# Introduction to Intelligent Systems (02461) Exam Project: Covid-19 Infections Forecasting: Univariate Time Series Prediction using an LSTM Neural Network

Johan B. Hanehøj (s194495)
August Brogaard Tollerup (s204139)
Andreas L. Fiehn (s204125)

Januar 2020

This project report was written collectively. For individualization purposes we have provided a table with the main responsible for each section:

| | |
|---|---|
| 1 Abstract | Johan B. Hanehøj |
| 2 Introduction | Johan B. Hanehøj |
| 2.1 Description of the LSTM neural network | August B. Tollerup |
| 3 Method | Andreas L. Fiehn |
| 4 Results | Johan B. Hanehøj |
| 5 Discussion | First half: Andreas L. Fiehn, Second half: August B. Tollerup |
| 6 Learning outcome | - |

# 1 Abstract

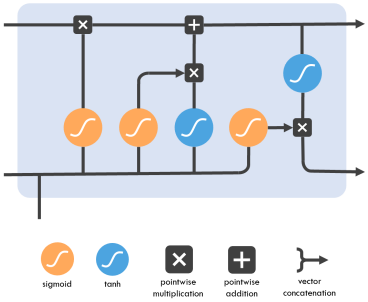
According to the World Health Organisation there has been almost 100 million cases of the novel coronavirus worldwide and around 2 million deaths caused by the virus have been reported [1]. Economies have been paralyzed and whole sectors have gone out of business. Predicting or forecasting how much the virus will spread in the future is essential if we want to take the right preventative measures and minimize lives lost and the damage done to society. Organizations and governmental institutions across the world have been attempting to create models for predicting and combating the viral outbreak.
We wish to explore how well we can predict future corona cases with a neural network by looking at a time series of new daily positive cases. For this we will be using an Long Short-Term Memory (LSTM) recurrent neural network (RNN). In our research we found that the model was able to predict some tendencies but not the unpredictable spikes that are often found in epidemiological data. And thus we can conclude that univariate time forecasting using an LSTM model is very limited on anomalous data and would not suffice as a tool for combating viral outbreaks with the amount of data we have at our disposal.

# 2 Introduction

In a highly urbanized global society viral outbreaks are not a rare occurrence. The effects of a pandemic can have long-lasting and detrimental impact on the world for years to come. Although these outbreaks can be very difficult to prevent from emerging, being able to take measured actions accordingly using AI-based forecasting could be an invaluable tool to minimize the over-all damage of a pandemic. Although an AI could prove to be crucial in the endeavour to stagger disease spread, we also discuss the ethical implications on personal information, such AI would have to be trained on for it to be reliable. Through a Recurrent Neural Network, such as the LSTM model, we hypothesize that we will be able to make approximations for Covid-19 infections based on previous data.

## 2.1 Description of the LSTM neural network

Figure 1: Illustration of an LSTM gate [2]



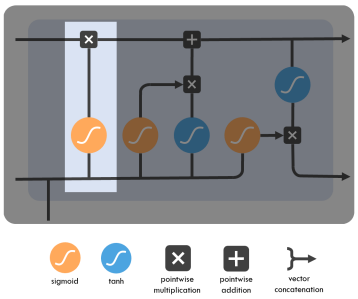

In general, conventional recurrent neural network (RNN) and neural network (NN) models suffer from the "vanishing gradients" problem. Vanishing gradients occur when neural networks use activation functions such as sigmoid or tanh. This is intuitively seen if: $y = \text{sigmoid}(x)$, has a very small or negative x as input, it will output a very small y. Throughout training, the NN will "prioritize" recent data more than prior data, therefore the gradients and weights associated with data from the early stages might diminish. When working with time series it is therefore imperative to preserve chronology and data from the first part of a data-set, because it tends to be just as relevant as recent data, hence the use of the Long-Short Term Memory model (LSTM). Instead of a single cell like like we see in conventional RNN's or NN's, the LSTM uses so called "gates" (See Figure 1) which modifies the candidate data to be added to the underlying NN. The LSTM model consists of 3 gates and a "highway" which will be explained in the following sections.

### 2.1.1 The Highway

The Cell Highway acts as a highway of information from neuron to neuron. The upper most vector represents the highway through each neuron on which a "cell" can travel. The cell carries information from past neurons. This means that information from the very first neuron has the potential to still be relevant at a much later state. Throughout the highway the cell is altered by the Forget Gate and the Input Gate.

