In [ ]:

```

```

In [1]:

```
# Importing Libraries
```

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
# Activities are the class labels
# It is a 6 class classification
ACTIVITIES = {
    0: 'WALKING',
    1: 'WALKING_UPSTAIRS',
    2: 'WALKING_DOWNSTAIRS',
    3: 'SITTING',
    4: 'STANDING',
    5: 'LAYING',
}

# Utility function to print the confusion matrix
def confusion_matrix(Y_true, Y_pred):
    Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_pred, axis=1)])

    return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])
```

In [ ]:

```

```

Data

In [3]:

```
# Data directory
DATADIR = 'UCI HAR Dataset'
```

In [4]:

```
# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration
SIGNALS = [
    "body_acc_x",
    "body_acc_y",
    "body_acc_z",
    "body_gyro_x",
    "body_gyro_y",
    "body_gyro_z",
    "total_acc_x",
    "total_acc_y",
    "total_acc_z"
]
```

In [16]:

```python
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
    signals_data = []

    for signal in SIGNALS:
        filename = f'{subset}/Inertial Signals/{signal}_{subset}.txt'
        signals_data.append(
            _read_csv(filename).as_matrix()
        )

    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))
```

In [18]:

```python
def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = f'{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).as_matrix()
```

In [7]:

```python
def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test
```

In [8]:

```python
# Importing tensorflow
np.random.seed(42)
import tensorflow as tf
tf.set_random_seed(42)
```

```
C:\Users\deep_learning\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Co
nversion of the second argument of issubdtype from `float` to `np.floating` is deprecated
. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

In [9]:

```python
# Configuring a session
session_conf = tf.ConfigProto(
    intra_op_parallelism_threads=1,
    inter_op_parallelism_threads=1
)
```

In [10]:

```python
# Import Keras
from keras import backend as K
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

```
Using TensorFlow backend.
```

In [12]:

```
# Importing libraries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
```

In [13]:

```
# Initializing parameters
epochs = 30
batch_size = 16
n_hidden = 32
```

In [14]:

```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [19]:

```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

In [20]:

```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)
```

- **Defining the Architecture of LSTM**

In [21]:

```
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))
```

In [22]:

```
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

In [23]:

```
# Training the model
model.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 92s 13ms/step - loss: 1.3018 - acc: 0.4395 -
val_loss: 1.1254 - val_acc: 0.4662
Epoch 2/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.9666 - acc: 0.5880 -
val_loss: 0.9491 - val_acc: 0.5714
```

```
Epoch 3/30
7352/7352 [==============================] - 97s 13ms/step - loss: 0.7812 - acc: 0.6408 -
val_loss: 0.8286 - val_acc: 0.5850
Epoch 4/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.6941 - acc: 0.6574 -
val_loss: 0.7297 - val_acc: 0.6128
Epoch 5/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.6336 - acc: 0.6912 -
val_loss: 0.7359 - val_acc: 0.6787
Epoch 6/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.5859 - acc: 0.7134 -
val_loss: 0.7015 - val_acc: 0.6939
Epoch 7/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.5692 - acc: 0.7477 -
val_loss: 0.5995 - val_acc: 0.7387
Epoch 8/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.4899 - acc: 0.7809 -
val_loss: 0.5762 - val_acc: 0.7387
Epoch 9/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.4482 - acc: 0.7886 -
val_loss: 0.7413 - val_acc: 0.7126
Epoch 10/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.4132 - acc: 0.8077 -
val_loss: 0.5048 - val_acc: 0.7513
Epoch 11/30
7352/7352 [==============================] - 89s 12ms/step - loss: 0.3985 - acc: 0.8274 -
val_loss: 0.5234 - val_acc: 0.7452
Epoch 12/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.3378 - acc: 0.8638 -
val_loss: 0.4114 - val_acc: 0.8833
Epoch 13/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.2947 - acc: 0.9051 -
val_loss: 0.4386 - val_acc: 0.8731
Epoch 14/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.2448 - acc: 0.9291 -
val_loss: 0.3768 - val_acc: 0.8921
Epoch 15/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.2157 - acc: 0.9331 -
val_loss: 0.4441 - val_acc: 0.8931
Epoch 16/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.2053 - acc: 0.9366 -
val_loss: 0.4162 - val_acc: 0.8968
Epoch 17/30
7352/7352 [==============================] - 89s 12ms/step - loss: 0.2028 - acc: 0.9404 -
val_loss: 0.4538 - val_acc: 0.8962
Epoch 18/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.1911 - acc: 0.9419 -
val_loss: 0.3964 - val_acc: 0.8999
Epoch 19/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1912 - acc: 0.9407 -
val_loss: 0.3165 - val_acc: 0.9030
Epoch 20/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1732 - acc: 0.9446 -
val_loss: 0.4546 - val_acc: 0.8904
Epoch 21/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1782 - acc: 0.9444 -
val_loss: 0.3346 - val_acc: 0.9063
Epoch 22/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1812 - acc: 0.9418 -
val_loss: 0.8164 - val_acc: 0.8582
Epoch 23/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1824 - acc: 0.9426 -
val_loss: 0.4240 - val_acc: 0.9036
Epoch 24/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1726 - acc: 0.9429 -
val_loss: 0.4067 - val_acc: 0.9148
Epoch 25/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1737 - acc: 0.9411 -
val_loss: 0.3396 - val_acc: 0.9074
Epoch 26/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1650 - acc: 0.9461 -
val_loss: 0.3806 - val_acc: 0.9019
```

```
Epoch 27/30
7352/7352 [==============================] - 89s 12ms/step - loss: 0.1925 - acc: 0.9415 - val_loss: 0.6464 - val_acc: 0.8850
Epoch 28/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.1965 - acc: 0.9425 - val_loss: 0.3363 - val_acc: 0.9203
Epoch 29/30
7352/7352 [==============================] - 92s 12ms/step - loss: 0.1889 - acc: 0.9431 - val_loss: 0.3737 - val_acc: 0.9158
Epoch 30/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1945 - acc: 0.9414 - val_loss: 0.3088 - val_acc: 0.9097
```

