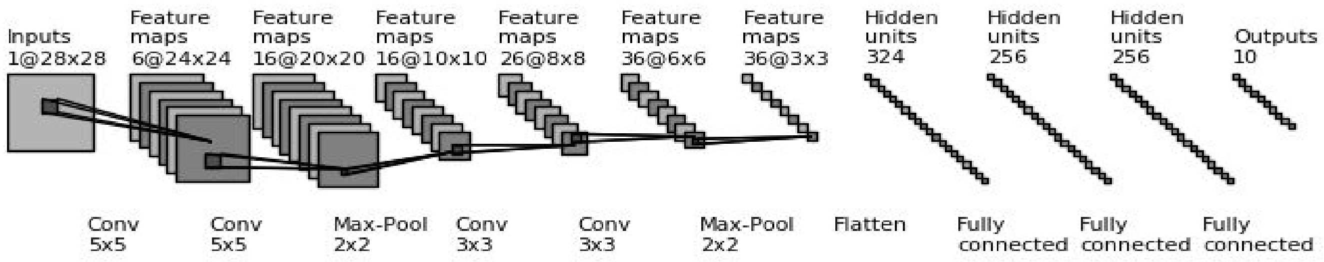
ML Assignment 2

Abhinav Moudgil
201331039

Train a neural network (CNN) on MNIST dataset with the following weight initializations:

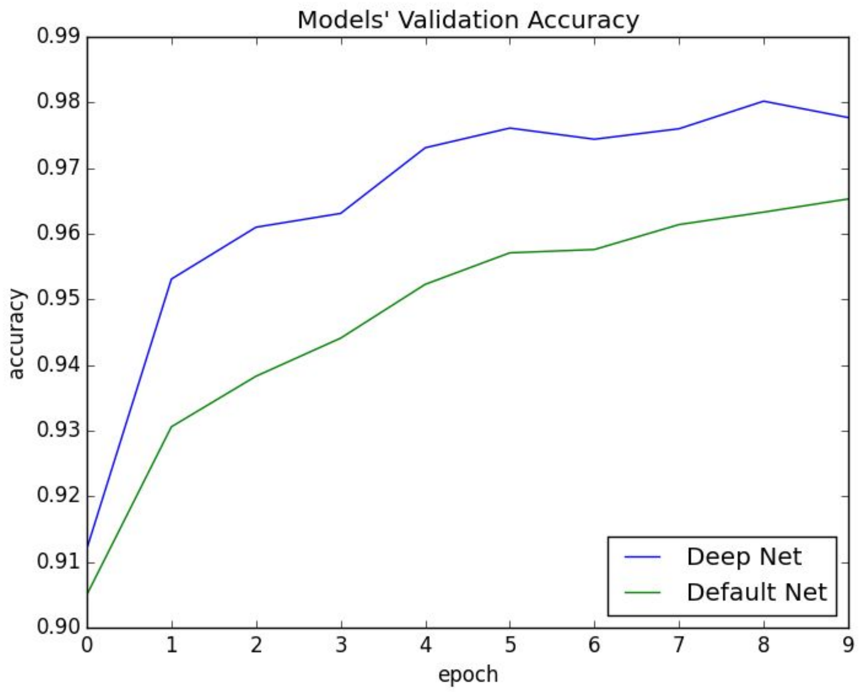
1. *All weights are either 0 or constant*
2. *All weights are random and random scale*
3. *Random within range [-1, 1]*
4. *Xavier weight initialisations*

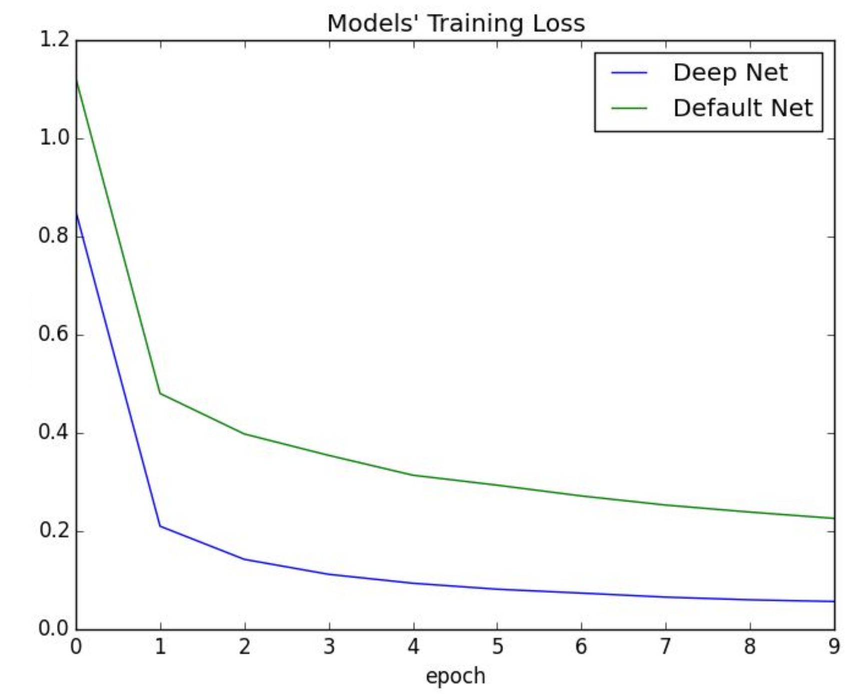
All the experiments in this assignment are done on CPU with the following architecture:



