HCMUS - VNUHCM / FIT / Computer Vision & Cognitive Cybernetics Department

Digital image & video processing - LQN

**Student ID:** 21127690

**Student name:** Ngo Nguyen Thanh Thanh

**Report:** Morphological Operations

**I. Evaluation summary:**

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Task** | **Details of Implementation** | **Completion Percentage (%)** |
| **1** | **Implement FFNN Code** | Implemented the FFNeuralNetwork class with methods: \_\_init\_\_, forward, backward, train, predict, save\_weights, load\_weights. | **100%** |
| **2** | **Implement Sigmoid Function** | Implemented the Sigmoid function and its derivative for use in hidden layers and the output layer. | **100%** |
| **3** | **Forward Propagation** | Processed input data through the layers of the model, from input to output, applying the sigmoid activation function. | **100%** |
| **4** | **Backward Propagation** | Updated weights using the error between the predicted output and the actual label, adjusting weights W1,W2W\_1, W\_2W1​,W2​. | **100%** |
| **5** | **Train the Model** | Trained the model for 1000 iterations using a learning rate η=0.1\eta = 0.1η=0.1, calculated the error after each iteration. | **100%** |
| **6** | **Predict New Data** | Normalized the new input data and used the predict method to generate prediction results. | **100%** |
| **7** | **Save and Load Weights** | Saved and loaded the model's weights after training using torch.save() and torch.load(). | **100%** |
| **8** | **Run Experiments on Sample Data** | Tested the implementation on the sample dataset X and y, printed the loss during training, and predicted new data. | **100%** |
| **9** | **Change the basic parameters of the network and conduct conduct experiments to observe close to the results.** | learning rate, class size, number of classes, class type, number of epochs.. | **100%** |

1. **List of features and file structure:**

**File structure:**

1. **nn\_simple.py**:
   * Contains the FFNeuralNetwork class and its related functions.
2. **Test\_ffnn.py**:
   * Contains the main logic to train and test the FFNN.

**Functions and methods used:**

**Activation functions:**

* **sigmoid(s)**: Computes the sigmoid activation function.
* **sigmoid\_derivative(s)**: Computes the derivative of the sigmoid function.

**FFNeuralNetwork class:**

* **Main methods:**
  + **\_\_init\_\_()**: Initializes network parameters and random weights.
  + **forward(X)**: Performs forward propagation.
  + **backward(X, y, o, rate)**: Performs backward propagation to compute gradients and update weights.
  + **train(X, y, rate)**: Trains the network using forward and backward propagation.
  + **predict(x\_predict)**: Makes predictions using new input data.
  + **save\_weights(model, path)**: Saves the model weights to a file.
  + **load\_weights(path)**: Loads the model weights from a file.

**Image proof**:

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated

A computer screen with text

Description automatically generated

A screenshot of a computer program

Description automatically generated

A computer screen with colorful text

Description automatically generated

**III. Summarization of the usage**

**Usage instructions:**

1. **Prepare the input data**:
   * Prepare the input X (features) and the output y (labels) and normalize them.

X\_max, \_ = torch.max(X, 0)

X = torch.div(X, X\_max)

y = y / 100 # If the target output is a percentage (0-100)

1. **Create the FFNN**:
   * Initialize the FFNeuralNetwork object.

NN = ffnn.FFNeuralNetwork()

1. **Train the model**:
   * Train the network for 1000 epochs with a learning rate of 0.1.

for i in range(1000):

print(f"Epoch #{i} Loss: {torch.mean((y - NN(X)) \*\* 2).detach().item()}")

NN.train(X, y, 0.1)

1. **Save and load weights**:

NN.save\_weights(NN, "NN\_weights") # Save the model weights

NN.load\_weights("NN\_weights") # Load the model weights

1. **Make predictions**:
   * Prepare and normalize the new input x\_predict before making a prediction.

x\_predict\_max, \_ = torch.max(x\_predict, 0)

x\_predict = torch.div(x\_predict, x\_predict\_max)

NN.predict(x\_predict)

**Algorithm explanation**

1. **Forward Propagation**:

A math equations on a white background

Description automatically generated

1. **Backward Propagation**:

A white paper with black text

Description automatically generated

1. **Training**:
   * For each epoch, forward propagation and backward propagation are executed to update the weights.
   * This process continues for a fixed number of epochs or until a stopping criterion is met.

