Master's Thesis

Energy Price Forecasting with Uncertainty Estimation

Sneha Senthil

Examiners: Prof. Dr. Hannah Bast, Prof. Dr. Frank Hutter

Advisers: Matthias Hertel, Oliver Mey

University of Freiburg
Faculty of Engineering
Department of Computer Science
Chair for Algorithms and Data Structures

Fraunhofer IIS/EAS
Fraunhofer Institute for Integrated Circuits
Division Engineering of Adaptive Systems
Dresden

September 16th, 2022

Writing Period

04. 04. 2022 – 16. 09. 2022

Examiner

Prof. Dr. Hannah Bast

Second Examiner

Prof. Dr. Frank Hutter

Advisers

Matthias Hertel, Oliver Mey

Declaration

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Freiburg, 16/09/2022	Sneha
Place, Date	Signature

Abstract

Forecasting is the task of predicting future values, taking into account historical data. We forecast energy prices taking into consideration load, generation, historical prices and weather data. This thesis aims to solve this task using a Transformer model. While Transformers are typically used for NLP tasks, there have recently been some successful applications of Transformers for forecasting. In addition to predicting the prices, we try to estimate the uncertainty of this prediction by probabilistic forecasting. We experiment with different types of distributions and determine which distribution would be the best for probabilistic forecasting. We also study how probabilistic forecasting affects the model's results compared to deterministic forecasting. We find that Transformers outperform other deep learning models. Additionally, probabilistic forecasting helps improve the accuracy in some cases and is helpful in understanding the uncertainty in the model's prediction.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem Statement

We intend to solve the following electricity price forecasting problem: Given the last n hours of data (which includes weather data, load, generation and prices), the model should be able to predict the hourly energy prices of the following day, along with the probability distribution of these predictions.

1.2 Motivation

There is an increasing share of renewable energies. Weather-dependent fluctuations in the amount of energy generated arise, which are directly reflected in the traded price at electricity stock exchanges. Hence, predictions of the energy price in connection with flexible electricity tariffs can help to shift peak loads to times of high availability of renewable electric energy and thus reduce carbon dioxide emissions. We attempt to predict the prices via time series forecasting.
Many models have been developed over time to tackle the challenge of time series forecasting. Among the deep learning models, RNNs and specifically LSTMs have shown the best results (Khodabakhsh et al., 2020). However, recently some experiments have been conducted that show Transformers are able to surpass RNNs for forecasting (Wu et al., 2020 and Zhou et al., 2022). Transformers are the recent

state-of-the art models for natural language processing tasks, replacing RNNs. These models can be adapted to time series data as well. In light of these findings, we hypothesize that Transformers could be useful for our energy data.
Additionally, probabilistic forecasting would help quantify the level of uncertainty of the predictions. This can help us understand how confident the model is in its predictions.

1.3 Contributions

In this thesis:

- We develop a time series forecasting model inspired by the Transformer architecture (Vaswani et al., 2017) to forecast energy prices.
- We compare the results of Transformers with other deep learning models.
- We check if the models benefit from data augmentation.
- We check if the Transformer model benefits from adding additional features: date, month and year encoding and addition of next day weather forecast as a feature.
- We convert the deterministic models to probabilistic models and compare how performance of the model changes.

1.4 Chapter Overview

The next chapter, Chapter 2, gives an overview of the existing research on time series forecasting using deep learning models, as well research on probabilistic forecasting.

In chapter 3, we provide the background information on artificial neural networks, recurrent neural networks and Transformers, which are all used in our experiments.

Next, in Chapter 4, we detail the datasets used, the data pre-processing steps, the architecture of the models and describe the training process.

In Chapter 5, we introduce the evaluation metrics for deterministic and probabilistic models. The chapter then presents the results of our deep learning models and some prediction graphs to visualize the model predictions.

Finally, Chapter 6 summarizes the findings of the thesis and suggests some possible future works.

2 Related Work

The topic of time series forecasting in general is a widely tackled issue. This section gives an overview of existing works on time series forecasting in general, as well as those related to energy prices, and the deep learning methods associated with it. Section 2.1 introduces the statistical methods. Section 2.2 continues with the deep learning methods which includes artificial neural networks, recurrent neural networks and convolution neural networks. Finally, Section 2.3 covers the existing works on transformer models .

2.1 Statistical Methods

A well known statistical method is ARIMA (AutoRegressive Integrated Moving Average) (G. E. P. Box & G. M. Jenkins, 1968). ARIMA is a linear regression model. ARIMA performs well over short-term forecasts. However, it has shown to perform poorly for long term forecasting and requires expert knowledge to manually select trend and other components. A time series with non-linear dependence would not be forecasted well by ARIMA.

2.2 Deep Learning Methods

2.2.1 Artificial Neural Network

Research into neural networks show that Artificial Neural Networks (ANNs) are capable of pattern classification and pattern recognition. Guoqiang Zhang et al. (1998) outline the advantage of using ANNs for forecasting. Few assumptions need to be made about the data at hand. ANNs learn from the data which features are relevant and use these features for future predictions. Since forecasting is about predicting future behaviour from historical data, it is an ideal field for neural networks to be applied. A huge advantage is that ANNs are non-linear.

The first application of ANNs for forecasting was by Hu (1964). He used a Widrow's adaptive linear network for weather forecasting. At the time, there was no training algorithm for multi-layer networks. Hence, the research at that time was limited. The research into ANNs for forecasting improved with the use of backpropagation. In 1986, the backpropagation algorithm was invented (Rumelhart et al., 1986).

Wilson and Sharda (1994) used neural networks for bankruptcy prediction. They compared the performance of ANNs with multivariate discriminant analysis and found that ANN outperformed it. The authors concluded that ANNs are a robust and promising approach for classification and encourage further research into this field.

Pino R et al. (2008) used ANNs to predict the next day price of electricity in the Spanish energy market. The ANNs performed better than ARIMA models, especially for weekends and holidays. The paper focuses on univariate forecasting, with historical prices being the only input features. The data ranged from 1996 to 2004.

Ioannis P.Panapakidis et al. (2016) also used ANNs for short term load and energy prices forecasting. They highlight that load forecasting reported an error rate of below 3%. For the task of prices forecasting, the model reported an error rate close to 20%. The issue with price forecasting is the highly volatile market prices. Load mainly depends on weather conditions and seasonal effects. However, prices can be influenced by many diverse and unpredictable factors and these might not be available to engineers. They also mention that in order to predict prices accurately, knowledge of historical prices alone is not enough. The input data should include natural gas prices, prices of other energy markets and renewable generation capacity.

2.2.2 Recurrent Neural Network and Long Short-Term Memory Network

RNNs can be used for sequence modeling and has proven to be useful for many natural language processing tasks.

S. Anbazhagan and N. Kumarappan (2012) proposed an RNN for the day ahead deregulated electricity market price forecasting using the Elman network. The prediction of a feature depends on earlier features and the time of occurrence of the feature. These characteristics can be captured by an RNN. In the Elman network, the outputs of the hidden layer are allowed to feedback into itself through a buffer layer. The weights from the hidden layer to the buffer layer are constant. All other connections are feed-forward. This model was compared with ARIMA, neural network, and many other models and the RNN model outperformed them. The training time of the RNN was quite large and does depend on the training data size and number of parameters. The authors talked about research for a better feature selection algorithm for different power markets.

David Salina et al. (2020) proposed an auto-regressive RNN model for probabilistic forecasting called DeepAR. Two distribution choices were considered during the experiments: Gaussian likelihood for real-valued data and negative binomial likelihood for positive count data. The authors use five datasets for the experiments. They found that forecasting using this approach drastically improved forecast accuracy compared to other state of the art forecasting methods. The DeepAR model was able to learn seasonality and uncertainty growth over time from the data. However, this model is only applicable to medium-sized datasets with only a few hundred rows of data. It is likely that the model would suffer from exploding or vanishing gradients with a longer range of time series.

Petnehází and Gábor (2018) compared the performance of Long short-term memory (LSTM) and Gated Recurrent Unit (GRU) networks. The network has a layer of LSTM/GRU consisting of 32 units, followed by a single unit dense layer. They found that the performances of the GRU and LSTM models, in forecasting, were quite similar. However, the models were efficient for only one-step forecasts. They used an iterative method for multi-step forecasting, which they admit to not be efficient. The authors proposed the use of bootstrapping, however this is computationally expensive. They also recognize that the lack of availability of real world datasets is an issue for the study of deep learning methods of time series forecasting.

J.F.Torres et al. (2022) used a deep LSTM network for the Spanish electricity consumption forecasting.

2.2.3 Convolution Neural Network

While RNNs and LSTMs are considered to be state of the art for time series forecasting, some Convolution Neural Network (CNN) models proved to be effective depending on the problem at hand.

Anastasia Borovykh et al. (2017) used a CNN for financial time series forecasting. They used multiple layers of diluted convolutions, in which the filter skips certain elements in the input. This allowed the model to learn the trends and relations between the data. The advantage of using a CNN is that the training and prediction is more efficient since the number of trainable parameters is small. It is a simple model that is easy to interpret, however there is still scope for improvement. There is a trade-off between model complexity and overfitting. A larger number of layers can result in a larger receptive field and thus an ability to learn more about the non-linearities, but can cause overfitting.

Khan Z et al. (2020) use a classic CNN for short-term price forecasting. When compared to a Multi Layer Perceptron (MLP) model, the CNN has better results. However, it is not clear which data is being used and which features are used for training. Additionally, there are no evaluation metrics specified for the CNN and MLP, so it is not clear how well the CNN performs. It is clear by just graphical figures that CNN was better at forecasting than the utilized MLP.

2.3 Transformers

The recently introduced Transformer architecture proposed by Vaswani et al. (2017) proved to be very efficient for sequence-to-sequence problems.

