Homework # 2

CS 550 - Machine Learning

Yusuf Dalva, Bilkent ID: 21602867
yusuf.dalva@bilkent.edu.tr

INTRODUCTION

In this assignment, the effects of the hidden layers in terms of learning a nonlinear distribution is examined. In order to be able to observe the effects of various hyper-parameters on the hidden layers, several experiments has been done where each examines a different hyper-parameter for the ANN (Artificial Neural Network) implemented. This report documents the data exploration stage conducted as the first stage of the assignment and then explains the implementation briefly. The main idea behind this explanation is to show the derivations done for the implementation. Then the discussion related to experiments done considering the hyper-parameters for the regressor are shown.

DATA EXPLORATION

As the initial stage, firstly the characteristics of the datasets are examined. In order to provide ease of use for the ANN model to be implemented. Data download is performed in an automated way. The implementation of the dataset module can be found in the file *dataset_setup.py*. That file contains a method for downloading the data and dataset utilities. By using the dataset utilities implemented, the normalization and loading of data can be performed easily.
It is observed that the data has non-linear characteristics. This fact creates the expectation that the hidden layer would increase the predictive performance compared to the linear regressor.

MODELS IMPLEMENTED

Considering the requirements provided for the assignment, the implementation provides two different architectures.

A. Linear Regression Model

The initial architecture involves a linear regressor, which is only able to capture linear relationships in the data distribution. The details of the forward pass and the backward pass are provided in this section. This architecture applies a linear discriminant function only to predict an output value y. The formulation for this model is as follows:

$$y = W^T X + b \tag{1}$$

Here W represents the weights and b represents the bias term. X is shown in a generic way, but in the implementation it is one dimensional considering the datasets provided for the assignment.

B. ANN with one hidden layer

For the ANN that includes a single hidden layer, the order of calculation is as follows:

$$z^{[1]} = W_1^{[1]} X + b^{[1]} \tag{2}$$

$$a^{[1]} = \sigma(z^{[1]}) \tag{3}$$

$$y = W_1^{[2]} a^{[1]} + b^{[2]} \tag{4}$$

Here the vales [1] and [2] show the layer number and σ denotes the nonlinear activation function. The alternatives implemented for the non-linear activation function are ReLU, LeakyReLU, tanh and sigmoid functions [1]. As it can be seen from the flow, first the non-linear activation function is used (after the initial linear discriminant), which is followed by the linear regressor for prediction.

C. Activation Functions

For the implementation, there are 4 activation functions implemented. These functions are Sigmoid, ReLU, LeakyReLU and Tanh activation functions. In the following parts, the tanh function is preferred mostly due to its property of learning non-linear structures easily and having larger gradients than Sigmoid activation function. The formulation of the Tanh activation function is as follows:

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{5}$$

FINDING A CONFIGURATION FOR EACH DATASET (1-A)

After the experiments conducted, two models that originate from the ANN implementation provided performed well on the given datasets (one model for each of the datasets). The results obtained are given below considering the dataset that the result obtained.

D. Dataset 1

For the first dataset, a model has been constructed with the following hyper-parameters:

- Using a hidden layer or not: Hidden layer used with 5 hidden units
- Activation function: Tanh
- Loss Function Used: Mean Squared Error Loss (MSE)
- Learning rate value: 0.01
- Weight Initialization: Weights are drawn from a uniform distribution with the range [-0.003, 0.003]
- Number of epochs: 800
- Use of momentum: Momentum is applied with $\alpha = 0.1$