**IV. Comments and evaluation**

**Results on sample data:**

**A screenshot of a math problem

Description automatically generated**

A black screen with white text

Description automatically generated

**Evaluation:**

1. **Strengths**:
   * **Simple and modular design**: The FFNN class is well-structured, making it easy to add more layers.
   * **Customizable**: The number of layers, hidden units, and learning rate can be easily modified.
   * **Weight saving and loading**: This allows the model to be reused without retraining.
2. **Weaknesses**:
   * **No bias term**: The weight update formula does not include bias, which may reduce model accuracy.
   * **Overfitting risk**: The model may overfit if trained too long on small datasets.
   * **Performance on larger datasets**: The model has only been tested on a small dataset and may struggle with larger, more complex datasets.
3. **Evaluation on the Wine dataset**:
   * The model could be modified to support multi-class classification.
   * Activation functions other than sigmoid (like ReLU) should be tested for better convergence.

**Conclusion**

* **Summary**:  
  Successfully implemented a Feed Forward Neural Network (FFNN) using PyTorch. The model was trained on a small dataset and tested on new data, showing good prediction performance.
* **Improvements**:
  + Add support for bias in the model.
  + Use dropout and regularization to reduce overfitting.
  + Test on larger datasets (like the Wine dataset) to analyze model performance on real-world problems.

**V. EXPERIMENT AND RESULTS:**

**ON Default result:**

**A screenshot of a computer

Description automatically generated**

The model successfully predicts an output of approximately 0.9212 for the input tensor [0.3750, 1.0000, 0.5000]. The output seems reasonable, given that the model was trained with a target output in the range [0, 1]. However, further evaluation, such as comparing the predicted output with expected results, would be useful for validating its accuracy.

**5.1 Change epochs number (learning rate = 0.1)**

* **Epochs = 50**

A screenshot of a computer

Description automatically generated

* **Epochs = 100**

A computer screen shot of numbers and letters

Description automatically generated

* **Epochs = 500**

A black screen with white text

Description automatically generated

* **Epochs = 1000:**

A computer screen with white text

Description automatically generated

* **Epochs = 2000:**

**A screenshot of a computer

Description automatically generated**

**Epoch 50:**

* **Output**: 0.6659
  + At epoch 50, the model has made some progress in learning but is still far from an accurate prediction. This suggests that the model is still in the early stages of training and is yet to understand the full relationship between the input and output.

**Epoch 100:**

* **Output**: 0.8653
  + By epoch 100, the model's prediction improves significantly, reflecting that the model has started to learn more effectively. The improvement in the prediction shows that the neural network is learning the patterns in the data and adjusting its weights accordingly.

**Epoch 500:**

* **Output**: 0.9487
  + At epoch 500, the model has achieved a higher degree of accuracy. The prediction is quite close to the actual value, indicating that the model is converging toward an optimal solution. The improvement from earlier epochs suggests that the training process is proceeding well.

**Epoch 1000:**

* **Output**: 0.8903
  + At epoch 1000, we observe a slight dip in performance. The output value decreases compared to epoch 500, which may indicate that the model has started to overfit or that the learning rate needs adjustment. The model might be bouncing between different solutions as it tries to converge further.

**Epoch 2000:**

* **Output**: 0.9323
  + By epoch 2000, the model returns to a higher accuracy, with the output approaching the value from epoch 500. This shows that the model has continued to refine its learning after the slight drop at epoch 1000. The prediction stabilizes, suggesting that the model may be close to its optimal state.

**General Observations:**

* The model improves significantly from epoch 50 to epoch 500, showing strong learning.
* However, after epoch 500, the model's performance fluctuates slightly, possibly due to overfitting or the training process reaching a local minimum.
* By epoch 2000, the model stabilizes at a reasonable prediction value (0.9323), but it’s important to monitor for potential overfitting or the need for adjustments to the training parameters (e.g., learning rate, regularization) to improve convergence.

In summary, the model's accuracy improves steadily through the first 500 epochs and then fluctuates, which could be a sign of the learning rate or training process requiring fine-tuning to avoid overfitting and further optimize the performance.

**5.2 Change learning rate (epochs = 100)**

* **rate = 0.1**

**A black screen with white text

Description automatically generated**

* **rate = 0.01**

**A black background with white text

Description automatically generated**

* **rate = 0.001**

**A black screen with white text

Description automatically generated**

* **rate = 0.5**

**A black background with white text

Description automatically generated**

* **Rate = 0.1**: Output 0.8758. This is a good prediction, indicating effective learning with a balanced rate.
* **Rate = 0.01**: Output 0.6420. The model learns slower, leading to a less accurate prediction due to a small learning rate.
* **Rate = 0.001**: Output 0.6978. Very slow learning, resulting in moderate prediction accuracy, but may need more epochs to improve.
* **Rate = 0.5**: Output 0.9081. Faster learning with a high rate, resulting in a strong prediction, but there's a risk of instability or overshooting.

**Conclusion**: A learning rate of 0.1 offers a good balance between speed and accuracy. Lower rates slow learning, and higher rates may cause instability.