Consiglio Nazionale delle Ricerche

Polimi - Financial Engineering AY 2023/2024

Lab: Electricity Price and Load Forecasting - L2

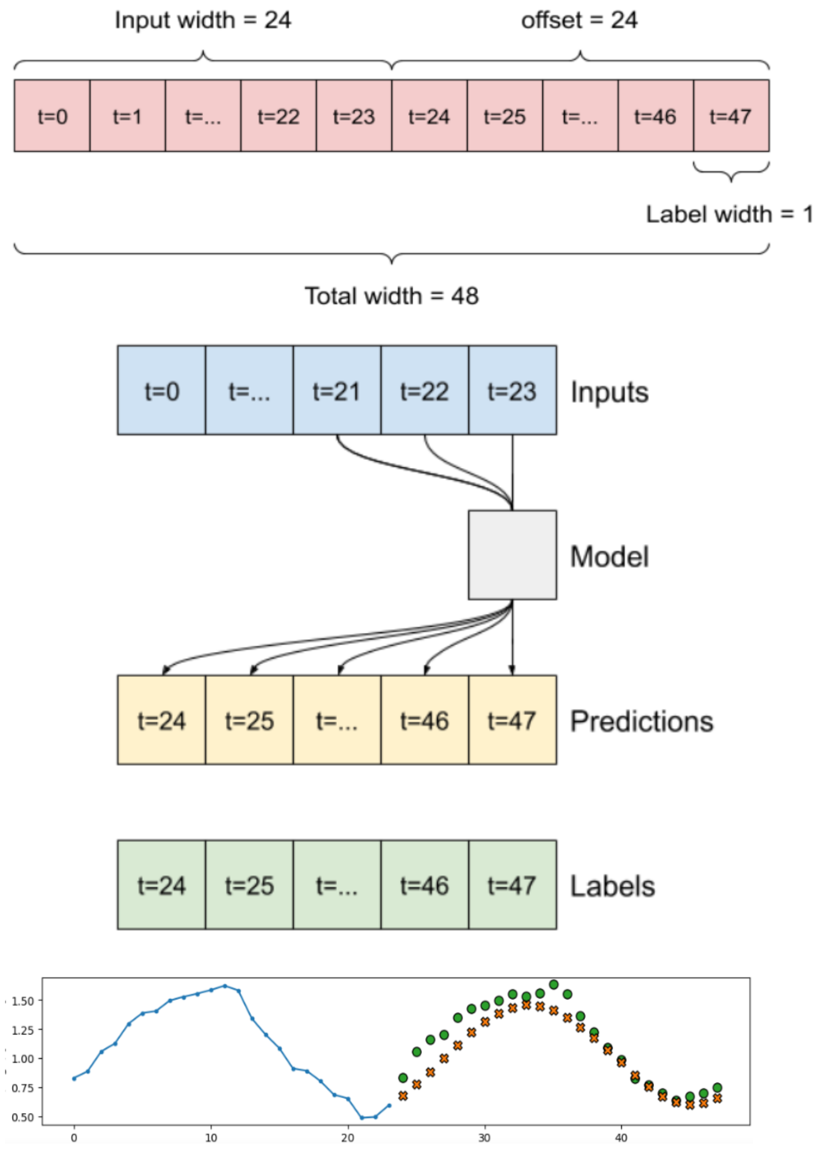
STIIMA
Alessandro Brusaferri

Summary

STIIMA

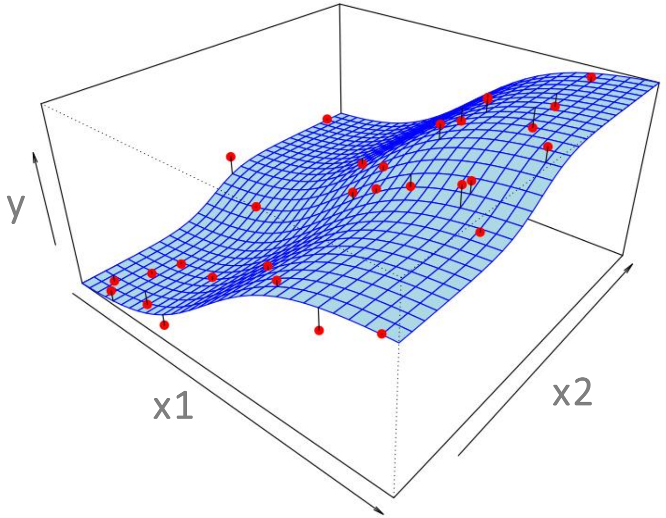
Goal L2:

- Recap lesson 1 / assignment discussion
- Intro to DNN (feedforward NN map)
- Intro to hyper-parameters search
- Hyper-params search implementation in Optuna
- DNN implementation in Tensorflow

