STATS M231A Homework 2

Laxman Dahal

Abstract

This homework aims at giving practical background on multiple advanced ML/AI algorithms. The first part of the homework gives an opportunity to implement ResNet with different number of layers, namely Resnet18, Resnet34, and Resnet50. The results from these three models indicated that even though the deeper model (Resnet50) has 23.52M parameters to learn, the shallower model such as Resnet18 with half the learnable parameters (11.17M), the Resnet50 models only does 0.5% better than the Resnet18 model. The second part of the homework implements RNN/LSTM/GRU models to predict the trends of energy consumption. It was found that the three models were successful in capturing and predicting the energy consumption trends.

1 Problem 1- Resnet

One of the problems in the deep networks is that they are hard to train because of the gradient vanishing problem. In the deep networks, the gradient is back-propagated to the earlier layers with reapeated multiplication which makes the gradient infinitively small. To counter this, identity mappings are stacked upon the current network which ensures that the deeper models would perform better than the shallower networks. Figure 1 shows one residual block in a multi-layer network. Mathematically, it can also be represented as

$$x_{l+1} = x_l + F(x_l)$$

where $l+1$ and l represent two different layers. As illustrated by the equation and figure below, $F(x_l)$ models the residual of the mapping from x_l to x_{l+1}, on top of the identity mappings.

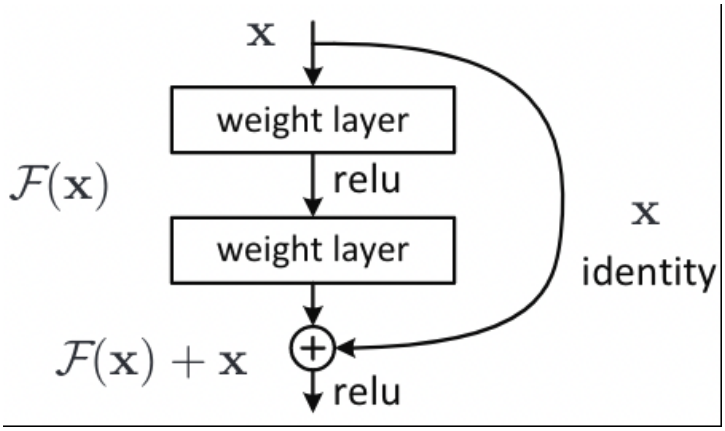


Figure 1: A residual block.

1.1 Equations

The underlying equation of the resnet is shown above where the input to the residual block (x_l) is transformed using feed forward network which is then used to update the input to produce the output (x_{l+1}). Each residual undergoes several transformations, such as batch normalization, ReLU. The equations for those transformations and networks are summarized below:

1. Convolution: $z^l = h^{l-1} * W^l$
2. MaxPooling $h^l_{xy} = max_{i=0,\ldots,s,j=0,\ldots,s} h^{l-1}_{(x+i)(y+j)}$
3. Fully connected layer: $z^l = W^l h^{l-1}$
4. ReLU: $ReLU(z_i) = max(0, z_i)$
5. Sigmoid: $\sigma(z_i) = \frac{1}{1+e^{-z_i}}$
6. Softmax: $softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$
7. Batch Normalization: $\text{BN}(z_i) = \gamma_i \hat{z}_i + \beta_i$, where $\hat{z}_i = \frac{z_i - E[z_i]}{\sqrt{Var[z_i]}}$

To determine the output size, the following equation is used:

$$N_{out} = 1 + \frac{N_{in} + 2 * p - k}{s}$$

where,
N_{out} = number of output features
N_{in} = number of input features
k = convolution kernel size
p = convolution padding size
s = convolution stride size

1.2 Network Structure

In this section, network architecture for the resnet is provided. First, a generic ResNet structure with 50 layers are presented in a graphical form. Figure 2 graphically illustrates how an individual residual block/unit is a part of a feed-forward network, and how that is a part of the entire network.

