Task 3: Neural Networks

Please refer to 'Task 3.ipynb' notebook for code and graphs supporting observations made in this report.

1.0 Introduction

In this task, we implement a multi-layer neural network using Python's NumPy library. The neural network will then be configured to train against a multiclass classification problem. For the purposes of this task, we'll be training it on the Fashion-MNIST dataset. This dataset consists of 70,000 28x28 grayscale images and contains a total of 10 classes.

2.0 Components of a Neural Network

The neural network class was implemented such that it accepts a customizable set of parameters for number of hidden layers, number of nodes and the activation functions. Some of these network parameters are discussed in greater detail under Task 4. Here, we will focus on the implemented activation functions and optimizers.

2.1 Activation Functions

Activation functions serve the purpose of introducing non-linearity into the network which helps networks learn better. They are typically applied to the output of hidden layers and determine whether a node fires or not much like mechanisms observed in biological systems. A key characteristic of all activation function is that they are differentiable everywhere. This is crucial as when training a model, during back-propagation the derivative of the function is needed in calculation of loss and determining how to update network parameters. For this task, 3 different activation functions were implemented.

Sigmoid : $f(x) = \frac{1}{1+ e^{-x}}$

This activation function accepts an input of any value and scales output between 0 and 1. This is useful for binary classification problems where the output can be interpreted as the probability of the predicted class. Some disadvantages in using this for activation are:

1. Mathematical operations during forward and backward pass can be computationally expensive
2. Suffers from vanishing gradient problem, that is values that are far away from the origin are close to zero. As such, in using gradients for updating weights this can result in either the network taking a long time to converge or if gradient falls to zero, no learning happens.

Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$

This activation function deactivates nodes with negative inputs. This is often used in hidden layers as it is computationally a more efficient solution that other activation functions. While this doesn't suffer from the vanishing gradient problem, for values x<0 , the gradient = 0. This can result in dead neurons in the network that are never activated and consequently never "learn" during back propagation.

SoftMax: $\boldsymbol{\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}}$

This activation function is typically used on the output layer of multi-class classification problems, as for given an input, the function outputs a vector where the sum of the elements is equal to one. This vector be interpreted as the probability distribution of the predicted classes, where the class with the highest probability is chosen as the predicted class.

2.2 Optimizer

To allow for updating of weights more frequently and faster convergence of the algorithm, a mini-batch gradient descent algorithm was implemented. This optimization technique has the advantages of both Batch Gradient Descent and Stochastic Gradient Descent.

It works by partitioning the input dataset into batches of a fixed size. Then for each epoch instead for passing all inputs at once, each batch goes through a forward pass where the error in prediction of that batch is calculated, and weights updated such that the network is already undergoes learning before the next batch is passed through the network. All batches are processed for each epoch however as weights are updated more frequently the network is likely to converge faster. For this technique, we may observe fluctuations in the loss function as a smaller set of inputs are processed each iteration.

3.0 Model Training & Results

Data Preprocessing

The dataset used was from the keras datasets library. As this is a standard machine learning dataset, there was minimal data preprocessing required. There were two main data preprocessing tasks:

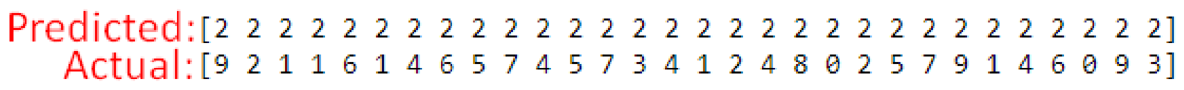
1. One hot encoding the target variable
2. Normalizing and flattening the input features

Results

Although different architectures were tried and evaluated, any single network configuration was able to only achieve a maximum of approximately 10% accuracy. This suggests that there could be a potential issue in the implementation of the neural network in one or all of the following areas:

1. Calculation of the loss function
2. Backpropagation of the gradients
3. Updating of the weight and biases

Closer examination of the output suggest that network is predicting the same class for all outputs. Therefore, based on the proportion of the train or test set that is of the class predicted by the network (for all inputs) , the network reports a roughly 10% accuracy.



```
Predicted:[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
   Actual:[9 2 1 1 6 1 4 6 5 7 4 5 7 3 4 1 2 4 8 0 2 5 7 9 1 4 6 0 9 3]
```

Figure 3.1 : Output of predicted and actual class by ANN

4.0 References

[1] Brownlee, J. (2021, January 21). *How to choose an activation function for deep learning*. Machine Learning Mastery. Retrieved January 2, 2022, from https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

[2] Choudhary, S. (2019, December 13). *How to choose Best Activation Function for You Model*. Medium. Retrieved January 2, 2022, from https://medium.com/@siddharthzs/how-to-choose-best-activation-function-for-you-model-8af90557245b

[3] Patrikar, S. (2019, October 1). *Batch, Mini Batch & stochastic gradient descent*. Medium. Retrieved January 2, 2022, from https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a

Code References:

[4] Skalski, P. (2018, October 19). *Let's code a neural network in plain NumPy*. Medium. Retrieved January 2, 2022, from https://towardsdatascience.com/lets-code-a-neural-network-in-plain-numpy-ae7e74410795

[5] Sentdex. (2020, December 23). *p006-softmax-activation.* GitHub. Retrieved January 2, 2022, from https://github.com/Sentdex/NNfSiX/blob/master/Python/p006-Softmax-Activation.py

[6] Jana, A. (2019, April 29). *Understanding and implementing neural network with Softmax in python from scratch*. A Developer Diary. Retrieved January 2, 2022, from http://www.adeveloperdiary.com/data-science/deep-learning/neural-network-with-softmax-in-python/

[7] Jana, A. (2020, June 1). *Understand and implement the backpropagation algorithm from scratch in Python*. A Developer Diary. Retrieved January 2, 2022, from http://www.adeveloperdiary.com/data-science/machine-learning/understand-and-implement-the-backpropagation-algorithm-from-scratch-in-python/

Percentage of code borrowed: 70%

Task 4: Implementing Neural Network using PyTorch

Please refer to 'Task 4.ipynb' notebook for code and graphs supporting observations made in this report.

1.0 Introduction

For this task the CIFAR-10 dataset was used. The dataset consists of a total of 60,000 images where each image is labelled with one of the following classes – airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

The objective of this task is to implement a classifier using the PyTorch library to accurately label the images in the CIFAR-10 dataset. For this multiclass classification problem, the strategy to find a suitable network architecture will be by first establishing a baseline neural network model with reasonable accuracy (approx. 50%) and then tuning the model parameters and network architecture through techniques taught in the course.

2.0 Establishing a baseline model

2.1 Data Preparation

The dataset used was loaded directly from the torchvision, PyTorch's datasets library. The data was split into 45,000 images for training, 5,000 for validation and 10,000 for testing. The library by default provides a test and train set however, to ensure the model doesn't overfit while training and generalizes well to new datapoints, a validation set was created by setting aside 5,000 images from the train set.

2.2 Defining a baseline model

To define a baseline model, multiple different network architectures were explored by tuning the following hyperparameters - number of hidden layers, number nodes in each layer, activation functions, optimizers and learning rate. This section will briefly discuss what the parameters are, the rationale for picking certain ranges and finally the parameter values used and performance of the baseline model.