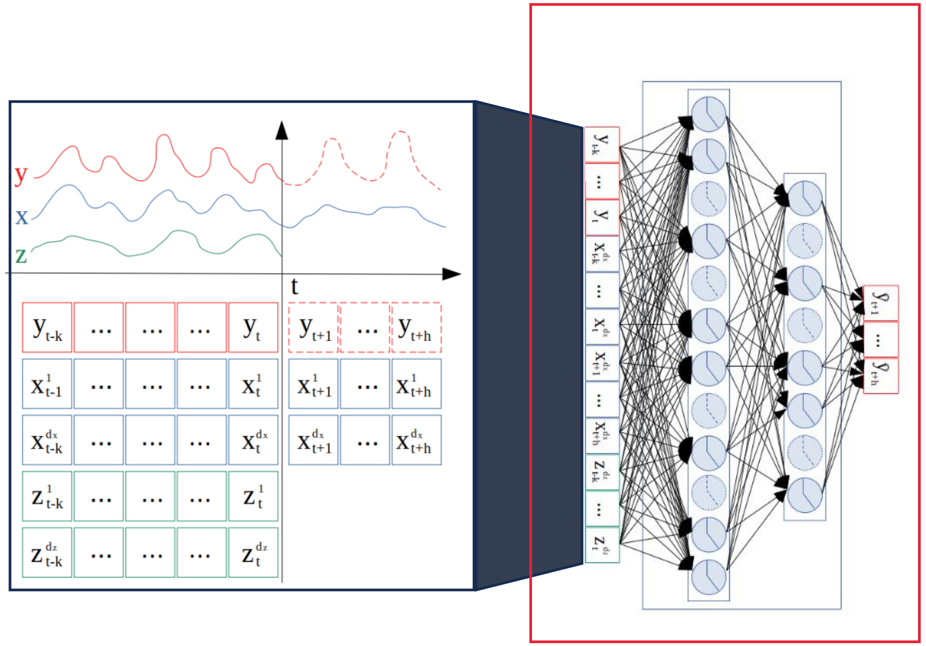


https://www.tensorflow.org/tutorials/structured_data/time_series

DNN – nonlinear feedforward map



- Increase **flexibility** wrt ARX linear regression
- **Nonlinear maps** identification from data
- Broad set of **architectures** in the literature (RNN, CNN,...)
- We focus on the simple **feedforward** maps (DNN)



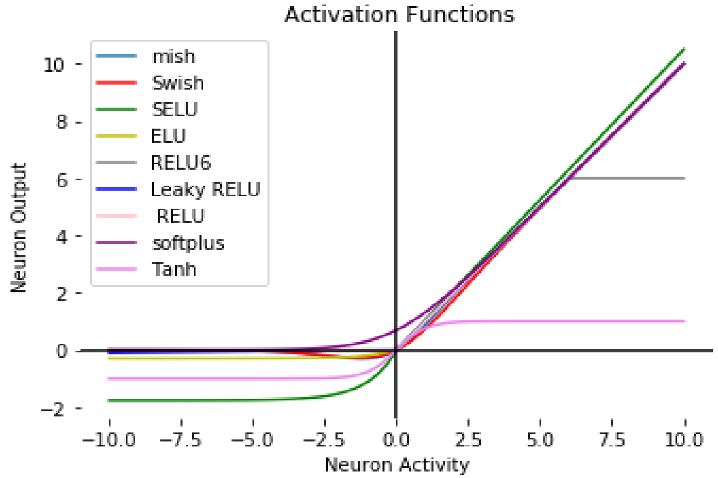
$$\ell_1 = \mathrm{g}(x_i W_1 + b_1)$$

$$\ell_2 = \mathrm{g}(\ell_1 W_2 + b_2) W_3 + b_3$$

$$W_1 \in \mathbb{R}^{n_x \times n_{u1}}, \; W_2 \in \mathbb{R}^{n_{u1} \times n_{u2}},$$

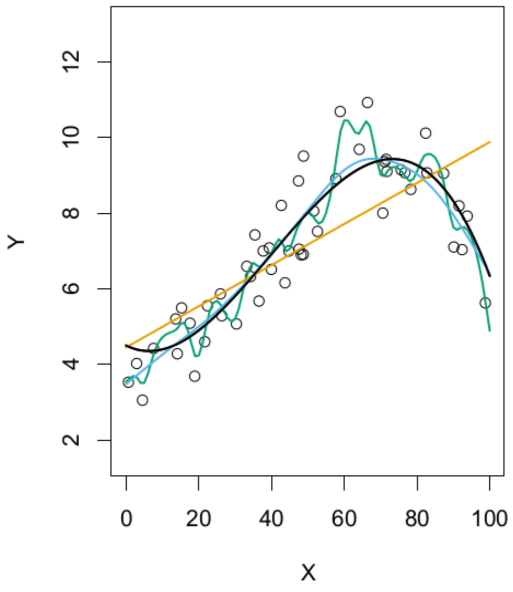
$$W_3 \in \mathbb{R}^{n_{u2} \times H \cdot n_p}, \; n_{u1}, n_{u2} \in \mathbb{Z}^+$$

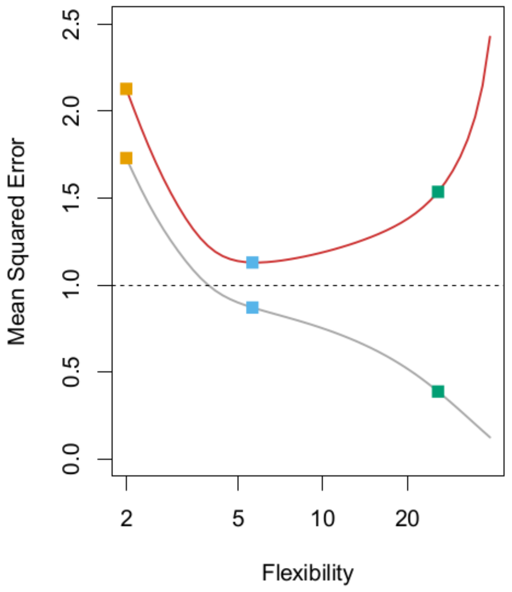
$$b_1 \in \mathbb{R}^{n_{u1}}, b_2 \in \mathbb{R}^{n_{u2}}, b_3 \in \mathbb{R}^{H \cdot n_p}$$