### 2.1.2 The Forget Gate

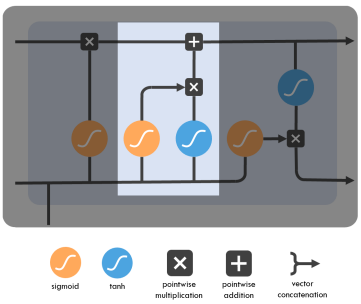
Figure 2: Focus on the forget gate in a neuron in the LSTM model




The Forget Gate determines which data the neuron should forget or deprioritise. This gate receives and concatenates input from the previous hidden layer (lower-left branch) and the current input (bottom) which it then scales or evaluates through a sigmoid function (See Figure 2). This function adjusts the incoming data from the hidden layer and the input to be between 1 and 0. The closer a value is to 0 the more that information will be "forgotten", effectively resulting in that gradient having less influence. To apply the changes from the Forget Gate, the new evaluated data is multiplied with the cell on the highway, implementing the new weights.

### 2.1.3 Input Gate

Figure 3: Focus on the input gate in a neuron in the LSTM model




The Input Gate, like the Forget Gate, receives input from the previous hidden state as well as the current input. This information is then duplicated and passed through 2 activation functions; sigmoid and hyperbolic tangent, which is combined via multiplication to implement the "forgetfulness" or evaluative functionality of the sigmoid function (See Figure 3). From this the combined new result, referred to as a candidate, is added to the cell on the highway.
At this point the cell has received feedback on which data to "forget" and which to add to its current state. From an algebraic standpoint, the cell can be represented by the following:

$$C_s = F \times C_{s-1} + i_s \times j_s$$

From this figure one should understand the cell in the current state ($C_s$) which is, **firstly**, formed by addition of the output from the Forget Gate (F) multiplied by the prior cell's information ($C_{s-1}$) and, **secondly**, the product of the output from both the sigmoid- and tanh function, forming the candidate.

### 2.1.4 The Output Gate

The Output Gate acts as the last link in the chain of events. Like the prior 2 gates the Output Gate also receives the concatenated information from the previous hidden state and the current input, which in turn is passed through a sigmoid function like the Input Gate. Additionally, the Output gate receives the current information of the cell on the highway passed through a regulating tanh function (See Figure 4). Therefore, the Output Gate acts as a "flush" of all

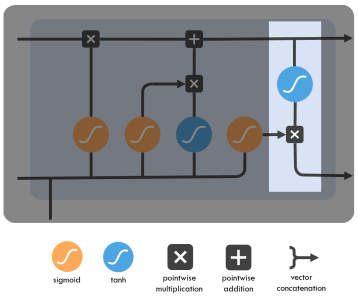



Figure 4: Focus on the output gate in a neuron in the LSTM model

the information in the current state into the new hidden state, which is passed parallel to the cell on the highway, to the next neuron to repeat the entire procedure.

# 3 Method

We trained and tested our model on new daily positive Covid-19 cases from SSI. We downloaded data ranging from the 27th of January 2020 to the 4th of January 2021 - almost a year's worth of data. We trained our model using an 80/20 train-test split.[1]

The data was scaled using min-max normalization which maps the features in the range [-1,1]. This was done by subtracting the minimum feature and dividing by the min-max range. For this we used the scikit learn library.[3]

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

In order to train the neural network we needed to define input and output features. This was done by sequencing the data into sequence lengths of 14 days (input) and using the subsequent day as our output feature. We overlapped the sequences so that our model was trained on every day in our data set. (See figure Figure 5)

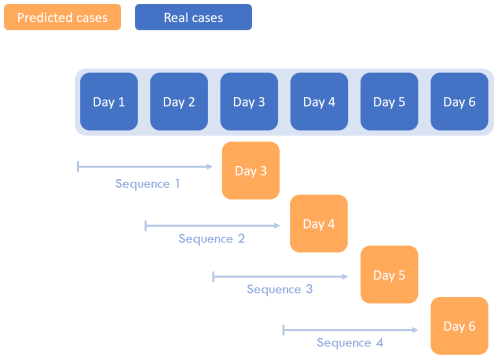



Figure 5: An example of the sequencing. Here the sequence size is 2, which means the model takes in two days as its feature and tries to predict the subsequent day

Our model used a mean squared loss function (MSE), which was the one we wanted to minimize by finding the optimal weights for our network.

