Network setup and training conditions

In this notebook we show how the models were built and trained. All settings except:

- data source (reduced data quality + number of features)
- number of training epochs

  are identical with those presented in the paper. Running this notebook will not yield the performance numbers presented in the paper because the beefy networks require larger variation in the data.

  The AttA3 network would probably require few days to train so running it with full epoch count is not advisable if one wants a quick look at the models.

In [1]:

```
%load_ext autoreload
%autoreload 2
# %matplotlib widget
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np
import pandas as pd
from pathlib import Path

import torch
from fastai.data.transforms import TfmdLists, DataLoaders
import fastai.learner
import fastai.callback.schedule
import fastai.callback.tracker
import fastai.basics
import fastai.losses


import network_definitions
```

In [2]:

```
data_load_path = Path("../data_sample")
train_data = np.load(data_load_path / "train_samples.npy")
valid_data = np.load(data_load_path / "valid_samples.npy")
train_samples = list(train_data)  # to match the requirements of fastai.TfmdLis
valid_samples = list(valid_data)

sample_length = train_data[0].shape[0]
# Settings to match the paper experiments. Except the number of columns, all the
future_length = 60
past_length = sample_length - future_length
no_features = train_data[0].shape[1]
known_column_indexes = list(range(no_features))  # [0, 1, 2, 3]
command_indexes = [2]
target_indexes = [3]
feature_groups = [(0, 1, 3)]
average_range = 0.2
batch_size = 32

feature_itemizer = network_definitions.FeatureItemizer(future_length, known_colu
```

```
                                        command_indexes, target_indexes, featur
tls_train = TfmdLists(train_samples, [feature_itemizer])
tls_valid = TfmdLists(valid_samples, [feature_itemizer])
data_dloader = DataLoaders.from_dsets(tls_train, tls_valid, bs=batch_size, drop_
print(f"Data volumes: Train: {len(train_samples)}, Validation: {len(valid_sample
```

```
Data volumes: Train: 7255, Validation: 1073
```

In [3]:

```
def get_DAffAffGau():
    model = network_definitions.ConstructDelayNet(no_features, len(command_index
                                                  filter_low_classname="AffineTra
                                                  aggregator_low_expansion=1, agg
                                                  temporal_contractor_classname='
                                                  filter_high_classname="GaussFil
                                                  aggregator_high_expansion=1, ag
    return model

def get_DLogAffGau():
    model = network_definitions.ConstructDelayNet(no_features, len(command_index
                                                  filter_low_classname="LogGauss'
                                                  aggregator_low_expansion=1, agg
                                                  temporal_contractor_classname='
                                                  filter_high_classname="GaussFil
                                                  aggregator_high_expansion=1, ag
    return model


def get_AttA1():
    model = network_definitions.ICCP_Wrap_AttentionModel(no_features, len(comman
                                                         hidden_size=8, num_laye
    return model


def get_AttA3():
    model = network_definitions.ICCP_Wrap_AttentionModel(no_features, len(comman
                                                         hidden_size=512, num_la
    return model


def instantiate_learner(model, data_loader):
    learner = fastai.learner.Learner(data_loader, model, loss_func=fastai.losses
                    cbs=[fastai.callback.tracker.ReduceLROnPlateau(patience=20,
                         fastai.callback.tracker.EarlyStoppingCallback(patience
                         fastai.callback.tracker.SaveModelCallback(fname="best_
                         fastai.callback.tracker.TerminateOnNaNCallback(),
                         fastai.callback.progress.CSVLogger("learning_progress.
                          ],)
    return learner

def evaluate_learner(crt_learner, samples):
    raw_preds, raw_targets = crt_learner.get_preds()
    eval_itemset_arr = np.array(samples)
    feature_itemizer = crt_learner.dls[0].fs[0]
    future_temp_pred_transf = network_definitions.decode_predictions_from_the_ne
    future_temp_target_transf = network_definitions.decode_predictions_from_the_
    mae = np.average(np.abs(future_temp_target_transf[:, -future_length:] - futu
    mae_wrt_to_time = np.average(np.abs(future_temp_target_transf[:, -future_len
    return mae, mae_wrt_to_time, future_temp_pred_transf, future_temp_target_tra
```

In [4]:
```python
model_names = []
learners = []
```

In [5]:
```python
# Experimental setup in paper
# global_lr_rate = 1e-3
# global_no_max_epochs = 1000

# Experimental setup here, for demonstration
global_lr_rate = 1e-3
global_no_max_epochs = 3
```

In [6]:
```python
DAffAffGau_learner = instantiate_learner(get_DAffAffGau(), data_dloader)
DAffAffGau_learner.fit_one_cycle(global_no_max_epochs, global_lr_rate)
model_names.append("DAffAffGau")
learners.append(DAffAffGau_learner)
```

epoch	train_loss	valid_loss	time
0	0.011518	0.017815	00:12
1	0.008342	0.010210	00:12
2	0.007646	0.009017	00:12

```
Better model found at epoch 0 with valid_loss value: 0.01781485415995121.
Better model found at epoch 1 with valid_loss value: 0.010210155509412289.
Better model found at epoch 2 with valid_loss value: 0.009016791358590126.
```

