COMP-551 Mini Project 2

Kaicheng Wu
kaicheng.wu@mail.mcgill.ca
Mathieu-Joseph Magri
mathieu-joseph.magri@mail.mcgill.ca
Mohammad Sami Nur Islam
mohammad.sami.islam@mail.mcgill.ca

School of Computer Science
McGill University
Canada
March 9th 2023

1 Abstract

The goal of this second mini-project was to implement a multilayer perceptron from scratch. This multilayer perception was then used to categorize image data from the CIFAR-10 dataset collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Further goals of this assignment were to implement a basic neural network and its training algorithm from the ground up as well as gain experience on how to make important decisions on how to train the developed models. Convolutional neural networks (CNN) were also experimented on, including pre-trained versions of them to perform image classification as well. The most important takeaway from the experiments concerning the MLP was that as we increased the number of hidden layers the accuracy increased. Also, the *tanh* activation was best able to capture the non-linearity of the data. Further analysis of these most important findings will be discussed throughout the report.

2 Introduction

From a global perspective, the main goal of this mini-project was to create and implement a multilayer perceptron from scratch, and use it to classify data garnered from images. Important decisions needed to be made during the training of these models. Experiments with convolutional neural networks and pre-trained convolutional neural networks to accomplish image classification were also done.

To be more precise, 3 separate tasks were done in order to complete this mini-project. The first task consisted of acquiring the data. The dataset that was used was the CIFAR-10 dataset with the default train and test partitions: sixty thousand 32 by 32 colour images in 10 classes, with 6000 images per class as well as 50 000 training images and 10 000 test images. With this dataset, the developed neural nets will be able to be trained in such a way as to learn the feature extractor together with the feature classifier. The manipulations that were applied to the dataset will be discussed in the *Dataset* section.

Task 2 involved implementing a multilayer perceptron (MLP) to classify image data. The implementation was done from scratch with only Numpy as a used package and the lecture slides and code used as inspiration. The implementation includes back-propagation and mini-batch gradient descent. From a coding perspective, the MLP was implemented as a class, where its constructor takes as input an activation function, a number of hidden layers, and a number of units for the hidden layers. It also initializes weights and biases and other important properties with pre-chosen initializers. The MLP class has three defined functions: fit, predict, and acc. The "fit" function uses training data and hyperparameters to train the model. The "predict" function takes a set of inputs and outputs predictions, while acc takes in target labels and true labels to then output an accuracy score of the trained model. Python libraries were also used to tune the hyper-parameters.

Task 3 involved experimenting with the trained model that was built from scratch as well as comparing its performance with a CNN and pre-trained models. The consequences of important decisions such as tuning hyperparameter values while training neural networks were also studied. More on the results of the experiments in the *Results* section.

Finally, CIFAR-10 is an important dataset from the university of Toronto which consists of many images of different classes used in many different studies and papers. Indeed, the project page for the cuda-convnet ran a CNN where training is done using the back-propagation algorithm in which they obtained a 11% error rate on the dataset. Furthermore, in a paper done by Microsoft Research, a 3.57% error rate was obtained using residual networks (He et al.) while a success rate of about 90% was obtained in Beaujuge's article where a pre-trained CNN was used (Beaujuge). There will be more on relevant papers later.

The most important takeaway from the experiments concerning the MLP was that as we increased the number of hidden layers, the accuracy increased. Also, the *tanh* activation was best able to capture the non-linearity of the data.

3 Datasets

The dataset that we are using is the CIFAR-10 dataset collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton for the University of Toronto. It is a labelled subset of the 80 million tiny images dataset. CIFAR-10 consists of 60 thousand 32 by 32 colour images divided into 10 classes, with 6 000 images per class. There are

10 classes that compose the dataset: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. All images can only be a part of one single class given that all classes are mutually exclusive. The dataset is composed of 10 000 test images and 50 000 training images. Every 10 000 images are classified as a batch, which leave us with 5 batches of training images and 1 batch of test images. The testing batch contains an equal distribution of randomly selected images from each class (1000 per class). The training batches contain all remaining images, where some may contain a higher proportion of images from a certain class than other. When combining all training batches, 5000 images from each and every class are present. Exploratory analysis was not really needed considering that all the information on how the data was obtained and divided was noted on the CIFAR-10 source page. However, for Task 1, the training and test sets were normalized. Finally, in terms of ethical concerns with the datasets, there seems to be none, as everything is available to the public.

