Deep Learning Project 2

Disclaimer: it is likely that we make small changes to the project description in the coming weeks. If we do update the project description, we will announce the changes on Blackboard.

This document describes the forecasting project in IT3030. The task at hand is to forecast electric power consumption 24 hours ahead, using publicly available historical data about power consumption and weather conditions in different parts of Norway.

Your task will be to create forecast models using different neural network architectures, compare the results and analyze and interpret their strengths and weaknesses.

The project will give you hands-on experience with applying both feed-forward and recurrent neural network models to a real-world application using real-world data. The main goal of this project is to give you a practice-oriented assignment that might illustrate how deep learning may be applied in the industry.

The rules are as follows:

- The task must be solved individually
- Coding requirements:
  - You **must** write your code in Python.
  - We **encourage** you to use a deep learning framework such as TensorFlow or PyTorch.
  - You may use libraries such as NumPy, SciPy and Pandas for preprocessing, interpolation etc.
  - You are **not** allowed to use pre-built RNN/CNN/Transformer-based sequence models, i.e. your code has to manage and interact with the respective units directly (e.g. by creating a Keras or Pytorch model containing an LSTM as one of its layers).
  - You are **not** allowed to use dedicated time-series forecasting libraries such as Sktime or Darts.
  - You may use Scikit-learn or similar libraries for feature preprocessing such as normalization.
- Your solution will be given between 0 and 20 points; the rules for the scoring are listed in Section 8.
- The deadline for the **demonstration** of this project is week 12. Information regarding the exact dates and times for demos, possible signups, etc., will be posted on Blackboard.
- Code must be properly commented and uploaded to Blackboard prior to or (immediately) after your demonstration session.

Although you might make use of libraries such as Sktime or Darts and/or other frameworks in an industry setting, we require that you both manage the neural network and perform feature and dataset manipulation yourself in this project. However, you are encouraged to consult both code and documentation from open-source libraries as part of finding out how to write your own code.

1 Introduction

The electric power grid is a complex network of power lines, substations and transformers that connects producers (wind farms, hydropower plants, nuclear power plants, etc.) to consumers (end-users such as residential homes, industry, etc.). Together, these three components make up the electric power system.

The grid does not store energy, so in order to keep the power system balanced, the demand from consumers must be met continuously and instantaneously by the producers. Good forecasts of future power consumption are therefore

essential for both planning and operation of the power system. In the Nordic countries, producers and consumers settle prices and volumes for the next day on the Nord Pool day-ahead market. As such, Nord Pool is the foundation for the producers to plan their production in the next 24 hours. Both buyers and sellers depend on reliable forecasts for the next few days when they participate in this market.

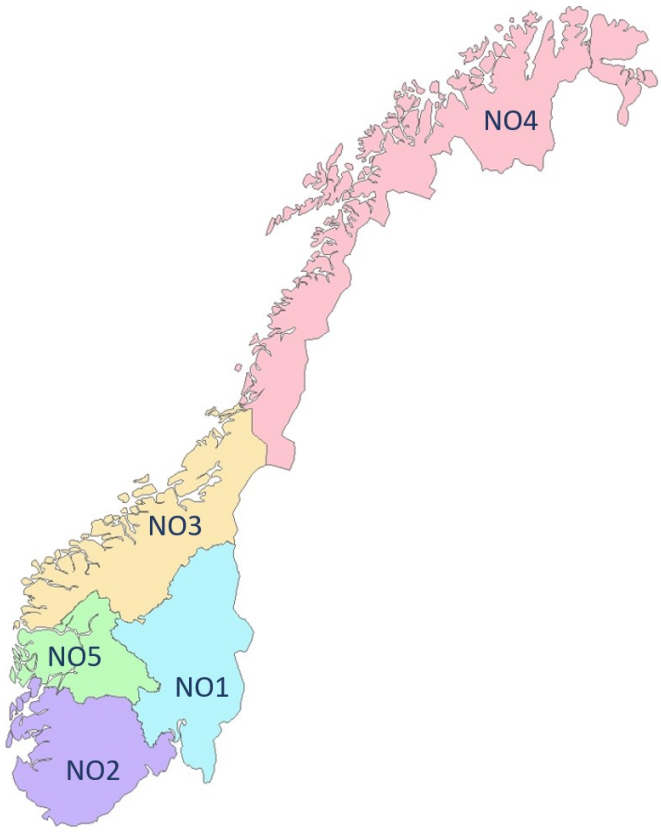

Figure 1: Electricity price areas in Norway in 2024. Image from NVE.

The Norwegian power system is divided into 5 different regions, so-called *bidding areas*, as shown in Figure 1[1].