[Total Parameters: 166364]

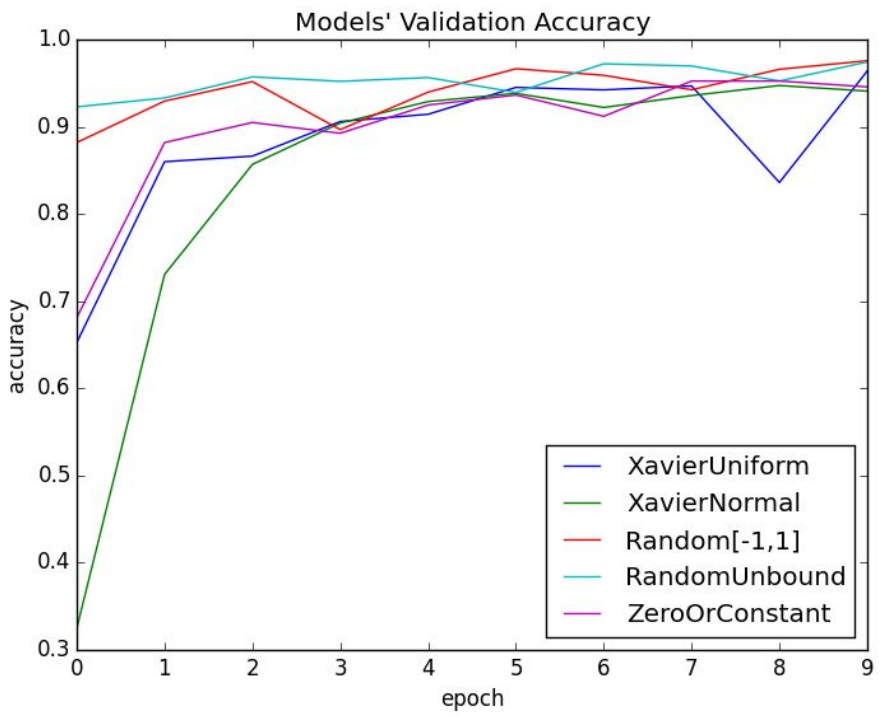
This deep net performs much better than the default shallow network (with two convolution and two max pool layers). Following are the plots of MNIST dataset which justifies its performance:

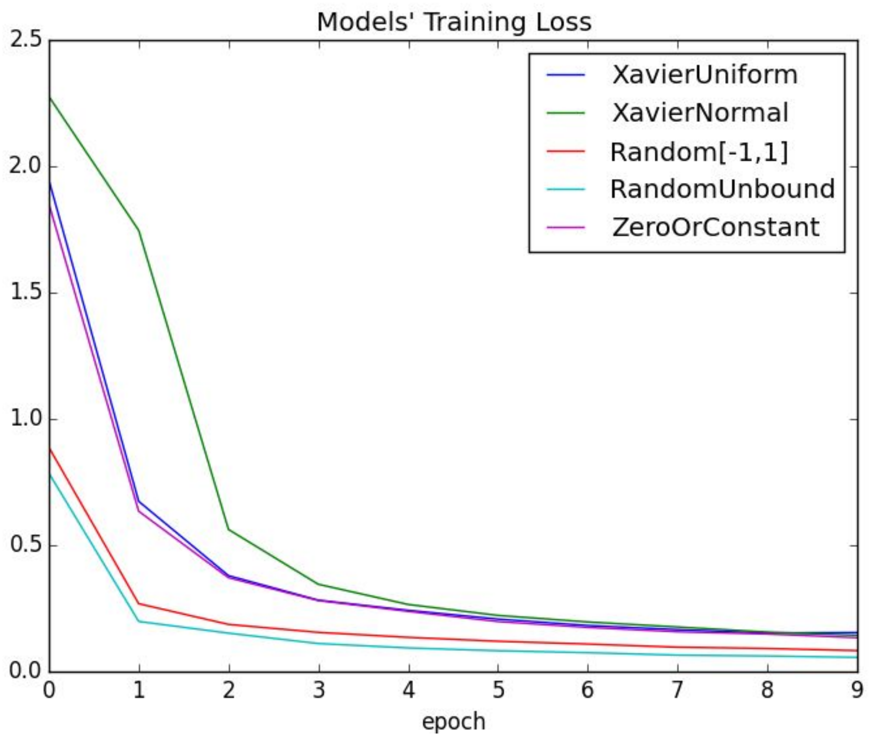




Following weight initializations are done:

1. Xavier Uniform
2. Xavier Normal
3. Random weights in range [-1, 1]
4. Random weights unbounded
5. Zero or Constant





If the network starts with too small values, then the signal dies as it passes through the layers. If network starts with large values, it gives too large to handle. Xavier initialisation keep the weights in 'range'.

Here network is small, so in all the above initializations, network converges. Also, inputs are normalised, so there is very less probability of overshooting or saturating values. Hence, Xavier Initialization and Random Initialization performs equally well (can be observed from plot).

Train the Neural Network with a custom loss function (for ex: Hinge loss, variation of hinge loss etc.). You can re-use the network from the 1st part and retrain it using your custom loss function.

Following loss functions are used:

1. **Hinge Loss**
   I wrote my own custom Hinge Loss function which is a bit different from the default one implemented in Keras. Default Hinge Loss function in Keras assumes that the labels are +1 or -1 but here labels are one hot vector i.e. 0 or 1. So the default Hinge Loss function won't work at all (also tested it, accuracy remains the same [~ 0.1] and loss doesn't decrease at all). But when I used my custom Hinge Loss function *without softmax*[1], it works as

[1] Softmax activation layer converts the output to log probabilities which works well with probabilistic logarithmic loss functions like Cross Entropy and KL divergence but Hinge Loss operates directly on the output, so softmax layer should be avoided with Hinge Loss.

expected. Following is the code snippet of my custom Hinge Loss function:

```
def custom_hinge(y_true, y_pred):
        y_true = y_true*2 - 1
        y_pred = y_pred*2 - 1
        return K.mean(K.maximum(1. - y_true * y_pred, 0.), axis=-1)
```

2. **Squared Hinge Loss**
   I modified my custom Hinge Loss to get Squared Hinge Loss function. I used it in the same architecture but without softmax activation layer. It works better than Hinge Loss.

```
def custom_hinge_square(y_true, y_pred):
        y_true = y_true*2 - 1
        y_pred = y_pred*2 - 1
        return K.mean(K.square(K.maximum(1. - y_true * y_pred, 0.)), axis=-1)
```

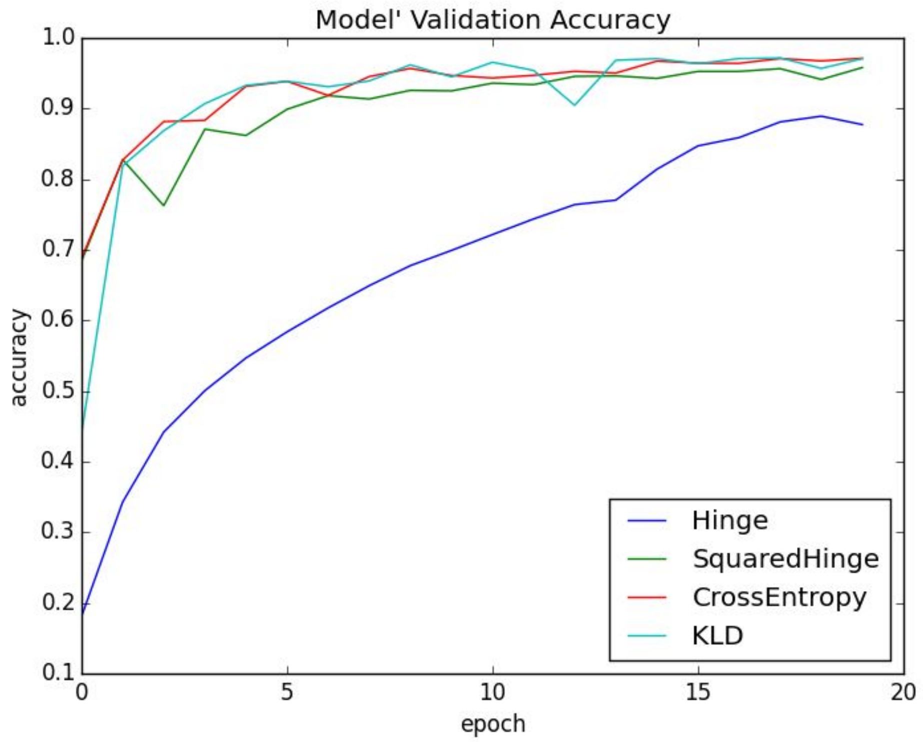
3. **Categorical Cross Entropy**
   (Used the default one implemented in Keras)
   It computes the cross-entropy between predictions (p) and targets (t).