DNN – nonlinear feedforward map

- Flexible function **approximation**
- Parameters learning by **gradient** descent
- **Overfitting** and over-parameterization





Loss

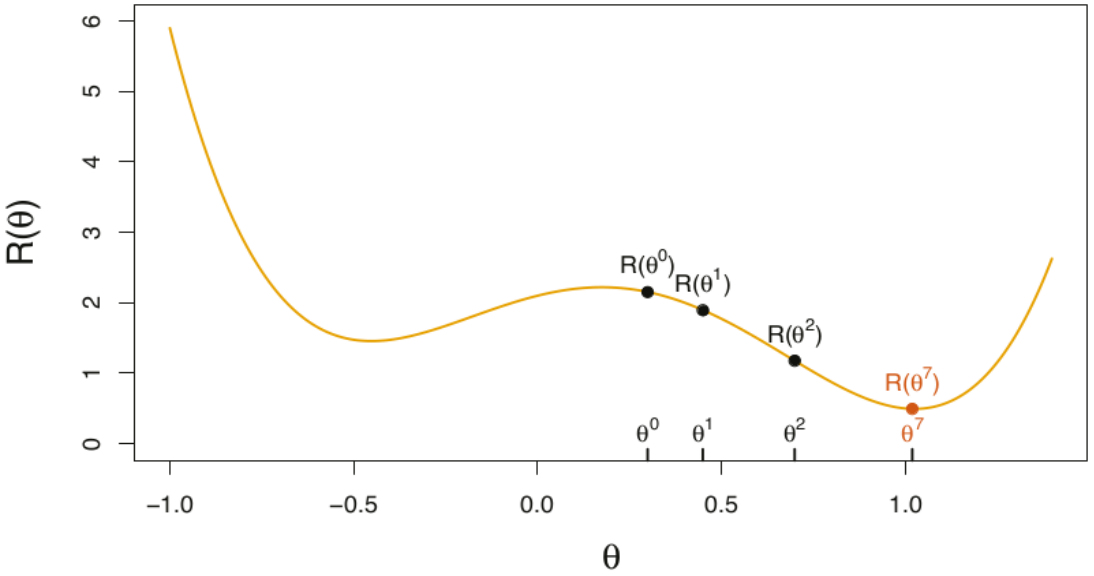
Function form

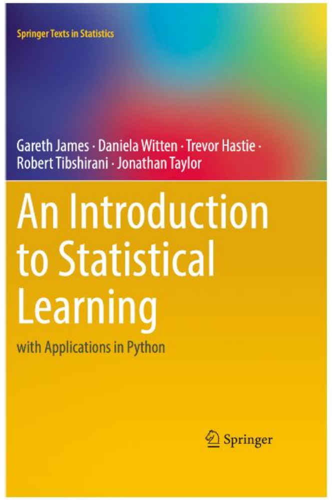
$$\boxed{R(\theta)} = \frac{1}{2}\sum_{i=1}^{n}(y_i - \boxed{f_\theta(x_i)})^2$$

Learning (grad)--> TF

$$\theta^{m+1} \leftarrow \theta^m - \rho \nabla R(\theta^m)$$

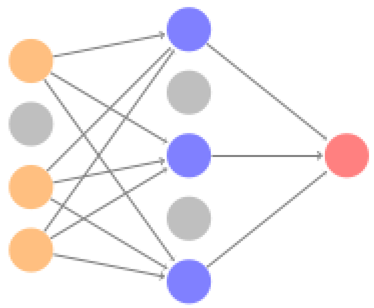
$$\nabla R(\theta^m) = \frac{\partial R(\theta)}{\partial \theta}\Big|_{\theta=\theta^m}$$

