Deep Learning Fundamental
Handwritten Digit Recognition
Assignment 1 Report

610921231 資工系碩一 李宸綾

Global Constants

Firstly, I used one section to declare some global constants we will use later for model training. I set up epochs, batch size, verbose and the ratio of validation split at the beginning of my code to ensure that we will not change the value of these variables accidently and the only difference between these models is optimizer. Otherwise, this experiment (discuss and compare the performance between the models using different optimizer) would be not enough fair and impartial.

```
EPOCHS = 50 # the number of epochs is a hyperparameter of gradient descent that controls the number of
complete passes through the training dataset

BATCH_SIZE = 128 # the batch size is a hyperparameter that defines the number of samples to work through
before updating the internal model parameters

VERBOSE = 1 # used for decide how much output we want when training the model

NB_CLASSES = 10 # number of outputs = number of digits (from 0 ~ 9)

N_HIDDEN = 128 # positive integer, dimensionality of the output space

VALIDATION_SPLIT = 0.2 # how much TRAIN is reserved for VALIDATION

DROPOUT = 0.3 # the ratio of dropout variable
```

Plot.1 Global Constants Declaration

Dataset Loading and Preprocessing

For the dataset loading, we could just call the function that Keras has already built-in in its library.

```
# loading MNIST dataset, verifing the split between train and test is 60,000, and 10,000 respectly.
# one-hot is automatically applied.
mnist = keras.datasets.mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

Plot.2 Dataset Loading

After we loaded the MNIST train dataset, we have to reshape it from 60000 x 28 x 28 to 60000 x 784 in order to adapt the input layer shape.

```
# X_train is 60000 rows of 28×28 values ⟶ reshaped in 60000 x 784
RESHAPED = 784
X_train = X_train.reshape(60000, RESHAPED)
X_test = X_test.reshape(10000, RESHAPED)
```

Plot.3 Dataset Reshape

Then we will process the normalization with these train and test dataset. Normalizing the data generally speeds up learning and leads to faster convergence. Do not forget to transfer data type from int to float because after the normalization the value of the data will not be integer anymore.

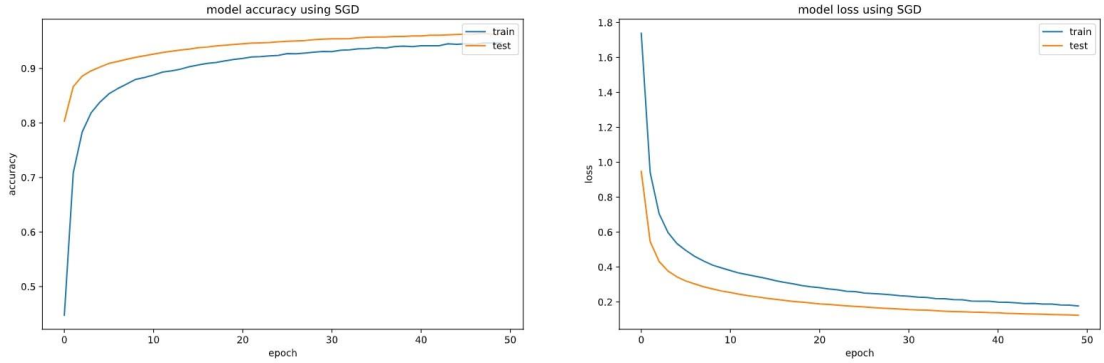
```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# applying normalization in [0,1]
X_train, X_test = X_train / 255.0, X_test / 255.0
```

Plot.4 Dataset Normalization

Comparison between different Optimizers

SGD (stochastic gradient decent)



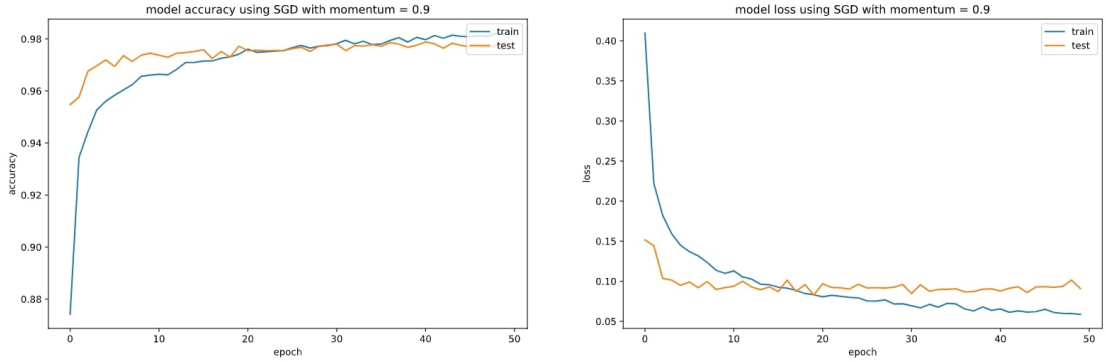
Plot.5 Training Process Curves of SGD

- After 50 epochs, the accuracy of train reaches around 0.9 and test reaches 0.94. The loss of both train and test is around 0.2.
- The curves from accuracy plot and loss plot both are very smooth.

SGD with momentum = 0.9

