Nifty Neural Networks!

Caleb French
School of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: cmfrench@wpi.edu

Sukriti Kushwaha
School of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: skushwaha@wpi.edu

Chad Nguyen
School of Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: ctnguyen@wpi.edu

***Abstract*—This report contains the results of creating a neural network by creating a linear, ReLU, and Softmax layer. The purpose of creating the neural network is to use it to identify 10 different categories of photos. The layers are then used to create a MLP and train the network to have a sufficient accuracy at labeling each photo.**

I. Introduction

The purpose of this lab was to categorize 32x32 pixel images into 10 different categories using neural networks. The neural networks were created by the following tasks:

1) Create linear, ReLU, and Softmax layers for the neural networks.
2) Train the Multi Layer Perceptron (MLP) Network for image categorizing.
3) Train the Convolutional Neural Networks (CNN) for image categorizing.

MLP Networks are a type of neural network based on forward and backward propagation to predict and train the weights of the network. The other important aspect of MLP Networks are the non-linear layer functions. Non-linear functions are integral to creating a MLP Network since consecutive linear layers could be simplified to a single layer meaning the complexity of the neural network is trivial.

CNNs are a type of neural network which specialize in image classification. CNNs are comprised of various node layers and are a type of deep learning algorithm. The main difference between the CNN and the MLP Network is the utilization of a convolutional layer in the CNN. The convolutional layer uses a kernel in which it filters the input image with various focuses.

A. Creating the Layers

Each layer is important to create our custom MLP model. The three custom layers used are a linear layer, a ReLU layer, and a Softmax layer. Each layer enacts a different filtering system on the raw pixel data in order to create patterns to look for when trying to predict the label of the images. Each layer needs to be able to handle a forward and backward pass. On the forward pass, the layers generate its predicted values and on the backward pass, it updates its weights in order to create a more accurate model.

1) Linear Layer: The equation for forward propagation is given by the following formula:

$$y = A^T x + b$$

where A represents the weights assigned to each neuron.

To update the weights, bias, and input during the back propagation step, it is necessary to calculate the gradient of the loss function with respect to each of these variables. Using partial derivatives, the following equations were derived:

$$\text{Gradient of Weights: } \frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial z} \cdot \mathbf{x}^T$$

$$\text{Gradient of Bias: } \frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \frac{\partial \mathcal{L}}{\partial z}$$

$$\text{Gradient of Input: } \frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \mathbf{W}^T \cdot \frac{\partial \mathcal{L}}{\partial z}$$

2) ReLU Layer: The ReLU function is defined as:

$$ReLU(x) = \max(0, x)$$

For back-propagation of the ReLU layer, the gradient of the loss with respect to the input layer is defined as follows:

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial \text{ReLU}(\mathbf{x})}{\partial \mathbf{x}}$$

where the partial derivative of the ReLU function was derived as follows:

$$\frac{\partial \text{ReLU}(\mathbf{x})}{\partial \mathbf{x}} = \begin{cases} 1 & \text{if } \mathbf{x} > 0 \\ 0 & \text{if } \mathbf{x} \leq 0 \end{cases}$$

3) Softmax Layer: The given equation for the Softmax function is as follows:

$$y_i = \frac{e^{x_i}}{\sum_k e^{x_k}}$$

This function was implemented into the forward propagation for the Softmax layer of the MLP Network. For back-propagation, the gradient of the loss with respect to the input is as follows:

$$\frac{\partial \mathcal{L}}{\partial z_j} = \sum_i \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial z_j}$$

The Jacobian matrix represents the partial derivative of each output of the Softmax function with respect to the input. For the Softmax function applied to a vector z, the Jacobian is defined as follows:

$$\frac{\partial z_j}{\partial \text{Softmax}(z_i)}$$

where the gradient of the Softmax output with respect to the input for elements on the diagonal of the Jacobian is given by the following equation:

$$\frac{\partial \text{Softmax}(z_i)}{\partial z_i} = \text{Softmax}(z_i) \cdot (1 - \text{Softmax}(z_i))$$

and the equation for elements off the diagonal is:

$$\frac{\partial \text{Softmax}(z_i)}{\partial z_j} = -\text{Softmax}(z_i) \cdot \text{Softmax}(z_j)$$

