p6_cnn.py

March 19, 2018

In [10]:
```
# Import all relevant libraries
import warnings
def fxn():
        warnings.warn("deprecated",DeprecationWarning)

with warnings.catch_warnings( ):
    warnings.simplefilter("ignore")
    fxn( )

# Keras imports
import keras
from keras.models import Sequential
from keras.layers import BatchNormalization, MaxPooling2D, Permute, Flatten, Softmax,
from keras.utils import np_utils
from keras.optimizers import Adam

# Other
import numpy as np
import h5py
import sklearn
from sklearn import preprocessing
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras import regularizers
import random
```

In [11]:
```
# Load data from specific trial
def get_trial(trial_num):
    trial = h5py.File('data/A0' + str(trial_num) + 'T_slice.mat', 'r')
    X = np.copy(trial['image'])
    y = np.copy(trial['type'])
    y = y[0,0:X.shape[0]:1]
    y = np.asarray(y, dtype=np.int32)
    y -= 769                            # shift class labels to [0-3]
    X = np.nan_to_num(X)[:, :22, :]     # remove EOG channels
    return X, y
```

```
    def get_all_trials():
        X_total = np.concatenate([get_trial(trial_num)[0] for trial_num in range(1, 9)], a
        y_total = np.concatenate([get_trial(trial_num)[1] for trial_num in range(1, 9)], a
        return X_total, y_total

    def stratified_train_test_split(X, y, k):
        ''' Returns a stratified train/test split, for k number of splits.
        Return value is in the form [(train indices, test indices), ... for k folds ]
        '''
        skf = StratifiedKFold(n_splits=k)
        return skf.split(X, y)
```

In [12]:
```
# Get the data from one person
num_folds = 5
# X, y = get_trial(1)
# num_trials = 1

def get_normalized_data():
    # Get the data from all the people
    X, y = get_all_trials()
    num_trials = 9
    X = np.transpose(X, (0,2,1))
    # 0 mean and unit variance
    temp = np.reshape(X, (X.shape[0], -1))
    X = np.reshape(preprocessing.scale(temp), X.shape)

    # Generate train/test split
    y_cat = keras.utils.to_categorical(y, num_classes=4)
    X_train, X_test, y_train, y_test = train_test_split(X, y)
    tt_splits = stratified_train_test_split(X_train, y_train, num_folds)

    one_hot_train = keras.utils.to_categorical(y_train)
    one_hot_test = keras.utils.to_categorical(y_test)

    return X_train, X_test, one_hot_train, one_hot_test, tt_splits

# The data for each trial is of the shape (288, 22, 1000)
#   There are 288 samples per trial (12 of each class per "run", 4 classes, 6 "runs"
#                                    at different time periods of the day)
#   There are 22 electrodes from the EEG (represents spatial aspect of the signals)
#   There are 1000 time units (4 seconds of data, sampled at 250Hz). The first 250 un
#                                    are when no movement occurs (but the cue is heard
#                                    the next 750 units are when the movement occurs
# The labels for each trial belong in one of 4 classes
#   0 - left
#   1 - right
#   2 - foot
```

