Deep Learning for a Classification Problem in the Case Study of Handwritten Digits Recognition

Zilu Su

Research School of Computer Science, Australian National University
u6342366@anu.edu.au

Abstract. Deep learning applied in the research of classification or regression problems are significantly meaningful for machine learning. Two network models that were used for a classification problem are the key factors in this research. Several methods were applied to evaluate and improve the performance of the model. The achievement of this study shows the importance of selecting appropriate optimizers. In addition, it is found that more complex model may not have better performance. The results of this research have reached a relatively satisfying level. However, there are still several aspects of the future work being left, which should be given high attention.

1 Introduction

The Modified National Institute of Standard and Technology (MNIST) database which is widely used for training and testing in the area of machine learning, is a large database of handwritten digits (Wikipedia, 2018). The MNIST contains the black and white images normalised to fit into a 28x28 pixel. It is a good database for both pattern recognition and machine learning on real-world data without complex preprocessing and formatting (LeCun, Cortes, & Burges, n.d.). The motivation of using this database is that it can help to recognise handwriting. The implementation of the model for classifying handwritten digits is beneficial to make the recognition of letters and words such that a useful handwriting recognition system can be established. The problem is about classifying a handwritten digit to its exact value. Classification is implemented by the model of recurrent neural network (RNN) and bidirectional recurrent neural network (BRNN). The investigations aim to evaluate which model performs better with higher accuracy and efficiency. Therefore, the measures of classification accuracy and running time are taken into consideration.

Table 1. Information of the MNIST data sets

MNIST Data Sets	Number
Training examples	60,000
Testing examples	10,000
Classes	10

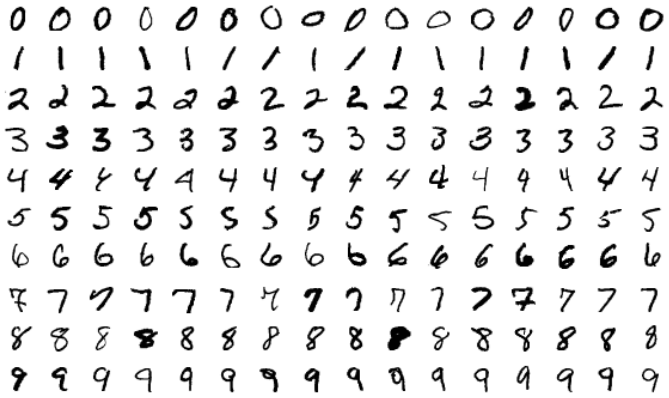

Fig. 1. Sample images from MNIST test dataset
Source: Steppan's own work

2 Methods

2.1 Data Pre-processing

The MNIST is easily accessed via the torchvision package, which consists of model architectures, popular datasets and common image transformations for computer vision (torchvision, 2018). The data was loaded for batch processing.

2.2 Bidirectional Neural Network

A long short-term memory recurrent neural network (LSTM) model was applied in this research. In addition to a recurrent neural network which has a feedback connection from its output back to the input, LSTM contains the memory blocks controlling the mechanisms of memorising and forgetting of contents. Furthermore, a bidirectional neural network uses output neurons as the inputs to gain bidirectional weights. The basic concept of bidirectional LSTM (BLSTM) is that presenting each training sequence forwards and backwards to two independent recurrent nets that are connected to the same output layer. Hence, the network has both complete and sequential information about all points before and after certain point in a given sequence (Zhang, Zheng, Hu, & Yang, 2015). This method was implemented to compare the performance with that of the model without the bidirectional process in this study.

