Neural Networks and Learning Systems : Homework II

Scribed by
Varun Krishna, Praneeth Vonteddu, Shubham Kumar

Question 1.

MultiClass Logistic Regression

Given training samples $(x_i, y_i)_{i=1}^N$ where $x_i \epsilon R^d$ and $y_i \epsilon \{1, 2....K\}$.
Here, we make a fixed nonlinear transformation of the inputs using a vector of basis functions $\phi(x)$. The resulting decision boundaries are thus linear in feature space ϕ, and correspond to nonlinear decision boundary in original x space.

For K classes, we use softmax function instead of logistic sigmoid, also known as softmax regression.

$$p(C_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\Sigma_j \exp(a_j)}$$

where the activations $a_k = w_k^T \phi$, $k = \{1, 2....K\}$ and the weight vectors $w_k = [w_{k1}, w_{k2}, ..., w_{kM}]^T$ and $a = \{a_1, a_2, ..., a_k\}$.

We learn a set of K weight vectors $\{w_1, w_2, ..., w_k\}$.

These weight vectors can be arranged as a matrix W.

$$W = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ w_k \end{bmatrix} = \begin{bmatrix} w_{11} & . & . & w_{1M} \\ w_2 & . & . & w_{2M} \\ . & . & . & . \\ . & . & . & . \\ w_k & . & . & w_{kM} \end{bmatrix}$$

To estimate the weight vectors we use Maximum likelihood. For estimating maximum likelihood we need the derivatives of y_k with respect to all activations a_j. These are calculated as

$$\frac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j)$$

where I_{kj} are the elements of the identity matrix.

I will represent the classes $C_1, ..., C_k$ as one hot vector. This can be done by using 1-of-K coding scheme in which the target vector t_n for a feature vector ϕ_n belonging to class C_k is a binary vector, with all elements zero except for the element k.

Class C_k is a K dimension vector, where $C_k = [t_1, ..., t_k]^T$, $t_i \epsilon 0, 1$.
These class probabilities obey $\Sigma_{k=1}^K p(C_k) = \Sigma_{k=1}^K t_k = 1$.

Classes have values 1,..,K and we have N labeled samples. Target matrix is a N x K matrix with elements t_{nk}.

$$T = \begin{bmatrix} t_{11} & . & . & t_{1K} \\ . & . & . & . \\ . & . & . & . \\ t_{N1} & . & . & t_{NK} \end{bmatrix}$$

The likelihood function can be written as

$$p(T|w_1, .., w_K) \; = \; \Pi_{n=1}^{N} \Pi_{k=1}^{K} \; p(C_k|\phi_n)^{t_{nk}} \; = \; \Pi_{n=1}^{N} \Pi_{k=1}^{K} \; y_{nk}^{t_{nk}}$$

where $y_{nk} \; = \; y_k(\phi_n)$ and T is the target matrix.
Taking the negative log-likelihood gives

$$E(w_1, .., w_K) \; = \; -ln \, p(T|w_1, .., w_K) \; = \; -\Sigma_{n=1}^{N} \Sigma_{k=1}^{K} \, t_{nk} \, ln \, y_{nk}$$

This is our objective function, known as Cross entropy function for multi class classification.

Computing Gradient of the error function wrt the parameter vector w_j,

$$\Delta_{w_j} E(w_1, .., w_K) = \Sigma_{n=1}^{N} \, (y_{nj} - t_{nj}) \, \phi_n$$

where $\Sigma_k \, t_{nk} = 1$ is used. This gradient is basically the product of error $(y_{nj} - t_{nj})$ and feature vector ϕ_n. This gradient has the same form as the sum of squares error for the linear model, and the cross entropy error for logistic model.

This gradient can be used to formulate a sequential algorithm, in which patterns are presented one at a time, and weight vector is updated as

$$w^{(n+1)} \; = \; w^{(n)} - \eta \, \Delta E_n$$

This completes the solution for Online algorithm.