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

This optimization was done by using a stochastic gradient-descent algorithm, which is built into Pytorch. [4]

As a part of our experiment design we performed single-variable control by adding a looping feature to our code that

[1] Click: See line 175 in code

changed each parameter separately [2]. We looped though the following learning rates $[0.1, 0.2, 0.3, 0.4, 0.5]$ and number of neurons $[1, 2, 3, 4, 5, 6]$. We kept all other model parameters constant. This procedure resulted in 30 plots (made with matplotlib) for train/test loss as well as predictions. From this data we were able to choose the parameter combinations with the best results. For our results section we picked the 9 most promising results for further analysis.
Furthermore we calculated how much the predicted amount of cases deviated from the real amount. The calculation was conducted on the whole test split and we determined the average deviation in percentage:

$deviation_{avg} = \frac{1}{n} \cdot \sum_{i=1}^{n} (\frac{\hat{y}_i - y_i}{\hat{y}_i} \cdot 100\%)$

Where $n$ is the amount of days we have predicted, $\hat{y}$ is the real cases and $y$ is the predicted cases
In order for us to validate our model we we tested it on flight passenger data from the Python Seaborn module [5]. This data consists of the number of passengers who flew in each month from 1949 to 1960. [6]. We chose the data because of its predictable trends and less anomalous nature. We trained the model by sequencing the data into a length of 12, corresponding to each of the 12 months of data (input) and used the subsequent month as our target value (output). We created the model as a benchmark for our LSTM model's performance and we will not discuss the results in this report. It can be found on Github [3].

To make a reference we also made a simple prediction by taking the difference of two days and adding to the latest day to predict the next.
$y_n = +y_{n-1} + (y_{n-1} - y_{n-2})$
For this we also calculated the average deviation percentage.

# 4 Results

Our LSTM model predictions for average deviation and computation time using different values for learning rate, hidden layers as well as neurons per layer are given in the table below:

| **Sequence length (days)** | **Learning rate** | **Hidden layers** | **Neurons per layer** | **Average deviation (%)** |
|---|---|---|---|---|
| LSTM Neural Network Predictions | | | | |
| 14 | 0.5 | 1 | 4 | 40.28 |
| 14 | 0.2 | 1 | 5 | 30.06 |
| 14 | 0.2 | 1 | 4 | 20.02 |
| 5 | 0.5 | 1 | 5 | 22.95 |
| 5 | 0.4 | 1 | 5 | 22.59 |
| 5 | 0.4 | 1 | 3 | 25.48 |
| 1 | 0.1 | 1 | 1 | 23.50 |
| 1 | 0.2 | 1 | 5 | 20.85 |
| 1 | 0.3 | 1 | 5 | 23.75 |
| Simple linear prediction | | | | |
| - | - | - | - | 32.45 |

A comparative prediction result using linear regression as a benchmark is given in the table above.

# 5 Discussion

By looking at our loss graphs we see, that our model is minimizing the loss as it is supposed to. Though, the test loss is still relatively high compared to the train loss (See Figure 6). This might be due to the fact, that we only verify our model on the last 20% of the data, where we see some high spikes in the reported positive Covid-19 cases compared to the rest of the data (See Figure 7). It is not trained on this type of data but rather the first 80% which shows a more conservative development.
Some type of cross validation would be preferred in this case. For time series it does not make sense to implement the

[2] Click: See line 190 through 298
[3] See: https://github.com/AndreasLF/02461_exam_project/tree/main/passenger-graphs

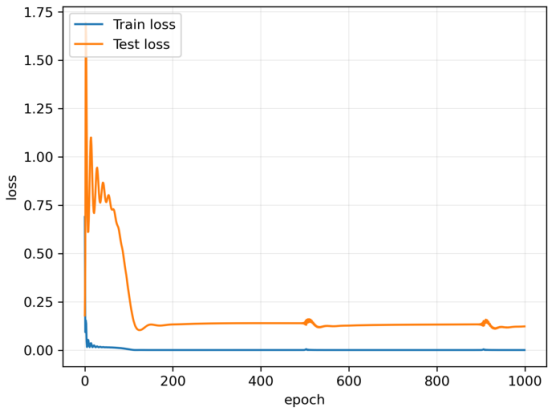



Figure 6: Train and test loss graph for a sequence length of 14 day, 5 neurons, 1 layer and a learning rate of 0.2. We see that the loss is minimized, but also that the test loss relative to the train loss is still high

widely used k-fold cross validation, but some type of "walk forward validation" might make our model more reliable[7]

We end up with a relatively high percentage wise deviation between the predicted cases and the real reported cases. As seen in the table, our deviation ranges between 20.02% and 40.28% [4] for the LSTM-model. And this is just for the nine best performing configurations of the network. At some configurations the LSTM model performs better than our simple linear prediction, which, on average, deviated 32.45% from the real recorded cases.
This shows that the model can be used, if configured properly, to make better predictions than the simple linear model. Although it it still very imprecise.