Li et al. (2019) pointed out some issues with using transformers for time-series forecasting. There is a memory bottleneck and the space complexity grows quadratically with the sequence length. So this makes it hard to model a long time series. Additionally, they claim that transformers are insensitive to local context. To solve these issues, the authors propose a self-attention with convolution to better capture local context. They proposed a LogSparse Transformer to improve the space complexity to $O(L(logL)^2)$, where L denotes the sequence length.

Zerveas et al. (2020) proposed a transformer encoder model for unsupervised representation learning of multivariate time series data. They generalized the use of transformers such that with minor modifications, it can be used for a variety of tasks. They were inspired by the good results of unsupervised pre-training of transformer models in NLP. The model performed extremely well for regression and classification and the model did not take too much time to train. The model was evaluated over several public datasets. It performs well even for datasets that have only a few hundred training samples.

Zhou et al. (2022) stated that transformers are computationally expensive and cannot capture an overall trend of time series. They proposed to combine the transformer with a seasonal-trend decomposition method which captures the global profile of time series. The Transformer captures more detailed structures. They proposed a Frequency Enhanced Decomposition Transformer (FEDformer) which exploits the fact that time series have a sparse representation in well-known basis (for example, Fourier Transform). The model is more effective than a standard Transformer.

Wu et al. (2020) used a basic transformer encoder-decoder architecture for influenza like illness forecasting. The model has 4 encoder layers and 4 decoder layers. Positional encoding with sine and cosine function was used. Compared to other deep learning methods, this technique could learn complex dependencies in the data since it used self-attention. As it used a generic transformer architecture, it can be used for other non-linear systems as well. The authors suggest this model to be used for spatio-temporal data They show that the attention mechanism is able to learn complex patterns in the time series data, since the transformer and a Seq2Seq with attention models outperformed a plain LSTM model.

2.4 Probabilistic Forecasting

Salinas et al. (2020) use a forecasting method based on autoregressive neural networks, incorporating a negative Binomial likelihood. The authors considered two distributions for the experiments: Gaussian likelihood for real-valued data and negative-binomial likelihood for positive count data. The method works well on a variety of datasets with little to no hyperparameter tuning and generates calibrated probabilistic forecasts with high accuracy.

Chen et al. (2020) used a probabilistic forecasting framework based on a CNN.

Zhu et al. (2017) proposed a novel end-to-end Bayesian deep model that provides time series predictions along with uncertainty estimations. The model uses an LSTM encoder-decoder followed by a prediction network. The authors check the coverage of the 95 % predictive interval on the test data to evaluate the probabilistic model. The proposed uncertainty estimate was used to measure special event uncertainty and to improve anomaly detection accuracy. The model was compared to some baseline models and was found to have the best accuracy.

Koochali et al. (2019) introduced ForGAN. It is a one step ahead probabilistic forecasting model with GANs. The authors found that in the presence of strong noise, the effectiveness of ForGAN is more prominent. The prediction and ground truth distributions are plotted against each other to show how effectively the model has predicted the distribution. KL Divergence is used to report the results of the uncertainty estimation.

Koochali et al. (2020) presented ProbCast. It is a novel probabilistic model that employs a conditional GAN framework and the model is trained with adversial training. The authors' main motivation is to transform an existing point forecast model to a probabilistic model. It is reported that converting a deterministic model into a probabilistic model actually increases the model's prediction accuracy. The performance of ProbCast is measured using the Continuous Ranked Probability Score (CRPS) as the metric. The results showed a great potential in probability forecasting using GANs.

3 Background

This chapter introduces relevant background information, notation, and definitions for the reader to understand the following chapters.

3.1 Artificial Neural Networks

Artificial neural networks are a type of machine learning model that is inspired by biological neural networks in the human brain. The network contains multiple nodes connected to each other. Each node performs some type of computation and passes on information to the next node. Depending on the architecture and the type of node used, there can be different types of neural networks such as feedforward neural networks, recurrent neural networks or transformers.

A feedforward neural network is the simplest architecture of a neural network. The input data is transformed by a series of computations performed at each node to generate an output. Information flows in a single direction. The most basic form of a feedforward neural network is the perceptron as seen in Figure 1.

For an input vector $x_1, x_2, ...x_n$, it undergoes the following transformation at a node:

$$z = \Sigma x_i w_i + b$$

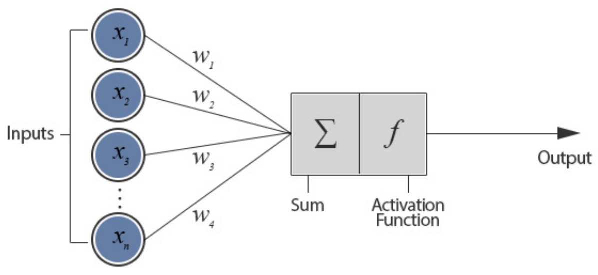


Figure 1: Perceptron Retrieved from https://sites.cc.gatech.edu/ san37/post/dlhc-fnn/

where, w_i is the weight matrix of each data point and b is the bias of the node.
This can be simplified as:

$$z = W^T x + b$$

where W represents the weight matrix and x is the input vector. This is then followed by an activation function $f(z)$ to produce the output. The activation function can be a linear function, a ReLU function, etc.
This would be the computation at a single layer of the neural network. If there are multiple layers, then the output o_n for a layer n would be calculated as:

$$o_n = f_n(h_{n-1} W_n^T + b_n)$$

where h_{n-1} is the output of the previous layer.
The weights and the biases are the trainable parameters in the neural network. So these are the values that will be learned during the training of the neural network. Hyperparameters are the parameters that can be decided by the user and control the training process. In this case, the hyperparameters would be the number of layers and the number of nodes in each layer.

A multilayer perceptron (MLP) as suggested by the name has perceptrons arranged in multiple layers as seen in Figure 2. It has multiple nodes in a single layer and multiple layers, where each node in a layer is connected to every single node in the layer ahead of it. It consists of at least 3 layers: input layer, hidden layer and output

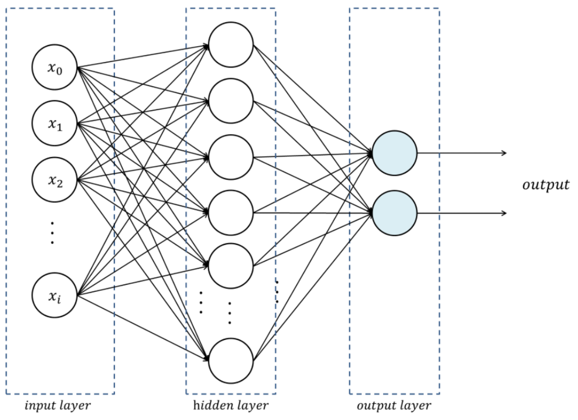


Figure 2: Multilayer Perceptron Retrieved from https://sites.cc.gatech.edu/ san37/post/dlhc-fnn/

layer. An MLP can solve non-linear and complex problems.

3.1.1 Training a Neural Network

The data used for training will have features and ground truth labels. Labels are what we want the neural network to be able to predict. When the neural network has access to the labels in the input data during training, it is referred to as supervised learning. During training, the neural network tries to learn the best weights and biases that can accurately predict the ground truth labels.

During the first step of learning, the weights and biases are initialized. There are different ways to do this, the most common being a random initialization. Only the features of the data and not the ground truth labels are passed through the neural network. The model calculates an output $\hat{y}$ using the initial values of weights and biases and the input feature values. This $\hat{y}$ is compared to the ground truth label y and the error is calculated using a loss function.

The **loss function** is then used to give the model feedback on the error in its predictions. Depending on the task at hand, there are different loss functions. For a regression task, a common loss function is the mean squared error (MSE) loss function.

$$L(y, \hat{y}) = \frac{1}{N}\Sigma_i(y - \hat{y_i})^2$$

Backpropagation: Now that there is an understanding of the error made by the model, the model needs to update its weights and biases. This is done by backpropagation. The weights and biases are updated by adding the negative gradient of the loss function with respect to the weights and biases to them, respectively.

$$w = w - \alpha\frac{\partial L(y, \hat{y})}{\partial w}$$

α is the learning rate. It controls how much the model parameters are updated with respect to the loss function gradient.

The model then predicts a new $\hat{y}$ based on the updated model parameters. A new loss function is calculated and the loss gradient is propagated through the network to update the model parameters. This process continues until the error is as minimum as possible.

Adam (Kingma and Ba, 2014) is an optimization method to dynamically change the learning rate during the training. Adam optimizer accumulates an exponentially decreasing average of the squared gradient of each parameter. The learning rate is multiplied with this squared gradient. So, each parameter would have an individual learning rate. This helps the model learn faster for parameters that need large value changes and slower for parameters that require small value changes.

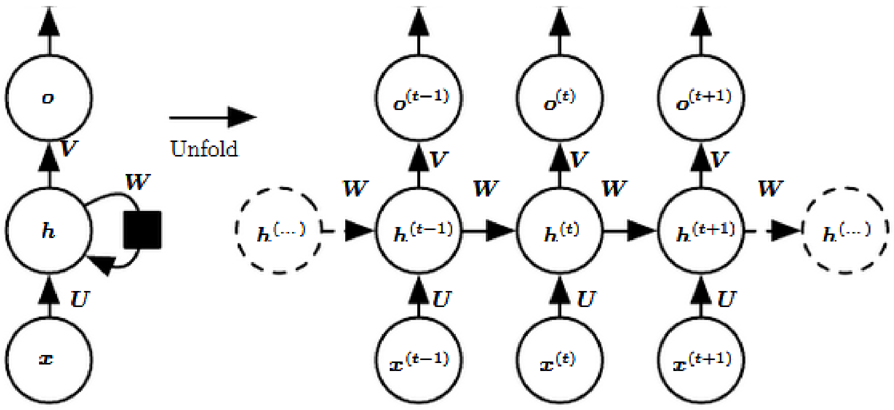


Figure 3: Recurrent Neural Network: Here the RNN architecture can be seen along with its time-unfolded structure to understand how the model processes the input sequential data. The computational graph gives an idea on how to calculate the output and training loss. The input sequence $\mathbf{x}$ is mapped to the output $\mathbf{o}$. The loss L is calculated based on the difference between $\mathbf{o}$ and the true target value $\mathbf{y}$. $\mathbf{U}$ is the weight matrix for the connections between the input and the hidden states. The hidden state to hidden state connections are weighted by $\mathbf{W}$. The connections between the hidden state and output are parametrized by $\mathbf{V}$. (Image retrieved from https://www.deeplearningbook.org/contents/rnn.html)

3.2 Recurrent Neural Networks

Recurrent neural networks are a type of artificial neural networks that is adapted to work well with sequential data, i.e. data of the form $x = (x_1, x_2, ...x_t)$. Ordinary feed forward neural networks are only meant for data points, which are independent of each other. However, if we have data in a sequence such that one data point depends upon the previous data point, we need to modify the neural network to incorporate the dependencies between these data points. RNNs have the concept of 'memory' that helps them store the states or information of previous inputs to generate the next output of the sequence. Figure 3 shows the architecture of the RNN.

