Lab1 - Simple Neural Network

Introduction

In this lab, I implement a MLP (Multi-Layer Perceptron), and then, I train and tested it on linear data and XOR data. This model achieved ≥ 99% and ≥ 90% accuracy respectively.

Implementation Details

Network Design

- **Initialization**

According to the given layer information, establish the neural networks by assigning random weights and bias for every layers.

```
for i in range(1, len(listOfLayers)):
    self.layers[i]['w'] = np.random.randn(listOfLayers[i-1], listOfLayers[i])
    self.layers[i]['b'] = np.random.randn(1, listOfLayers[i])
```

- **Forward**

$$z = XW + b$$

$$a = \frac{1}{1 + \exp(-z)}$$

```
self.layers[i]['z'] = np.dot(self.layers[i-1]['a'], self.layers[i]['w']) +
                      self.layers[i]['b']
self.layers[i]['a'] = self.activation(self.layers[i]['z'])
```

- **Back Propagation**

First, calculate the partial derivative ∂J/∂a based on the prediction and the labels.

$$\frac{\partial J}{\partial a} = -(\frac{y}{a} - \frac{1 - y}{1 - a})$$

```
self.layers[-1]['dJda'] = -y/self.layers[-1]['a'] +
                          (1.0-y)/(1.0-self.layers[-1]['a'])
```

Second, from the last layer to the first layer, calculate partial derivative ∂J/∂z, ∂J/∂b and ∂J/∂w based on the previously calculated ∂J/∂a for each layer according to the chain rule.

$$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial J}{\partial a} \times a(1-a)$$

$$\frac{\partial J}{\partial b} = \frac{1}{n}\sum \frac{\partial J}{\partial z}$$

$$\frac{\partial J}{\partial w} = a \times \frac{\partial J}{\partial z}$$

$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial Z} \times W$$

```
self.layers[i]['dJdz'] = self.inverseActivation(self.layers[i]['a']) *
                         self.layers[i]['dJda']
self.layers[i]['dJdb'] = np.mean(self.layers[i]['dJdz'], axis=0)
self.layers[i]['dJdw'] = np.dot(self.layers[i-1]['a'].T,self.layers[i]['dJdz'])/bs
self.layers[i-1]['dJda'] = np.dot(self.layers[i]['dJdz'], self.layers[i]['w'].T)
```

- **Update**

After back propagation, update the weights and the biases.

```
self.layers[i]['w'] -= learning_rate * self.layers[i]['dJdw']
self.layers[i]['b'] -= learning_rate * self.layers[i]['dJdb']
```

Training

In every epoch:

1. Shuffle the training data
2. Split the training data into batches
3. Forward → Back propagation → Update

```
shuffled_trainX, shuffled_trainY = self.shuffle(trainX, trainY)
for i in range(trainX.shape[0] // bs):
      x = shuffled_trainX[i*bs:(i+1)*bs, :]
      y = shuffled_trainY[i*bs:(i+1)*bs, :]
      self.yHat = self.forward(x)
      self.lossList.append(self.loss(y))
      self.backProp(y, bs)
      self.update(lr)
```

Experimental Result

Train & Test on Linear Data

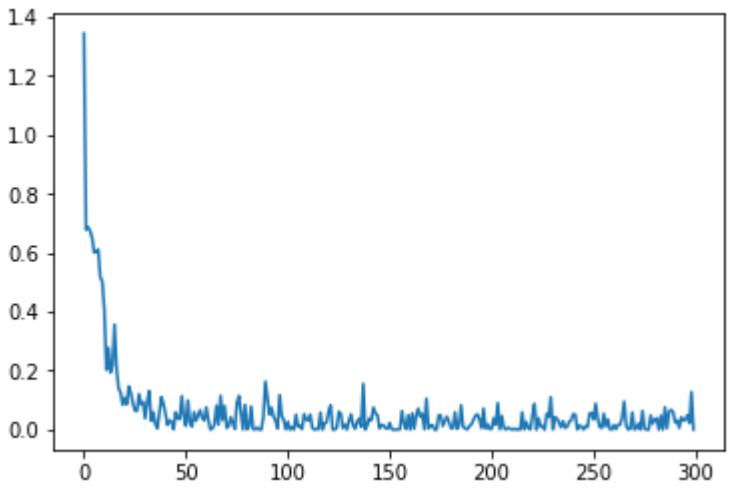
Setup

- Layers: [2, 3, 3, 1]
- Training data size: 100,000
- Testing data size: 10,000
- Epoch: 3
- Learning rate: 0.1 → 0.01 → 0.001
- Batch size: 10
- Seed: 0

