Final Project

Aniqa Rahim

Abstract

For my project, I chose the third option of the Type A projects. The model I experimented with is the original LeNet-5 CNN architecture with the MNIST dataset for testing and training. I had to test 4 different configurations of the LeNet-5 architecture, with the addition of dedicated layers for regularization (batch normalization or dropout) and evaluate which one of these configurations performs the best overall. I evaluated the performance of dropout on fully-connected layers and batch normalization on convolutional layers. From these experiments, I can conclude that the LeNet-5 model configuration with the addition of convolutional layers with batch normalization and no dropout is the best performing network, despite not having the shortest execution time. It achieves 99% test accuracy on the MNIST test set.

Introduction

Deep learning requires the use of deep neural networks which have more than 2 hidden layers. This is why you can turn any neural network into a deep neural network by adding more layers to the network architecture. For these specific experiments, we are designing a deep Convolutional Neural Network. Convolutional Neural Networks or CNNs are the most commonly used type of architecture used for image recognition. One of the many applications of deep CNNs include recognizing patterns such as the task of isolated handwriting recognition [1]. Some downsides of deep CNNs are how they use up a lot of computational resources when training and generally take a long time to run, especially if they are using a large dataset and/or multi-layered model.

For the purposes of this project, I tested a small convolutional neural network (CNN) on a small dataset of images on my personal laptop. The CNN utilized was the multi-layered LeNet-5 CNN architecture. The dataset is the entire MNIST database consisting of images of handwriting samples. The MNIST training set has 60,000 sample grayscale images of size 28x28 pixels and the test set contains 10,000 sample grayscale images of size 28x28 pixels. The goal of these experiments was to test the performance of LeNet-5 in four different architectural configurations. Each configuration uses a number of different regularization methods. The first configuration is the addition of dropout layers to fully connected layers and the addition of convolutional layers with batch normalization. The second configuration is the addition of dropout layers to fully connected layers and without batch normalization on the convolutional layers. The third configuration is fully connected layers without dropout and with batch normalization on the convolutional layers. The fourth configuration is the base LeNet-5 with fully connected layers without dropout layers and convolutional layers without batch normalization. The network trains on the MNIST training set. Then after the testing phase, the program returns the test accuracy, average loss, and total training and testing execution time of

the network. From this data, I can determine which LeNet-5 architectural configuration performs the best out of the four configurations.

Related Work

There has been research done in improving the architecture of LeNet-5 for the purposes of improving its performance and/or runtime for image recognition tasks. In a recent conference paper submitted to the EITCE 2020 [7], the authors improve the structure of the LeNet-5 architecture by replacing the full connection algorithm with the global average pooling algorithm. The average pooling algorithm computes the average value of all pixels in the sampling area as the pixel value. The authors also increase the number of convolution kernels and reduce the amount of subsampling layers to an optimal value. All of this significantly reduces the number of network training parameters to 21408 or 34.8% of the original amount. The authors find that the network's recognition accuracy for the MNIST handwritten Arabic numeral data set is 99.3%.

Improvements in training can help the LeNet-5 architecture perform better in other real world applications such as traffic sign recognition. In paper [8] which was submitted to ICVIP 2019, the authors increase the number of convolution kernels in the C1 and C3 layers in LeNet-5 while simultaneously reducing the size of the convolution kernel in C3 layer. They also replace the tanh(x) activation function with the better RELU function and replace the average pooling layers with max pooling layers. They also add a support vector machine (SVM) to the output layer in order to combine the benefits of an SVM and a CNN to shorten the execution time of the network. These structural changes to the model architecture showed a higher ecognition accuracy of 98.12% over the classical LeNet-5 accuracy of 93.73% and a shorter identification time of 0.154s versus 0.193s for traffic signs in the German Traffic Sign Recognition Benchmark (GTSRB).

Data Description

I used the entire MNIST database [2] for both testing and training. MNIST stands for "Modified National Institute of Standards and Technology". The original NIST database took their training set images from handwriting samples of US Census Bureau employees and the test set images from American high school students. This makes drawing conclusions from their results more difficult. MNIST improves on the original NIST database by taking images of handwriting samples from 250 different writers for the training set and then another separate group of 250 writers for the test set. This keeps the data hygienic. The MNIST images are also more clear and easier to recognize than NIST's. It is a very popular dataset of handwritten numerical digits used for performing character recognition tasks in image recognition. Individually sampled images of the digits 0-9 are shown in multiple different handwriting styles.

The MNIST training set has 60,000 sample images and the test set contains 10,000 sample images. The images are all 28x28 pixels in size and colored in grayscale.