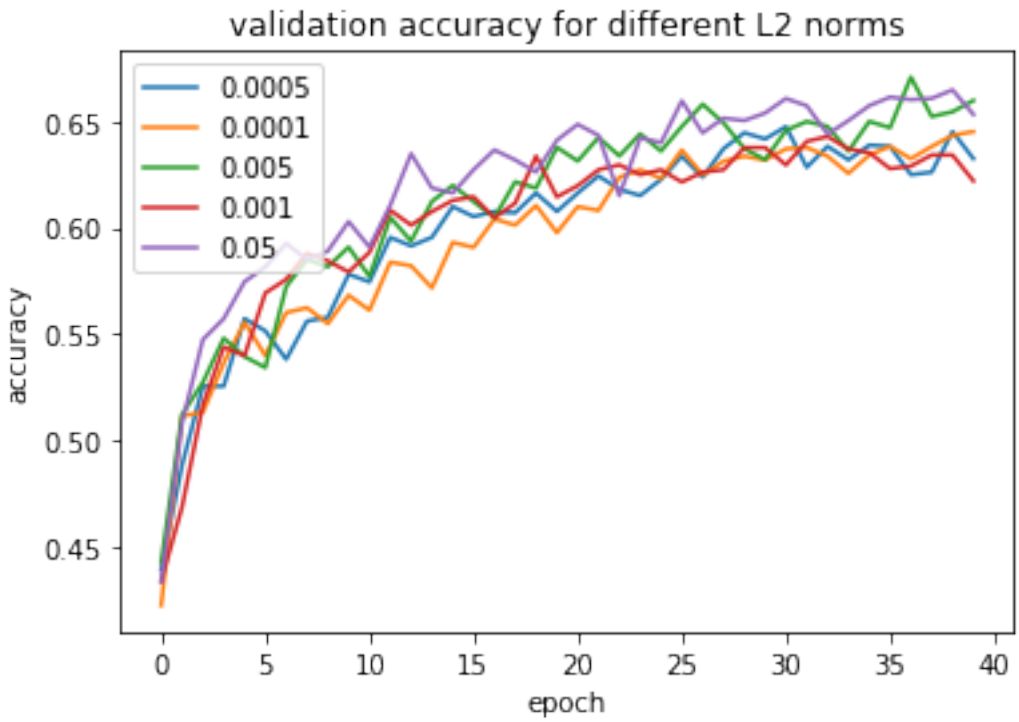
```
#   3 - tongue
# print(X_train.shape)
# print(X_test.shape)
# print(one_hot_train.shape)
# print(one_hot_test.shape)
```

In [67]:
```
# more accurate version of 1D NN
# def CNN_1D(nl=2):
#     # Naive implementation
#     num_layers = nl
#     model = Sequential()
#     for _ in np.arange(num_layers):
#         model.add(Conv1D(filters = 25, kernel_size = 10, activation = 'elu', input_
#         model.add(BatchNormalization())
#         model.add(Dropout(0.4))
#     model.add(AveragePooling1D(pool_size=(75), strides=(15)))
#     model.add(Dropout(0.5))
#     model.add(Flatten())
#     model.add(Dense(units=4, kernel_initializer='glorot_normal', activity_regulariz


#     adam = Adam(lr=0.0007, decay=0.005)
# #     sgd = keras.optimizers.SGD(lr=0.01, momentum=0.7, decay=0.001, nesterov=True)
#     model.compile(optimizer=adam,
#                   loss='categorical_crossentropy',
#                   metrics=['accuracy'])
#     return model
```