For Batch Method,
We can use the Newton-Raphson update to obtain the corresponding IRLS algorithm for the multi class problem. So, we need to evaluate the Hessian matrix of size M x M in which each block j,k is given by

$$\Delta_{w_k} \, \Delta_{w_j} E(w_1, .., w_K) = -\Sigma_{n=1}^{N} \, y_{nk} (I_{kj} - y_{nj}) \, \phi_n \phi_n^T$$

The hessian for multi class logistic regression is positive definite. Hence, we can achieve a unique minimum for error function.

Question 2.

Problem 4.2 : How does the learning-rate parameter η in back-propagation algorithm affect the trajectory in weight space and the rate of learning ? Is there any method to avoid the danger of instability of the algorithm?

Solution

Let w_{ij}^k be the weight connecting the i^{th} neuron of $k-1^{\text{st}}$ layer with j^{th} neuron of the k^{th} layer.

Update Equation :

$w_{ij}^k(t) = w_{ij}^k(t-1) - \eta \frac{\partial E}{\partial w_{ij}^k(t-1)}$
η is the learning rate

Effect of having a very small learning rate :

If η is small , the correction term $\eta \frac{\partial E}{\partial w_{ij}^k(t-1)}$ will also be small, leading to $w_{ij}^k(t) \approx w_{ij}^k(t-1)$. Thus convergence is slow and weight trajectory takes small strides at each updates.

Effect of having a large value of learning rate :

- Having a reasonably large values of learning rate increases the convergence rate, However
- Very large values of learning rate causes the weights to explode to a very high values and algorithm diverges

Steps to decide optimal learning rate :

- Empirical way to choose optimal learning rate is to perform a grid search on different learning rates and choose the learning rate that performs best on the validation set.
- However if the cost function is **quadratic** then , Hessian of cost function helps to decide good learning rates.
- If $\sigma_{\max}$ and $\sigma_{\min}$ are the max and min eigenvalues of the Hessian matrix , then learning rate $\eta = \frac{2}{\sigma_{\max}+\sigma_{\min}}$, guarantees linear convergence.
- If the cost function is not Quadratic , then using 2^{nd} order Taylor series approximations of the cost function and computing hessian to determine the values for learning rate may help in convergence.

Problem 4.3 : The Momentum constant α is normally assigned a positive value in the range $0 < \alpha \leq 1$. Investigate the difference that would be made in the behaviour of $\Delta w_{ij}(n) = -\eta \sum_0^n \alpha^{n-t} \frac{\partial \epsilon(t)}{w_{ij}(t)}$ with respect to time t if α were assigned a negative value in the range $-1 < \alpha \leq 0$.

Solution

If α's were assigned negative values, then replace α by $-\alpha$ in the equation given in the question , then the equation becomes
$\Delta w_{ij}(n) = -\eta \sum_0^n (-1)^{n-t} \alpha^{n-t} \frac{\partial \epsilon(t)}{w_{ij}(t)}$
Clearly we see that $(-1)^{n-t}$ alternates in sign every iterations. Thus if derivative $\frac{\partial E}{\partial w_{ij}}$ has the same algebraic sign on the consecutive iterations of algorithm, then the magnitude if exponentially weighted sum is reduced.If derivative $\frac{\partial E}{\partial w_{ij}}$ has alternating signs on consecutive iterations, the exponentially weighted sum increases in magnitude.
Thus the effect of having negative α is reverse as that of having positive α.

Question 3.

Goal

We want to classify the Iris dataset into respective classes based on the labels with a Multi Layer Perceptron.

Iris dataset has 4 features namely *sepal length,sepal width,petal length,petal width* and three labels **Setosa , Virginica , Versicolor**.

Network Architecture

Based on the low complexity of the problem we'll fix one hidden layer and start experimenting with the following architecture of MLP.