Fig. 1. ANN model fitted to dataset 1 with optimal hyper-parameters

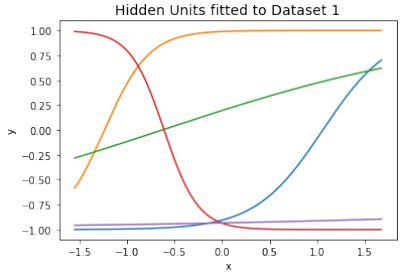

Fig. 2. Visualization of the hidden units for the model fitted to Dataset 1

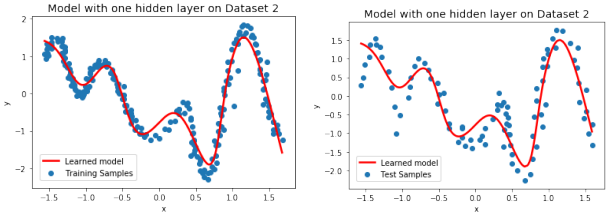

Fig. 3. ANN model fitted to dataset 2 with optimal hyper-parameters

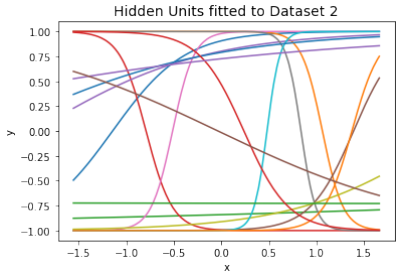

Fig. 4. Visualization of the hidden units for the model fitted to Dataset 2

- Learning Algorithm: Stochastic Gradient Descent
- Data Normalization: Data normalization is applied to both training and testing data

The loss values found as a result of training are:

- Training loss: 0.0216 with standard deviation 0.0408
- Test loss: 0.0284 with standard deviation 0.0469

The regression line that is constructed by the model having the given hyper-parameters are given in Fig. 1. In the plots, the input and output values are provided in normalized form. After the training and the assessment process of the learned model on dataset 1, in order to observe the effect of the hidden units in terms of capturing non-linearity, the hidden units learned are visualized. The corresponding plot is given in Fig. 2. In the plot, each line shows a different hidden unit. As it can be seen from the figure, the different non-linear regions are learned by different hidden units and then they are combined to form the final curve as the learned model with a regressor on the output layer. This method shows the contribution that the hidden layer makes in terms of learning the non-linearity in the data distribution.

E. Dataset 2

For the second dataset provided for the assignment, an ANN regressor is fitted with the following hyper-parameters:

- Using a hidden layer or not: Hidden layer used with 16 hidden units
- Activation function: Tanh
- Loss Function Used: Mean Squared Error Loss (MSE)
- Learning rate value: 0.001
- Weight Initialization: Drawn from a uniform distribution with the range [-0.003, 0.003]
- Number of epochs: 8000
- Use of momentum: Momentum is applied with $\alpha = 0.1$
- Learning Algorithm: Stochastic Gradient Descent
- Data Normalization: Data normalization is applied to both training and testing data

After training, the following loss values are obtained:

- Training loss: 0.0733 with standard deviation 0.1709
- Test loss: 0.2022 with standard deviation 0.3269

The resulting regressor obtained by training the model with dataset 2 can be seen on Fig. 3. This model was not able to find a line that perfectly fits the dataset, considering the higher amount of non-linear complexity. However, the results were promising as the predictions were accurate for most of the sample points. Just like the process performed for the first dataset, as the final stage of the assessment the hidden units learned are visualized. For doing that 500 points within the range of the training set are used (random uniform sampling). Considering that the non-linearity of the second dataset it higher, more hidden units are used. The visualization of the hidden units learned are given in Fig. 4. The different non-linear characteristics can be observed in the figure. The contribution of the hidden layer can be observed clearly here also.

NOTE: For all of the following parts, the abbreviation *std* is used to denote the standard deviation of the loss values obtained.

EFFECT OF NUMBER OF HIDDEN UNITS (1-C)

In this part of the assignment, the complexity contribution of the hidden units are discussed with results obtained on the dataset. To give a clearer view on about the effect of the hidden units the architecture of the model using one hidden layer is given in Fig. . Here the hidden units apply an activation function where the visualization of such units can be observed in Fig 4. By these units, a non-linear data distribution can be captured by combining them via a linear discriminant function. For the linear regressor, there are no hidden units and the input values are combined via a linear discriminant function.

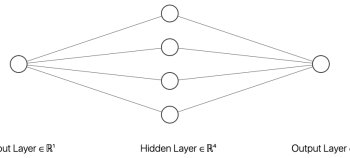

Fig. 5. Architecture of the ANN with one hidden layer (containing 4 hidden units)

TABLE I
LOSS VALUES OBTAINED WITH DIFFERENT HIDDEN UNIT COUNTS FOR TRAINING SET

	Dataset 1		Dataset 2	
Model	Loss	Std	Loss	Std
Linear Regressor	0.5843	0.4966	0.8954	1.1183
2 Hidden Units	0.2038	0.2756	0.8294	1.0252
4 Hidden Units	0.0224	0.0334	0.8348	1.0287
8 Hidden Units	0.0214	0.0369	0.0697	0.1942
16 Hidden Units	0.0805	0.1222	0.0857	0.1766

The results obtained for dataset 1 is given in Fig. 6 and results for dataset 2 are given in Fig. 7. Also a brief summary of the final loss values obtained are provided in Table I and II. The implementation of the experiments conducted for this part are given in *Effect of number of hidden units (1-C).ipynb* file with necessary explanations and comments.

Observations: Summarizing the experiment performed to observe the effect of the number of hidden units, the relationship between the complexity of the model and the number of hidden units were able to be observed clearly. For the first dataset, by adding 4 hidden units the complexity of the dataset modeled successfully. As it can be seen from Fig. 6, the models having less than 4 hidden units underfits the training data provided. As the number of hidden units increase to 4, the additional complexity that the hidden units add gives the model the ability to learn the non-linear complexity of the first dataset. Similar to this result, in the second dataset as the hidden unit count increases to 8, with careful hyper-parameter tuning, the model was able to capture the non-linear data distribution that the second dataset has. When the model has the required ability to learn the distribution, it is observed that adding more hidden units do not effect the model performance in any positive way. As the result of this discussion, the number of hidden units is directly proportional with the ability of the model to learn more complex data distributions.

TABLE II
LOSS VALUES OBTAINED WITH DIFFERENT HIDDEN UNIT COUNTS FOR TEST SET

	Dataset 1		Dataset 2	
Model	Loss	Std	Loss	Std
Linear Regressor	0.5145	0.4464	0.8954	1.1840
2 Hidden Units	0.1589	0.1885	0.7663	1.0764
4 Hidden Units	0.0247	0.0479	0.7695	1.0798
8 Hidden Units	0.0270	0.0341	0.2017	0.3249
16 Hidden Units	0.1247	0.1611	0.2266	0.3074

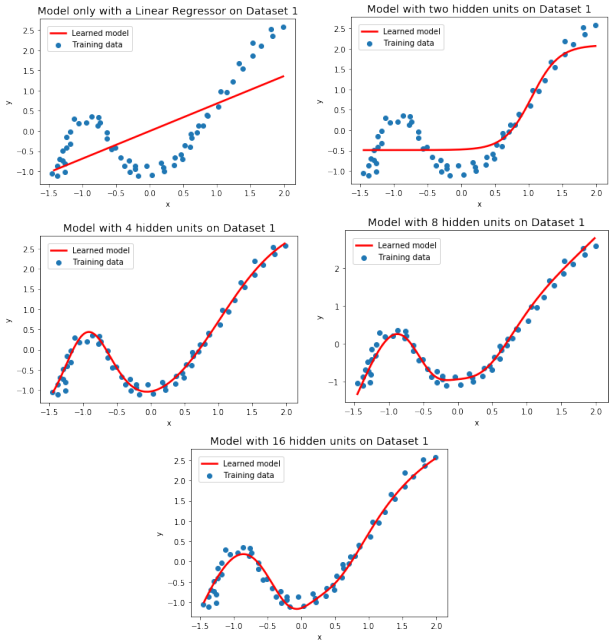

Fig. 6. Optimal models for different hidden unit counts for dataset 1

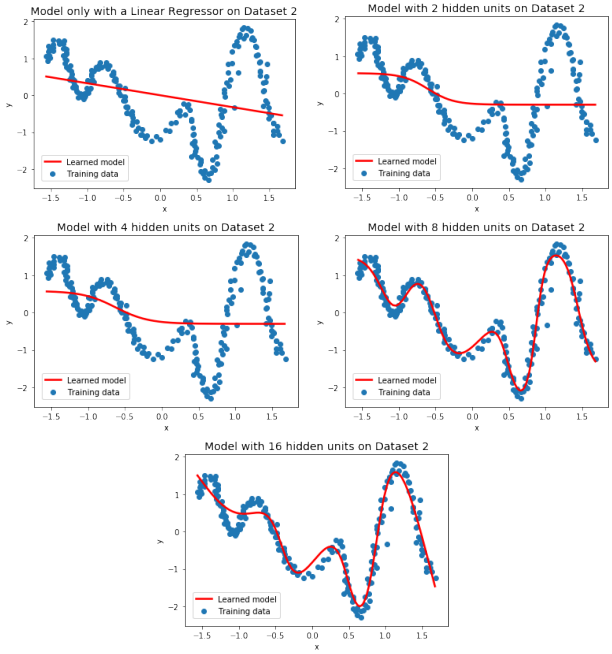

Fig. 7. Optimal models for different hidden unit counts for dataset 2

EFFECT OF LEARNING RATE (1-D)

In order to observe the effect of learning rate in training, the experiments have been conducted on the second dataset using the learning rate values $lr \in (1, 0.1, 0.01, 0.001, 0.0001)$. Here lr denotes the learning rate. To explain the parameter lr, the chnage of a particular weight ω is shown. This change is for

TABLE III
LOSS VALUES OBTAINED WITH DIFFERENT LEARNING RATE VALUES

	Dataset 2			
Learning Rate (lr)	Loss	Epoch	Threshold	Std
lr = 1	0.0895	53	0.90	1.1828
lr = 0.1	0.1100	6173	0.11	0.2225
lr = 0.01	0.3428	73	0.20	0.5242
lr = 0.001	0.0998	4090	0.10	0.2146
lr = 0.0001	0.8017	6286	0.80	1.0982

the i^{th} unit in j^{th} layer.

$$\Delta\omega_i^{[j]} = -lr \times \frac{\partial loss}{\partial \omega_i^{[j]}} \tag{6}$$

The results for the final losses, thresholds and the number of epochs that the mode has been trained is given in table III. For each of the learning rate values, a separate hyper-parameter tuning is performed. The comments included in this section are based on the observed effect of the loss curves. The curves can be found at file *Effect of learning rate (1-D).ipynb*. Threshold values given are identified considering the hyper-parameter search done for part c. The plots of the obtained models optimal for the given hyper-parameters are given in Fig. 8.

Observations: The main result that is observed with this experiment is the effect of the learning rate on the ability of the model to pass certain local minima values. With smaller updates, the learned model seems to be more successful on passing the local optimum points and approaches to global optimum value of the gradients more. This can be seen primarily from the amount of fluctuations and the shape of the loss values obtained. With learning rate values too big, the model struggles to continue its way to find the global optimum point by passing the local minima values. Also it is observed that with greater learning rate values, batch learning performs better where for lower rates it is the opposite case.

This comparison will be provided on section *Effect of Learning Algorithm (1-F)*. In general, learning rate determines the ability of the model of identifying local minimas. However, if the rate becomes too small the model can also fail to find a global optimum value. This issue is also observed in the comparison between the learning rates 0.001 and 0.0001. This fact can be observed from the similarities of the loss plots for these two models. The model with learning rate 0.0001 has been trained by 50000 epochs initially but since the progression is so slow, the training stage has been cut short. This is the main reason why the complexity of the learned model is so low.

