University of Ljubljana
Faculty of Computer and Information Science

Kristjan Sešek

Electricity consumption forecasting using deep neural networks

MASTER'S THESIS

THE 2ND CYCLE MASTER'S STUDY PROGRAMME
COMPUTER AND INFORMATION SCIENCE

Supervisor: prof. dr. Marko Robnik Šikonja

Ljubljana, 2018

Ste zaprosili za pisanje naloge v angleščini? Če ne, oddajte vlogo čimprej.

Chapter 1

Introduction

omnipresent in tedhics and natural sciences.

Time series are ~~everywhere around us.~~ Every physical measurement can be listed in time order, which effectively makes it a time series. We can significantly improve and optimize our lives, if we are able to successfully predict time series. Time series are widely used in statistics, finance, meteorology, astronomy and energy sector. We will primarily focus on energy sector, specifically on electricity consumption forecasting.

Stability and response of the electrical grid is directly connected with our ability to forecast electricity consumption. Improved forecast accuracy leads to more dynamic electricity pricing and it also minimizes the chance of grid overload. Grid managers can order additional resources or motivate consumers to consume less in electricity in a variety of innovative products. We will focus on short term forecasting (one day ahead).

4. V podpoglavje ali poglavje: sorodna dela
1. opišite na kratko osnovne ideje in težave pri napovedovanju porabe električne energije
2. Opišite na kratko, kaj želite doseči in kako ter [1] konkreten problem, ki ga rešujete.
3. Napovejte vsebino dela po poglavjih

Poglavje razbijte na dve poglavji: dela.
- najprej opis podatkov, njihovo čiščenje itd.
- nato gradnje, elbow, vizualizacija, autokorelacija

Chapter 2

Data

Vsako poglavje naj ima uvodni del, kjer napovete vsebino

2.1 Data set

We based our analysis on the public data set of 6000 time series [1] to ensure simple reproduction and integrity of the results. Time period of the data is from January 2012 until March 2014. Image 2.1 shows number of measurements for each date and time. Notice that the number of measurements is doubled every few periods. These are duplicated values that we filtered out. We can observe that the amount of measurements stabilizes around August 2012. Some time series joined the project later and this caused the delay. We removed all values prior to August 2012, because we wanted to ensure that all time series forecasts are done in the same period and with the same amount of measurements. Some consumers have small periods of missing data, which we filled with *pad* function of *Pandas.DataFrame*. The final data set for time series cross validation is on the image 2.2. Data set also includes weather and holiday data for the same time period.

Opišite najprej, kaj predstavljajo meritve, za koga veljajo (gospodinjstva, podjetja, ...) in geografsko lokacijo

Opišite razbitje na učno/validacijsko in testno množico

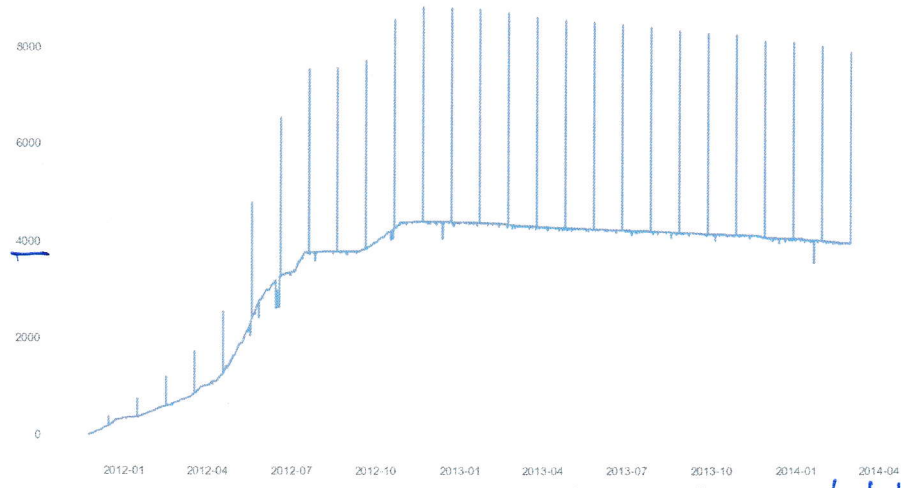


Figure 2.1: Number of measurements per date time ~~on~~ initial data set

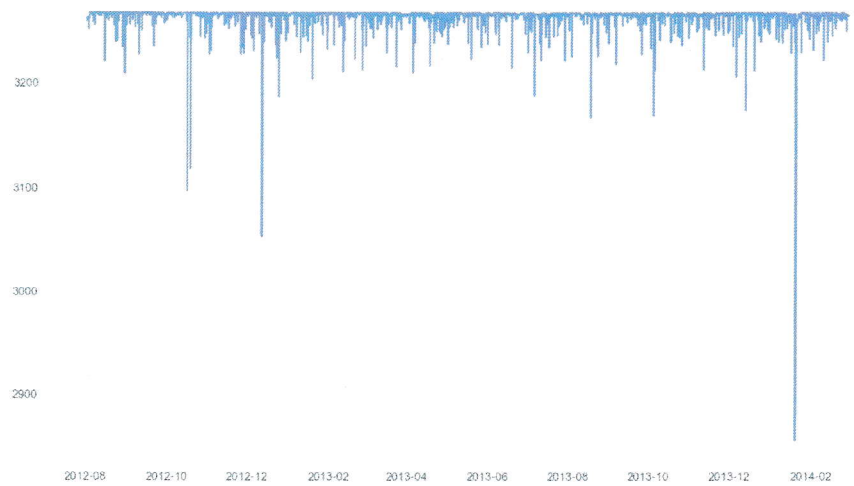


Figure 2.2: Number of measurements per date time ~~on~~ final data set

2.2 Data analysis

Time series measurements are collected in the metering devices every 30 minutes, however they don't stream the data directly to the server. Time series data for each device is sent at the end of the day, at midnight. This is important, because we will be making forecasts one day (48 measurements) ahead. One of the most important features for time series forecasting is lagged target variable. We can get very successful forecasts, if we are allowed to use a feature with lag value of one (last measurement). However because of the nature of our problem we must not use any feature with a lag smaller than 48. If we did, we would be looking in the future and using information that is not yet known to us in practice. We will cover this in depth in Chapter 3.

2.3 Data visualizations

Final data set contains 3500 time series, which is very hard to visualization and evaluate. We encountered this problem when we tried to plot autocorrelation and partial autocorrelation for all time series on one image. It is impossible to see which lag sizes are dominant and should be used as a lagged target feature.

We started with the assumption that there are probably many similar time series and subplot of those should give us better insights. We used clustering to group similar time series together. Notice that we only used cluster for visualizations. Data preprocessing and predictions were made for each time-series separately.

There are many metrics for measuring time series similarity. We defined time series similarity as difference between two normalized time series. We didn't use dynamic time warping, which is one of the most common ways to assess time series similarity, because we are interested in grouping time series

(consumers), who behave similarly at the same point in time.

Time series were normalized prior to clustering, because two time series can behave similarly at the same point in time, but may have different consumption measurement. We want to group these time series together.

These time series will also have similar autocorrelations and partial autocorrelations, which we can easily compare in one chart.

2.3.1 K-means clustering

K-means clustering is a tool for grouping N-dimensional vectors into predefined number of groups. In the first run K-means algorithm assigns initial random N-dimensional mean vector to each group. The next step is to calculate the distance between group and time series and assign each time series to the closest group. Mean vector of each group is recalculated based on the N-dimensional vectors in the group. Time series are then reassigned to the closest group based on the distance. This is repeated until convergence.

We represent each of the 3500 time series as a vector with 25.000 dimensions. We used *k-means elbow method* [2] to determine optimal number of groups. On the image 2.3 we can see the result of the *k-means elbow method.* We divided 3500 time series to 70 groups, because more groups didn't satisfy our goal of simplified visualizations and and less groups didn't show enough difference between each time series in the group.

