Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 1

Learning and generalisation in feed-forward networks — from perceptron learning to backprop

ZiYuan Wang, Ming Jiang

January 26, 2021

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to get a deeper understanding with regard to neural networks algorithms
- to explore the learning and generalising capacity for single-layer neural network and multiple-layer neural network
- to understand the substructure of the library of our choice, TensorFlow

The scope of this particular assignment is well-structured and fulfilled for our capacity. And the limitation for the given project is not too much apart from the computational limitation when it comes to tensor calculation.

2 Methods

The coding language for our task is built upon Python 3.6 due to the overall universality for dependences. For the first part, we conduct the code based on NumPy 1.19.2. For this assignment's second part, we utilised TensorFlow 1.15.1 for building neural networks, and Pandas 1.1.3, SciKit Learn 24.1 for model comparison.

3 Results and discussion - Part I

3.1 Classification with a single-layer perceptron

3.1.1 Generation of linearly-separable data

We generate the data with the instruction given, with the parameter of $mA \in 2, 15, mb \in -1.5, -1.0, sigmaA = 0.5, sigmaB = 0.5$ with randomly sample of 100 data points. And we added some bias as instructed due to if we don't, the initial weight for models could just be separate the dataset perfectly.

3.1.2 Classification with a single-layer perceptron and analysis

Due to inherently difference between two algorithms, by adjusting the learning rate for two models, we can see that perception will continuously update the weight till just separated, however, with enough training, Delta learning would targeting at the best boundary for the dataset.

- Given the situation that our dataset is linearly separable, delta rule in batch mode and sequential mode have similar results, the latter however takes longer convergence time. The convergence time varies with the data distribution.
- The learning_rate affects the convergence time, and if it's set as a small value, the convergence time will be large. However, if it's set as a great value, the error will become more diverge.
- The weight initialisation for this particular task is quite sensitive for converge, due to if the dataset is distant enough, the chance for the initial weight could just be separate the dataset, then the perceptron is converge naturally.
- The mA and mB decide the centres of two distribution, if the data is separate, but the separate space is not through the origin(Due to No Bias), the perceptron learning will continuously finding the solution.

3.1.3 Classification of samples that are not linearly separable

We followed the instruction given to adjust the dataset for Delta Rule training, and the observations we have are given:

- If the dataset is not linear separable, the perceptron learning will keep looking for a solution that does not exist, until the epochs end. Delta rule will find a solution that minimizes the error.
- If one class has more data than the other, the direction of gradient descent will towards the class with more data points.
- If the data of the class is not representing the whole distribution, the model will not be fully trained.

3.2 Classification and regression with a two-layer perceptron

3.2.1 Classification of linearly non-separable data

We use two non-linearly separable Gaussian distribution datasets in this part. Both mean squared error and the ratio of misclassification are evaluated in the process parameter tuning. The basic parameter settings are: **epochs = 300, learning_rate = 0.01**, according to our testing. Fig. 1 show the error curves changed by the parameter $Nhidden$, which denotes the number of hidden nodes. We set it as a range of [4, 5, 10, 12]. Fig. 2, 3 show the boundary with $Nhidden = 5$ and $Nhidden = 10$.We then split the data into training set and validation set, with the ratio ranged in [0.2, 0.3, 0.5, 0.6]. Fig. 4 show the error curves change in this situation. Finally, we plot the boundary in Fig. 5, 6.
Observations and Answers:

- With the ratio equals 0.2, we can observe much more similarity.
- A larger hidden node number leads to less error and better boundary.

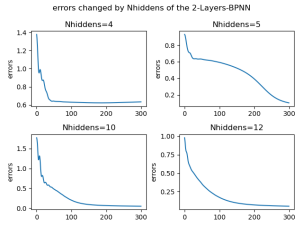


Figure 1: Error Curves Changed by $Nhidden$

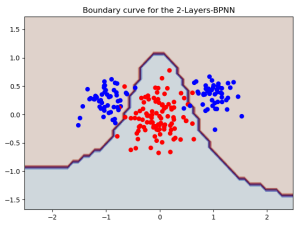


Figure 2: $Nhidden = 5$

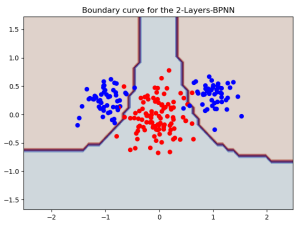


Figure 3: $Nhidden = 10$

- In a sequential model, each time we only choose one data in the matrix as the input and update the weight, thus the learning will be much more stochastic.
- The model is not well trained with a low proportion of training data and the boundary cannot separate the two class completely.