The dataset used in this project contains hourly recordings of the historical power consumption for each of the different bidding areas for the years 2017-2023. The data are publicly available on eSett, but a preprocessed dataset is also available on Blackboard for your convenience. If you prefer to download the data yourself, be aware that both the sign and the resolution change during this period.

The electric power consumption, in particular residential consumption, is strongly correlated with weather. This is especially true in Norway, where we have a fairly cold climate and electric heating is prevalent. In addition to the measured electricity consumption, the dataset on Blackboard therefore also contains temperature measurements at 5 selected locations, one for each bidding area.

The temperature measurements have been downloaded from Frost, which is MET Norway's public archive of historical weather and climate data. In an actual forecasting situation, temperature *measurements* would not be available for the forecast period (since it would be in the future), so you would have to use the weather *forecast* as input. In this project, you will use temperature *measurements* shifted into the future as a "proxy" for an actual weather forecast.

Your task in this project is to train deep neural networks to forecast the consumption 24 hours into the future for each of the 5 bidding areas. The following sections will flesh out more practical information on the data, as well as some general tips for sequence modelling and the different sub-tasks.

[1] NVE have written a good summary (in Norwegian) of why we have different bidding areas here.

2 Data

The data are provided in a zipped CSV (Comma-Separated Values) file consumption_and_temperatures.csv.zip. The first line contains the header of each time series in the file and describes the data stored within them. The following lists provide an extended explanation of the different time series:

2.1 Data

- **timestamp**: The timestamp of each datum, UTC offset information. Keep in mind that Norway changes between summer time and normal time, which are UTC +2 and +1, respectively, so you must decide if and how you want to take this into account when developing you model.
- **NOx_temperature**: Columns named NO1_temperature, NO2_temperature and so on give minimum temperature observed per hour for each bidding area.
- **NOx_consumption**: Columns named NO1_consumption, NO2_consumption and so on give the power consumption for each bidding area. **These are the target variables that you will train your network to predict**.

3 The Forecasting Task

Your task is to develop models that can forecast the power consumption **24 hours into the future**. Since your data has 1-hour resolution, for a point in time $t = t_i$, given historical consumption and temperature data from $t = t_0$ to $t = t_i$ as well as temperature "forecasts" for $t = t_{i+1}$ to $t = t_{i+24}$[2], your model should produce 24 forecasts covering the period $t = t_{i+1}$ to $t = t_{i+24}$.

As you will see when visualizing the data, the power consumption data exhibit strong *seasonal patterns* at multiple levels; daily, weekly and yearly are the most pronounced. It is up to you whether you train your models to make predictions only at a specific time of day, or at every hour of the day. However, you may be asked to reflect on your choice during the demo, taking into account data set size, feature engineering, model structure, and so on.

Your system should be able to train a model to predict the consumption for each of the 5 bidding areas separately. It is up to you whether you choose to train separate models for each bidding area, or to encode the bidding area as a feature and train a single model. You are not required to actually *train* (i.e. fit) models for each of the 5 bidding areas, it will suffice that you fit models as described below, but it should be possible to retrain a model for another bidding area by adjusting a single parameter.

In order to achieve full score on the project, you will have to develop models using several different architectures, and set them up as follows:

1. Implement three different forecasting models, each model using one the four following architectures[3]:
   - An RNN in the form of an LSTM or GRU (i.e. it contains one or more LSTM/GRU layers, possibly in addition to several other layers).
   - A CNN.
   - A transformer.
   - A feed-forward regression model that uses a combination of historical consumption data and past and future weather data, as indicated in the paragraph above.
2. Choose one bidding area (e.g. NO3) on which each of the models described in 1 above is trained.
3. Choose one model architecture to train on another bidding area (e.g. NO2).
4. Finally, use the model trained in 2 with the same architecture as in point 3 and use it to make predictions for the bidding area you selected in point 3. In other words, apply a model to another bidding area than the one it was trained on, and compare the results with a model trained for that specific bidding area.

[2] As a minimum, you should use the historical weather data as a proxy for a weather forecast, but if desired you can download actual forecasts and/or other meteorological data from MET and include these in your model.

[3] Obviously, you should not use the same architecture for multiple models. 3 different CNN models will not results in a full score.

For each of these models, you should be able to produce summaries and samples of forecasting performance, as described in more detail in Section 8.

4 Sequence Alternatives

There are many ways of setting up the input sequences, all of which affect model structure and forecast quality. In this section, we describe a number of alternatives with illustrations and any perceived pros and cons. While only one of these strategies has been confirmed to work well, multiple have been listed to help you understand the finer details of the task and sequence modeling in general, as well as provide you with some ideas should you decide to try a different strategy.

Forecasting more than one step into the future is often called *multi-step forecasting*. Irrespective of the choice of model, there are different overall approaches to multi-step forecasting:

1. Use your model to forecast the next timestep t_{i+1}. Then reuse the same model and include the forecast for t_{i+1} in the input to produce the forecast for timestep t_{i+2}. This is sometimes referred to as a *recursive*, *autoregressive* or *step-by-step* strategy.

