Project 1 – Classification, weight sharing, auxiliary losses

Zahra Jaleh, Seyedvahid Mousavinezhad

Abstract

In this project, we are testing different neural network architectures to compare two digits visible in a two-channel image. The purpose of this project is to demonstrate the impact of weight sharing and the use of auxiliary loss in training.

1 Introduction

Deep Learning algorithms are mathematical structures with multiple layers of processing that can separate data features into various abstraction layers. The input feature data is sequentially passed from one layer to the next during many repeating iterations of a deep neural network. In each layer, there are multiple perceptrons that accept weighted inputs (output of the previous layer, weights, and bias). The sum of these inputs is passed to an internal activation function, which optimizes the model's performance. Once the information has passed through all layers, as described above, the model generates an output that is compared with the truth. Based on the true label value, the loss is calculated and then used to update the weights.
The goal of this project is to demonstrate the impact of weight sharing and auxiliary loss on multilayer perceptron and a convolutional neural network. The idea of weight sharing is coming up from using the same weight for two different input images. The idea of auxiliary losses is to use two different neural nets for two images and compare them as a concatenated image by another neural net which is inspired by this paper [1]. To study the impact of weight sharing and auxiliary losses together, we use two neural nets, the first one predicts the label of each image and the second one takes the output of the first neural nets (concatenation of the output for two images) as input and predicts the comparison result. For implementing all the networks, other approaches such as batch normalization, max pooling, and dropout help us to achieve improved results.

2 Data

The training set and test set of this project are 1,000 pairs of $2\times14\times14$ tensors, corresponding to pairs of 14×14 grayscale images from the MNIST data set. The label of the data indicates if the first digit is lesser or equal to the second image.

3 Network Architecture

in this project, we implement two basic neural networks, a fully connected and a Convolutional. For the fully connected we implement a three layers Multilayer perceptron and for the Convolutional we implement LeNet-5.
We have six different neural networks:

- **MLPNet with Weight sharing:** As it has mentioned earlier, for weight sharing implementation we use two neural net classes, one for classifying the images and one for predicting the label of comparing the digits.
- **LeNet5 with Weight sharing:** To implement weight sharing with LeNet5, instead of using two different neural net classes, we use a single class, in its forward function it splits the pair images then does their classification, and then concatenates the result of two images, and finally predicts the label of comparing the digits.
- **MLPNet with Auxiliary loss:** In this part, we implement three neural net classes, two of them are the same and classify each image of pairs (used for auxiliary loss computation), and the third one takes the concatenation of the output of pair images and predicts the final label. The losses of the first and second neural net classes were weighted by 0.5 and the loss of predicting the final label was weighted by 0.3.
- **LeNet5 Auxiliary loss:** We implement the same procedure as MLPNet with Auxiliary loss. The structure is shown at figure 1.
- **MLPNet with Weight sharing + Auxiliary loss:** : In this part, we implement two neural net classes, one for classifying each image of pairs and one for predicting the final label of pair images.
- **LeNet5 with Weight sharing + Auxiliary loss:** We implement the same procedure as MLPNet with Weight sharing + Auxiliary loss. The structure is shown at figure 2.

4 Result

All types of neural networks were run ten times on 1000 pairs of images and the results(train and test accuracy) are shown in the following tables. We set the learning rate = 0.01, the number of epochs = 25, batch size = 200, and the optimizer as Adam.

By considering the result in the following tables we can conclude that weight sharing improves the performance of models and then by adding auxiliary loss the performance of models is getting much better. We can see that the accuracy of MLP with weight sharing + auxiliary loss is about 92% and LeNet-5 is about 95%.

Table 1: LeNet 5

Neural Nets	Train accuracy		Test accuracy	
	Mean	STD	Mean	STD
Basic	98.38	0.218	79.56	1.18
Weight sharing	99.13	0.205	85.03	1.546
Auxiliary loss	98.99	0.137	93.76	2.192
Weight sharing + Auxiliary loss	98.3	0.3	95.09	0.756

Table 2: MLPNet

Neural Nets	Train accuracy		Test accuracy	
	Mean	STD	Mean	STD
Basic	99.05	0.415	78.75	0.964
Weight sharing	98.22	0.46	84.32	0.796
Auxiliary loss	98.83	0.5	92.04	1.197
Weight sharing + Auxiliary loss	98.71	0.448	92.87	1.455

5 Conclusion

In this project a multilayer perceptron and a convolutional neural network are examined for the impact of weight sharing and auxiliary loss. We can conclude that weight sharing improves model performance and by adding auxiliary loss to weight sharing, models perform significantly better.

