Assignment_1 Report

Performance comparison and analysis on Perceptron with different activation functions and loss functions.

Project Description

This report aims to implement two simple neural networks – Perceptron for image classification. Perceptron is the simplest feedforward neural network that doesn't contain any hidden layer. Two group activation functions and loss functions are shown in Figure 1 (a) for group 1 and Figure 1 (b) for group 2, respectively.

Group 1:

✓ Activation function:

$$a_{w,b}(x^i) = \frac{1}{1+e^{-(w^T \times 5x^i + 3b)}}$$

✓ Loss function:

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\{(y^i - \hat{y^i})^2\}$$

FIGURE 1 (a) Activation function and loss function for group 1

✓ Loss function:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\{(y^i - \hat{y^i})^2\}$$

Group 2:

✓ Activation function:

$$a_{w,b}(x^i) = \frac{1}{1+e^{-(w^T x^i + b)}}$$

FIGURE 1 (b) Activation function and loss function for group 2

Training dataset (train_catvnoncat.h5) and testing dataset (test_catvnoncat.h5) are provided. The hyper-parameters for both perceptron: learning rate: 0.01, iterations: 2000. Results will show the difference between training accuracies and testing accuracies. The differences in learning curve plots for two Perceptron will be compared and analyzed as well.

Model Description

Figure 1 shows the activation function and loss function, which were used to calculate the error between ground truth and predicted value. By improving the prediction accuracy and reducing the error, backpropagation has an important role in this process in getting weight and bias. The equation of updated weight and bias show as below:

w_new = w_old + learning rate * dw

b_new = b_old + learning rate * db

Derivatives of weight and bias are used to calculate new weight and new bias. After that, new weight and new bias will keep being continuously applied to the activation function and loss function again to get another weight and bias until the loss function is close to zero which means the predicted value has a better fit to the ground truth.

Figure 2 (a) shows the backpropagation calculation process and Figure 2 (b) show the backpropagation in the coding part for group 1. Figure 3 (a) shows the backpropagation calculation process and Figure 2 (b) show the backpropagation in the coding part for group 2.

Group 1

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w}$$
$$= \left[\frac{1}{2m} \sum_{i=1}^{m} 2 \cdot (y-\hat{y}) \cdot (-1)\right] \cdot \left[\hat{y} \cdot (1-\hat{y}) \cdot 5x\right]$$
$$= \frac{-5}{m} \sum_{i=1}^{m} (y-\hat{y}) \cdot \hat{y} \cdot (1-\hat{y}) \cdot x$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b}$$
$$= \frac{-2}{2m} \sum_{i=1}^{m} (y-\hat{y}) \cdot \left[\hat{y} \cdot (1-\hat{y}) \cdot 3\right]$$
$$= \frac{-6}{2m} \sum_{i=1}^{m} (y-\hat{y}) \cdot \hat{y} \cdot (1-\hat{y})$$
$$= \frac{-3}{m} \sum_{i=1}^{m} (y-\hat{y}) \cdot \hat{y} \cdot (1-\hat{y})$$

FIGURE 2 (a) Backpropagation for weight and bias of group 1

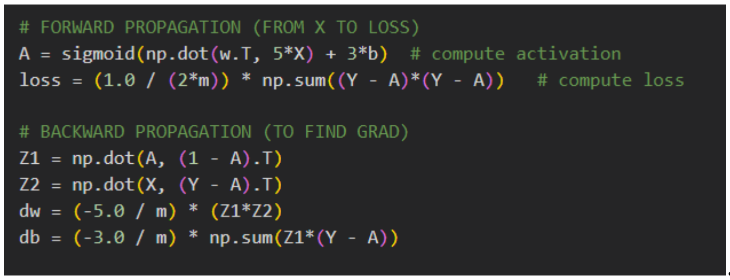

```
# FORWARD PROPAGATION (FROM X TO LOSS)
A = sigmoid(np.dot(w.T, 5*X) + 3*b)  # compute activation
loss = (1.0 / (2*m)) * np.sum((Y - A)*(Y - A))   # compute loss

# BACKWARD PROPAGATION (TO FIND GRAD)
Z1 = np.dot(A, (1 - A).T)
Z2 = np.dot(X, (Y - A).T)
dw = (-5.0 / m) * (Z1*Z2)
db = (-3.0 / m) * np.sum(Z1*(Y - A))
```

