CS-550 - Spring 2023 HW 2: Artificial Neural Network

Salih Deniz Uzel (deniz.uzel@bilkent.edu.tr)
22201382

I. Section A

- Using linear regressor or the given data sets isn't sufficient. Data distribution of the train1 and train2 have higher order polynomial function like distribution. Data is not vary on a linear line. Therefore a single hidden layer can be used to find a higher order polynomial representation. According to Universal Approximation Theorem, approximately $degreeofthepolynomial$ number of neurons are sufficient to learn a polynomial function. In the homework, I selected the number of neurons in hidden layer as 8 to fit train1 data set and 12 to fit the train2 data set.
- I used $tanh$ activation function.
- I used $MeanSquaredError(MSE)$ loss function.
- I found that the $LearningRate = 0.1$ is good enough to learn the given task
- I used Xavier weight initialization method for the weight. And zero initialization for the biases. In addition, glorot uniform and normal and even without scaling the normal distribution initializations resulted in a good fit.
- I stopped training when loss didn't improve for 5 times in a row and epsilon for the minimum amount of improvement is $1e-5$ for the train1 dataset. Then the program returns the best weights. However the process of backing up weights and biases add extra overhead to training time. I used 10k epochs for train1 data set. I trained train2 20000 epochs without patience.
-
  - Using momentum is not necessary for $train1$ data set. The function can be learned successfully without using momentum.
  - I used momentum for the $train2$ data set with batch learning. Momentum factor $beta = 0.999$. The concave curve around $x = 75$ was falling short. Momentum helped the model to make the concave curve turn downwards later due to momentum. Same behaviour also improved the fitting at the left tip of the data. It was continuing upwards. Momentum slowed down the these gradients and helped the model to learn these data.
- I have used batch learning in my algorithm. It was sufficient to learn given task. Also, SGD can be used with momentum to dampen SGD-induced oscillation.
- Normalization affect the learning process. Activation functions like sigmoid and tanh saturates quickly for the small values of x. And output values are greater than the saturation value of these function. ReLu activation functions only passes the positive values. Therefore model can't present the negative part of the data for this regression task. I used z-score normalization $(x-\mu)/\sigma$ for the input data. And I de-normalized $(x \cdot \sigma) + \mu$ the output data to plot in the fit method.

TABLE I: Configurations

	Datasets	
Parameters	**train1**	**train2**
ANN used	8	12
Selected activation function	Tanh	Tanh
Selected loss function	MSE	MSE
Learning rate	0.1	0.1
Range of initial weights	Xavier	Xavier
The number of epochs	8755/10k	20k
When to stop	P(5), 1e-5	# of epochs
Is momentum used	No	Beta=0.999
Is normalization used	z-score	z-score
Stochastic or batch learning	batch (10)	batch (32)
Training loss	1.22	11.24
Test loss	1.25	13.81

* $P(5), 1e-5$: the training process stops when no improvement in the loss value more than 1e-5 is observed after 5 iterations.

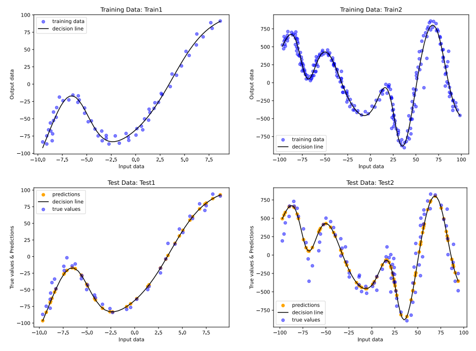

Fig. 1: HW Section A. Train1 (upper left, Test1 (lower left), Train2 (upper right), Test2 (lower right))

II. Section C

The linear regression resulted in a position where it could minimize the loss of MSE. Looking at the decision line, it can be interpreted that the train1 and train2 cut the data sets in the middle. It is not sufficient to represent the given data sets. The Single hidden layer of different sizes, on the other hand, has resulted in a way that minimizing loss allow the model to reach the necessary complexity to fit the function. All the models trained on train1 is complex enough to learn the data. However, for the train2 data set, models with 2 and

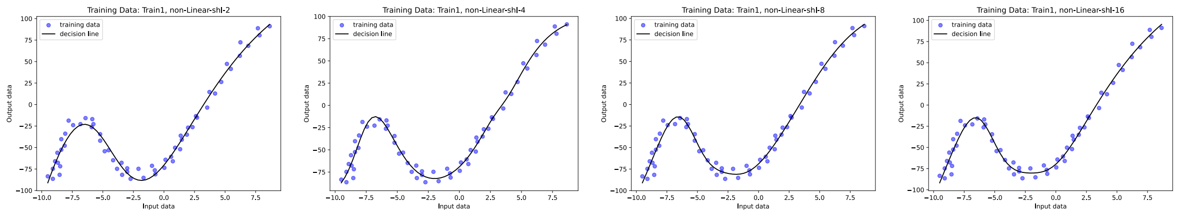

Fig. 2: HW Section C. Non-linear runs from hidden layer size 2, 4, 8, 16 for Train1 data set.

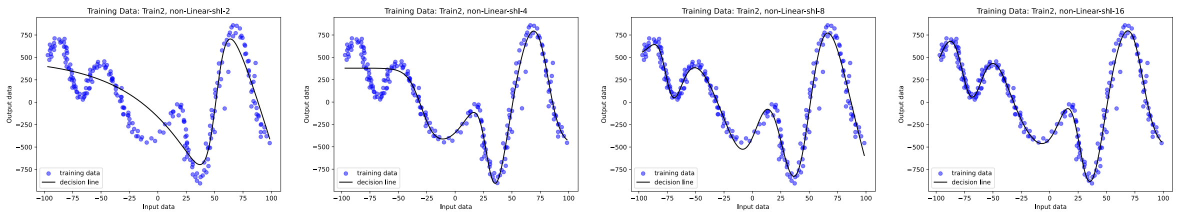

Fig. 3: HW Section C. Non-linear runs from hidden layer size 2, 4, 8, 16 for Train2 data set.

4 hidden units seems to can't form a good combination for train2 data set complexity. The general behavior of all the models is to have a high learning rate between 0.1 and 1. If the network complexity is low, it has been observed that high learning rate prevents high bias. The momentum method overcame the problem of models struggling to learn concave points effectively, resulting in a reduction in the total number of epochs needed.

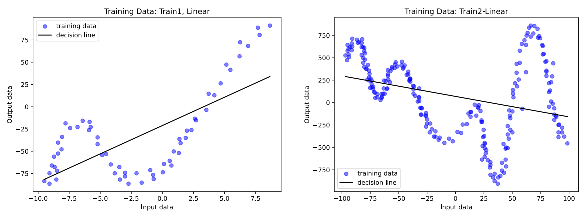

Fig. 4: HW Section C. Linear regression runs Train1, Train2 data set. "The images in the row are presented from left to right as train1 and train2.svg, respectively."

TABLE II: HW Section C. Linear regression Training and Test MSE Losses on the Train1 and Test1 data.

	Train1		Test1	
Method	Loss	Std.	Loss	Std.
Linear Regression	30.8338	0.5139	24.310	0.5892

TABLE III: HW Section C. Linear regression Training and Test MSE Losses on the Train2 and Test2 data.

	Train2		Test2	
Method	Loss	Std.	Loss	Std.
Linear Regression	205.0413	0.8954	90.0387	0.9786

TABLE IV: HW Section C. ANN regression Training and Test MSE Losses on the Train1 and Test1 data.

	Train1		Test1	
Hidden units	Loss	Std.	Loss	Std.
2	1.5740	0.0262	1.5193	0.0368
4	1.1472	0.0191	1.1901	0.0288
8	1.2018	0.0200	1.2028	0.0290
16	1.2231	0.0204	1.2478	0.0302

TABLE V: HW Section C. ANN regression Training and Test MSE Losses on the Train2 and Test2 data.

	Train2		Test2	
Hidden units	Loss	Std.	Loss	Std.
2	49.0108	0.2140	35.3337	0.3840
4	26.2361	0.1145	19.9477	0.2168
8	13.3672	0.0584	15.3352	0.1667
16	11.2430	0.0491	13.7125	0.1490

III. Section D

In this section, I chose the loss threshold of 12 by looking at the loss/epoch Fig. 5 in the model I trained with the same values I used for 8 neurons in section c . Model training for $lr = 1$ resulted with overflow error. Base model utilized momentum and batch learning with a learning rate of 0.1. Greater learning rate help to model to learn faster. The use of a full batch led to overfitting. Probably the model encountered an excessive amount of data that was specific to certain parts of the dataset. Reducing the batch size to 32 helped to generalize the model. In addition, momentum helped the model to use the speed that comes from the steep parts of the target function and better generalized the concave part.

In the models with a lower learning rate, other methods for applied for successful generalizations. The overfit due to the decrease in the learning rate was compensated by reducing the batch size. The speed and accuracy lost due to the reduction of the batch size were compensated by the momentum method by increasing the beta value.

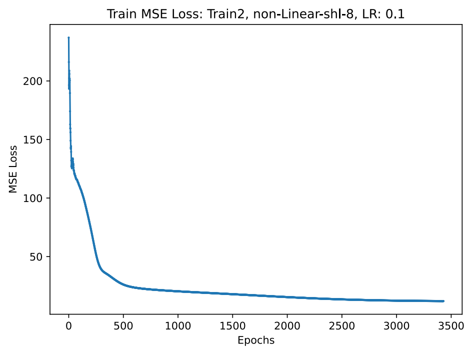

Fig. 5: HW Section d: Loss/Epoch graph of the non-linear run for learning rate 0.1 hidden layer size 8 for Train2 data set.