By looking closely at our predictions graph (See Figure 7), we see that the predicted data is almost the same as the day before, which means our deviation will be very high, if there is a big surge in cases from one day to the next.

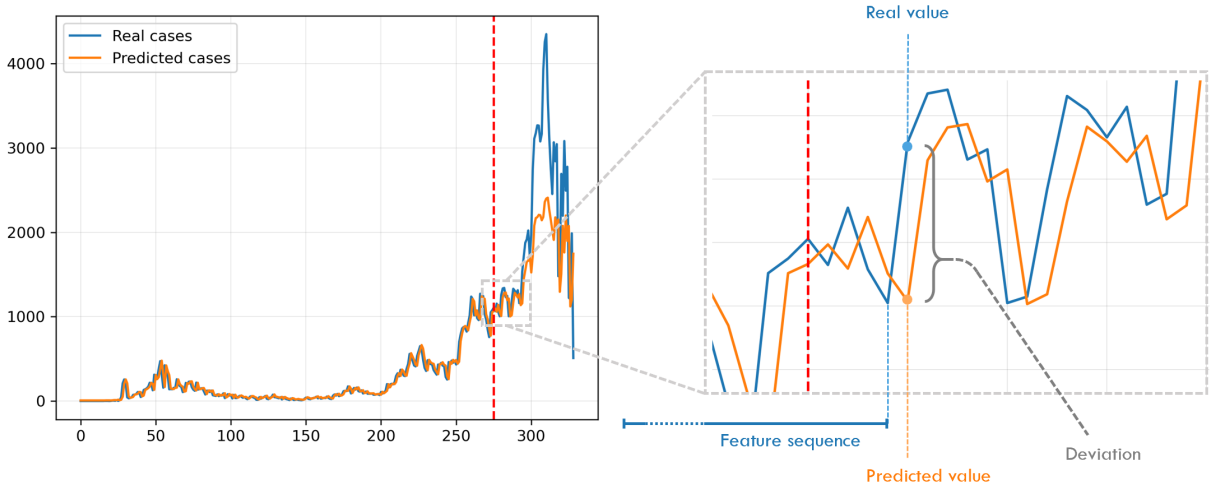



Figure 7: Prediction graph at a sequence length of 14, 5 neurons, 1 layer and a learning rate of 0.2. We see that it looks like the predicted day is displaced by one day, which is due to the fact that our model is too conservative in its prediction.

[4] See Results section

Our model is very conservative, which might be explained by the fact that the data we train on does not fluctuate as much as the data we test on.

With clarification of the difficulty of predicting time series data a relevant ethical quandary emerges; how can one believe any mathematical prediction of how Covid-19 spread will develop, and is it ethical to even suggest to "forecast" the Covid-19 spread using a neural network? Even though, our LSTM model is a simple one-to-one neural network taking only previous cases into account, one might speculate as to how accurate the myriad of mathematical models predicting Covid-19 actually are. Denmark, in its modern state, has never dealt with a situation of this sort before and there in an immense amount of uncertainty as to how the spread will play out. Especially humans, if considered as individuals, has the tendency to be somewhat irrational and unpredictable. With even more in play when adding variables such as weather and holidays, it seems unfathomable that one would suggest to have a reasonable prediction model. This begs the prior question; is it ethical to suggest to "forecast" the Covid-19 spread using a neural network or a complex mathematical model? While society condemns fake news and does all to combat it, one could argue that every time an institution suggests to have a prediction of the spread in the near future, they are contributing to the uncertain or flawed information in the news stream. Lately, the Danish institute SSI has withdrawn from the field of Covid-19 spread predictions, after trying to predict 14 days ahead and failing terribly. [8] This proves the point as to just how difficult and challenging it is to predict something as complex as a virus spread pattern, and at the essence of it, predicting human behaviour.

To sufficiently predict human behaviour one would have to collect data on said human, which raises yet another ethical dilemma. To what degree should or would we sacrifice our personal information and privacy in the attempt to predict and/or prevent Covid-19 spread? One factor to potentially substantially increase the relevance of a prediction, is the geolocational information on us. With data on where people fare, a model could get much more relevant. Although, people in general do not like the idea of being "spied on", which has already been seen when Denmark released the "Smittestop" app. Even though the app does not actively track and store ones location the idea itself had some people rejecting the suggestions to download it [9]. So, how much personal data will we give up? And how much data is sufficient to a model?