1. This network has 3 layers 1 input layer ,1 hidden layer, 1 output layer.
2. Input layer has four nodes one for each feature.
3. Hidden layer has 'h' nodes.
4. The labels are encoded with **one-hot encoding** scheme.
5. Hence the output layer has 3 nodes.
6. Activation function at hidden layer is experimented with *tanh* and logistic functions.
7. Activation function at output layer is taken to be logistic.
8. Optimization is done by **MMSE**(Minimum Mean Square Error) criteria.

Hyperparameter Tuning

- We want to estimate the optipum number of hidden neurons for our network.
- Hence we perform experiments with a particular random split of data into train and test sets and the results are tabulated.

Hidden layer Activation	Hidden neurons	Epochs for convergence	Train Accuracy	Test Accuracy
logistic	1	3450	98.33	100
logistic	2	279	98.33	100
logistic	3	136	98.33	96.66
logistic	4	175	98.33	100
logistic	5	160	98.33	96.66

Hidden layer Activation	Hidden neurons	Epochs for convergence	Train Accuracy	Test Accuracy
tanh	1	401	98.33	100
tanh	2	66	98.33	100
tanh	3	47	98.33	100
tanh	4	39	98.33	96.66
tanh	5	59	98.33	96.66

Conclusions

1. We see that any network with more than 2 hidden neurons converges almost in similar number of epochs and much faster than a network with 2 or less hidden neurons.
2. Hence we fix 3 as the optimum number of hidden neurons to lower computational complexity.

Training

1. The Iris dataset is divided into training smaples (80 percent) and test samples(20 percent).
2. The features are normalized to lie in range [0,1]
3. The network is operated in Online mode and Batch mode which use **Stochastic Gradient Descent** and **Gradient Descent** algorithms respectively for training the network i.e. updating the weight matrices and bias vectors.
4. The weight matrices are initialized with small random numbers and the bias vectors are initialized with zeros.
5. Stopping criteria was set to 98 percent accuracy.

Observations

1. For different random splittings of data into train and test sample , we get different results.
2. Online mode runs significantly faster than batch mode.

Testing

1. Test accuracy is obtained between 89 percent and 100 percent for different splits of data into train and test samples and different activaion functions.
2. There is no order between online or batch and between tanh or logistic when comparing test accuracies.
3. Although in batch mode test accuracy was more close to train accuracy than in online mode.
4. Test accuracy is higher than train accuracy some times.
5. This might be due to the fact that we have very low test samples.