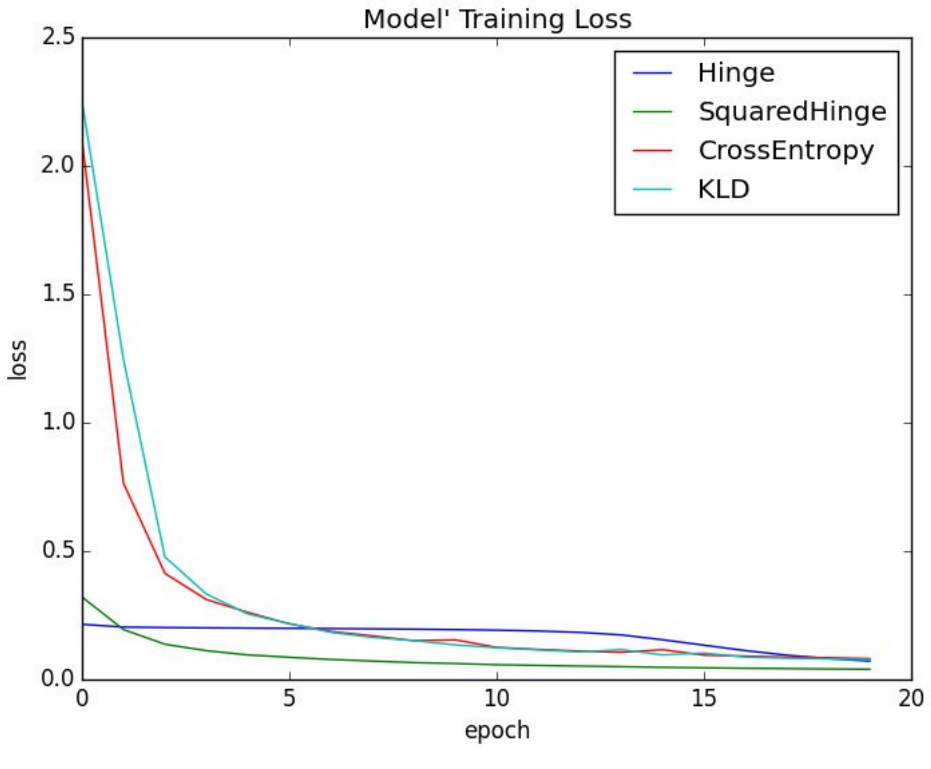
$$L_i = -\sum_j t_{i,j} \log(p_{i,j})$$

4. **KL Divergence**
   (Used the default one implemented in Keras)
   Let us say we have two probability distributions: Y (true labels) and P(predictions). Then KL Divergence loss is defined as:

$$\mathrm{KL}(\mathrm{Y}\|\mathrm{P}) = \sum_{\mathrm{k}=1}^{\mathrm{K}} \mathrm{Y}_{\mathrm{k}}(\mathrm{x}) \log \frac{\mathrm{Y}_{\mathrm{k}}(\mathrm{x})}{\mathrm{P}_{\mathrm{k}}(\mathrm{x})}$$

On this MNIST dataset, Cross Entropy Loss performs better than Hinge Loss but Squared Hinge Loss function performs equally well as Cross Entropy Loss and KL Divergence.


Model' Validation Accuracy
accuracy
epoch
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
5
10
15
20
Hinge
SquaredHinge
CrossEntropy
KLD


Model' Training Loss
loss
epoch
2.5
2.0
1.5
1.0
0.5
0.0
0
5
10
15
20
Hinge
SquaredHinge
CrossEntropy
KLD

Appendix

This section contain the codes which generated the plots.

