Hyperopt-LSTM

September 28, 2019

0.0.1 Motivation and assumptions

Using the following procedure, you can easily build and test LSTM Networks. Hyperparameters are set using Hyperopt Python module.

There are some assumptions to remember regarding this particular example:

- Data are contained in .csv files. Please note, that if you created the .csv with Excel, it can use ";" instead of "," as value delimiter. In this case replace all ";" with ",".
- A complete usage of these functions expects to create two folders where any number of .csv files can be placed. The first folder is used to train AND validate the algorithm. The second folder is used to test the accuracy of our model.
- The first row of all .csv files must be the column names. Each column corresponds to an input variable. Each row (from the 2nd on) is a set of such input variables values.
- It follows that all .csv files must have the same number of columns, but can have a different row number.
- By putting different .csv files in the same folder, you let the program merge them as they would be a "single" bigger .csv file.

0.0.2 Libraries

[15]:
```
import glob
import os
import numpy as np
import pandas as pd
from datetime import datetime
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from hyperopt import fmin, hp, tpe, STATUS_OK, space_eval, Trials
from keras import backend, optimizers
import pickle
```

0.0.3 Define your architecture

In this example we are going to optimize a feed forward neural network. It's a regression problem.

Settings

[16]:
```
training_folder = 'train' # we store the .csv training data
# in the "train" folder. All .csv will be merged together into a single
# dataset
```

Hyperparameters

[17]:
```
hyper_space = {
    'lstm_units_1': hp.choice('lstm_units_1', np.arange(0, 30, 1)),
    'lstm_units_2': hp.choice('lstm_units_2', np.arange(0, 30, 1)),
    'lstm_units_3': hp.choice('lstm_units_3', np.arange(0, 30, 1)),
    'lstm_units_4': hp.choice('lstm_units_4', np.arange(0, 30, 1)),
    'dense_units_1': hp.choice('dense_units_1', np.arange(0, 8, 1)),
    'dense_units_2': hp.choice('dense_units_2', np.arange(0, 8, 1)),
    'dense_units_3': hp.choice('dense_units_3', np.arange(0, 8, 1)),
    'dense_units_4': hp.choice('dense_units_4', np.arange(0, 8, 1)),
    'dense_units_5': hp.choice('dense_units_5', np.arange(0, 8, 1)),
    'lstm_dropout_1' : hp.uniform('lstm_dropout_1', 0, 0.9),
    'lstm_dropout_2' : hp.uniform('lstm_dropout_2', 0, 0.9),
    'lstm_dropout_3' : hp.uniform('lstm_dropout_3', 0, 0.9),
    'lstm_dropout_4' : hp.uniform('lstm_dropout_4', 0, 0.9),
    'dense_dropout_1' : hp.uniform('dense_dropout_1', 0, 0.9),
    'dense_dropout_2' : hp.uniform('dense_dropout_2', 0, 0.9),
    'dense_dropout_3' : hp.uniform('dense_dropout_3', 0, 0.9),
    'dense_dropout_4' : hp.uniform('dense_dropout_4', 0, 0.9),
    'rec_dropout_1': hp.uniform('rec_dropout_1', 0, 0.9),
    'rec_dropout_2': hp.uniform('rec_dropout_2', 0, 0.9),
    'rec_dropout_3': hp.uniform('rec_dropout_3', 0, 0.9),
    'rec_dropout_4': hp.uniform('rec_dropout_4', 0, 0.9),
    'batch_size': hp.choice('batch_size', np.arange(1,66,16)),
    'timesteps': hp.choice('timesteps', np.arange(1,10,1))
}
```

