Deep Learning Project 3

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 third assignment in IT3030. The task at hand is to forecast transmission system imbalance using a Recurrent Neural Network (RNN) based model. In this work, you will get hands-on experience with applying sequence models to a real-world application using real-world data. The main goal of the assignment is to be a more practical oriented assignment that might better illustrate how deep learning may be applied in the industry.

The rules are as follows:

- The task must be solved individually
- The main part of the model **must** be RNN-based. We suggest you use LSTM or GRU units.
- 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-based sequence models, i.e. your code has to manage and interact with RNN units directly.
  - 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 40 points; the rules for the scoring are listed in Section 8.
- The deadline for the **demonstration** of this project is week 17 (April 25-29). Information regarding the exact 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 would probably make heavy use of Scikit-learn, 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 these 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.). 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. Failure to balance the power system leads to frequency deviations and, in extreme cases, outages. It is the responsibility of the Transmission System Operator (TSO) to ensure that the system is balanced at all times. In Norway, Statnett is the TSO and is responsible for keeping the frequency between 49.9 and 50.1 Hz.

The power system is balanced through a number of steps involving different markets. In the Nordic countries, producers and consumers settle prices and volumes for the next day on the Nord Pool day-ahead market, and may subsequently trade on the Intra-day market to restore balance whenever their plans or expectations change (e.g.

the weather forecast changes). As we move closer to real time, a series of reserve markets ensure that balance is maintained at all time and in the presence of unexpected fluctuations in production, consumption or transmission.

Your task in this project is to train a deep neural network to forecast the need for tertiary reserves, i.e. to forecast the remaining imbalance after the day-ahead and Intra-day markets are cleared. A similar system is currently under development for use in the upcoming Automated Nordic mFRR energy activation market.

The data available for training and running your model includes production plans and historical imbalance data. You should use this data to construct datasets with appropriate features for your model, typically including estimated imbalances from the immediate past, as well as historical and future grid plans to forecast future imbalance. 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.

2 The Data

The data are provided in a CSV (Comma-separated values) file. The first line contains the header of each time series in the file and describes the data stored within them. The following list provides an extended explanation of the different time series:

- **start_time:** The timestamp of each datum.
- **hydro:** The **planned** reservoir hydropower production at the time step.
- **micro:** The **planned** small-scale hydropower production at the time step.
- **river:** The **planned** run-of-river hydropower production at the time step.
- **thermal:** The **planned** thermal power plant production at the time step.
- **wind:** The **planned** wind power plant production at the time step.
- **total:** The total **planned** production at the time step. Equal to the sum of the listed "planned production" features.
- **sys_reg:** The **planned** "system regulation" at the time step: activation of balancing services to accommodate specific needs (e.g. bottlenecks) in the power system.
- **flow:** The **planned** total power flow in or out of the current area of the grid at the time step. At any moment, power flows between connected bidding areas. This value is the net power flow when considering export to be negative and import to be positive relative to the current area.
- **y: The target variable that you train your network to predict**. The **estimated** "open loop" power grid imbalance at the time step. Think of this as the imbalance, per area, that would have occurred if balancing services were not activated. It is not possible to observe this value directly, thus it has to be estimated using other measurements of the power grid.

It may often be beneficial to define so-called *lagged variables* as part of a time series forecasting system. In this project, since you will train an RNN, you must at least define ***previous_y***, the estimated power grid imbalance 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.

3 The Forecasting Task

Your task is to develop a model that can forecast the imbalance with a 5- or 15-minute resolution, **two hours into the future**. In other words, in the case of 15-minute resolution, for any point in time $t = t_i$ and given historical imbalances from $t = t_0$ to $t = t_i$ as well as plans for $t = t_0$ to $t = t_{i+8}$, your model should produce 8 forecasts covering the period $t = t_{i+1}$ to $t = t_{i+8}$.

The dataset provided for this task contains time series of plans and imbalances for the Norwegian bidding area NO1.

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* or *step-by-step* strategy.

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

2. Given inputs, including values for $\mathbf{y}$, for any number of timesteps up to and including t_i, and, if available, plans 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.

For 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, potentially, moving on to other strategies.

