Machine Learning Lab Homework-4

Gourav Yadav (2020KUCP1007)

1 MLP Implementation

1.1 Code Explanation

MLP (Multilayer Perceptron) is a type of artificial neural network (ANN) consisting of multiple layers of interconnected nodes, known as neurons. It is a feedforward neural network, meaning information flows from the input layer through the hidden layers to the output layer without any feedback loops.

1.1.1 Data Loading and Preparation

Importing the libraries. Then loading the data sets and separating them into two parts.Normalizing the first part of data sets.

1.1.2 relu() Method

The "relu()" function implements the Rectified Linear Unit (ReLU) activation function. The ReLU activation function is commonly used in neural networks, particularly in hidden layers, to introduce non-linearity. It is defined as the maximum between 0 and the input value. ReLU helps the network learn complex patterns and can alleviate the vanishing gradient problem. The function relu returns the result of the np.maximum operation, which is the array with negative values replaced by 0.

1.1.3 softmax() Method

The "softmax ()" method is used to computes the softmax activation for an array.The softmax function is commonly used in multiclass classification problems to convert raw scores (logits) into a probability distribution over multiple classes. It ensures that the predicted probabilities sum up to 1.

1.1.4 cross_entropy_loss() Method

The "cross_entropy_loss()" method is used to computes the cross-entropy loss between the true labels y_true and the predicted probabilities y_pred. Cross-entropy loss is a common loss function used in classification problems, particularly when the predicted probabilities are compared to one-hot encoded true labels. It quantifies the dissimilarity between the predicted probabilities and the true labels. Minimizing the cross-entropy loss encourages the predicted probabilities to match the true labels more closely.

1.2 Implementing MLP

Class named MLP is constructed containing the following functions:

- __init__() : Initializes the MLP object. It takes three parameters: input_dim (dimensionality of the input features), hidden_dim (number of neurons in the hidden layer), and output_dim (number of classes in the output layer). It initializes the model's weights and biases randomly using the np.random.randn function, scaled by the square root of the respective layer dimensions.
- forward(): Performs forward propagation through the neural network. It takes an input array X and computes the scores (logits) of the output layer and the intermediate hidden layer activations. It uses the relu function to apply the rectified linear unit activation to the hidden layer outputs.

- backward(): Performs backpropagation to compute the gradients of the model's parameters (weights and biases) and updates them using gradient descent. It takes input X, true labels y_true, predicted probabilities y_pred, hidden layer activations h, and learning rate 'lr' as parameters.
- predict(): Predicts the class labels for the input X by performing forward propagation and returning the class with the highest score (argmax) from the output layer.
- evaluate(): Computes the loss and accuracy of the model on the given input X and true labels y. It uses the forward method to obtain the predicted probabilities, and then calculates the cross-entropy loss using cross_entropy_loss and the accuracy by comparing the predicted labels with the true labels.

1.3 Training MLP with different number of hidden units

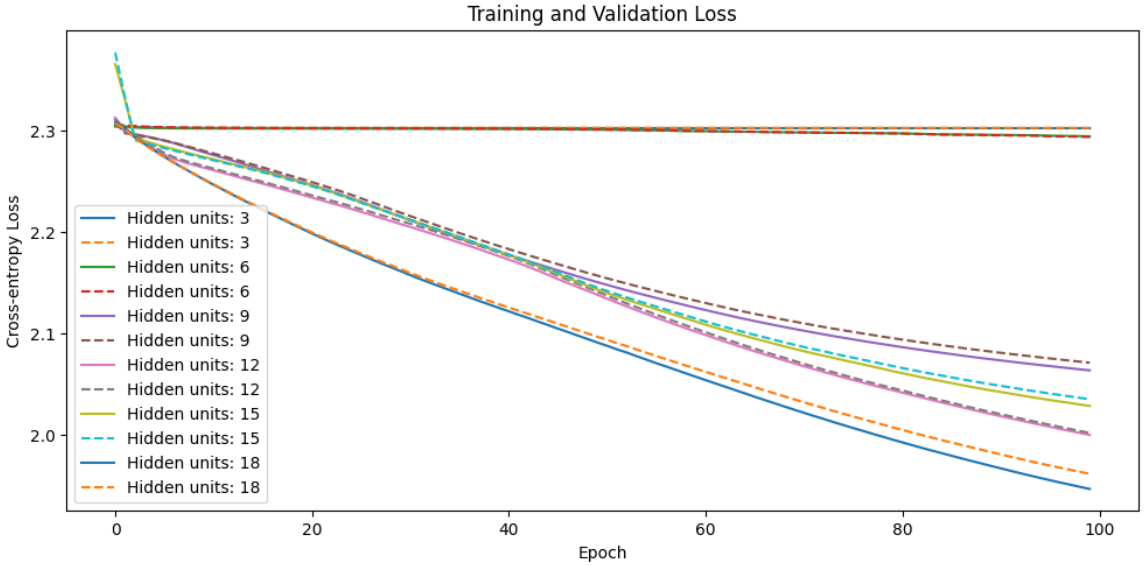
evaluates a multi-layer perceptron (MLP) model with with 3, 6, 9, 12, 15, 18 hidden units. Here we trains a K-nearest neighbors (KNN) model with different values of k (i.e., 1, 3, 5, and 7) using the "KNN" class implemented earlier. The performance of each model is evaluated using accuracy as the evaluation metric, which is calculated using the "accuracy_score" function from the "sklearn.metrics" module.