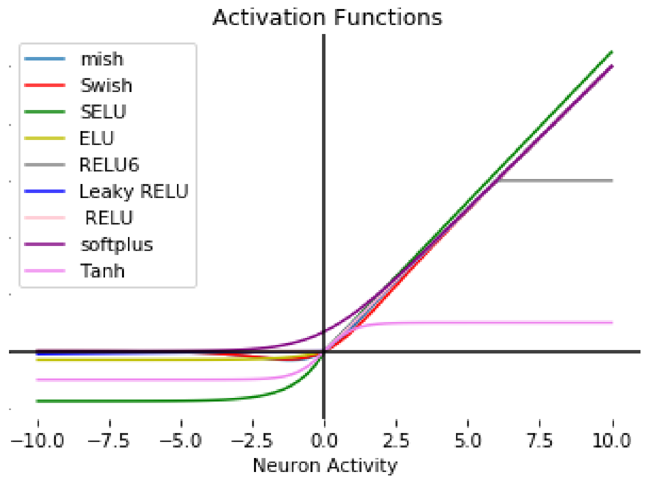


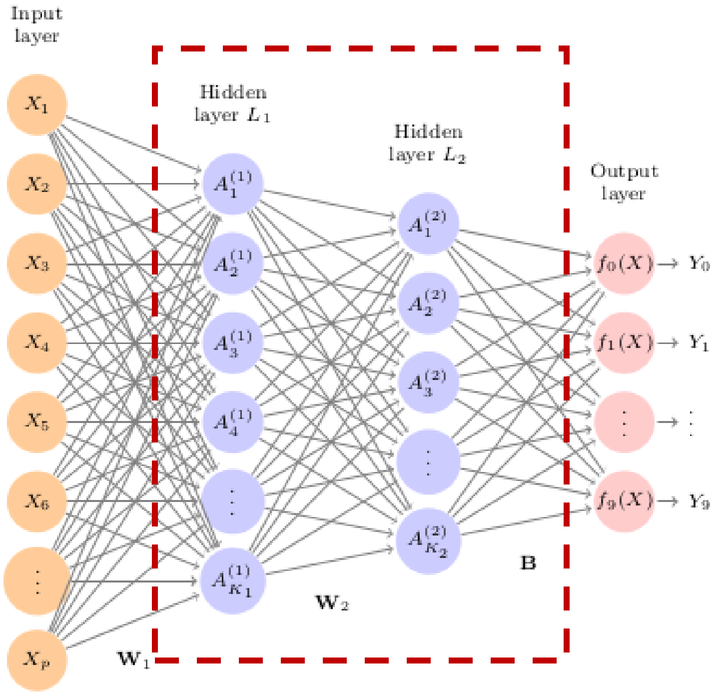


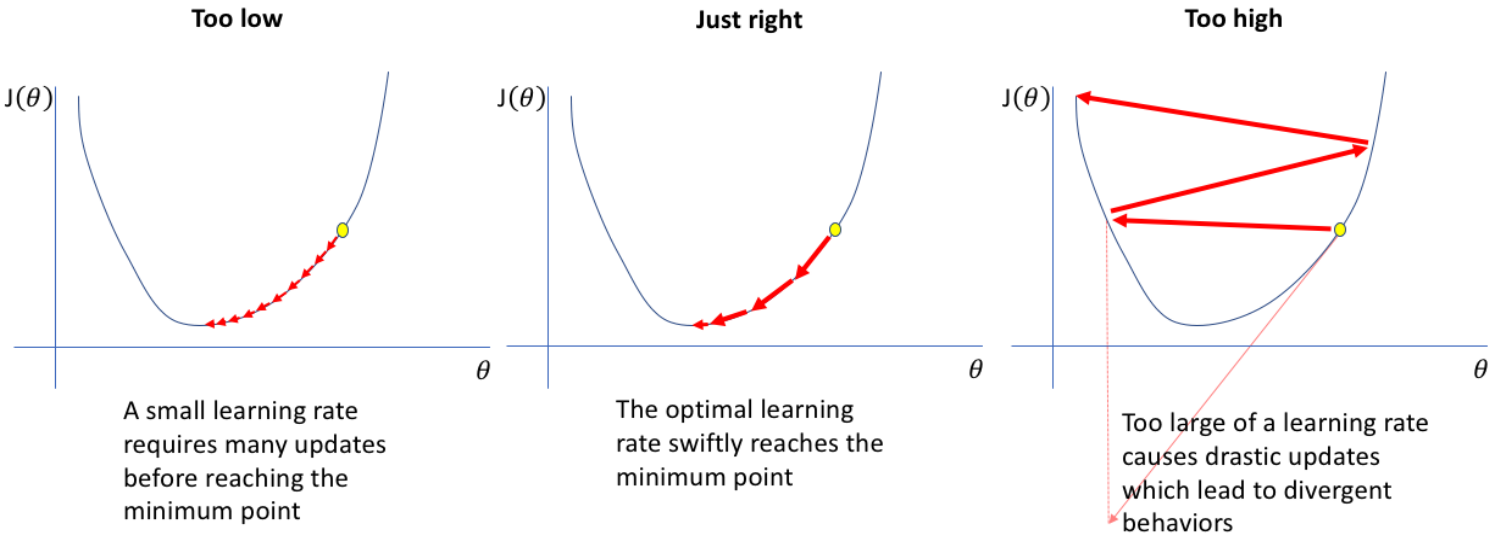
Hyper-parameters tuning

- Which **architecture**? How many **layers**?
- How many **neurons**? Which **activation**?
- Learning **rate**? **Regularizations**?