Again, make sure your model does not use any knowledge of estimated imbalances 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 144 steps, each of which will contain plan features and previous imbalance estimates for 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.

4.2 n in, 1 out

In this strategy, each input is **n** long, but the model only produces a single forecast for the next time step. E.g. given 5 min resolution, you may have sequence lengths of 144 (the preceding 12 hours) and forecast 5 minutes 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, and add **plan features** for the next time step and then **switch out** the previous imbalance estimate with your **forecast**. See Figure 1 and the following pseudocode for more details.

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
model_input = x[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
  model_input = x[start_ind:start_ind+seq_len]
  # Replace prev_y with prev forecast
  model_input[seq_len-1,prev_y_ind] = pred
```

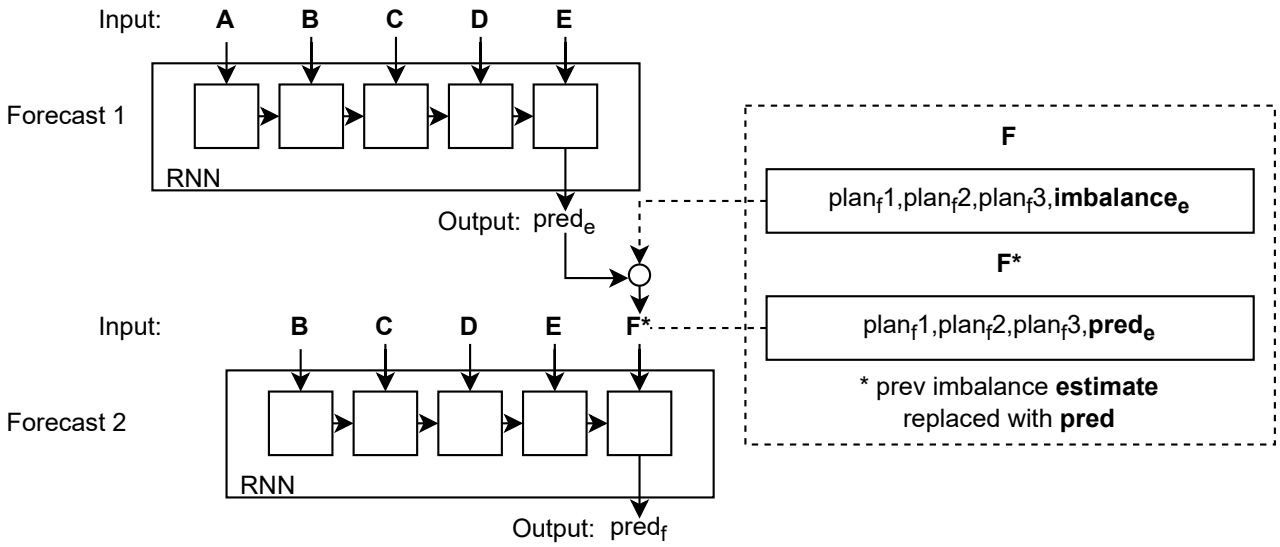


Figure 1: Two-step forecasting scenario using "**n** in, **1** out". In this scenario, we emulate that we have imbalance estimates for every step up to, but not including, **E**, and plan features for the foreseeable future beyond **E**. Note the difference between **F** and **F***. When forecasting the imbalance in time step **F**, we have to replace the estimate of the imbalance in the previous time step with our previous forecast (time step **E**).

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

4.2.2 Pros and Cons

Pros:

- **Verified to perform well**.
- 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, see Section 7.
- 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 2 for more details.

4.3.1 Pros and Cons

Pros:

- Lighter computational load.

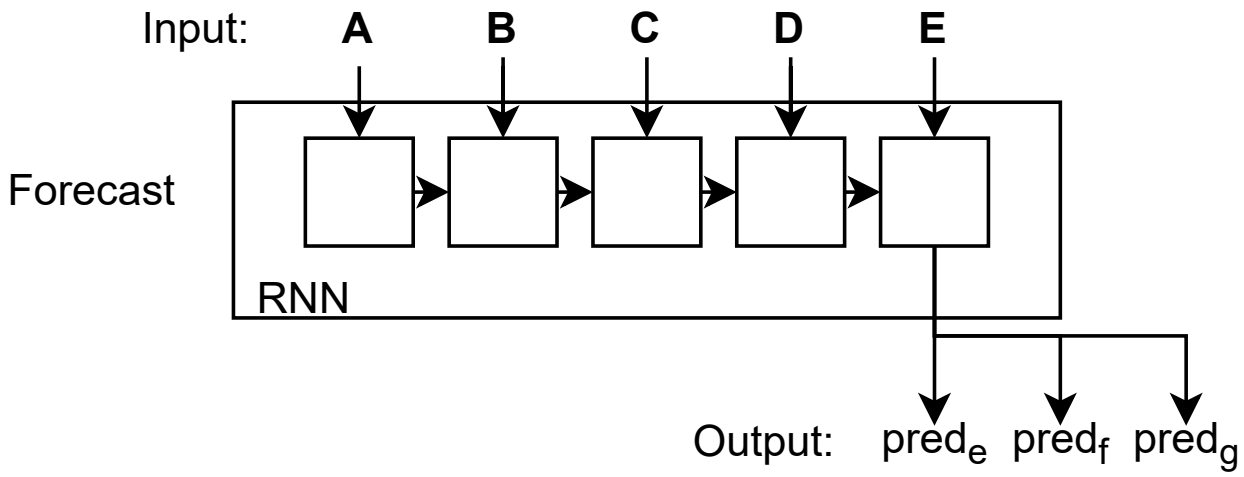


Figure 2: Three-step forecasting scenario using "**n** in, **m** out". In this scenario, we emulate that we have imbalance estimates for every step up to, but not including, **E**, and plan features for the foreseeable future beyond **E**. However, we are not able to use the plan features for steps beyond **E** with the more straight forward implementations of this strategy. The model outputs three forecasting steps in one go.

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

Cons:

- **Not verified to work**, smaller tests displayed trouble with overly conservative forecasting that stayed close to the previous provided imbalance estimate.
- Harder to model dynamics between forecasts far into the future and providing the model with plans for these.

4.4 n in, n out