In [82]:
```
# grid search for number of layers
batch_size = 32
num_epochs = 40
history = []
for l2 in [0.0005, 0.0001, 0.005, 0.001, 0.05]:
    X_tr, X_te, y_tr, y_te, tt_splits = get_normalized_data()
    print("l2_norm", l2)
    nl_histories = []
    for i, (train,test) in enumerate(tt_splits):
        print("running fold", i)
        model = CNN_1D(l2=l2)
        nl_histories.append(model.fit(X_tr[train], y_tr[train], validation_data=(X_tr
    history.append(np.average(nl_histories, axis=0))

for hist_tuple in history:
    plt.plot(hist_tuple)

plt.title('validation accuracy for different L2 norms')
plt.ylabel('accuracy')
plt.xlabel('epoch')
```

```
        plt.legend(['0.0005', '0.0001', '0.005', '0.001', '0.05'], loc='upper left')
        plt.show()
        plt.savefig('l2_norm.pdf')

        for hist_tuple in history:
            print(sum(hist_tuple[-10:])/len(hist_tuple[-10:]))
```

```
l2_norm 0.0005
running fold 0
running fold 1
running fold 2
running fold 3
running fold 4
l2_norm 0.0001
running fold 0
running fold 1
running fold 2
running fold 3
running fold 4
l2_norm 0.005
running fold 0
running fold 1
running fold 2
running fold 3
running fold 4
l2_norm 0.001
running fold 0
running fold 1
running fold 2
running fold 3
running fold 4
l2_norm 0.05
running fold 0
running fold 1
running fold 2
running fold 3
running fold 4
```




```
0.6352318688295282
0.636594899038365
0.6514149575248166
0.6332849951630579
0.6570739652386945
```

```
<matplotlib.figure.Figure at 0x146858198>
```

```
In [87]: def CNN_1D(nl=2, optimizer='adam', filters=30, lr=0.001, ld=0.0001, dp1=0.3, dp2=0.4,
             # Naive implementation
             num_layers = nl
             model = Sequential()
             for _ in np.arange(num_layers):
                 model.add(Conv1D(filters = filters, kernel_size = 10, activation = 'elu', inpu
                 model.add(BatchNormalization())
                 model.add(Dropout(dp1))

             model.add(AveragePooling1D(pool_size=(75), strides=(15)))
             model.add(Dropout(dp2))
             model.add(Flatten())
             model.add(Dense(units=4, kernel_initializer='glorot_normal', activity_regularizer=
```

```
            optimizer = Adam(lr=lr, decay=ld)
        #     sgd = keras.optimizers.SGD(lr=0.01, momentum=0.7, decay=0.001, nesterov=True)
            model.compile(optimizer=optimizer,
                          loss='categorical_crossentropy',
                          metrics=['accuracy'])
            return model
```

```
In [78]: CNN_1D()
```

```
(None, 991, 25)
(None, 982, 25)
```

```
Out[78]: <keras.models.Sequential at 0x14e3b3470>
```

```
In [76]: # grid search for number of layers
         batch_size = 32
         num_epochs = 40
         history = []
         for nl in [1,2,3,5,7,9]:
             X_tr, X_te, y_tr, y_te, tt_splits = get_normalized_data()
             print("running number of layers", nl)
             nl_histories = []
             for i, (train,test) in enumerate(tt_splits):
                 print("running fold", i)
                 model = CNN_1D(nl)
                 nl_histories.append(model.fit(X_tr[train], y_tr[train], validation_data=(X_tr
             history.append(np.average(nl_histories, axis=0))

         for hist_tuple in history:
             plt.plot(hist_tuple)

         plt.title('validation accuracy for different layers')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(['One layer', 'Two layers', 'Three layers', 'Five layers', 'Seven layers',
         # plt.legend(['One layer', 'Two layers'], loc='upper left')
         plt.show()
         plt.savefig('layers.pdf')

         for hist_tuple in history:
             print(sum(hist_tuple[-10:])/len(hist_tuple[-10:]))
```

```
running number of layers 1
running fold 0
running fold 1
running fold 2
running fold 3
running fold 4
```

```
running number of layers 2
running fold 0
running fold 1
running fold 2
running fold 3
running fold 4
running number of layers 3
running fold 0
running fold 1
running fold 2
running fold 3
running fold 4
running number of layers 5
running fold 0
```

KeyboardInterrupt Traceback (most recent call last)

<ipython-input-76-9e2a98c44712> in <module>()

```
 10         print("running fold", i)
 11         model = CNN_1D(nl)
---> 12         nl_histories.append(model.fit(X_tr[train], y_tr[train], validation_data=(X_
 13     history.append(np.average(nl_histories, axis=0))
 14
```

~/code/EC239/p6/ece239_project/.env/lib/python3.6/site-packages/keras/models.py in fit

```
961                                 initial_epoch=initial_epoch,
962                                 steps_per_epoch=steps_per_epoch,
--> 963                                 validation_steps=validation_steps)
964
965     def evaluate(self, x=None, y=None,
```

~/code/EC239/p6/ece239_project/.env/lib/python3.6/site-packages/keras/engine/training.p

```
1703                                 initial_epoch=initial_epoch,
1704                                 steps_per_epoch=steps_per_epoch,
-> 1705                                 validation_steps=validation_steps)
1706
1707     def evaluate(self, x=None, y=None,
```