Weight initialization

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

from keras import backend as K

#Load the dataset
from keras.datasets import mnist

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

batchSize = 512                    #-- 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 = 28, 28

"""Load and format Data
"""
# 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, 1, 28, 28)
X_test = X_test.reshape(10000, 1, 28, 28)

# 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
```

```python
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, num_classes)
Y_test = np_utils.to_categorical(y_test, num_classes)

inits = []

def binaryConstant(shape, name=None):
    value = np.random.choice([0, np.random.random()], shape)
    return K.variable(value, name=name)

def randomUnbound(shape, name=None):
    value = (2*np.random.random(shape) - 1)*(np.random.randint(10)+1)
    return K.variable(value, name=name)

def randomMinusOneToOne(shape, name=None):
    value = 2*np.random.random(shape) - 1
    return K.variable(value, name=name)

inits.append('glorot_uniform')
inits.append('glorot_normal')
inits.append(randomMinusOneToOne)
inits.append(randomUnbound)
inits.append(binaryConstant)


models = []

for init in inits:
    # build custom net
    model = Sequential()
    #-- layer 1
    model.add(Convolution2D(6, 5, 5,
                            border_mode='valid',
                            init = init,
                            input_shape=(1, 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(Flatten())
    model.add(Dense(256))
    model.add(Activation('relu'))
    #-- layer 4
    model.add(Dense(256))
    model.add(Activation('relu'))
    #-- layer 5
    model.add(Dense(num_classes))
    #-- loss
    model.add(Activation('softmax'))                                    #-- converts the
output to a log-probability. Useful for classification problems
    print(model.summary())
    models.append(model)

histories = []

for model in models:
    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))

    histories.append(history)

for history in histories:
    plt.plot(history.history['val_acc'])

plt.title('Models\' Validation Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['XavierUniform', 'XavierNormal', 'Random[-1,1]', 'RandomUnbound',
'ZeroOrConstant'], loc='lower right')
plt.show()

for history in histories:
    plt.plot(history.history['loss'])

plt.title('Models\' Training Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['XavierUniform', 'XavierNormal', 'Random[-1,1]', 'RandomUnbound',
'ZeroOrConstant'], loc='upper right')
plt.show()
```

Loss functions

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

from keras import backend as K

#Load the dataset
from keras.datasets import mnist

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

batchSize = 512                     #-- Training Batch Size
num_classes = 10                   #-- Number of classes in CIFAR-10 dataset
num_epochs = 20                    #-- 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 = 28, 28

"""Load and format Data
"""
# 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, 1, 28, 28)
X_test = X_test.reshape(10000, 1, 28, 28)

# 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, num_classes)
Y_test = np_utils.to_categorical(y_test, num_classes)
```

```
models = []

for i in xrange(4):
    model = Sequential()
    #-- layer 1
    model.add(Convolution2D(6, 5, 5,
                           border_mode='valid',
                           input_shape=(1, 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(Flatten())
    model.add(Dense(256))
    model.add(Activation('relu'))
    #-- layer 4
    model.add(Dense(256))
    model.add(Activation('relu'))
    #-- layer 5
    model.add(Dense(10))
    #-- loss
    if (i > 1):
        model.add(Activation('softmax')) #-- converts the output to a log-probability. Useful
for classification problems
    print(model.summary())
    models.append(model)

def custom_hinge(y_true, y_pred):
    y_true = y_true*2 - 1
    y_pred = y_pred*2 - 1
    return K.mean(K.maximum(1. - y_true * y_pred, 0.), axis=-1)

def custom_hinge_square(y_true, y_pred):
    y_true = y_true*2 - 1
    y_pred = y_pred*2 - 1
    return K.mean(K.square(K.maximum(1. - y_true * y_pred, 0.)), axis=-1)

losses = []

losses.append(custom_hinge)
losses.append(custom_hinge_square)
losses.append('categorical_crossentropy')
losses.append('kullback_leibler_divergence')
```

```
histories = []

for model, loss in zip(models, losses):
    sgd = SGD(lr=learningRate, decay = lr_weight_decay)
    model.compile(loss=loss,
                  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))
    histories.append(history)

for history in histories:
    plt.plot(history.history['val_acc'])

plt.title('Model\' Validation Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Hinge', 'SquaredHinge', 'CrossEntropy', 'KLD'], loc='lower right')
plt.show()

for history in histories:
    plt.plot(history.history['loss'])

plt.title('Model\' Training Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Hinge', 'SquaredHinge', 'CrossEntropy', 'KLD'], loc='upper right')
plt.show()
```