This setup outputs one prediction for every input by propagating every "hidden states" of the RNN further 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 3 for more details.

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 previous provided imbalance estimate.
- 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.

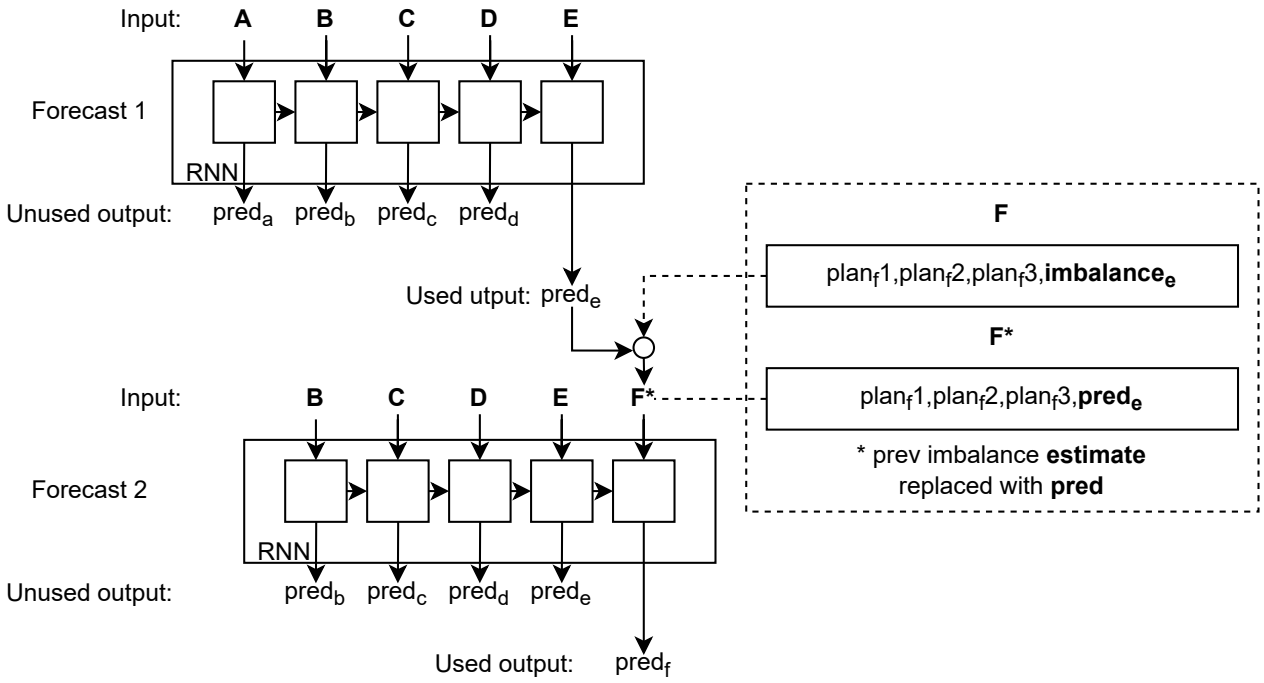


Figure 3: Two-step forecasting scenario using "**n** in, **n** out". In this scenario, we emulate that we have imbalance estimates for every step up to, but not including, **E**, and plan features for the foreseeable future beyond **E**. Note the difference between **F** and **F***. This model outputs a forecast for every single input, but we only propagate the very last forecast step when doing multistep forecasting. When forecasting the imbalance in time step **F**, we have to replace the estimate of the imbalance in the previous time step with our previous forecast (time step **E**).

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 plans for each future time step it is to forecast, ultimately propagating values to produce multiple forecasted time steps in one run. See Figure 4 for more details.

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 will 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 forecast on the hold-out test set. The test data will have the same form as the provided data.

As this is a more practically oriented project, the input time series have not been completely cleaned, nor prepared for neural network processing. You are also expected to implement and try out a number of different types of feature engineering.

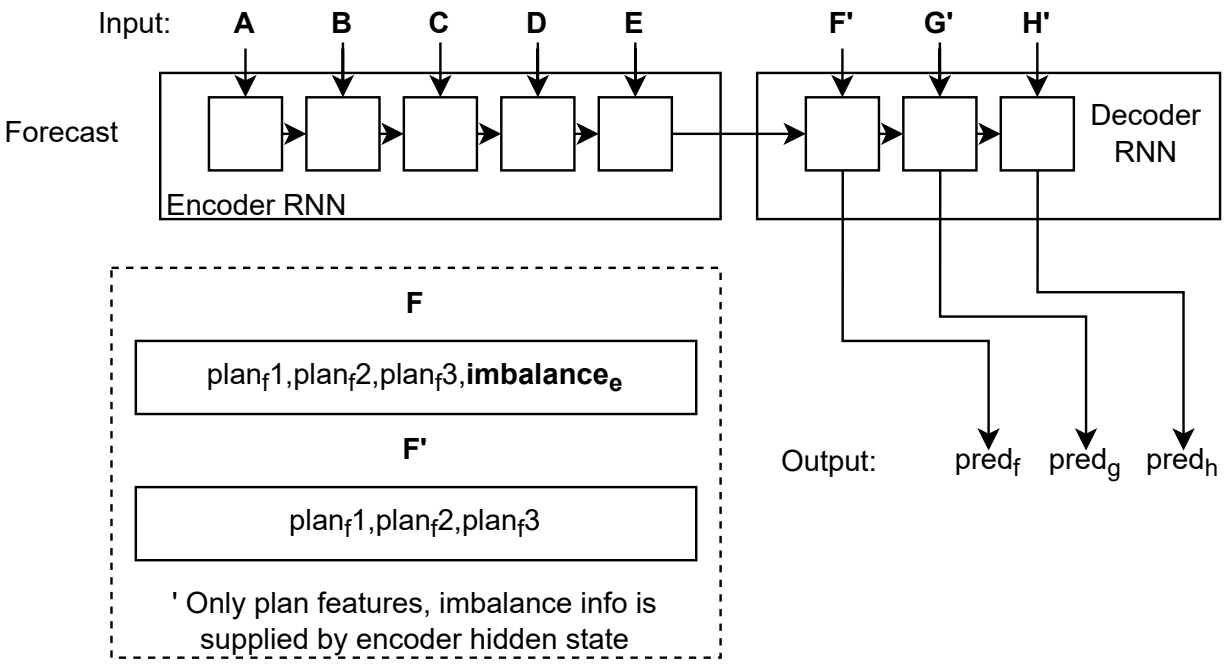


Figure 4: Three-step forecasting scenario using "Encoder-Decoder". In this scenario, we emulate that we have imbalance estimates for every step up to, but not including, **E**, and plan features for the foreseeable future beyond **E**. This model outputs multistep forecasts in one go, and is able to utilize plan features for all of these through inputs to the decoder component. **Note** the difference between **F** and **F'**. Unlike in the other strategies, the inputs to steps further into the future do not contain forecasts of imbalance, the imbalance component is simply removed all together. This is denoted with **'** instead of *****.

5.1 Preprocessing

The estimated imbalance, i.e. the target, contains some spikes in value magnitude, likely due to errors in the input data used to estimate imbalance. These make out a small part of the total data amount, but you still probably want to replace these rows with more likely imbalance values. A simple approach is to clamp y based on upper and lower quantiles of the data.

You will also have to normalize/standardize the data to avoid that the magnitudes of the different input values vary too much.

