Training Neural Networks with Iris dataset

BEIJING INSTITUTE OF TECHNOLOGY

GLEB PIMENOV

1820243077

I hereby promise that the experimental report and codes written by me are independently completed without any plagiarism or copying of others' homework. All the views and materials involving other classmates have been annotated. If there is any plagiarism or infringement of others' intellectual property rights, I will bear all the consequences arising from it.

Training Neural Networks with Iris dataset

1 Experiment Introduction

The experiment aims to train neural networks using the Iris dataset. The Iris dataset is a classic dataset in machine learning, containing features such as sepal length, sepal width, petal length, and petal width of iris flowers, along with their corresponding species. The goal is to train neural networks to classify iris flowers into their respective species based on these features.

Neural networks are a powerful machine learning technique inspired by the structure and function of the human brain. They consist of interconnected layers of artificial neurons that process input data to produce output predictions. Training a neural network involves iteratively adjusting its parameters (weights and biases) using optimization algorithms like gradient descent to minimize a loss function.

In this experiment, we will use the Iris dataset to train a neural network model. The dataset will be split into training, validation, and testing sets to evaluate the model's performance. The neural network will be trained using gradient descent to minimize the loss function, and its performance will be evaluated based on accuracy metrics on both the training and testing datasets.

By the end of the experiment, we aim to have a trained neural network model capable of accurately classifying iris flowers based on their features, demonstrating the effectiveness of neural networks in solving classification tasks.

2 Experiment Objectives

1. Implement gradient descent method to train neural network by yourself, and cannot use ready-made library functions.

2. Development language: Python.

3. Submit source code and implementation report

3 Relevant Theories and Knowledge

4 Experimental Tasks and Grading Criteria

No.	Task Name	Specific Requirements	Grading Criteria (100-point scale)
1	Training Neural Networks with Iris dataset		100

5 Experimental Conditions and Environment

Requirements	Name	Version	Remarks
Programming Language	Python		
Development Environment	PyCharm		
Third-party toolkits/libraries/plugins	Numpy, scikit-learn (sklearn), Matplotlib		
Other Tools			
Hardware Environment	Windows 11		

6 Experimental Data and Description

Attribute (Entry)	Content
Dataset Name	**Iris**
Dataset Origin	UCI Machine Learning Repository
Main Contents of the Dataset	Length and width of the flowers
Dataset File Format	Bunch

7 Experimental Steps and Corresponding Codes

Step number	1
Step Name	Data Preparation and Model Setup
Step Description	The code starts by loading the Iris dataset using scikit-learn's load_iris function. It then separates the features (X) and target labels (y) from the dataset. The target labels are one-hot encoded using OneHotEncoder from scikit-learn to convert them into a binary matrix representation. The dataset is split into training, validation, and testing sets using the train_test_split function. The training set comprises 60% of the data, the validation set comprises 10%, and the test set comprises the remaining 30%. A simple neural network architecture is defined with an input layer of size 4, a hidden layer with 10 neurons, and an output layer with 3 neurons. Weights for the neural network are initialized randomly using NumPy
Code and Explanation	import numpy as np from sklearn.datasets import load_iris from sklearn.model_selection import train_test_split from sklearn.preprocessing import OneHotEncoder import matplotlib.pyplot as plt #%% md

```
# Load Iris dataset
#%%
iris = load_iris()
X = iris.data
y = iris.target.reshape(-1, 1)
#%% md
# One-hot encode the target variable
#%%
encoder = OneHotEncoder()
y_onehot = encoder.fit_transform(y).toarray()  # Convert sparse matrix to
array
#%% md
# Split the dataset into training, validation, and testing sets
#%%
X_train, X_test, y_train, y_test = train_test_split(X, y_onehot, test_size=0.3,
random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.1, random_state=42)
#%% md
# Define neural network architecture
#%%
input_size = 4
hidden_size = 10
output_size = 3
learning_rate = 0.1
#%% md
# Initialize weights
#%%
```

	np.random.seed(0)
	weights_input_hidden = np.random.randn(input_size, hidden_size)
	weights_hidden_output = np.random.randn(hidden_size, output_size)
	#%% md
	# Sigmoid activation function
	#%%
	def sigmoid(x):
	    return 1 / (1 + np.exp(-x))
	#%% md
	# Derivative of the sigmoid function
	#%%
	def sigmoid_derivative(x):
	    return x * (1 - x)
	#%% md
	# Define lists to store training, validation, and test accuracies
	#%%
	train_accuracies = []
	val_accuracies = []
	test_accuracies = []

Step number	**2**
Step Name	Training the Neural Network: Forward Pass - Training Data:
Step Description	During each epoch, the input data X_train is passed through the neural network to compute the output. The dot product of the input data and the weights connecting the input layer to the hidden layer is computed to obtain the input to the hidden layer (hidden_input). The sigmoid activation function is applied to the hidden_input to compute the

	output of the hidden layer (hidden_output). Similarly, the dot product of the hidden layer output and the weights connecting the hidden layer to the output layer is computed to obtain the input to the output layer (output_input). The sigmoid activation function is applied to the output_input to compute the final output of the neural network (output).

