Neural Network Project 1: Multilayer Perceptron

Zhicong Lu, Boxuan Zhang

1. Problem Definition

1.1. Multilayer Perceptron

A Multilayer Perceptron (MLP) is a type of artificial neural network, which is usually used for solving supervised learning tasks of classification and regression.

The MLP consists of one or more layers of artificial neurons, also known as perceptrons, which are connected in a feedforward manner. Each neuron in the MLP takes a set of inputs which are usually a 2D matrix of real number. Then neuron performs a linear transformation on input and applies a non-linear activation function to the result. Finally, it passes the output as new input to the next layer of neurons until the last layer. In the last layer, output will go through different types of function depending on the type of task, whether it's a classification or regression task.

For classification tasks, the last layer of the MLP usually consists of a set of output neurons equal to the number of classes to be predicted. Each output neuron corresponds the probability that the input belongs to a class. To convert the outputs into probabilities, the softmax activation function is typically applied. The softmax function $P(y = j|x) = \frac{e^{W_j x}}{\sum_{j=0}^{N} e^{W_j x}}$, where $\boldsymbol{N}$ equals to the number of classes and j represents each class. We can recognize that softmax function takes vector of real number and normalizes it to $\boldsymbol{P}$, which belongs to (0,1) and sum to 1 so it can represent a probability distribution.

For regression tasks, the last layer of the MLP usually consists of neurons with no activation function, so the output can take on any real number.

1.2. Training Process

The training process of a MLP model is the process of iteratively adjusting the parameters of each perceptron to minimize the difference between the predicted output and the true result.

The loss function is to measure the difference between the predicted output of the MLP and the true result. For classification tasks, the cross-entropy loss function is commonly used, and for regression tasks, the mean squared error loss function is commonly used.

The first step is to initialize MLP with random or specific initial values. By using these values MLP can make predictions on training dataset. Then we use optimization algorithm like gradient descent to adjust parameters of each perceptron and minimize the loss function.

The backpropagation algorithm is efficient to compute the gradient in neural network. By

chain rule, it will compute the gradient one layer at a time using the result of former layer and iterate backward from the last layer to input layer.

2. Implementation

In this project, we implemented a pyTorch-like automatic differentiation Tensor library to construct the neural network. Each Tensor object A is constructed by 3 parameters: values(ndarray), required_grad(bool) and dependencies(list). Each dependency is an object, whose "tensor" field is an input tensor B_i of A and "grad_fn" field is the function (grad_of_A)=>(additional grad_of_Bi from A).
We only need to define some basic operations of Tensor, and leave the differentiation done automatically by the Tensor.

For example, if we want to define matrix multiplication C = W @ X, We construct the Tensor C from W and X:

```
C.values = W.values * X.values
C.required_grad = W.required_grad or X.required_grad
C.dependencies = [
    {
        "tensor": W,
        "grad_fn": lambda gradC: gradC @ X.values.T
    },
    {
        "tensor": X,
        "grad_fn": lambda gradC: W.values.T @ gradC
    },
]
```

Where "@" is the matmul operator and .T is the transpose operation.

By defining such matrix multiplication and other operations like plus, scalar multiplication and log, the forward and backward operations of neural network can be calculated easily. We mark all the trainable Parameter (a subclass of Tensor). For each input data batch, we reconstruct the whole Network with the input Tensor and saved Parameters, call backward from the loss function to calculate grads for Parameters, and update them.

3. Dataset Description

In this project we choose 2 different datasets (one classification and one regression task) on UCI machine learning repository, which are ***Breast Cancer Wisconsin (Diagnostic) Data Set***[1] and ***Yacht Hydrodynamics Data Set***[2].

---

[1] https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29, Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science

[2] https://archive.ics.uci.edu/ml/datasets/Yacht+Hydrodynamics, Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

- The Breast Cancer Wisconsin (Diagnostic) Data Set (WDBC) is a classification task of 2 classes to predict diagnosis result. The dataset consists of 569 instances, 32 attributes and no missing attributes. We use 30 real-valued attributes as input features, and the categorical attribute (Diagnosis) as label. According to label, there are 357 instances belongs to "B" class and 212 instances belongs to "M" class.

- The Yacht Hydrodynamics Data Set is a regression task to predict resistance of sailing yachts. The dataset consists of 308 instances, 7 attributes and no missing attributes. We use 6 attributes of real number as input features and 1 real-valued attributes (Residuary resistance per unit weight) as label.

