Energy Data Super Resolution Report

xc720@ic.ac.uk

ABSTRUCT:

Nowadays, high-resolution data is becoming more and more demanding, however, the resolution of available data is lower than expected. Thus, proper algorithms are needed to up sample the lower-resolution data to high-resolution data. In this report, inspired by the essay, *Poster Abstract: Super-Resolution Reconstruction of Interval Energy Data. Jieyi Lu and Baihong Jin. 2020,* I proposed a data super-resolution algorithm based on a LSTM time-series-data prediction model and evaluated it in the real-world energy data set.

1 INTRODUCTION:

The target of energy data super-resolution is to estimate a high-resolution energy time-series data for a low-resolution energy time-series data. For example, given 5-minute data, in which each 5-minute data is the average of the corresponding 1-minute data, we will try to predict the 1-minute data by our data super-resolution model.

Just like image super resolution, we normally assume neighbouring pixels in an image to have similar values. I think that there may exist some similarities in the training set to values that we need to predict. Thus the original model is inspired by a time-series-data prediction model.

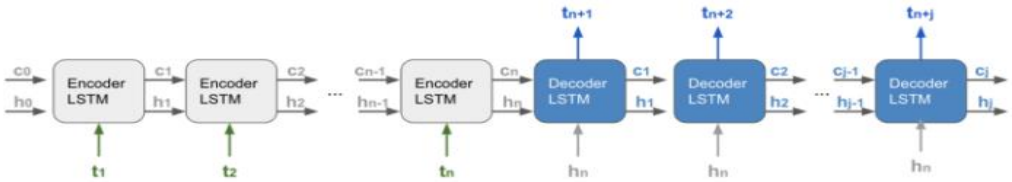
All the work carried out are based on the fact that the researched data are from only one house and energy consumption on weekdays are similar, which will probably results inaccuracy if our model is used in other houses.

2 METHOLOGY:

2.1 Data and Model Selection

Gained by Advanced Metering Infrastructure, energy time-series data (from Pecan Street Project) are initially given in the form of 1 minute data and kWh. I found that energy consumption can be impacted by a lot of factors, such as house ID, hour and day etc. So I only used the 1 minute data from one house to train my model.

However, the amount of 1-minute data of one house is still quite huge, which is about 260,000. Consequently, it will cause long-term dependencies problem as some desired output depend on inputs presented at times far in the past. Thus a LSTM time-series-data prediction model is applied, which can remember useful information for a long period of time. As the inputs and outputs have different sizes and categories, a Seq-to-Seq architecture is used, so both input and output data will be in sequence.




```python
class Seq2Seq(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size, batch_size):
        super().__init__()
        self.output_size = output_size
        self.Encoder = Encoder(input_size, hidden_size, num_layers, batch_size)
        self.Decoder = Decoder(input_size, hidden_size, num_layers, output_size, batch_size)

    def forward(self, input_seq):
        batch_size, seq_len, _ = input_seq.shape[0], input_seq.shape[1], input_seq.shape[2]
        h, c = self.Encoder(input_seq)
        outputs = torch.zeros(batch_size, seq_len, self.output_size).to(device)
        for t in range(seq_len):
            _input = input_seq[:, t, :]
            output, h, c = self.Decoder(_input, h, c)
            outputs[:, t, :] = output

        return outputs[:, -1, :]
```

Seq2Seq LSTM Model Figure and Code

2.2 Model Working Process

2.2.1 Data Process

First, 1-minute data of one house will be loaded by Pandas. And 80% of it will be divided for train dataset, 10% for validation dataset and 10% for test dataset. 1-minute data serves as labels for the model.

If we want to upsample 3-minute data to 1-minute data, each 3-minute data will be generated by taking average of 3 1-minute data. The code to achieve this is shown below.

```python
def nn_seq_mo(seq_len, B, num):
    data = load_data('1min__data_newyork_sum_1240.csv')

    train = data[:int(len(data) * 0.8)]
    val = data[int(len(data) * 0.8):int(len(data) * 0.9)]
    test = data[int(len(data) * 0.9):len(data)]
```

```python
for i in tqdm(range(0, len(dataset) - seq_len - num, step_size)):
    train_seq = []
    train_label = []
    avg = 0
    for j in range(i, i + num):
        avg += load[j]
        if (j-i+1) % 3==0:
            train_seq.append([avg/3])
            avg = 0

    for j in range(i, i + num):
        train_label.append(load[j])

    train_seq = torch.FloatTensor(train_seq)
    train_label = torch.FloatTensor(train_label).view(-1)
    seq.append((train_seq, train_label))
```

Then we can assign the 3-minute data to variable train_seq to be sequences of input data. While for 1-minute data to variable train_label as it's the desired form of data for

the output. And both will be fed into variable seq to form a new dataset which can be read by torch.