As observed, past features are maintained in $\mathbf{h}^{(t)}$ which is known as cell state. At each step, the computation is done considering the input at the time step $\mathbf{x}^{(t)}$ and the previous cell state $\mathbf{h}^{(t-1)}$. The cell state is updated at each time step. Data is processed sequentially through the time steps.

Forward Pass: Assuming a discrete output, a hyperbolic tangent activation function and output softmax activation function, the forward pass would be as follows.

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(\mathbf{t-1})} + \mathbf{U}\mathbf{x}$$

$$\mathbf{h}^{(t)} = tanh(\mathbf{a}^{(t)})$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}$$

$$\hat{\mathbf{y}}^{(t)} = softmax(\mathbf{o}^{(t)})$$

where c and b are bias terms.

Backward Pass: The back-propagation algorithm used here is known as back-propagation through time (BPTT). Once the loss function is calculated, the weights $\mathbf{U}$, $\mathbf{V}$, $\mathbf{W}$ and bias b, c are updated using the learning rate and the respective gradient. The gradient at each output depends on the calculations of the current time steps and the previous time steps.

3.3 LSTM

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. A typical RNN suffers from the problem of vanishing and exploding gradients. Vanishing gradients occur when during backpropagation, the gradients tend to zero and the model loses information about earlier data. Exploding gradients is when the gradients tend to infinity. LSTM cells avoid these problems by having 'gates' that can choose which information it wants to forget and which information it wants to retain by allowing gradients to flow unchanged.

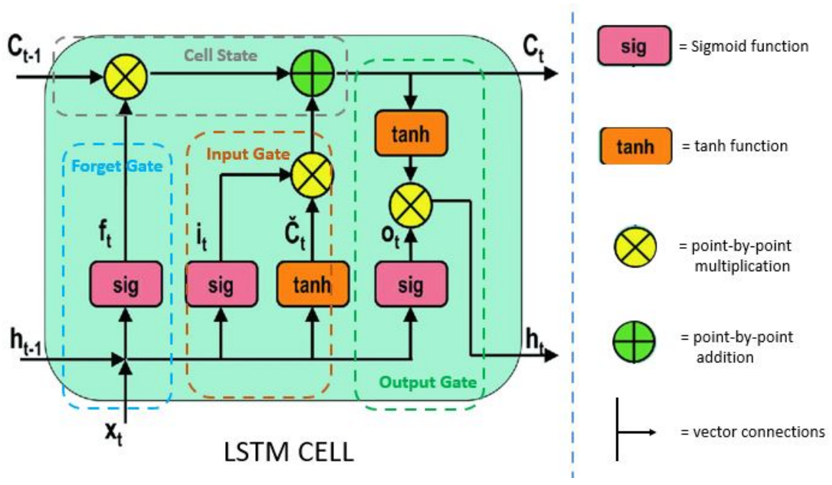


Figure 4: LSTM Cell: A single LSTM with the flow of information through the cell. $\mathbf{x}_t$ is the input data at time step t. $\mathbf{h}_{t-1}$ is the hidden state from the LSTM cell at the previous time step. $\mathbf{C}_{t-1}$ is the previous cell state. $\mathbf{C}_t$ is the current cell state and $\mathbf{h}_t$ is the new hidden state. (Image retrieved from https://www.pluralsight.com/guides/introduction-to-lstm-units-in-rnn)

Each LSTM unit has the following gates:

- Forget Gate: The forget gate decides which information needs attention and which can be ignored.

- Input Gate: The input gate performs operations on the input data before it used to update cell status.

- Output Gate: The output gate determines the value of the next hidden state. This state contains information on previous inputs.

- Cell State: This operation updates the current cell state based on the forget gate. The previous cell state is either dropped or retained depending on the forget gate and then this is added to the input.

Each LSTM cell is connected and information passes through the LSTM cells. Information can flow in one-direction or in both directions.

Forward Pass: Information flows in one direction as shown below.

- Forget Gate:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$

  where $\mathbf{W}_f$ is the weight matrix between the forget gate and the input gate and $\mathbf{b}_f$ is the connection bias at the forget gate.

- Input Gate:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i.[\mathbf{h}_t, \mathbf{x}_t] + b_i)$$

$$\tilde{\mathbf{C}}_t = tanh(\mathbf{W}_C.[\mathbf{h}_{t-1}, \mathbf{x}_t] + b_C$$

  where $\mathbf{W}_i$ is the weight matrix between the input gate and the output gate, $\mathbf{W}_C$ is the weight matrix of the tanh operator between the cell state information and output, $\mathbf{b}_i$ is the bias vector at t with respect to $\mathbf{W}_i$ and $\mathbf{b}_C$ is the bias vector at t with respect to $\mathbf{W}_C$.

- Cell State:

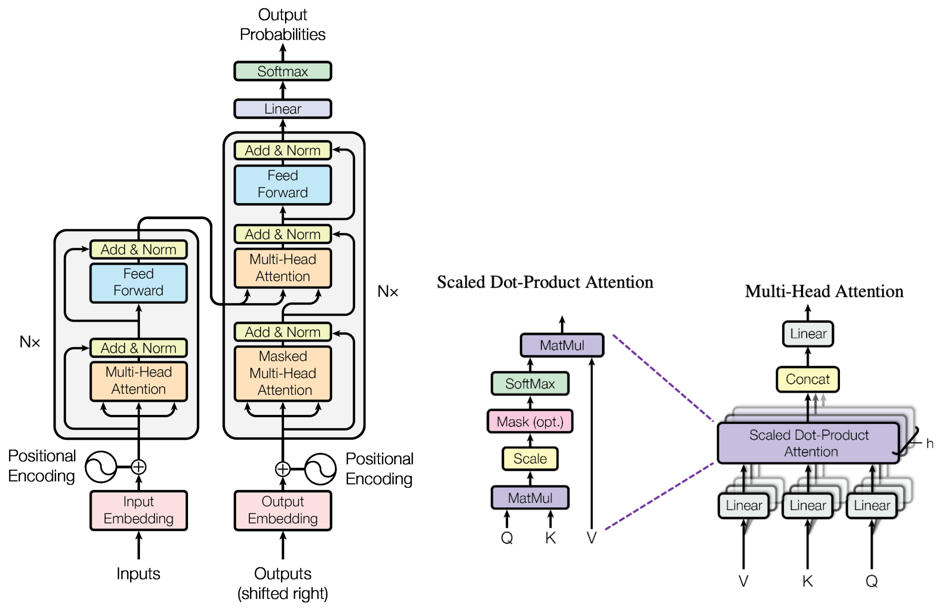
$$\mathbf{C}_t = \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{C}}_t$$

- Output Gate:

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o$$

$$\mathbf{h}_t = \mathbf{o}_t * tanh(\mathbf{C}_t)$$

  where $\mathbf{W}_o$ is the weight matrix of output gate and $\mathbf{b}_o$ is the bias vector



(a) Transformer Model **(b)** Attention Mechanism

Figure 5: Attention Mechanism: (Image retrieved from Vaswani et.al (2017))

3.4 Transformers

Like RNNs, Transformers are a machine learning model designed to deal with sequential data. It uses a mechanism of self-attention to determine the importance of each part of the input data. Unlike RNNs, Transformers process the entire input sequence at once. Transformer was first introduced by Vaswani et al. (2017). Figure 5(a) shows the transformer architecture introduced by Vaswani et al. (2017)

- Attention: The attention mechanism, as seen in Figure 5(b) focuses on the most important parts of the input sequence by assigning them higher weights/attention scores.

  Every input vector into the attention block is used in three different ways in the attention mechanism: the Query, the Key and the Value tensors, denoted as Q, K and V respectively in the diagram above.

Scaled Dot-Product Attention: This can be seen in Figure 5(b). The query and key are dot-producted and scaled. This value is then softmaxed to give attention probabilities. The value tensors are interpolated by these attention probabilities to give the output.

$$attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}}V)$$

Multi-Head Attention: As seen in the figure, there a multiple attention heads in the Transformer architecture. This mechanism projects the Query, Key and Value h number of times, each time using a different learned projection. Each attention head runs in parallel to produce an output and the outputs are concatenated and projected again to create a final result. The multi-head attention is as follows:

$$multihead(Q, K, V) = concat(head_1, head_2, ..., head_h)W^0$$

Each head implements its own attention mechanism as follows:

$$head_i = attention(QW_i^Q, KW_i^K, VW_i^V)$$

where h is the number of attention heads and W^O, W_i^K, W_i^K, W_i^V are weight matrices.

The concatenated output is multiplied with the weight matrix W^O to output the final result of the multi-head attention block.

Self Attention: When these Query, Key, Value tensors come from the same input sequence, then it is known as self-attention. Essentially the input sequence pays attention to itself. There is a self-attention mechanism in each the encoder and decoder. This mechanism relates every data point in the input sequence to every other data point. In the decoder self-attention a look-ahead mask is used to prevent the model from looking ahead in the target sequence when it needs to make a prediction.

