Report of Practical Session 1: Hyper-parameters and training basics with PyTorch

Xiaozhen WANG, Haiwei FU and Nan CHEN

1 Multi-classification problem

We tackle the problem of multi-classification on handwritten digits from the USPS dataset, which has 10 classes. To achieve optimal performance, we fine-tune the hyperparameters and explore various architectures featuring 1 to 4 convolutional layers. The highest accuracy on the validation set is obtained from the model with 3 convolutional layers, and it is selected as our final solution. Next, we experimente with various configurations of the selected model, including different numbers of neurons (64, 128, 256) and activation functions (sigmoid, tanh, ReLU). The best validation accuracy was achieved with 64 neurons and the ReLU activation function. The optimized architecture consisted of three convolutional layers with 64 neurons each, a 3×3 kernel size, and batch normalization. This is followed by a max-pooling layer, a dropout layer with a 0.1 probability, and a fully connected layer with a softmax activation function. During training, the best results were obtained using 30 epochs, a batch size of 50, Adam optimization with a learning rate of 0.01, and MSE loss. The final model achieved an accuracy of 99.15% on the validation set and 97.11% on the test set, and it required around 14 minutes of training time.

2 Regression Problem

Problem 2 is a regression task aimed at predicting house selling prices based on four features: Overallqual, YearBuilt, TotalBsmtSF, and GrLivArea. We normalize the data to ensure feature values were in the same range. After hyperparameter tuning, we find the best model to be a fully connected network with 4 layers, 3 inner layers with 128 neurons each, and ReLU activation function. This model has the lowest loss error of 893181020.80 on the validation set. Despite trying different learning rates, the SGD optimizer resulted in NaN values, likely due to the exploding gradients issue. Thus, we compare the RMSprop and Adam optimizers and find the best performance with an Adam optimizer, a learning rate of 0.0003, a batch size of 100, and 2000 epochs. In a subsequent step, we use the Gaussian likelihood loss to predict the mean(μ) and log(σ^2). This model results in a loss of 1875367424.0 on the validation set and 1750655104.0 on the test set and has low computational requirements, taking only 2 minutes to train.

3 Global Discussion for two problems

In this section, we focus on the effect of different parameters on the results and understand how to adjust the parameters to get a better-performing model. Due to the limitation of the page, we do not show all the results, but only select representative plots or tables to show.

3.1 Impact of the architecture of the model

- Number of layers: Generally, we find that adding more fully connected layers can improve the performance of the model. In the multi-classification, adding more convolutional layers improved the model performance on the validation set, but the 3 layers are best in the test set (Table 1).

Number of Layers	Epochs	Learning rate	Accuracy
1	30	0.01	94.97%
2	30	0.01	95.71%
3	30	0.01	96.51%
4	30	0.01	94.52%

Table 1. Different number of layers in problem 1

- Number of Neurons : According to both problems, adding more neurons can give better performance up to a certain value. For example, we use one layer convolutional layer in the first problem and one hidden layer in the second problem. When we take a different number of neurons, we can see that the appropriate addition of some neurons will perform well, but overfitting will occur. The optimal parameter for the first of the two problems is 512 and the second is 128 (Table 2).

Number of Neurons	Layer	Validation Loss
64	1	2,045,553,779.20
128	1	1,875,367,424.00
256	1	1,972,901,030.40

Table 2. Different number of neurons in problem 2

- Activation function: After using the different functions, we find that the sigmoid function performed the worst and the ReLU function performed the best for both problems (Table 3).

3.2 Impact of the optimizer

- Batch size : Our findings indicate that a larger batch size slows the convergence of the loss function in multi-classification problems.

Activation function	layer	learning rate	Accuracy
ReLU	2	0.01	96.46%
Sigmoid	2	0.01	94.52%
Tanh	2	0.01	94.97%

Table 3. Different Activation function in problem 1

However, the results may vary in regression problems, where higher batch sizes can converge faster. Additionally, we found that a too large batch size can lead to poor generalization, as indicated by the lower accuracy on the validation set with increasing batch size. As a result, we chose a batch size of 50 for Problem 1 and 100 for Problem 2 to balance training speed and model quality (See Figure 1,2).