4 Results

4.1 Task 3.1 - 3.4

Note that here we decided to only use one out of the five different training parts for the MLP because it was quicker to train, and the maximum of only 256 neurons per layer meant that accuracy became lower while using all 50,000 images to train. Further analysis of the results are done in the *Discussion and Conclusions* section of the report. In this subsection so as to not be redundant, the resulting graphs and a results table are shown. More information can be found in the MLP notebook.

Figure 1: Results For Experiments 3.1 to 3.4

	task 3	activation_function	hidden_layers	first_layer_units	second_layer_units	regularization	regularization_parameter	accuracies (percent)
0	1.1	NA	0	0	0	NA	0.0	10.00
1	1.2	ReLU	1	256	0	NA	0.0	13.98
2	1.3	ReLU	2	256	256	NA	0.0	16.03
3	2.1	tanh	2	256	256	NA	0.0	11.33
4	2.2	Leaky-ReLu	2	256	256	NA	0.0	20.10
5	3.1	ReLU	2	256	256	Lasso	0.1	14.88
6	3.2	ReLU	2	256	256	Ridge	0.1	16.33
7	4.0	ReLU	2	256	256	N	0.0	10.00
8	5.0	tanh	2	1000	1000	N	0.0	25.02

Figure 2: Results For Tanh Activation Function

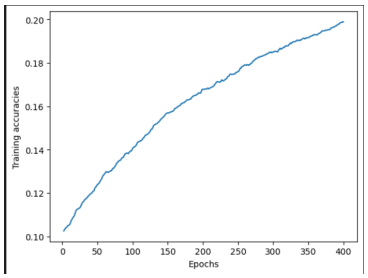


Figure 3: Results For ReLU Activation Function

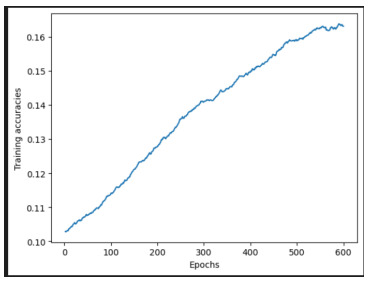
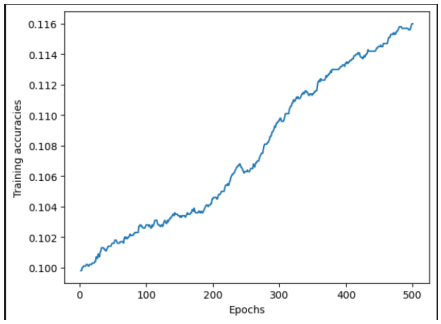


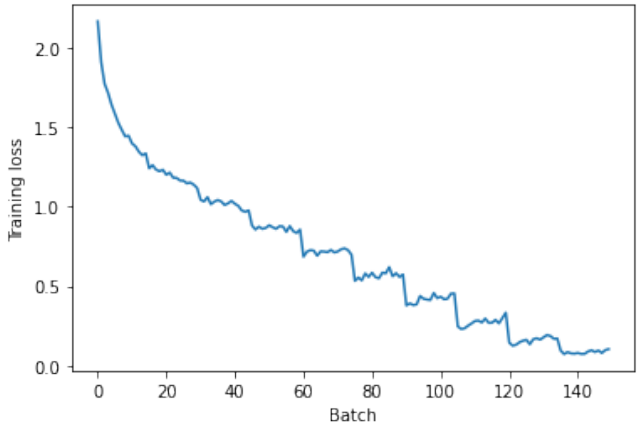
Figure 4: Results For Leaky ReLU Activation Function



4.2 Task 3.5

For this task, we developed a convolutional neural network (CNN) with 2 convolutional and 2 fully connected layers using pyTorch. The CNN reports an accuracy of roughly 67%, which is an improvement from our result with the MLP. As we can observe from figure 1, the training loss steadily decreases as more batches are fed to the model. We can also observe some steep descents on the figure, which are associated with the start of new epochs.