4. Program Manual

4.1. Example Usage

This project is programmed in Python 3.
Use this command to run train and test program:

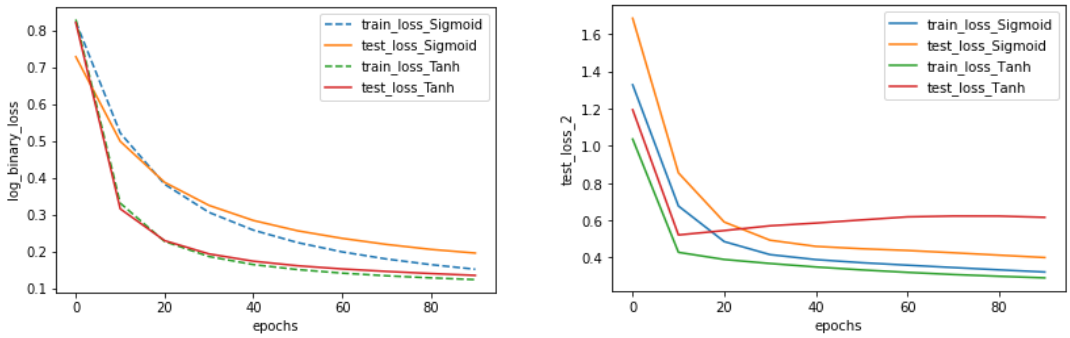
```
$ python main.py --input_data=./data/yacht.data --num_units=32,16 \
                 --loss_func=square_loss --activation_func=sigmoid --batch_size=16 \
                 --num_epochs=50 --learning_rate=1e-2 --momentum=0.9 \
                 --l2_norm=1e-3 --test_ratio=0.2
$ python main.py --input_data=./data/BreastCancer.data --num_units=32,16,8 \
                 --loss_func=log_binary_loss --activation_func=sigmoid --batch_size=16 \
                 --num_epochs=50 --learning_rate=1e-2 --momentum=0.9 \
                 --l2_norm=1e-3 --test_ratio=0.2
```

4.2. Hyperparameters

- --input_data : Dataset filename, string, **required**.
- --num_units : The size of hidden layers, integers separated by ',' character, **required**.
- --loss_func : Loss function, "square_loss" and "log_binary_loss" available, "square_loss" for regression tasks and "log_binary_loss" for classification tasks, **required**.
- --activation_func : The activation function of perceptrons, "sigmoid" and "tanh" is available, "sigmoid" by default.
- --batch_size : Batch size, integer, 16 by default
- --num_epochs : The number of training epochs, integer, 50 by default.
- --learning_rate : Learning rate, float, 0.01 by default.
- --momentum : Momentum of optimizer, float, 0.9 by default.
- --l2_norm : L2 normalization coefficient, float, 0.001 by default.
- --test_ratio : The ratio we split test dataset from entire dataset, float, 0.2 by default.

5. Experiments and Analysis

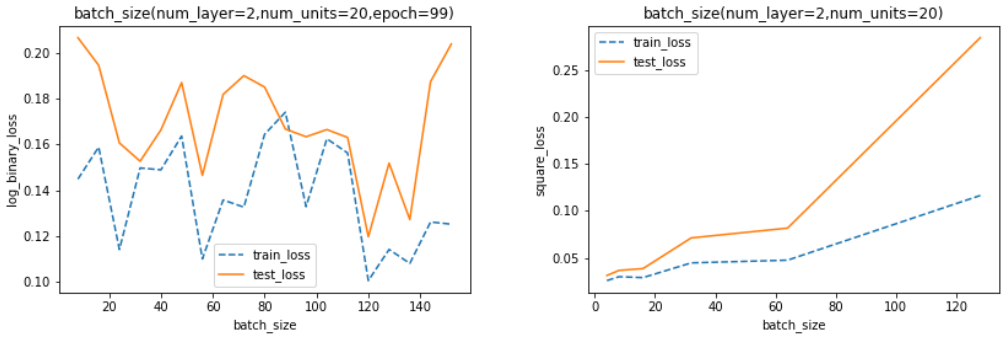
5.1. Activation Function



Left : Classification task (WDBC); Right: Regression task (yacht)

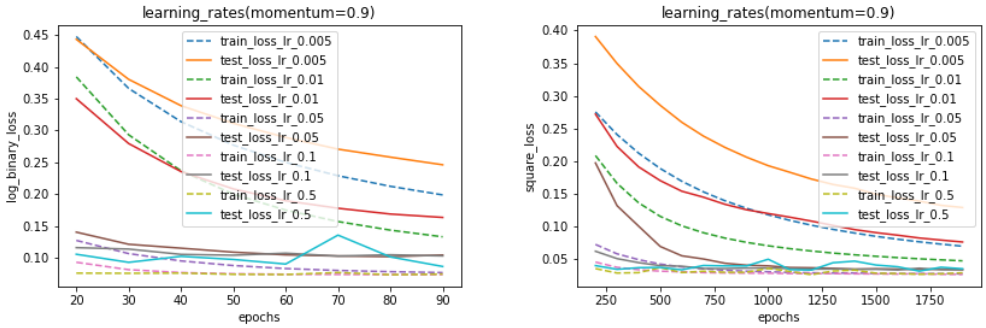
We can notice that in train dataset, tanh activation function performs better than sigmoid activation function in both tasks. In test dataset of classification task, using tanh activation function still performs better. However, in test dataset of regression task, the result of tanh activation function is worse. The reason is that result may differ according to different dataset.