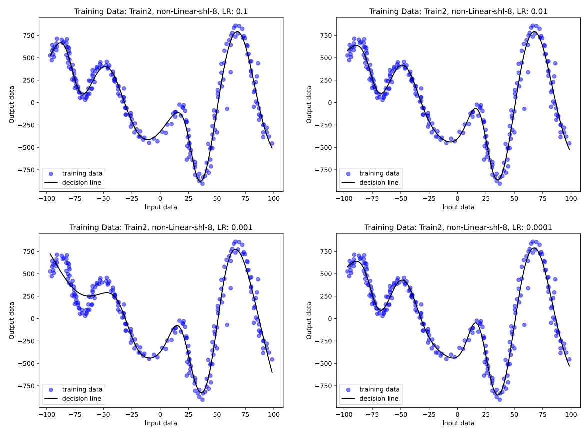

Fig. 6: HW Section D. Model decision plots of non-linear runs for the learning rate 0.1, 0.01, 0.001, 0.0001 and hidden layer size 8 for Train2 data set.

TABLE VI: HW Section D. Training MSE Losses and the total number epochs the model trained for threshold value of 12 on the Train2 data.

	Train2		
Learning Rate	Loss	Std.	Numbers of Epoch
1	-	-	-
0.1	11.9210	0.0521	3430
0.01	11.9817	0.0523	29782
0.001	18.9482	0.0827	100000
0.0001	11.9815	0.05232	22631

IV. Section E

In this section, only the effect of momentum is observed while keeping other parameters constant. Comparing the graph without momentum and the graph with momentum, it can be observed that the concave parts are better generalized by using the velocity from the parts of the slope with higher momentum in the function. However, the part on the left side of x=-25 does not seem to have the same success. The distribution of samples encountered by the model during learning may have caused the MSE to be minimized at a local optima.

For the parameters, I used the similar parameter that i used in c and d. Looking at the Loss/Epoch graph of the model trained on train2 in the section c and d 5, threshold selected as 18. The beta value is selected as $beta = 0.992$ for the model with momentum.

lr: 0.01
activation function: tanh
weight initializer: xavier
bias initializer: zeros
batch size: 32
stopping loss threshold: 18

TABLE VII: HW Section E: Training MSE Losses and the total number epochs the model trained for threshold value of 12 on the Train2 data.

	Train2		
Learning Rate	Loss	Std.	Numbers of Epoch
Without Momentum	17.9999	0.1037	9375
With Momentum	17.9280	0.0783	30000

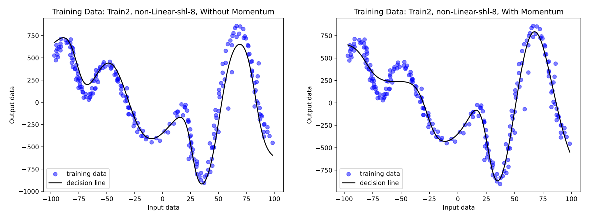

Fig. 7: HW Section E. Model decision plots of non-linear runs for the learning rate 0.1, hidden layer size 8, without momentum (left), with momentum $beta = 0.992$ (right) for Train2 data set.

V. Section F

Stochastic learning algorithm uses 1 sample for gradient update. Therefore an update with a high learning rate can slow down or hinder the process of finding global optima. As a solution to that learning rate was decreased from selected base-case 0.1 to 0.001. This change alone led to good result. However Stochastic learning algorithm needed more epochs, as a result more training time.

Batch learning algorithm uses less samples than full batch and more samples than stochastic. For this reason, it provides a faster learning opportunity compared to stochastic. Therefore, while it provides faster learning compared to the stochastic learning algorithm, it gains slightly more stochasticity than the full batch. A good result was obtained when the batch size were selected as 32 and the learning rate decreased from base-case 0.1 to 0.7.

TABLE VIII: Configurations

	Train2	
Parameters	**Stochastic Learning**	**Batch Learning**
ANN used	8	12
Selected activation function	Tanh	Tanh
Selected loss function	MSE	MSE
Learning rate	0.001	0.07
Range of initial weights	Xavier	Xavier
number of epochs run	30k	10k
Is momentum used	No	No
Is normalization used	z-score	z-score
Batch size	batch (1)	batch (32)
Training loss	12.7551	14.0551
Loss Std.	0.0556	0.0613

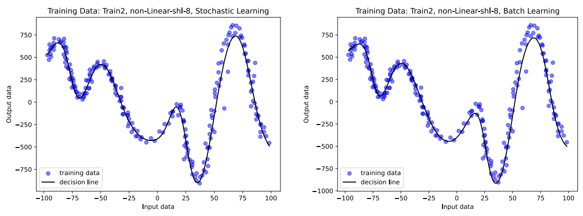

Fig. 8: HW Section F: model decision line plot of non-linear run with the stochastic learning algorithm (left), batch learning algorithm (right) for Train2 data set.