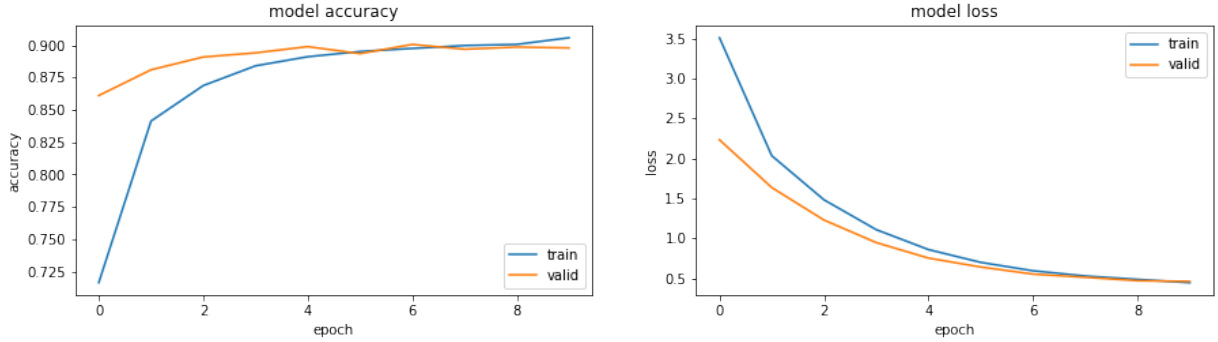
Figure 5: The training loss of the CNN as more batches are fed to the model



4.3 Task 3.6

For 3.6, 4 different pre-trained datasets were loaded. The first dataset is a DenseNet201 pre-trained model from the Keras API. For this dataset, the convolutional layers were frozen, the fully connected ones were removed and a number of fully connected layers were added. The added layers were also then trained on the CIFAR-10 dataset. The code and implementation of this DenseNet201 model was heavily inspired and sourced from Beaujuge's article which can be found at the end of the report.

Figure 6: Accuracy and Loss Results of the DenseNet201 pre-trained model



As seen from the graph above, the results for the pre-trained model are quite good. Furthermore, the model stops running after 10 epochs, given that after 10 epochs, the model starts to overfit the values, and the model automatically stops, with an overall accuracy of 89.75%.

Furthermore, experimentation with three other models were done to compare results. The second model is also from Beaujuge's article, the third model is from Dabydeen's article, and the fourth model is from PyTorch's tutorials. All code was heavily sources and inspired from their respective sources which can be found at the end of the report. For more information on the models and their performance, the A2_3-6 Colab file is available with this report. The second model, although it performs quite fast, heavily underperforms with its accuracy (about 10%). The third model runs as fast as the first model (about a 50 minute runtime) and gets an accuracy of about 84.88%. Finally, the fourth model runs quite fast and for its speed, gets a quite good accuracy of 80.86% as well. Loss rates are also included in the Colab for the many different models.

4.4 Task 3.7

More plots can be found in the notebooks including the justification for different hyperparameters.

5 Discussion and Conclusions

We observed that as the depth of the MLP increased, so did the accuracy. This could be because as more layers are added to the model, it captures increasingly complex and abstract relationships between the input and output variables. This is because each layer in the MLP can learn to represent features at a higher level of abstraction than the previous layer. This hierarchical representation allows the MLP to model more complex functions, which can lead to higher accuracy.

In addition to network depth, the choice of activation function can also affect the accuracy of the MLP model. The nonlinearity of the activation function is what allows the MLP to model non-linear relationships between the input and output variables.

We observed that the tanh activation function performed better than the rectified linear unit (ReLU) and leaky ReLU functions. This could be because it has a steeper gradient near the origin, allowing for faster convergence during training. Additionally, the tanh function is bounded between -1 and 1, which can prevent exploding gradients and improve the stability of the network.

From our experiments, we found that L2 regularization was the most effective in improving the generalization performance of MLP. L2 regularization achieves this by shrinking the weights toward zero, thereby reducing the complexity of the model. As the regularization parameter increases, the weights are shrunk towards zero, leading to a simpler model and better generalization performance.

In contrast, L1 regularization has a sparsity-inducing effect on the weights, leading to a smaller number of non-zero weights in the model. The sparsity-inducing effect can lead to underfitting. In our investigation, we observed that L1 regularization only improved accuracy up to a certain point and did not improve accuracy beyond that point. This observation may be due to the fact that the sparsity-inducing effect of L1 regularization can lead to underfitting when the regularization parameter is set too high.