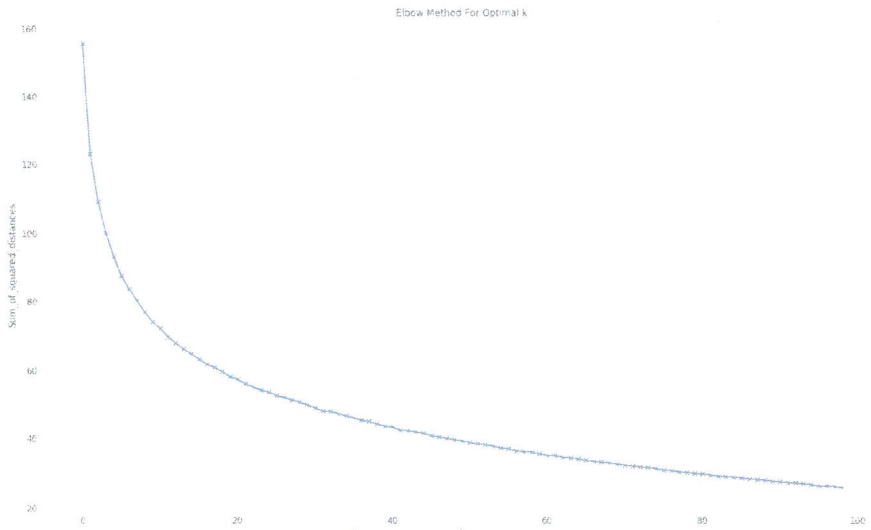


Figure 2.3: Result of the clustering with k-means elbow method

2.3.2 Autocorrelation

Autocorrelation measures how given time series is correlated with itself. It is used to find repeating patterns in time domain. Each value is tested against its previous values for all predefined lag sizes. It is not well-defined for all time series, because it is based on mean and variance reduction and they may not exist or be zero. Our dataset is stationary with well defined mean and non zero variance and we can use autocorrelation to measure which past values are the most useful for predicting the future. Image 2.4 displays time series autocorrelations for all lag values up to one month. Notice that there is very high correlation on every 48 measurements, which is exactly one day. This was expected, since data the best approximation for the future is the value at the current time on the last day. This provided us with the starting point for target variable feature engineering. Image 2.5 displays one week autocorrelation chart where 48 cycle is even more evident.