1. Clamp the values of the target series to exclude noisy spikes in magnitude. You are at most allowed to clamp a total of 1% of the values.

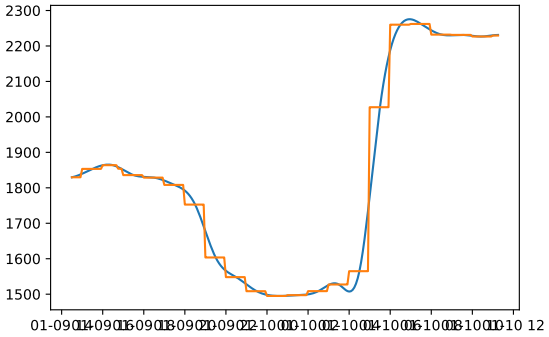
2. Write code for normalizing/standardizing input data.

When implementing the preprocessing, you should reflect on things like the ordering of the clamping and the standardization, the type of standardization as well as the value range after standardization and any potential implications concerning the network architecture. Clamping and 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(depends on your reflections on the ordering) standardize values outside the range you have standardized.

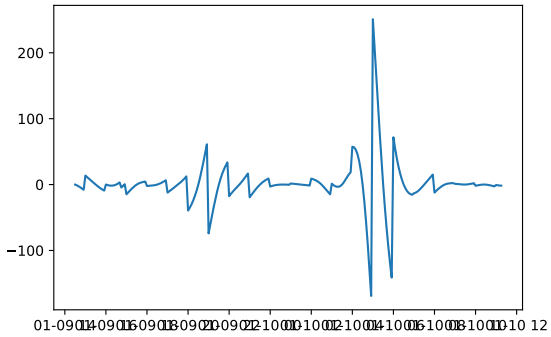
5.2 Feature Engineering

5.2.1 Simple Features

You are to implement a selection of features as described in the list below. You have to 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



(a) Plot of the stepwise production and a fitted interpolation between its midpoints.



(b) Plot of corresponding difference between stepwise production and interpolation.

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 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 power imbalance 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 state of the grid may also be similar. Perhaps the mean imbalance yesterday can tell you something about today?. Implement at least two lag features. **NB** Beware that the hold-out test set is not that big, thus you should not implement lag features reliant on data from more than a week prior.

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 an average temperature forecast per bidding area, or the forecasted power consumption (available from Nord Pool).

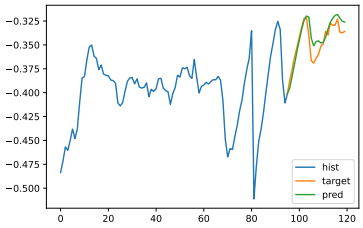
If you choose to include this kind of additional features, keep in mind that you should only include information that was available at the time of the forecast. In other words, you may include weather or consumption *forecasts*, but you should not include *observed weather* or *actual consumption*.

5.2.2 Altered forecasting: Avoid Structural Imbalance

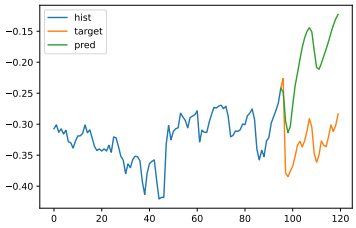
Part of the imbalance is caused by the structural difference between the continuously, slowly changing consumption in the power system vs the discontinuously, abruptly changing power production of regulated power plants at the transition between market time units (hours, or quarters, depending on the market). Knowing this, we may be able to explain "away" some variation in the target.