Cross Attention: The Query, Key and Value Tensors are all not derived from the same input sequence. This attention mechanism is seen between the encoder and decoder, where the Query is the decoder input and the Value and Key tensors are the encoder output. The target sequence (decoder input) pays attention to the input sequence (encoder output).

- Encoder-Decoder Architecture: The encoder maps an input sequence of symbol representation $(x_1, ..., x_n)$ to a sequence of continuous representations $z = (z_1, ..., z_n)$. The decoder then generates the output sequence $y_1, ..., y_m$ one at a time, from z and the output of the self-attention in the decoder.

3.5 Probability Distributions

A probability distribution is the function that expresses all the possible values and likelihoods that a random variable can take between a certain range. Two things that can be learnt from a probability distribution are the expected value and the variance. For a continuous random variable, the probability distribution can be used to find the probability of a value in a particular specified interval.

<u>Probability Distribution Function:</u> Probability of a value for a continuous random variable.
<u>Cumulative Distribution Function:</u> Probability that a variable takes a value less than or equal to a value x.

Some of the probability distributions of a continuous random variable are described below.

3.5.1 Normal or Gaussian Distribution

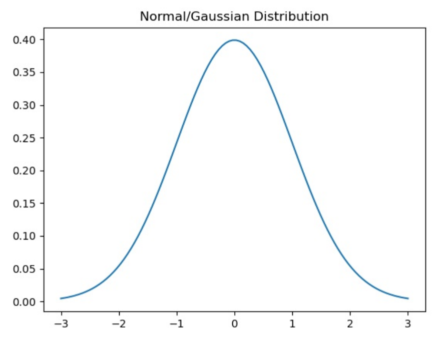


Figure 6: Normal Distribution

This is the most common distribution. The distribution curve is bell-shaped and symmetrical along the line at mean. Figure 6 shows a normal distribution. The probability distribution function is given by

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

where μ and σ represent the mean (center of the distribution) and the standard deviation of the population, respectively.

3.5.2 Log-Normal Distribution

It is a continuous probability distribution of a random variable in which the logarithm is normally distributed. The distribution curve is not necessarily symmetric and can be right-skewed as well. The probability distribution function is

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{ln(x)-\mu}{\sigma})^2}$$

where μ is the location parameter and σ is the standard deviation of the distribution. Figure 7 shows the log normal distribution.

Properties of log-normal distribution:

- The log-normal is a right-skewed distribution.
- The PDF starts from 0, so its values are always greater than 0.
- As observed in figure 7, for a given μ the degree of skewness increases as σ increases.

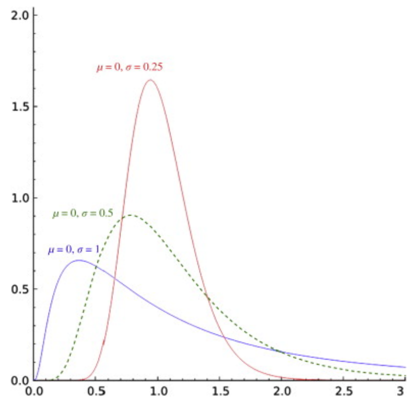


Figure 7: Log Normal Distribution The figure shows how the log normal distribution changes with respect to the μ and σ parameters. Image retrieved from https://www.sciencedirect.com/topics/engineering/lognormal-distribution.

3.5.3 Gamma Distribution

It is a continuous probability distribution which is used to model continuous variables that are always positive and have a skewed distribution. This distribution has been frequently used to model the time between events. The parameters of the distribution are:

- Threshold parameter: It defines the smallest value in the distribution. All values in the distribution must be grater than this parameter. It is usually set to 0, therefore making it possible to only have positive values.

- Shape parameter (k): Specifies the number of events being modeled.

- Scale parameter (θ): It represents the mean time between events. It determines how much the distribution graph is spread.

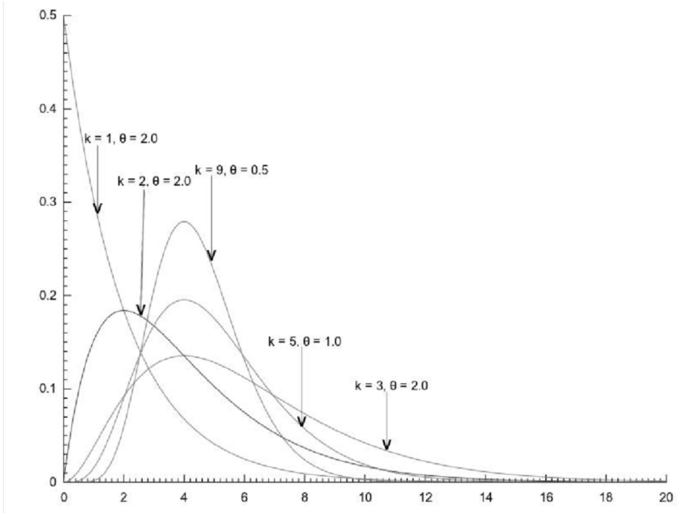


Figure 8: Gamma Distribution The effects of the shape and scale parameters on the distribution.
Image from Ghose, Partha. (2017)

The probability density function is

$$f(x; \alpha, \beta) = \frac{x^{\alpha-1} e^{-\beta x} \beta^{\alpha}}{\Gamma(\alpha)}$$

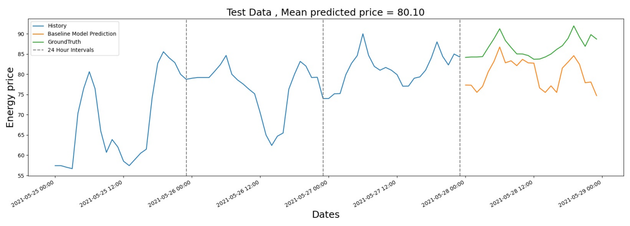
for $x > 0$ and $\alpha, \beta > 0$ where, $\Gamma(\alpha)$ is the gamma function,

$\alpha =$k and

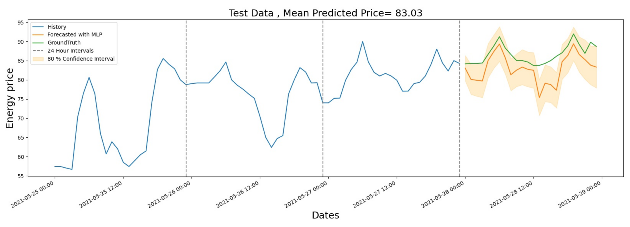
$\beta = \frac{1}{\theta}$

3.6 Probabilistic Forecasting

When a machine learning model is used to forecast point values, it is deterministic forecasting. A probabilistic forecast predicts a probabilistic distribution over future quantities of interest. This is useful in learning the confidence of the model in predicting future values. The difference can be visualized in Figure 9.



(a) Deterministic Forecasting



(b) Probabilistic Forecasting

Figure 9: Deterministic Forecasting vs Probabilistic Forecasting: Deterministic forecasting predicts future values, whereas probabilistic forecasting predicts future distributions of values from which expected values and deviations can be derived

3.7 Continuous Ranked Probability Score

The Continuous Ranked Probability Score (CRPS) measures how a proposed distribution approximates the data, without knowledge about the true distributions of the data. It measures the squared distance between the predicted distribution and the

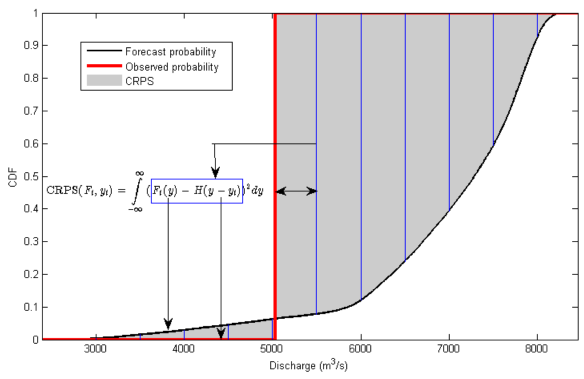


Figure 10: CRPS Score Calculation Illustrated The black curve denotes the CDF of the predicted probability distribution and the red line denotes the CDF of the true value.
(Image retrieved from https://www.mathworks.com/matlabcentral/fileexchange/47807-continuous-rank-probability-score)

target. The CRPS score is defined as:

$$CRPS(P, x_a) = \int_{-\infty}^{\infty} ||P(x) - H(x - x_a)||_2 dx$$

where x_a is the true value of x

P(x) is the proposed distribution for x

H(x) is the Heaviside step function

$$H(x) = \begin{cases} 1, & x = 0 \\ 0, & x \leq 0 \end{cases}$$

As observed in Figure 10, the CRPS score is essentially calculated by finding the difference between the predicted cumulative distribution function (CDF) and the CDF of the true value. The lower the CRPS score, the better the distribution is at fitting the data.

3.8 Log Likelihood

Log likelihood is the log of the probability density function evaluated at a value. It can be used to evaluate how well a probability function fits a random variable.
Negative Log Likelihood: The higher the log likelihood value of the observed data, the better the distribution fits the data. However, while training a machine learning model, the loss function needs to minimized. So the log likelihood cannot be used as a loss function, since ideally the log likelihood needs to be maximized. Hence, the negative log likelihood is used a loss function. Minimizing the negative log likelihood function naturally maximizes the log likelihood and hence the likelihood of the observed data.

3.9 Previously Trained Models

In previous experiments that we have done [1], we trained MLPs, Residual MLPs and LSTMs with the dataset described in Chapter 4. At the time, data was available only until mid 2021. Residual MLP had the lowest Mean Absolute Error (MAE) of 5.66, followed by MLP (6.06) and then LSTM (11.33). However, not enough experiments were conducted with LSTMs to definitely prove that it cannot have a lower MAE than Residual MLP. It is important to keep in mind that since these experiments were conducted more data has become available and the dataset is updated until the end of 2021. These models will be trained with the current data and compared with the Transformers trained with the same data.