   This approach lends itself to use with RNN and transformer models, and is the basis for the "n in, 1 out" described below.

2. Given inputs, including historic values for $\mathbf{y}$ (the target, i.e. consumption), for any number of timesteps up to and including t_i, and, if available, external regressors (in this case, temperature) for the period $t = t_{i+1}$ to $t = t_{i+n}$ where n is your forecast horizon, the model outputs all n forecasts at once. This is sometimes referred to as a *multi-output* strategy.

3. Develop a separate model for each step of the forecast horizon. This is sometimes referred to as a *direct* or *independent value* strategy.

More in-depth descriptions of these strategies can be found here or here.

We recommend that you try to get some decent results with the "n in, 1 out" strategy (Section 4.2), before, moving on to other strategies.

Whichever strategy you employ, you must set your model up so that it takes advantage of any external regressors (weather forecast, constant features as described in Section 5, etc.) in the entire forecast window.

Again, make sure your model does not use any knowledge of measured consumption within the forecast window.

4.1 Sequence Notes

It is important to understand that in sequence models, each **step** of each input sequence may contain multiple features. For this project, one might send in a sequence of for instance 24 or 168 steps, each of which may contain consumption, weather and other features relevant to that specific time step. Thus, a single input may take the shape $[n_{\text{seq}}, n_{\text{features}}]$, and a mini-batch the shape $[n_{\text{batch}}, n_{\text{seq}}, n_{\text{features}}]$. Keep this in mind in the following sections, as the sequence structure may be hard to grasp for those unaccustomed to them.

4.1.1 Figure Notes

Most additional features are either forecasts (like weather) or plans. Thus, their values within the forecasting window are known when performing the consumption forecast. Such features are referred to as *plan* in the figures, and you will see that they are often easier to deal with than, for instance, *lagged target*/*previous_y* when forecasting. See Table 1 for an overview of terms and symbols used in the following figures.

Term/symbols	**Description**
a,b,c,...	Timestep indexing.
A,B,C,...	Input at corresponding timestep, e.g. input A represents timestep a.
plan1_a, plan2_a, plan3_a...	Different *plan*/*forecast* features for timestep *a*.

Table 1: Overview of terms and symbols appearing in the figures.

4.2 n in, 1 out strategy

In this strategy, each input is **n** long, but the model only produces a single forecast for the next time step. E.g. you may have sequence lengths of 24 and forecast a single timestemp into the future. To achieve this, all inputs are propagated through the RNN, but only the very last "hidden state" is fed to subsequent layers downstream from the RNN unit. For forecasting multiple steps into the future during evaluation, you have to drop the first input, add the **weather forecast** for the next time step and then **switch out** the consumption with your **forecast**. See Figure 2 and the following pseudocode for more details.