In [7]:
```python
DLogAffGau_learner = instantiate_learner(get_DLogAffGau(), data_dloader)
DLogAffGau_learner.fit_one_cycle(global_no_max_epochs, global_lr_rate)
model_names.append("DLogAffGau")
learners.append(DLogAffGau_learner)
```

epoch	train_loss	valid_loss	time
0	0.009463	0.088862	00:13
1	0.007932	0.023762	00:14
2	0.007335	0.007523	00:13

```
Better model found at epoch 0 with valid_loss value: 0.0888623520731926.
Better model found at epoch 1 with valid_loss value: 0.023762168362736702.
Better model found at epoch 2 with valid_loss value: 0.007523125037550926.
```

In [8]:
```python
AttA1_learner = instantiate_learner(get_AttA1(), data_dloader)
AttA1_learner.fit_one_cycle(global_no_max_epochs, global_lr_rate)
model_names.append("AttA1")
learners.append(AttA1_learner)
```

epoch	train_loss	valid_loss	time
0	0.031977	0.009789	00:47
1	0.009757	0.011009	00:45
2	0.008705	0.010445	00:47

```
Better model found at epoch 0 with valid_loss value: 0.009788864292204388.
```

In [9]:
```
AttA3_learner = instantiate_learner(get_AttA3(), data_dloader)
AttA3_learner.fit_one_cycle(global_no_max_epochs, global_lr_rate)
model_names.append("AttA3")
learners.append(AttA3_learner)
```

epoch	train_loss	valid_loss	time
0	0.008202	0.008381	01:12
1	0.006932	0.006418	01:09
2	0.006595	0.006736	01:09

```
Better model found at epoch 0 with valid_loss value: 0.008381090126931667.
Better model found at epoch 1 with valid_loss value: 0.006417920347303152.
```

Network summary.

Note where the bulk of parameters is concentrated in each network.

In [10]:
```
for k in range(len(learners)):
    print(f"Architecture for {model_names[k]}:")
    network_definitions.print_model_weights_rec(learners[k].model, max_level=1)
```

```
Architecture for DAffAffGau:
<class 'network_definitions.ConstructDelayNet'> with 12049 parameters:
      <class 'network_definitions.BankedFilters'> with 64 parameters:
      Parameters: 64
      ------------------
      <class 'network_definitions.FeatureAggregationStack'> with 664 parameters:
      Parameters: 664
      ------------------
      <class 'network_definitions.AffineTransform'> with 32 parameters:
      Parameters: 32
      ------------------
      <class 'network_definitions.BankedFilters'> with 776 parameters:
      Parameters: 776
      ------------------
      <class 'network_definitions.FeatureAggregationStack'> with 10513 parameter
s:
      Parameters: 10513
      ------------------
Parameters: 12049
------------------
Architecture for DLogAffGau:
<class 'network_definitions.ConstructDelayNet'> with 13681 parameters:
      <class 'network_definitions.BankedFilters'> with 64 parameters:
      Parameters: 64
      ------------------
      <class 'network_definitions.FeatureAggregationStack'> with 2296 parameter
s:
      Parameters: 2296
      ------------------
      <class 'network_definitions.AffineTransform'> with 32 parameters:
      Parameters: 32
      ------------------
      <class 'network_definitions.BankedFilters'> with 776 parameters:
      Parameters: 776
      ------------------
      <class 'network_definitions.FeatureAggregationStack'> with 10513 parameter
s:
      Parameters: 10513
      ------------------
```

```
Parameters: 13681
------------------
Architecture for AttA1:
<class 'network_definitions.ICCP_Wrap_AttentionModel'> with 8105 parameters:
      <class 'network_definitions.AttentionModel'> with 8105 parameters:
      Parameters: 8105
      ------------------
Parameters: 8105
------------------
Architecture for AttA3:
<class 'network_definitions.ICCP_Wrap_AttentionModel'> with 16239929 parameters:
      <class 'network_definitions.AttentionModel'> with 16239929 parameters:
      Parameters: 16239929
      ------------------
Parameters: 16239929
------------------
```

Performance evaluation

There is no expectation that these numbers will match the paper's results. The networks are identical but the data quality is lower.

In [11]:

```
maes = []
maes_timewise = []
predictions = []

for l in learners:
    mae, mae_t, scaled_preds, scaled_targets = evaluate_learner(l, valid_samples
    maes.append(mae)
    maes_timewise.append(mae_t)
    predictions.append(scaled_preds)

# Adding "zero" predictor
raw_preds, raw_targets = learners[0].get_preds()
zero_preds =  torch.zeros_like(raw_preds)
eval_itemset_arr = np.array(valid_samples)
feature_itemizer = learners[0].dls[0].fs[0]
zero_transf = network_definitions.decode_predictions_from_the_network(zero_preds
mae_zero = np.average(np.abs(scaled_targets[:, -future_length:] - zero_transf[:,
mae_zero_time = np.average(np.abs(scaled_targets[:, -future_length:] - zero_tran
model_names.append("Zero")
maes.append(mae_zero)
maes_timewise.append(mae_zero_time)
predictions.append(zero_transf)
model_names = model_names[:len(learners)+1] # just to make sure that on re-runs
for k in range(len(model_names)):
    print(f"Performance of {model_names[k]}: MAE: {maes[k]:.4}")
```

```
Performance of DAffAffGau: MAE: 0.1356
Performance of DLogAffGau: MAE: 0.1133
Performance of AttA1: MAE: 0.1457
Performance of AttA3: MAE: 0.09819
Performance of Zero: MAE: 0.1366
```