Classification and Regression Task - Report

Firstly, the base model with different combinations of hyperparameters was trained for the classification task. The base model is a simple feedforward neural network built using PyTorch's 'nn.Module' class. The network has two linear layers, l1 and l2, with a hidden size specified by the argument's hidden size. The activation function for the hidden layer is specified by the argument activation. The input is first passed through the first linear layer (l1), followed by the activation function, then passed through the second linear layer (l2), and finally, the output is obtained by applying the SoftMax function to the output of l2. Further, a little bit complex model compared to the base model was used by adding the conv2d layers. It is a convolutional neural network (ConvNet) built using PyTorch's nn.Module class. The network has two convolutional layers (conv1 and conv2) and two fully connected layers (fc1 and fc2). The activation function for the hidden layers is specified by the argument activation. The input is first passed through the first convolutional layer (conv1), followed by the activation function, then through max pooling to reduce the spatial dimensions, and then through the second convolutional layer (conv2). The output of conv2 is then flattened and passed through the first fully connected layer (fc1), followed by the activation function, and then through the second fully connected layer (fc2) to obtain the final output. Further, the train function was developed to train the classification models that take the different hyperparameters as input. The inner body of the train function performs the following steps for the training of the model:

- Training Loop: The function performs the training process over a number of epochs. In each epoch, it first initializes the average loss as 'epoch-average-loss' to 0 and the count of correctly classified samples.
- One-hot encoding: The function one-hot encodes the labels to calculate the mean squared error (MSE) loss.
- Loss Calculation and Backpropagation: The function calculates the loss using the specified criterion and sets the gradients of the parameters to zero using. It then computes the gradients of the loss with respect to the parameters using loss.backward(), and updates the parameters using optimizer.step().

The base model and the proposed model were trained using the train function with different combinations of hyperparameters. Both models were trained on 10 epochs with different values of hyperparameters. The best-fitted parameters for both models were extracted based on the accuracy score. All the possible combinations of the hyperparameters were empirically used to best fit the models. After extracting the best hyperparameters for both models, the accuracy of the models was evaluated on the optimal parameters to get the final model. The finalized model with the highest accuracy score and optimal hyperparameters values was further used to train on the different number of epochs. The base model showed the highest 95.33% accuracy score after parameter tunning while the customized model showed the highest 98.98% accuracy score during the training of the model. As the accuracy of the customized model is high, so the customized model was evaluated by training on the different number of epochs ranging from 10 to 500. The customized model showed the highest 99.73% accuracy score during the training of the model on 500 epochs. The finalized proposed model trained on 500 epochs with fine-tuned hyperparameters showed a 96.66% accuracy score on the samples of test data.

By analyzing the results of the trained models, we concluded that the optimizers and activation functions play a significant role in the performance of the classification model. The Sigmoid activation function with AdaGrads and SGD was completely unable to train the model while the sigmoid with Adam optimizer showed significant results. Similarly, the tanh activation function with SGD optimizer completely failed to train the model. Moreover, the ReLU activation function also performed well with Adam and AdaGrads compared to the SGD. Collectively, the SGD optimizer has no significant role compared to the Adam and AdaGrads. Moreover, the Adam and AdaGrads showed prominent results with ReLU activation function rather than the tanh and Sigmoid.

A regression model in the PyTorch framework was developed for house price prediction. It was a feedforward neural network for regression tasks with 3 linear (dense) layers. The first layer has 4 input neurons, the second layer has 128 neurons, and the third layer has 64 neurons, with the output layer having only 1 neuron. The activation function is provided as an input argument and can be any activation function, such as ReLU or Sigmoid. The forward method defines the forward pass through the network, where the input tensor is passed through each layer and transformed by the activation function after each linear transformation. Numerous values of the different hyperparameters including the hidden size, activation functions, optimizers, learning rates and loss functions were empirically used to best fit the model. All the possible combinations of the selected hyperparameters were used to train the model. The performance of the model was evaluated on the MSE for every combination of hyperparameters. The model outer performed with the ReLU activation function, 0.01 learning rate, RMSprop optimizer, and 10 batch size with the training on the 500 epochs compare to the other trained models. The outer perform model with the tunned hyperparameters showed the 1165575763.61 MSE on the training set and 2325866752.0 on the test set.

By analyzing the results of the trained model with different combinations of hyperparameters, we concluded that the tanh and Sigmoid activation functions are completely unable to best fit the model for the house price prediction dataset. While the ReLU activation function performed well in a different situation with the combination of other hyperparameters. Moreover, the ReLU activation function performs well with 0.01 rather than the 0.001 and 0.0001 values of the learning rate. It also concluded that the complexity of the dataset is not much high, and the model quickly learns from the dataset with a large value of the learning rate. All the selected values for different hyperparameters are presented in the below table:

Hyperparameter Name	Selected Values
Activation	ReLU, Tanh, Sigmoid
Learning Rate	0.01, 0.001, 0.0001
Batch Size	10, 100, 200, 400
Optimizer	Adam, RMSprop
Loss	MSE
Epochs	20, 50, 100, 200, 500