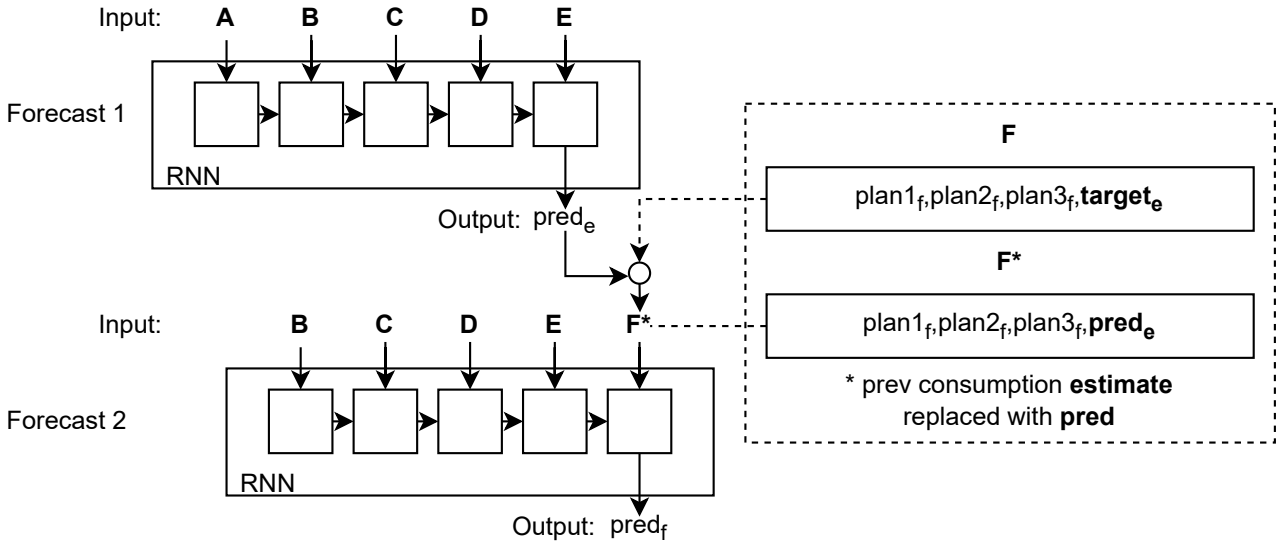

Figure 2: Two-step forecasting scenario using "**n** in, **1** out". In this scenario, we emulate that we have consumption data for every step up to, but not including, **e**, and weather forecast for the foreseeable future beyond **e**. Note the difference between input **F** and input **F***. When forecasting the consumption in time step **f**, we have to replace the consumption in the previous time step with our previous forecast (time step **e**).

4.2.1 Multistep Forecast Pseudocode

```
# NOTE: Simple indexing is used for clarity, your implementations will
# likely use pre-split mini-batches and/or data loaders
# Copy input to ensure that inplace changes only affect this specific multistep forecast
x_copy = copy(x)
model_input = x_copy[start_ind:start_ind+seq_len]
forecasts = []
forecast = model(model_input)
forecasts.append(forecast)
for pred_no in (0..forecast_window_len-1):
  start_ind += 1
  # Replace the last prev_y in the new input range with prev forecast
  x_copy[start_ind+seq_len-1, prev_y_ind] = forecast
  # Extract new input from copied x. Note that all prev_y from consumption in
  # the forecast window will be replaced with forecasts iteratively
  model_input = x_copy[start_ind:start_ind+seq_len]

  forecast = model(model_input)
  forecasts.append(forecast)
```

4.2.2 Pros and Cons

Pros:

- Somewhat easy to tune.
- Highly specialized for single time step forecast.

Cons:

- Long sequences incur heavy computational cost since the model has to process **n** inputs for every single forecast step.
- Not penalized for diverging forecasts over multiple steps
- Somewhat challenging coding because you must replace estimates when forecasting multiple steps into the future, see Section 4.2.1.

4.3 n in, m out

This strategy is identical to "**n** in, **1** out", Section 4.2, with the exception that it outputs the forecast of multiple steps into the future in one go. A challenge with these setups is that trivial implementations will not be able to provide the model with plans for anything other than the very first forecast step. It may also be difficult to model sequential relations between the individual forecasted steps. See Figure 3 for more details.

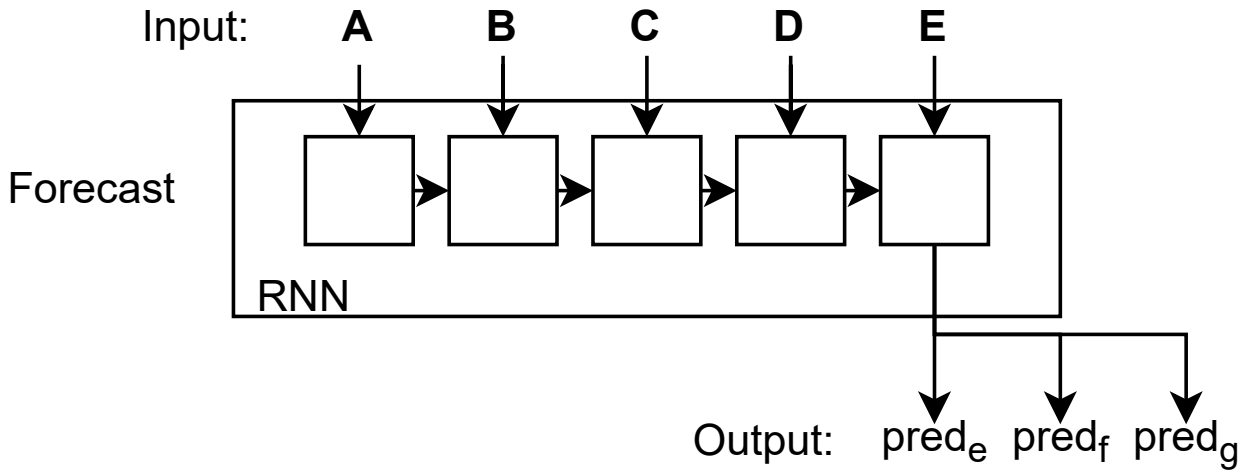

Figure 3: Three-step forecasting scenario using "**n** in, **m** out". In this scenario, we emulate that we have consumption data for every step up to, but not including, **e**, and weather forecasts for the foreseeable future beyond **e**. However, a naïve implementation of this strategy would be unable to use the weather forecasts for steps beyond **e**. The model outputs three forecasting steps in one go.