vrednost prejšnjega dneva naj se uporabi kot privzeta vrednost za primerjavo

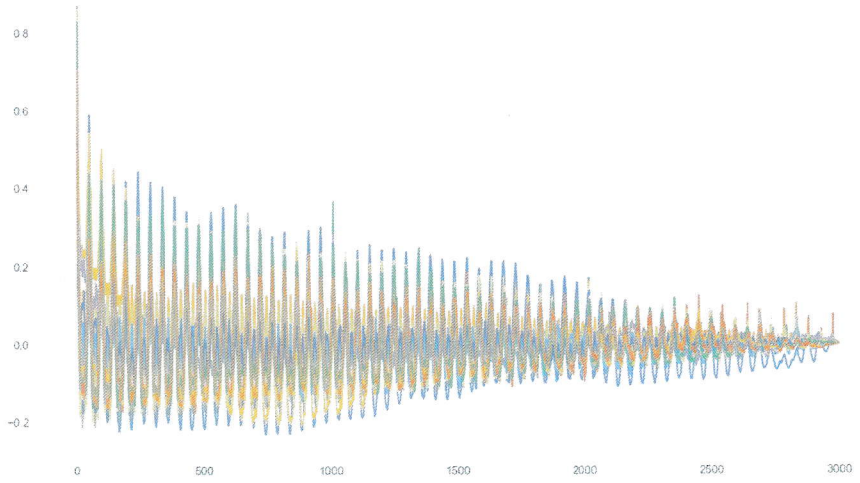


Figure 2.4: Time series autocorrelation and lag up to one month

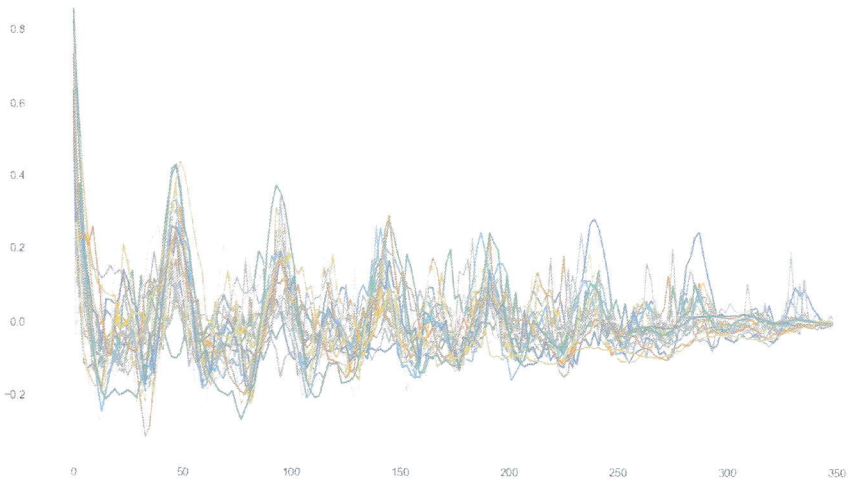


Figure 2.5: Partial autocorrelation of each time series and lag up to one week

Chapter 3

Feature engineering

+ introduction on motivation and contents of the chapter

3.1 Existing features

this shall be described in the data Chapter

Consumption and weather measurements were loaded into *Pandas.DataFrame* and joined by date and time index. Original data provided us with eleven weather features and consumer consumption measurements as target variable.

3.2 New categorical features

Two categorical features were already present in the data set (precipitation type and summary). We used *Pandas.get_dummies* to create new boolean features for each category, because they are independent from each other. We also engineer the following categorical features from date and time: holiday, weekend, morning, midday, afternoon, evening and night.