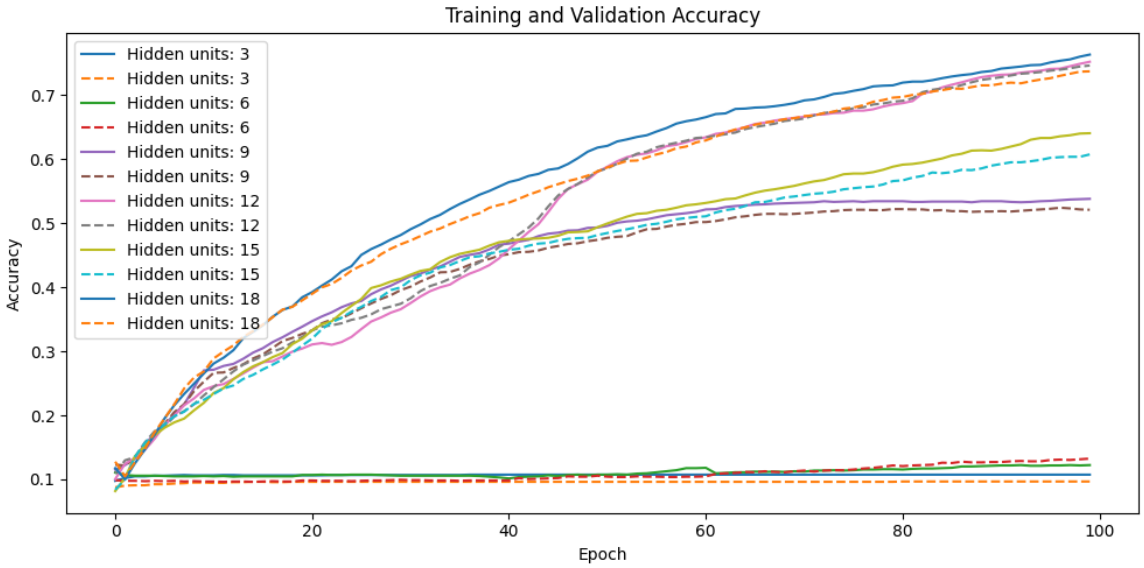
For each value of k in the list "ks", the following steps are performed:

- hidden_units() = [3, 6, 9, 12, 15, 18]: Defines a list of different numbers of hidden units to be tested.
- num_epochs() = 100: Sets the number of training epochs.
- learning_rate() = 0.2: Sets the learning rate for the model.
- Initializes arrays to store the training and validation losses, as well as the training and validation accuracies for each combination of hidden units and epoch.
- The code then iterates over each number of hidden units and trains an MLP model for that configuration. It performs forward and backward propagation, computes the losses and accuracies on the training and validation sets for each epoch, and stores the values in the corresponding arrays.
- After training each model, it prints the training loss, training accuracy, validation loss, and validation accuracy for the last epoch of each model.
- Next, the code plots the training and validation loss curves in the first subplot and the training and validation accuracy curves in the second subplot. Each curve corresponds to a different number of hidden units.
- The code then evaluates the best model (determined by the highest validation accuracy) on the test set. It prints the training loss, training accuracy, validation loss, validation accuracy, test loss, and test accuracy for the best model.

The code provides a way to compare the performance of the MLP model with different numbers of hidden units and visualize the training and validation curves. It also evaluates the best model on the test set to assess its generalization performance.

```
Hidden units: 3, Train Loss: 2.3024, Train Accuracy: 0.1073, Valid Loss: 2.3027, Valid Accuracy: 0.0966
Hidden units: 6, Train Loss: 2.2943, Train Accuracy: 0.1223, Valid Loss: 2.2937, Valid Accuracy: 0.1324
Hidden units: 9, Train Loss: 2.0638, Train Accuracy: 0.5382, Valid Loss: 2.0713, Valid Accuracy: 0.5211
Hidden units: 12, Train Loss: 2.0001, Train Accuracy: 0.7523, Valid Loss: 2.0021, Valid Accuracy: 0.7464
Hidden units: 15, Train Loss: 2.0287, Train Accuracy: 0.6407, Valid Loss: 2.0353, Valid Accuracy: 0.6076
Hidden units: 18, Train Loss: 1.9469, Train Accuracy: 0.7635, Valid Loss: 1.9618, Valid Accuracy: 0.7373
```






```
Best Hidden units: 12, Train Loss: 1.9720, Train Accuracy: 0.7261, Valid Loss: 1.9792, Valid Accuracy: 0.7186, Test Loss: 1.9879, Test Accuracy: 0.6905
```