Evaluation

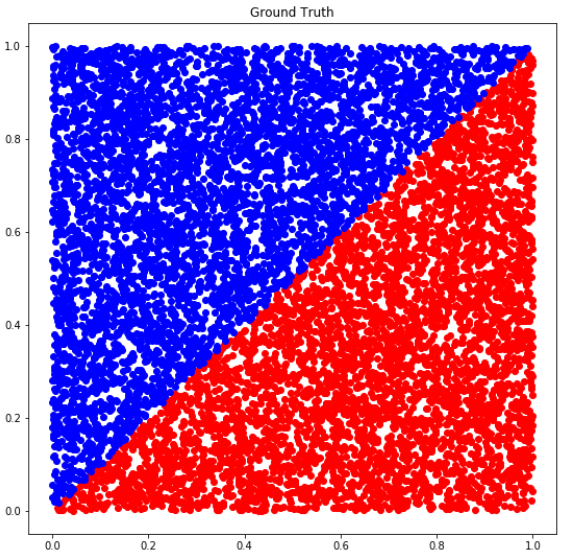
- Accuracy: 0.9992
- Recall: 0.9984
- Precision: 1.0
- F1 score: 0.9992

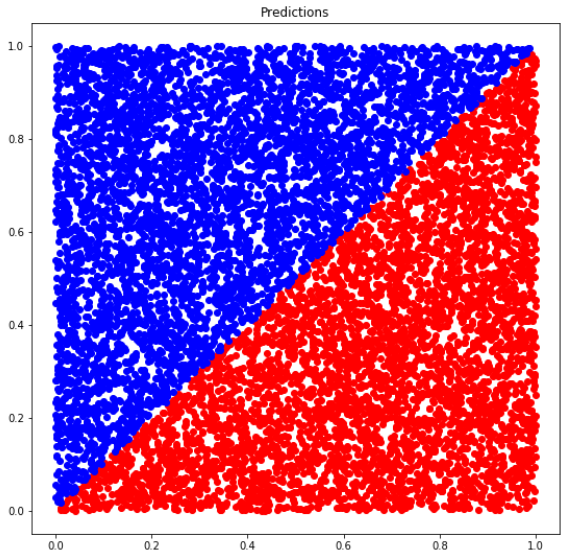
Loss



Visualization

Ground Truth & Predictions





Train & Test on XOR Data

Setup

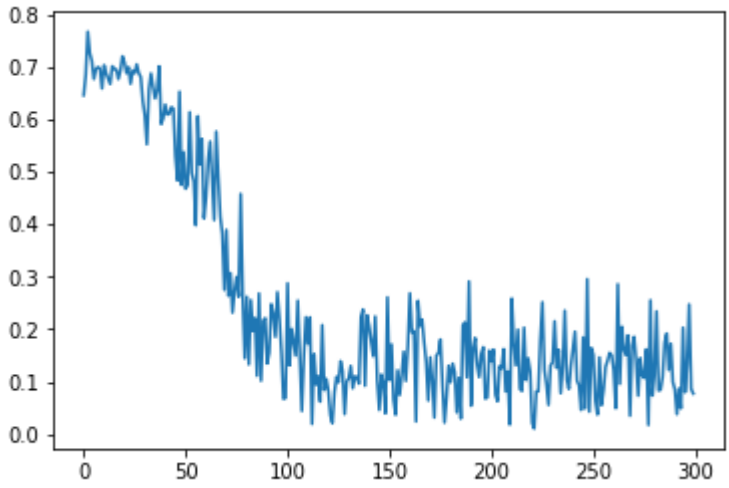
- Layers: [2, 5, 5, 1]
- Training data size: 100,000
- Testing data size: 10,000
- Epoch: 3
- Learning rate: 0.1 → 0.01 → 0.001
- Batch size: 10