Data preparation

[18]:
```
def serie_shift(dataset, column_name, timesteps):
    data = pd.DataFrame()
    data[column_name] = dataset[column_name]
    for i in range(1, timesteps):
        data['%s+%d' % (column_name, i)] = data[column_name].shift(-1*i)
    data = data.dropna()
    return data

def convert_date(df, df_col, date_format = '%Y-%m-%d'):
    df[df_col] = df[df_col].apply(
        lambda el: int(datetime.strptime(el,date_format).timestamp()/86400))
    df[df_col] -= df[df_col].min() # shifting (to start from 0)
    return df
```

```
def get_data(folder):
    df_list = []
    for f in glob.glob(os.path.join(folder,'*.csv')):
        df_list.append(pd.read_csv(f))
    df = pd.concat(df_list)
    df = df.dropna() # we filter out rows with non valid values
    df = convert_date(df,"Date")
    df = df.astype(np.float64)
    return df

def normalize_data(df):
    for columnName in df:
        df[columnName] -= df[columnName].mean()
        df[columnName] /= df[columnName].std()
    return df

## OPTIONAL, not used here
def get_random_data(cols, samples, timesteps): # cols the columns lists
    randata = np.random.random((samples, timesteps))
    randata = pd.DataFrame(data=randata, index=np.arange(0,1000), columns=cols)
    return randata

# LSTM for BINARY CLASSIFICATION problem
def lstm_in_out(df, output_column_number, timesteps):
    # output_column_number starts from 0 (-> first column)
    tmp = []
    for columnName in df:
        tmp.append(serie_shift(df, columnName, timesteps+1).to_numpy())
    tmp = tuple(tmp)
    X = np.dstack(tmp)
    Y = X[:,-1,1:2] # timestep+1 assigned to Y
    X = np.delete(X, -1, axis=1) # timestep+1 deleted from X
    diff = Y-X[:,-1,output_column_number:output_column_number+1]
    Y = np.where(diff >= 0, [1,0], [0,1])
    # [1,0] => next number is higher or equal
    # [0,1] => next number is lower
    return X,Y
```

[19]:
```
df = get_data(training_folder)
```

Model Function

[20]:
```
def train_hyper_model(dataFrame, hyper_params):
    print(hyper_params)
    X,Y = lstm_in_out(dataFrame, 1,hyper_params['timesteps'])
    print(X.shape[2])
    model = Sequential()
```

```
model.add(LSTM(
    units=hyper_params['lstm_units_1'],
    activation='selu',
    recurrent_dropout=hyper_params['rec_dropout_1'],
    return_sequences = True,
    input_shape=(hyper_params['timesteps'], X.shape[2])
))

model.add(Dropout(hyper_params['lstm_dropout_1']))

model.add(LSTM(
    units=hyper_params['lstm_units_2'],
    recurrent_dropout=hyper_params['rec_dropout_2'],
    return_sequences = True,
    activation='selu',
))

model.add(Dropout(hyper_params['lstm_dropout_2']))


model.add(LSTM(
    units=hyper_params['lstm_units_3'],
    recurrent_dropout=hyper_params['rec_dropout_3'],
    return_sequences = True,
    activation='selu',
))

model.add(Dropout(hyper_params['lstm_dropout_3']))


model.add(LSTM(
    units=hyper_params['lstm_units_4'],
    recurrent_dropout=hyper_params['rec_dropout_4'],
    return_sequences = False,
    activation='selu',
))

model.add(Dropout(hyper_params['lstm_dropout_4']))

model.add(Dense(units=hyper_params['dense_units_1'], activation='selu'))

model.add(Dropout(hyper_params['dense_dropout_1']))

model.add(Dense(units=hyper_params['dense_units_2'], activation='selu'))

model.add(Dropout(hyper_params['dense_dropout_2']))
```

```
    model.add(Dense(units=hyper_params['dense_units_4'], activation='selu'))

    model.add(Dropout(hyper_params['dense_dropout_3']))

    model.add(Dense(units=hyper_params['dense_units_5'], activation='selu'))

    model.add(Dense(units=2,activation="softmax"))

    model.
↪compile(optimizer="adam",loss="binary_crossentropy",metrics=["accuracy"])
    history = model.fit(
        X,
        Y,
        batch_size=hyper_params['batch_size'],
        validation_split=0.2,
        epochs = 20,
        shuffle = True,
        verbose=0)

    # take the last 8 accuracy values, and return their mean value:
    return np.mean(history.history['val_acc'][-8:])
```

Objective function

[21]:
```
def hyperopt_fn(hyper_params):
    accuracy = train_hyper_model(df, hyper_params) # X,Y are globally defined!
    backend.clear_session() # clear session to avoid models accumulation in␣
↪memory
    return {'loss': -accuracy, 'status': STATUS_OK}
```

Note: The STATUS_OK value is very important to avoid numerical errors problems produced by some particular set of (hyper)parameters values.

0.0.4 Let's optimize!

Thanks to Trials, which stores and track the progress, you have the possibility to execute a new optimization process, but starting from previous ones.

[22]:
```
keep_trials = Trials()

# we can also load trials from file using prickle:
f = open('store_trials_LSTM.pckl', 'rb')
keep_trials = pickle.load(f)
f.close()
```

By setting the option trials = keep_trials, if you run again the same cell it will not compute any furter iteration, since it consider the previous ones as completed. For example, if you have done 10 iterations, than you change the iterations to 30 (max_evals = 30) and run the cell again,

the optimization will perform 20 iteration (from 11 to 20!). If you want to reset the iteration after each code execution, just move the trials parameter.