When the input data is not normalized, the range of the input features can lead to some neurons in the first layer having very large outputs, which in turn leads to very large gradients and the exploding gradient effect. This is consistent with our observations. Also, to prevent the exploding behaviour we used Xavier Initialization to initialize the weights such that the variance of the activations are the same across every layer. This constant variance helps prevent the gradient from exploding or vanishing.

In conclusion, we developed a MLP from scratch, a CNN with existing libraries and compared them to 4 pre-trained datasets we found. We obtained the highest accuracy with the pre-trained datasets. To be precise, the accuracy of the pre-trained models we found is 89.75%. Compared to the best MLP in part 1, it is vastly superior. Compared to the regular CNN in part 5, it is also better by about 12%. When comparing training time, the pre-trained model takes about 49 minutes, while the best MLP in part 1 takes almost an hour, and the CNN in part 5 takes 80 minutes. As we can see, the pre-trained model takes a decent amount of time to train but has a higher accuracy value than the two other types of models.

After running the CNN we developed, we made a few interesting discoveries. First, a decrease in the loss can be observed as the number of epochs grows, which is a good sign as it indicates that the model is improving with

additional training. Second, as the training progresses, the rate of decrease in loss slows down, indicating that the model is converging to a solution. Finally, the loss of the model goes through a very steep descent at the beginning of the training, seemingly suggest that the model is initialized at a very sub-optimal state but is rapidly improving with more training data.

For the pre-traned models, we discovered through experiments and multiple different runs, that the number of convolutional layers plays a major role in the performance of the model. It was possible to notice that reducing the number of dense layers can be a factor in increasing testing and training accuracy, thus also making it a factor for reducing losses. Furthermore, it is possible to save training time the more layers we freeze in a pre-trained model. On the other hand, adding more layers can increase computational costs and increase the risk the model has of overfitting. All in all, the amount of layers we added took into consideration all the previously mentioned factors and we fill that the pre-trained model we've implemented is well balanced.

In the future, it would be interesting to run even more experiments on the models with varying hyperparameters and see how they perform. For example, trying more than two layers in the MLP and conducting a Grid Search to tune the hyperparameters of the different models could be a fruitful experiment.

Moreover, it would definitely be helpful to conduct our experiments on specialized hardwares, such as high-performance GPUs, to minimize the time needed to train each model.

6 Statement of Contributions

The workload was distributed evenly across all 3 team members. Task 2 was handled by Mohammad Sami Nur Islam, while tasks 1 and 3 were handled by Kaicheng Wu, Mathieu-Joseph Magri, and Mohammad Sami Nur Islam. Finally, all team members also contributed to the report.

7 Bibliography

Beaujuge, Pierre. "Classifying Images from the CIFAR10 Dataset with Pre-Trained CNNs Using Transfer Learning | by Pierre Beaujuge | Medium." Medium, Medium, 4 July 2020, https://medium.com/@pierre.beaujuge/classifying-images-from-the-cifar10-dataset-with-pre-trained-cnns-using-transfer-learning-9348f6d878a8.

becauseofAI. "3.2.2 ResNet_Cifar10 - PyTorch Tutorial." PyTorch Tutorial, 2019, https://pytorch-tutorial.readthedocs.

Dabydeen, Andrew. "Transfer Learning Using ResNet50 and CIFAR-10 | by Andrew Dabydeen | Medium." Medium, Medium, 7 May 2019, https://medium.com/@andrew.dabydeen/transfer-learning-using-resnet50-and-cifar-10-6242ed4b4245.

"Google Code Archive - Long-Term Storage for Google Code Project Hosting." Google Code, Google Code, 18 July 2014, https://code.google.com/archive/p/cuda-convnet/.

Hamdi, Yasmine. "Transfer Learning Experiment on CIFAR 10 Dataset | by Yasmine Hamdi | Medium." Medium, Medium, 25 Sept. 2020, https://medium.com/@yasminehamdi/transfer-learning-experiment-on-cifar-10-dataset-41aa75f75989.

He, Kaiming, et al. "Deep Residual Learning for Image Recognition." Microsoft Research, Microsoft Research, Dec. 2015, https://arxiv.org/pdf/1512.03385.pdf.

Krizhevsky, Alex. "Learning Multiple Layers of Features from Tiny Images." Computer Science, University of Toronto, 8 Apr. 2009, http://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf.