CS 260 Homework 3

Wenlong Xiong (204407085)

March 10, 2019

1 Introduction

In this assignment, I used PyTorch to implement a Multilayer Perceptron (MLP) and a Convolutional Neural Network (CNN), and used them to perform multi-class classification on images from the CIFAR-10 dataset.

2 Metrics

We define the average training loss in each epoch as the following:

$$\frac{1}{B}\sum_{b=1}^{B}\sum_{d=1}^{D_b}\frac{1}{D_b}loss(\text{labels}_{b,d}, \text{model}_b(\text{data}_{b,d}))$$

B is the number of batches, D_b is the number of data points in batch b, and $\text{model}_b(\text{data}_{b,d})$ is the prediction by the model after batch b on the current batch of data.

3 Network Architecture and Training

I used the Adam optimizer for both these models, with the default parameters (learning rate = 10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$). In each model, I used the rectified linear unit (ReLU) nonlinearity after each layer. I trained each model for 100 epochs, and recorded the average training loss for each of my models over 10 epochs, as well as the final test accuracy. Each model's output is a 1D vector of size 10 (each element of which represents one of the 10 possible classes). The loss was calculated using categorical cross entropy between the true labels and the most likely class from the outputs. For testing, I took the output class with the highest probability as the predicted class.

The MLP architecture consisted of 7 dense (fully connected) layers. The layers had 512, 256, 128, 64, 32, 32, and 10 perceptron units, respectively. After each of the layers, I had a ReLU nonlinearity layer.

The CNN architecture consisted of 4 2D convolution layers, followed by 3 dense (fully connected) layers. The convolutional layers were as follows: 1) 3x3 kernel, 1 padding, 1

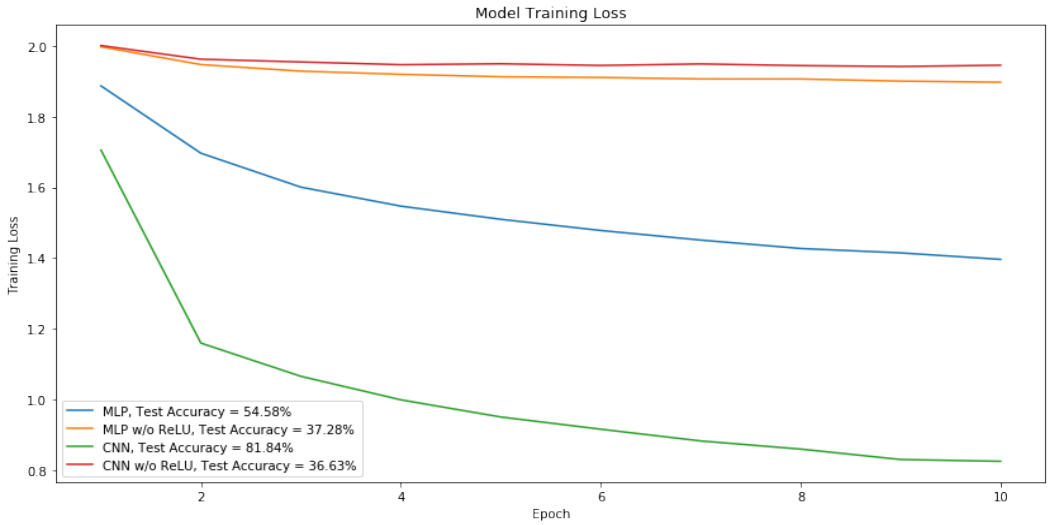
stride, 3 input channels, 64 output channels, 2) 2x2 kernel, 0 padding, 2 stride, 64 input channels, 64 output channels, 3) 3x3 kernel, 1 padding, 1 stride, 64 input channels, 64 output channels, 4) 2x2 kernel, 0 padding, 2 stride, 64 input channels, 64 output channels. After the convolutional layers, I reshaped the 3D output of the final convolutional layer (64 x 8 x 8) to a 1D tensor of size (4096), to use as the input to the first dense layer. The dense layers had 1024, 512, and 10 perceptron units, respectively. After each of the layers, I had a ReLU nonlinearity layer. In addition, during training, I had a dropout layer after the 1024 and 256 unit dense layers (set after the nonlinearity), which zeroed outputs randomly with a probability of p=0.5. This was only used during training time, and removed during test time.

In a later part of this homework, I tested out a MLP without non-linear activations as well as a CNN without non-linear activations; I used the same architectures but simply removed the ReLU nonlinearities.

4 Results

To compare the classification accuracy of the MLP and CNN architectures with and without nonlinearities, I ran these two models with the Adam optimizer and cross-entropy loss for 100 epochs. The training loss is recorded below for the 4 combinations, along with the final model accuracy (at 100 epochs) when applied to the test set. The exception is that the CNN's final model accuracy was obtained at 90 epochs (the test accuracy at 100 epochs was lower). I plotted the training loss per epoch for each of these models in the figures below. From this, we can see that the CNN obtains a much higher test accuracy than the MLP, and given the same loss function, converges much faster to a lower training loss. This makes sense, as each convolutional filter is convolved with the entire previous layer. Therefore, for a single filter, the weights for the convolution calculations at different locations are shared. This means that if there are similar features in multiple different locations, the CNN does not need to relearn weights that recognize these identical but spatially separated features, but can just learn a single filter instead. Because of this, there are generally fewer weights per layer to learn in a CNN than a MLP, which allows a CNN to converge faster.

Epoch	MLP	MLP w/o ReLU	CNN	CNN w/o ReLU
1	1.88734	1.99805	1.70560	2.00152
2	1.69665	1.94767	1.15904	1.96289
3	1.60061	1.92912	1.06496	1.95506
4	1.54682	1.92010	0.99840	1.94734
5	1.50955	1.91321	0.94976	1.94991
6	1.47790	1.91110	0.91520	1.94522
7	1.45075	1.90699	0.88192	1.94947
8	1.42686	1.90685	0.85888	1.94479
9	1.41448	1.90094	0.82944	1.94229
10	1.39606	1.89771	0.82432	1.94586
Test Acc.	54.58%	37.28%	84.41%	36.63%



We can also see that for a given model, training the same architecture without the nonlinearities does not really decrease the loss. The models without nonlinearities quickly converge within the first few epochs and do not decrease much afterwards. In comparison, training the models with nonlinarities continue to decrease the loss from epoch to epoch. The models without nonlinearities obtain a much lower testing accuracy and also a higher training error than their counterparts that include the nonlinear activation functions. The reason behind this is that nonlinearities allow a neural network to become more expressive. We can model each layer in a neural network as a linear function, and because a composition of linear functions is also a linear function, if there were no nonlinearities after each layer, a multi-layer neural network would still only model a linear function (it would only be as expressive as a single layer network). However, by adding nonlinearities after each layer, the neural network becomes able to approximate more complex nonlinear functions, making it more expressive and able to better approximate the true multi-class classification boundary for CIFAR-10.