The demand curve is not part of the given dataset or known in advance of the prediction, but we can approximate it by fitting an interpolation using the midpoints of each step in the production, as shown in Figure 5a. In other words, if you subtract the sum of planned flow and production from a smoothed (e.g. by spline interpolation) version of the same sum, you end up with an estimate of the "structural imbalance". Finally, you can subtract this from the estimated imbalance, that is to say, the target. Ideally, this process will remove much of the explainable imbalance without introducing too much bias, the remaining variation in the target should look quite different from before. **Feel free to derive additional features from this approach and use them in the main task.**

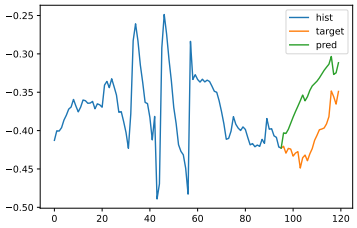
1. Write code for performing the described feature engineering and make sure it is easy to activate and deactivate for new runs, both for training and evaluation. **Note:** you are not expected to spend much time on tuning your model for this task. You are free to use the same architecture as in the main task, but do try a couple of different "rate"-configurations, e.g. dropout rate, regularization rate and/or learning rate. You are free to use the interpolation of your choice, but keep in mind that the demand curve typically is quite smooth, and you may have to argue for your selection during the demo.



(a) Good fit of oscillation, matches trend and variation.



(b) Decent fit of oscillation, misses trend but captures variation.



(c) Subpar fit of increasing imbalance, pre-mature upwards trend.

Figure 6

6 Visualization

Your implementation should cover two different types of visualization: the training and multistep forecasting.

1. The training visualization must represent the development of the training loss as well as validation loss across the training epochs.

2. The forecast visualization must show the imbalance across the input part (sequence length) and how the forecast differs from the actual values in the forecast window. 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 produce clear plots.

7 What to Expect from a Good Model

Multistep forecasting has a tendency to struggle with divergence after a couple of steps, especially when there are no abrupt events that are known to cause immediate responses. One reason for this is reinforcement of existing bias, e.g. if one iteration of the model tends to overshoot, the forecast of step one may be slightly higher than the target. The subsequent forecast of step two will perhaps perceive an upwards trend, which in turn may be further reinforced by the general trend of overshooting. Unless there are some strong signals that can bring it back down, the forecast can easily diverge this way. In our case, this is further complicated since the model has to learn a significant oscillation that occurs every morning and evening, and is accustomed to rapid increases and decreases.