```
# setting up the optimization of our weights
"""
Arguments of SGD(lr, decay, momentum, nesterov)

lr: float ≥ 0. Learning rate.
momentum: float ≥ 0. Parameter updates momentum.
decay: float ≥ 0. Learning rate decay over each update.
nesterov: boolean. Whether to apply Nesterov momentum.
"""
sgd = keras.optimizers.SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
```

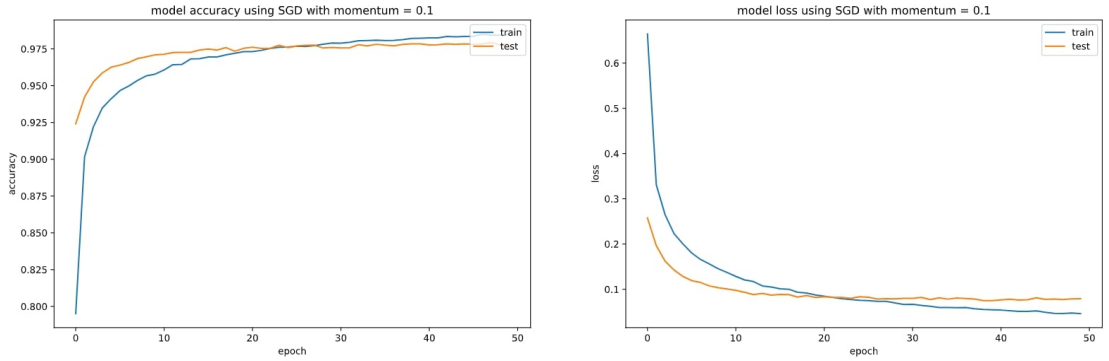


Plot.6 Training Process Curves of SGD with momentum=0.9

- With momentum = 0.9, both accuracy and loss have better performance than the original SGD in 50 training epochs.
- We can say the momentum is like an accelerator for the training process, causing the gradient decent method to find out the minimum (or local minimum) faster. Therefore, we can see the zig-zag pattern curve from both accuracy and loss plot.

SGD with momentum = 0.1

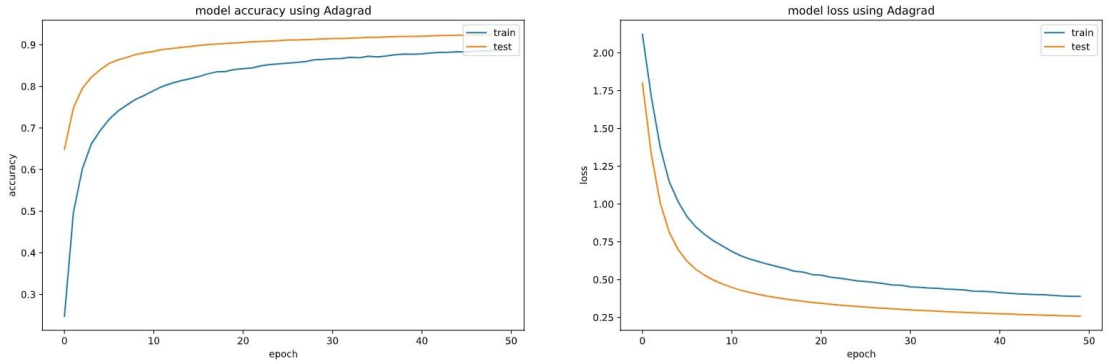
```
sgd = keras.optimizers.SGD(lr=0.1, decay=1e-6, momentum=0.1, nesterov=True)
```



Plot.7 Training Process Curves of SGD with momentum=0.1

- Smaller value of momentum causing smaller zig-zag pattern for the curve.
- Has slightly different performance after 50 epochs compared to momentum = 0.9.

Adagrad

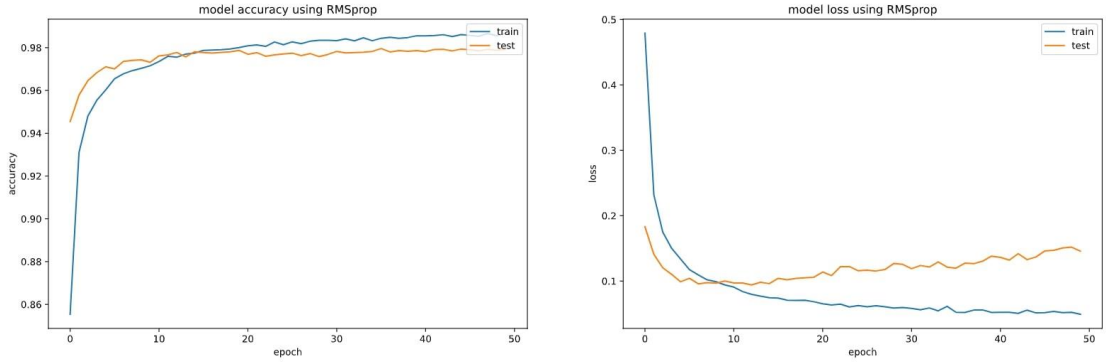


Plot.8 Training Process Curves of Adagrad

- Has very smooth curve for both accuracy and loss plot as SGD.

- Slower convergence for training process after 50 epochs (same training epochs but lesser accuracy and higher loss can be observed in both train and test data compared to SGD)

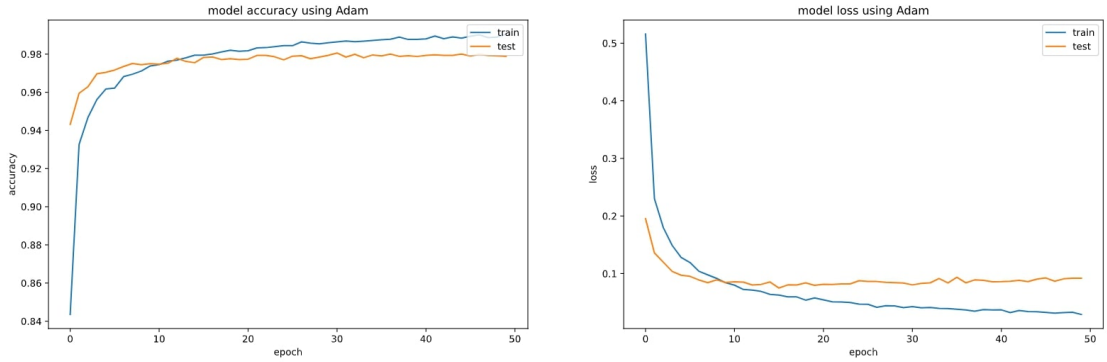
RMSprop



Plot.9 Training Process Curves of RMSprop

- Getting high accuracy for both test and train data in very early epoch (around epoch 15) and very low loss (around epoch 10).
- The loss of test is slightly increasing since epoch 5 seems there is an overfitting occurred.

Adam



Plot.10 Training Process Curves of RMSprop

- Has really similar curve pattern as RMSprop for both accuracy and loss plot.
- Observing the loss plot we can find out that since epoch 9 the training process has no significant improvement (although the loss of train is decreasing, the loss of test is stuck around 0.1 means epoch 9 is a good timing to stop the training process).