Code and Explanation	# Forward pass - training data
	hidden_input = np.dot(X_train, weights_input_hidden)
	hidden_output = sigmoid(hidden_input)
	
	output_input = np.dot(hidden_output, weights_hidden_output)
	output = sigmoid(output_input)

Step number	**3**
Step Name	Backpropagation
Step Description	Backpropagation calculates the gradients of the loss function with respect to the weights of the neural network, allowing the weights to be updated during training. The error between the true labels (y_train) and the predicted output (output) is computed (output_error). The derivative of the sigmoid activation function is applied to the output layer's output to compute the delta, which is the error signal propagated back through the network (output_delta). The error in the hidden layer is computed by propagating the output delta backwards through the weights connecting the hidden layer to the output layer (hidden_error). Similarly, the delta for the hidden layer is computed using the derivative of the sigmoid activation function applied to the hidden layer's output (hidden_delta).

Code and Explanation	# Backpropagation
	output_error = y_train - output
	output_delta = output_error * sigmoid_derivative(output)
	
	hidden_error = np.dot(output_delta, weights_hidden_output.T)
	hidden_delta = hidden_error * sigmoid_derivative(hidden_output)

Step number	**4**
Step Name	Weight Update
Step Description	The weights of the neural network are updated using gradient descent to minimize the error. The updated weights for the connection between the hidden layer and the output layer (weights_hidden_output) are computed by multiplying the transpose of the hidden layer output with the output delta and scaling it by the learning rate. Similarly, the updated weights for the connection between the input layer and the hidden layer (weights_input_hidden) are computed by multiplying the transpose of the input data with the hidden delta and scaling it by the learning rate.
Code and Explanation	# Update weights weights_hidden_output += np.dot(hidden_output.T, output_delta) * learning_rate weights_input_hidden += np.dot(X_train.T, hidden_delta) * learning_rate

Step number	**5**
Step Name	Forward Pass - Validation and Test Data
Step Description	After updating the weights based on the training data, the neural network's performance is evaluated using the validation and test data.

	The validation data (X_val) and test data (X_test) are passed through the trained neural network to compute their respective outputs. The calculated outputs are used to compute the validation and test accuracies, which measure the model's performance on unseen data. The accuracies are then appended to the corresponding lists (val_accuracies and test_accuracies).

Code and Explanation	# Forward pass - validation data
	hidden_output_val = sigmoid(np.dot(X_val, weights_input_hidden))
	output_val = sigmoid(np.dot(hidden_output_val, weights_hidden_output))
	
	# Calculate validation accuracy
	predicted_labels_val = np.argmax(output_val, axis=1)
	val_accuracy = np.mean(predicted_labels_val == np.argmax(y_val, axis=1))
	val_accuracies.append(val_accuracy)
	
	# Forward pass - test data
	hidden_layer_test = sigmoid(np.dot(X_test, weights_input_hidden))
	predictions_test = sigmoid(np.dot(hidden_layer_test, weights_hidden_output))
	
	# Convert probabilities to class labels
	predicted_labels_test = np.argmax(predictions_test, axis=1)
	
	# Calculate test accuracy
	test_accuracy = np.mean(predicted_labels_test == np.argmax(y_test, axis=1))
	test_accuracies.append(test_accuracy)

	if epoch % 100 == 0:
	# Calculate training accuracy
	predicted_labels_train = np.argmax(output, axis=1)
	train_accuracy = np.mean(predicted_labels_train == np.argmax(y_train, axis=1))
	train_accuracies.append(train_accuracy)
	print(f"Epoch {epoch}: Training Accuracy = {train_accuracy}, Validation Accuracy = {val_accuracy}, Test Accuracy = {test_accuracy}")
	#%%

Output results and Interpretation	The training, validation, and test accuracies are printed at intervals of 100 epochs to monitor the training progress.

8 Experiment Difficulties and Precautions

To prevent overfitting, a validation set is utilized to monitor the model's performance during training. Additionally, early stopping could be implemented based on the validation accuracy to prevent the model from overfitting to the training data.

9 Experiment Results and Interpretation

After training for 290 epochs, the final test accuracy achieved by the model is reported.

The plot displays the validation and test accuracies over epochs. Each point on the plot represents the accuracy at a particular epoch. The decreasing trend in accuracies over epochs could indicate potential overfitting if the test accuracy starts to decline significantly compared to the validation accuracy.

Final test and validation accuracy:

Test Accuracy = 0.8888888888888888

Validation Accuracy = 0.9090909090909091

10 References

UCI Machine Learning Repository - Iris Dataset

11 Experiment-related Metadata

Metadata Item	Content
Case name	Training Neural Networks with Iris dataset
Applicable course name	Machine learning Fundamentals
Keyword/Search Term	Training, Neural Networks, Iris dataset
AliTianchi URI	

12 Remarks and Others