~/code/EC239/p6/ece239_project/.env/lib/python3.6/site-packages/keras/engine/training.p

```
1247                             val_outs = self._test_loop(val_f, val_ins,
1248                                                        batch_size=batch_size,
```

```
-> 1249                                                     verbose=0)
   1250                             if not isinstance(val_outs, list):
   1251                                 val_outs = [val_outs]


    ~/code/EC239/p6/ece239_project/.env/lib/python3.6/site-packages/keras/engine/training.p
   1424                     ins_batch[i] = ins_batch[i].toarray()
   1425
-> 1426                 batch_outs = f(ins_batch)
   1427                 if isinstance(batch_outs, list):
   1428                     if batch_index == 0:


    ~/code/EC239/p6/ece239_project/.env/lib/python3.6/site-packages/keras/backend/tensorflo
   2476         session = get_session()
   2477         updated = session.run(fetches=fetches, feed_dict=feed_dict,
-> 2478                               **self.session_kwargs)
   2479         return updated[:len(self.outputs)]
   2480


    ~/code/EC239/p6/ece239_project/.env/lib/python3.6/site-packages/tensorflow/python/clien
    903     try:
    904       result = self._run(None, fetches, feed_dict, options_ptr,
--> 905                          run_metadata_ptr)
    906       if run_metadata:
    907         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)


    ~/code/EC239/p6/ece239_project/.env/lib/python3.6/site-packages/tensorflow/python/clien
   1135     if final_fetches or final_targets or (handle and feed_dict_tensor):
   1136       results = self._do_run(handle, final_targets, final_fetches,
-> 1137                              feed_dict_tensor, options, run_metadata)
   1138     else:
   1139       results = []


    ~/code/EC239/p6/ece239_project/.env/lib/python3.6/site-packages/tensorflow/python/clien
   1353     if handle is None:
   1354       return self._do_call(_run_fn, self._session, feeds, fetches, targets,
-> 1355                            options, run_metadata)
   1356     else:
   1357       return self._do_call(_prun_fn, self._session, handle, feeds, fetches)


    ~/code/EC239/p6/ece239_project/.env/lib/python3.6/site-packages/tensorflow/python/clien
   1359   def _do_call(self, fn, *args):
   1360     try:
```

```
-> 1361       return fn(*args)
   1362     except errors.OpError as e:
   1363       message = compat.as_text(e.message)


~/code/EC239/p6/ece239_project/.env/lib/python3.6/site-packages/tensorflow/python/clie
   1338         else:
   1339           return tf_session.TF_Run(session, options, feed_dict, fetch_list,
-> 1340                                    target_list, status, run_metadata)
   1341
   1342     def _prun_fn(session, handle, feed_dict, fetch_list):


KeyboardInterrupt:
```

In [ ]:
```
# batch_size = 32
num_epochs = 40
X_tr, X_te, y_tr, y_te, tt_splits = get_normalized_data()
history = []
for batch_size in [32, 64, 128]:
    for nl in [1,2]:
        print("running number of layers", nl)
        nl_histories = []
        for i, (train,test) in enumerate(tt_splits):
            print("running fold", i)
            model = CNN_1D(nl)
            nl_histories.append(model.fit(X_tr[train], y_tr[train], validation_data=(X_
            history.append(np.average(nl_histories, axis=0))
for hist_tuple in history:
    plt.plot(hist_tuple)
plt.title('validation accuracy for different layers')
plt.ylabel('accuracy')
plt.xlabel('epoch')
# plt.legend(['One layer', 'Two layers', 'Three layers', 'Five layers', 'Seven layers'
plt.legend(['One layer', 'Two layers'], loc='upper left')
plt.show()
savefig('layers.pdf')
```