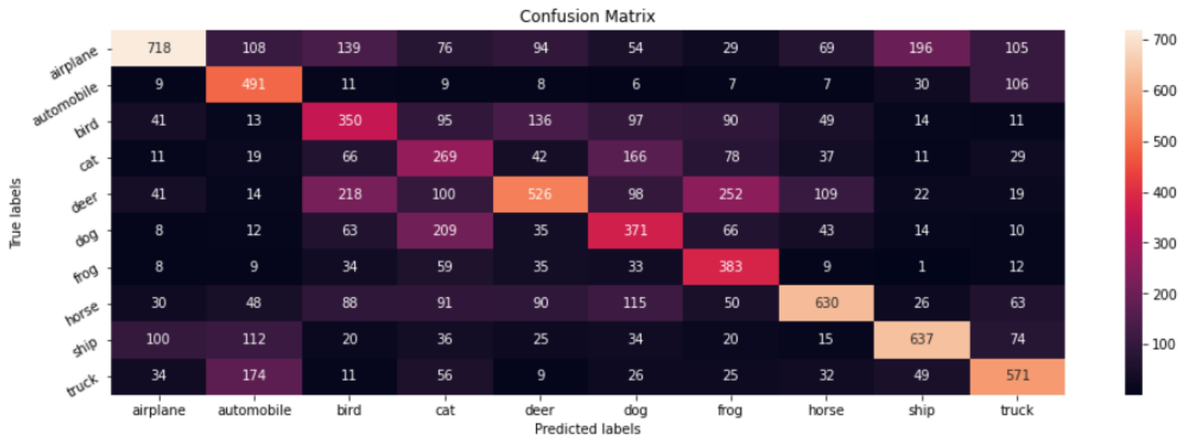
1. **Number of hidden layers** - this parameter represents the number of layers between the output and input layers. They perform non-linear transformation of the input which is useful when classes in a multiclass classification problem aren't linearly separable. We assume that this is the case for an image classification problem, thus hidden layers between 3-6 were explored. (Parameter value : 4 – excluding input layer).
2. **Number of nodes in each layer** – this parameter represents the number of "computational units" in each layer. While choosing a parameter value, as the size of the input is 3*32*32 = 3072, the number of nodes in the first layer was of the same order of magnitude at 1024. The rationale in not using a value much smaller than the input size was that certain features of the dataset may be lost if the first hidden layer has too few nodes. In each subsequent hidden layer however, the number of nodes was gradually lowered by a power of 2. . (Parameter value : [1024, 512, 256, 128, 64]).
3. **Activation function** – this parameter introduces an element of non-linearity to the system by deciding when a node activates. Here a Rectified Linear Unit (ReLU) was used for hidden layers as generally this has good performance. For the output layer a logarithmic SoftMax is used to transform the probability distribution of the predicted classes for each input to a single predicted class. (Activation function: ReLU)

4. **Optimizers** – this parameter determines the algorithm used to change the weights and biases of the neural network such that loss (error in prediction) is minimized. For this parameter both Stochastic Gradient Descent (SGD) and Adam were explored. (Optimizer: SGD)
5. **Learning Rate** - this parameter controls how weights/biases are adjusted with respect to the loss gradient. Initially parameter values for this function used the default value which was later adjusted to a higher/lower value based on model performance and time taken for the network to converge. (Loss Function: Cross Entropy Loss)

When training the model, mini batches were used so that the network didn't require the full set of inputs to pass before updating weights. By using batches, this updates the weights more frequently allowing the network to converge faster. The number of epochs to train the network was set by evaluating the accuracy and loss of the validation set. When the loss or accuracy of the validation set plateaus, or the loss of the training set is lower than or accuracy of the training set if higher the respective measure in the validation set, this indicates a sign of overfitting and a lack of the model's ability to generalize well to new datapoints. For this model, at around roughly 15 epochs, an accuracy 50.99% was obtained on the training set.

The same model on the test set achieved an accuracy of 49.46 % (comparable to that of the training set). While the model generalizes well, we note that accuracy of the baseline model leaves much room for improvement.

An additional metric that was observed to understand model performance is the confusion matrix. By inspecting the list of classes, we note that there's two broader categories of classes – vehicles and animals. By observing the confusion matrix, we notice that there's a lot of misclassifications made between each of the broader categories. For example, for vehicles, we note that an automobile is most frequently misclassified as a truck and a truck an automobile. For the second category, animals, we observe a similar patter with a cat being most frequently misclassified a dog and a dog a cat. This gives some insights into how the neural network is recognizing some features of each of the classes but can't identify the full set of features exclusive to a class. In the next section, we'll explore some techniques that were attempted to improve model performance.



