HOMEWORK 2

Comparison of 3 Classifiers on MNIST dataset

INDEX

1.	**Abbreviation/Symbols List**	**2**
2.	**Introduction**	**3**
3.	**Problem Definition**	**5**
4.	**Experimental Evaluation**	**6**
5.	**Conclusion**	**13**
6.	**Reference**	**14**

Abbreviation/Symbols List

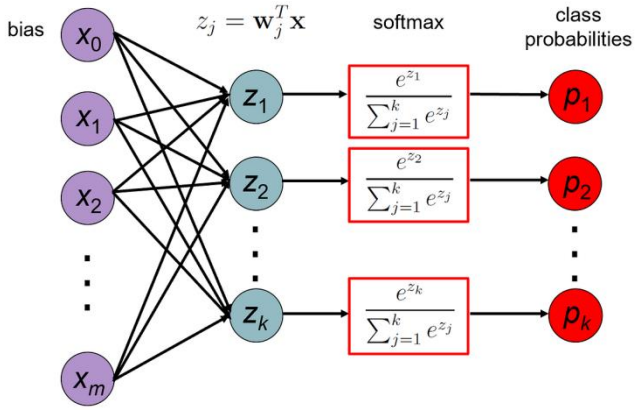
MNIST	Modified National Institute of Standards and technology database
MLP	Multilayer Perceptron
CNN / ConvNet	Convolutional Neural Network
BN	Batch Normalization
x_train	Training dataset input
y_train	Training dataset output
x_test	Test dataset input
y_test	Training dataset output

Introduction

MNIST dataset is a simple and popular dataset for image classification. It contains large amount images of handwritten digits which are commonly used for training and testing image processing systems.

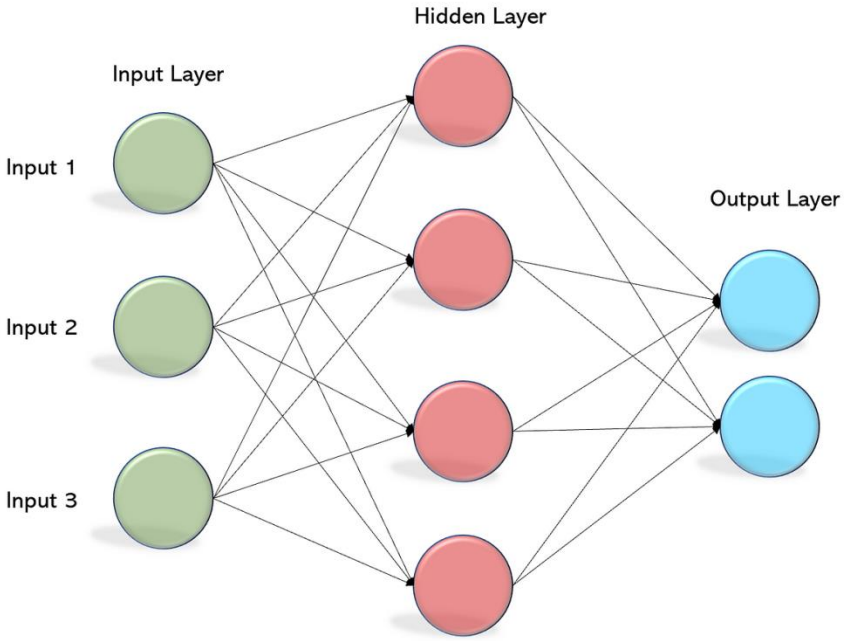
We are applying 3 classifier models on the MNIST dataset:

1) Softmax Classifier



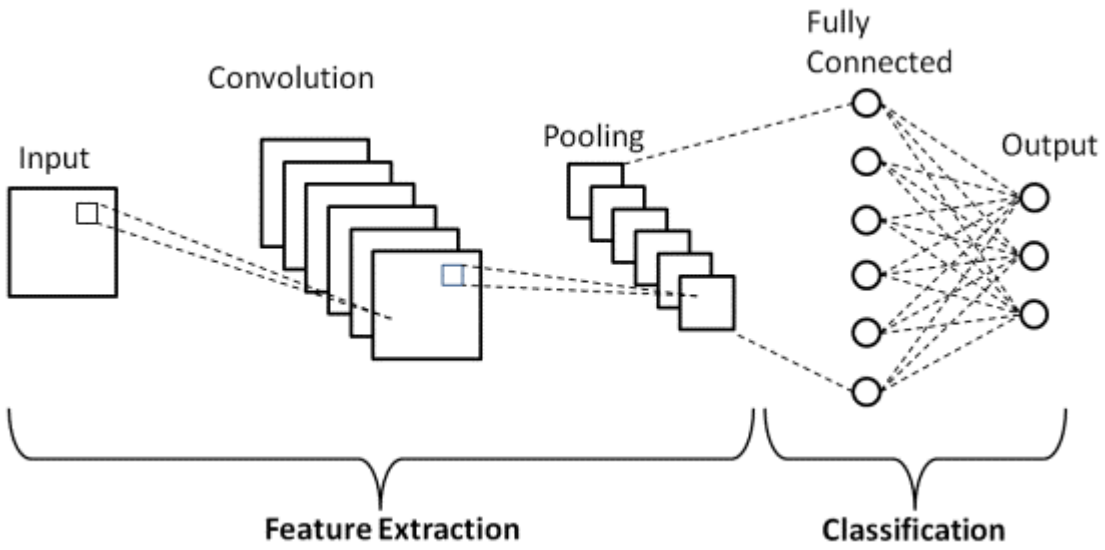
It is a supervised learning algorithm which is mostly used when multiple classes are involved. It assigns probability distribution to each class. As softmax is mostly used as an activation function in the final layer of a deep learning model, softmax classifier has only 2 layers; the input and the output layer.
It is used here as output of MNIST model has N classes.

2) Multilayer Perceptron (MLP)



A fully connected multi-layer neural network is called a MLP. A basic MLP has 3 layer; input, hidden and output layer. The input signal, which is in the form of vector, is given to the inner layer where as task like prediction and classification is done in outer layer. Hidden layer does other kinds of computation. In MLP the mapping between the input and output in non-linear. It is used for image classification as it has hidden layers, so it involves more parameters for feature representation.

3) Convolutional Neural Network (CNN)



CNN is a special type of neural network that roughly imitates human vision. It uses the method of convolution to reduces images into a form that is easier to process, without losing features which are important for getting a good prediction. Beside the input layer and the output layer, it mainly has 2 layers:

- Convolutional Layer: It is mainly used for feature extraction. Few of its components consist of input data, a filter and feature map.
- Pooling Layer: This layer is responsible for reducing the special size of the convolved feature, in other words, it reduces dimension.

This model is used for image classification because it is able to automatically learn the spatial hierarchies to features, such as edges, textures, and shapes, and these are important for recognizing objects in image.

Problem definition

The main motivation behind this part of the assignment is to explore classifier for image classification. The classifiers we are exploring are soft-max regression, MLP and CNN.

These classifiers are selected based on their complexity; starting from simplest classifier (Soft-max Regression) to more complex classifier (CNN). As er are provided with the MNIST dataset, each of these classifiers can be used for image classification.

At the same time, we also have to study the behavior of these classifiers, when dropout and batch normalization are added.

Experimental Evaluation

Methodology common for all three classifiers

1. Import the libraries

   The first step of this assignment is selecting the required packages and importing them. I have selected libraries such as keras to help in model building and model training. The libraries are as follows:

   - **keras.datasets**: Imports MNIST dataset
   - **keras.models.Sequential:** helps to build sequential model with linear stack of layers.
   - **keras.layers.Dense:** Creates a classic fully connected neural network layer.
   - **keras.layers.Dropout**: The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting.
   - **keras.layers.Conv2D:** This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs.
   - **keras.layers.MaxPool2D:** Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window (of size defined by pool_size) for each channel of the input.
   - **keras.layers.Flatten:** Flattens the input.
   - **keras.layers.BatchNormalization:** Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.
   - **keras.utils.to_categorical**: Converts a class vector (integers) to binary class matrix.

2. Load the MNIST dataset

   Here we are loading and dividing the dataset into training and test dataset.