[23]:
```
%%time
while True:
    try:
        opt_params = fmin(
                        fn=hyperopt_fn,
                        space=hyper_space,
                        algo=tpe.suggest,
                        max_evals=18, # stop searching after 18 iterations
                        trials = keep_trials
                        )

        # store trials in a file
        f = open('store_trials_LSTM.pckl', 'wb')
        pickle.dump(keep_trials, f)
        f.close()

        print(space_eval(hyper_space, opt_params))
        print('number of trials:', len(keep_trials.trials))
        break
    except:
        continue
```

```
{'batch_size': 49, 'dense_dropout_1': 0.48699661421709556, 'dense_dropout_2':
0.6928242445573329, 'dense_dropout_3': 0.2131704551687931, 'dense_dropout_4':
0.7760827974352934, 'dense_units_1': 0, 'dense_units_2': 0, 'dense_units_3': 2,
'dense_units_4': 2, 'dense_units_5': 0, 'lstm_dropout_1': 0.3766645626364943,
'lstm_dropout_2': 0.7146163328005891, 'lstm_dropout_3': 0.5217070799108651,
'lstm_dropout_4': 0.018811127823735032, 'lstm_units_1': 20, 'lstm_units_2': 9,
'lstm_units_3': 1, 'lstm_units_4': 20, 'rec_dropout_1': 0.7161561723657223,
'rec_dropout_2': 0.252321183897106, 'rec_dropout_3': 0.8237432438020351,
'rec_dropout_4': 0.8217842139064441, 'timesteps': 4}
7
  0%|
| 0/1 [00:00<?, ?it/s, best loss: ?]
{'batch_size': 49, 'dense_dropout_1': 0.0721473042600411, 'dense_dropout_2':
0.32602936225208196, 'dense_dropout_3': 0.5028806307061477, 'dense_dropout_4':
0.740980277010866, 'dense_units_1': 2, 'dense_units_2': 5, 'dense_units_3': 6,
'dense_units_4': 0, 'dense_units_5': 5, 'lstm_dropout_1': 0.8999097366144988,
'lstm_dropout_2': 0.39799735948073384, 'lstm_dropout_3': 0.779607093750467,
'lstm_dropout_4': 0.04984406125647999, 'lstm_units_1': 25, 'lstm_units_2': 21,
'lstm_units_3': 20, 'lstm_units_4': 9, 'rec_dropout_1': 0.349882518574385,
'rec_dropout_2': 0.1876240421641633, 'rec_dropout_3': 0.14322208180231855,
'rec_dropout_4': 0.30162985694882194, 'timesteps': 3}
7
  0%|
```

```
| 0/1 [00:00<?, ?it/s, best loss: ?]
{'batch_size': 49, 'dense_dropout_1': 0.6661327424956357, 'dense_dropout_2':
0.3383175020939377, 'dense_dropout_3': 0.6480838911410937, 'dense_dropout_4':
0.813727398921859, 'dense_units_1': 5, 'dense_units_2': 2, 'dense_units_3': 3,
'dense_units_4': 7, 'dense_units_5': 0, 'lstm_dropout_1': 0.00417820346060116,
'lstm_dropout_2': 0.01857171807420326, 'lstm_dropout_3': 0.603529079344333,
'lstm_dropout_4': 0.038679169420591956, 'lstm_units_1': 14, 'lstm_units_2': 3,
'lstm_units_3': 22, 'lstm_units_4': 12, 'rec_dropout_1': 0.4872674171416302,
'rec_dropout_2': 0.2159069060891396, 'rec_dropout_3': 0.4377790627176319,
'rec_dropout_4': 0.7030066749920194, 'timesteps': 1}
7
  0%|
| 0/1 [00:00<?, ?it/s, best loss: ?]
{'batch_size': 49, 'dense_dropout_1': 0.6239571715125253, 'dense_dropout_2':
0.623955686328828, 'dense_dropout_3': 0.656322633929649, 'dense_dropout_4':
0.21072461139098855, 'dense_units_1': 0, 'dense_units_2': 1, 'dense_units_3': 5,
'dense_units_4': 1, 'dense_units_5': 2, 'lstm_dropout_1': 0.656480292996167,
'lstm_dropout_2': 0.21862505804232726, 'lstm_dropout_3': 0.8954691790979349,
'lstm_dropout_4': 0.5716926440683059, 'lstm_units_1': 15, 'lstm_units_2': 8,
