Deep learning in practice: HW1

Mahdi Kallel, Noureddine Sedki, Raphael Reme

January 2021

1 Introduction

.

The goal of this homework is to learn the impact of different hyper parameters on training a neural network.

For this matter, we will be studying 4 use cases ('Binary classification', Multi class classification', 'Multi task problem' , 'Regression'), through the same pipe of experiences: "Impact of the architecture of the model", "Impact of the optimizer" and "Impact of the loss function".

In this report we will present our conclusion, and all the codes, figures and results can be retrieve from the associated notebooks.

2 Binary classification

.

In this section, we are training a multi layer perceptron classifier, will try to test different architecture of the network by varying hyper-parameters, such as the depth (number of layers), the with (number of neurons) and the activation function. As the size of the problem is small we can test different settings which is not all time possible in real world application. Our architecture consists of an MLP of n_{hidden} layers, this MLP is connected to two output layers with 1 output and Sigmoid activation to compute a probability.

We tried network with number of hidden layers from 1 to 4 and numbers of neurons from 5 to 100 alongside to different activation function (ReLU, Sigmoid, Tanh), we found that changing the architecture of the model without changing the training parameters don't have a huge impact on the loss and accuracy of the model, we decided to keep a model of 3 layers, 100 neurons each and Tanh as activation function, as this architecture gives us best accuracy at this stage, (experiment of changing architecture are represented in figures a to c).

We will vary the training parameters, we start by testing different batch size after the learning rates and finally the training duration with the number of epochs.

As we can see in figure 1, training with large batch size slower the training procedure and we notice a drop in accuracy on validation set, the model generalize less. For our experiment we will keep best batch size set to 10.

Varying the speed of convergence trough the learning rate, a small lR slower the convergence and large value of lR the optimzation procedure can diverge from optimal solution.

Optimizer affect the learning procedure, if we keep SGD we need to adjust learning hyper-parameters and the learning rate. for Adam and RMSprop we choose a Lr of 0.001 which gives as an accuracy of 99% .

For classification task the Binary cross entropy is more suitable loss, since it penalize the miss classified point, previous experiments where run using the MSE error, in this experiment we compare the effect of changing it to Binary Cross entropy for the same subset of optimizers (SGD,Adam,RMsProp).

	SGD	RMSProp	Adam
BCE	69.1	99.9	98.2
MSE	76.0	99.5	97.5

3 Multi-Class classification

.

In this section, we are training a CNN classifier, will try to test different architecture of the network by varying hyper-parameters, such as the number of convolution, the number of hidden neurons and the activation function.

Our architecture consists of a stack n_{conv} of CNN's followed by an MLP of n_{hidden} layers this MLP is connected to two output layers with 10 outputs for digit classification.

We tried network with number of number of convolution layers from 1 to 3, having 3 convolution don't help the model as it needs more change in optimizer parameter, as we will add a MLP layer we will keep the number of convolution SET to 1 (accuracy of 76%). The impact of neurons in the MLP was tested by varying their number from 5 to 100, we found that more is the best and with 100 neurons the loss and accuracy of the model improve (accuracy of 86%), we decided to keep a model of 1 convolution, 100 neurons in the MLP.

By testing different activation function, ReLU outperform Tanh and Sigmoid.

We will vary the training parameters, we start by testing different batch size after the learning rates and finally the training duration with the number of epochs.

As we saw before, small value of batch size (5 per example) the model to generalize more and help speed up convergence, high value of batch size the convergence is more difficult, there is a trade off between Batch size and the learning rate. For the following we will keep a batch size of 10.

For this example, when using SGD as optimizer Learning rate of 0.1 or 1 will lead to an accuracy of 97%, and for Adam and RMSprop a lR of 0.001 gives similar accuracy.

For classification task, cross entropy loss is more suitable, since it penalize the miss classified point, previous experiments where run using the MSE error, in this experiment we compare the effect of changing it to Cross entropy for the same subset of optimizers (SGD,Adam,RMsProp).

	SGD	RMSProp	Adam
CE	97.8	99.5	99.3
MSE	97.1	98.5	98.6

One of advantage of changing the loss function is that convergence is achieved after only 10 Epochs for the same performance.

4 Multi task :

.

In this experiment we are training a neural network for two tasks of classification, the first is to identify the number in the image and the second is to identify it's color.

Our architecture consists of a stack n_{conv} of CNN's followed by an MLP of n_{hidden} layers this MLP is connected to two output layers with 10 and 4 outputs for digit, color identification respectively.

The loss was computed by adding the loss for both tasks. Which we can still backpropagate through the network.

In all our experiments we noticed that the model always classified the colors with a 100% accuracy after the first couple of epochs, therefore, it's associated loss term has almost no impact with respect to that of digit classification which takes about 20 epochs to reach 80% accuracy. Therefore we saw no interest in changing the depth at which we divide the network for each task.

In order to fine tune a network for this task, we realized the following experiments, who's plots you can check in the figure below :

1. **Number of convolutions (a)** : We change the number of convolutions before the MLP, we use 3x3 CNN's. Having a convolution definetely helps our network, having two our model performance decreases, maybe because it needs more time to train. Including one convolution seems like the good trade off as it yields the best "both accuracy" in the shortest time.

2. **Number of hidden units (b):** After fixing the number of convolutions, we change the number of units in the commun MLP, from the plots we see that the more the better as it gives the network more expressive power. We settle for 100 units.