4.3.1 Pros and Cons

Pros:

- Lighter computational load.
- (Possibly) simpler code for forecasting multiple steps.
- Easier to penalize forecasts diverging after multiple steps.

Cons:

- **Not verified to work**, tests on other datasets displayed trouble with overly conservative forecasting that stayed close to the last recorded consumption.
- Harder to model dynamics between consumption forecasts far into the future and providing the model with weather forecasts for these.

4.4 n in, n out

This setup outputs one prediction for every input by propagating every "hidden state" of the RNN through the rest of the network. Has similar downsides as "**n** in, **1** out", Section 4.2, when forecasting multiple steps into the future as you have to replace estimates with forecasts before forecasting the next step. See Figure 4 for more details.

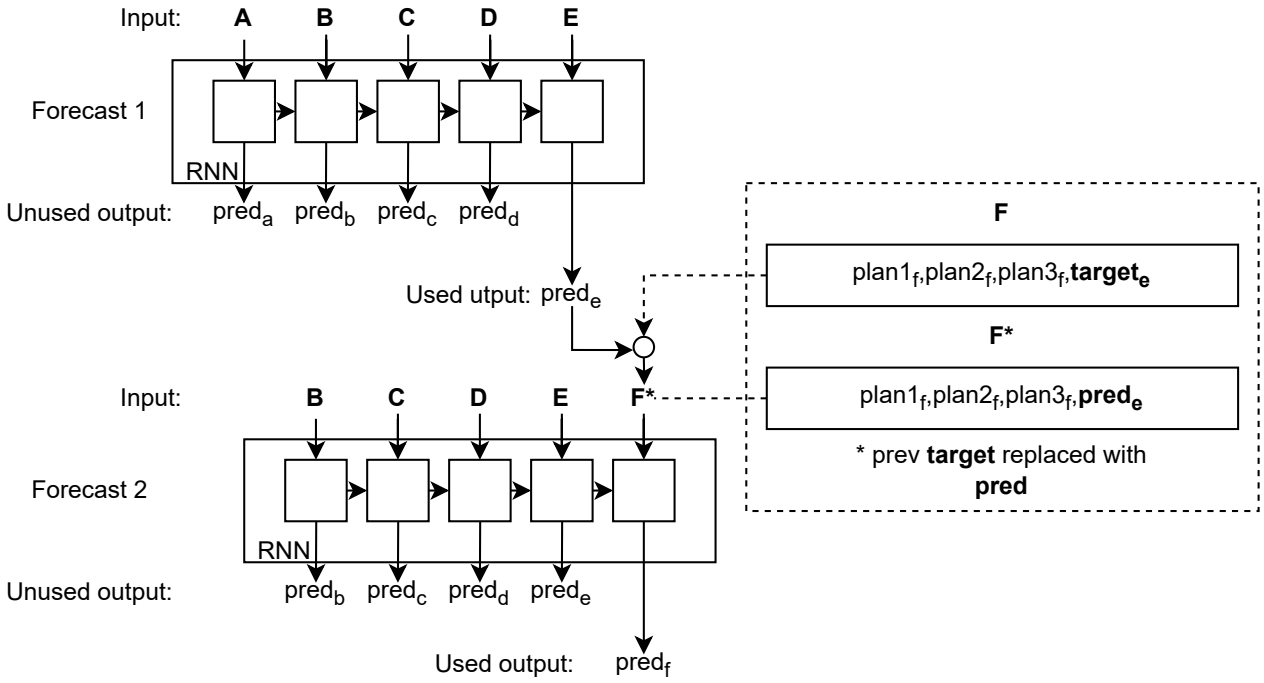

Figure 4: Two-step forecasting scenario using "**n** in, **n** out". In this scenario, we emulate that we have consumption data for every step up to, but not including, **e**, and weather forecasts for the foreseeable future beyond **e**. Note the difference between input **F** and input **F***. This model outputs a consumption forecast for every single input, but we only propagate the very last forecast step when doing multistep forecasting. When forecasting the consumption in time step **f**, we have to replace the estimate of the consumption in the previous time step with our previous forecast (time step **e**).

4.4.1 Pros and Cons

Pros:

- Very light computational load.

Cons:

- **Not verified to work**, smaller tests displayed trouble with overly conservative forecasting that stayed close to the last recorded consumption.
- Highly generalized model. Input 1 and its intermediaries affects all subsequent outputs, which may result in conflicting gradients and possibly make the model weigh short-term input higher.
- More challenging coding because you must replace estimates when forecasting multiple steps into the future, see section 4.2.1.

4.5 Encoder-Decoder

Possibly a good combination between "**n** in, **1** out" and "**n** in, **m** out". Feed the last hidden state from an RNN, used as an encoder, to another RNN, used as a decoder. The decoder is fed weather forecasts for each future time step it is to forecast, ultimately propagating values to produce multiple forecasted time steps in one run. See Figure 5 for more details.