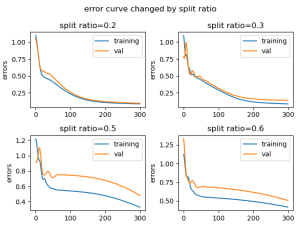


Figure 4: Error Curves Changed by *Ratio*

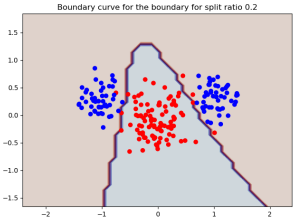


Figure 5: Boundary with 20% Training Data

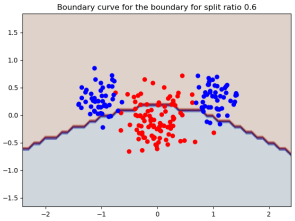


Figure 6: Boundary with 60% Training Data

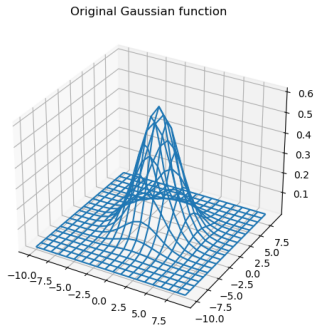


Figure 7: The Original Function

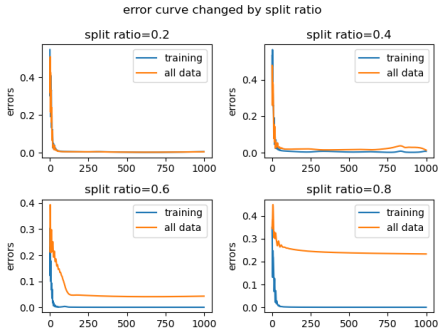
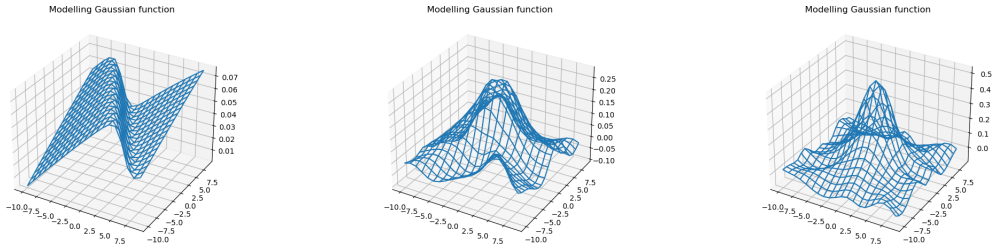


Figure 8: Error Curves Changed by Split Ratio

3.2.2 Function approximation

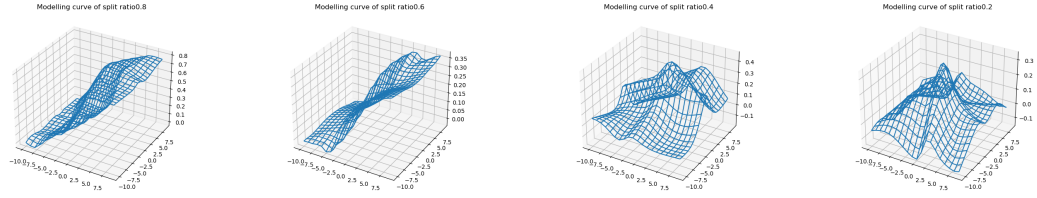
We generate 2-D grid data from -10 to 10, every one step a data. Then we combine all pairs to form the training data, and add an extra rows with value one as the bias. As a result, we now have the training set *pattern* with the size of (400, 3), and the corresponding labels Z generated



(a) Nhidden = 2 (b) Nhidden = 10 (c) Nhidden = 20

Figure 9: Fitting Curve with Different Hidden Node Numbers(*epochs=1000, learning_rate=0.01*)

from the given bell shaped Gauss function, with the size of (400,). And Fig. 7 shows the original 3-D distribution of the data.
We first test the fitting results with hidden layer nodes varying in the range of [2, 10, 20], using all data for training. Fig. 9 show the results.



(a) 20% Training (b) 40% Training (c) 60% Training (d) 80% Training

Figure 10: Fitting Curve with Different Ratio of Training Data

Observations and Answers:

- When the nodes number is very small(equals 2), it's just a uni-dimensional approximation.
- When the nodes number is a little bit larger(equals 15 for example), it's bi-dimensional approximation.
- When the nodes number is very large(equals 25), the curve becomes more complicated because the regression is more complicated. However beyond 25 nodes the improvement is not so significant.
- The best model is with low error and meanwhile not so complicated. So 20 nodes is a good choice.
- We might use the mini-batch to speed up the convergence.

Selecting the nodes number as 20, we train the model by sampling different ratios of data. We reduce the training data ratio as the range of [0.8, 0.6, 0.4, 0.2], and obtain Fig. 10.

- When we use just a few part of data for training(just 20%, for example), the model might have a bad performance.
- The dataset we selected is not enough for representing the entire data space.

4 Results and discussion - Part II

4.1 Three-layer perceptron for time series prediction - model selection, validation

As for the grid search, the parameters are $nh1 \in \{3, 4, 5\}$ and $nh2 \in \{2, 4, 6\}$. Other parameters are given by $learning\_rate : 0.001, momentum : 0.99, epochs : 40$. The combinations of different models with their validation loss(MSE) as well as convergence time in Table 1.