3.3 New numerical features

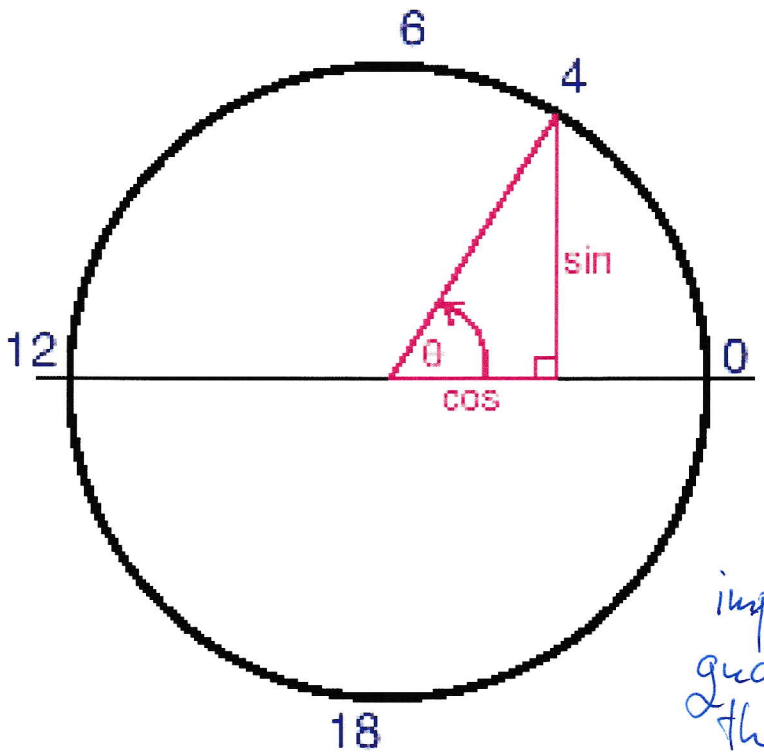
3.3.1 Numerical features from date time

We also extracted hour, day of the week, day of the month, month and season from date and time. However, these features are not cyclical and not independent from each other. Simple integer representation for date and time features is not accurate, because it doesn't account for the fact that the first and the last values are sequential in human notation of date and time. We want to preserve information that hour 23 and hour 0 are neighbours. Current integer representation doesn't account for that. We solved this by transforming each cyclical date and time feature to unit circle with sinus and cosinus. The first and the last value are now sequential. An example of hour feature mapped onto the unit circle is displayed on image 3.1.

3.3.2 Numerical features from target variable

We can generally make two types of predictions: classification and regression. The first is about predicting a label for given features and the second is about predicting a quantity for given features. The nature of our problem suggests the use of regression models, because our target variable is decimal number. We could transform our target variable to predefined categories and use classification models, but this is out of scope of this research.

The most popular features in regression models are constructed from lagged target variable. Some prediction models like autoregression rely solely on them. As discussed in Chapter 2, we must be very cautions on how to construct features from target variable. We will be making predictions for one day (48 values) ahead. This means that we must construct features for 48 values ahead. We can easily construct date and time and weather features, because they are not time dependent. In the example below 5.1 we will first try to build lagged target variable with lag size one.



improve the quality of the image.

Figure 3.1: Hour feature mapped to unit circle by sinus and cosinus

24- is the = provide information on its cycling nature. of time.

Table 3.1: Train and test features with lagged target variable with lag size one.

month	day	hour	lagged variable with lag size one	Target variable
1	1	1	None	5
1	1	2	5	8
1	1	3	8	9
1	1	4	9	3
1	1	5	3	PREDICTION
1	1	6	?	PREDICTION
1	1	7	?	PREDICTION
1	1	8	?	PREDICTION

As we can see in the Table 5.1, we can't use lagged target variable with lag size one, because we don't know lagged value for last three rows. There are three common ways to solve this problem.

- we can use the predicted value from the last row, but each prediction contains an error and since we are making 48 predictions ahead, we may not want to stack up that much error in each iteration
- we could use the same value as in the row before, also known as last known value, but this contradicts the way we train the model, because in the training phase we always use the previous value, not two, or three or even more values behind the previous value. We would need to add additional features that would account for two, three or more values in the past
- we can use value with the lag of forecast size. Since we are forecasting 48 values in advance, we will start with lag size of 48 (one day ago).

All features were normalized to ensure that prediction models wouldn't put more significance to specific feature, just because it has bigger absolute values.

Table 3.2: Final feature set

Name	Type
visibility	float
wind bearing	float
temperature	float
dew point	float
pressure	float
apparent temperature	float
wind speed	float
precipitation type	string
humidity	float
summary	string
consumption	float
holiday	boolean
weekend	boolean
morning	boolean
midday	boolean
afternoon	boolean
evening	boolean
ngiht	boolean
sin_hour + cos_hour	float
sin_weekday + cos_weekday	float
sin_day + cos_day	float
sin_month + cos_month	float
sin_season + cos_season	float
lagged_value_of_48	float

Chapter 4

Cross validation

Cross validation is a way to assess how particular prediction model will perform in practice. We can achieve this by splitting our data to a set of known and unknown data. This prevents overfitting and gives us a glimpse of how the model will generalize to unknown data set in production. Generally data in cross validation is split randomly by some fraction, however time series data sets require special care, because data order is important.

4.1 Time series cross validation

Time series cross validation simulates the way that prediction model would be used in practice. Prediction model is first trained on the first part of time ordered data set as seen on the image 4.1. Size of the training data set is predefined based on domain knowledge and amount of data. Our data set includes one and a half year of data for each time series. We chose one year as starting point for our time series cross validation, because it represents one complete time series cycle. This way we can teach the model how each time series behaves in different months and seasons. Time series cross validation walks forward in time and trains on the values from the past and predicts in the future. This is repeated until the end of the data set.