Out[23]:

```
<keras.callbacks.History at 0x29b5ee36a20>
```

In [24]:

```
# Confusion Matrix
print(confusion_matrix(Y_test, model.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 512        0        25        0                   0
SITTING                  3      410        75        0                   0
STANDING                 0       87       445        0                   0
WALKING                  0        0         0      481                   2
WALKING_DOWNSTAIRS       0        0         0        0                 382
WALKING_UPSTAIRS         0        0         0        2                  18

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            3
STANDING                           0
WALKING                           13
WALKING_DOWNSTAIRS                38
WALKING_UPSTAIRS                 451
```

In [27]:

```
score = model.evaluate(X_test, Y_test)
```

```
2947/2947 [==============================] - 4s 2ms/step
```

In [28]:

```
score
```

Out[28]:

```
[0.3087582236972612, 0.9097387173396675]
```

- **With a simple 2 layer architecture we got 90.09% accuracy and a loss of 0.30**
- **We can further imporve the performace with Hyperparameter tuning**

In [35]:

```
# We are taking the same 2 layer architecture as above
# with softmax activation function in the dense layer
model = Sequential()
model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='softmax'))
```

In [36]:

```
# compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
```

```
              metrics=['accuracy'])
```

In [ ]:

```
epochs1 = 35
```

In [37]:

```
# Training the model
history = model.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/35
7352/7352 [==============================] - 96s 13ms/step - loss: 1.2585 - acc: 0.4687 -
val_loss: 1.0608 - val_acc: 0.5650
Epoch 2/35
7352/7352 [==============================] - 91s 12ms/step - loss: 0.9040 - acc: 0.6114 -
val_loss: 0.9134 - val_acc: 0.6474
Epoch 3/35
7352/7352 [==============================] - 93s 13ms/step - loss: 0.8370 - acc: 0.6730 -
val_loss: 0.7006 - val_acc: 0.7472
Epoch 4/35
7352/7352 [==============================] - 96s 13ms/step - loss: 0.6101 - acc: 0.7666 -
val_loss: 0.5642 - val_acc: 0.7737
Epoch 5/35
7352/7352 [==============================] - 97s 13ms/step - loss: 0.5956 - acc: 0.7610 -
val_loss: 0.5059 - val_acc: 0.8103
Epoch 6/35
7352/7352 [==============================] - 97s 13ms/step - loss: 0.6400 - acc: 0.7669 -
val_loss: 0.5601 - val_acc: 0.7981
Epoch 7/35
7352/7352 [==============================] - 95s 13ms/step - loss: 0.4527 - acc: 0.8599 -
val_loss: 0.4037 - val_acc: 0.8700
Epoch 8/35
7352/7352 [==============================] - 96s 13ms/step - loss: 0.3668 - acc: 0.8945 -
val_loss: 0.3414 - val_acc: 0.8809
Epoch 9/35
7352/7352 [==============================] - 94s 13ms/step - loss: 0.3080 - acc: 0.9120 -
val_loss: 0.3403 - val_acc: 0.8775
Epoch 10/35
7352/7352 [==============================] - 95s 13ms/step - loss: 0.2858 - acc: 0.9067 -
val_loss: 0.3419 - val_acc: 0.8982
Epoch 11/35
7352/7352 [==============================] - 95s 13ms/step - loss: 0.3548 - acc: 0.8913 -