FIGURE 3 (b) Backpropagation for weight and bias of group 1 in coding

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w}$$
$$= \left[\frac{1}{m} \sum_{i=1}^{m} 2 \cdot (y-\hat{y}) \cdot (-1)\right] \cdot \left[\hat{y} \cdot (1-\hat{y}) \cdot x\right]$$
$$= \frac{-2}{m} \sum_{i=1}^{m} (y-\hat{y}) \cdot \hat{y} \cdot (1-\hat{y}) \cdot x$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b}$$
$$= \frac{-2}{m} \sum_{i=1}^{m} (y-\hat{y}) \cdot \hat{y} \cdot (1-\hat{y})$$

FIGURE 3 (a) Backpropagation for weight and bias of group 1

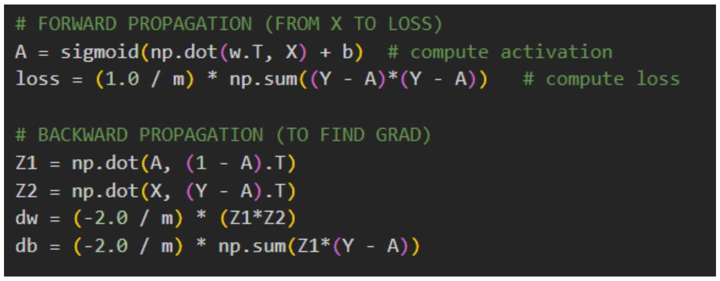
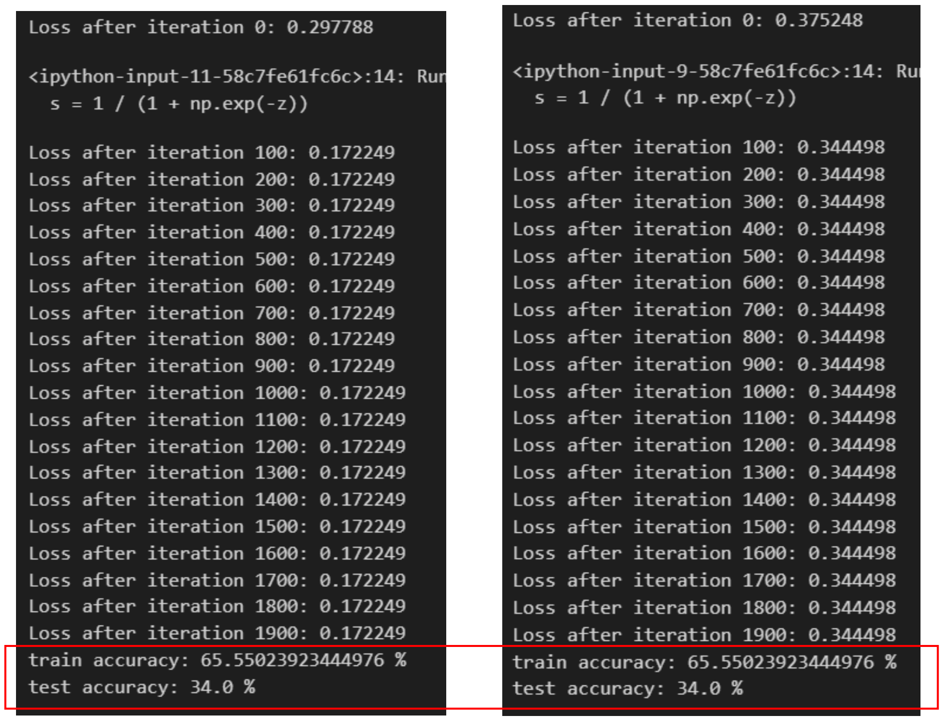

```
# FORWARD PROPAGATION (FROM X TO LOSS)
A = sigmoid(np.dot(w.T, X) + b)  # compute activation
loss = (1.0 / m) * np.sum((Y - A)*(Y - A))   # compute loss

# BACKWARD PROPAGATION (TO FIND GRAD)
Z1 = np.dot(A, (1 - A).T)
Z2 = np.dot(X, (Y - A).T)
dw = (-2.0 / m) * (Z1*Z2)
db = (-2.0 / m) * np.sum(Z1*(Y - A))
```