3. Reshape the dataset as per the dimensional needs for model training

   Here we reshape the dataset dimension as per our needs. For soft-max and MLP we reshape the input dataset from (60000, 28, 28, 1) to (60000, 784) while for CNN we are keeping it the same.

4. Normalize the data in dataset to the range of 0 to 1

   This step involves normalizing of input dataset by dividing the data by 255 as it is the maximum pixel value.

5. One-Hot Encoding as the output is categorical

   As we are using MNIST dataset for image classification, we have given the classes into numerical values by using one hot encoding. It converts the output from a simple vector to a matrix with N class columns.

6. Initializing the parameters for the model before training

   Parameters such as output dimension, input dimension, batch size, and epochs are initialized to a constant value.

7. Defining / building the model

   In this section we are defining our model. We are specifying:
   - the number of layers
   - the type of layers
   - the parameters to each layer

   CNN has Conv2D layer and MaxPool2d layer while MLP has a lot of hidden layers.

8. Compiling and training the model with training dataset and calculating the accuracy with validation and test dataset

   Here we are just compiling the model and checking the fitting of our model with the dataset. We are also checking the performance of the dataset by calculating its accuracy on the test dataset.

9. After which we will do step 7 and 8 for BN and dropout.

Observation and Discussion

1. **Comparison between the models**

Soft-Max Regression

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_1 (Dense)	(None, 10)	7850

MLP

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_2 (Dense)	(None, 100)	78500
dense_3 (Dense)	(None, 10)	1010

CNN

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 25)	250
max_pooling2d_1 (MaxPooling2D)	(None, 26, 26, 25)	0
flatten_1 (Flatten)	(None, 16900)	0
dense_2 (Dense)	(None, 100)	1690100
dense_3 (Dense)	(None, 10)	1010

	Number of layers	Accuracy	Loss
SoftMax Regression	2	0.9263	0.2651
MLP	3	0.9772	0.0744
CNN	5	0.9820	0.0728

From the above table, we can see that the CNN model has performed the best in image classification of the MNIST dataset. Also the CNN also has largest number of layers, so we can also see a relation between no. of layers and performance of the model. Softmax with only 2 layer performed the lowest. Whereas MLP with only 3 layers has shown a comparable performance to CNN. But very complex models with lot of layers are generally not recommended because it will then over fit to the dataset and will have poor generalization where as models with too little layers will under fit. We can also see that as the loss decrease the accuracy of a model increases.

2. Effect of adding dropout to the model

Soft-Max Regression

Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 784)	0
dropout (Dropout)	(None, 784)	0
dense_2 (Dense)	(None, 10)	7850

MLP

Layer (type)	Output Shape	Param #
flatten_23 (Flatten)	(None, 784)	0
dropout_18 (Dropout)	(None, 784)	0
dense_40 (Dense)	(None, 100)	78500
dense_41 (Dense)	(None, 10)	1010

CNN

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 25)	250
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 25)	0
dropout (Dropout)	(None, 26, 26, 25)	0
flatten_2 (Flatten)	(None, 16900)	0
dropout_1 (Dropout)	(None, 16900)	0
dense_4 (Dense)	(None, 100)	1690100
dropout_2 (Dropout)	(None, 100)	0
dense_5 (Dense)	(None, 10)	1010

	Accuracy	Accuracy after Dropout
SoftMax Regression	0.9263	09240
MLP	0.9772	0.9178
CNN	0.9820	0.9850

Adding the dropout to models have slightly fluctuation in softmax regression and CNN but no major trend can be seen it these two models, but the performance of the MLP deteriorated by a large amount by just adding one layer of dropout. This either concludes that MLP doesnot fare will with dropout or we can say that dropout needs to be placed in another layer.
As per my observation, placement of the dropout affects the accuracy of the model. Dropouts at different position affects the model in different ways. Also at the same time, dropout rate also affects the accuracy of the model. So the number of dropouts that can be placed in a model, the rate of dropout and the position of dropouts can only be determined by trial and error. But according to my assumption, if the input units are too large, then we can also increase the dropout rate, and visa versa…again this just an assumption.

