**Neural Networks and Deep Learning – ICP 6**

**GitHub Link: <https://github.com/shanmukha1610/NNDL_ASS6>**

**Video Link: <https://drive.google.com/file/d/1yPGnpvbYJUWrX4gzXK4Or2ipzP90Xkdk/view?usp=drive_link>**

**Name: Pasumarthi Shanmukha Laxmi**

**Student ID: 700756736**

**ICP\_Basics in Keras:**

1. **Use the use case in the class:**

The code is implementing a simple neural network using the Keras API. The dataset is loaded using pandas from a CSV file and split into training and testing sets using the train\_test\_split function from scikit-learn.

The neural network has one hidden layer with 20 nodes, an input layer with 8 nodes (corresponding to the 8 features in the dataset), and an output layer with a single node (as it is a binary classification task). The activation function used in the hidden layer is ReLU, and the activation function used in the output layer is sigmoid.

The neural network is compiled using the binary\_crossentropy loss function, Adam optimizer, and accuracy as the evaluation metric. The model is then trained on the training set for 100 epochs using the fit method, and the summary of the model is printed using the summary method. Finally, the accuracy of the model is evaluated on the testing set using the evaluate method, and the loss and accuracy are printed.

Text

Description automatically generated

**Text

Description automatically generated with medium confidence**

**Text

Description automatically generated**

**a. Add more Dense layers to the existing code and check how the accuracy changes**

I have now added more dense layers to the code. The new addition to the code is the addition of two new hidden layers with 20 neurons each. The model now has three hidden layers, with the input layer having 8 input features and the output layer having a sigmoid activation function.

The model is compiled using binary cross-entropy loss and the Adam optimizer. The metric used to evaluate the model performance is accuracy.

The training is done using the fit() method, with 100 epochs and verbose set to 0 to suppress the output during training. The evaluation of the model is done using the evaluate() method on the test set.

The summary of the model is printed using the summary() method, which shows the number of trainable parameters and the layer-wise architecture of the model. Finally, the accuracy and loss of the model are printed after evaluation.

The accuracy improved from the first model to the second model. The first model had only one hidden layer with 20 neurons, while the second model had three hidden layers, each with 20 neurons. This allowed the second model to learn more complex features in the data, which resulted in a higher accuracy. Additionally, the second model had more parameters to optimize during training, which likely contributed to the improved accuracy.

The output is shown below :

Text

Description automatically generated

I have further made changes to the model. The model has an input layer, three hidden layers with 20, 22, and 24 neurons respectively, and an output layer with one neuron. The activation function used in the hidden layers is ReLU, and the output layer uses the sigmoid activation function.

The model is then compiled with binary\_crossentropy loss and adam optimizer. The metrics parameter is set to ['acc'] to evaluate the accuracy of the model during training.

The model is trained on the training data using fit method for 100 epochs. X\_train and Y\_train are the input and output variables of the training data respectively.

**Finally, the summary of the model is printed using the summary method, and the accuracy of the model is evaluated on the test data using evaluate method, and the result is printed.**

**we can see that the accuracy decreased from model 2 to model 3. Model 2 had an accuracy of 0.6979, while model 3 had an accuracy of 0.6823. This could be due to the increased complexity of the model, as model 3 has more parameters (1219) compared to model 2 (1041). It's possible that model 3 is overfitting the training data, resulting in a lower accuracy on the test set. Additionally, the increase in hidden layers in model 3 may be causing some of the weights to be updated causing the changes in the performance.**

Below is the output:

Text

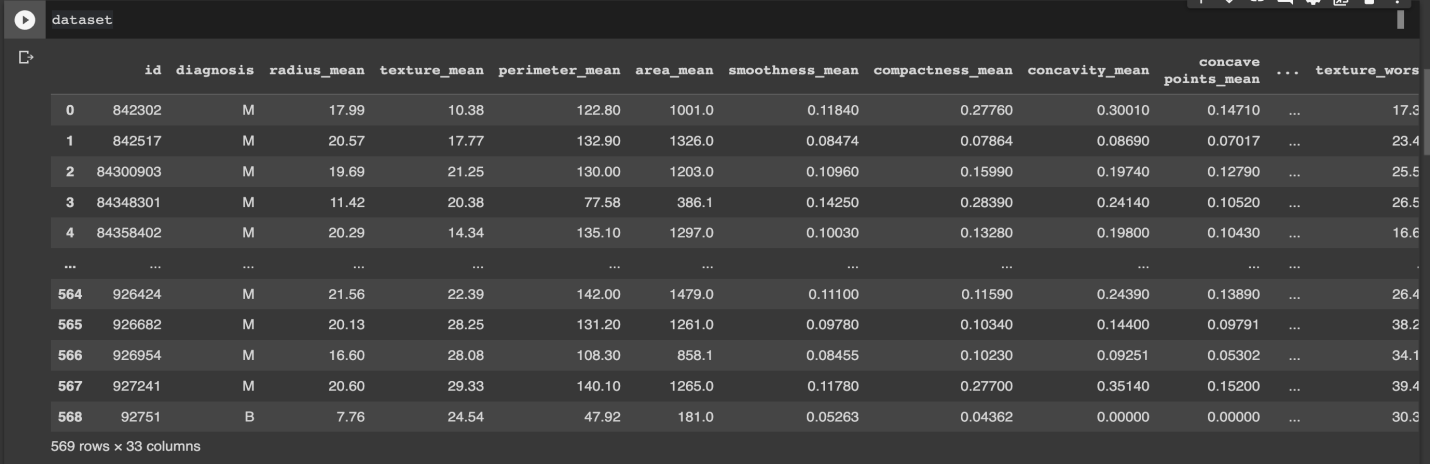
Description automatically generated

1. **Change the data source to Breast Cancer dataset \* available in the source code folder and make required changes. Report accuracy of the model.**

I have now changed the previous dataset to a new dataset which is from the breastcancer.csv file. The code to this question is available in the python file named breastcancer.ipynb.

**Text

Description automatically generated**

****

I have now used the LabelEncoder class from the scikit-learn library to convert categorical data into numerical data. Specifically, it is converting the values in the 'diagnosis' column of the DataFrame 'dataset' into numerical values. The LabelEncoder() function assigns a unique integer to each unique value in the column, so that each unique value is represented by a unique integer.

Text

Description automatically generated

The model is then trained using the fit() function, with the training data (X\_train and Y\_train) as inputs, for 100 epochs with verbose set to 0 to suppress the output during training.

After training, the model summary is printed using the summary() function, which displays the architecture of the model and the number of parameters in each layer. Finally, the model is evaluated on the test data (X\_test and Y\_test) using the evaluate() function, which returns the loss value and accuracy of the model on the test data. The accuracy of this model is 0.9161 or 91.61%, as shown by the acc metric in the model evaluation results as shown below:

Text

Description automatically generated

1. **Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below). from sklearn.preprocessing import StandardScaler sc = StandardScaler()**

my\_first\_nn.add(Dense(1, activation='sigmoid')) adds a dense output layer with 1 neuron and sigmoid activation function.

my\_first\_nn.compile(loss='binary\_crossentropy', optimizer='adam',metrics=['acc']) compiles the model with binary\_crossentropy loss function, adam optimizer, and accuracy metric.

sc = StandardScaler() creates a StandardScaler object.

X\_train = sc.fit\_transform(X\_train) fits the StandardScaler object to the training data and standardizes it.

X\_test = sc.transform(X\_test) standardizes the test data using the previously fitted StandardScaler.

my\_first\_nn\_fitted = my\_first\_nn.fit(X\_train, Y\_train, epochs=100, verbose=0, initial\_epoch=0) trains the model on the standardized training data for 100 epochs.

This output below shows the summary of a neural network model with two layers: a hidden layer with 20 neurons, and an output layer with a single neuron. The input dimension of the hidden layer is 30, which is not shown in the summary because it is automatically inferred from the input data. In this case, the model achieves an accuracy of 0.9650 and a loss of 0.1907 on the test data.