2.2.2 Training

During the training process, the model will keep predicting 1-minute data based on sequences of input 3-minute data and calculating the loss using the results and label, which is also known as the ground truth. The core code of training is shown below.

```
for epoch in tqdm(range(args.epochs)):
    train_loss = []
    for (seq, label) in Dtr:
        seq = seq.to(device)
        label = label.to(device)
        y_pred = model(seq)
        loss = loss_function(y_pred, label)
        train_loss.append(loss.item())
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

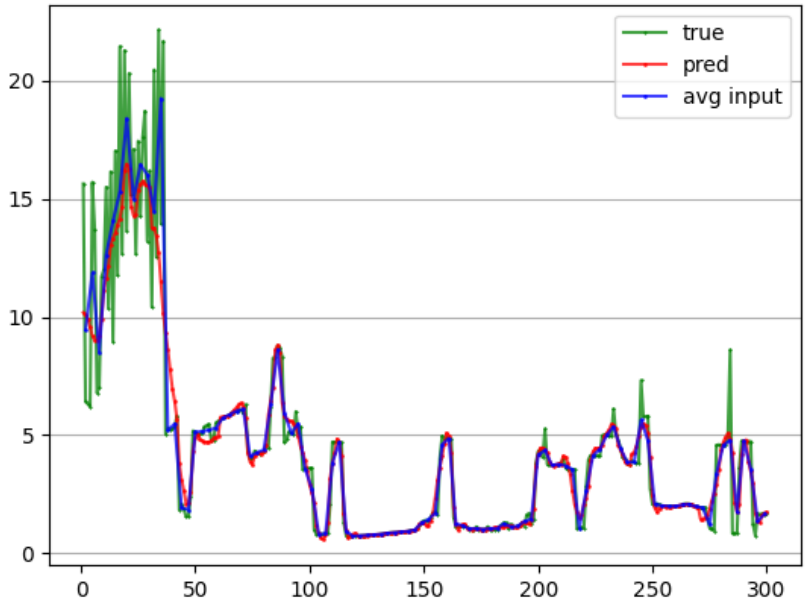
Meanwhile, as epochs iterating, we can see the value of loss decreases to very small values and stabilizes at the value of 0.0035 or so. This indicates that the model gradually learned the pattern of desired prediction. In my training process, the model went through 70 training epochs. It's unnecessary for model to train more epochs if the value of loss is stabilized at certain value.