'lstm_units_3': 23, 'lstm_units_4': 20, 'rec_dropout_1': 0.4657769849391857,
'rec_dropout_2': 0.1817153192911972, 'rec_dropout_3': 0.7563983640665052,
'rec_dropout_4': 0.255022820766406, 'timesteps': 4}
7
  0%|
| 0/1 [00:00<?, ?it/s, best loss: ?]
{'batch_size': 49, 'dense_dropout_1': 0.4518490476937244, 'dense_dropout_2':
0.40472315532047326, 'dense_dropout_3': 0.11717092123147403, 'dense_dropout_4':
0.31465370133380177, 'dense_units_1': 0, 'dense_units_2': 2, 'dense_units_3': 0,
'dense_units_4': 3, 'dense_units_5': 6, 'lstm_dropout_1': 0.21671778425309915,
'lstm_dropout_2': 0.6392227047093507, 'lstm_dropout_3': 0.4887208537175433,
'lstm_dropout_4': 0.7264241041910109, 'lstm_units_1': 25, 'lstm_units_2': 1,
'lstm_units_3': 10, 'lstm_units_4': 28, 'rec_dropout_1': 0.7277937631148911,
'rec_dropout_2': 0.17770489667273698, 'rec_dropout_3': 0.253745068298071,
'rec_dropout_4': 0.42171928408981174, 'timesteps': 3}
7
  0%|
| 0/1 [00:00<?, ?it/s, best loss: ?]
{'batch_size': 65, 'dense_dropout_1': 0.03599926217877152, 'dense_dropout_2':
0.7095489657248593, 'dense_dropout_3': 0.3874762400964262, 'dense_dropout_4':
0.6466706369644084, 'dense_units_1': 7, 'dense_units_2': 7, 'dense_units_3': 7,
'dense_units_4': 3, 'dense_units_5': 4, 'lstm_dropout_1': 0.39800230390586894,
'lstm_dropout_2': 0.4297913613464064, 'lstm_dropout_3': 0.4860874301171336,
'lstm_dropout_4': 0.6554287879045518, 'lstm_units_1': 7, 'lstm_units_2': 6,
'lstm_units_3': 18, 'lstm_units_4': 14, 'rec_dropout_1': 0.34012392735356045,
'rec_dropout_2': 0.5597054480111028, 'rec_dropout_3': 0.6417811769370548,
'rec_dropout_4': 0.3406868371326607, 'timesteps': 4}
7
100%|| 1/1 [01:42<00:00,
```

```
102.94s/it, best loss: -0.5577242621328942]
{'batch_size': 49, 'dense_dropout_1': 0.06460775478555035, 'dense_dropout_2':
0.007243995311981222, 'dense_dropout_3': 0.042320383674304095,
'dense_dropout_4': 0.6528945560223345, 'dense_units_1': 7, 'dense_units_2': 4,
'dense_units_3': 0, 'dense_units_4': 5, 'dense_units_5': 5, 'lstm_dropout_1':
0.08736910244735684, 'lstm_dropout_2': 0.5092850351473164, 'lstm_dropout_3':
0.4489553003035543, 'lstm_dropout_4': 0.48984868329938447, 'lstm_units_1': 23,
'lstm_units_2': 11, 'lstm_units_3': 18, 'lstm_units_4': 7, 'rec_dropout_1':
0.49394303437575504, 'rec_dropout_2': 0.5564609595913076, 'rec_dropout_3':
0.5715505497132145, 'rec_dropout_4': 0.3807587355481572, 'timesteps': 9}
number of trials: 18
Wall time: 2min 8s
```

[24]:
```
print(keep_trials.trials[-1]['misc']['vals']) # print last hyperparameters␣
↪value
```

```
{'batch_size': [4], 'dense_dropout_1': [0.03599926217877152], 'dense_dropout_2':
[0.7095489657248593], 'dense_dropout_3': [0.3874762400964262],
'dense_dropout_4': [0.6466706369644084], 'dense_units_1': [7], 'dense_units_2':
[7], 'dense_units_3': [7], 'dense_units_4': [3], 'dense_units_5': [4],
'lstm_dropout_1': [0.39800230390586894], 'lstm_dropout_2': [0.4297913613464064],
'lstm_dropout_3': [0.4860874301171336], 'lstm_dropout_4': [0.6554287879045518],
'lstm_units_1': [7], 'lstm_units_2': [6], 'lstm_units_3': [18], 'lstm_units_4':
[14], 'rec_dropout_1': [0.34012392735356045], 'rec_dropout_2':
[0.5597054480111028], 'rec_dropout_3': [0.6417811769370548], 'rec_dropout_4':
[0.3406868371326607], 'timesteps': [3]}
```