**Text

Description automatically generated**

**Text

Description automatically generated**

**Use Image Classification on the hand written digits data set (mnist)**

For the above task I have used Image Classification on the MNIST dataset.

I have first loaded the MNIST dataset, which is a large database of handwritten digits commonly used for image classification tasks as shown below.

The mnist.load\_data() function returns two tuples: (train\_images, train\_labels) and (test\_images, test\_labels).

I have then written code to visualize the image and its corresponding label to get a better understanding of the data.

plt.imshow(train\_images[0,:,:],cmap='gray') displays the first image in the training data. The cmap='gray' argument sets the color map to grayscale.

plt.title('Ground Truth : {}'.format(train\_labels[0])) adds a title to the image with the label of the image.

plt.show() shows the image on the screen.

**Text

Description automatically generated**

**Graphical user interface, qr code

Description automatically generated**

I have then written code to process the data before training the neural network for image classification on the MNIST dataset.

train\_images.shape[1:] is used to get the shape of the input image data for each training sample, which in this case is (28,28).

dimData is calculated by taking the product of the dimensions of the input images, which gives 784. This is because the neural network requires a one-dimensional input feature vector, which can be obtained by flattening each image from a 28x28 matrix into a single 784 dimensional vector.

The pixel values of the input images are then scaled to the range of 0 to 1, by dividing them by the maximum pixel value of 255.

The labels are then converted from integer format to one-hot encoding using the to\_categorical function from Keras. This is because the output layer of the neural network uses a softmax activation function, which requires one-hot encoded labels.

Finally, the neural network model is created using the Sequential API from Keras. It consists of three layers - two hidden layers with 512 neurons each, and an output layer with 10 neurons (one for each digit 0-9) using softmax activation.

**Text

Description automatically generated**

**Text

Description automatically generated**

1. **Plot the loss and accuracy for both training data and validation data using the history object in the source code.**

To plot the loss and accuracy for both the training and validation data, we can use the history object returned by the fit method. The history object contains information about the loss and metrics during training. We can plot the training and validation loss and accuracy using matplotlib

model.compile(optimizer='rmsprop', loss='categorical\_crossentropy', metrics=['accuracy']) is compiling the neural network model. It sets the optimizer to RMSprop, the loss function to categorical cross-entropy, and the metric to be monitored to accuracy.

history = model.fit(train\_data, train\_labels\_one\_hot, batch\_size=256, epochs=10, verbose=1, validation\_data=(test\_data, test\_labels\_one\_hot)) is training the neural network model. It takes in the preprocessed training data and training.

I have now plotted the accuracy and loss of the model during training and validation. The history object generated by the fit() method contains the metrics of the model at each epoch, which can be used to plot the training and validation accuracy and loss curves.

Here, the code is using the plot() function from matplotlib to plot the following:

history.history['accuracy']: training accuracy

history.history['val\_accuracy']: validation accuracy

history.history['loss']: training loss

history.history['val\_loss']: validation loss

The title(), ylabel(), xlabel(), and legend() functions are used to add labels to the plot. Finally, show() is used to display the plot as shown below:

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

A picture containing graphical user interface

Description automatically generated

1. **Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.**

I have written the code as shown below. The first line, plt.imshow(test\_data[0].reshape(28,28)), plots the image in the 0th index of the test data. The image is reshaped from a 784-dimensional array to a 28x28 matrix using the reshape() method.

The second line, print("predicted label:",model.predict(test\_data[0].reshape(1,784))), uses the trained model (model) to predict the label for the image. The predict() method is called on the model object and given the image as input. However, the input image must first be reshaped to have a batch size of 1, so it is reshaped to a 1x784 array using reshape(1,784). The predicted label is printed to the console. In our case, the predicted label is the class with the highest probability, which is class 7 (digit "7"), with a probability of approximately 0.99999964.

**Graphical user interface, text, website

Description automatically generated**

1. **We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens.**

Below I have increased the number of hidden layers in the neural network model from 2 to 4. It creates a new Sequential model and adds 4 dense layers with 512 nodes and Relu activation function, followed by a final dense layer with 10 nodes and SoftMax activation function.