val_loss: 0.4764 - val_acc: 0.8663
Epoch 12/35
7352/7352 [==============================] - 94s 13ms/step - loss: 0.2430 - acc: 0.9238 -
val_loss: 0.3241 - val_acc: 0.8901
Epoch 13/35
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1997 - acc: 0.9363 -
val_loss: 0.2781 - val_acc: 0.9033
Epoch 14/35
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1772 - acc: 0.9385 -
val_loss: 0.3028 - val_acc: 0.8955
Epoch 15/35
7352/7352 [==============================] - 95s 13ms/step - loss: 0.2516 - acc: 0.9286 -
val_loss: 0.2964 - val_acc: 0.8958
Epoch 16/35
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1994 - acc: 0.9384 -
val_loss: 0.3219 - val_acc: 0.8955
Epoch 17/35
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1837 - acc: 0.9372 -
val_loss: 0.2741 - val_acc: 0.9080
Epoch 18/35
7352/7352 [==============================] - 97s 13ms/step - loss: 0.2741 - acc: 0.8958 -
val_loss: 0.3114 - val_acc: 0.8897
Epoch 19/35
7352/7352 [==============================] - 96s 13ms/step - loss: 0.2023 - acc: 0.9212 -
```

```
val_loss: 0.3049 - val_acc: 0.8985
Epoch 20/35
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1586 - acc: 0.9433 -
val_loss: 0.3861 - val_acc: 0.8721
Epoch 21/35
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1599 - acc: 0.9430 -
val_loss: 0.5422 - val_acc: 0.8812
Epoch 22/35
7352/7352 [==============================] - 95s 13ms/step - loss: 0.2218 - acc: 0.9249 -
val_loss: 0.3455 - val_acc: 0.8962
Epoch 23/35
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1980 - acc: 0.9327 -
val_loss: 0.2768 - val_acc: 0.9040
Epoch 24/35
7352/7352 [==============================] - 93s 13ms/step - loss: 0.1812 - acc: 0.9374 -
val_loss: 0.2726 - val_acc: 0.9111
Epoch 25/35
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1492 - acc: 0.9445 -
val_loss: 0.3101 - val_acc: 0.9172
Epoch 26/35
7352/7352 [==============================] - 93s 13ms/step - loss: 0.1904 - acc: 0.9325 -
val_loss: 0.3113 - val_acc: 0.8999
Epoch 27/35
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1674 - acc: 0.9395 -
val_loss: 0.2649 - val_acc: 0.9030
Epoch 28/35
7352/7352 [==============================] - 93s 13ms/step - loss: 0.1523 - acc: 0.9446 -
val_loss: 0.2623 - val_acc: 0.9138
Epoch 29/35
7352/7352 [==============================] - 92s 12ms/step - loss: 0.1603 - acc: 0.9429 -
val_loss: 0.3490 - val_acc: 0.9074
Epoch 30/35
7352/7352 [==============================] - 93s 13ms/step - loss: 0.1475 - acc: 0.9459 -
val_loss: 0.3415 - val_acc: 0.9063
Epoch 31/35
7352/7352 [==============================] - 93s 13ms/step - loss: 0.2168 - acc: 0.9263 -
val_loss: 0.3463 - val_acc: 0.8867
Epoch 32/35
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1577 - acc: 0.9419 -
val_loss: 0.3425 - val_acc: 0.8962
Epoch 33/35
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1495 - acc: 0.9470 -
val_loss: 0.3356 - val_acc: 0.8989
Epoch 34/35
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1414 - acc: 0.9479 -
val_loss: 0.3048 - val_acc: 0.8982
Epoch 35/35
7352/7352 [==============================] - 92s 13ms/step - loss: 0.1289 - acc: 0.9475 -
val_loss: 0.3016 - val_acc: 0.9063
```

