2020

Smart Distribution Systems

Introduction:

In recent years, predicting the prices of electricity has become a vital part of energy producers as it has direct input on decision making. In this lab we were provided with Belpex price data using which we implemented a neural network that is able to predict the future values based on the training of model. The load factors of wind and solar energy are also utilized to reduce MSE (mean square error). Lags from previous prices were also used to reduce MSE.

Neural network model characteristics:

As per the guidelines provided in exercise session, we implemented neural network with following characteristics:

- **Training data:** 70%
- **Validation data:** As provided in files
- **Number of neurons/layer:** 100
- **Number of hidden layers:** 6
- **Method used:** Repeated K-fold method
- **Epochs:** 2500
- **Batch size:** 32 for training model, 24 for validation
- **Over-fitting of data prevention:** Early-stopping was used
- **Total Features used:** Belpex data, load factors of both solar and wind, Belpex lags

Training data:

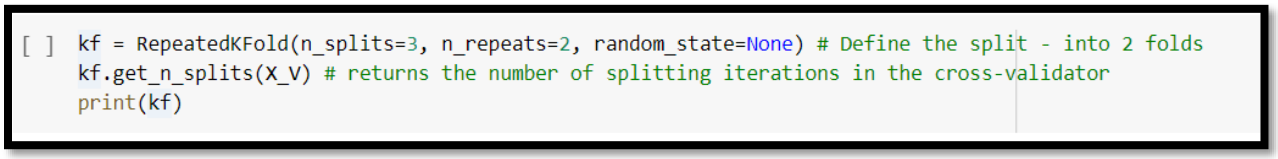
As guided during lab session to use 70% of total dataset for training. In case of our model, neural network was trained using data from 6th Feb, 2016 to 2nd May, 2017.

Validation data:

As guided during lab session to use 30% of total dataset for validation. In case of our model, neural network was validated using data from 19th May, 2017 to 22nd Dec, 2017.

K-fold method description:

We have used k-fold method as it produced better and reduced MSE (mean squared error). This is cross validation method. The data sets are divided into **k subsets**. For every iteration, k subsets are used as test set while **k-1 subset** are combined to form training data set.



```
[ ]  kf = RepeatedKFold(n_splits=3, n_repeats=2, random_state=None) # Define the split - into 2 folds
     kf.get_n_splits(X_V) # returns the number of splitting iterations in the cross-validator
     print(kf)
```

We implemented repeated k-fold method with 3 splits repeating procedure twice.

Early stopping:
It is better to implement a neural network that is neither under-fitting nor over-fitting. In order to avoid the over-fitting, we used a function called “Early stopping”. This helps to stop training when the chosen performance measure stops improving.

Number of Neurons selection:
After extensive hit and trial method and research on internet a formula was found to select optimal number of neurons:

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

N_i = number of input neurons.
N_o = number of output neurons.
N_s = number of samples in training data set.
α = an arbitrary scaling factor usually 2-10.

https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw

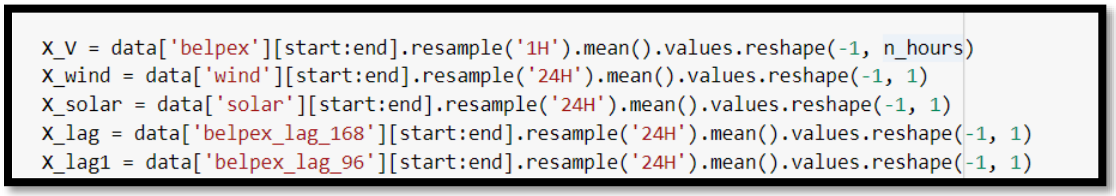
Input and output neurons were 28 and 24 respectively in our case. After putting all the values in above formula, we found the upper limit of neurons. So the number of neurons were kept below that limit i.e. 100.

Hidden layers:
Different numbers were tried and tested for number of hidden layers. As there is no particular formula/ algorithm to determine this number. We used **6** hidden layers in our case.

Epochs:
According to definition in Keras documentation, epochs is defined as number of iterations over whole data set. We tried different numbers 2500, 3000, 5000. The time it took for 5000 epochs were significantly larger but the results were almost same as 2500 epochs.