- seed: 0

Evaluation

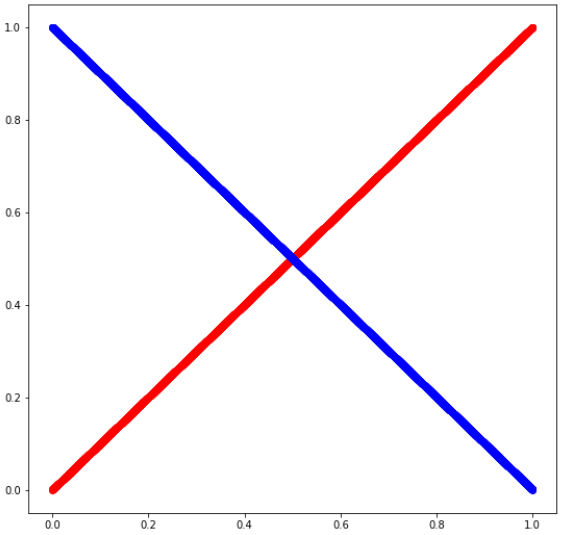
- Accuracy: 0.9865
- Recall: 0.973
- Precision: 1.0
- F1 score: 0.9863

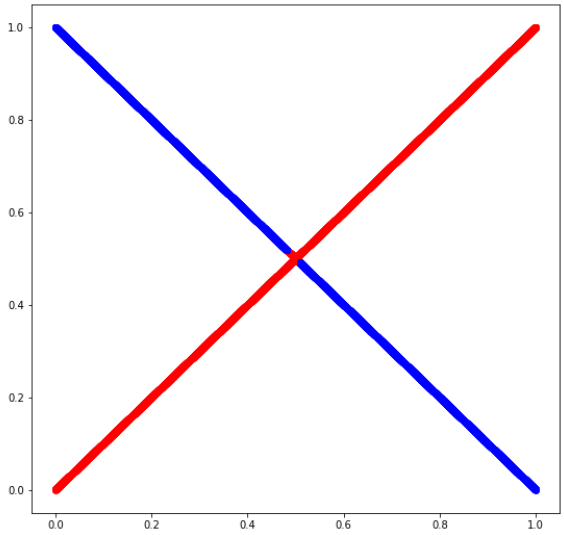
Loss



Visualization

Ground Truth & Predictions





Discussion and Extra Experiments

Different Network Design

In this section, I am going to discuss how different network designs affect the training results of these two type of training data. I design three kind of networks (smaller, medium, larger) and use 100 different seeds to initialize the networks, and then, based on the mean and the variance of the result accuracy, we can know the performance and the stability of the network on the data.

Smaller Network - [2, 1]

- Linear data:
  - Mean: 0.9991
  - Var: 2.67 * 10^-7
- XOR data:
  - Mean: 0.5028
  - Var: 0.0069

Medium Network - [2, 3, 1]

- Linear data
  - Mean: 0.9993
  - Var: 2.47 * 10^-7
- XOR data:

- Mean: 0.8679
- Var: 0.0132

Larger Network

- Linear data
  - Mean: 0.9992
  - Var: 2.72 * 10^-7
- XOR data:
  - Mean: 0.8679
  - Var: 0.0132

According to the statistics above, we can observe that:

1. Because Linear data is linear separable, all of the three kind of networks perform well in this kind of data. Also, using larger network cannot enhance the performance much.
2. Because XOR data is not linear separable, the result of using smaller network with this kind of data is almost random guess. The result of using medium network with this kind of data is not so bad, but it is vulnerable to the initial state of the network. The result of using larger data is much better in both of performance and stability. Therefore, we can see that if we want to train non-linear separable data, larger network is needed.