Hyperopt_FFNN

September 24, 2019

0.0.1 Motivation and assumptions

Using the following procedure, you can easily build and test Feed Forward Neural 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

[2]:
```
import glob
import os
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import 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

[3]:
```
training_folder = 'train' # we store the .csv training data
# in the "train" folder. All .csv will be merged together into a single
# dataset
```

Hyperparameters The model objective function describes our model and needs to be optimize by tuning its hyperparameters. The hyperparameters are here: - number of hidden layers - units of hidden layers - dropout - loss function - batch size

We do not consider the learning rate because we are going to use the Adam Optimizer. Such parameters are divided into two categories: continuous and discrete values. As activation function I've chosen the selu function.

[4]:
```
hyper_space = {
    'nr_hlayers': hp.choice('nr_hlayers', np.arange(0, 6, 1)), # integers from␣
 ↪0 to 5
    'layer_units' : hp.choice('layer_units', np.arange(1, 6, 1)), # integers␣
 ↪from 1 to 5
    'dropout' :  hp.uniform('dropout', 0, 0.9),
    'loss_function': hp.choice('loss_function', ['mean_squared_error',
                               'mean_absolute_error',
                               'mean_absolute_percentage_error',
                               'mean_squared_logarithmic_error'
                              ]),
    'batch_size': hp.choice('batch_size', np.arange(1,66,16))
}
```

Data preparation

[5]:
```
# get data from comma separated values files in the specified folder
# all files are merged together in a unique dataset
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.astype(np.float64)
    df = df.dropna() # we filter out rows with non valid values
    return df

# prepare the inputs and outputs for our ffnn model
def feed_in_out(df,output_column_number):
    XY = df.to_numpy()
    Y = XY[:,output_column_number]
    X = np.delete(XY,output_column_number, axis=1)
    return X,Y
```

[6]:
```
df = get_data(training_folder)
X,Y = feed_in_out(df,2) # the third column is considered as output
```

Model Function

[7]:
```python
def train_hyper_model(X,Y,hyper_params):
    model = Sequential() # We sequentially add layers
    model.add(Dense(units=X.shape[1]+1, input_dim=X.shape[1],␣
 ↪activation='selu')) # input layer

    # hidden layers
    for h in np.arange(0, hyper_params['nr_hlayers']):
        model.add(Dense(units=hyper_params['layer_units'], activation='selu'))
        model.add(Dropout(hyper_params['dropout']))

    model.add(Dense(units=1)) # output layer with linear activation (default)
    model.
 ↪compile(optimizer="adam",loss=hyper_params['loss_function'],metrics=["mean_squared_error"])
    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 validation losses, and return their mean value:
    return np.mean(history.history['val_mean_squared_error'][-8:])
```

Objective function We could directly use the Model function to optimize, but it's (in general, maybe not in this case) more modular to embed it in the final objective function which will be optimized. In this way you can define more functions with different outputs and sequentially embed them in the objective function.

[8]:
```python
def hyperopt_fn(hyper_params):
    loss = train_hyper_model(X, Y, hyper_params) # X,Y are globally defined!
    backend.clear_session() # clear session to avoid models accumulation in␣
 ↪memory
    return {'loss': loss, '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.

[10]:
```python
keep_trials = Trials()

# we can also load trials from file using prickle:
f = open('store_trials.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.

[11]:
```
%%time
opt_params = fmin(
                fn=hyperopt_fn,
                space=hyper_space,
                algo=tpe.suggest,
                max_evals=75, # stop searching after 50 iterations
                trials = keep_trials
                )

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

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

```
  0%|
| 0/5 [00:00<?, ?it/s, best loss: ?]WARNING:tensorflow:From
C:\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:74: The name
tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph
instead.

WARNING:tensorflow:From C:\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:517: The name tf.placeholder is
deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From C:\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:4138: The name tf.random_uniform is
deprecated. Please use tf.random.uniform instead.

WARNING:tensorflow:From C:\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:3217:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From C:\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:133: The name
tf.placeholder_with_default is deprecated. Please use
tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From C:\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
WARNING:tensorflow:From C:\Anaconda3\lib\site-packages\keras\optimizers.py:790:
The name tf.train.Optimizer is deprecated. Please use
tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From C:\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:986: The name tf.assign_add is
deprecated. Please use tf.compat.v1.assign_add instead.

100%|| 5/5 [16:38<00:00,
198.27s/it, best loss: 1405.3673982174782]
{'batch_size': 1, 'dropout': 0.0025297289561878322, 'layer_units': 1,
'loss_function': 'mean_absolute_error', 'nr_hlayers': 4}
number of trials: 75
Wall time: 16min 38s
```