As stated our model is a very simplistic single input neural network, which could be an explanation for the poor result. An interesting addition to our model would be to add more parameters and turn it into a multivariate LSTM model. With more data one could hope to avoid the Persistent algorithm problem. With data from Danmarks Meterologiske Institut (DMI) or a simple calendar we could attempt to develop the model to the point where we could begin to see patterns emerging. One could imagine to see a pattern where more positive-cases will appear near a holiday such as New Years Eve or Christmas where people usually gather. Additionally, we could possibly see patterns where cases are lower when it is raining (we suspect that fewer people will get tested if the weather is bad) and higher when the sun is shining. These correlations are purely speculation since we have not conducted such experiment, but this serves as a reminder to the myriad of variables in play.

# 6 Learning outcome

In the span of this project we have gained instrumental experience within AI-based research and programming. Through continuous implementation and testing we have deepened our theoretical understanding of Neural Networks (especially LSTM's and RNN's) significantly. In continuation of this we have become acquainted with the workflow of an AI-engineer with regards to problem-solving, research and experimentation. More specifically we have gained hands-on experience with data exploration/processing as well as state-of-the-art data science libraries such as Pytorch, Numpy, scikit-learn, etc. Lastly we have learned how difficult it can be to combine unrelated/unprocessed data in advanced neural networks.

# References


[1] "WHO Coronavirus Disease (COVID-19) Dashboard ."

[2] M. Phi, "Illustrated Guide to LSTM's and GRU's: A step by step explanation," 9 2018.

[3] "sklearn.preprocessing.MinMaxScaler ."

[4] D. P. Kingma and J. Lei Ba, "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION," tech. rep., 2015.

[5] mwaskom, "Data repository for seaborn examples."

[6] "Data structures accepted by seaborn."

[7] J. Brownlee, "How To Backtest Machine Learning Models for Time Series Forecasting," 12 2016.

[8] Ritzau, "SSI dropper forudsigelser om coronasmitten i Danmark," 11 2020.

[9] Smittestop / Net Commpany, "Svar på spørgsmål omkring appen."

# 7 Appendix

## 7.1 Corona LSTM model

Our code is inspired by the following three examples:

https://git.io/Jt3zK
https://stackabuse.com/time-series-prediction-using-lstm-with-pytorch-in-python/
https://curiousily.com/posts/time-series-forecasting-with-lstm-for-daily-coronavirus-cases/

