EE 559:Deep Learning-Miniproject 1

Tianyang Dong[a], Yu-Ting Huang[b], Shengzhao Xia[b]
[a] *Institute of Mathematics, EPF Lausanne, Switzerland*
[b] *Institute of Electrical Engineering, EPF Lausanne, Switzerland*

I. Project Description

This project aims at comparing the performance of different architectures used for training and predicting a data set of pairs of two-channel image. The purpose is to show the impact of weight sharing and the use of an auxiliary loss on improving the performance.

The data set includes a series of $2 \times 14 \times 14$ tensors, corresponding to pairs of 14×14 grayscale images. The Boolean value target predicts for each pair if the first digit is lesser or equal to the second. And additional information about the classes of the two digits is provided.

II. Project Structure

A. Methods

In the models, many useful methods are used for improving the training process.

- **Batch Normalization:** It explicitly forces the activation statistics during the forward pass by re-normalizing them. It makes the layer's input concentrate so that the distribution of a certain layer's input is relatively stable, no matter what the data looks like.
- **Dropout:** When the training error goes to zero, 'dropout' helps avoid over-fitting by randomly drop the units and also the connections so that the hidden layers will not rely on the input of some specific units. The probability to drop neurons is always 0.5.
- **Maxpooling:** Max-pooling computes max values over non-overlapping blocks. Its objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned. Such an operation aims at grouping several activations into a single more meaningful one.

B. Loss Function

Because the aim of the architectures is classification, we use softmax as output layer and use following loss functions.

- **L_2 Regularization:** We use L_2 regularization to force the parameters of the models to be relatively small, so that they will not change drastically to fit the training data and avoid over-fitting.
- **Cross-entropy Loss:** It's defined as -$\sum_k p(k)log(q(k))$. Compared with mean squared error, cross-entropy loss stays convex when using softmax. So that it prevents the learning process from falling into local minima.
- **Auxiliary loss:** Auxiliary loss is the cross-entropy loss introduced by classification error of classes of the two digits in each pair, and each data point (a pair of grayscale images) could generate two auxiliary losses. Auxiliary loss could optimize the learning process and parameters before the layer generating auxiliary loss. Besides, auxiliary loss can help to mitigate gradient vanishing problem by generating additional gradient.

C. Models

Three different architectures are implemented. The first architecture (Baseline Model) contains only linear layers. The second architecture with weight sharing (CNN Model) is based on the first one, combining convolutional layers. Structure of these two models are shown in tables, see Table I, II, Fig.1.

TABLE I
Structure of Baseline Model

Layer	Model
1	Linear: $392 \rightarrow 160$
	BN
	ReLU
	Maxpool(kernel_size=2,stride=2)
2	Dropout(p=0.5)
	Linear($80 \rightarrow 128$)
	BN
	ReLU
	Maxpool(kernel_size=2)
3	Dropout(p=0.5)
	Linear($64 \rightarrow 2$)

TABLE II
Structure of CNN Model

Layer	Model
1	Conv(2,16,kernel_size=5,padding=3)
	BN
	ReLU
	Maxpool(kernel_size=2)
2	Conv(16,20,kernel_size=3,padding=3)
	BN
	ReLU
	Maxpool(kernel_size=2)
3	BN
	Linear($720 \rightarrow 100$)
	ReLU
	Dropout(p=0.5)
	Linear($100 \rightarrow 2$)

Based on the CNN Model above, the third architecture (CNN Model with Auxiliary Loss) is derived by using auxiliary loss. It means that in the loss function, except the loss of predicted target, the loss of predicted classes is also used for better learning. As shown in Figure 1,

$$Loss = MainLoss + AuxiliaryLoss$$

There are 2 fully connected layers in CNN Model with Auxiliary Loss that output the results. The Linear($120 \rightarrow 10$) outputs the predicted classes of the images, while the Linear($10 \rightarrow 1$) outputs the targets. To achieve this, we split each pair of images into two separate images to learn the features of images to predict their classes.

III. Project Result

The three models are compared under the same conditions. All training and testing experiments are done with 1000 pairs of images.

- **Parameters:** All models contain 70000 parameters.