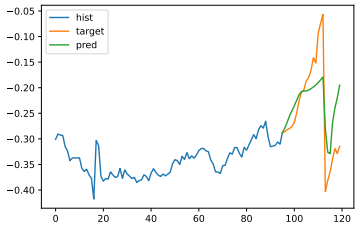
It has been observed that it is often easy to train a model that forecasts oscillation well, but struggles in flatter and smoother parts. Your model performance will therefore mainly be evaluated qualitatively, i.e. can we see that it is able to capture some trends in x out of y plots? Is it able to capture the easier parts?

Look at Figure 6 and Figure 7 for some idea of what a good model will look like. Your model will probably perform well for some patterns, reproducing the characteristic oscillation of the imbalance, but not so much on other patterns. Figure 8 gives an idea of how an under-trained model will look, if you get plots like these you probably have to train longer, increase the learning rate or adjust other hyperparameters.

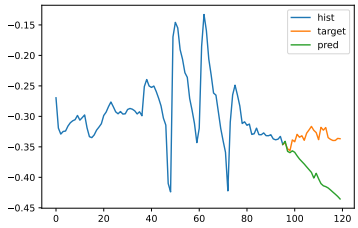
8 Deliverables

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 at least two **pretrained** models, one for the main task and one for the altered task, see Section 5.2.2. We will not set aside time for completely training new models, but you will 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.

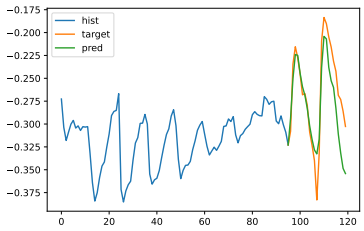
If possible, it is encouraged that you train your model on the given time resolution, which is one time step for each 5 minutes. However, this is only achievable if you do have access to GPU. Every student has been given 50 USD



(a) Good fit of start of oscillation, matches trend and variation.

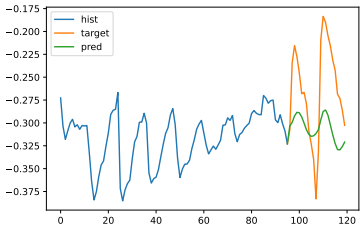


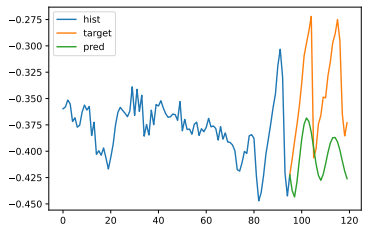
(b) Bad fit of stable imbalance, completely misses trend.



(c) Good fit of oscillation, matches trend and variation.

Figure 7





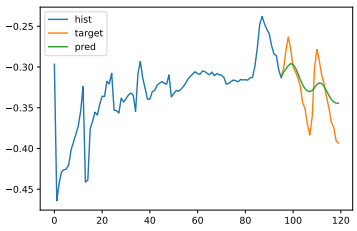


Figure 8: Models that are under-trained typically produce forecasts like these of oscillating imbalance.

worth of credits at Google Cloud, but should you for any reason encounter problems that impedes your productivity etc, you are allowed to simplify the problem by down sampling the data to 15 min resolution. By down sampling, the problem becomes tractable on CPUs. The simplified problem on CPU may be a bit easier to tune, but comes with the cost of longer training cycles. If you have down sampled, you are still to target a 2-hour forecast.

Regardless of time resolution, you will be asked to forecast two hours into the future, with the constraint that you cannot use "previous_y" in the forecast window. Make sure that you truly understand what this mean 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. Preprocessing: Code, correct application on your own validation set and the hold-out test set. **(3 points)**.
