decay = lr_weight_decay)
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

#-- switch verbose=0 if you get error "I/O operation from closed file"
history = model.fit(X_train, Y_train, batch_size=batchSize, nb_epoch=num_epochs,
          verbose=1, shuffle=True, validation_data=(X_test, Y_test))

#-- summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

#-- summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

#-- test the network
score = model.evaluate(X_test, Y_test, verbose=0)
```

```
print 'Test score:', score[0]
print 'Test accuracy:', score[1]
```