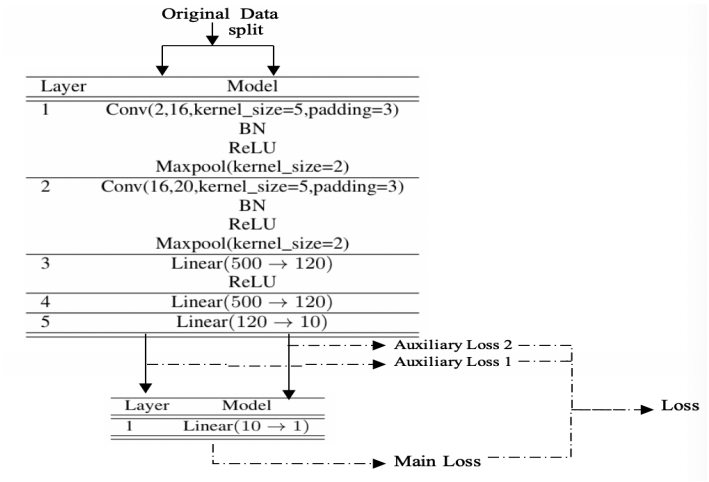

Fig. 1. Structure of CNN Model with Auxiliary Loss

- **Epoch:** Size 25.
- **Mini-batches:** Size 100. Mini batches make the training more efficient by updating the parameters by small batches, and also avoid huge amount of computation.
- **Optimizer:** Adam with learning rate 0.0005 for Baseline Model and CNN Model, 0.001 for CNN Model with Auxiliary Loss.

The comparison results of the three models are shown in Table III, Fig 2-Fig 6. To better show the influence of weight sharing and auxiliary loss, we control the number of parameters of the three models. The test error rate is estimated through 15 rounds for each architecture and every round the training and testing data are randomized.

TABLE III
PERFORMANCE OF MODELS

	Model 1	Model 2	Model 3
Nb of Parameters	73954	77530	69849
Average train error rate	7.51%	1.58%	7.37%
Train error std. deviation	0.012	0.008	0.012
Average test error rate	25.04%	19.76%	13.53%
Test error std. deviation	0.014	0.012	0.014

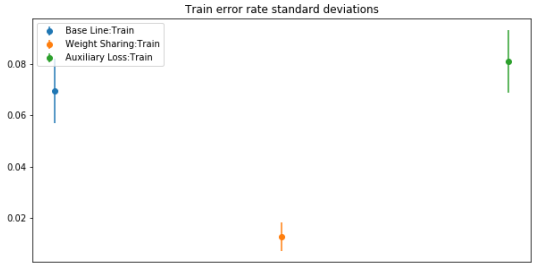

Fig. 2. Train Error Rate Standard Deviations

It can be seen that from Baseline Model to CNN Model, weight sharing improves the performance by about 5%. Auxiliary loss improves CNN Model, and reduces the error rate by 6%. From Fig 5, 6, we can see that model with auxiliary loss outperforms CNN Model stably within 15 rounds.

Fig. 3. Test Error Rate Standard Deviations

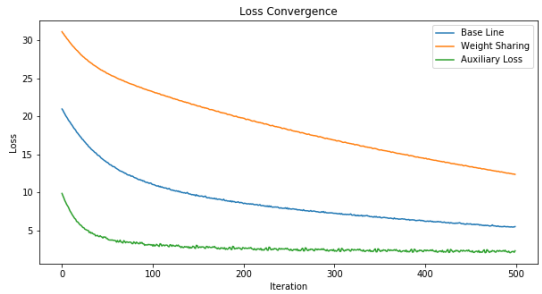

Fig. 4. Convergence of Loss Function

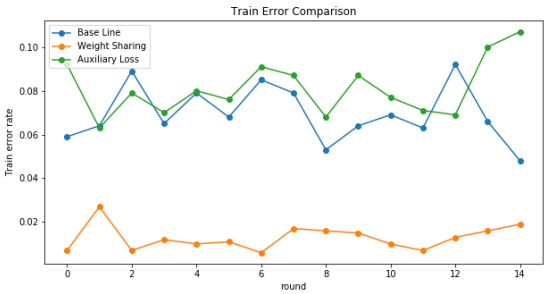

Fig. 5. Comparison of Train Errors

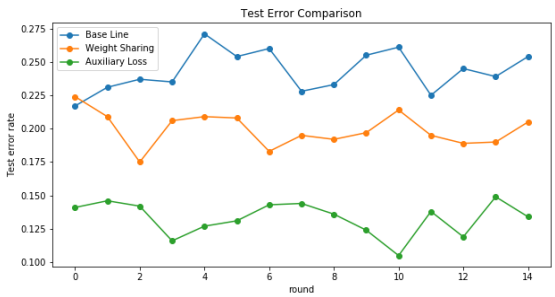

Fig. 6. Comparison of Test Errors