Error Trajectories

Online Mode

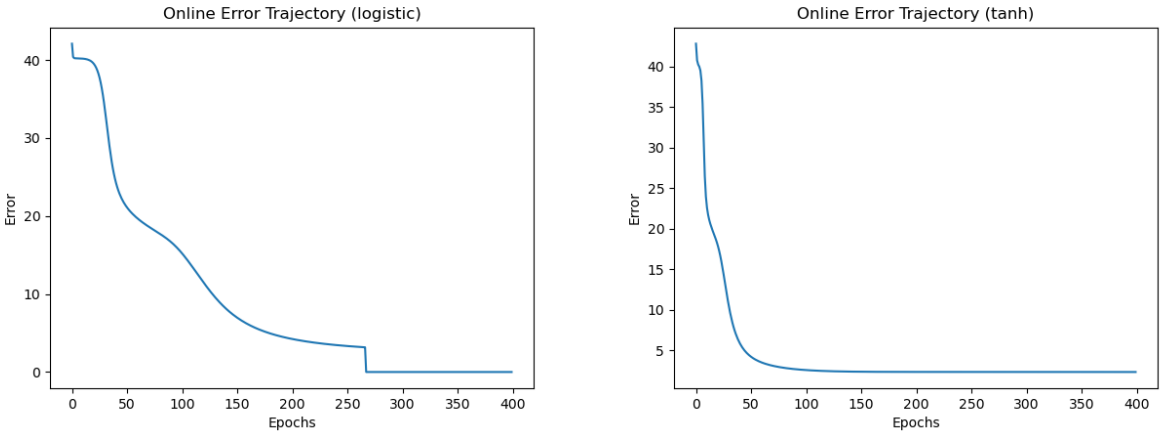

Figure 1: Error Trajectories with learnrate=0.1

Batch Mode

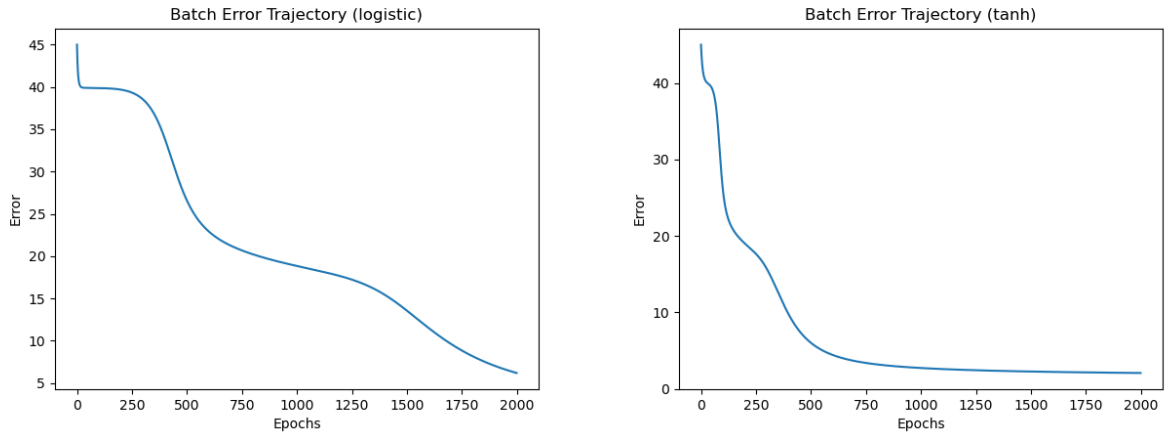

Figure 2: Error Trajectories with learnrate=1

Decision Boundaries

For plotting decision boundaries we first drop a feature with low variance which in this case is *sepal width*

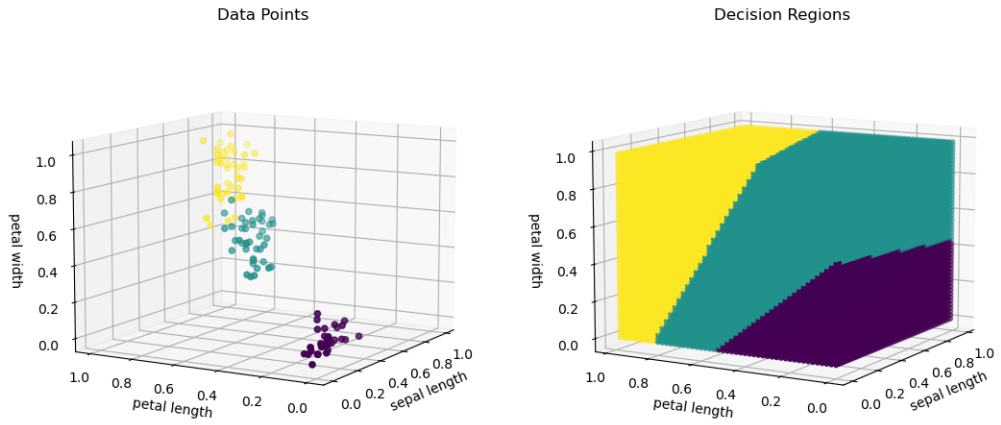

Figure 3: Online Mode Decision Boundary(tanh) after convergence (236 epochs)