3. **Activation(c):** We test changing all the activations of our network to "Sigmoids", "Tanh" and "ReLU". The latter outperforms the rest.

4. **Batch Size (d):** Batch size represents the number of training samples used to compute the approximation of the loss and perform SGD. One would expect that the higher the batch size, the more accurate our estimate would be and thus we get better gradients. From our experiment we see that the best batch size is in the order of 10, we found this to be quite counter intuitive. One possibility is that with higher batch sizes most of the gradients get closer to zero, making us get stuck more easily in local minimas.

5. **Learning rate(e):** Learning rate may be the most import parameter to tune. If it's too low training gets slower and it's easier to get stuck in saddle points. If it's too high you can never converge as you keep jumping in and out of minimum valleys. There's no formula to chose this parameters as it's highly dependant of the dataset and the task in hand as it must be tuned to fit the curvature of the loss landscape.

In our case we see that a learning rate of 0.1 of 1 is the best choice as it yields approx. 94% both accuracy after 20 epochs.

6. **Optimizers (f):** Optimizers control the way the computed gradients are applied to our parameters. For the learning rate we choose 1, 0.01 and 0.001 for SGD, RMSProp and Adam (for Adam and RmsProp these correspond to the default values). In practice these two optimizers are less senstivie to the learning rate than SGD. Despite it's simplicity, SGD seems to outperform RMSProp for this task. It's noisier gradients seem to have a regularizing effect. Adam with it's node wise adaptive learning rate seems to be the best candidate with no tuning for it's learning rate.

7. **Loss(g):** Classification being a discrete task, we cannot back propagate through this action and thus we need to settle for continuous loss functions that can approximate this metric. Previous experiments where run using the RMSE error, in this experiment we compare the effect of changing it to Cross entropy for the same subset of optimizers (SGD,Adam,RMsProp). In the table below, we report the both accuracy after 20 epochs of training for both of the losses. We see that using CE improves our performance in all scenarios.

	SGD	RMSProp	Adam
RMSE	91.1	85.7	94.2
CE	92.8	93.3	95.3

8. **RESULT :** After training a network with all the best parameters we found we report on a both accuracy of 92.68 on the test set. With 100% for the colors.

5 Regression :

In this section the goal is to train a network to predict the house selling prices given 4 features. We studied the impact of the hyper-parameters.

The first thing we had to do, was to preprocess the data so that the gradient does not explode. We therefore have standardized the inputs (the mean and scale are extracted from the training set) and in order to keep meaningful outputs, we have divided all the prices by 10 000.

One can then see that a single layer neural network achieves to be trained and reaches a Mean Absolute Error (MAE) of around 26 000 € on the validation set. (In comparison, predicting simply the mean of the train prices have a MAE of 56 000 €.)

In order to improve things, we studied the impact of the hyper-parameters:

1. We first tried to see the impact of the architecture: We restricted our self to a Sequential network, with linear layers and an activation layer between each of the linears. We have observed that the ReLU outperforms Tanh, Sigmoid and Elu. And that more width (neurons by layer) increases the results. However, increasing the number of layers has not proven itself to be useful. With the default optimizer we have reached a MAE of 22 000 € on the validation set with 1 hidden layer of 100 neurons.

2. Then we tried see the role of the number of epochs and the batch size. As expected, with smaller batch, we learn much quicker (because we do more step each epochs) but noisier. A good trade off with SGD seems to be below 64. With 100 epochs we do not observed overfitting, and for large batch size there were still room for improvements. We tried therefore to learn a large network of 5 hidden layers of 100 units each, over 1000 epochs and with a 64 batch size, to see if it would overfit. It has not. It even reached our best MAE (19 500 €) over the validation set.

3. We also studied the impact of the optimizer. We used a basic model with 2 hidden layers of 20 units with several batch sizes for Adam, RMSprop and SGD (keeping the learning rate at 0.1). In conclusion, SGD is much slower than the two others. RMSprop is the fastest, but is much more noisy than Adam and SGD. The good trade-off seems to be Adam, he is almost as fast to converge than RMSprop and is really stable for batch size ≥ 64.

4. We then tried to explore different learning rates. We were using 0.01 that yields good results, but we have tried 0.001, 0.01, 0.1 and 1 with different architectures (different in size). The conclusion are that with bigger architectures, it's best to take smaller learning rates. The training is not stable for big architectures and big learning rates. And is not stable at all for learning rate greater than 0.1. Best learning rates seems to be below 0.01 and one have to keep in mind that a small learning rate also reduces the training speed.

5. Finally we tried another loss than the MSEloss. For this loss our network is trained to predict the mean and scale of a Gaussian Law for each input such that the output maximizes the likelyhood of the law. We observed overfitting in this case (but it does not seems to really impact our predictions, and we probably overfitted the scale.) The results were not as good as with the previous loss (it could be expected, as we are learning a more difficult problem) but it gives us information over its confidence on its prediction.

Finally we've reached our best results using a batch size of 64, a learning rate of 0.01 and 5 hidden layers of 100 units each on the validation set. We trained such a network with Adam and the MSE loss over 1500 epochs and save the model that performs best over the validation set. (In order to prevent overfitting on the train set). It reached a MAE of 21 500 € over the test set.

We also try to train a network with the second loss and observes the relation of the predicted scales w.r.t. the 4 features. It does seem that the network predicts smaller scale in the frequent cases (which is an expected behavior). It predicts also bigger scales for higher prices (as expected). (And thus bigger scales for bigger Basement Surface/Living Area/Overall Quality)