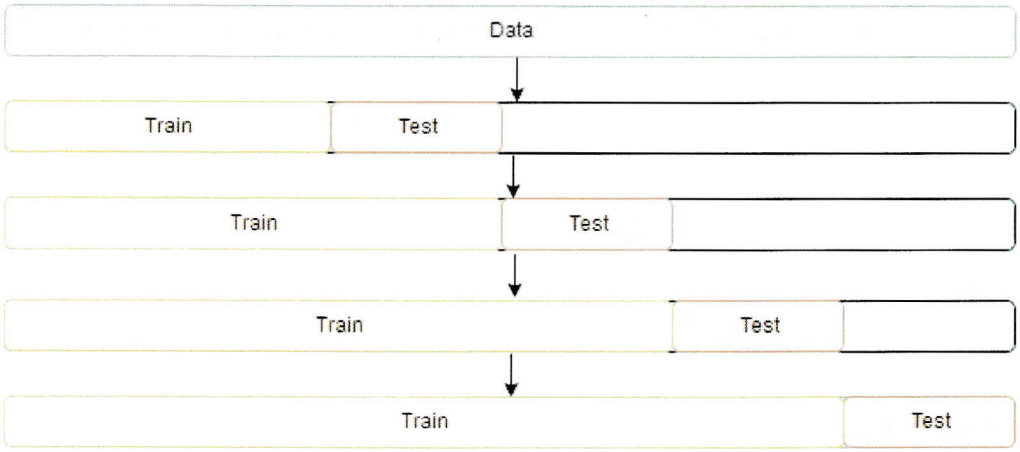


Figure 4.1: Time series cross validation [referenca na sliko]

```
FORECAST_SIZE = 48
TRAIN_START_IDX = 48 * 365
for model in models:
 model_rmse = 0
 for c in consumers:
  n = len(c)
  consumer_rmse = 0
  X, y = model.transform_data(c)
  for step in range(TRAIN_START_IDX, n - FORECAST_SIZE, FORECAST_SIZE):
   y_hat = model.get_forecast(X, y, step)
   y_actual = y[step:step + FORECAST_SIZE]
   consumer_rmse += mean_squared_error(y_actual, y_hat) ** 0.5

 nr_of_steps = (n - FORECAST_SIZE - TRAIN_START_IDX) / FORECAST_SIZE
 model_rmse += consumer_rmse / nr_of_steps
```

pojasnite v besedilu

Each prediction model undergoes the same time series cross validation technique and must comply to the predefined model interface defined below:

```
class AbstractModel:
    def get_prediction(x_values, y_values, step):
        pass
    def transform_data(df):
        pass
```

Each prediction model implements its own data transformation which is then fed to the *get_prediction* function. It trains and optionally caches the model ~~and~~ before returning prediction for given iteration. Faster prediction models like linear regression can afford to retrain the model on each iteration, however in the case of neural network the model is cached.

+ opišite različne delitve train / validate / test

+ opišite učenje različnih granularnosti: vsaka TS posebej, po gručah, vsi skupaj

Chapter 5

Prediction models

5.1 Traditional models

The first step was to implement the simplest naive model and three of the best performing models for consumer electricity prediction that were defined in [3]. Our research was based on the new data set and we needed to make sure that previous models actually worked. All previous models were implemented with the help of *sklearn* library.

5.1.1 Naive model

Naive model was used as a baseline for traditional models. The model assumes that the next 48 measurements will be the same as the last 48 measurements and therefore just forwards past data to the output.

5.1.2 Previously best performing models

Our engineered features were enough to be directly plugged to previously best performing models. We grid searched the best basic hyperparaters for these models. We used **linear regression** with a different regularization parameter to prevent overfitting. Linear regression model was very fast and it was possible to retrain it on each iteration, which resulted in significant improve-

ment of RMSE. **Random forest** was grid searched for number of estimators and **autoregressive integrated moving average** was searched for best auto regressive (max lag) and integrated (moving average) component. We skipped differencing component, because our time series are stationary.