Method Description

I used a combination of the lecture 10 slides on CNN architecture, the lecture 11 with the coding examples of how to build a neural network, and the original LeNet architecture paper [1] in order to design the LeNet-5 CNN model for this project. Since we had already designed a modified LeNet-5 architecture for our fourth homework assignment, it was very easy to adjust the code to create the original LeNet-5 CNN architecture. I just replaced layers and changed layer parameters when needed. I also changed the uploaded dataset from CIFAR-10 to the MNIST dataset. The transformation for tensors was taken from previous homework 3 where we used the MNIST dataset with our custom neural network design.

LeNet-5 will perform isolated character recognition on the MNIST dataset. The network will first train on the 60,000 sample training MNIST dataset using gradient descent. It learns by extracting features from this data so that it can perform well in the testing phase. This phase uses the 10,000 samples in the MNIST test set. This is where the network will attempt to accurately identify the handwritten characters in the MNIST data. The console will log the output for each training epoch and give statistics for the overall execution time, test accuracy, and average loss for the test set. I have set my specific model to output these results to .txt files.

Model Description

I modeled my project after the original LeNet-5 CNN architecture from its associated paper [1]. The LeNet-5 architecture is a multi-layered convolutional neural network. Below is the architecture code.

```
        self.convnet = nn.Sequential(OrderedDict([
                    ('c1', nn.Conv2d(1,6,kernel_size=(5,5), padding=2)),
                    ('batch1', nn.BatchNorm2d(6)),                      ##
BATCH NORMALIZATION LAYER
                    ('tanh1', nn.Tanh()),
                    ('s2', nn.AvgPool2d(kernel_size=(2,2), stride=2)),
                    ('c3',nn.Conv2d(6, 16, kernel_size=(5,5))),
                    ('batch2', nn.BatchNorm2d(16)),                      ##
BATCH NORMALIZATION LAYER
                    ('tanh3', nn.Tanh()),
                    ('s4', nn.AvgPool2d(kernel_size=(2,2), stride=2)),
                    ('c5',nn.Conv2d(16, 120, kernel_size=(5,5))),
                ]))
```

```
            self.fc = nn.Sequential(OrderedDict([
                ('f6', nn.Linear(120,84)),
                ('tanh6', nn.Tanh()),
                ('drop1', nn.Dropout(p=0.5)),
## DROPOUT LAYER
                ('f7', nn.Linear(84,10)),
                ('sig7',nn.LogSoftmax(dim=-1))
            ]))
```

The first convolutional layer performs a 2D convolution operation on a 5x5 window of the feature map. The stride is automatically set to 1, such that the filter moves one column over at a time. I padded the image with 2 pixels on each side such that the input MNIST character image changes from a 28x28 pixel image to a 32x32 pixel image and the convolution preserves spatial dimension. Without this padding and stride, the image would be too small for such a large kernel size. The layer takes the image with one input channel and outputs 6 channels (feature maps).

Here is where the first batch normalization layer would be stacked. It is a regularization method which calculates the mean and standard deviation for the batch in order to standardize them. The layer takes in all 6 output channels.

The output feature maps are then put through the tanh(x) activation function. The tanh(x) activation function squashes numbers to the range [-1, 1]. It is also zero-centered. This is not the best activation function, but it is what is used in paper [1]. It has the crucial function of providing nonlinearity to the network.

Then, the average pooling aka subsampling layer comes in. This layer uses 2x2 filters with a stride of 2 to downsample the 6-layer 14x14 feature map output of the previous layer.

Then, another convolutional layer is added which takes in the 6 14x14 layers and outputs 16 layers of 10x10 feature maps. There is no padding necessary here and the kernel size and the stride is the same.

Here is where the second batch normalization layer would be stacked. The layer takes in all 16 output channels.

The output feature maps are then put through the tanh(x) activation function.

Another average pooling layer transforms the 16 10x10 feature maps to 16 5x5 feature maps.

Here is where we call another convolutional layer to convert the 16 layers to 120 fully connected feature maps using a 5x5 kernel.

Next, we flatten the maps. We then move onto the other fully connected layer. This linear layer takes in the 120 nodes from the flattened maps and fully connects them with the 84 nodes of this layer. It updates the weights and adds bias. Then we activate them with the tanh(x) activation function again in the next layer.

If we wish, we can use the dropout layer here. The probability that each channel will be zeroed is p=0.5. This is a regularization technique meant to allow the model to more easily generalize what a handwritten character in the MNIST dataset looks like by zeroing or "dropping out" random feature nodes during training.

Here, we set another linear layer which computes the 10 output nodes of our network. We use LogSoftmax to get the final output of our network.