3. **Effect of adding batch noralization to the model**

Soft-Max Regression

Layer (type)	Output Shape	Param #
flatten_5 (Flatten)	(None, 784)	0
batch_normalization_2 (BatchNormalization)	(None, 784)	3136
dense_5 (Dense)	(None, 10)	7850

MLP

Layer (type)	Output Shape	Param #
flatten_24 (Flatten)	(None, 784)	0
dense_42 (Dense)	(None, 100)	78500
batch_normalization_8 (BatchNormalization)	(None, 100)	400
dense_43 (Dense)	(None, 10)	1010

CNN

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 26, 26, 25)	250
max_pooling2d_3 (MaxPooling2D)	(None, 26, 26, 25)	0
batch_normalization (BatchNormalization)	(None, 26, 26, 25)	100
flatten_3 (Flatten)	(None, 16900)	0
dense_6 (Dense)	(None, 100)	1690100
dense_7 (Dense)	(None, 10)	1010

	Accuracy	Accuracy after Batch Normalization
SoftMax Regression	0.9263	0.9242
MLP	0.9772	0.9720
CNN	0.9820	0.9817

To be honest I have not observed any particular improvement after adding batch normalization in any of these models. So it can be, either my implementation is wrong, or batch normalization doesn't have a particular effect of these models.

4. Effect of adding both dropout and batch normalization

	Accuracy	Accuracy after adding both
SoftMax Regression	0.9263	0.9220
MLP	0.9772	0.9172
CNN	0.9820	0.9840

Softmax did not show any improvement after adding both, where as performance of MLP drop. Most likely, it has to be because of dropout. And as for CNN, there is only a slight increase in the performance.

5. Performance of a model

Also the performance of the model can also be adjusted by many other factors such as the optimizer, number of epochs and activation functions. But all of these can be selected ob the trial and error basis.

6. Time Taken

Time Taken for computation was the longest for CNN more probably due to more number of layers as compared to the other models. SO we can also say that more the complexity of a model, more the time it will take to compute.

Conclusion

According to my observation, we can see that it is very difficult to improve the performance of softmax as it is a very simple model. To improve its performance we can slightly tune its parameters, but it wont be helpful for complex dataset, and it is considered as last in comparison to the other two models for image classification.

On the other hand, MLP having 3 layers have shown much better performance and that too, in very less time. CNN on the other hand has shown an excellent performance w. r. t. accuracy, but the time it took to classify the datas for 10 epoch was much longer as compared to MLP.

So in my opinion, CNN can be selected for complex models, as it has good number of hidden layers and provides the best accuracy among the three, but MLP gave a comparable performance with lesser layers and lesser time, so it can be used for solving simpler problems. And we can improve the accuracy of the model by using dropout in CNN but using dropout in MLP need further study from my end. Also batch normalization have not shown much of an improvement it any of the models, so this will also need further exploration.

But it was interesting to get to know and explore all the models, and the parameters.

Reference

- https://medium.com/analytics-vidhya/multi-layer-perceptron-using-keras-on-mnist-dataset-for-digit-classification-problem-relu-a276cbf05e97
- https://keras.io/guides/sequential_model/
- https://keras.io/api/layers/core_layers/
- https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/
- https://stackoverflow.com/questions/34716454/where-do-i-call-the-batchnormalization-function-in-keras
- https://medium.com/analytics-vidhya/neural-network-mnist-classifier-from-scratch-using-numpy-library-94bbcfed7eae
- https://www.kaggle.com/code/scaomath/simple-neural-network-for-mnist-numpy-from-scratch
- https://www.kaggle.com/code/valentynsichkar/convolutional-neural-network-from-scratch-mnist
- https://dmkothari.github.io/Machine-Learning-Projects/MLP_with_MNIST.html
- https://www.kaggle.com/code/gpreda/simple-introduction-to-cnn-for-mnist-99-37
- https://standardfrancis.wordpress.com/2015/04/16/batch-normalization/
- https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md
- https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/
- https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/
- https://stackoverflow.com/questions/42883547/intuitive-understanding-of-1d-2d-and-3d-convolutions-in-convolutional-neural-n
- https://stackoverflow.com/questions/58398491/valueerror-shape-mismatch-the-shape-of-labels-received-15-should-equal-th
- https://towardsdatascience.com/pitfalls-with-dropout-and-batchnorm-in-regression-problems-39e02ce08e4d