4 Approach

This chapter describes the approach we followed to tackle the problem of energy prices forecasting. We explain the datasets used, data-preprocessing, the proposed models for forecasting and the model training and evaluation.

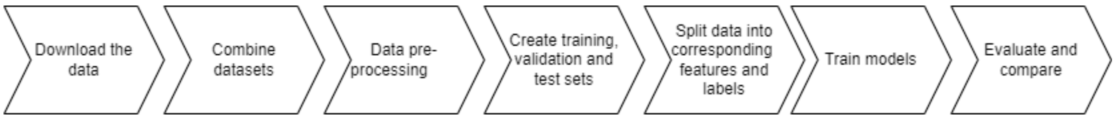

Download the data
Combine datasets
Data pre-processing
Create training, validation and test sets
Split data into corresponding features and labels
Train models
Evaluate and compare

Figure 11: Workflow

4.1 Data

There are 2 publicly available datasets which contain relevant information. In both datasets, data is available at an hourly frequency. The data for Spain and Switzerland is downloaded and preprocessed before being used separately for model training.

Datasets:
1. Copernicus: Copernicus is the European Union's Earth Observation programme. It offers information services that draw from satellite observations and in situ data. The European Commission manages the programme. The information is free and openly accessible to users. The dataset 'Climate and energy indicators for Europe from 1979 to present derived from reanalysis' is used. Data is downloaded at a country level,

i.e there is a separate column for each country in the EU with the respective values. The following climate variables are downloaded: wind speed, surface downwelling shortwave radiation, air temperature and total precipitation. Additionally, solar photovoltaic power generation and wind power generation are also downloaded. Each feature is downloaded as a separate .csv file. Data is available beginning from 1979.
2. Entsoe: This is the European Network of Transmission System Operators. Load, day ahead prices and generation can be directly downloaded for a specific country using API requests. Power generation values are provided for all sources, however since we are mainly concerned with solar and wind, we aggregate the remaining values into a single column as 'Other energy sources'. Data was available from 2014 to 2021. The following features are used from Entsoe:

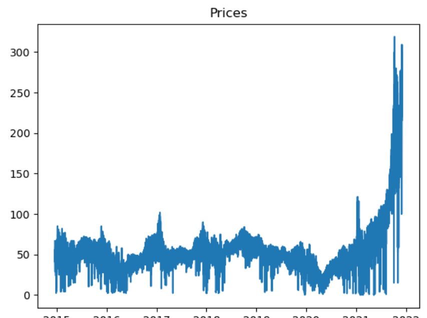
- Load: Data about power consumption in Megawatt (MW).
- Generation: Energy production for solar and wind energy sources. Other energy sources are aggregated together (MW).
- Prices: For every market time unit the day-ahead prices in each bidding zone (Euro/MWh).

The deep learning models are trained separately on the Spain and Switzerland datasets respectively.

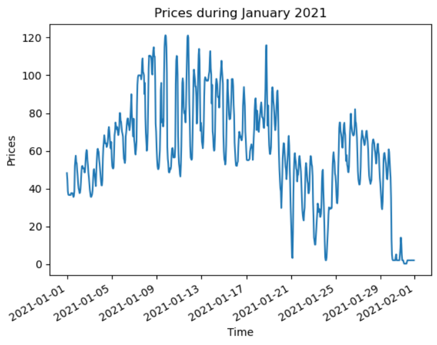
4.1.1 Spain Dataset

Figure 12 shows the plots for the electricity prices (in €/MWh) in the Spain dataset. Since this is the feature to be predicted, it would be interesting to observe how the prices change over time.

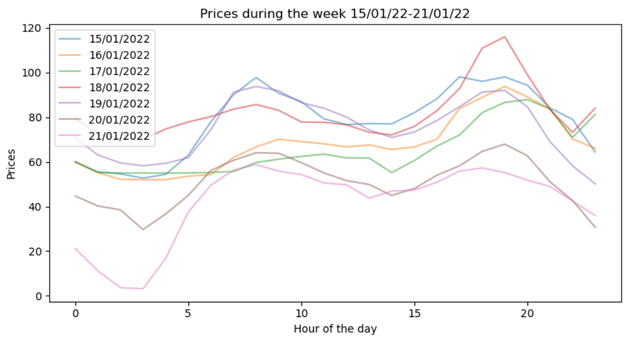
Figure 12(a) shows the prices from 2014 until the end of 2021. There is a sharp increase in the prices from 2021 onwards. This can be attributed to to the fact that



(a) Energy Prices in the Spain dataset



(b) Energy Prices for January 2021



(c) Daily prices for a week in January 2021

Figure 12: Data Plots of Spain Dataset

in 2020, the demand for electricity, gas and coal had greatly dropped due to the COVID-19 pandemic. However, in 2021, the global economy recovered and there was a large increase in fuel and carbon dioxide prices and this resulted in a very sharp increase of electricity prices in 2021. This is reflected in figure 12(a) with a steep increase in the prices from mid 2021.

Figure 12(b) shows the daily changes in price over a period of one month. January 2021 was chosen as an example. The prices seem to decrease towards the end of the month.

Figure 12(c) shows the hourly price change in a day, for a period of one week. The price is naturally lowest at late night, increasing in the morning and evenings.

4.2 Data Preprocessing

Data Preprocessing includes cleaning and correcting the data. After downloading the data, there are 2 datasets, one each from Entsoe and Copernicus.

Firstly, with the Entsoe data, by default, data is downloaded with the datetimes as an index column. The dates column is shifted from the index to its own column. This is done as it helps in a later step of combining the data with the Copernicus data. Additionally, the dates and times in the Entsoe dataset are in the timezone of Madrid and Zurich respectively. This does not match with the Copernicus data which is in UTC timezone. This date column is converted to datetime datatype and converted to UTC timezone to match the Copernicus data.
Next with the Copernicus data: data specifically only for the chosen country is extracted from each file and combined. Duplicates are dropped. The 'Date' column is converted to datetime datatype. Finally the Entsoe and copernicus datasets are merged on the 'Date' column. Any NaN values in the dataset are identified and replaced with the value from 24 hours before. The dataset is checked to make sure there are no missing time steps.

Relevant features are chosen. Wind offshore is not considered in the Spain dataset since all its values are 0. This is because Spain is yet to establish an offshore wind industry. Switzerland naturally has no offshore wind industry, since it is not a coastal country. The power generation variables of Entsoe are used. 'Other Energy Sources' is also discarded as the analysis is more focused on solar and wind power. Additionally the data is scaled. The whole data is split into three sets: training, validation and test set. Training set is the first 85% of the whole dataset. The remaining 15% is divided equally into the validation and test dataset. The training data lies between 20/12/2014 11:00 to 15/11/2020 7:00. The validation data ranges from 16/11/2020 0:00 to 23/05/2021 23:00. The test data is between 25/05/2021 0:00 and 29/11/2021

23:00. The model was evaluated on the validation set as training occurred. The test dataset was used for final evaluation after training was complete.

Each of the training, validation and test sets is further split into sets of features and labels. The features will have ‘n’ number of rows, each row containing electric and weather data for a certain hour. The number of rows in the features is determined by the number of hours we go back in the history. These features are used to predict 24 future hourly energy price values, which would be the labels. For example, if we take a history of 72 hours, each data sample would have 72 rows of features (each row containing electric and weather data for the hour) corresponding to 24 labels which are the future price values.

4.3 Models

Models 4.3.1 to 4.3.3 were trained and evaluated during the study project [1].

4.3.1 Linear Model

The model used a single dense layer as the output layer, with number of output neurons = 24 (number of price values to predict). This is used as a baseline to compare with the results of more complex models.

4.3.2 Residual MLP

The model can be seen in Figure 13. A custom mean layer is introduced to an MLP to help make the process of training the model easier. The objective of the mean layer is that it calculates hourly averages of energy prices from the past data. It has no trainable parameters. So its output is 24 values, each signifying the mean of

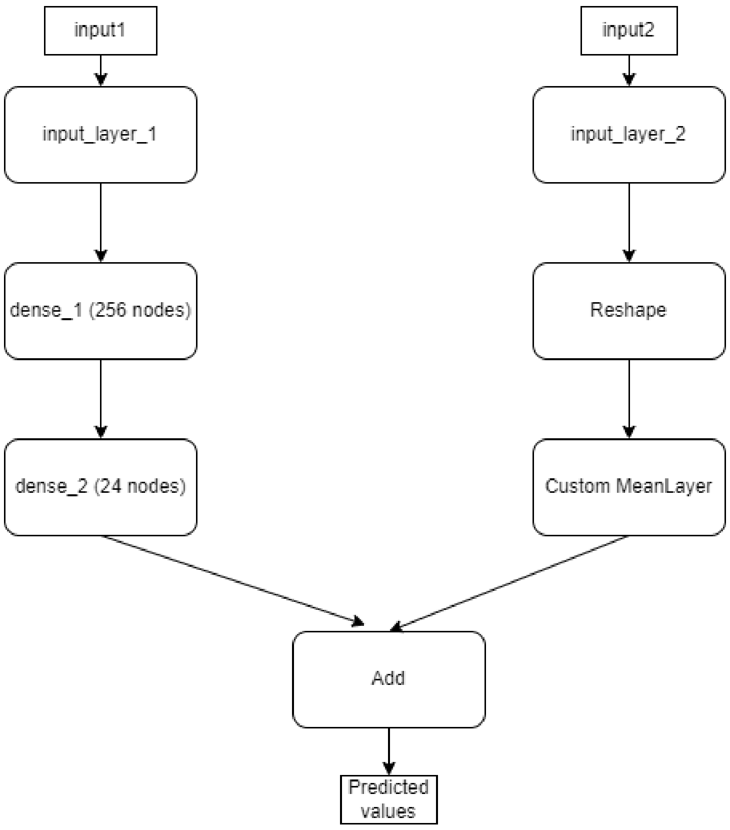


Figure 13: Residual MLP: A multilayer perceptron with a residual connection

energy prices over the past few days at that specific hour. For example, given the data of the past 72 hours. This is energy price data over three days. The mean of the values over each hour is calculated. The final result is 24 values, a mean energy price value for each hour of the day.