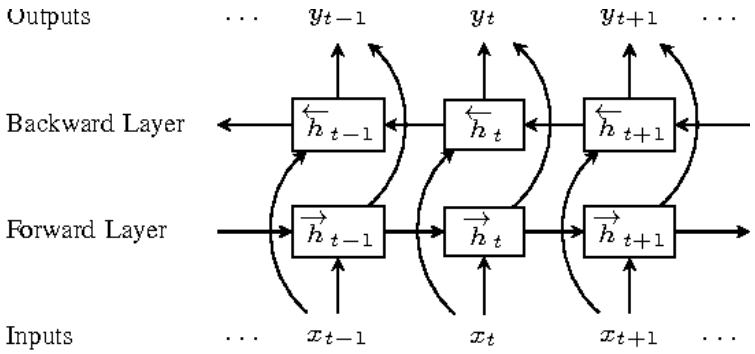

Fig. 2. Bidirectional RNN
Source: Wang, Hong, Soong, He, & Zhao

```
# Define Bidirectional LSTM Model
class BRNN(nn.Module):
    def __init__(self, input_size, hidden_size,
                 num_layers, num_classes):
        super(BRNN, self).__init__()
        self.num_layers = num_layers
        self.hidden_size = hidden_size
        self.lstm = nn.LSTM(input_size, hidden_size,
                            num_layers, batch_first = True,
                            bidirectional = True)    #bidirectional
        self.fc = nn.Linear(hidden_size*2, num_classes)
        # "*2" for bidirection

    def forward(self, x):
        out, _ = self.lstm(x)
        out = out[:, -1, :]
        out = self.fc(out)
        return out
```

2.3 Optimizer Selection

Several optimizers are available to update the weights, such as Stochastic Gradient Descent (SGD), Momentum, RMSProp, Adam, etcetera. Visualization of loss function in terms of these four optimizers can help to choose the most appropriate one for the model. Here, the four optimizers are for the LSTM model with three epochs; and the learning rate is equal to 0.001.

```
optimizer_SGD = optim.SGD(net_SGD.parameters(), lr=learning_rate)
optimizer_Momentum = optim.SGD(net_Momentum.parameters(),
                               lr=learning_rate, momentum=0.8)
optimizer_RMSprop = optim.RMSprop(net_RMSprop.parameters(),
                                  lr=learning_rate, alpha=0.9)
optimizer_Adam = optim.Adam(net_Adam.parameters(),
                            lr=learning_rate, betas=(0.9, 0.99))
```

2.4 Running Time

Recording running time is a good way to evaluate the efficiency of the models for classification. In this research, the running time of the LSTM and the BLSTM for training were compared in order to analyse which model is more time-saving.

```
    starttime = datetime.datetime.now()
    train(epoch)
    endtime = datetime.datetime.now()
    print((endtime - starttime).seconds)    # Compute time costs
```

3 Results and Discussion

The optimizer Adam was selected as the only optimizer for the results.

Table 2. Training time and testing accuracy (training epoch=4, batch size=1000)

Training Epoch (Batch Size=1000)		1	2	3	4
Training Time (Seconds)	LSTM	93	90	87	87
	BLSTM	268	266	267	263
Testing Accuracy (%)	LSTM	76.89	90.61	94.46	95.81
	BLSTM	80.22	90.00	94.14	94.95

Table 3. Training time and testing accuracy (training epoch=8, batch size=640)

Training Epoch (Batch Size=640)		1	2	3	4	5	6	7	8
Training Time (Seconds)	LSTM	91	91	90	94	89	88	88	89
	BLSTM	267	255	254	253	256	262	261	258
Testing Accuracy (%)	LSTM	88.31	94.45	95.92	96.47	97.30	97.50	97.69	97.73
	BLSTM	86.98	93.61	95.45	96.25	96.72	97.00	97.49	97.59

Table 4. Training time and testing accuracy (training epoch=12, batch size=500)

Training Epoch (Batch Size=500)		1	2	3	4	5	6	7	8	9	10	11	12
Training Time (Seconds)	LSTM	97	94	93	96	92	93	97	92	93	92	93	97
	BLSTM	266	265	266	265	266	272	277	271	269	270	274	264
Testing Accuracy (%)	LSTM	90.57	93.76	95.84	96.51	97.25	97.56	97.89	97.89	98.22	98.34	98.04	98.10
	BLSTM	90.39	94.72	96.14	96.92	97.30	97.51	97.97	98.40	98.40	98.35	98.55	98.48

Table 2, 3 and 4 show that testing accuracy is related to the number of training epochs as well as the batch size. With the increase of the number of training epochs and batch size, testing accuracy for both models was improved. Besides, the training time of BLSTM is roughly three time as long as that of LSTM. The results indicate that BLSTM did not perform better than LSTM as the proportions of the testing accuracy of them are similar. Conversely, LSTM saved more time with a little bit higher testing accuracy than that of BLSTM.