Batch size:
The batch size was set to 32 for training and 24 for validation.

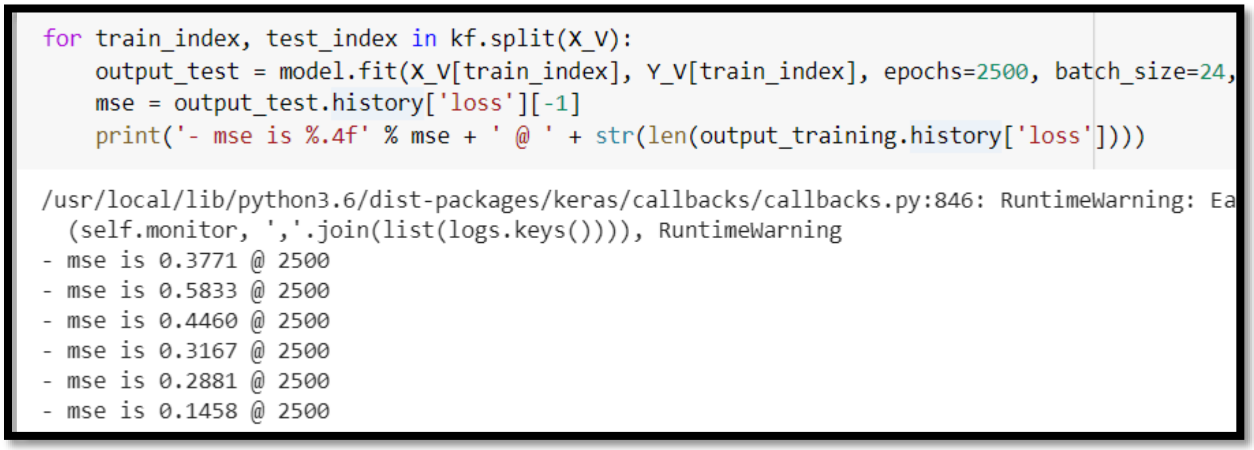
Decreasing Mean Squared error
In order to decrease mean squared error, we used load factor of wind and solar. We also used lag prices of Belpex.



```
X_V = data['belpex'][start:end].resample('1H').mean().values.reshape(-1, n_hours)
X_wind = data['wind'][start:end].resample('24H').mean().values.reshape(-1, 1)
X_solar = data['solar'][start:end].resample('24H').mean().values.reshape(-1, 1)
X_lag = data['belpex_lag_168'][start:end].resample('24H').mean().values.reshape(-1, 1)
X_lag1 = data['belpex_lag_96'][start:end].resample('24H').mean().values.reshape(-1, 1)
```

Training Mean Squared error

Using epochs as 2500 with batch size = 24, we arrived achieved low MSE in training model. As shown in figure below:



```
for train_index, test_index in kf.split(X_V):
    output_test = model.fit(X_V[train_index], Y_V[train_index], epochs=2500, batch_size=24,
    mse = output_test.history['loss'][-1]
    print('- mse is %.4f' % mse + ' @ ' + str(len(output_training.history['loss'])))
```

```
/usr/local/lib/python3.6/dist-packages/keras/callbacks/callbacks.py:846: RuntimeWarning: Ea
  (self.monitor, ','.join(list(logs.keys()))), RuntimeWarning
- mse is 0.3771 @ 2500
- mse is 0.5833 @ 2500
- mse is 0.4460 @ 2500
- mse is 0.3167 @ 2500
- mse is 0.2881 @ 2500
- mse is 0.1458 @ 2500
```

Prediction Results:

Predictions for different weeks were generated and combined in a single csv file. Following are the results:

Week 1:

After training of model, the data provided to us for prediction was imported and it was predicted as follows.