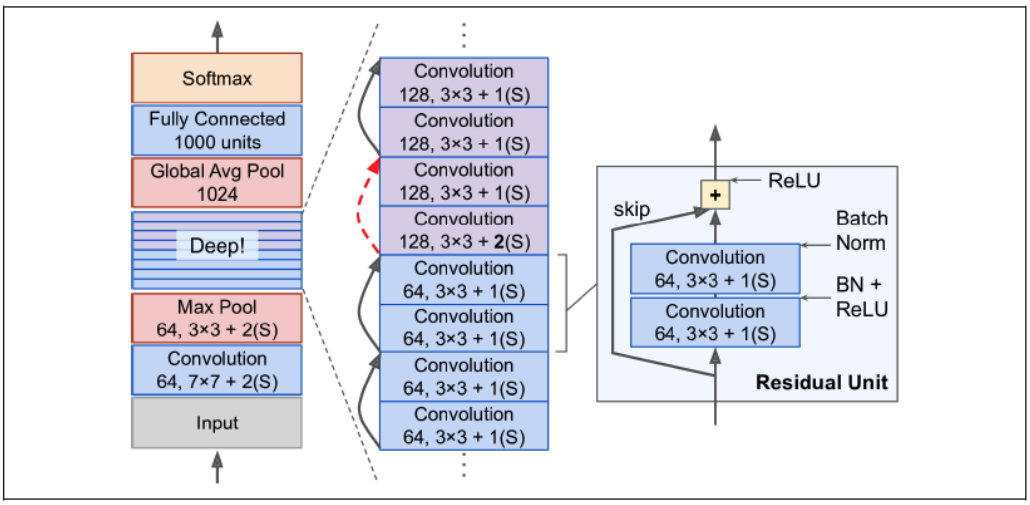


Figure 2: A graphical representation of ResNet.

Specifically for the code that was given to us, a part of the structure for the ResNet50 looks like Figure 3. As shown in the Figure 3, bottleneck represents one layer which is repeated multiple times depending on which layer it is at.

```
resnet50.eval()
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
```

Figure 3: A part of the resnet50 model.

1.3 Accuracy Plot

Figure 4 shows the accuracy from the three models obtained by running the code on google colab. Please refer to the attached code for verification. The results are consistent with what is reported on the github page.

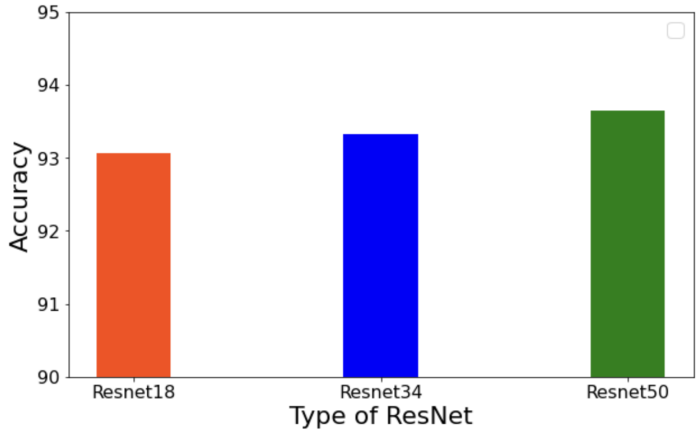


Figure 4: Accuracy of the three variations of ResNet

2 Problem 2- RNN/GRU/LSTM

2.1 Gated Recurrent Unit (GRU)

GRU is a type of recurrent neural network (RNN) that can effectively retain long-term dependencies in sequential data. Even though it uses a recurrent architecture, it is popular because of its superior speed while achieving the same accuracy as the other non-simplified models. The GRU is like a long

short-term memory (LSTM) with a forget gate, but has fewer parameters than the LSTM. In practice, GRU is shown to exhibit better performance on certain smaller and less frequent datasets. Figure 5 shows the architecture and the key equations involved in the GRU. In the figure, z_t represents update gate vector, x_t represents input vector, h_t represents output vector, $\hat{h}_t$ represents candidate activation vector, r_t represents reset gate vector and W, U, and b are the parameter matrices and vector.