The model is then forced to learn the difference between these past energy prices and the energy prices 24 hours in the future. The differences predicted are added to the past hourly averages and this is the final output of the model. The idea is that training might be easier if the model only has to learn the slight differences between energy prices, instead of predicting the energy price itself. Input1 is all the features in the past 72 hours and input2 is the energy prices in the past 72 hours.

Hyperparameter	Value
number of steps in	72
number of steps out	24
Dense1 neurons	256
Dense2 neurons	24
optimizer	Adam
learning rate	10^{-4}
loss	Mean squared error
epochs	300

Table 1: Residual MLP Hyperparameters

The inputs need to be flattened to a simple vector. The input before flattening would be of the form (number of samples x number of hours back x number of features). So for example, if we choose to look back 72 hours in the past for training, the data would be of the form (72x8) since each row of data has 8 features. This is flattened to a simple vector with size 576, obtained by multiplying both values. The input layer would now be receiving the data in the form (number of samples x 576). The hyperparameters used can be seen in Table 1.

Regularization Methods:

- Early stopping: This is used to avoid overfitting. Training automatically stops when the validation loss does not show any improvement after 15 epochs. Naturally, early stopping helps in reducing training time if there is no improvement after a number of epochs.

- Reduce learning rate on plateau: Reduces learning rate when validation loss has plateaued over 10 epochs. This was particularly useful as it was observed in earlier experiments while training a model without this method, validation loss would plateau and not decrease at all. There is an observation of a decrease in training and validation loss in most cases, when the learning rate is reduced during training.

4.3.3 LSTM

The model uses two LSTM layers, followed by a dense layer. It was mentioned previously that the input to an MLP needs to be flattened to a simple vector. However, this is not the case with LSTMs. The input can be passed in the form (number of hours back x number of features). LSTMs can easily process time series data. The input data would be of the form (number of samples x number of hours back x number of features). The hyperparameters used can be seen in Table 2. Regularization: Early

Hyperparameter	Value
number of steps in	72
number of steps out	24
lstm1 nodes	64
lstm2 nodes	64
dense1 nodes	24
optimizer	Adam
learning rate	10^{-4}
loss	Mean squared error
batch size	64
epochs	150

Table 2: LSTM Hyperparameters.

Stopping, Reduce learning rate on plateau

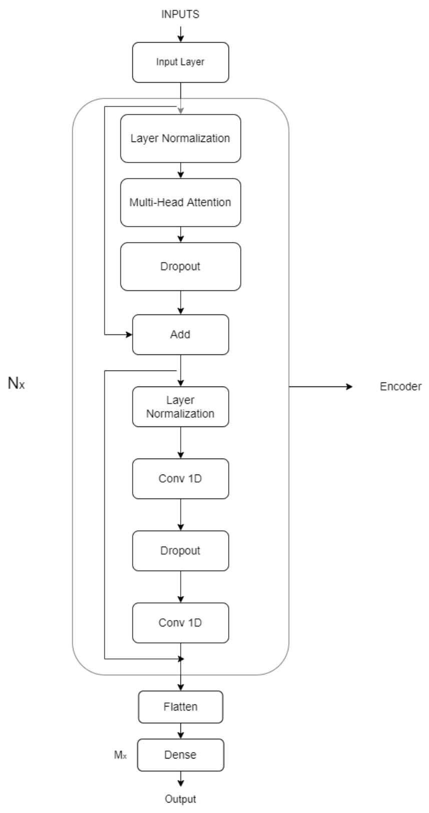


Figure 14: Transformer Architecture

4.3.4 Transformers

The transformer model used in our experiments is an encoder only model. The architecture is shown in Figure 14. The layers are as follows:

- Input Layer
- Encoder
- Flatten Layer
- Hidden Dense Layers
- Output Layer

The encoder unit contains:

- Layer Normalization: Normalizes the activations of the previous layer for each given example in a batch independently,
- Multi Head Attention Layer: This is the multi head attention layer as described in Vaswani et al. (2017). Since the query, key and values parameters are the same, it is self-attention.
- Dropout Layer: Randomly nullifies the contribution of some neurons during training. The other neurons are not affected. This is done to avoid overfitting. The output of this layer is added to the input of the encoder unit (residual connection).
- Layer Normalization
- Convolution 1D Layer: Has a convolution kernel that is convolved with the input over a single spatial dimension. This type of convolution is layer is commonly used with time series data as it can learn the internal representations is time series data.
- Dropout Layer
- Convolution 1D Layer: the output from this layer is added to the output from the first dropout layer via a residual connection.

The hyperparameters used are mentioned in Table 3.

Regularization: Early Stopping, Learning Rate Scheduler with Exponential Decay

Hyperparameter	Value
head size	32
number of heads	4
Conv1D kernel size	1
Number of Conv1D filters	4
number of encoder blocks	4
mlp dropout	0
drop in attention block	0.1
mlp nodes	[256]
learning rate	10^{-4}
loss	Mean squared error
epochs	200
training batch size	64
testing batch size	16

Table 3: Transformer model Hyperparameters

4.3.5 Probabilistic Models

All the models described until now are deterministic models. Each of these models can be used for probabilistic forecasting by adding a DistributionLambda layer to the end, doubling the number of nodes in the final Dense layer and changing the loss function to negative log likelihood (as explained in Section 3.8). The DistributionLambda layer is a Keras layer that enables plumbing distributions through a deep learning model and outputs a distribution. The number of nodes in the final Dense layer are doubled because half of them would represent mean (or the appropriate parameter depending on the probability distribution) and the other half represents the variance. So in our case, the output Dense layer of the deterministic models has 24 nodes since we need to predict 24 future price values. For probabilistic model, the number of nodes of this final Dense layer becomes 48. This architecture is shown in Figure 15

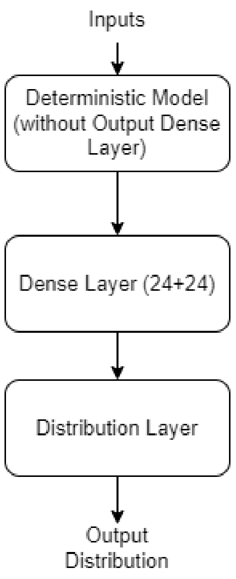


Figure 15: Probabilistic Model Architecture

4.4 Training

Before we begin to train the model, the data needs to be further processed. The data is scaled using a Min-Max Scaler. This scaler transforms the features by scaling each feature to a given range. The transformation, for data x, is as follows:
$Xstd = \frac{x - min(x)}{max(x) - min(x)}$
$Xscaled = Xstd * (max - min) + min$
where max and min are the feature range used. In our case, we use a range of (0,1). The data for the deterministic Transformer model is scaled using Robust Scaler. Robust Scaler scales the data between the inter-quantile range. So a data point x_i is transformed as $\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$. Robust Scaler was used for this model, since this scalar is supposed to be robust to outliers and could help with the accuracy.
If there are features with large differences in their values, it will result in different step sizes for the weights of each feature and will make gradient descent convergence much harder. Scaling helps avoid this issue.

Furthermore, the data is augmented by multiplying the training data by 3.5. This is used as additional training data to the original training dataset. It is observed that in the recent year, the energy prices have drastically increased to values that do not exist in the training data at all. In order for the model to be able to make accurate predictions, we augment the training data to introduce higher price values and add this to the original training dataset.

Each of the training, validation and test set are then split into inputs and the corresponding label. Each input is of the shape (number of hours back, number of features) and the label (which contains only prices) is an array of 24 values. This corresponds to taking the features of historical data to predict energy prices 24 hours into the future. This input data will be given to the model and it will learn to predict the corresponding label.

For the transformer, the information about the day, month and year were encoded into the training data. This was done to add additional information to the dataset as these are features that could also affect energy prices. The day and month of each datapoint are extracted and cyclically encoded. For a day or month $x \in X$, it is transformed as follows:

$$x = \frac{2\pi x}{max(X)}$$

The sin and cos of this value are then stored as the day or month information for the datapoint. This preserves the cyclical nature of days and months. Since year is not cyclical, it is directly encoded as an integer.

Model	Time
Linear	4 minutes and 20 seconds
Residual MLP	27 minutes and 5 seconds
LSTM	150 minutes and 28 seconds
Transformer	12 minutes and 30 seconds

Table 4: Training times of the models

Training for the LSTM and Transformer models were done using an NVIDIA GeForce RTX 2080Ti GPU. The training times are mentioned in Table 4.

Model	Trainable Parameters
Linear	192
Residual MLP	153,880
LSTM	33,224
Transformer	674,304

Table 5: Trainable parameters of the models

Table 5 shows the number of trainable parameters in the models. LSTM is advantageous in that it has the least number of trainable parameters. Transformer has the most number of parameters, having the most complexity.

5 Experiments

5.1 Evaluation Metrics

In order to evaluate the trained models, we need to test the models against the test dataset and use some metric to determine how good the model is at forecasting.

5.1.1 Deterministic Models

For deterministic forecasting, the following metrics are used:

- Mean Absolute Error (MAE): MAE measures the average magnitude of errors in the predictions. It is the average of the absolute differences between predictions and observations (test dataset).

  $$MAE = \frac{1}{n}\Sigma_{j=1}^{n}|y_j - \hat{y}_j|$$

  where, n is the total number of observations
  y is the predicted value and
  $\hat{y}$ is the true value.

- Root Mean Squared Error (RMSE): RMSE is a quadratic scoring method. It is calculated by the square root of the average of squared differences between the predicted value and actual value.

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

RMSE penalizes large errors made by the model.