EFFECT OF MOMENTUM (1-E)

In order to find the effect of using momentum in the ANN implemented, the model that achieves peak performance in part c is used. Then the momentum is applied with the

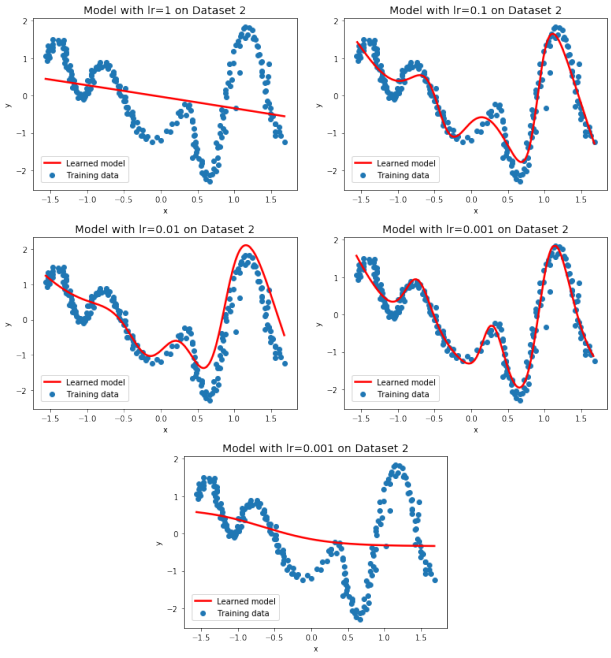

Fig. 8. Optimal models found for varying learning rate values for part d

TABLE IV
COMPARISON OF MODELS USING MOMENTUM AND NOT

	Dataset 2				
Model	Loss	Std	Epoch	Threshold	α
With Momentum	0.0994	0.1788	5637	0.1	0.3
Without Momentum	0.1531	0.2476	8000	0.1	None

optimum α value. By using momentum, the following modification is performed to the update rule:

$$\omega_i^{[j]} = \omega_i^{[j]} + \alpha \times \Delta\omega_i^{[j](t)} + (1-\alpha) \times \Delta\omega_i^{[j](t-1)} \tag{7}$$

In the formulation, $\Delta\omega_i^{[j](t)}$ shows the update found by the gradient of weight $\omega_i^{[j](t)}$ at epoch t. As it can be observed from the formulation, the α parmeter prevents the unexpected gradient jumps in training. By doing this, momentum is expected to fasten the training in terms of passing plateaus involving local minima values for the gradient.
For the optimum network, the threshold is selected as 0.1. The optimum model found for the dataset (with 8 hidden units) uses momentum and the momentum scalar (factor of the most recent gradient) α is selected as 0.3. The optimum model is given in Fig. 9. Depending on using momentum or not, the models are tuned by changing hyper-parameters. The details of the hyper-parameters are provided in file *Effect of momentum (1-E).ipynb*. The results (loss values and the monentum parameters) are given in table IV.

Observations: As it can be observed from the loss plots of training the two models (with momentum and without momentum) and the final results obtained, momentum allows the model to learn faster when plateaus exist. Both the final loss value and the number of epochs that the training continued shows the existence of such plateaus and how momentum gives the advantage to the model of learning them in a more accurate way. Visually, this nature of the gradients shows their existence

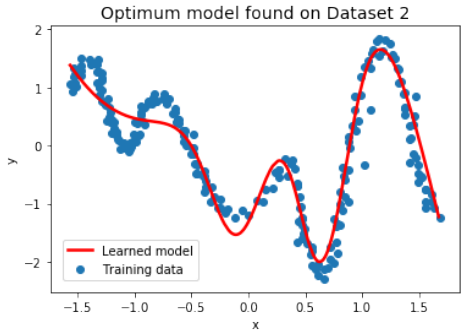

Fig. 9. Optimum Model found with 8 hidden units

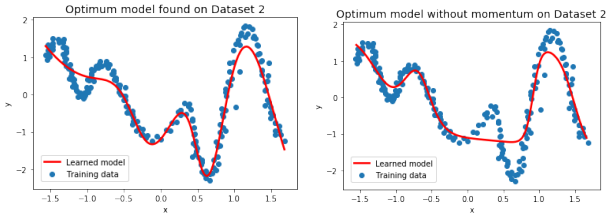

Fig. 10. Plots on training data showing the effect of momentum on learned ANN models

of the second local minimum value observed in input-output pairs. With momentum, ANN model was able to learn this non-linearity whereas the model without momentum struggles to do it.