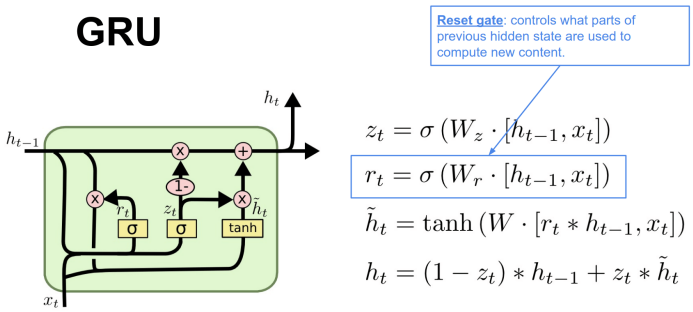


Figure 5: Architecture of the GRU and the key equations.

2.2 Long Short-Term Memory(LSTM

Different from the standard feedforward structure, the LSTM has feedback connections which can process not only single data points (such as images), but also the entire sequences of data (e.g., video, speech). LSTM is similar to RNN but has an additional gate mechanism. The gates ensure that the long-term dependencies in sequential data is preserved. In general, there are three gates in LSTM: input gate, forget gate, and output gate. The input gate determines which new information will be stored in the long-term memory. In other words, it filters out the information. The forget gate decides which information from the long-term memory should be kept os discarded. This is done by multiplying the incoming long-term memory by a forget vector generated by the current and incoming short-term memory. The output gate will take the current input, the previous short-term memory, and the newly computed long-term memory to produce the new short-term memory. Figure 6 illustrates the architecture and the main equations involved in the three different gates.

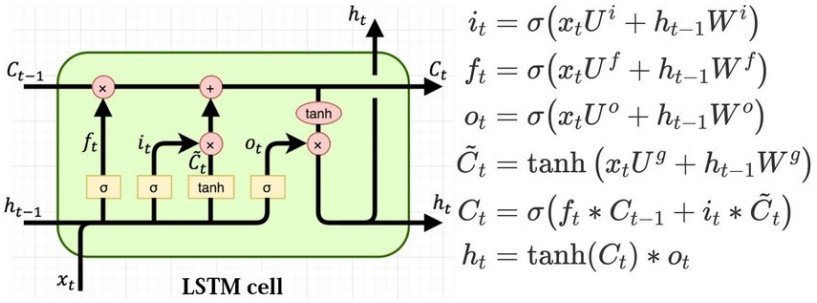
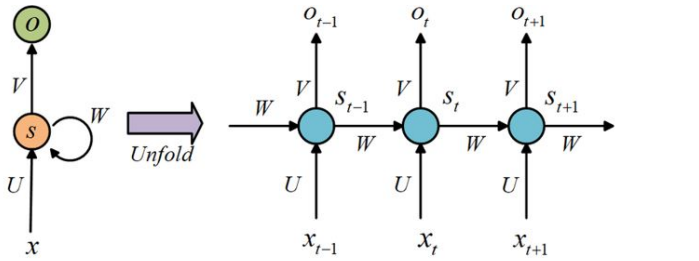


Figure 6: Architecture of the LSTM and the key equations.

2.3 Recurrent Neural Network (RNN)

Unlike traditional feedforward network, RNN process inputs in a sequential manner. Traditional feedforward networks take in a fixed amount of input data all the same time and produce a fixed amount of output data each time. However, RNN takes the input data one at a time in a sequential manner. Figure 7 shows the architecture of the RNN alongside the equation used.



Equation: $S_t = f(S_{t-1}.W + x_t.U)$

t = time step
W & U = weight parameters
S_t = new state
S_{t-1} = old state
x_t = input vector at t
S_{t-1} = next state (predict)
O_t = final output at t

Figure 7: Architecture of the RNN and the key equations.