5.1.2 Probabilistic Models

The mean/median of the predicted distribution can be compared against the actual observations using MAE and RMSE. However, this does not give an indication of whether the predicted distribution performed well at learning the target distribution. Different metrics are needed to evaluate probabilistic models.

- Quantile Difference: For a predicted distribution, if we consider 2 quantiles $0 < q1 < q2 < 1$, we expect that $(100 * (q2 - q1))\%$ of observations are covered. For example, for $q1 = 0.25$ and $q2 = 0.75$, we expect that the prediction distributions cover 50% of the observations. The closer to 50%, the better the model is.

- Continuous Ranked Probability Score (CRPS): As described in Section 3.7, the CRPS score measures the squared distance between the predicted distribution and the target. The model that has the lower CRPS value is the model that has a better fit of the data. When the CRPS metric is used with a deterministic forecast, then it is equivalent to calculating the MAE (Koochali et al., 2020) . Therefore, this metric can be used to directly compare probabilistic models to deterministic models.

- Log Likelihood: As described in Section 3.8, log likelihood is the logarithm of the probability density function of the observed data. The higher the log likelihood value, the better the model is at fitting the data.

5.2 Model Results

The accuracy of the different models was measured using the above mentioned metrics. These values are calculated using the test dataset.

5.2.1 Deterministic Models

Spain Dataset

The results of the baseline linear model are mentioned in Table 6

Model	MAE (€/MWh)	RMSE (€/MWh)
Linear	47.98	59.63

Table 6: Baseline Model Results

Model	MAE (€/MWh)	RMSE (€/MWh)
Residual MLP	13.42	19.17
LSTM	13.18	19.22
Transformer	9.75	15.09

Table 7: Deterministic Model Results These are the results of the models trained on the augmented dataset

Model	MAE (€/MWh)	RMSE (€/MWh)
Residual MLP	15.13	20.49
LSTM	23.82	32.66
Transformer	10.65	16.56

Table 8: Deterministic Model Results These are the results of the models trained on the original data without any augmentation

Upon comparing the results between using augmented (Table 7) and non-augmented data (Table 8), it is obvious that data augmentation helps a lot. It makes a huge difference to the predictive capability of the residual MLP and LSTM and a slight improvement to the Transformer model as well.
There are three types of Transformer models that were trained, based on the input data:

- The input data contains only the features.
- The input data contains the features and day, month and year encoding.
- The input data contains features, day, month and year encoding and next day weather forecast. The weather forecast in the input data was obtained by using the weather features of the next day. Hence, this is a perfect weather forecast, which we cannot expect in reality.

These models were trained on the augmented dataset. The results can be seen in Table 9.

Adding the date, month and year details and weather forecast data does not

Data	MAE (€/MWh)	RMSE (€/MWh)
Features	9.75	15.09
Features+time	11.6	16.75
Features+time+weather forecast	11.64	17.02

Table 9: Transformer Model Results Results of the transformer models trained on different input data

improve the MAE and RMSE. While it was speculated that these features would help improve the accuracy, it seems to be that the transformer does not require this information to make accurate predictions.

Switzerland Dataset

All the models that were trained on the Spanish dataset were also trained on the Switzerland dataset separately, using the same hyperparameters.
The results of the baseline linear model are mentioned in Table 10

Model	MAE (€/MWh)	RMSE (€/MWh)
Linear	85.9	106.92

Table 10: Baseline Model Results

Model	MAE (€/MWh)	RMSE (€/MWh)
Residual	22.64	32.18
LSTM	21.71	31.4
Transformer	17.29	26.05

Table 11: Deterministic Model Results These are the results of the models trained on the augmented dataset

Model	MAE (€/MWh)	RMSE (€/MWh)
Residual	39.61	51.8
LSTM	1013.18	6131.05
Transformer	22.91	31.75

Table 12: Deterministic Model Results- Switzerland These are the results of the models trained on the original data without any augmentation

Upon comparing the results between using augmented (Table 11) and non-augmented data (Table 12), it is seen that in this case, data augmentation also helps. The MAE of the LSTM model is extremely high without augmentation. The LSTM predicts extremely high values in the last month of the test dataset, leading to a very high MAE.
While the results are not as good as for the Spain dataset, it is worth noting that the model hyperparameters were left unchanged. Experimenting with the

hyperparameters specifically for the Swiss dataset could help increase accuracy. Additionally, Switzerland has a lesser percentage of renewable energy usage than compared to Spain. So it could be harder to forecast prices based on the features that we have chosen.
But it is observed that Transformer is still the best performing model among the 3 deep learning models. Similar to the Spain dataset, there are three types of Transformer models used and the results are shown in Table 13.

The observation is still the same that adding the additional features does not

Data	MAE (€/MWh)	RMSE (€/MWh)
Features	17.29	26.05
Features+time	17.91	27.07
Features+time+weather forecast	19.1	28.45

Table 13: Transformer Model Results Results of the transformer models trained on different input data

reduce the error.

5.2.2 Probabilistic Models

The initial experiments into probabilistic forecasting were done using only the residual MLP models on the Spanish dataset. Table 14 shows the results of these probabilistic models.

Taking into consideration the CRPS values it is interesting to note that the CRPS values are much lower than the MAE of the deterministic Residual MLP (13.42). This means that making the model a probabilistic model improves the accuracy. This can also be corroborated by comparing the MAE values of the distributions. The MAE is calculated using the mean values of the predicted price distributions against the true price values. The MAE values are also lesser than the MAE of the deterministic model.

Distribution	Quantile Difference(80%)	CRPS	Log Likelihood	MAE
Normal	50.49	8.44	0.67	10.85
Log Normal	48.81	10.92	-0.67	12.44
Gamma	63.96	12.46	0.68	12.47
Normal (Transformer model)	48.81	12.39	0.27	15.96

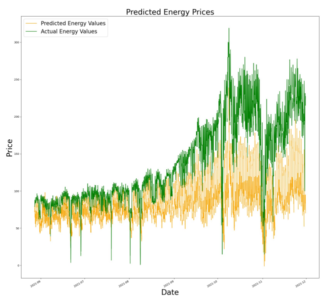
Table 14: Probabilistic Model Results

While the CRPS scores suggest that the normal distribution is the better distribution to use, the quantile difference values say otherwise. Quantile difference as a metric is more suitable for methods that estimate prediction intervals directly such as quantile and conformal regression (Romano et al., 2019). CRPS is applied to compare algorithms that model conditional distributions (Herbach, 2000). So, it would be better to take into consideration the CRPS score over the quantile difference. Hence, the normal distribution would be the best distribution to use to model the predicted prices. The comparison of the MAE values would also support this conclusion. The difference in the log likelihood values of the normal and gamma distributions seems negligible.
The transformer probabilistic model was trained with normal distribution and the results can also be seen in Table 14.

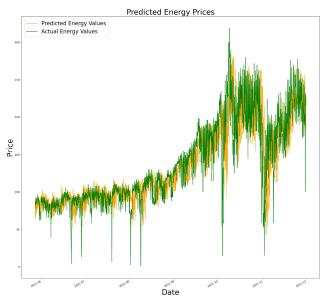
While the residual MLP probabilistic model decreased the error compared to the deterministic model, this does not seem to be the case for the Transformer model. The MAE for the deterministic Transformer is 9.75. In comparison, there is a slight increase in the error of the probabilistic transformer model.

5.3 Prediction Graphs

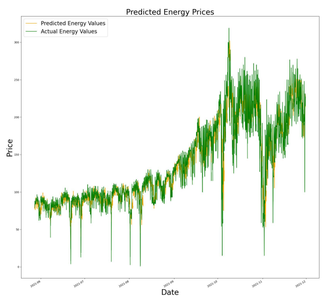
Deterministic Models: As seen from the plots in Figure 16, the simple Linear model is unable to predict the higher prices observed during the more recent months.



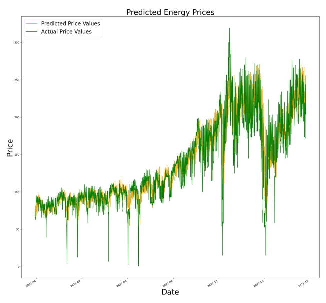
(a) Linear Model



(b) Residual MLP



(c) LSTM



(d) Transformer Model

Figure 16: Prediction over entire Test Dataset

The Residual MLP, LSTM and Transformer models are more successful in predicting the prices. The Transformer model not only has a slightly better MAE compared to LSTM, but the Transformer has a faster training time.

Transformer Deterministic Model: Figure 17 shows some day-ahead predictions of a single day of the best Transformer model. It is observed that the model is quite successful in predicting the prices and performs well in predicting how the price changes over the 24-hour period. While there are some instances where the magnitude of the predicted price is not the same as the true value, the model is successful in learning the daily price fluctuation behaviour.