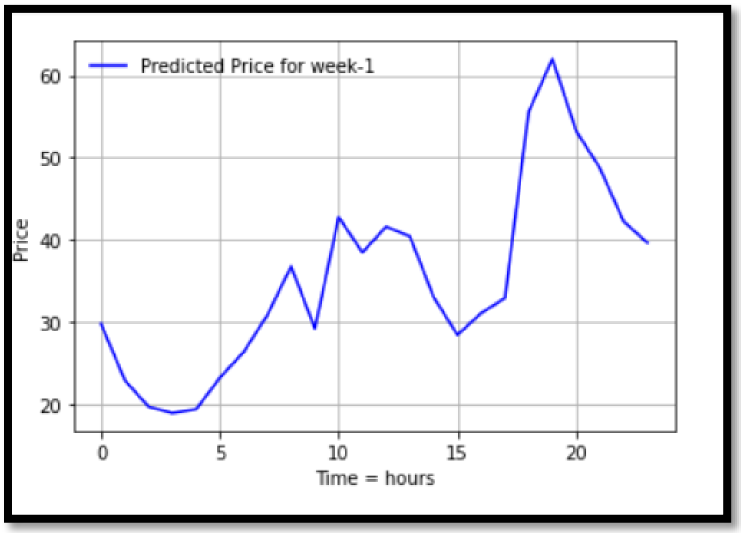


Fig 1: Price predictions for week-1

Week 2:

Similarly after training of model, the data provided to us for prediction was imported and it was predicted as follows.

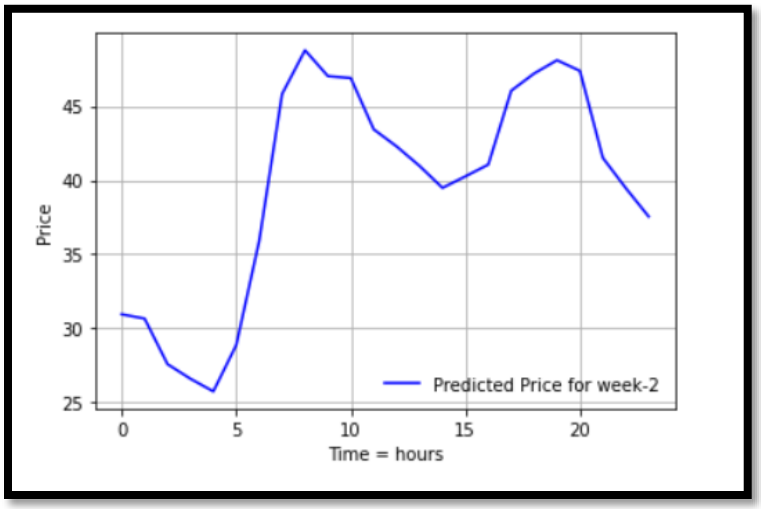


Fig 2: Price predictions for week-2

Week 3:

Predictions for week 3 are as follows:

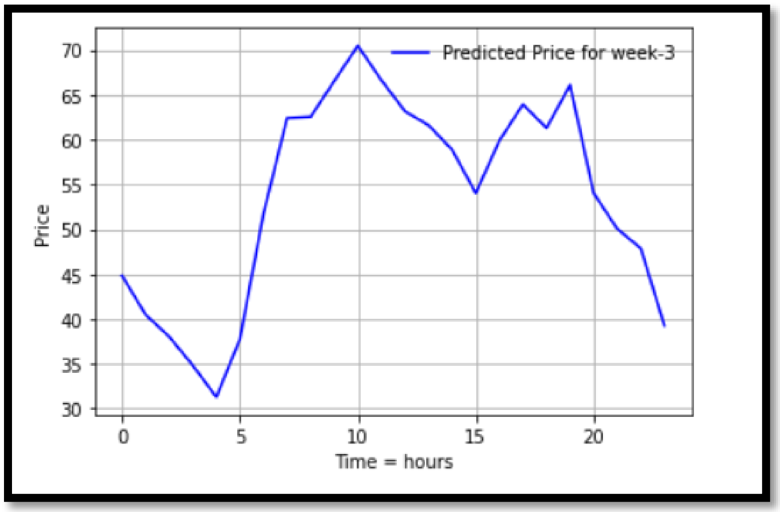


Fig 3: Price predictions for week-3

<u>Conclusion:</u>

This labs served as a brief introduction of neural networks and its implementation in smart distribution systems. We gained insights in machine learning concepts, neural networks, mean squared error, optimizers etc. We predicted future prices using old data of Belpex and solar and wind load factors. We learned to set number of neurons for improved accuracy as well as how to save our predictions.