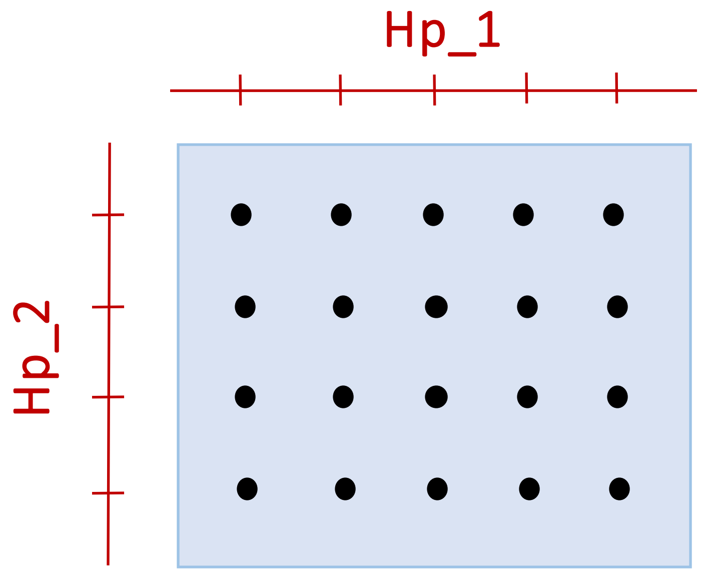




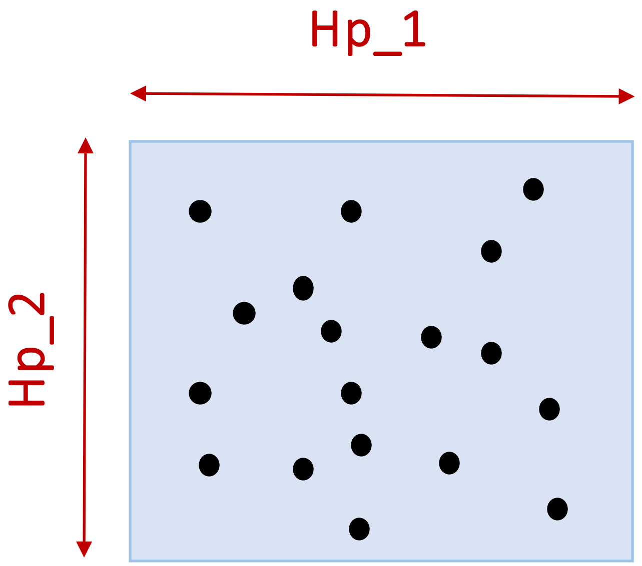
https://www.kaggle.com/code/residentmario/tuning-your-learning-rate

Hyper-parameters search: grid vs random

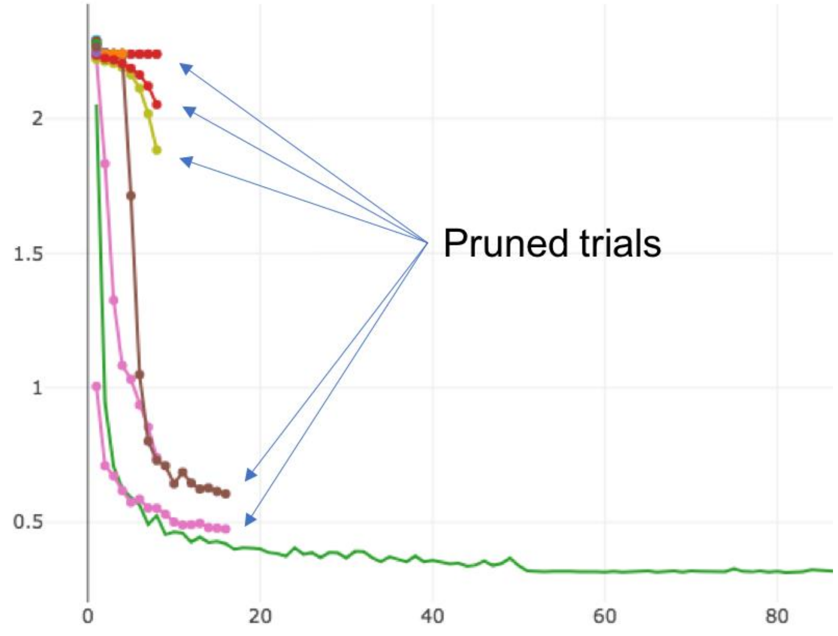
- Define the **search space**: discrete vs continuous
- Define the hyper-params **sampling** procedure
- Rank **validation performances** and choose the test setup



Grid



Random



Add hyperparam tuning function

```python
def get_model_hyperparams(self, method, optuna_m='random'):
    self.optuna_m = optuna_m
    self.hyper_mode = method
    path = os.path.join(self.get_exper_path(), 'tuned_hyperp-' + optuna_m + '.json')
    if method=='load_tuned':
        print('-----------------------------------------')
        print('Loading tuned hyperparams')
        print('-----------------------------------------')
        with open(path) as f:
            return json.load(f)

    elif method=='optuna_tuner':
        print('-----------------------------------------')
        print('Starting optuna tuner')
        model_hyperparams= self.run_hyperparams_tuning(optuna_m=optuna_m)
        print('-----------------------------------------')
        # save model hyperparams to json
        with open(path, 'w') as f:
            json.dump(model_hyperparams, f)
        return model_hyperparams
    else:
        sys.exit('ERROR: uknown hyperparam method')
```

Add tuner function

Define Optuna objective

STIIMA

```
1 usage
def run_hyperparams_tuning(self, optuna_m:str='random', n_trials: int=10):
    """
    Model hyperparameters tuning routine
    """
    def objective(trial):
        # Clear clutter from previous session graphs.
        tf.keras.backend.clear_session()
        # Update model configs with hyperparams trial
        self.model_configs = self.model_class.get_hyperparams_trial(trial=trial, settings=self.model_configs)

        # Build model using the current configs
        model = regression_model(settings=self.model_configs,
                                 sample_x=train_vali_block.x_vali[0:1])

        # Train model
        model.fit(train_x=train_vali_block.x_train, train_y=train_vali_block.y_train,
                  val_x=train_vali_block.x_vali, val_y=train_vali_block.y_vali,
                  pruning_call=TFKerasPruningCallback(trial, monitor: "val_loss"),
                  plot_history=False)

        # Compute val loss
        results = model.evaluate(x=train_vali_block.x_vali, y=train_vali_block.y_vali)
        return results
```