5.2 Neural networks

Most traditional prediction models are heavily relying on the user constructed feature set. These models know how to learn which features are more important and which are less based on the given target variable. The problem is that none of these models can change these features or even construct new ones. Neural networks are able to create new features based on provided original features. They can also recursively expand these new features to arbitrary number of predefined levels. These neural networks are called **deep neural networks**. Apart from input layer, which is provided by us and output layer, which is given as a result, they also contain any number of hidden layers. These layers are used by the model to determine which relationships between input layers or other hidden layers are significant for given target variable. This allows deep neural networks to construct its own feature set that minimizes predefined error function.

5.2.1 Long short term memory neural networks

Long short-term memory neural networks are a family of recurrent neural networks. They contain cells with input, output and forget gates that enable them to regulate information flow 5.2. Each cell takes an input and stores it for some period of time. This allows it to model temporal values for time series. They can use their stored state to affect the calculations in the future. The problem are recurrent neural networks with multiple hidden layers. Neural network gradients are multiplied several times when they are propagated back to the initial layer to retrieve long-term dependency information. If initial gradient values are less then one, this results in even

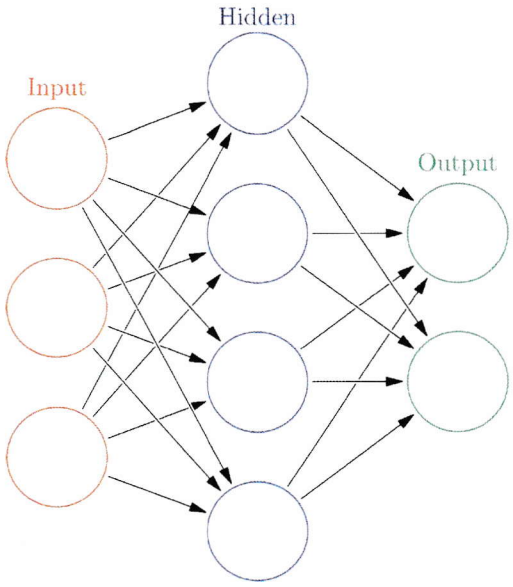
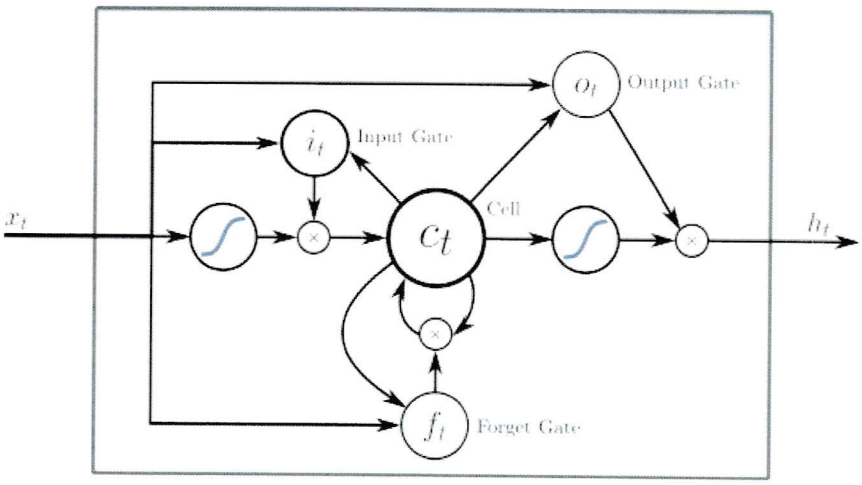


Figure 5.1: Neural network architecture

smaller gradient that may vanish. On the other hand, if the initial gradient values are bigger than 1, this results in even bigger gradient that may overflow. Long short-term memory neural networks solved this by removing the exponentially fast vanishing or exploding gradient. They are ~~most~~ useful for time series predictions, because they can easily generalize and learn which lag values are connected and significant for given target variable.