B. Training the MLP Network

With working and accurate layers now created, the next step is to utilize them in a MLP Network and train it to be at least 40% accurate when predicting which category each image is in. The optimizer used to find the local minima in loss value was the Adam optimizer. The Adam optimizer is an algorithm which minimizes the loss function when training a neural network. The main parameter that needed to be tweaked over various testing results is the learning rate. Learning rate affects how large of a change the algorithm will make each epoch. There is an importance when tuning the learning rate since too low of a learning rate and the algorithm will more and more epochs in order to find the local minimum but too high of a learning rate and the algorithm will skip over the local minimum and never find it. Through a series of tests the sufficient learning rate found for 30 epochs was 0.0001.

1) Creating Confusion Matrix: A Confusion matrix is a visual representation of how accurate the model was at predicting the correct label for each photo.

It is a 2d matrix comprised of the actual label and the predicted label. Each row of the confusion matrix represents how many test samples are that label. Each column of the confusion matrix represents how many test samples were predicted to be that label. The confusion matrices are created at the same time of the evaluate step in the training pipeline in which the network checks how accurate it was at predicting the correct label for each image.

The confusion matrix itself was created utilizing *matplotlib* to visualize the confusion matrix and *sklearn* to create the confusion matrix based off of the corresponding arrays. To utilize the sklearn built-in function of creating a confusion matrix, three arrays are required as inputs: an array of the class names, an array of all of the predicted classes for each image, and an array of all of the actual classes for each image. Creating the array for all of the class names was simple since there is a preset 10 classes that we are identifying our images to. Creating the arrays of predicted and actual classes is trickier since they are generated in each epoch. The first step was to consolidate all of the batches of data into one large array of data for each. This was done via a for-loop to go through each incoming batch of images processed and appending the relevant data to its corresponding array. The second caveat is that the data returned from the neural network is a *0-9* integer, where each number represents a different class. These integers were then mapped to the corresponding label and the data was passed into sklearn to create the confusion matrix.

C. Implementing CNN With Pytorch Layers

To implement our own CNN to test, we utilized the inbuilt pytorch layers. CNN uses a convolutional layer to process the image before passing it through the model. The CNN created takes one input and creates up to 64 different filtered channels which get processed by the model. This filtering step with various kernels is what creates higher performance for the CNN model when compared to the MLP Network. Our model was comprised of a linear layer, then ReLU layer, then linear layer, then ReLU layer, then linear layer, and finally a Softmax layer.

II. Conclusion

When training our neural networks, the number of epochs have an impact on our accuracy. This can be best visualized with a training loss curve.

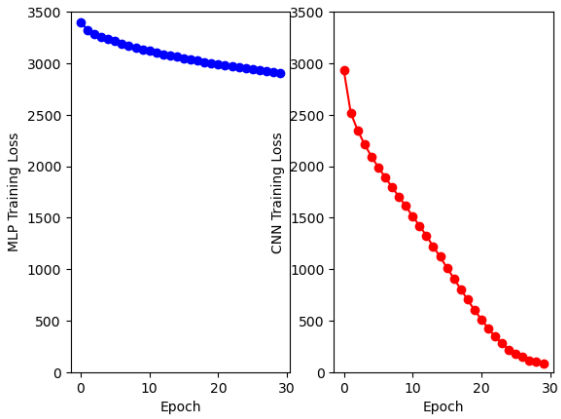

Fig. 1. The Loss Curve of MLP Network and CNN

As seen in Figure 1, the smallest loss value, or highest accuracy reading for MLP was 2904.78. Similarly, as seen in Figure 1, the smallest loss value, or highest accuracy for CNN reading was 80.44.

