Mini Project 1 – Classification, weight sharing, auxiliary losses

Luca Bracone, Omid Karimi, Gianni Lodetti

Introduction

The objective of this project is to design and test different architectures in Pytorch, in order to compare two digits visible in a two-channel image. It aims at showing in particular the impact of weight sharing, and of the use of an auxiliary loss to help the training of the main objective.

Data

The goal of this project is to implement a deep network such that, given as input a series of 2×14×14 tensor, corresponding to pairs of 14 × 14 grayscale images, it predicts for each pair if the first digit is lesser or equal to the second. To generate the data we use the generate_pair_sets(1000) to generate 1000 pairs of images. it returns a train_input containing 1000 pairs of of 14x14 images, a train_target containing the target of the comparison of the image pair(representing digits), and a train_classes tensor of size 1000x2 containing the the classes corresponding to the digit of each image. This method also similarly returns data for the tests in the variables test_input, test_target, test_classes.

Architectures

To derive our architectures we followed a bottom-up approach, starting by the simplest architecture and adding different optimizations as we went along. This approach allows us to observe the improvements braught by different architectures.

.1. Fully Connected Neural Network. Our first approach was to simply feed the entire input to a fully connected neural network with one hidden layer and an output of two nodes for yes and no outputs, as you can see in Table 1 the results were poor. To improve the results we added batch normalization between the layers, to standardize the mean and variance of each unit in order to stabilize learning. The average results for 1000 train pairs, 1000 test pairs and 25 epochs are the following.

Table 1. Fully connected neural network trainign results, with and without batch normalization

Arch	Error[%]	σ[%]	train time[s]
FNN	44.7	1.01	0.32
FNN + BN	2.13	1.20	0.74

We observe that the fully connected neural network without batch normalization has poor performance, it is only a little better then flipping a coin, however adding batch normalization is very effective at improving the training of our neural network.

.2. Convolutional Neural Network. The second architecture is the same as the previous one but with 2 extra convolutional layers with max pooling, before the fully connected layers, this model is similar and inspired by "LeNet" model. The average results for 1000 train pairs, 1000 test pairs and 25 epochs are the following

Table 2. Convolutional neural network without auxiliary loss

Arch	Error[%]	σ	train time[s]
CNN + BN	18.5	0.76	2.41

We can conclude that adding convolutional layers successfully extracts meaningful feature maps that allow the network to better compare the images. Convolutions are also a form a weight sharing since we reuse the the filter weights over all the inputs, however adding convolutions increases the training time of our model.

.3. Auxiliary Loss. Up until now we have not used the individual target test and train classes provided to us by the dataset. The following model take advantage of these classes by seperating the data pairs into individual digit images and training the model to classify each of the digits in the pairs. This is done by using the train classes and adding an auxiliary loss for each digit of the pair, before comparing them in the final layer of the neural net.
Our first model with this improvement the "CNN_AUX" model. This model is in essence the same CNN model as the previous one but with this added technique.
To add auxiliary loss we started by modifying the forward pass to seperate the individual digit in each pair and then train the model on each digit individually, by giving each digit its own convolutions and network layers, before comparing the two digits in the last layer of our neural network. The forward pass returns a (x, y ,z) triplet containing the result for the classification of the first digit x, second digit y, and final comparison result z. Finally we updated the "train_model" function to compute a loss for each of the elements of the returned triplet and simply summed them together to obtain the total loss.
The results obtained with 1000 pairs and 25 epochs are:

The overall improvement of the average test error shows that this technique is effective and that allowing the network to

Table 3. Convolutional neural network with auxiliary loss

Arch	Error[%]	σ	train time[s]
CNN+BN+AUX	16.3	1.59	4.60

classify the digits before comparing them in a connected layer, improves the performance of the network at comparing the digits.

.4. Siamese Network for more Weight Sharing. A Siamese neural network (sometimes called a twin neural network) is an artificial neural network that uses the same weights while working in tandem on two different input vectors to compute comparable output vectors.
In our case we notice that we could add further weight sharing by using a siamese model in the classification of the digits, before we compare the two digits in the last layer of our model. This is implemented in our "SIAMESE_CNN_AUX" model by simply reusing the same convolutions and layers for the two digits (otherwise it is very similar to the "CNN_AUX" model). The results obtained are:

Table 4. Siamese Convolutional neural network with auxiliary loss

Arch	Error[%]	σ[%]	train [s]
SIAM+CNN+BN+AUX	10.4	0.92	4.71

It is clear that by adding weight sharing for the classfication of the individual digits allows our model to train better and improves the error rate further.

.5. Experimenting with activation functions and optimizers. Until now we have always used the same functions to test our code, that is we have used ReLU as activation and SGD as optimizer. In this section will try the sigmoid and leaky-ReLU for activation as well as the Adam optimizer instead of SGD on our Siamese model with weight sharing and auxiliary loss. We observe the following results, averaged over ten runs:

Table 5. Siamese Convolutional neural network

Arch	Error[%]	σ[%]	train time[s]
SGD+SIG	17.5	2.04	4.67
SGD+ReLU	12.26	1.24	4.62
SGD+Leaky	11.80	0.89	4.65
Adam+Sig	10.20	1.54	4.95
Adam+ReLU	11.37	5.14	5.10
Adam+Leaky	8.07	1.46	4.80

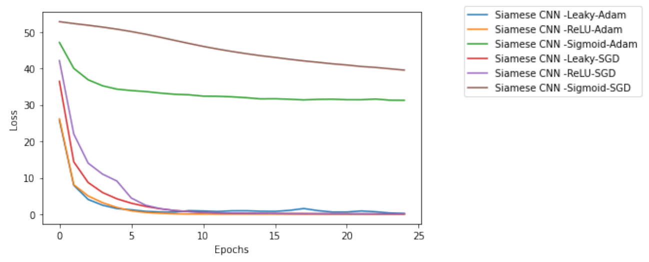

Fig. 1. Evolution of loss (in logarithm) over the course of training

We can observe that the leaky-ReLU activation with the Adam optimizer obtain the best results.

Conclusion

We started by implementing a simple fully-connected neural network and as expected it performed poorly, but adding batch normalization greatly improved it.
We then added convolutions, a common technique for image classification and observed a slight improvement in the error rate and standard deviation, as convolutions help to filter and detected features within images. This improvement was magnified by taking advantage of the provided "train_classes" in our dataset, to first classify each image of the image pair using an auxiliary loss, before comparing them in a final fully-connected layer.
Our final architecture takes advantage of weight sharing with a siamese architecture, by passing both images of the pair through the same convolutions and fully-connected layers, so the neural network classifying the images of the pairs can train twice as much with the same data. Each improvement in the architectures achieved better result than the previous and the siamese network with auxiliary loss achieves the best result.
We wrapped up by trying out different activation functions and optimizers on our final architecture and notice that using leaky-ReLU in tandem with the Adam optimizer achieves the best results in our case with an error rate of 8% and standard deviation of 1.4%.