Deep Learning Mini Project 1: Classification, weight sharing, auxiliary losses

Julien Salomon* - Romain Caristan*

I. Introduction

The goal of this project is to train different models on pairs of hand-written digits from the *MNIST* data set, to output "1" if the first digit is inferior or equal to the second one else "0".

The different models have the following guidelines:

- The models take as input two 14x14 grayscale images of hand-written digits and should output a 0 or a 1.
- The first model is a convolutional neural network.
- The second model is a convolutional neural network that uses weight sharing.
- The third model is a convolutional neural network that uses weight sharing and auxiliary losses.

II. Finding the model

By doing some research on deep neural networks used to recognize hand-written digits, the **LeNet5** model seemed like the most adapted model for this task.

This is the structure of the LeNet5 model:

- **Layer 1 :** The input is a 32x32 grayscale image (with one channel). A first convolutional layer with 6 feature maps or filters having size 5x5 and a stride of one is applied to the input that changes dimensions from 32x32x1 to 28x28x6.
- **Layer 2 :** This is an average pooling layer with a filter size 2x2 and a stride of 2 is applied. The resulting image dimensions is of **14x14x6**. The primary function of this layer is to reduce the number of parameters learned by the network. A sub-sampling or averaging layer often comes after a convolutional layer.
- **Layer 3 :** This is a second convolutional layer with 16 feature maps having size 5x5 and a stride of 1. This second convolution lowers the number of connections in the network. The resulting image dimensions is of 10x10x16.
- **Layer 4 :** This layer is the same as the second layer (filter size 2x2 and a stride of 2). The output is reduced to 5x5x16.
- **Layer 5:** This is a fully connected convolutional layer with 120 feature maps each of size 1x1. It is not followed by an average pooling or sub-sampling as the output is 1x1.
- **Layer 6:** This is a fully connected layer with 84 units. This layer and the previous one take the multiple samples obtained by the convolutional layers (1 to 4), and output into a single vector of values the probability that a certain feature belongs to an output number.
- **Output layer:** The final layer is a fully connected **softmax** output layer with 10 possible values corresponding to the digits from 0 to 9: softmax assigns decimal probabilities to each of the 10 possible digits.

All layers excepted the output layer, have *tanh* for activation function. Modifying this model, to take as input two 14x14 images, and adding layers comparing the two images can thus be a solution to this project.

III. Implementing Sub_LeNet5

As the input of the desired model is two 14x14 images, the model will follow the LeNet5 model, but starting with the third layer, that takes as input a 14x14 image. To detect the two digits, layer 3 to 7 (the output layer) are passed to each image. The resulting model thus has 10 layers to recognize each image.

The two 10x1 vectors obtained are then concatenate into a 20x1 vector, that is then passed through 3 linear layers:

- The first linear transformation takes the 20x1 vector and outputs a 100x1 vector.
- The second linear transformation takes the 100x1 vector and outputs a 100x1 vector.
- The third and final linear transformation takes the 100x1 vector and outputs a 1x1 value.

The first two linear layers have a relu activation function, and the last one has a sigmoid activation function, as the desired output is a binary classification. This model has 130877 features.

As a loss function, binary cross-entropy measures how far away from the true value (which is either 0 or 1) the prediction is for each of the classes and then averages these class-wise errors to obtain the final loss. It is thus ideal for a binary classification task like this one and will be used for this model.

The result of this model, is thus a binary classifier that recognizes the two input images independently, and then uses the recognized image to classify them.

IV. Implementing Sub_LeNet5 with weight sharing

As the first 10 layers recognize independently the images (5 layers for the first image, 5 layers for the second), and as the two images are from the same data set and have the same attributes, weight sharing can be applied between those identical layer.

The result of weight sharing, is the decay in the number of parameters of the neural network, which compresses the neural network and reduces over fitting.

When applying weight sharing as described above (Layer 1 and Layer 6, Layer 2 and layer 3...), the number of parameter is 71589 which is far inferior to the previous model.

As the output is the same as in the previous model, the binary cross-entropy loss is kept for this model.

V. Implementing Sub_LeNet5 with auxiliary loss

As the model defined has clear modules (recognize image 1, recognize image 2, compare the images), it is possible to use **auxiliary losses** to train the model.

Auxiliary losses can be used to classify using intermediary outputs from layers. As the Sub_LeNet5 model is more fitted to do hand-written digit recognition, it would make logical sense to use the auxiliaries image recognition (output of layer 5 for the first image and the output of layer 10 for the second image), and compute the loss given their prediction.