After finding the most optimized epoch of weights and biases, that set was saved to try against the test data. Confusion matrices were generated in order to visualize the accuracy of our neural networks.

A confusion matrix would have its accurate predictions be in the square in which both the actual and the prediction labels cross paths with each other. Since each corresponding row and column are the same index, this means that the crossovers will always appear in a diagonal line starting from the top-left to the bottom-right. This means that a highly accurate confusion matrix will appear to be a diagonal line. As seen in Figures 2 and 3, our neural networks were able to generate accurate test values based on their respective confusion matrices since most of the data corresponds to the correct label. The only caveat found is in Figure 3, in which the neural network never

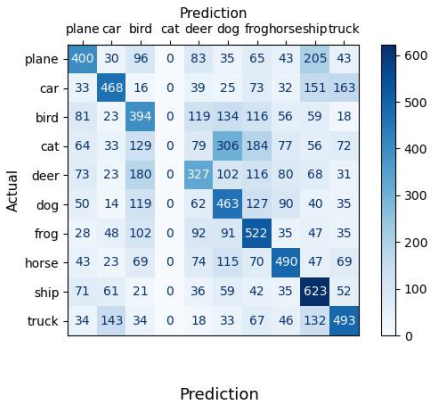

Fig. 2. Confusion matrix of MLP Network

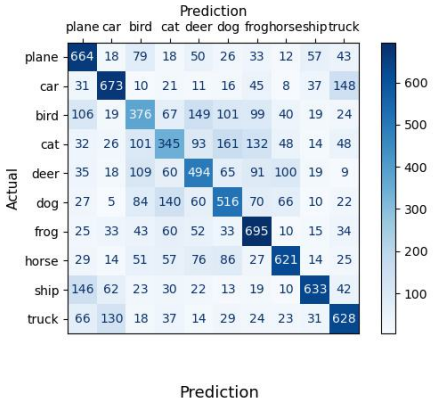

Fig. 3. Confusion matrix of CNN Network

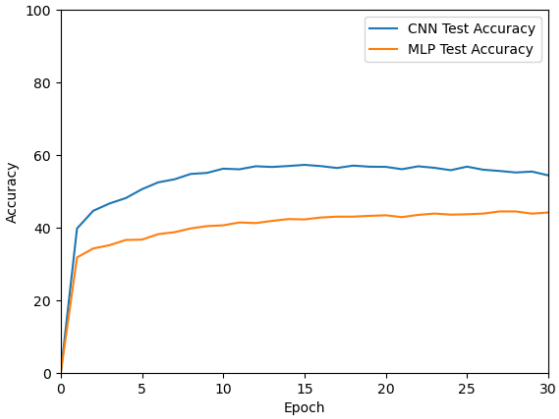

Fig. 4. Accuracy Comparison between CNN and MLP

predicted that the photos were cats, no matter the actual image.

Between the two neural networks, it is clear that CNN was more accurate than MLP. As seen in Figure 4, while MLP was able to generate a 44.15% accuracy on the test data, CNN was able to generate a 57.28% accuracy. Interestingly, our highest accuracy for our CNN came from our 18th epoch. After that, the loss function kept decreasing, but our accuracy also began decreasing. This is because the model was starting to overfit to our training data. Similarly, this conclusion can be proven in the two confusion matrices generated. As seen in Figure 3, there is a higher amount of accurate predictions for CNN when compared to the MLP confusion matrix. Overall, this shows that the CNN was a more accurate neural network than the MLP Network. Simply put, the CNN was able to predict which image was a cat when the MLP was not able to do so.

ACKNOWLEDGMENT

The authors would like to thank Professor Nitin Sanket for teaching us all of the necessary material to complete such a project and Manoj Velmurugan for grading our project and providing assistance in solving the mathematical equations necessary for such a project.