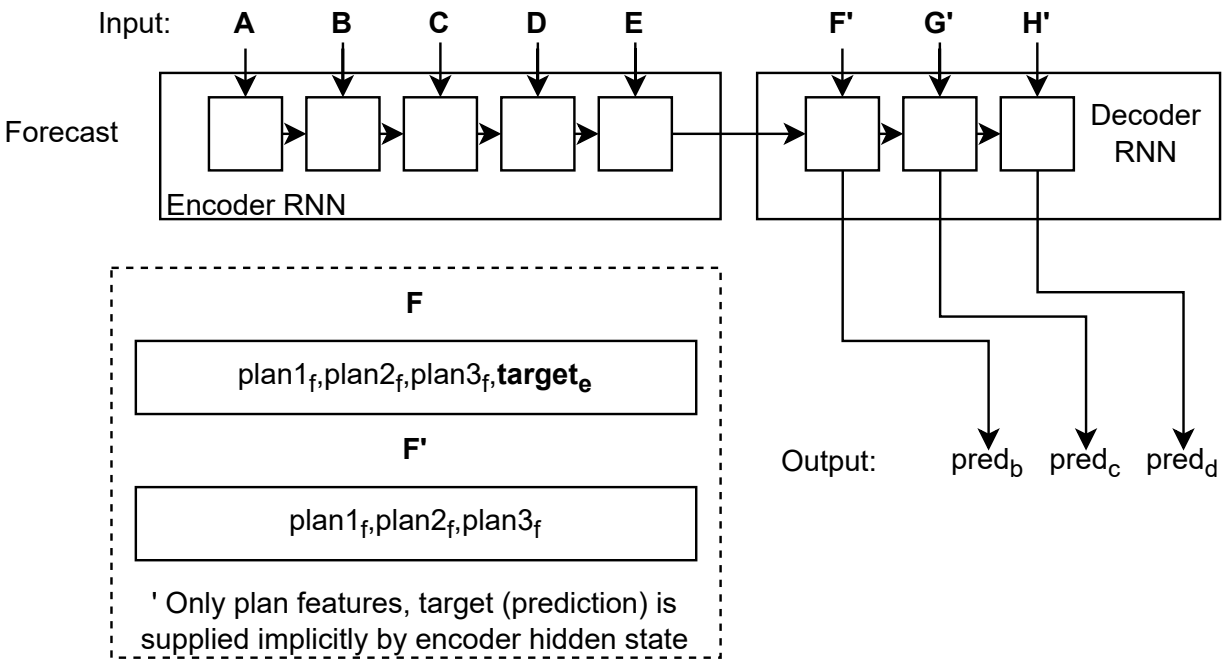

Figure 5: Three-step forecasting scenario using "Encoder-Decoder". In this scenario, we emulate that we have consumption data for every step up to, but not including, **e**, and weather forecasts for the foreseeable future beyond **e**. This model outputs multistep forecasts in one go, and is able to utilize the weather forecast for all of these through in puts to the decoder component. **Note** the difference between input **F** and input **F'**. Unlike in the other strategies, the inputs to steps further into the future do not use the previous consumption forecasts as input, nor the previous targets. This is denoted with ' instead of *.

4.5.1 Pros and Cons

Pros:

- (Possibly) able to penalize diverging forecast steps further into the future.
- (Possibly) able to balance long-term and short-term consideration of observed input in encoded representation.

Cons:

- **Not verified to work or tested at all.**
- Very high computational cost due to multiple RNN components.
- Challenging model structure to implement and tune.

5 Preprocessing and Feature Engineering

NB: in the demos we might use a hold-out test set, so you have to be able to process a new dataset in the same way you processed your own training and validation+test data in order to show us forecasts on the hold-out test set. If we do decide to use a hold-out test set, the test data will have the same form as the provided data.

You are expected check and preprocess the provided data, and to implement and try out a number of different types of feature engineering as described below.

5.1 Preprocessing

As always, you should check your dataset for anomalies such as spikes, missing data and so on. Any such anomalies must be handled before you continue to use the dataset. We have preprocessed the data to some extent, so it should

not contain any holes, but may still contain other anomalies that must be handled. This could involve removing spikes and filling the resulting holes with interpolation.

You will also have to normalize/standardize the data to avoid that the magnitudes of the different input values vary too much. Take care to normalize each feature separately, and plot your data to inspect the results. You can use an off-the-shelf scaler for this or calculate it yourself. Either way, you should store the scaler or the normalization parameters, such that you can transform your forecast back to the original input data scale. Remember to fit your scaler on the training data and not on the validation or test data.

As mentioned in Section 3, the data may exhibit both (weak) trends and (strong) seasonal patterns, that your models must handle. Typical options include diffing the time series, trying to extract seasons before prediction (and then adding them back afterwards), or building support for seasons into the model structure.

When implementing the preprocessing, you should reflect on things like the ordering of any hole-filling and the standardization, the type of standardization as well as the value range after standardization and any potential implications concerning the network architecture.

The standardization will be based on the values you observe in your provided data, you need to make sure that the implementation is able to apply the same preprocessing to the hold-out testing set provided during the demo, i.e. it should be able to clamp values of greater magnitude than you have observed and potentially (depending on your reflections on the ordering) standardize values outside the range you have standardized.

You should be able to provide both forecast plots and error measures using non-normalized data, i.e. showing plots of consumption forecasts in the original scale of the input data.

5.2 Feature Engineering

You are to implement a selection of features as described in the list below. You should run your model with different combinations of said features, noting whether it improves forecasting performance and/or training convergence. Make sure that you save some plots from applying different combinations for discussion at the demo. You should have **some** of these active in the final demo, but you are allowed to deactivate ones that you believe hurt the performance in any way. If you deactivate some features, do not remove the code.

1. Date and time features: Since it is not practical nor easy to apply infinite sequence lengths, it is often useful to inform sequence models about different time aspects when applying them to time series. Power consumption may vary greatly across the day, week and season. Implement "time_of_day", "time_of_week" and "time_of_year" features. Think about useful properties of these features.
2. Lag features: The consumption 24 hours ago may reveal something about the current state? The time of the day is identical, the weather might be similar, and the overall circumstances in society may also be similar. Perhaps the mean consumption yesterday can tell you something about today? Implement at least two lag features. **NOTE** Beware that the hold-out test set is not that big, thus you should not implement lag features that rely on data from more than two weeks prior. **NOTE 2** Lag features have to be designed and handled carefully to guarantee that no knowledge of future consumption leaks into the model input.

For the RNN model, you must at least define ***previous_y***, the consumption at the **previous** time step (**y** shifted one step forward in time). This series is required in any setup, and you must add it yourselves. This is because preprocessing and feature engineering may change it, thus, you should add it after applying these operations to the original **y**.

Caution: Take care that you do not, by accident, allow your neural network to observe the target variable.

You may also include additional features based on information outside the provided dataset, but this is not required. For instance, it might be relevant to include more weather data or temperatures for other locations within each bidding area.

Write code for performing the described feature engineering and make sure it is easy to activate and deactivate for new runs, both for training and evaluation.

6 Visualization

Your implementation should cover four different types of visualization:

1. The **training progress** visualization must represent the development of the training loss as well as validation loss across the training epochs. You can create your own plots for this, or use Tensorboard or similar tools.
2. The forecast visualization should be a line plot which displays observed and forecasted consumption on the y axis (unscaled) and time on the x axis. The plot must show the measured consumption over the input part (sequence length), and both forecasts and target (measured consumption) in the forecast window. In other words, if you model uses 24 hours of historic data to predict 24 hours into the future, your plot will show 48 hours, where the first half contains historic consumption, and the second half shows both the forecast and the target (which your model does not know about, of course). Furthermore, your implementation should be made such that you are able to print numerous forecasts at the same time, either live or by dumping the plots to files that can easily be retrieved. **Note** that these plots will be the main way of evaluating your models qualitatively, since the losses and metric values are highly dependent on the sequence model you use. Thus, you should spend some time making sure this part works as intended and produces clear plots.
3. An error plot showing the mean and standard deviation of the error for each hour of the forecast horizon over the test set. This plot will have forecast horizon (1-24 hours) on the x axis and absolute error on the y axis.
4. A summary bar plot that compares the different models you have trained using metrics of your choice (e.g. test set RMSE, MAE, MAPE or similar).

7 Model tuning

You are not expected to spend extreme amounts of time on tuning your model for this task, but you should try out different sizes for each architecture type (i.e. vary the number of kernels, LSTM units, layers and so on) and adjust training length to sensible values (keep an eye on how the validation error progresses through training).

You **must** add regularization to your models. You should try a couple of different "rate"-configurations, e.g. dropout rate, regularization rate and/or learning rate.

As described in Section 8 below, you must tune your models sufficiently to get decent (but not necessarily great) results in order to get a full score for each model.

8 Deliverables

The subsystems and procedures listed below will be demonstrated by running the system multiple times and by discussing your code with an instructor or assistant. You should show up to the demo with all models **pretrained**. We will not set aside time for completely training new models, but you may be asked to train a new model for a couple of epochs to demonstrate visualization of the learning. You must have a fully functioning pipeline for applying the same preprocessing and feature engineering you applied to your training set to the hold-out test set used in the demo. Throughout the demo you will have to explain your code, results from your tuning and reason about architecture choices.