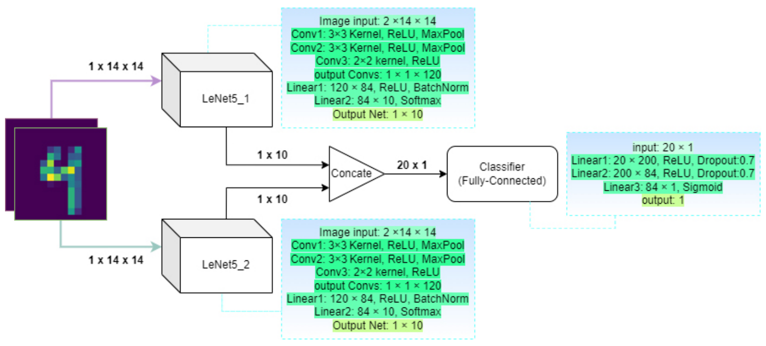
Image input: 2 ×14 × 14
Conv1: 3×3 Kernel, ReLU, MaxPool
Conv2: 3×3 Kernel, ReLU, MaxPool
Conv3: 2×2 kernel, ReLU
output Convs: 1 × 1 × 120
Linear1: 120 × 84, ReLU, BatchNorm
Linear2: 84 × 10, Softmax
Output Net: 1 × 10
1 x 14 x 14
LeNet5_1
1 x 10
Concate
20 x 1
Classifier (Fully-Connected)
input: 20 × 1
Linear1: 20 × 200, ReLU, Dropout:0.7
Linear2: 200 × 84, ReLU, Dropout:0.7
Linear3: 84 × 1, Sigmoid
output: 1
1 x 10
1 x 14 x 14
LeNet5_2
Image input: 2 ×14 × 14
Conv1: 3×3 Kernel, ReLU, MaxPool
Conv2: 3×3 Kernel, ReLU, MaxPool
Conv3: 2×2 kernel, ReLU
output Convs: 1 × 1 × 120
Linear1: 120 × 84, ReLU, BatchNorm
Linear2: 84 × 10, Softmax
Output Net: 1 × 10

Figure 1: LeNet5 with Auxiliary loss

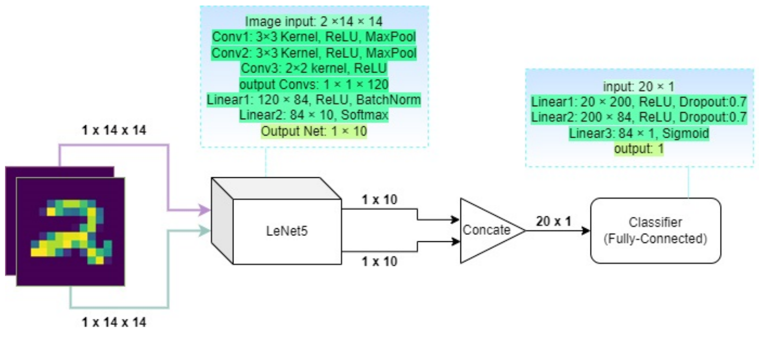
Image input: 2 ×14 × 14
Conv1: 3×3 Kernel, ReLU, MaxPool
Conv2: 3×3 Kernel, ReLU, MaxPool
Conv3: 2×2 kernel, ReLU
output Convs: 1 × 1 × 120
Linear1: 120 × 84, ReLU, BatchNorm
Linear2: 84 × 10, Softmax
Output Net: 1 × 10
input: 20 × 1
Linear1: 20 × 200, ReLU, Dropout:0.7
Linear2: 200 × 84, ReLU, Dropout:0.7
Linear3: 84 × 1, Sigmoid
output: 1
1 x 14 x 14
LeNet5
1 x 10
Concate
20 x 1
Classifier (Fully-Connected)
1 x 10
1 x 14 x 14

Figure 2: LeNet5 with Weight sharing and Auxiliary loss

References

[1] Christian Szegedy et al. “Going Deeper with Convolutions”. In: *arXiv* (sep 17, 2014).