---

```
import torch
import torch.nn as nn

import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Import MinMaxScaler from sklearn
from sklearn.preprocessing import MinMaxScaler
# Import floor function from math module
from math import floor
from sklearn.metrics import accuracy_score

from extract_ssi_data import *


class LSTM(nn.Module):
    """LSTM time series prediction model
    Attributes:
        num_classes (int): Size of output sample for nn.Linear
        input_size (int): Number of features fed to the model
        hidden_size (int): Number of neurons in each layer
        num_layers (int): Number of layers in the network
        fc: Instance of the nn.Linear module
        lstm: Instance of the LSTM module
    """

    def __init__(self, seq_length, num_classes=1, input_size=1, hidden_size=1, num_layers=1):
        """ Initialize LSTM object
        Args:
            seq_length (int): Sequence length for the input
            num_classes (int): Size of output sample for nn.Linear
            input_size (int): Number of features fed to the model. Defaults to 1
            hidden_size (int): Number of neurons in each layer. Defaults to 1
            num_layers (int): Number of layers in the network. Defaults to 1
        """
        super(LSTM, self).__init__()

        # Set the class attributes
        self.num_classes = num_classes
        self.num_layers = num_layers
```

```
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.seq_length = seq_length

        # Define the lstm model
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, batch_first=True)

        # Define instance of nn.Linear
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        """ Propagate through the NN network layers
        Args:
        x (torch): is the input features
        """
        h_0 = torch.zeros(
            self.num_layers, x.size(0), self.hidden_size).to(device)

        c_0 = torch.zeros(
            self.num_layers, x.size(0), self.hidden_size).to(device)

        # Propagate input through LSTM
        ula, (h_out, _) = self.lstm(x, (h_0, c_0))

        h_out = h_out.view(-1, self.hidden_size)

        out = self.fc(h_out)

        return out

def create_sequences(data, seq_length):
    """ Create sequences from the data
    Params:
        data (list): The data you want to split in sequences
        seq_length (int): the length of the sequences
    Returns:
        A list of features and a list of targets
    """
    xs = []
    ys = []
    for i in range(len(data) - seq_length - 1):
        # Create a sequence of features
        x = data[i:(i + seq_length)]
        # Take the next value as a target
        y = data[i + seq_length]
        # Append to lists
        xs.append(x)
        ys.append(y)
    # Convert to ndarrays and return
    return np.array(xs), np.array(ys)


print("GPU Driver is installed: "+str(torch.cuda.is_available()))

device = torch.device('cpu')

RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)
```

```
    ### Variables ###

# If true the test results and graphs will be printed to a folder.
# If False plots will be opened in a new window
print_tests_to_folder = False

# If true it will loop through a range of hidden_size and learning_rate
loop_through_tests = False

# Test folder name. The folder we want to print the tests to
test_folder = "test"

# Percentage of test size
test_size_pct = 0.20

# Sequence length
seq_length = 14

# Number of iterations
num_epochs = 100

# Print each 10th epoch value
epoch_print_interval = 100

# Learning rate and hidde size, will only be used if loop_through_tests is False
learning_rate = 0.4
hidden_size = 6

# Other variables
input_size = 1
num_layers = 1
num_classes = 1


if loop_through_tests:
    # Testing range for leaning rate
    learning_rate_range = np.arange(0.1, 0.6, 0.1)
    # Testing range for hidden size
    hidden_size_range = np.arange(1,7,1)
else:
    learning_rate_range = [learning_rate]
    hidden_size_range = [hidden_size]


# Initialize counter
i = 1




# Load flight data from seaborn library
df = extract_ssi_data()
# Convert monthly passengers to float
df = df.values.astype(float)

# Define a scaler to normalize the data
scaler = MinMaxScaler(feature_range=(-1, 1))
# Scale data. Data is fit in the range [-1,1]
data_normalized = scaler.fit_transform(df.reshape(-1, 1))

# Create feature sequences and targets
```

```
x, y = create_sequences(data_normalized, seq_length)

# Split data in training and test
train_size = int(floor(len(df)*(1-test_size_pct)))

# Convert all feature sequences and targets to tensors
x_data = torch.Tensor(np.array(x)).to(device)
y_data = torch.Tensor(np.array(y)).to(device)

# Split train and test data and convert to tensors
x_train = torch.Tensor(np.array(x[0:train_size])).to(device)
y_train = torch.Tensor(np.array(y[0:train_size])).to(device)

x_test = torch.Tensor(np.array(x[train_size:len(x)])).to(device)
y_test = torch.Tensor(np.array(y[train_size:len(y)])).to(device)


# Loop over our desired ranges
for learning_rate in learning_rate_range:
    for hidden_size in hidden_size_range:
        # Create LSTM object
        lstm = LSTM(seq_length, num_classes, input_size, hidden_size, num_layers)
        # Mean squared error loss function defined
        loss_fn = torch.nn.MSELoss()
        # Adam optimizer is used
        optimizer = torch.optim.Adam(lstm.parameters(), lr=learning_rate)


        nn_log = {"train_loss": [], "test_loss": []}


        # Train the model
        for epoch in range(num_epochs):
            y_pred = lstm(x_train)
            optimizer.zero_grad()

            # Get loss function
            loss = loss_fn(y_pred, y_train)

            # Make prediciton
            y_test_pred = lstm(x_test)
            # Get loss function
            test_loss = loss_fn(y_test_pred, y_test)

            # test_accuracy = np.mean(y_test.detach().numpy() == p_test)

            # Backward propagate
            loss.backward()

            optimizer.step()

            # Save loss
            nn_log["train_loss"].append(loss.item())
            nn_log["test_loss"].append(test_loss.item())
            # Save accuracy
            # nn_log["accuracy"].append(test_accuracy*100)

            # Print loss
            if epoch % 10 == 0:
                print("Epoch: %d, Train loss: %1.5f, Test loss: %1.5f" % (epoch, loss.item(), test_loss.item()))
```

```
# Plot loss by epochs
plt.plot(nn_log["train_loss"])
plt.plot(nn_log['test_loss'])
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(["Train loss", "Test loss"], loc='upper left')
plt.grid(color='gray', linestyle='-', linewidth=0.1)

if print_tests_to_folder:
    plt.savefig("{}/{}_loss.png".format(test_folder, i))
    plt.clf()
else:
    plt.show()



# Plot train and predict data
train_predict = lstm(x_data)
data_predict = train_predict.data.numpy()
dataY_plot = y_data.data.numpy()

data_predict = scaler.inverse_transform(data_predict)
dataY_plot = scaler.inverse_transform(dataY_plot)

plt.axvline(x=train_size, c='r', linestyle='--')

plt.plot(dataY_plot)
plt.plot(data_predict)
plt.suptitle('Time-Series Prediction')
plt.grid(color='gray', linestyle='-', linewidth=0.1)


if print_tests_to_folder:
    plt.savefig("{}/{}_pred.png".format(test_folder, i))
    plt.clf()
else:
    plt.show()


# Calculate mean accuracy

y_pred = scaler.inverse_transform(lstm(x_test).detach().numpy())
y_real = scaler.inverse_transform(y_test.detach().numpy())
test_accuracy = np.mean((abs(y_real - y_pred) / y_real) * 100)
print(f'Accuracy: {test_accuracy}')

if print_tests_to_folder:
    file_object = open('{}/tests.txt'.format(test_folder), 'a')
    file_object.write("Test {}".format(i))
    file_object.write("\n Test size: {}".format(test_size_pct))
    file_object.write("\n Epochs: {}".format(num_epochs))
    file_object.write("\n Fineal prediction accuracy: {}".format(test_accuracy))
    file_object.write("\n Hidden size: {}".format(hidden_size))
    file_object.write("\n Num layers: {}".format(num_layers))
    file_object.write("\n Learning rate: {}".format(learning_rate))
    file_object.write("\n Num classes: {}".format(num_classes))
    file_object.write("\n Input size: {}".format(input_size))
    file_object.write("\n Loss: ")

    for ep in np.arange(0, num_epochs, epoch_print_interval):
```