FIGURE 3 (b) Backpropagation for weight and bias of group 2 in coding

Results Analysis

By using a 0.01 learning rate and 2000 iterations, the loss function was printed every 100. Figure 4 (a) shows the output for two models and Figure 4 (b) shows the learning curve for two models. Based on the results, training accuracy and testing accuracy for the two groups are the same. However, the loss of two groups is diverse. Group 1 has a lower loss function which means group 1's model fits the ground truth better than group 2's model. Besides, compared to the training accuracy, testing accuracy has a lower value that identifies that two models are overfitting. Two models have the same learning curve based on Figure 4 (b). Two models update the loss very fast, and there are no changes since 100 iterations keeping a low loss.



```
Loss after iteration 0: 0.297788

<ipython-input-11-58c7fe61fc6c>:14: Ru
  s = 1 / (1 + np.exp(-z))

Loss after iteration 100: 0.172249
Loss after iteration 200: 0.172249
Loss after iteration 300: 0.172249
Loss after iteration 400: 0.172249
Loss after iteration 500: 0.172249
Loss after iteration 600: 0.172249
Loss after iteration 700: 0.172249
Loss after iteration 800: 0.172249
Loss after iteration 900: 0.172249
Loss after iteration 1000: 0.172249
Loss after iteration 1100: 0.172249
Loss after iteration 1200: 0.172249
Loss after iteration 1300: 0.172249
Loss after iteration 1400: 0.172249
Loss after iteration 1500: 0.172249
Loss after iteration 1600: 0.172249
Loss after iteration 1700: 0.172249
Loss after iteration 1800: 0.172249
Loss after iteration 1900: 0.172249
train accuracy: 65.55023923444976 %
test accuracy: 34.0 %
```

```
Loss after iteration 0: 0.375248

<ipython-input-9-58c7fe61fc6c>:14: Ru
  s = 1 / (1 + np.exp(-z))

Loss after iteration 100: 0.344498
Loss after iteration 200: 0.344498
Loss after iteration 300: 0.344498
Loss after iteration 400: 0.344498
Loss after iteration 500: 0.344498
Loss after iteration 600: 0.344498
Loss after iteration 700: 0.344498
Loss after iteration 800: 0.344498
Loss after iteration 900: 0.344498
Loss after iteration 1000: 0.344498
Loss after iteration 1100: 0.344498
Loss after iteration 1200: 0.344498
Loss after iteration 1300: 0.344498
Loss after iteration 1400: 0.344498
Loss after iteration 1500: 0.344498
Loss after iteration 1600: 0.344498
Loss after iteration 1700: 0.344498
Loss after iteration 1800: 0.344498
Loss after iteration 1900: 0.344498
train accuracy: 65.55023923444976 %
test accuracy: 34.0 %
```

FIGURE 4 (a) Accuracy test for group 1 (left) and group 2 (right)

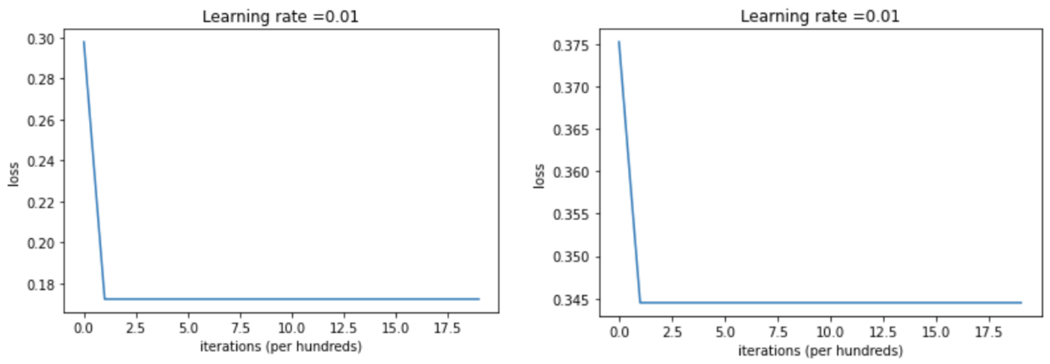


FIGURE 4 (b) Learning curve for group 1 (left) and group 2 (right) with 0.01 learning rate

Accuracy Improvement

By enhancing model accuracy, I tried different learning rates with 0.001, 0.0001, and 0.00001. Table 1 shows training accuracy and testing accuracy for two models with learning rates of 0.01, 0.001, 0.0001, and 0.00001. We can see that group 2's model has a better performance when the learning rate is 0.0001, and group 1's model has a greater accuracy percentage.