Once those numbers are classified using the auxiliary losses, it is a deterministic task to define if the first digit is smaller or equal to the second one. By training the model using auxiliary losses on the digit classification, the last 3 layers can be thus deleted.

Combined with weight sharing, this lowers the number of parameters consequently (59288 parameters), and should augment the accuracy of the model, once trained.

The negative log-likelihood function is defined as $loss = -log(y)$ and produces a high value when the values of the output layer are evenly distributed and low. As the auxiliary losses to test are on 2 vectors of probabilities, this loss seems to be adapted to the new model and will be used as auxiliary losses.

VI. Optimizing the learning rate and the batch size

The learning rate and batch size are features that can highly influence the model's accuracy. To find the optimal learning rate and batch size for the models, grid search has been done for those two features. Due to limitations in computing power and time, the learning rate has been optimized on the arbitrarily set batch size of 100. The batch size was then optimized using the optimal learning rates obtained.

A. The learning rate

The tested learning rates were: 0.1, 0.05, 0.01, 0.001, 0.0001, 0.00001. By training 20 times the three models Sub_LeNet5, Sub_LeNet5_WS (the Sub LeNet5 with weight sharing), and the Sub_LeNet5_WS_auxLoss (the Sub LeNet5 with weight sharing and auxiliary loss), and averaging the resulting test accuracies, the optimal learning rates were obtained.

	Sub_LeNet5	Sub_LeNet5_WS	Sub_LeNet5_WS_AL
0.10000	0.534067	0.513667	0.558933
0.05000	0.558200	0.555400	0.897600
0.01000	0.841600	0.864333	0.962667
0.00100	0.833933	0.852067	0.960067
0.00010	0.800267	0.819000	0.906933
0.00001	0.598667	0.560467	0.697800

Fig. 1. Mean of the test accuracies obtained by the grid search on the learning rates

The results can be observed on Figure 1. The optimal learning rate for the Sub_LeNet5 is 0.01, for the Sub_LeNet5_WS is 0.0001 and for the Sub_LeNet5_WS_auxLoss is 0.01.

B. The batch size

The same has been done on the following batch sizes: 50, 100, 250, 500, 1000. The results can be observed on Figure 2.

	Sub_LeNet5	Sub_LeNet5_WS	Sub_LeNet5_WS_AL
50	0.807933	0.836867	0.959533
100	0.818333	0.819000	0.962667
250	0.820667	0.758600	0.961800
500	0.701933	0.604800	0.958667
1000	0.600133	0.576667	0.952000

Fig. 2. Mean of the test accuracies obtained by the grid search on the batch sizes

The optimal batch sizes for the Sub_LeNet5 is 250, for the Sub_LeNet5_WS is 100 and for the Sub_LeNet5_WS_auxLoss is 50.

VII. Adaptive learning rate

The performance of the model on the training data set can be monitored by the learning algorithm and the learning rate can be adjusted in response. This update of the learning rate can help the loss to converge.

For the models, adaptive learning rate is implemented very simply in the following way: begin with the optimal learning rate obtained by the grid search. For each epochs in the fitting of the model, if the difference between the loss of the previous epoch and the current loss is less than 3% than the previous loss, the learning rate is divided by 2.

VIII. Results

By training each model 20 times, using the optimal batch size and learning rate obtained, and by both activating or not adaptive learning rate, the best model can be highlighted by obtaining the best average test accuracy. The results can be observed in Figure 3.

The auxiliary loss is a very efficient technique that heightens the accuracy significantly. By saving the best accuracy of each model, before they overfit, the optimal means and the standard deviation for each model can be observed on Figure **??**. The adaptive learning rate also lowers the standard deviation of the results.

There are multiple deep learning techniques that can be used to fine tune a model. Using a simple convolutional neural network to do this classification task creates a model with many features that is not very accurate. By very intuitively adding weight sharing and auxiliary loss, the model obtained and doing simple feature optimization, the accuracy can highly be increased.

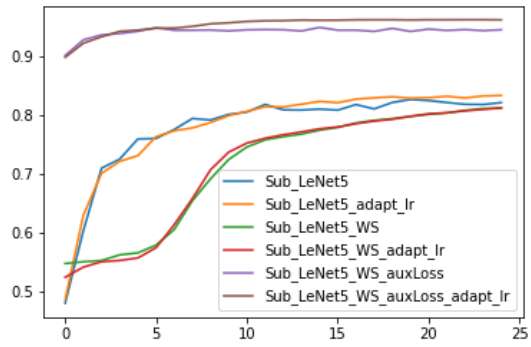

Fig. 3. Mean test accuracies by epoch for running the models 20 times