```
100%|██████████| 211928/211928 [00:04<00:00, 51676.84it/s]
100%|██████████| 26456/26456 [00:00<00:00, 59248.88it/s]
100%|██████████| 882/882 [00:00<00:00, 58801.46it/s]
finish
  1%|          | 1/70 [00:47<54:41, 47.56s/it]epoch 000 train_loss 0.01467881 val_loss 0.00045237
epoch 001 train_loss 0.01178495 val_loss 0.00043747
  4%|▍         | 3/70 [02:25<54:24, 48.73s/it]epoch 002 train_loss 0.01089758 val_loss 0.00048481
  6%|▌         | 4/70 [03:14<53:49, 48.93s/it]epoch 003 train_loss 0.01078353 val_loss 0.00041921
epoch 004 train_loss 0.01068947 val_loss 0.00041559
  9%|▊         | 6/70 [04:56<53:36, 50.25s/it]epoch 005 train_loss 0.01066885 val_loss 0.00044654
epoch 006 train_loss 0.01060016 val_loss 0.00040823
 11%|█         | 8/70 [06:43<53:27, 51.74s/it]epoch 007 train_loss 0.01058095 val_loss 0.00038234
 13%|█▎        | 9/70 [07:33<51:56, 51.09s/it]epoch 008 train_loss 0.00987499 val_loss 0.00037148
epoch 009 train_loss 0.00996569 val_loss 0.00039162
 16%|█▌        | 11/70 [09:13<49:46, 50.62s/it]epoch 010 train_loss 0.00904325 val_loss 0.00036661
 17%|█▋        | 12/70 [10:01<48:12, 49.87s/it]epoch 011 train_loss 0.00899075 val_loss 0.00036380
 19%|█▊        | 13/70 [10:51<47:23, 49.89s/it]epoch 012 train_loss 0.00896539 val_loss 0.00035983
epoch 013 train_loss 0.00895467 val_loss 0.00035493
 21%|██        | 15/70 [12:26<44:36, 48.67s/it]epoch 014 train_loss 0.00894645 val_loss 0.00035514
epoch 015 train_loss 0.00894267 val_loss 0.00035478
 24%|██▍       | 17/70 [14:05<43:31, 49.27s/it]epoch 016 train_loss 0.00894305 val_loss 0.00035530
```

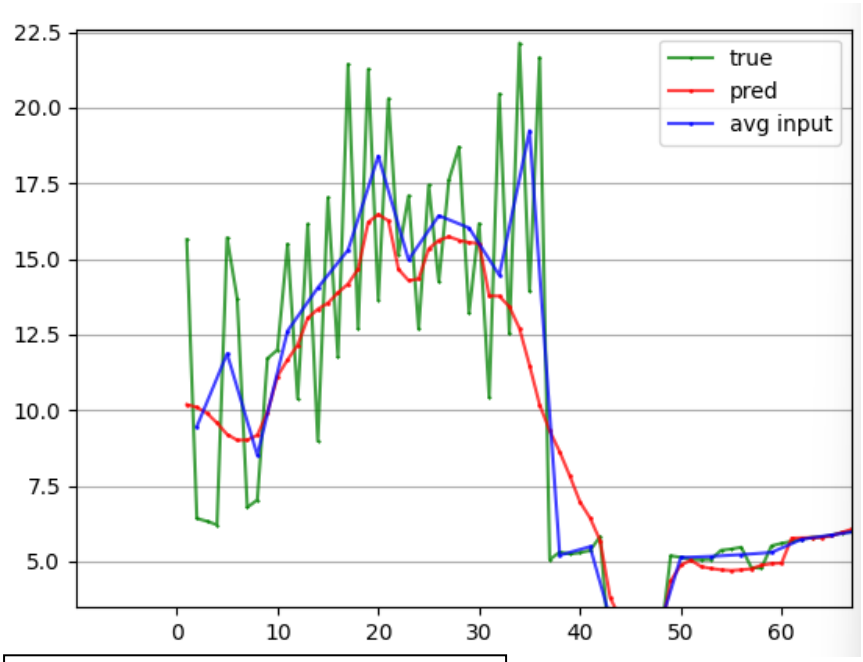
2.2.3 Testing

When it comes to testing, as mentioned before, 10% of 1-minute data will serve as true values and generate 3-minute data for testing as well. The trained model is input with the generated 3-minute data and will predict how this 3-minute data consists of 1-minute data.

At last, the output 1-minute data will be fed into get_mape function to calculate the MAPE (mean absolute percentage error) of it and the true values. The results are shown below with MAPE = 0.15008.



General Graph of Result



True---1-minute data, ground truth

Avg input---3-minute data, which is fed into the model as the input

Pred--- predicted 1-minute data, the result

If we zoom in the graph, we will find there exit a lot of peaks for true values which are very hard to predict. This is because our predicted values based on the averaged input

and our model didn't fully learn the feature of instability of the real-world 1-minute data when it was training. So I believe if more related features, such as time, day and month, are added at the data process step, the model could be improved.

3 CONCLUSIONS

In my model I introduced my solution to achieve data super-resolution. As shown above, 3-minute data are reconstructed to be 1-minute data. And we can also reconstruct low-resolution data to higher-resolution but with higher MAPE (mean absolute percentage error). I believe that this model can be further improved in the future and it's worthwhile exploring potential better-performing models, namely, data super-resolution model with transformer.

REFERENCES

[1] https://blog.csdn.net/Cyril_KI/article/details/126450247 Open source Code

[2] Smith, C. The Pecan Street Project: developing the electric utility system of the future.

[3] https://levelup.gitconnected.com/building-seq2seq-lstm-with-luong-attention-in-keras-for-time-series-forecasting-1ee00958decb

[4] Poster Abstract: Super-Resolution Reconstruction of Interval Energy Data. Jieyi Lu and Baihong Jin. 2020,

[5] Rithwik Kukunuri, Nipun Batra, and Hongning Wang. 2020. An Open Problem: Energy Data Super-Resolution. In The 5th International Workshop on Non-Intrusive Load Monitoring (NILM'20), November 18, 2020, Virtual Event, Japan