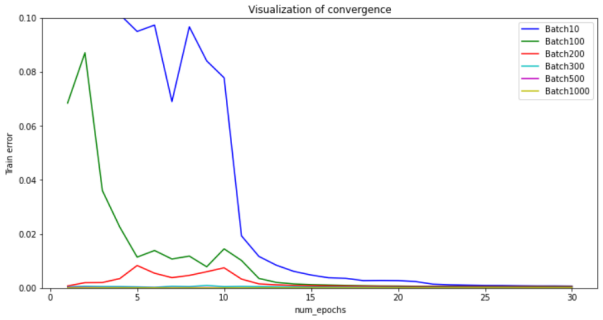

Figure 1. Different Batch size in problem 1

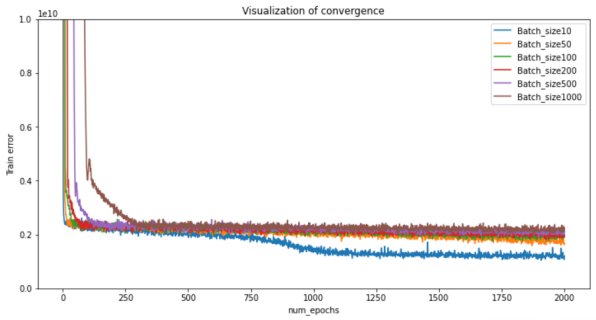

Figure 2. Different Batch size in problem 2

- Learning rate : We observe that the learning rate plays a crucial role in the adaptation of the model to the problem at hand. A small learning rate can cause slow training and get stuck in the optimization process. On the other hand, a large learning rate can lead to faster convergence but also instability or divergence of the model, as shown in Figure 3. In our problems, a moderate learning rate resulted in the best performance. In Problem 1, we also adjust the learning rate according to the number of iterations. For every 10 iterations, the learning rate is reduced to 1/10 of the original rate from a start lr = 0.01. And in Problem 2, we take the lr = 0.0003.
- Number of epochs : The number of epochs affects the training accuracy and loss. The longer the training, the higher the accuracy and the lower the loss. However, it's important to keep an eye on the validation loss to prevent overfitting and ensure better generalization. More epochs lead to longer training times but better performance, up to a point where the validation loss stops decreasing. In our problem, we take the number of epochs as 30 and 2000 according to the convergence of train loss and validation loss.

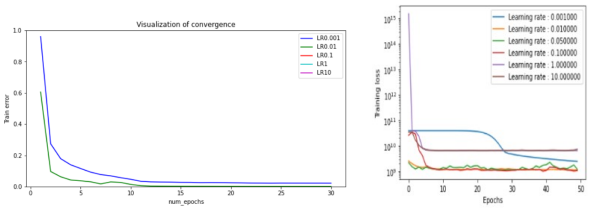

Figure 3. Different Learning Rate in classification regression

- Optimization algorithm : The optimization algorithm depends on the problem. In Problem 1, SGD optimizer is stable and resulted in better performance. But in Problem 2, we require a different approach as SGD resulted in exploding values. Both Adam and RMSprop optimizers give good results and a stable training process (See Figure 4).

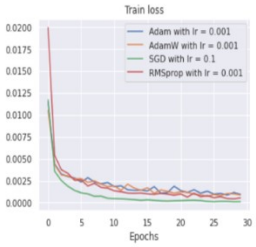

Figure 4. Different Optimization algorithm in problem 1

3.3 *Impact of Loss function*

In multi-classification problems like Problem 1, cross-entropy loss is a common choice. For Problem 2, in the regression problem like Problem 2, mean square error loss is a common choice.

Here there is a different point in our project, we use Gaussian likelihood loss and which predicts a probability distribution over possible samples, allowing us to handle ambiguous cases and express uncertainty through variance. The probability becomes:

$$p(y_i|x_i) = \frac{1}{\sqrt{2\pi\sigma(x_i)^2}} e^{-\frac{(y_i-\mu(x_i))^2}{2\sigma(x_i)^2}}$$

Then the loss function becomes:

$$L = \sum_{i=1}^{N} \frac{1}{2}\log(2\pi\sigma_i^2) + \frac{1}{2\sigma_i^2}(y_i - \mu_i)^2$$

If we set $\sigma = 1$, we obtain MSE the loss function. We can see the different loss function performance for training (See Figure 5).

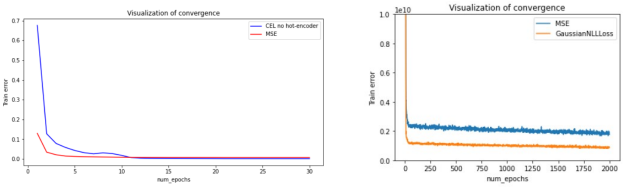

Figure 5. Different Loss function in classification and regression