5.2. Batch Size



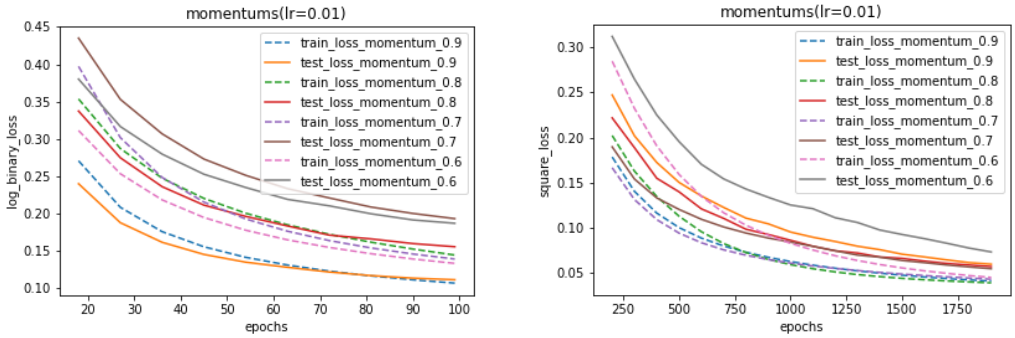
Left : Classification task (WDBC); Right: Regression task (yacht)

5.3. Learning Rates



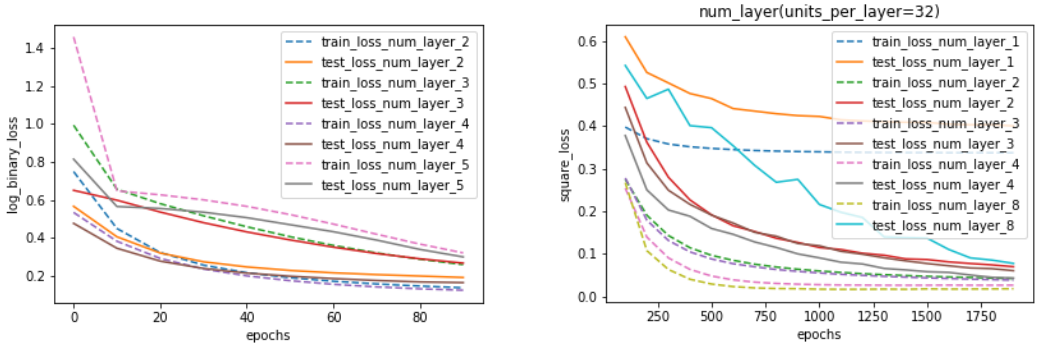
Left : Classification task (WDBC); Right: Regression task (yacht)

5.4. Momentum



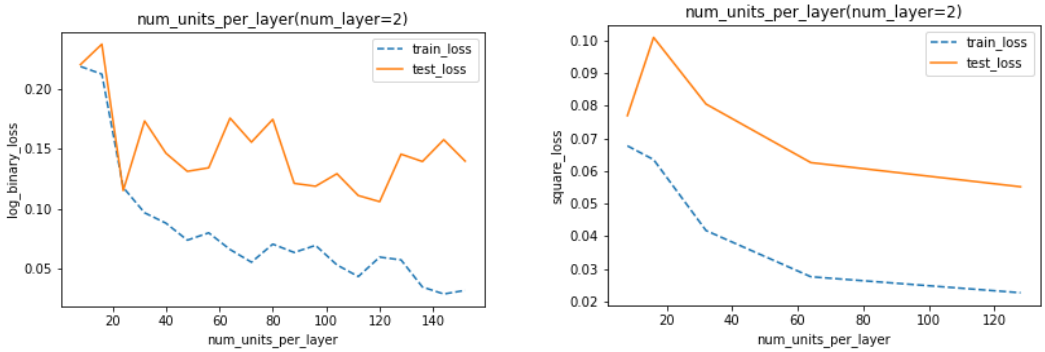
Left : Classification task (WDBC); Right: Regression task (yacht)

5.5. Layers



Left : Classification task (WDBC); Right: Regression task (yacht)

5.6. Hidden Layer Size



Left : Classification task (WDBC); Right: Regression task (yacht)

6. Future Work

The possible future work is to implement different kinds of layer including recurrent layers and the convolutional layers. In addition, the activation functions, operators, optimization algorithm in our project is also limited, so implementation of more function is also required in future.

We could also compare our result with other machine learning algorithm like linear/ logistic regression, gradient boot decision tree etc.