Create model with hyper sample

Train

Evaluate

Define optuna study

```python
# start from first train sample
init_sample = 0
# employ validation set till first test sample
test_sample_idx = self.test_set_idxs[0]
train_vali_block = self.__build_recalib_dataset_batches__(
    self.dataset[init_sample:test_sample_idx + self.data_configs.pred_horiz],
    fit_preproc=True).recalibBlocks[0]
```

Get train/vali samples

```python
if optuna_m == 'grid_search':
    search_space = self.model_class.get_hyperparams_searchspace()
    sampler = optuna.samplers.GridSampler(search_space)
    pruner = None
elif optuna_m == 'random':
    sampler = optuna.samplers.RandomSampler()
    pruner = optuna.pruners.MedianPruner(n_startup_trials=10, n_warmup_steps=5)
```

Define sampler and pruner

- **n_startup_trials** (*int*) – Pruning is disabled until the given number of trials finish in the same study.
- **n_warmup_steps** (*int*) – Pruning is disabled until the trial exceeds the given number of step. Note that this feature assumes that `step` starts at zero.

```python
# Add stream handler of stdout to show the messages
optuna.logging.get_logger("optuna").addHandler(logging.StreamHandler(sys.stdout))
# Unique identifier of the study.
study_name = (self.data_configs.task_name
              + self.model_configs['model_class'] + '-'
              + self.model_configs['PF_method']
              + '-' + optuna_m)
storage_name="sqlite:///db.sqlite3"
```

Define storage

Define optuna study

STIIMA

```python
study = optuna.create_study(direction="minimize",
                            sampler=sampler,
                            pruner=pruner,
                            storage= storage_name,  # Specify the storage URL here.
                            study_name=study_name,
                            load_if_exists=True
                            )
```

Create study

```python
timeout = 3600 * 24.0 * 7  # 7 days
study.optimize(objective, n_trials=n_trials, timeout=timeout)
```

Run study

```python
pruned_trials = study.get_trials(deepcopy=False, states=[TrialState.PRUNED])
complete_trials = study.get_trials(deepcopy=False, states=[TrialState.COMPLETE])
print("Study statistics: ")

print("Number of finished trials: ", len(study.trials))
print("  Number of pruned trials: ", len(pruned_trials))
print("  Number of complete trials: ", len(complete_trials))

print("Best trial:")
trial = study.best_trial
print("  Value: ", trial.value)
print("  Params: ")
for key, value in trial.params.items():
    print("    {}: {}".format( *args: key, value))
    # store best hyper in the config dict
    self.model_configs[key] = value

return self.model_class.get_hyperparams_dict_from_configs(self.model_configs)
```

Print and save results

Run Optuna tuner

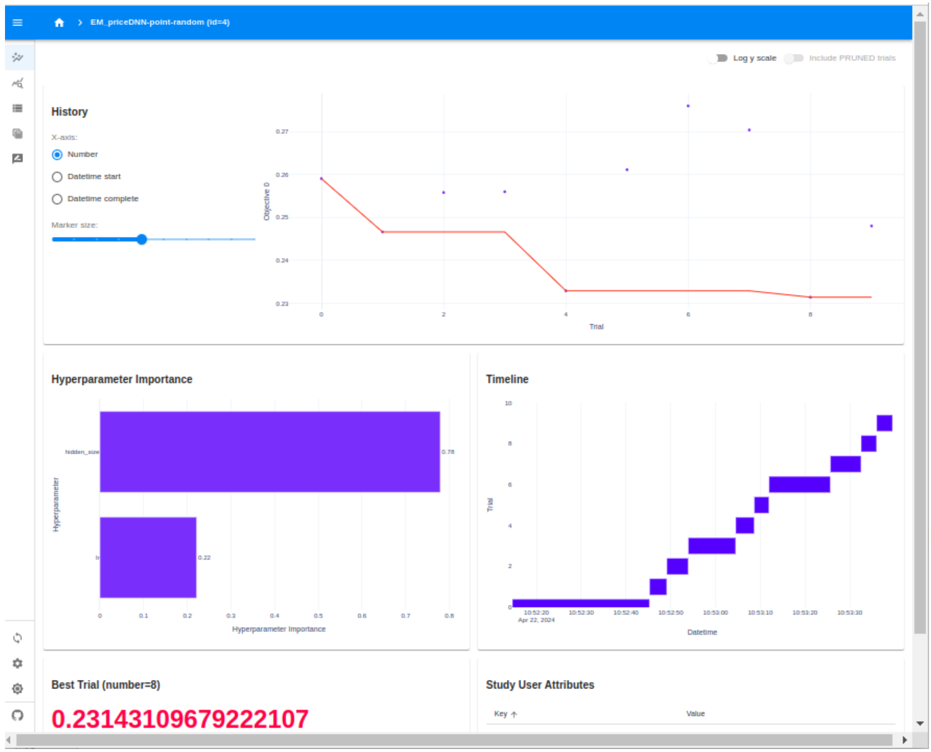
Set 'grid_search' or 'random'

```
"model_config": {
    "PF_method": "point",
    "model_class": "DNN",
    "optuna_m": "grid_search",
    "target_alpha": [
    ],
    "max_epochs": 800,
    "batch_size": 64,
    "patience": 20,
    "num_ense": 1
}
```