TABLE 1 Accuracy results of training and testing for two models

Group \ Learning Rate		0.01	0.001	0.0001	0.00001
Train Accuracy	Group 1	65%	65%	65%	99%
	Group 2	65%	65%	99%	89%
Test Accuracy	Group 1	34%	34%	34%	68%
	Group 2	34%	34%	72%	62%

Figure 5 displays the learning curve for group 1 (left) and group 2 (right) with different learning rates. Learning curves have no difference for both groups when the learning rate is equal to 0.001. However, the curve starts to change since the learning rate is 0.0001 for two models.

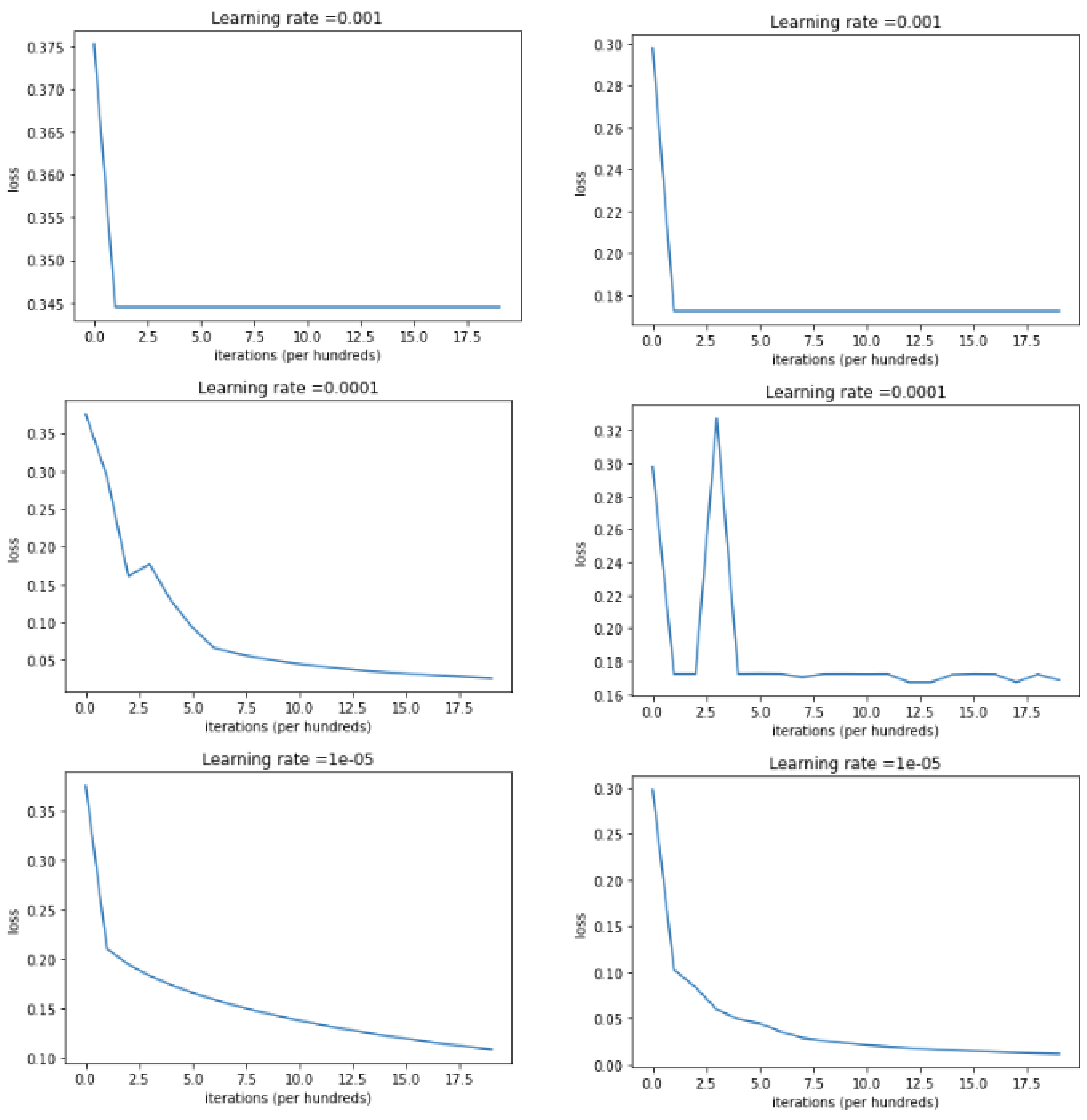


FIGURE 5 Learning curve for group 1 (left) and group 2 (right) with different learning rates