2.4 Comparison

Table 1 lists the training and evaluation time in addition to the evaluation sMAPE for GRU, LSTM, and RNN models. It is clear that the LSTM takes longer and RNN takes about one-third of the time taken my LSTM. This is because in GRU, c_t and h_t are merged.

Table 1: Training/Evaluation time and evaluation sMAPE of GRU, LSTM, and RNN with two hidden layers.

Type of Model	Training Time (sec)	Evaluation Time (sec)	Evaluation sMAPE
GRU	1242.81	6.85	0.26366
LSTM	1449.93	8.79	0.30842
RNN	548.33	2.47	0.25496

Figure 8 shows that the models are able to almost perfectly capture the trends in energy consumptions.

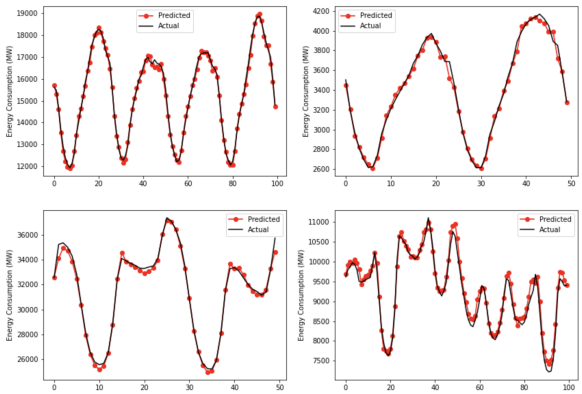


Figure 8: Plot with energy consumption with 2 hidden layers.

2.5 Experiment: One More Hidden Layer

Table 2 lists the training and evaluation time in addition to the evaluation sMAPE for GRU, LSTM, and RNN models with one additional hidden layer. It is clear that the LSTM takes longer and RNN takes about one-third of the time taken my LSTM. This is because in GRU, c_t and h_t are merged.

Table 2: Training/Evaluation time and evaluation sMAPE of GRU, LSTM, and RNN with three hidden layers.

Type of Model	Training Time (sec)	Evaluation Time (sec)	Evaluation sMAPE
GRU	2125.61	10.91	0.3315
LSTM	2430.81	14.32	0.3231
RNN	934.29	3.88	0.3264

Figure 9 shows that the models are able to almost perfectly capture the trends in energy consumptions. There are some discrepancies but it is likely because of the need to fine-tune the hyper-parameters such as learning rate.

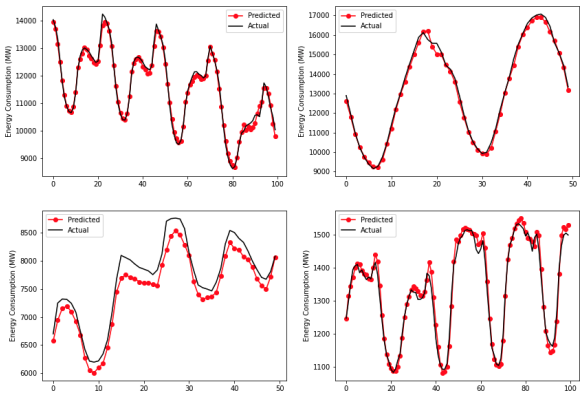


Figure 9: Plot with energy consumption with 3 hidden layers.

How to run There are two jupyter notebook attached with this written report. The first notebook titled "hw2_ResNet.ipynb" is the notebook for the first part of the coding part with variations of ResNet comprising of different number of layers. The second notebook titled "hw2_part2.ipynb" is used for the RNN/LSTM/GRU. Both the notebooks were executed in the Google Colab using their GPU capability.

2.6 Conclusion

The ResNet structure with different number of layers indicate that the deeper networks with 50 layers do not necessarily perform extremely better as compared to the network with just 18 layers. The validation accuracy increased by 0.5% which is almost negligible while the 50-layer network had 23.521M learnable parameters as compared to the 18-layer network which had about half the parameters (11.174M). The GRU/LSTM/RNN showed that they models are able to capture trends of energy consumption and are able to predict the it.