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	718	108	139	76	94	54	29	69	196	105
automobile	9	491	11	9	8	6	7	7	30	106
bird	41	13	350	95	136	97	90	49	14	11
cat	11	19	66	269	42	166	78	37	11	29
deer	41	14	218	100	526	98	252	109	22	19
dog	8	12	63	209	35	371	66	43	14	10
frog	8	9	34	59	35	33	383	9	1	12
horse	30	48	88	91	90	115	50	630	26	63
ship	100	112	20	36	25	34	20	15	637	74
truck	34	174	11	56	9	26	25	32	49	571

Figure 1 : Confusion matrix of test set for baseline models

3.0 Improving model performance

In this section, we perform a thorough analysis of the techniques used to improve model performance.

3.1 Dropout

A pattern that was observed while training the neural network and testing it on a previously unseen dataset is that often the accuracy of the model against the train set was higher than that of the test set. This is a result of a generalization error known as overfitting.

To minimize the effects of this, dropout is a regularization method that randomly drops the output of several nodes in a given layer. The benefit of this technique is two-fold. One, it introduces more noise in the training data making the model more robust and improving its ability to generalize well to new datapoints. And two, by dropping a certain number of outputs/connections from each layer, this has the effect of the layer being treated like a layer of a different configuration. Therefore, each update made to a layer during training is made with a different view of the configured layer.

To examine the effects of dropout, a dropout layer was incorporated after each layer in the network of the baseline model. This layer, at random, zero elements of the input tensor with a probability of p. Although, through exploration of this parameter, the overall accuracy of the model could not be improved from that of the baseline model, it is interesting to note that we're able to see that for most cases the difference between testing and training accuracy is bounded between 1-2% which shows the effect of dropout in preventing the network from overfitting.

3.2 Convolution Neural Networks (CNN)

Another well-known deep learning technique that was also attempted was convolution neural networks. This type of network is typically used for image classification problems. Often when using images as inputs to neural networks, without any feature engineering, each pixel forms a feature meaning training a neural network with the full set of inputs becomes computationally expensive.

CNNs offer an efficient solution that reduces dimensionality of the input layers without loss of key features. These networks have three main components to them:

1. Convolution layers – the purpose of this layer is to extract features from the image and create a feature map. Here dimensionality is lowered by passing a filter that slides across the image multiplying its value with the value from the input. For each window a single value is obtained by summing the result of the multiplication creating an output of lower dimensionality than the input. Multiple filters can be used to detect many features. This is typically followed by an activation function to incorporate non-linearity.
2. Pooling layers – are used to reduce the size of input and control overfitting. Typically, max pooling is used. This technique partitions the input into different sections and the maximum value in each section is output to identify only the most important parts of the feature. This ensures the model wont overfit to a specific instance of the feature from the train set.
3. Fully connected neural network – this is the neural network that will train on the reduced set of features engineered by the convolution and pooling layers

By incorporating convolution and pooling layers in the network architecture, we immediately realize the benefit in terms of model performance and training time. The table below shows these metrics for the baseline model and the best performing CNN model(with 3 sets of convolution and pooling layers).

	ANN (Baseline) Model	**CNN Model – best model**
Model Test accuracy	49.46%	69.41%
Model Train accuracy	50.99%	78.29%
Time elapsed for training	2.531 min	1.491 min
Number of epochs	15	5

Table 1: Comparing model performance of ANNs vs. CNNs

From the Table 1 we observe an improvement in all metrics in the CNN model over the ANN model. Although, by noting the model train accuracy of the CNN model and comparing it to its test accuracy, it appears as though the model might be overfitting to the train data. On testing this by stopping training at 4 epochs, the test accuracy drops by nearly 3% to 66.36% .

One of the parameters that was adjusted while training CNNs were the number of convolution and pooling layers. The effect of having 2 to 4 convolutions layers were studied, the best performing of which had 3 convolution layers. This shows a higher number of convolution layers may not always yield higher model performance. As the best performing model had 3 convolution layers, any tuning of other model parameters was based on the same model.