The plots showing the learned model on training set are given on Fig. 10. For this experiment, the upper limit of epoch count is selected as 8000 and the number of hidden units is fixed to 8.

Effect of learning algorithm (1-f)

As the final part of the assignment, the difference of the batch gradient descent and stochastic gradient descent algorithms are examined. In order to make a comparison and see their limits, two models have been trained where one uses batch learning and other using stochastic gradient descent algorithm. As the main differences in the choices made in hyper-parameter selection are ob the initial weight selection and the learning rate, the differing parameter selections and the results obtained are summarized in Table V.

Observations: Considering the observations regarding batch learning and stochastic learning, different factors are considered while selecting the appropriate hyper-parameters. The initial change was on the selection of the learning rate. Since in batch learning one update occurs each epoch, that update should be big enough such that it can reflect the traversal of all of the training samples. In this regard there should

TABLE V
Comparison of models on learning algorithm

	Dataset 2					
Algorithm	Loss	Std	Epoch	Threshold	lr	ω
Batch	0.1200	0.1675	5857	0.12	0.1	0.0001
Stochastic	0.0991	0.1997	9064	0.10	0.001	0.003

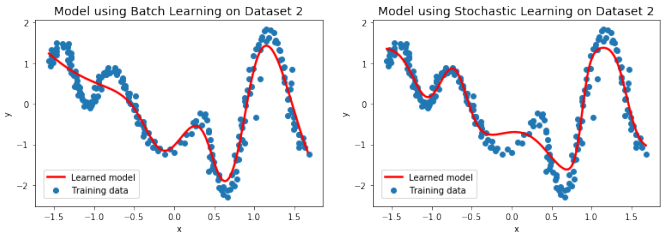

Fig. 11. Resulting models after applying Batch and Stochastic Learning to Dataset 2

be a balance between the learning rates that it should enable fast convergence while not missing the minima points. As the second hyper-parameter, the weight initialization parameter ω tuned. Here the selection for the batch learning is smaller than the one for stochastic learning. The main reasoning behind that is the ability of stochastic learning of tuning the weights more frequently. In the experiments conducted it is observed that stochastic gradient descent take longer time but can converge the weights even they are high up to a reasonable point, whereas batch learning is considerably sensitive to this parameter. About the results obtained from these two models, the ability of fitting to the training data and the convergence speed were the main factors that changed. The observations are given as follows:

- **Convergence Speed**: As it can be observed in Table V, batch learning is faster than stochastic learning in terms of epoch count. The reason behind it was the selection of the learning rate and the applicability of batch learning to such a rate. In the prior experiments it is observed that not every learning algorithm supports every learning rate. Due to this fact, batch learning is considered as a fast algorithm since it is compatible with higher learning rates.
- **Ability of Fitting:** After tuning the learning rate parameter, it is observed that stochastic gradient descent takes more epochs to converge considering the threshold set for loss. Considering the results, and the trials performed it is observed that with good hyper-parameters, stochastic learning can achieve lower loss values compared to batch learning. As a result of this observation, stochastic learning is observed as a more powerful algorithm than batch learning in terms of the ability of making a fit since it takes smaller steps each time.

The optimal ANN models found for stochastic learning and batch learning are given in Fig. 11. The plots of the changes in the loss values are given in *Effect of learning algorithm (1-F).ipynb* file with necessary explanations.

References

[1] Ç. Gündüz Demir, *Neural Networks - CS 550: Machine Learning*, URL: http://www.cs.bilkent.edu.tr/~gunduz/teaching/cs550/documents/CS550_NeuralNetworks.pdf.