5.2.2 Convolutional neural networks

Convolutional neural networks execute mathematical dot product with predefined kernel on the inputs of the neurons in the hidden layers. Pooling is optionally applied to achieve non-linearity. General convolutional neural network is promising for time series, because it can model spatiotemporal user defined features. We built models for simple 1D and 2D convolution and then proceeded to convolutional neural network specifically for time series prediction [4].



improve quality,

Figure 5.2: LSTM cell give reference to figure source.

5.2.3 Ensembles

Ensembles can generally improve prediction accuracy by stacking individual prediction models together and since we built many different neural network models the next logical step would be to stack them and use them as ensembles. We implemented the ensambles defined by Kourteszes and coworkers [5]. They propose the following ensemble operators:

- mode
- median
- mean

5.3 Results

The following models were built and tested against 3500 time series. Each model was grid searched for the most common hyperparameters.

- naive model

- linear regression with ridge regularization
- random forest
- autoregressive integrated moving average
- LSTM NN
- Stacked LSTM NN
- 1D convolutional NN
- 2D convolutional NN
- Undecimated Fully Convolutional NN
- Ensemble

komentirajte razultate, statistično pravilno primerjajte različne metode, komentirajte praktično uporabnost

Table 5.1: RMSE results per model

model	RMSE
naive	0.2342
LR(alpha=0.01)	0.2015
LR(alpha=0.025)	0.1995
LR(alpha=0.05)	0.2003
LR(alpha=0.075)	0.2133
LR(alpha=0.1)	0.2301
RF(n_estim=10)	0.2112
RF(n_estim=25)	0.1955
RF(n_estim=50)	0.1964
RF(n_estim=75)	0.1970
RF(n_estim=100)	0.1990
LSTM(hidden=64, days2see=7, epoch=200, batch_size=64)	0.1933
LSTM(hidden=64, days2see=30, epoch=200, batch_size=64)	0.1925
LSTM(hidden=128, days2see=30, epoch=200. batch_size=64)	0.1901
StackedLSTM(hidden=64, days2see=7, epoch=200, batch_size=64)	0.1905
StackedLSTM(hidden=64, days2see=30, epoch=200, batch_size=64)	0.1900
StackedLSTM(hidden=128, days2see=30, epoch=200, batch_size=64)	0.1898
1DCNN(hidden=64, days2see=7, epoch=200, batch_size=64)	0.1835
1DCNN(hidden=64, days2see=30, epoch=200, batch_size=64)	0.1834
1DCNN(hidden=128, days2see=30, epoch=200, batch_size=64)	0.01798
2DCNN(hidden=64, days2see=7, epoch=200, batch_size=64)	0.1875
2DCNN(hidden=64, days2see=30, epoch=200, batch_size=64)	0.1899
2DCNN(hidden=128, days2see=30, epoch=200, batch_size=64)	0.1892
UFCNN(epoch=200, batch_size=64)	0.3342
UFCNN(epoch=100, batch_size=128)	0.4544
Ensemble(mode)	0.1855
Ensemble(median)	0.1867
Ensemble(mean)	0.1825

Bibliography

[1] U. P. Networks, Smartmeter energy consumption data in households [dostopano 6.1.2017], https://data.london.gov.uk/dataset/smartmeter-energy-use-data-in-london-households.

[2] Elbow method (clustering) [accessed 20.8.2018], https://en.wikipedia.org/wiki/Elbow_method_(clustering).

[3] J. Kraljič, Forecasting the electricity consumption data stream, Diplomsko delo, Fakulteta za računalništvo in informatiko, 2011.
URL http://eprints.fri.uni-lj.si/1375/

[4] R. Mittelman, Time-series modeling with undecimated fully convolutional neural networks, CoRR abs/1508.00317.

[5] N. Kourentzes, D. K. Barrow, S. F. Crone, Neural network ensemble operators for time series forecasting, Expert Systems with Applications 41 (9) (2014) 4235 – 4244.

+ razširite sokodne raziskave