Machine Learning Framework – Cheat Sheet(s)

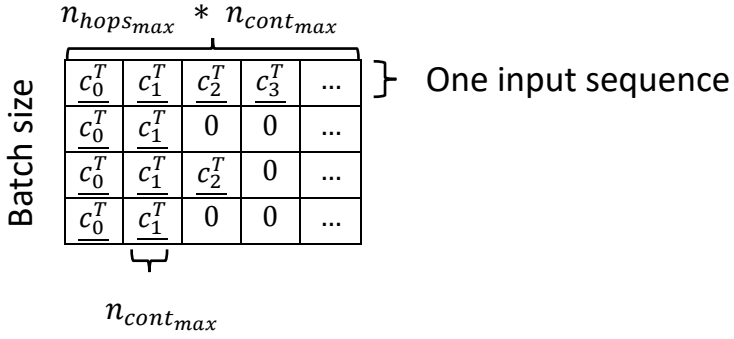
I. Pyhton IDE – Pycharm

Description	Commands
Start Pycharm	*cd /usr/local/labs/ML/* *./bin/ml pycharm &*
Open the project	Click on File → Open… Change directory to: */usr/local/labs/ML/ab12cde/eda_code* Click Ok
Open a Terminal in Pycharm	Click on View → Tool Windows → Terminal
Open thy Python Console in Pycharm	Click on View → Tool Windows → Python Console
Start Tensorboard	Run in Python Terminal: *tensorboard --logdir=path/to/log/directory*

II. Project Structure

a) Training Data

- Features:
  2d-array (batch size x concatenated input vectors) stored in *data/features.npy*
  $n_{cont_{max}}$ ≔ maximal number of contentions: 10
  $n_{hops_{max}}$ ≔ maximal number of hops: 25



- Labels:
  1d-array (batch size) stored in *data/labels.npy*
  Average latency in cycles (from SystemC simulation)

- Note:
  The first half of the data set was generated for lowly utilized NoCs while the other half was generated for highly utilized NoCs

b) Logs

- Output directory for all log-files generated during the training process including
  - Hyperparameters in *hyperparameters.txt*
  - Neural Network weights in *weights.h5*
  - Event files for Tensorboard in *event.out.tfevents…*

c) RNN models

- SimpleLSTM.py
  - Simple RNN model using LSTM cells
  - Used for Task 1 (at this point the parameter *hyperparameters* can be ignored)

- GenericRNN.py
  - Parameterizable RNN
  - Used for the hyperparameter optimization in Task2

d) Virtual Environment

- Isolated Python environment
- Stores ML libraries (e.g. Keras and tensorflow)
- Do not modify this directory (*venv/*)

e) Hyperparameter Optimization Class (used in Task2)

- Wrapper class for the scikit-optimize library
- Main Methods:
  - *__init__(self, dimensions, default_parameters)*
    dimensions: list of of all hyperparameters
    default_parameters: llist of default values for all parameters (same order as in dimensions)

  - *optimize(self, fitness_func, iterations, method)*
    fitness_func: black-box function (here GenericRNN.fitness)
    iterations: integer number of iterations (>11)
    method: 'rs' for random search and 'gp' for Gaussian process

f) Neural Network Class

- Abstract base class for all models in the "*models/*" directory
- Main Methods:
  - *fitness(cls, hyperparameters)*
    Trains a neural network for 5 epochs and returns the mean squared validation error

  - *split_data_set(cls, features, labels)*
    Splits data set into training, validation and test data

III. Recurrent Neural Networks in Keras

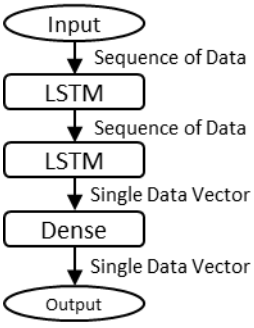
a) Masking Class

- Supports masking for input data of variable number of time steps
- Important parameters:

  - Mask_value: the value which should be masked
  - Input_shape: the shape of the input tensor

b) LSTM & GRU Class

- Implementation not suited for GPUs
- Important parameters
  - Units: number of LSTM cells per layer
  - Return_sequence: returns the full output sequence if enabled



c) CuDNNLSTM & CuDNNGRU Class

- GPU only implementation of LSTM and GRU
- Do not support masking layer (→ do not use these layers during the lab)

Further Information: https://keras.io/layers/recurrent/

IV. Task 1

a) Data pre-processing

- Load data set from the files „data/features.npy“ and „data/labels.npy“ in Task1.py
- Implement the *split_data_set* method in *NeuralNetworks.py* where 70% of the data set shall be used for training, 15% for validation and 15% test data.

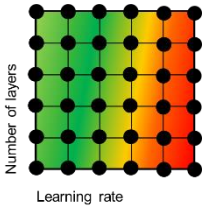
b) Train the RNN

- Implement a LSTM network in the models/SimpleRNN.py class
- Tune the hyperparameters to obtain a mean absolute percentage error (*loss ='mape'*) below 5%.

V. Hyperparameter Optimization Methods

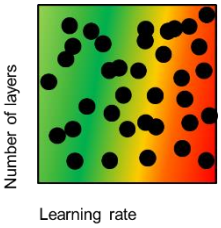
a) Grid Search

1. Space each of the hyperparameters evenly or logarithmically
2. Train the neural network for all combinations
3. Select the hyperparameters which yield the lowest validation error



b) Random Search

1. Select bounds for all hyperparameters
2. Train the neural network for N random samples
3. Select the hyperparameters which yield the lowest validation error



c) Bayesian Optimization

1. Choose an initial set of hyperparameters
2. Train the neural network with the hyperparameters
3. Update the Gaussian Model according to the validation error
4. Choose the hyperparameters which maximize the expected improvement
5. Goto 2. if termination criteria is not fullfilled
6. Select the hyperparameters which yield the lowest validation error

VI. Optimization Library – Optimization Library

a) Classes for the Definition of Hyperparameters

- Real e.g. for the learning rate
- Integer e.g. for the number of layers
- Categorical e.g. for the activation function

b) Examples

```
hp_learning_rate = Real(low=1e-6, high=1e-2, name='learning_rate')
hp_n_layers = Integer(low=1, high=5, name='n_layers')
hp_activation_function = Categorical(categories=['sigmoid', 'relu'],
                                     name='activation_function')
```

VII. Task 2

a) Hyperparameter Optimization

- Define a set of hyperparameters for your RNN using the classes Real, Integer and Categorical.
- Implement a parameterizable RNN in GenericRNN.py.

b) Random Search

- Run a random search for 15 Iterations.
- Plot the results using Tensorboard and find the best network.

c) Bayesian Optimization

- Run a Bayesian optimization for 15 iterations (change the log directory in NeuralNetwork.py).
- Plot the results using Tensorboard and find the best network.