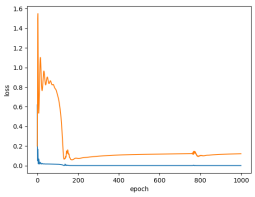
```
            file_object.write("\n Epoch: {}, Train loss: {}, Test loss: {}".format(ep,
                nn_log["train_loss"][ep], nn_log["test_loss"][ep]))
        file_object.write("\n \n")
        file_object.close()

    i += 1
```

## 7.2 Extract data code

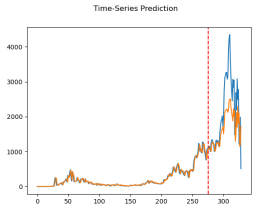
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pylab import rcParams
from matplotlib import rc
from pandas.plotting import register_matplotlib_converters
import numpy as np


def extract_ssi_data():
    """ Extract new daily positive cases

    Returns:
        (pandas.DataFrame) daily positive covid cases from SSI
     """
    test_pos_over_time_file = "data/Data-Epidemiologiske-Rapport-04012021-ja21/Test_pos_over_time.csv"

    # Read data
    test_pos_over_time = pd.read_csv(test_pos_over_time_file, sep=";", decimal=",")

    # Remove last two rows
    df = test_pos_over_time["NewPositive"].iloc[:-2]

    # Set index to datetime
    df.index = pd.to_datetime(test_pos_over_time["Date"].iloc[:-2])

    # Remove . from data
    df = df.str.replace(".","").astype(int)

    return df

def data_exploration(df):
    """ Plot data
    Args:
        df (pandas.DataFrame): Daily positive covid cases
    """
    df.plot()
    plt.title("Daily covid cases");
    plt.show()

def extract_dmi_data(stat):
    """ Extract data from dmi csv-files
    Args:
        stat (str): Statistics to load [FUGTIG, LUFTRYK, NEDBR, SOLSKIN, TEMP, VIND]
    Returns:
        df (pd.DataFrame) containing statistics from dmi 2020
    """
    stat = f"DMI-{stat.upper()}"
```

```
    months = ["januar", "februar", "marts", "april", "maj", "juni", "juli", "august", "september", "oktober",
        "november", "december"]

    first_month_file = f"data/{stat}/hele-landet-{months[0]}-2020.csv"
    df = pd.read_csv(first_month_file, sep=";")
    for month in months[1:]:
        dmi_file = f"data/{stat}/hele-landet-{month}-2020.csv"
        dmi_data = pd.read_csv(dmi_file, sep=";")
        df = df.append(dmi_data)

    df.index = pd.to_datetime(df["DateTime"])
    df = df.drop(["DateTime"],axis=1)
    return df
# data_exploration(extract_ssi_data())