- **With a simple 2 layer architecture and sigmoid activation we got 90.06% accuracy and a loss of 0.30**
- **We can further imporve the performace with Hyperparameter tuning**
- **Peformance did not vary much when we replace the sigmoid activation function with softmax**

In [38]:

```python
%matplotlib notebook
import matplotlib.pyplot as plt
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

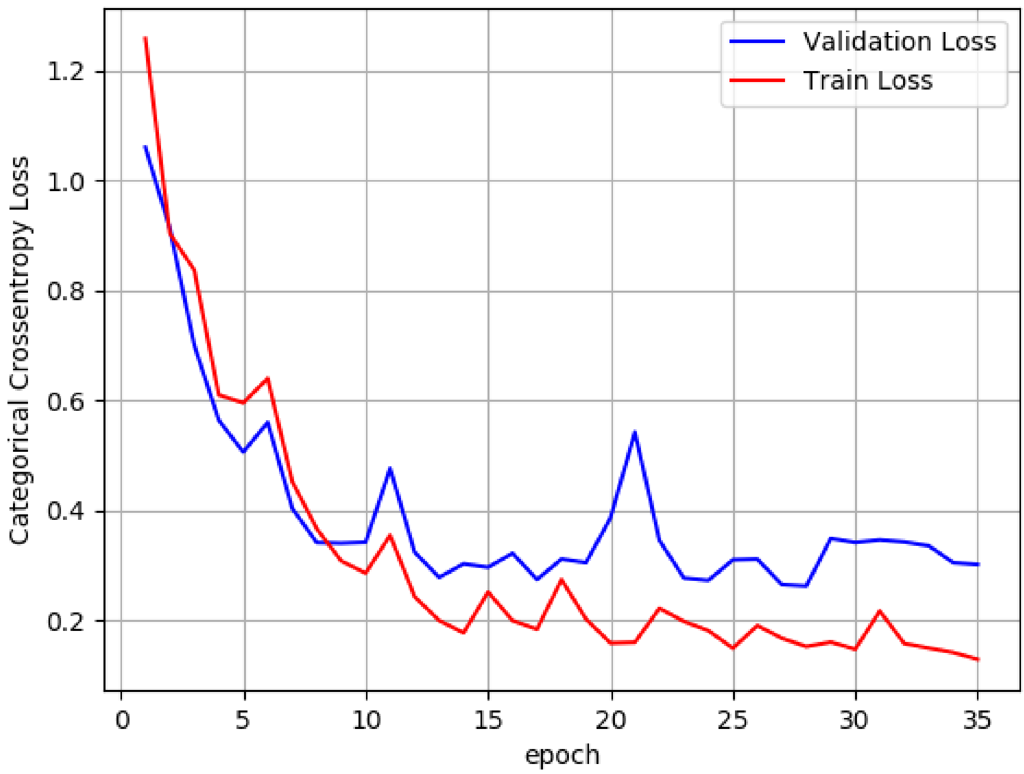
In [40]:

In [40]:

```
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs1+1))


vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```



In [41]:

```
model.evaluate(X_test, Y_test)
```

```
2947/2947 [==============================] - 4s 1ms/step
```

Out[41]:

```
[0.3016489237536808, 0.9063454360366474]
```

- **If we see the plot, as the number of epochs increases model gradually overfits**

Conclusions

- **I have tried mutiple architures and with different number of layers with varying dropouts and different number of activation using per cell.**
- **This simple architure with 2 layers and 32 activation cells per layer was giving the better performance of all.**
- **Further permance can still be increased with Hyperparameter tuning**

In [ ]:

```

```

In [ ]:

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```