Experimental Procedure and Results

I wrote my project code in a file called **proj1.py** to finish this project. I used the hyperparameters of *batch_size=128, epochs=10,* and *lr = 0.05* where lr is the learning rate of the model. I chose to use CrossEntropyLoss for the loss function for training. The optimizer was the torch stochastic gradient descent (SGD) function. My dropout layer was left at the default probability of *p=0.5*. I set *no_cuda = False* so the model trained on the GPU which is a 2047MB NVIDIA GeForce MX330 (Lenovo). If you do not have a GPU, you can set this variable to False.

I ran my Python program in a Conda environment in Visual Studio Code. You can find the environment.yml file in the .zip file to import my Conda environment if you wish to replicate my results. In order to get the output of each network version for training and testing, I set the code to output the console log to a file as it runs. You can see the .txt files in the .zip folder. Also, if you wish to replicate my results, you must manually comment out or uncomment the lines of code with the batch normalization layer and the dropout layers. They should be labeled with comments in **proj1.py.** (See Appendix for all file descriptions.)

All the test accuracies for all of the versions of LeNet-5 using MNIST are all 99%.

Project Results

	Model Name	**Test Accuracy**	**Test Set Average Loss**	**Training and Testing Execution Time**
1	LeNet-5 - FC with dropout, CONV with BN	9892/10000 (99%)	0.0314	854.982631444931 seconds
2	LeNet-5 - FC with dropout, CONV without BN	9878/10000 (99%)	0.0324	262.257784843448 seconds

3	LeNet-5 - FC without dropout, CONV with BN	9906/10000 (99%)	0.0297	290.8610923290 2527 seconds
4	LeNet-5 - FC without dropout, CONV without BN	9877/10000 (99%)	0.0358	282.9163019657 135 seconds

Conclusion

The test accuracy for LeNet-5 model configuration 3 is 99% with very little loss on average for the test set. However, the extra batch normalization layers caused the total execution time to be the second longest, at 290.86109232902527 seconds. Model configuration 1 has the second highest performance, performing only slightly worse than the third configuration. This configuration also has two batch normalization layers, which contributes to its significantly longer execution time.

The test accuracy for the configuration 4 is the lowest. It has the highest average loss for the test set. However, this model configuration also executed in the second shortest amount of time at 262.2577848434448 seconds due to its lack of additional layers. Model configuration 2 has the second lowest test accuracy, performing close to configuration 4. Its total execution time is the shortest out of all configurations. This means that the base LeNet model performed worse than the models with added regularization techniques, which is to be expected.

From this, I can conclude that the LeNet-5 model configuration 3 with the addition of convolutional layers with batch normalization and no dropout is the best performing network, despite not having the shortest execution time. It seems that dropout, with its zeroing out of nodes, helps decrease the overall training time of the network, while the process of batch normalization increases the network's overall training time.

References

1. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.
2. LeCun, Yan. "The Mnist Database." MNIST Handwritten Digit Database, Yann LeCun, Corinna Cortes and Chris Burges, 1998, http://yann.lecun.com/exdb/mnist/.
3. https://github.com/mbornet-hl/MNIST/tree/master/IMAGES/GROUPS
4. "MNIST Database." *Wikipedia*, Wikimedia Foundation, 17 Apr. 2022, https://en.wikipedia.org/wiki/MNIST_database#cite_note-1.
5. https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

6. https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html
7. He Yanmei, Wang Bo, and Zhu Zhaomin. 2020. An improved LeNet-5 model for Image Recognition. I*n Proceedings of the 2020 4th International Conference on Electronic Information Technology and Computer Engineering (EITCE 2020)*. Association for Computing Machinery, New York, NY, USA, 444–448. https://doi-org.proxy.libraries.rutgers.edu/10.1145/3443467.3443797
8. Wenlong Li, Xingguang Li, Yueya Qin, Wenjun Song, and Wei Cui. 2019. Application of Improved LeNet-5 Network in Traffic Sign Recognition. *In Proceedings of the 3rd International Conference on Video and Image Processing (ICVIP 2019)*. Association for Computing Machinery, New York, NY, USA, 13–18. https://doi-org.proxy.libraries.rutgers.edu/10.1145/3376067.3376102

Appendix

- See proj1.py in the .zip file for the code.
- See lenet_cfg1.txt for the console output from the network - FC with dropout, CONV with BN
- See lenet_cfg2.txt for the console output from the network - FC with dropout, CONV without BN
- See lenet_cfg3.txt for the console output from the network - FC without dropout, CONV with BN
- See lenet_cfg4.txt for the console output from the network - FC without dropout, CONV without BN
- See environment.yml to import my Conda environment