As described above, your models should forecast 24 hours into the future, with the constraint that you cannot use measured consumption (i.e. the "previous_y" described in Section 5 or any other lagged features based on measured consumption) in the forecast window. Make sure that you truly understand what this means by looking at the different sequence models and multistep forecasting. The demo instructor will make sure that this constraint is satisfied, and failure to comply will result in the deduction of points.

The Point Breakdown is as follows:

1. Feature engineering: Implemented features and normalization (including the inverse transform to get predictions on the scale of the original data), correct application on data sets (including hold-out set) and discussion of results during tuning: **1 point**.
2. 24-hour forecasts using an RNN (LSTM or GRU), CNN, transformer or feed-forward regression model. The model should be able to make reasonably good predictions the majority of the time.
    - Training progress visualization with verification that your model learns *something*: Loss decreases over time, and the model shows decent performance with the first few forecast steps being in the same ballpark as the last measured consumption: **1 point**.

- Forecast visualization (unscaled): **1 point**.
- Error plot over forecast horizon for test set: **1 point**.

3. For each of the two remaining model architectures:
   (a) 24-hour forecasts using the given model. The same requirements apply as above.
   - Training progress visualization, forecast visualization (unscaled) and error plot over forecast horizon for test set: **1 point** (per model).
   - Decent model performance: **1 point** (per model).
4. Comparison of the models from points 2-4 above (bar chart or similar) with discussion about results: **1 point**.
5. Demonstration and comparison of model trained on another bidding area with one of the models from points 2-4 above applied to this bidding area: **1 point**.

You can create your models in any order. If you choose a basic feed-forward architecture for one of your models, you may want to create this model first and use as a baseline for the other models (it may turn out to be harder than expected to beat a good baseline model).

As shown in the point breakdown, the first model may give you at most 3 points, whereas the subsequent models are worth at most 2 points each.

8.1 The Demonstration Session

To ensure a smooth demonstration, you should have the following readily available during your session:

1. Pre-trained models that can be loaded during the demo.
   - **NB:** Remember that you need to be able to process the hold-out test data in the same way you processed the training data used in the pre-trained models.
   - Your setup should be able to display multiple forecasts in the test data on request. You may be asked "First show me the forecast when given the first part of the test data (respecting your sequence length), then shift the start point by 24 hours and produce another forecast" etc.
2. Plots/notes from the testing of different feature combinations, and from the general tuning of your model. Be prepared to discuss which changes you have made to your feature composition, architecture, learning rate, regularization, dropout, loss etc.

8.2 Checklist and Tips

- Make sure your preprocessing and feature engineering pipeline will be able to handle the hold-out test set.
- In general, start off with the "**n** in, **1** out" sequence strategy before considering something more exotic. This strategy should be capable of giving you full score on the project and is the only strategy.
  - If you struggle with forecasts diverging after x steps, you may want to try reducing learning rates and discourage the model from relying too much on the very last "previous_y". The latter can for instance be achieved by adding a probability of replacing said value with random uniform noise, or adding some Gaussian noise to it.
  - It can also be useful to set up your code in a way that allows you to train it for x more epochs, as the diverging may vary between epochs.
- Make sure you have not included the target of the **current step** as a feature!
  - For time steps **before** the forecast window, you should include **"previous_y"** as a feature. This is a simple shift operation, e.g. df['y'].shift(1) in Pandas(Python).
  - If you have added the target as a feature, you will typically get more or less perfect results straight away, so it should be easy to debug.
- You are not allowed to use provided "previous_y" values in the forecast window.

- If you are using a sequence strategy that outputs fewer forecast steps than the forecast window length for each run, you have to replace "previous_y" in the forecast window with your predicted values when forecasting the following step(s). See Section 4 for deeper understanding of the procedure.

9 Important Practical Details

WARNING: Failure to properly explain ANY portion of your code (or to convince the reviewer that you wrote the code) can result in the loss of 3 to 10 points, depending upon the seriousness of the situation. This is an individual exercise in programming, not in downloading nor copying.

A zip file containing your commented code must be uploaded to BLACKBOARD before or (immediately) after your demonstration. You will not get explicit credit for the code, but it is crucial that we have the code online in the event that you decide to register a formal complaint about your grade (for the entire course).

The 10 points for this project are 10 of the 30 project points that are available for the entire semester. To pass this particular project, you must receive at least 3 points.

10 Online Resources

The course book has an excellent description of the motivation behind the LSTM and its inner workings. It may nonetheless be a good idea to also consult other sources to get a clear mental picture of the algorithm. In this case Christopher Olah's blog post on LSTM is very accessible.

Additionally, this blog post gives some hints on how to construct and train an LSTM for timeseries forecasting in PyTorch. The presented code contains a number of errors and will not run as-is, but the concepts and explanations are still well worth a read.