Number of convolution layers	**Train accuracy**	**Test accuracy**
2	66.03%	64.97%
3	78.29%	69.41%
4	73.88%	68.41%

Table 2 : Effects of number of convolution layers on model performance

In order to study the effects of increasing non-linearity in the system, a dropout layer was introduced after each hidden layer in the ANN. A dropout rate of 10% was used, however this resulted in a drop of the model's accuracy against the test set at 66.23%. When introducing non-linearity in the feature extraction layers by having a ReLU layer between the convolution and pooling layer, this too had a negative impact on the model performance yielding an accuracy of 65.01%

Another parameter that was adjusted is the kernel size. This is a hyperparameter which defines the size of the convolving kernel used to create the feature map. On increasing the size of the kernel of the convolution layers, this once again resulted in a drop in model performance to 60.35%. We can interpret this as in using a larger window from which features are extracted, this could be introducing more such that the model can't identify the features distinctly.

4.0 Conclusion

In using CNNs to classify images from the CIFAR-10 dataset, we were able to achieve moderate model performance at 69.41%. By inspecting the confusion matrix of the test set of the CNN and ANN we note that the number of misclassifications is reduced dramatically, and the model can extract features that are specific to each class better than an ANN. To improve performance, it may be worth including the validation set in training so that model has more datapoints it can learn from. Another option is to strategically continue exploration of other hyper parameters until the model can achieve better performance.

5.0 References

[1] DeepAI. (2019, May 17). *Hidden layer*. DeepAI. Retrieved January 2, 2022, from https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning

[2] Brownlee, J. (2019, August 6). *How to configure the number of layers and nodes in a neural network*. Machine Learning Mastery. Retrieved January 2, 2022, from https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

[3] Kumar, S. (2020, June 9). *Overview of various optimizers in Neural Networks*. Medium. Retrieved January 2, 2022, from https://towardsdatascience.com/overview-of-various-optimizers-in-neural-networks-17c1be2df6d5

[4] Doshi, S. (2020, August 3). *Various optimization algorithms for training neural network*. Medium. Retrieved January 2, 2022, from https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

[5] Zulkifli, H. (2018, January 27). *Understanding learning rates and how it improves performance in Deep Learning*. Medium. Retrieved January 2, 2022, from https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10

[6] Brownlee, J. (2019, August 6). *A gentle introduction to dropout for Regularizing Deep Neural Networks*. Machine Learning Mastery. Retrieved January 2, 2022, from https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

[7] Bonner, A. (2019, June 1). *The complete beginner's guide to deep learning: Convolutional Neural Networks*. Medium. Retrieved January 2, 2022, from https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb

[8] Mishra, P. (2019, July 20). *Why are convolutional neural networks good for image classification?* Medium. Retrieved January 2, 2022, from https://medium.datadriveninvestor.com/why-are-convolutional-neural-networks-good-for-image-classification-146ec6e865e8

[9] Sorokina, K. (2019, February 26). *Image classification with Convolutional Neural Networks*. Medium. Retrieved January 2, 2022, from https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8

Code references:

[10] PyTorch. (n.d.). *Neural networks*. Neural Networks - PyTorch Tutorials documentation. Retrieved January 2, 2022, from https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

[11] Zivkovic, S. (2021, November 15). *#010 pytorch - artificial neural network with Perceptron on CIFAR10 using pytorch*. Master Data Science. Retrieved January 2, 2022, from https://datahacker.rs/009-pytorch-building-an-artificial-neural-network-with-perceprton-on-cifar10-using-pytorch/

[12] Datahacker.rs. (2021, February 10). *#014 pytorch - convolutional neural network on mnist in Pytorch*. Master Data Science. Retrieved January 2, 2022, from https://datahacker.rs/005-pytorch-convolutional-neural-network-on-mnist-in-pytorch/

[13] Hussain, S. (2021, June 1). *Cifar 10- CNN using pytorch* . Retrieved January 2, 2022, from https://www.kaggle.com/shadabhussain/cifar-10-cnn-using-pytorch

Percentage of code borrowed: 60%