In [88]:
```
batch_size = 32
num_epochs = 40
val_split = 0.2
X_tr, X_te, y_tr, y_te, tt_splits = get_normalized_data()
avg_acc = 0
# for train_idx, test_idx in tt_splits:
#     print(train_idx, X_tr.shape)
#     X_train = X_tr[train_idx]
#     y_train = y_tr[train_idx]
```

```
        #     X_test = X_tr[test_idx]
        #     y_test = y_tr[test_idx]

        #     print(X_train.shape)
        model = CNN_1D()
        history = model.fit(X_tr, y_tr, validation_split=val_split, epochs=num_epochs, batch_s
        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()
        metrics = model.evaluate(X_test, y_test, batch_size=batch_size)
        avg_acc += metrics[0]

        print(metrics)


        metrics = model.evaluate(X_te, y_te, batch_size=batch_size)
        print('testing', metrics)
```

```
Train on 1382 samples, validate on 346 samples
Epoch 1/40
1382/1382 [==============================] - 14s 10ms/step - loss: 2.7161 - acc: 0.3459 - val_l
Epoch 2/40
1382/1382 [==============================] - 6s 4ms/step - loss: 2.3723 - acc: 0.4624 - val_los
Epoch 3/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.2231 - acc: 0.5195 - val_los
Epoch 4/40
1382/1382 [==============================] - 6s 4ms/step - loss: 2.1472 - acc: 0.5637 - val_los
Epoch 5/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.1084 - acc: 0.5948 - val_los
Epoch 6/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0944 - acc: 0.6107 - val_los
Epoch 7/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0823 - acc: 0.6165 - val_los
Epoch 8/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0731 - acc: 0.6187 - val_los
Epoch 9/40
1382/1382 [==============================] - 6s 4ms/step - loss: 2.0638 - acc: 0.6462 - val_los
Epoch 10/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0657 - acc: 0.6324 - val_los
Epoch 11/40
1382/1382 [==============================] - 6s 4ms/step - loss: 2.0523 - acc: 0.6773 - val_los
Epoch 12/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0469 - acc: 0.6751 - val_los
Epoch 13/40
```

```
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0527 - acc: 0.6556 - val_los
Epoch 14/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0463 - acc: 0.6758 - val_los
Epoch 15/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0376 - acc: 0.6932 - val_los
Epoch 16/40
1382/1382 [==============================] - 6s 4ms/step - loss: 2.0381 - acc: 0.6932 - val_los
Epoch 17/40
1382/1382 [==============================] - 6s 4ms/step - loss: 2.0479 - acc: 0.6874 - val_los
Epoch 18/40
1382/1382 [==============================] - 6s 4ms/step - loss: 2.0321 - acc: 0.7062 - val_los
Epoch 19/40
1382/1382 [==============================] - 7s 5ms/step - loss: 2.0161 - acc: 0.7330 - val_los
Epoch 20/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0259 - acc: 0.7062 - val_los
Epoch 21/40
1382/1382 [==============================] - 6s 4ms/step - loss: 2.0138 - acc: 0.7258 - val_los
Epoch 22/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0172 - acc: 0.7395 - val_los
Epoch 23/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0111 - acc: 0.7547 - val_los
Epoch 24/40
1382/1382 [==============================] - 6s 4ms/step - loss: 2.0168 - acc: 0.7359 - val_los
Epoch 25/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0090 - acc: 0.7323 - val_los
Epoch 26/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0051 - acc: 0.7402 - val_los
Epoch 27/40
1382/1382 [==============================] - 6s 4ms/step - loss: 2.0079 - acc: 0.7402 - val_los
Epoch 28/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0069 - acc: 0.7352 - val_los
Epoch 29/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0075 - acc: 0.7395 - val_los
Epoch 30/40
1382/1382 [==============================] - 5s 4ms/step - loss: 2.0008 - acc: 0.7656 - val_los
Epoch 31/40
1382/1382 [==============================] - 5s 4ms/step - loss: 1.9984 - acc: 0.7525 - val_los
Epoch 32/40
1382/1382 [==============================] - 6s 4ms/step - loss: 1.9978 - acc: 0.7648 - val_los
Epoch 33/40
1382/1382 [==============================] - 6s 4ms/step - loss: 1.9922 - acc: 0.7713 - val_los
Epoch 34/40
1382/1382 [==============================] - 6s 4ms/step - loss: 1.9940 - acc: 0.7670 - val_los
Epoch 35/40
1382/1382 [==============================] - 6s 4ms/step - loss: 1.9895 - acc: 0.7619 - val_los
Epoch 36/40
1382/1382 [==============================] - 7s 5ms/step - loss: 1.9945 - acc: 0.7576 - val_los
Epoch 37/40
```

```
1382/1382 [==============================] - 6s 4ms/step - loss: 1.9962 - acc: 0.7706 - val_los
Epoch 38/40
1382/1382 [==============================] - 6s 4ms/step - loss: 1.9880 - acc: 0.7742 - val_los
Epoch 39/40
1382/1382 [==============================] - 6s 4ms/step - loss: 1.9866 - acc: 0.7786 - val_los
Epoch 40/40
1382/1382 [==============================] - 6s 4ms/step - loss: 1.9860 - acc: 0.7931 - val_los
```


```
346/346 [==============================] - 1s 2ms/step
[1.9914478477025996, 0.7543352590820004]
576/576 [==============================] - 1s 2ms/step
testing [2.075380047162374, 0.6458333333333334]
```