Probabilistic Residual MLP Model: Figure 18 shows the probabilistic predic-

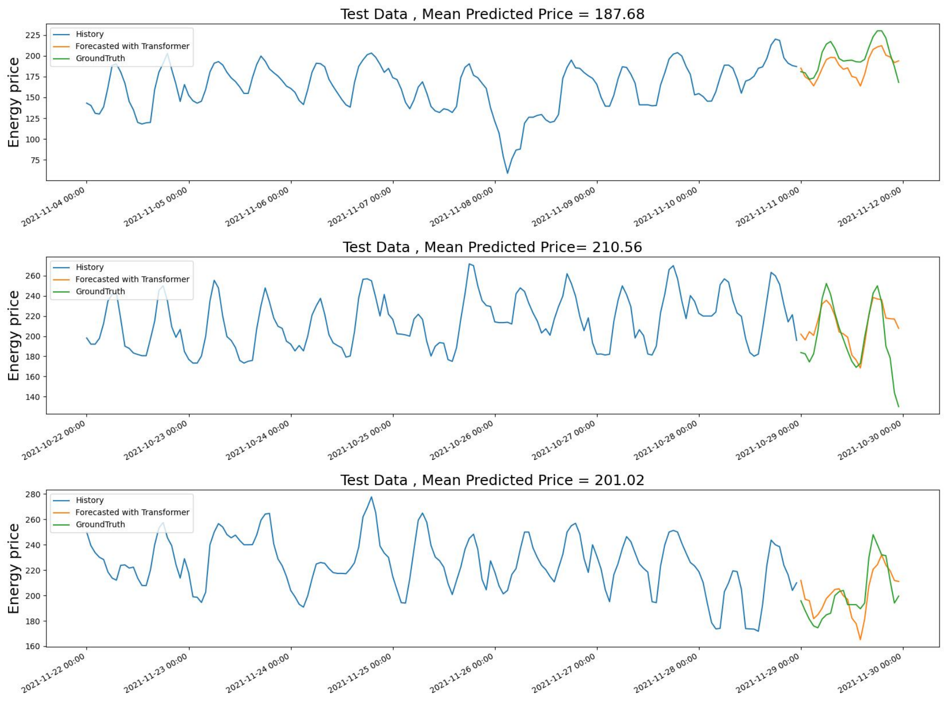


Figure 17: Transformer Predictions: A closer look at the day-ahead predictions of the best performing Transformer model showing examples in which the model performs well and in which the model does not

tions of the best performing Residual MLP model using normal distribution. The shaded section in the graphs shows the 80% confidence interval of the predicted distribution. This gives an idea about how confident the model is in its prediction.

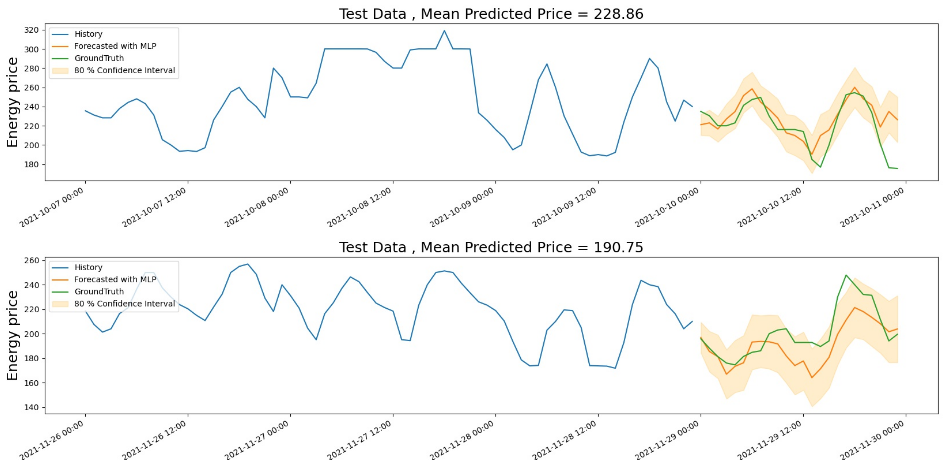


Figure 18: Probabilistic Predictions

6 Conclusion

6.1 Summary

Based on the experiments carried out and the reported results, we can draw the following conclusions:

- **Transformers are effective in time series forecasting:** Transformers were initially developed for NLP tasks and are state-of-the art. We have shown that Transformers can also be modified to address time series forecasting tasks. Among the models we have trained, Transformers had the least MAE. It even outperformed our LSTM model. This is quite promising since Transformers can be trained faster. The multi-head attention mechanism performs well in learning the dependencies in the input data.

- **Probabilistic prediction further decreased the error.** It is observed that converting a deterministic model to a probabilistic one, has hugely decreased the error of the residual MLP model. A comparison of the CRPS and MAE scores shows the difference. So, a probabilistic model not only helps in reducing the error but shows how confident the model is in its predictions. While the error did not decrease for the probabilistic transformer model, it is still useful to know the uncertainty in the model's prediction.

- **Energy price prediction is challenging:** One thing that we observed is that energy price prediction poses many challenges. Especially, in recent times it has been observed that prices have drastically increased. This can be attributed to the pandemic and the Ukraine war crisis. Such sudden events cannot be included within the data and it cannot be anticipated as well. It would be challenging for the model to deal with such events that affect energy prices, that could occur in the future.

6.2 Future Works

- **More experiments on probabilistic models:** The probabilistic models that are used still struggle a bit to fit the data well. This can be observed by the difference in the expected quantile difference percentage to the actual percentage. It would be worth looking into other distributions as well.

- **Data augmentation techniques:** More efficient means of data augmentation can be used.

- **Transformer encoder-decoder architecture:** While the transformer encoder only model has the best accuracy among our experiments, it could be worth checking if adding a decoder helps improve the accuracy.

- **Testing more hyperparameter settings:** It could be worth it to spend more time testing the hyperparameters of the models and observe how the accuracy is affected.

7 Acknowledgments

First and foremost, I would like to thank:

- Matthias Hertel and Oliver Mey for all the support and guidance throughout my thesis and over the last one year.

- Prof. Dr. Hannah Bast and Prof. Dr. Frank Hutter for being my examiners and agreeing to review my thesis.

- The Chair for Algorithms and Data Structures for providing me access to GPU clusters and for all the technical support.

- André Schneider for mentoring me during Oliver's absence.

- My boyfriend, friends and family for their emotional support.

- Rusha Gupta for proofreading my thesis.

Bibliography

[1] S. Senthil, “Energy price forecasting,” 2022. Available at `https://ad-blog.cs.uni-freiburg.de/post/energy-price-forecasting/`.

[2] A. Khodabakhsh, I. Ari, M. Bakır, and S. M. Alagoz, “Forecasting multivariate time-series data using lstm and mini-batches,” in *Data Science: From Research to Application* (M. Bohlouli, B. Sadeghi Bigham, Z. Narimani, M. Vasighi, and E. Ansari, eds.), (Cham), pp. 121–129, Springer International Publishing, 2020.

[3] N. Wu, B. Green, X. Ben, and S. O’Banion, “Deep transformer models for time series forecasting: The influenza prevalence case,” *CoRR*, vol. abs/2001.08317, 2020.

[4] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, “Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting,” *CoRR*, vol. abs/2201.12740, 2022.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS’17, (Red Hook, NY, USA), p. 6000–6010, Curran Associates Inc., 2017.

[6] G. E. P. Box and G. M. Jenkins, “Some recent advances in forecasting and control,” *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 17, no. 2, pp. 91–109, 1968.

[7] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International Journal of Forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[8] M. Hu and S. U. D. of Electrical Engineering, *Application of the Adaline System to Weather Forecasting.* Stanford University, 1964.

[9] R. Díez and J. Parreño, "Forecasting the price of energy in spain's electricity production market," 09 2005.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*, p. 318–362. Cambridge, MA, USA: MIT Press, 1986.

[11] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, pp. 339–356, 1988.

[12] R. L. Wilson and R. Sharda, "Bankruptcy prediction using neural networks," *Decision Support Systems*, vol. 11, no. 5, pp. 545–557, 1994.

[13] R. Pino, J. Parreno, A. Gomez, and P. Priore, "Forecasting next-day price of electricity in the spanish energy market using artificial neural networks," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 1, pp. 53–62, 2008.

[14] I. P. Panapakidis and A. S. Dagoumas, "Day-ahead electricity price forecasting via the application of artificial neural network based models," *Applied Energy*, vol. 172, pp. 132–151, 2016.

[15] S. Anbazhagan and N. Kumarappan, "Day-ahead deregulated electricity market price classification using neural network input featured by dct," *International Journal of Electrical Power and Energy Systems*, vol. 37, no. 1, pp. 103–109, 2012.

[16] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "Deepar: Probabilistic forecasting with autoregressive recurrent networks," *International Journal of Forecasting*, vol. 36, no. 3, pp. 1181–1191, 2020.

[17] J. Torres, F. Martínez-Álvarez, and A. Troncoso, "A deep lstm network for the spanish electricity consumption forecasting," *Neural Computing and Applications*, vol. 34, pp. 1–13, 07 2022.

[18] A. Borovykh, S. Bohte, and C. W. Oosterlee, "Conditional time series forecasting with convolutional neural networks," 2017.

[19] Z. Khan, S. Fareed, M. Anwar, A. Naeem, H. Gul, A. Arif, and N. Javaid, *Short Term Electricity Price Forecasting Through Convolutional Neural Network (CNN)*, pp. 1181–1188. 03 2020.

[20] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y. Wang, and X. Yan, "Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting," *CoRR*, vol. abs/1907.00235, 2019.

[21] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, "A transformer-based framework for multivariate time series representation learning," *CoRR*, vol. abs/2010.02803, 2020.

[22] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, "Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting," *CoRR*, vol. abs/2201.12740, 2022.

[23] Y. Chen, Y. Kang, Y. Chen, and Z. Wang, "Probabilistic forecasting with temporal convolutional neural network," *Neurocomputing*, vol. 399, pp. 491–501, 2020.

[24] L. Zhu and N. Laptev, "Deep and confident prediction for time series at uber," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, IEEE, nov 2017.

[25] A. Koochali, P. Schichtel, S. Ahmed, and A. Dengel, "Probabilistic forecasting of sensory data with generative adversarial networks - forgan," *CoRR*, vol. abs/1903.12549, 2019.

[26] A. Koochali, A. Dengel, and S. Ahmed, "If you like it, GAN it. probabilistic multivariate times series forecast with GAN," *CoRR*, vol. abs/2005.01181, 2020.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[28] Y. Romano, E. Patterson, and E. J. Candès, "Conformalized quantile regression," 2019.

[29] H. Hersbach, "Decomposition of the continuous ranked probability score for ensemble prediction systems," *Weather and Forecasting*, vol. 15, no. 5, pp. 559 – 570, 2000.

[30] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich, "A survey on long short-term memory networks for time series prediction," *Procedia CIRP*, vol. 99, pp. 650–655, 2021. 14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 15-17 July 2020.

[31] G. Petneházi, "Recurrent neural networks for time series forecasting," *CoRR*, vol. abs/1901.00069, 2019.