2. Feature engineering: Implemented features, correct application on sets and discussion of results during tuning. **(4 points)**.
3. Visualization: Plot of loss after training for a couple of epochs. Plot of two-hour forecast, depicting imbalance in time-steps provided to the model for the very first forecast step, true imbalance in forecast window and forecasted values. **(3 points)**.
4. Verification that your model learns *something*: Loss decreases over time, and the first forecast step is in the same ballpark as the last provided historical imbalance. **(5 points)**.
5. Main forecasting task: Model performance on hold-out test set and/or own validation set. Hold-out set is weighted more. Examine Section 7 for an idea of our expectations. **(Total: 17 points, see breakdown below)**.
    - Forecasting quality: We will use the hold-out set unless we encounter problems. In the case your code is not able to process the hold-out set, a penalty may incur, and you will have to use your own validation set instead. **(10 points)**
    - Discussion of tuning process and historical results. **(3 points)**.
    - Reasoning about architecture choices and model validity. **(4 points)**.
6. Altered forecasting task as described in Section 5.2.2: Implementation, discussion of results on own validation set and hold-out set. **(8 points)**.

8.1 The Demonstration Session

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

1. Two pre-trained models that can be loaded during the demo. One for the main task and one for the altered task.
    - **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.
2. Plots/notes from the testing of different feature combinations, Section 5.2.1, 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

- Remember that the main part of the model **must** be RNN-based.
- Make sure your preprocessing and feature engineering pipeline will be able to handle the hold-out test set. Validate this through the provided validation set.
- In general, start off with the "**n** in, **1** out" sequence strategy before considering something more exotic. This strategy is perfectly capable of giving you full score and is the only strategy that is verified to do so.
    - 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 5 to 40 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 40 total points for this project are 40 of the 100 points that are available for the entire semester.

10 Online Resources

If you choose to try out LSTMs for this task, the course book has an excellent description of the motivation behind this algorithm 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.