The first block of code increases the number of hidden layers from 2 to 4, keeping the number of nodes in each layer at 512. The second block of code keeps the number of hidden layers at 4 but increases the number of nodes in each layer.

These modifications are made to explore the effect of changing the architecture of the neural network on its performance.

**Text

Description automatically generated**

**Text

Description automatically generated**

The model is then compiled with the RMSprop optimizer, categorical cross-entropy loss function, and accuracy as the metric to monitor during training. The fit() method is called on the model with the training data, training labels, batch size of 256, 10 epochs, and validation data consisting of the test data and test labels. The history object is then used to plot the loss and accuracy curves.

After training, the model's performance is evaluated on the test data using the evaluate() method, and the test loss and accuracy are printed.

**Text, timeline

Description automatically generated**

**Text

Description automatically generated**

I have now changed the activation to tanh and sigmoid as shown below. Using tanh activation in all the hidden layers of a neural network can have different effects on the model's performance. Compared to the relu activation function, tanh can sometimes be more effective at capturing complex patterns in the data, but it can also be more susceptible to the vanishing gradient problem when used in deep neural networks.

When all hidden layers use sigmoid activation, the network will learn to represent complex input-output relationships by gradually building up a hierarchy of features that capture increasingly abstract and high-level concepts. Evaluation result for both tanh and sigmoid activation is as shown below:

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

Printing all the evaluation results from above changes:

The results show the evaluation metrics (loss and accuracy) for each of the models trained and evaluated on the test data.

The first model with 2 hidden layers achieved an accuracy of 0.9819 and a loss of 0.0744 on the test data.

The second model with 4 hidden layers achieved an accuracy of 0.9804 and a loss of 0.0935 on the test data.

The third model with increased dense in the hidden layers achieved an accuracy of 0.9814 and a loss of 0.0803 on the test data.

The fourth model with tanh activation achieved an accuracy of 0.9664 and a loss of 0.1125 on the test data.

The fifth model with sigmoid activation achieved an accuracy of 0.9596 and a loss of 0.1305 on the test data.

Overall, the first model with 2 hidden layers achieved the highest accuracy and lowest loss on the test data. Increasing the number of hidden layers or the number of neurons per layer did not result in significant improvements in performance, and using sigmoid activation resulted in the lowest accuracy among the models evaluated.

Based on the evaluation results on the test data, the model with 2 hidden layers has the highest accuracy of 0.9819

**A picture containing graphical user interface

Description automatically generated**

1. **Run the same code without scaling the images and check the performance?**

I have now run the same code without scaling the images as shown below:

The evaluation result on Test Data without scaling shows a loss of 0.3727 and accuracy of 0.9711. This means that the model trained on unscaled data is not as efficient as the previous models which were trained on scaled data.

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

Scaling the data to a range of 0 to 1 is important because it helps to normalize the data and ensures that each feature contributes equally to the learning process. When data is not scaled, some features with larger values may dominate the learning process and bias the model towards those features. Thus, scaling the data is an important preprocessing step for building efficient machine learning models.

I have now printed the Evaluation results for all the changes made as below:

**Graphical user interface

Description automatically generated with medium confidence**

Based on the evaluation results on the test data, the model with 2 hidden layers and ReLU activation function seems to be the most efficient, with an accuracy of 0.9819 and a loss of 0.074. The model with 4 hidden layers and the model with increased dense in hidden layers have slightly lower accuracies of 0.9804 and 0.9814 respectively, but still perform well.

On the other hand, the model with tanh activation function has a lower accuracy of 0.9664 and a higher loss of 0.112, which indicates that it is not as efficient as the other models. Similarly, the model with sigmoid activation function has the lowest accuracy of 0.9596 and the highest loss of 0.130, making it the least efficient among the models evaluated.

Finally, the model trained without scaling had an accuracy of 0.9711 and a loss of 0.373, which is lower than all the other models evaluated. This indicates that scaling the data is important for improving the efficiency of the model.