In [ ]:
```
# Deep Implementation CNN

        model = Sequential()

        # Conv Pool Block 1
        model.add(Conv2D(filters=25, kernel_size=(10,1), input_shape=(1000, 22, 1), strides=1,
        model.add(Conv2D(filters = 25, kernel_size = (1,22), activation = 'elu'))
        model.add(Permute((1,3,2)))
        model.add(BatchNormalization())
```

```
model.add(MaxPooling2D(pool_size = (3,1), strides = (3,1)))
model.add(Dropout(0.5))
print(model.output_shape)

# Conv Pool Block 2
model.add(Conv2D(filters = 50, kernel_size = (10,25), activation = 'elu'))
model.add(Permute((1,3,2)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (3,1), strides = (3,1)))
model.add(Dropout(0.5))
print(model.output_shape)

# Conv Pool Block 3
model.add(Conv2D(filters = 100, kernel_size = (10,50), activation = 'elu'))
model.add(Permute((1,3,2)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (3,1), strides = (3,1)))
model.add(Dropout(0.5))
print(model.output_shape)

# Conv Pool Block 4
model.add(Conv2D(filters = 200, kernel_size = (10,100), activation = 'elu'))
model.add(Permute((1,3,2)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (3,1), strides = (3,1)))
model.add(Dropout(0.5))
print(model.output_shape)

# Dense Layers
model.add(Flatten())
model.add(Dense(units=4, kernel_initializer='glorot_normal', activation = 'softmax'))
print(model.output_shape)


#Adam = Adam(lr=0.15)
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])



X_train = X_train.transpose((0,2,1))
X_train = X_train[:, :, 0:22]

X_test = X_test.transpose((0,2,1))
X_test = X_test[:, :, 0:22]
```

```
        model.fit(X_train[:, :, :, None] , one_hot_train, epochs=30, batch_size=32)
        score = model.evaluate(X_test, one_hot_test, verbose=0)
        print(score)
```

In [ ]:
```
        # Shallow CNN
        model = Sequential()
        model.add(Conv2D(filters=40, kernel_size=(25,1), activation='elu', input_shape=(1000, 1
        print(model.output_shape)
        model.add(Permute((3, 2, 1)))
        print(model.output_shape)
        model.add(Conv2D(filters=40, kernel_size=(22, 40), activation='elu', data_format="chan
        print(model.output_shape)
        model.add(Permute((3, 2, 1)))
        print(model.output_shape)
        model.add(AveragePooling2D(pool_size=(75,1), strides=(15,1)))
        print(model.output_shape)
        model.add(Flatten())
        model.add(Dense(units=4, kernel_initializer='glorot_normal', activation = 'softmax'))
        print(model.output_shape)


        Adam = Adam(lr=0.15)
        model.compile(optimizer='adam',
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])



        X_train = X_train.transpose((0,2,1))
        X_train = X_train[:, :, 0:22]

        X_test = X_test.transpose((0,2,1))
        X_test = X_test[:, :, 0:22]


        model.fit(X_train[:, :, :, None] , one_hot_train, epochs=30, batch_size=32)
        score = model.evaluate(X_test, one_hot_test, verbose=0)
        print(score)
```

In [71]:
```
        print(nl_histories)
         np.average(nl_histories, axis=0)
```

```
[]
```

```
/Users/jayendrajog/code/EC239/p6/ece239_project/.env/lib/python3.6/site-packages/numpy/lib/fun
  avg = a.mean(axis)
/Users/jayendrajog/code/EC239/p6/ece239_project/.env/lib/python3.6/site-packages/numpy/core/_m
```

```
  ret = ret.dtype.type(ret / rcount)
```

```
Out[71]: nan
```