Convolutional Neural Network (CNN) Experiment for Fashion MNIST Dataset

David Ishak Kosasih (20195033)

November 1, 2019

NB : Experiment was conducted by using keras. Training epoch was fixed to 10 for all of the experiment. Source code can be found on https://github.com/ishakdavidk/Machine-Learning.

1. Mean subtraction and normalization :

   Mean subtraction is usually calculated by subtracting the mean value of our dataset from every individual input data. In keras, suppose x is our input matrix, then to calculate mean subtraction we only need to perform x = x - x.mean().

   For normalization, I divide each data dimension by its standard deviation after it has been zero-centered (Mean subtraction). In order to achieve this in keras, we only need to perform x = x / x.std().

	Original Pixel Value			
	Min	Max	Mean	Std
Train	0	255	72.94035223214286	90.02118235130519
Test	0	255	73.14656658163265	89.87325907809718

   The table above is the original pixel value.

	New Pixel Value			
	Min	Max	Mean	Std
Train	-0.8102577	2.0224090	-1.7480801e-17	1.0
Test	-0.8138858	2.0234432	-5.3145696e-17	0.99999999

The table above is the new pixel value of train and test dataset after being subtracted by its mean value and normalized. We can see now that the mean value of our dataset is zero and its standard deviation is one.

2. Xavier and He initialization experiment :

   In order to perform Xavier initialization in keras, we only need to specify the value of kernel_initializer parameter in keras.layer.Dense function to glorot_normal. If we want to use He initialization, we just need to assign he_normal as the value of the parameter.

Initialization	Accuracy on Training Data		
	1 Hidden 128 Neurons	20 Hidden 128 Neurons Each	1 Hidden 2560 Neurons
Xavier	0.9170	0.7278	0.9223
He	0.9175	0.8532	0.9239

Initialization	Accuracy on Test Data		
	1 Hidden 128 Neurons	20 Hidden 128 Neurons Each	1 Hidden 2560 Neurons
Xavier	0.8812	0.7081	0.8799
He	0.8824	0.8384	0.8829

   The two table above compare the performance of xavier and he initialization on training and test data. These two initialization technique have similar performance; however, He initialization tend to perform better when the model has a large number of layers.

3. Network configuration experiment :

   In this experiment, 3 models were used. The first model has 1 hidden layer with 128 neurons, the second model has 10 hidden layers with 128 neuron each while the last model has 1 hidden layer with 2560 neurons. With this set-up, I tried to observe the performance of the model when it has normal number of neurons and hidden layers, large number of layers and large number of neurons. Among all of these three set-up, large number of neurons had the best performance compared to the other models while when we use too large number of hidden layer, the model tend to have a bad performance. However, when we use too large number of neurons, the model will be

theoretically prone to over-fit our dataset. This cannot be seen from my experiment since the training epoch was fixed to only 10. For observing this theory, we need to conduct further experiment.

The experiment result can seen in the table below.

Dataset	Accuracy		
	1 Hidden 128 Neurons	20 Hidden 128 Neurons Each	1 Hidden 2560 Neurons
Training	0.9165	0.7875	0.9217
Test	0.8807	0.8011	0.8837

4. Gradient optimization techniques experiment :

   In this experiment, I observed Adam and RMSprop optimization techniques. These two techniques have a very similar result. If we read the paper named "ADAM - A Method for Stochastic Optimization", the author himself said that his technique, Adam, has a very similar performance with RMSprop; however, since RMSprop lacks a bias-correction term, Adam optimization still tend to have better performance. After running several experiment, the superiority of Adam optimization can be seen, but the performance difference was actually very small.

   The experiment result can seen in the table below.

Gradient Optimization	Accuracy on Training Data		
	1 Hidden 128 Neurons	20 Hidden 128 Neurons Each	1 Hidden 2560 Neurons
Adam	0.9169	0.7839	0.9224
RMSprop	0.9101	0.7902	0.9162

Gradient Optimization	Accuracy on Test Data		
	1 Hidden 128 Neurons	20 Hidden 128 Neurons Each	1 Hidden 2560 Neurons
Adam	0.8816	0.733	0.8821
RMSprop	0.8741	0.756	0.8853

5. Activation functions experiment :

   For the activation function, I opted to run experiment on ReLU and Leaky ReLu. The Difference between these two activations is that

Leaky ReLu allows for a small, non-zero gradient. Theoretically, Leaky ReLu should outperform ReLU since it ensure that the neuron will never die; however, from this experiment, ReLU tend to perform better than Leaky ReLu except when the model has a very deep layer. In this situation, Leaky ReLU completely outperform ReLU.

The experiment result can seen in the table below.

Activation Functions	Accuracy on Training Data		
	1 Hidden 128 Neurons	20 Hidden 128 Neurons Each	1 Hidden 2560 Neurons
ReLU	0.9176	0.7393	0.9219
Leaky ReLU	0.8962	0.8752	0.8814

Activation Functions	Accuracy on Test Data		
	1 Hidden 128 Neurons	20 Hidden 128 Neurons Each	1 Hidden 2560 Neurons
ReLU	0.881	0.7408	0.8746
Leaky ReLU	0.8741	0.8481	0.857

6. Regularization techniques experiment :

In this experiment, I opted to observed Dropout and L1. From the experiment result, we can clearly see that L1 tend to perform better on training data while Dropout tend to perform better on test data. This indicates that Dropout outperform L1 in term of preventing over-fit problem. However, both regularization techniques made the model performance even worse when it has too large number of layers.

The experiment result can seen in the table below.

Regularization	Accuracy on Training Data		
	1 Hidden 128 Neurons	10 Hidden 128 Neurons Each	1 Hidden 1280 Neurons
Dropout	0.8600	0.3713	0.8739
L1	0.8908	0.0988	0.8917

Regularization	Accuracy on Test Data		
	1 Hidden 128 Neurons	10 Hidden 128 Neurons Each	1 Hidden 1280 Neurons
Dropout	0.8669	0.1054	0.8759
L1	0.851	0.1	0.8482