Table 1: Grid Search Result for nh1 and nh2

nh1	3	3	3	4	4	4	5	5	5
nh2	2	4	6	2	4	6	2	4	6
val_loss	0.0108	0.0152	0.0506	0.0085	0.0165	0.0087	0.0193	0.0169	0.0126
elapsed_time	1.0719	0.9328	0.4541	1.0046	1.0859	1.0758	1.1129	1.1712	1.2249

To take a peak for how hidden nodes affect the performance of three-layers neural networks at this task, we generate the clear time-series dataset in Fig **??** and applied grid search technique for finding the best fit model and the worst fit model. The MES value given in the Fig 12 is the performance on test set.

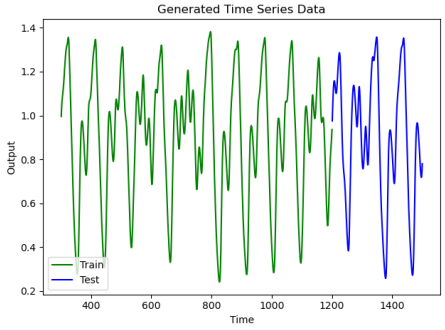


Figure 11: Clean Time Series Data

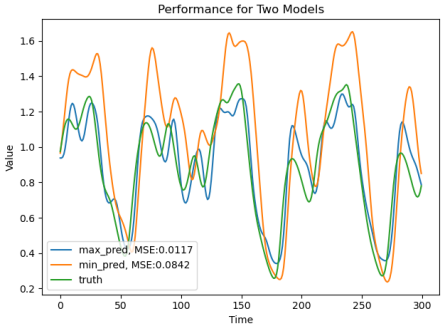


Figure 12: Best/Worst Model Performance

Observation:

- Whether the model is early stopped is strongly related to the weight initialisation, with some weight initialisation, the worse case situation could just be stopped by 15 epochs with the EarlyStopping patience of 10 epochs.
- Worst case scenario, the prediction could just be a horizontal line with little fluctuation.

4.2 Three-layer perceptron for noisy time series prediction with penalty regularisation

The data set we generate with Gaussian noise $\delta \in 0.05, 0.15$, and the distribution of dataset could be seen in Fig 13 and Fig 14.With the dataset, we can see what is the impact of $nh2$ one our prediction, with $nh2 \in 3, 6, 9$. The result could be found in Table 2:
Observation:

- The more noise we added, the poorer the performance is overall, regardless of the structure of the network.
- Regularisation will leads to weight shrinkage as well as and let the decent of error be moother and smaller.
- As λ become smaller and smaller, we can see more and more time or epochs needed to converge in Table 3, and with high level of noise data, the lower the λ, the better for the model.

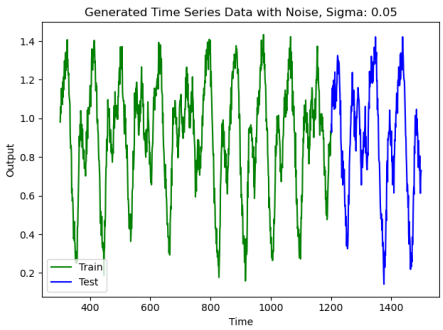


Figure 13: Noise Level:$\delta = 0.05$

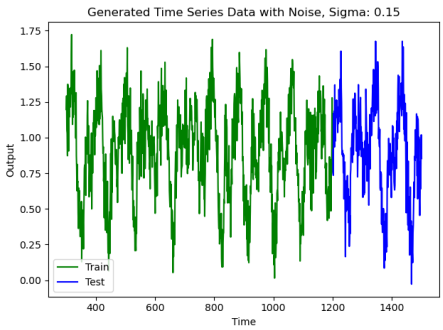


Figure 14: Noise Level:$\delta = 0.15$

Table 2: Grid Search for nh2 and δ

nh1	nh2	δ	val_loss	loss
4	3	0.05	0.0139	0.0146
4	3	0.15	0.0506	0.0533
4	6	0.05	0.0148	0.0152
4	6	0.15	0.0328	0.0325
4	9	0.05	0.0171	0.0194
4	9	0.15	0.0366	0.0357

Table 3: Grid Search for λ and δ

nh1	nh2	λ	δ	val_loss	loss
4	9	0.01	0.05	0.0173	0.0167
4	9	0.001	0.05	0.0174	0.0188
4	9	0.0001	0.05	0.0208	0.0214
4	9	0.01	0.15	0.0433	0.0458
4	9	0.001	0.15	0.0386	0.0452
4	9	0.0001	0.15	0.0397	0.0321

5 Final remarks

This lab assignment enabled us to understand the neural network and control the model based on the dataset with various parameters. And through the lab, we learn the impact of noise data on a given task, and the difference between batch learning and sequential learning.
Since one of our teammates decided not to continue the class, some of the tasks about plotting various figures could be tricky and non-mandatory tasks. During this lab process, trying to find the relation of the parameter of weight decay regularisation and the amount of noise is a bit tricky to dig deep. This is because setting the parameter with too many possible values could steeply increase computational consumption and duration.