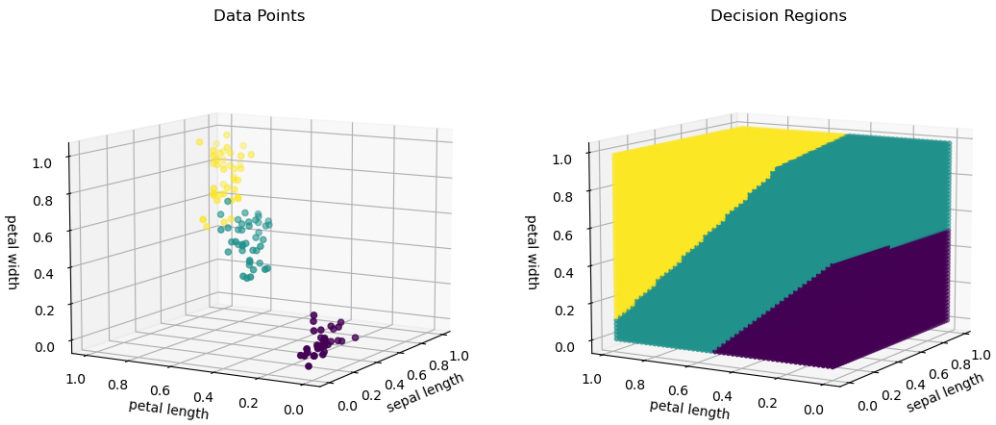

Figure 4: Batch Mode Decision Boundary(tanh) after 2000 epochs

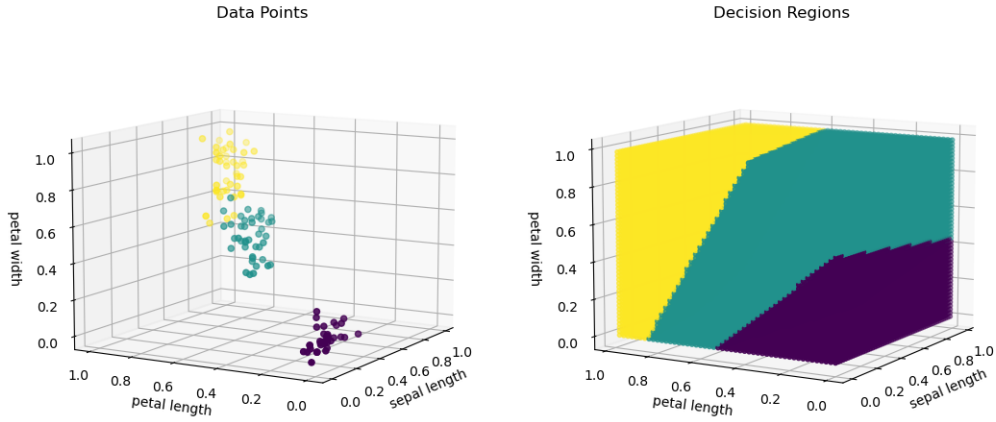

Figure 5: Online Mode Decision Boundary(logisitc) after convergence(326 epochs)

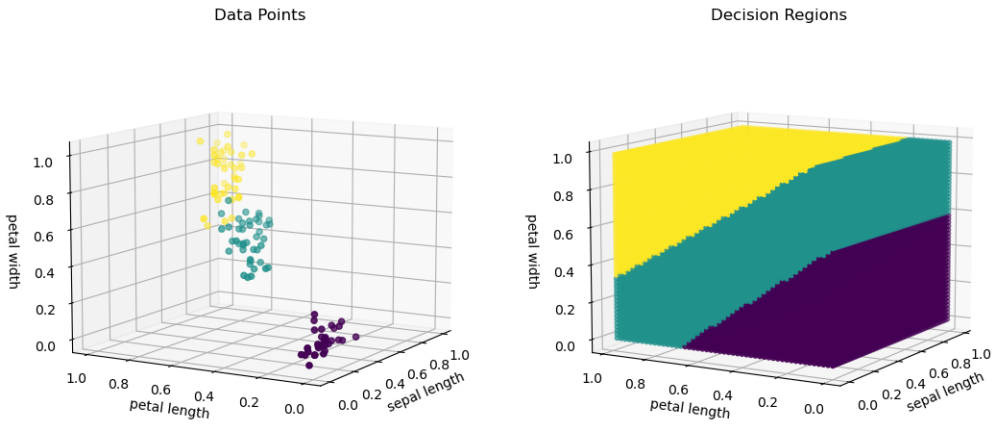

Figure 6: Batch Mode Decision Boundary(logistic) after convergence (1781 epochs)

Observations

1. Online Mode decision boundaries are similar for both tanh and logistic activation functions.
2. Batch Mode decision boundaries are also similar for both tanh and logistic activation functions.
3. Batch mode takes significantly more epochs than Online mode for convergence.