def get_data():
    df = pd.concat([extract_dmi_data("fugtig"), extract_dmi_data("temp")], axis=1)
    df.insert(0, "NewPositive",extract_ssi_data())
    df = df.dropna()
    return df
```

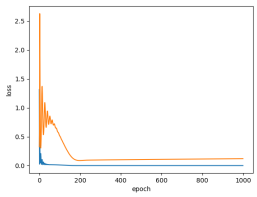
## 7.3 Graphs



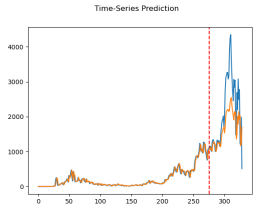
(a) **Loss graph** sequence length: 14, neurons: 4, layers: 1, learning rate: 0.2



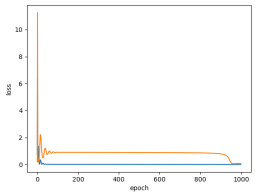

(b) **Prediction graph** sequence length: 14, neurons: 4, layers: 1, learning rate: 0.2



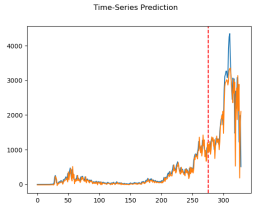
(c) **Loss graph** sequence length: 14, neurons: 5, layers: 1, learning rate: 0.2



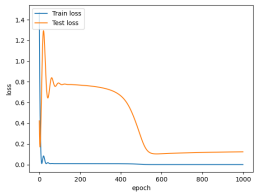

(d) **Prediction graph** sequence length: 14, neurons: 5, layers: 1, learning rate: 0.2



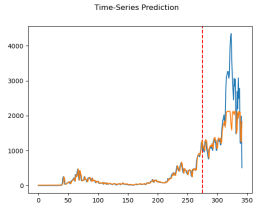
(e) **Loss graph** sequence length: 14, neurons: 4, layers: 1, learning rate: 0.5



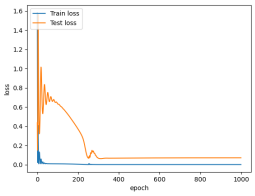

(f) **Prediction graph** sequence length: 14, neurons: 4, layers: 1, learning rate: 0.5



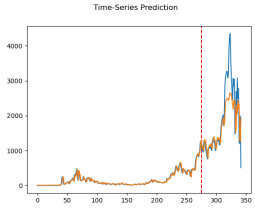

(g) **Loss graph** sequence length: 1, neurons: 1, layers: 1, learning rate: 0.1



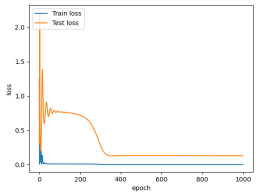

(h) **Prediction graph** sequence length: 1, neurons: 1, layers: 1, learning rate: 0.1



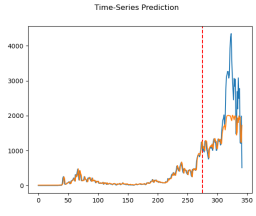

(i) **Loss graph** sequence length: 1, neurons: 5, layers: 1, learning rate: 0.2



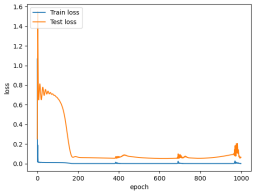

(j) **Prediction graph** sequence length: 1, neurons: 5, layers: 1, learning rate: 0.2



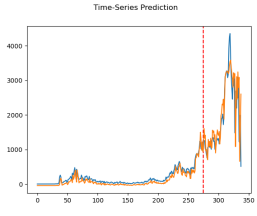

(k) **Loss graph** sequence length: 1, neurons: 5, layers: 1, learning rate: 0.3



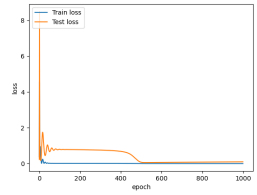

(l) **Prediction graph** sequence length: 1, neurons: 5, layers: 1, learning rate: 0.3



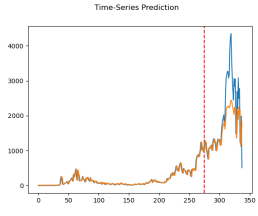

(m) **Loss graph** sequence length: 5, neurons: 3, layers: 1, learning rate: 0.4



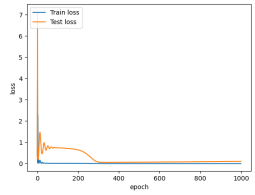

(n) **Prediction graph** sequence length: 5, neurons: 3, layers: 1, learning rate: 0.4



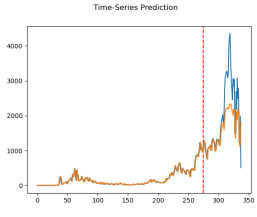

(o) **Loss graph** sequence length: 5, neurons: 5, layers: 1, learning rate: 0.4




(p) **Prediction graph** sequence length: 5, neurons: 5, layers: 1, learning rate: 0.4




(q) **Loss graph** sequence length: 5, neurons: 5, layers: 1, learning rate: 0.5




(r) **Prediction graph** sequence length: 5, neurons: 5, layers: 1, learning rate: 0.5