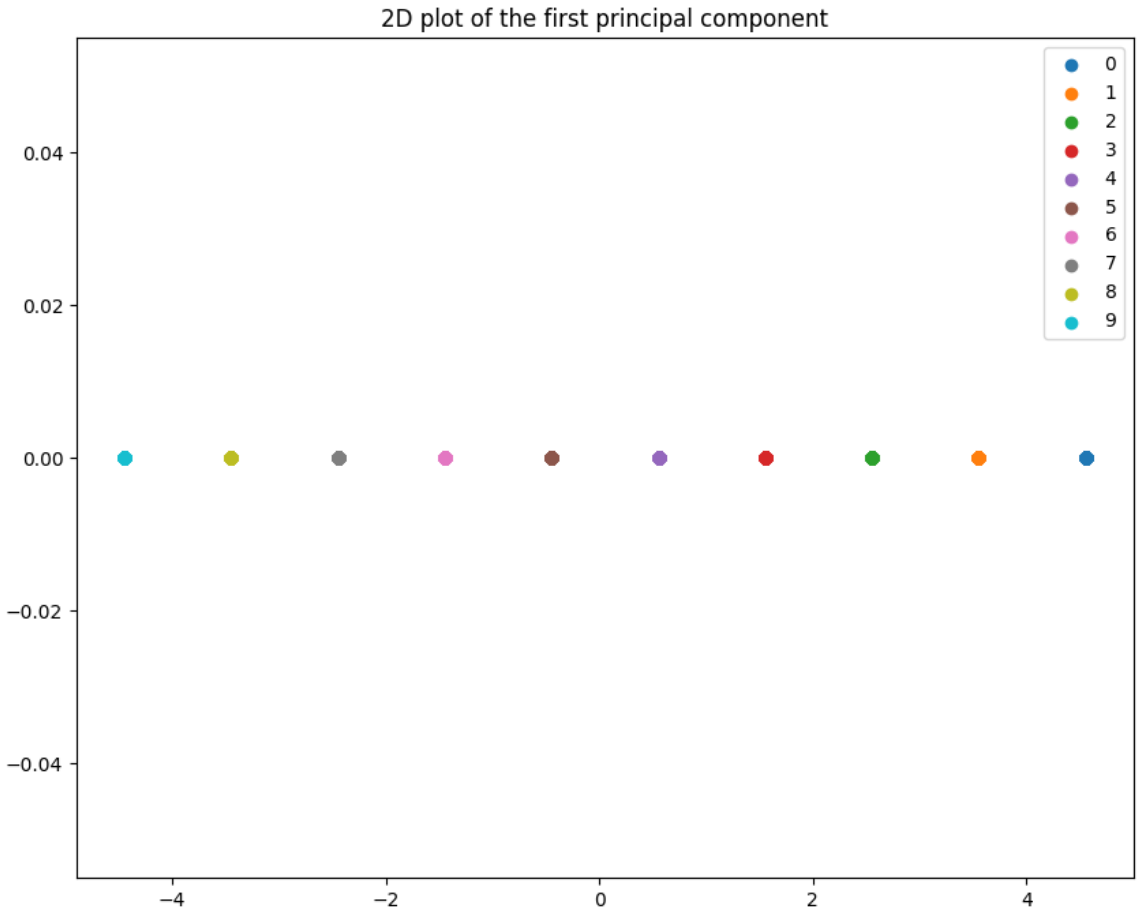
2 Testing and Ploting

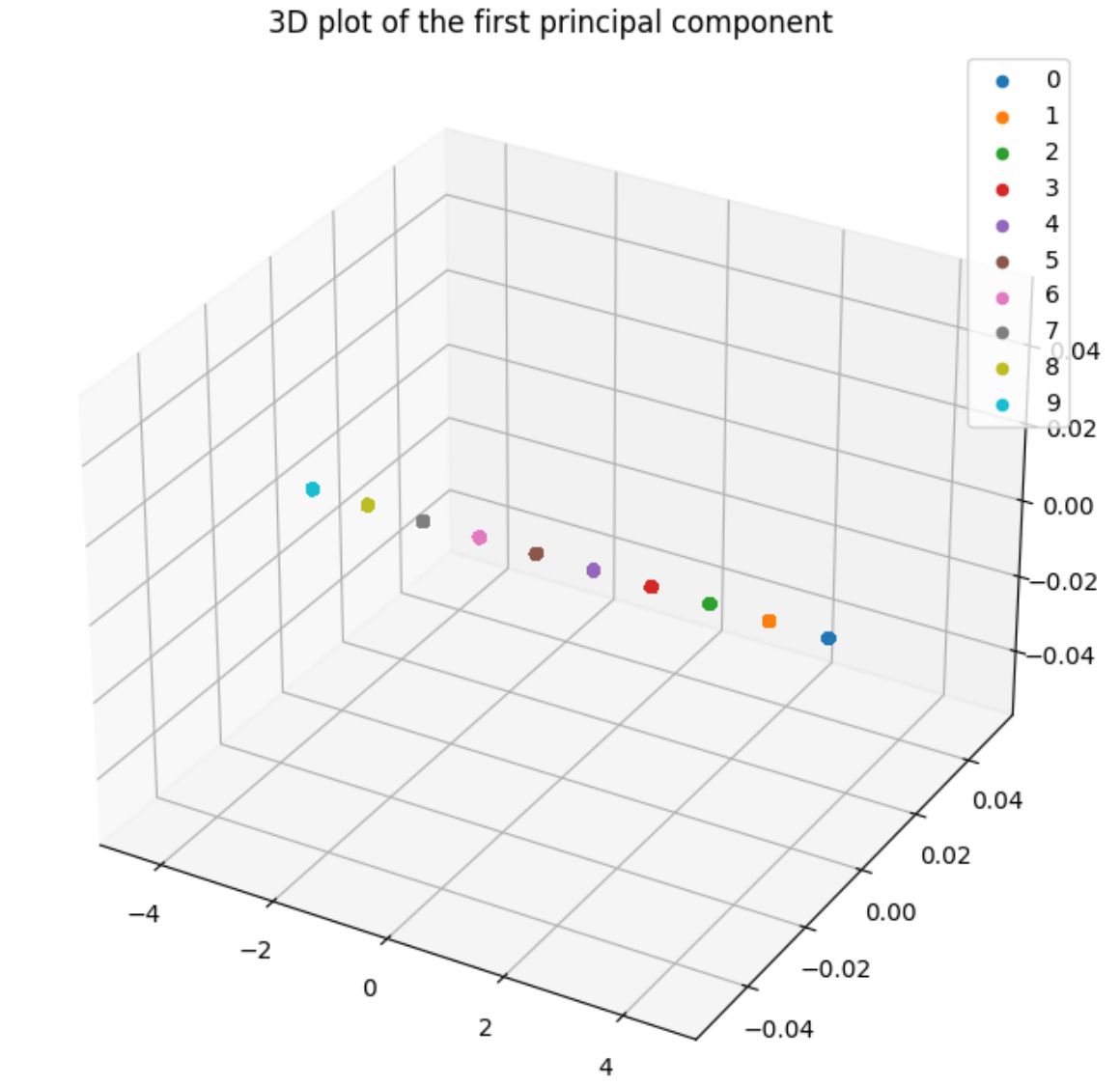
Training the MLP with the best number of hidden units obtained. Combining the training set and the validation set as one (training+validation) dataset to run the trained MLP with the data. Appling PCA to the values obtained from the hidden units. Now plotting.

- Combines the training and validation sets by concatenating the feature arrays (train_X and valid_X) and the label arrays (train_y and valid_y).
- Trains an MLP classifier using the best number of hidden units (best_hidden_units) obtained previously. The classifier is initialized with specific settings such as the activation function ('relu'), solver ('sgd'), learning rate ('0.2'), maximum number of iterations ('1000'), and random state ('0').
- Uses the trained MLP classifier to predict the hidden units for the combined training and validation sets (X_train_val). The predicted hidden units are stored in the hidden_units variable.
- Expands the dimension of the hidden_units array to make it compatible for PCA.

- Applies Principal Component Analysis (PCA) with n_components=1 to the hidden_units array. The PCA transformation is fit on the hidden_units and then applied to the same data to obtain the first principal component. The transformed values are stored in the X_pca variable.
- Plots a 2D scatter plot of the first principal component, where each class is represented by a different color. The X_pca values are plotted on the x-axis, and the y-coordinate is set to 0 for all samples.
- Plots a 3D scatter plot of the first principal component, where each class is represented by a different color. The X_pca values are plotted on the x-axis, and the y- and z-coordinates are set to 0 for all samples.

These plots provide a visualization of the first principal component after reducing the predicted hidden units using PCA. Each class is represented by a different color, allowing for a better understanding of the distribution of hidden units in the reduced space.




3D plot of the first principal component
0
1
2
3
4
5
6
7
8
9
0.04
0.02
0.00
−0.02
−0.04
−4
−2
0
2
4
0.04
0.02
0.00
−0.02
−0.04