Data Shuffling

Online Mode

- Convergence is obtained faster by data shuffling.
- Error trajectory gets a bit noisy with data shuffling.

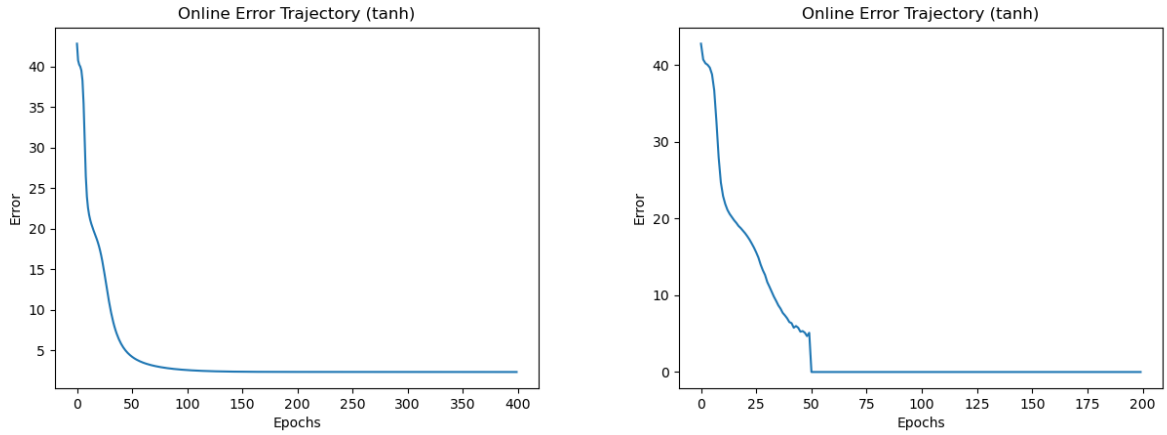

Figure 7: Error Trajectories before and after data shuffling(tanh)

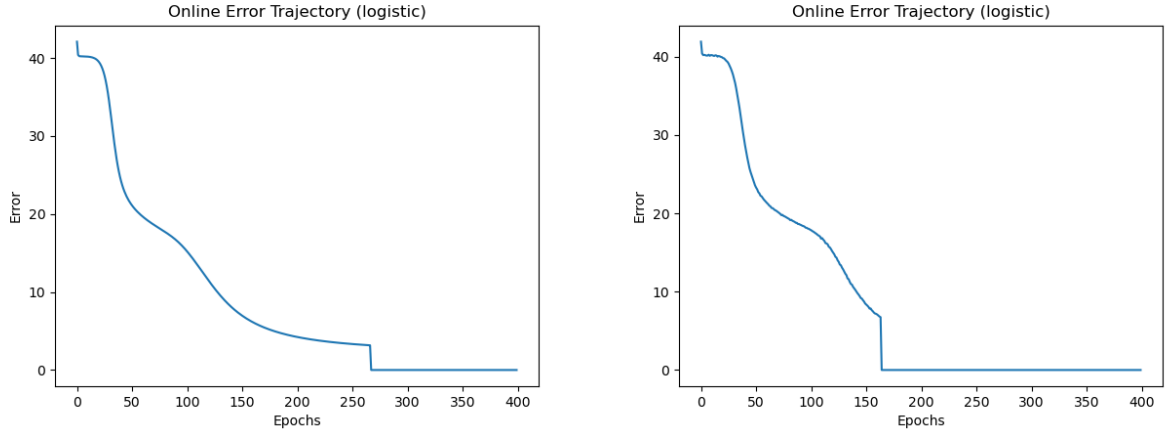

Figure 8: Error Trajectories before and after data shuffling(logistic)

Batch Mode

Data shuffling will not have any impact in Batch mode as we are eventually adding all the errors and updating the parameters only once an epoch.

