Deep Learning - lab 5

Hyperopt and RNNs

Prof. Stefano Carrazza

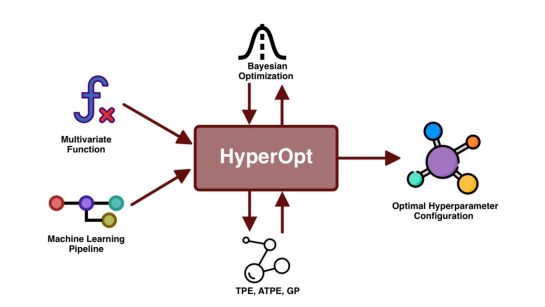
University of Milan and INFN Milan

Hyperparametrization

Hyperopt

Hyperopt is a simple python package which provides:

- Random Search
- Tree of Parzen Estimators (TPE)
- Adaptive TPE



`https://github.com/hyperopt`

Hyperopt expressions

```
from hyperopt import hp, pyll

# define search space
a = hp.uniform('a', -10, 10)
b = hp.choice('b', [1, 2, 3, 4])
c = hp.loguniform('c', -5, 0)
# randint, normal, lognormal, ...

sample = pyll.stochastic.sample(a) # generate 1 sample
```

Hyperopt the trials object

```
import time
from hyperopt import fmin, tpe, rand, hp, STATUS_OK, Trials

def objective(x):
  return {'loss': x ** 2, 'status': STATUS_OK, 'eval_time': time.time()}

algorithm = tpe.suggest # or rand.suggest
trials = Trials() # objecting collecting sequential trials
best = fmin(objective, space=hp.uniform('x', -10, 10),
            algo=algorithm, max_evals=100,
            trials=trials)

print(best) # returns dict with {'x': value}
iterations = trials.idxs_vals[0]['x']
x_values = trials.idxs_vals[1]['x']
```

Hyperopt the trials object

```
from hyperopt import fmin, tpe, hp, Trials, space_eval, STATUS_OK

search_space = {
      'layer_size': hp.choice('layer_size', np.arange(10, 100, 20)),
      'learning_rate': hp.loguniform('learning_rate', -10, 0)
}

trials = Trials()
best = fmin(hyper_func, search_space, algo=tpe.suggest,
            max_evals=5, trials=trials)

space_eval(search_space, best) # translates dict to real search space
```

Trials

```
trials = Trials()

# Extracting history
iterations = [ t['tid'] for t in trials.trials]
losses = [ t['result']['loss'] for t in trials.trials]
learning_rates = [ t['misc']['vals']['learning_rate'] for t in trials.trials]
```

RNNs

Built-in RNN models

Keras provides 3 built-in RNN layers:

- `tensorflow.keras.layers.SimpleRNN`: fully-connected RNN.
- `tensorflow.keras.layers.LSTM`
- `tensorflow.keras.layers.GRU`

and there are three built-in RNN cells, matching the layers:

- `tensorflow.keras.layers.SimpleRNNCell`
- `tensorflow.keras.layers.GRUCell`
- `tensorflow.keras.layers.LSTMCell`

A simple LSTM regression

```
from tensorflow import keras

model = keras.Sequential()
# suppose
model.add(keras.layers.Input(shape=(32,1)))

# add a LSTM layer with 10 internal units
model.add(keras.layers.LSTM(10))

# add a dense layer with 1 unit
model.add(keras.layers.Dense(1))
```

A simple GRU regression

By default the output of a RNN layer is a single vector per sample.

```
from tensorflow import keras

model = keras.Sequential()
# suppose
model.add(keras.layers.Input(shape=(32,1)))

# add a GRU layer with 10 internal units
# returns (batch_size, timesteps, 10)
model.add(keras.layers.GRU(10, return_sequences=True))

# add a dense layer with 1 unit
model.add(keras.layers.Dense(1))
```

Encoder-decoder sequence-to-sequence

Using the functional API we can design more complex models:

```
encoder_input = keras.layers.Input(shape=(None,1))

# return states in addition to output
output, state_h, state_c = layers.LSTM(64, return_state=True)(encoder_input)
encoder_state = [state_h, state_c]

decoder_input = layers.Input(shape=(None,1))
decoder_output = layers.LSTM(64)(decoder_input, initial_state=encoder_state)

output = layers.Dense(10)(decoder_output)

model = keras.Model([encoder_input, decoder_input], output)
```

Bidirectional RNNs

```
model = keras.Sequential()

model.add(layers.Bidirectional(layers.LSTM(64, return_sequences=True),
                               input_shape=(5, 10)))

model.add(layers.Bidirectional(layers.LSTM(32)))
model.add(layers.Dense(10))
```