A fairly standard model comprising two convolutional layers which are followed by two fully connected layers and a softmax output layer with ten classes was used for the MNIST benchmark (the MNIST data without any transformation) to evaluate Adam's baseline performance. With a high value of accuracy, 99.63%, Adam performed pretty well without synchronous training (Chilimbi, Suzue, Apacible, & Kalyanaraman, 2014). Compared with this result, what I have done in this study still has a lot of room for improvement. Concerning the reason of this gap, convolutional neural networks might perform better than LSTM. Therefore, for the next research, it is necessary to apply a convolutional model to inspect whether there will be a new achievement.

In terms of the setting of hyperparameters, input size is 28 which is equal to the image size; and the number of classes was set according to the digits from 0 to 9. Although the parameters for the number of layers (2), hidden size (128) and the learning rate (0.001) seem reasonable, the idea that this is the best combination of the parameters cannot be proved.

Additionally, parameters also need to be considered in the part of optimizer selection. Regarding to the optimizers Momentum, RMSprop and Adam, the parameters for attenuation are defined by `momentum=0.8`, `alpha=0.9` and `betas=(0.9, 0.99)`, which have been showed above. Obviously, Figure 3 illustrates that Adam performed better than RMSProp with lower loss from approximately the 60th step to the end. Undoubtedly, the advantage of Adam, offset correction, makes the learning rate for its each iteration have a certain range, contributing to more stable parameters. However, it is noteworthy that the values of loss of the optimizers, both SGD and Momentum, stay at the worst level. The reason might be that the number of epochs (only 3) is quite small and the exact answer is supposed be figured out in the future work.

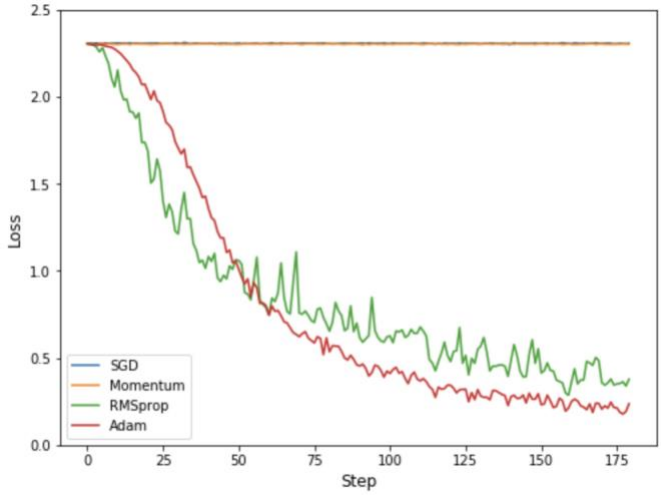

Fig. 3. Loss of Four Optimizers

4 Conclusion and Future Work

This research aims to correctly classify handwritten digits into their corresponding classes. Long short-term memory recurrent neural network and a bidirectional model were applied for the data sets. The findings show that the model without bidirectional technique performed better with the relatively satisfying efficiency and accuracy. In future work, the reason why bidirectional long short-term memory network did not have more strongly positive influence on the training set should be analysed. Besides, more effective and efficient hyperparameters as well as the models that are more time-saving with higher accuracy are supposed to be found through a deeper research. More importantly, graphics processing unit will be expected to be implemented so that a better classification performance can be achieved.

References

Chilimbi, T. M., Suzue, Y., Apacible, J., & Kalyanaraman, K. (2014, October). Project Adam: Building an Efficient and Scalable Deep Learning Training System. In *OSDI* (Vol. 14, pp. 571-582).

LeCun, Y., Cortes, C., Burges, J.C., C. (n.d.). The MNIST Database of Handwritten Digits. Retrieved from http://yann.lecun.com/exdb/mnist/

MNIST database. (2018, May 23). Wikipedia, the free encyclopedia. Retrieved May 26, 2018, from https://en.wikipedia.org/wiki/MNIST_database

Steppan, J. (2018). A few samples from the MNIST test dataset. Retrieved from https://en.wikipedia.org/wiki/MNIST_database#/media/File:MnistExamples.png

Torchvision. (2018). Retrieved from https://pytorch.org/docs/master/torchvision/

Wang, P., Hong, Z., Soong, F.K., He, L., & Zhao, H. (2015). A Unified Tagging Solution: Bidirectional LSTM Recurrent Neural Network with Word Embedding. *CoRR, abs/1511.00215*.

Zhang, S., Zheng, D., Hu, X., & Yang, M. (2015). Bidirectional long short-term memory networks for relation classification. In *Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation* (pp. 73-78).