Start optuna dashboard:

optuna-dashboard sqlite:///db.sqlite3

```
(TF_p310) brus@brus-ThinkPad-T15p-Gen-3:~/PycharmProjects/PEPF_lab_v2$
optuna-dashboard sqlite:///db.sqlite3
Listening on http://127.0.0.1:8080/
Hit Ctrl-C to quit.
```

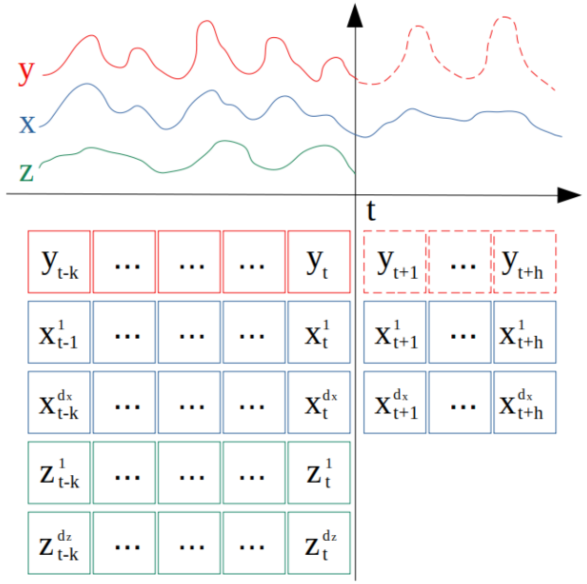


Run optuna tuner (or load json file)

```
# Load hyperparams from file (select: load_tuned or optuna_tuner)
hyper_mode = 'optuna_tuner'
```

Setup model input

```python
class DNNRegressor:
    def __init__(self, settings, loss):
        self.settings = settings
        self.__build_model__(loss)

@staticmethod
def build_model_input_from_series(x, col_names: List, pred_horiz: int):
    # get index of target and past features
    past_col_idxs = [index for (index, item) in enumerate(col_names)
                     if features_keys['target'] in item or features_keys['past'] in item]

    # get index of const features
    const_col_idxs = [index for (index, item) in enumerate(col_names)
                      if features_keys['const'] in item]

    # get index of futu features
    futu_col_idxs = [index for (index, item) in enumerate(col_names)
                     if features_keys['futu'] in item]

    # build conditioning variables for past features
    past_feat = [x[:, :-pred_horiz, feat_idx] for feat_idx in past_col_idxs]
    # build conditioning variables for futu features
    futu_feat = [x[:, -pred_horiz:, feat_idx] for feat_idx in futu_col_idxs]
    # build conditioning variables for cal features
    c_feat = [x[:, -pred_horiz:-pred_horiz + 1, feat_idx] for feat_idx in const_col_idxs]

    # return flattened input
    return np.concatenate(past_feat + futu_feat + c_feat, axis=1)
```



Build DNN model in Tensorflow

STIIMA

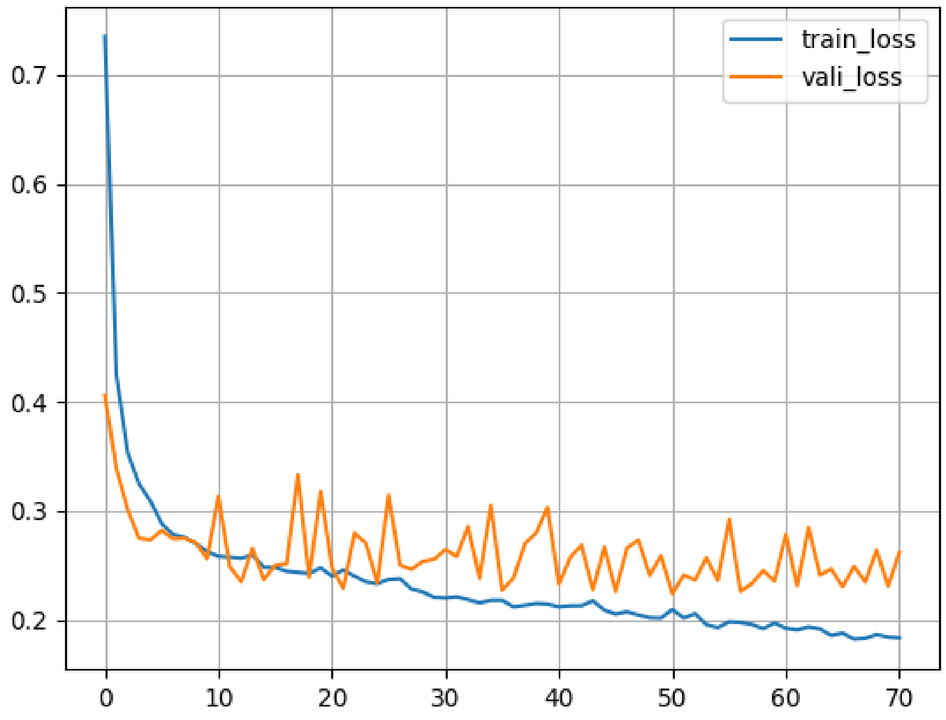
```python
def __build_model__(self, loss):
    x_in = tf.keras.layers.Input(shape=(self.settings['input_size'],))
    x_in = tf.keras.layers.BatchNormalization()(x_in)
    x = (tf.keras.layers.Dense(self.settings['hidden_size'],
                               activation=self.settings['activation'],
                               )(x_in))
    for hl in range(self.settings['n_hidden_layers'] - 1):
        x = tf.keras.layers.Dense(self.settings['hidden_size'],
                                  activation=self.settings['activation'],
                                  )(x)
    if self.settings['PF_method'] == 'point':
        out_size = 1
        logit = tf.keras.layers.Dense(self.settings['pred_horiz'] * out_size,
                                      activation='linear',
                                      )(x)
        output = tf.keras.layers.Reshape((self.settings['pred_horiz'], 1))(logit)
# Create model
self.model= tf.keras.Model(inputs=[x_in], outputs=[output])
# Compile the model
self.model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=self.settings['lr']),
                   loss=loss)
```