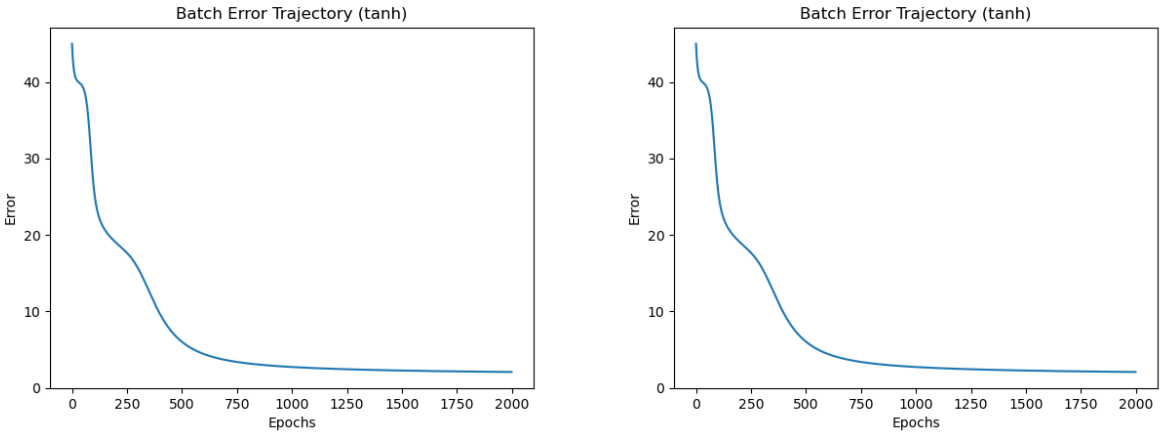

Figure 9: Error Trajectories before and after data shuffling(tanh)

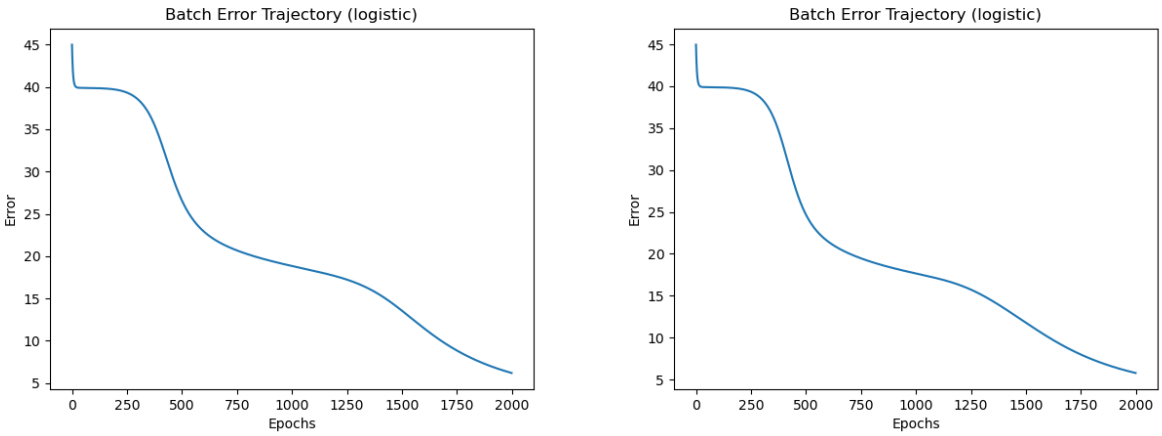

Figure 10: Error Trajectories before and after data shuffling(logistic)