Keras

Functional API

Model fitting function recap

```python
def fit(self, train_x, train_y, val_x, val_y, verbose=0, pruning_call=None):
    # Convert the data into the input format using the internal converter
    train_x = self.build_model_input_from_series(x=train_x,
                                                  col_names=self.settings['x_columns_names'],
                                                  pred_horiz=self.settings['pred_horiz'])
    val_x = self.build_model_input_from_series(x=val_x,
                                                col_names=self.settings['x_columns_names'],
                                                pred_horiz=self.settings['pred_horiz'])
    es = tf.keras.callbacks.EarlyStopping(monitor="val_loss",
                                          patience=self.settings['patience'],
                                          restore_best_weights=False)

    # Create folder to temporally store checkpoints
    checkpoint_path = os.path.join(os.getcwd(), 'tmp_checkpoints', 'cp.weights.h5')
    checkpoint_dir = os.path.dirname(checkpoint_path)
    if not os.path.exists(checkpoint_dir):
        os.makedirs(checkpoint_dir)

    cp = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                            monitor="val_loss", mode="min",
                                            save_best_only=True,
                                            save_weights_only=True, verbose=0)
    if pruning_call==None:
        callbacks = [es, cp]
    else:
        callbacks = [es, cp, pruning_call]

    history = self.model.fit(train_x,
                             train_y,
                             validation_data=(val_x, val_y),
                             epochs=self.settings['max_epochs'],
                             batch_size=self.settings['batch_size'],
                             callbacks=callbacks,
                             verbose=verbose)
    # Load best weights: do not use restore_best_weights from early stop since works only in case it stops training
    self.model.load_weights(checkpoint_path)
    # delete temporary folder
    shutil.rmtree(checkpoint_dir)
    return history
```



2 usages (1 dynamic)

```python
def predict(self, x):
    x = self.build_model_input_from_series(x=x,
                                            col_names=self.settings['x_columns_names'],
                                            pred_horiz=self.settings['pred_horiz'])
    return self.model(x)
```

2 usages (1 dynamic)

```python
def evaluate(self, x, y):
    x = self.build_model_input_from_series(x=x,
                                            col_names=self.settings['x_columns_names'],
                                            pred_horiz=self.settings['pred_horiz'])
    return self.model.evaluate(x=x, y=y)
```

Define Optuna search space

STIIMA

1 usage (1 dynamic)

```python
@staticmethod
def get_hyperparams_trial(trial, settings):
    settings['hidden_size'] = trial.suggest_int('hidden_size', 64, 960, step=64)
    settings['n_hidden_layers'] = 2  # trial.suggest_int('n_hidden_layers', 1, 3)
    settings['lr'] = trial.suggest_float('lr', 1e-5, 1e-1, log=True)
    settings['activation'] = 'softplus'
    return settings
```

Random search

1 usage (1 dynamic)

```python
@staticmethod
def get_hyperparams_searchspace():
    return {'hidden_size': [128, 512],
            'lr': [1e-4, 1e-3]}
```

Grid search

1 usage (1 dynamic)

```python
@staticmethod
def get_hyperparams_dict_from_configs(configs):
    model_hyperparams = {
        'hidden_size': configs['hidden_size'],
        'n_hidden_layers': configs['n_hidden_layers'],
        'lr': configs['lr'],
        'activation': configs['activation']
    }
    return model_hyperparams
```

Utility function (map to json)

- Model specific
- Can be customized and extended

Assignment PEF2

```
def run_hyperparams_tuning(self, optuna_m:str='random', n_trials: int=10):
```

- Run 'random' hyperparameter search on DNN (n_trials=50)
- Provide a brief report of the results obtained (from dashboard trialTable)

Facultative:

- Run test recalibration (3 times) on January 2017 using the best hyperparams selected by Optuna and report the results obtained (use the metrics of the previous assignment)

To run/store multiple recalibrations, copy and rename the related folder

```
# Select run
run_id = 'recalib_opt_grid_1_2'
# Load hyperparams from file (select: load_tuned or optuna_tuner)
hyper_mode = 'load_tuned'
# Plot train history flag
```

```
point-DNN
  recalib_opt_grid_1_1
    results
      recalib_test_results-tuned-grid_search.p
      recalib_test_results-tuned-random.p
    exper_configs.json
    tuned_hyperp-grid_search.json
    tuned_hyperp-random.json
  recalib_opt_grid_1_2
    results
    exper_configs.json
    tuned_hyperp-grid_search.json
    tuned_hyperp-random.json
```

Install Tensorflow probability for L3

tensorflow/
probability

Probabilistic reasoning and statistical analysis in TensorFlow

In your venv:

--> pip install tf-keras

--> pip install --upgrade tensorflow-probability

Consiglio Nazionale delle Ricerche

Thanks

Alessandro Brusaferri