Principal Component Anlysis and Neural network By JUSTINE Jean using Chatgpt 24/11/2024

I. Principal Component Analysis

A. Create dataset with more than 2 dimensions

Firstly, we had to create a small dataset with more than 2 dimension, we used the make_classification function from keras to build the dataset of 1000 samples and 5 features and 4 informative. This library allows us to create a dataset with specific needs.

B. Findingthe first 2 PC using sklearn and without using it

To find the 2 pc using sklearn we used the function PCA from sklearn decomposition that find the number of PC we want. Without sklearn, I used chatgpt to help me using numpy, finding the center then the convolution matrix and to perform the rest like sorted the indices and finding the top 2 Eigen vectors.

C. Using PCA to preserve a certain variance

We want to preserve a variance of 80%, we use once again the PCA function from the keras library using n_components=0.80 as parameters, it tells the pca algorithm to choose the minimum number of PC that explain at least 80% of the variance, then we use the fit_transform function, on the pca with X as parameter calculate the Eigen value, and transform project it to the new PC. Then we store it in a variable using explained_variance_ratio that contains the percentage of variance in each PC.

D. Train a classification

1) On the original dataset
   We split the data firstly, into test and training set we then use the function MLPClassifier from sklearn with 500 iterations as parameters to define a neural network used in classification, we then use the fit function on the train data to train the model and adjust the weight.

2) On the PC
   We used the same library as before but on the PC data and it was much quicker with better accuracy.

E. Repeat with different kernel

We used the same steps using a for loop to iterate through each kernel, but instead of using MLPClassifier, we used KernelPCA function and with different kernel: RBF, Linear and Sigmoid, we define multiple hyperparameters, and train it using the fit_transform function again. I asked chatgpt for help as I had a lot of trouble training it without transforming it and in the end I had to transform the data.

F. Building Pipeline

We used the Pipeline function from sklearn and grid searchCV also, to build the pipeline to tune the parameters, we firstly build the pipeline with PCAKernel then with MLPClassifier, we then create a grid parameter with the different kernel and hyperparameters with the learning rate, alpha gamma and the number of hidden layers that can be modified. We then perform the grid search and store the best parameters and best scores. The gridsearchCV function, train all the model with each combination of hyperparameters possible and store the best one and also use the pipeline to decide which Neural Network to use.

II. Classification problem

A. Loading the data

We are using sklearn to import iris using the load_iris function, we then encode the data with onehotencode function which let us change the word into number for the model to train on it later, and after that we split it into train and test set.

We do the same with the fashion mnist dataset but we scaled it by dividing it with 255 to have it under one (255 for the pixel's grey scale) we then reshape the image to only 1 channel to change from rgb to grayscale channel, then we also one hot encode the label to a 10-dimension vector to use it with the neurons later.

B. Build and train the neural network

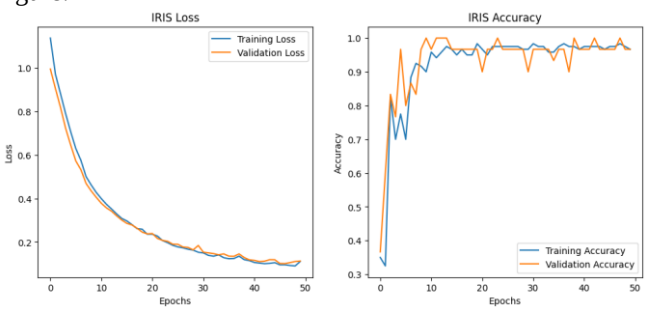
We are using Sequential and Denser from keras, we firstly create the model using sequential as the model and dense to create the neurons layers after that we compile them using optimizer loss and metrics for each dataset, for IRIS and MNIST Fasion we use Adam, categorical crossentropy and accuracy as the metrics. Then we train it using the fit function with different parameters such as epochs the batch size and we also use validation data on top of the training data. We do the same for both dataset, but with different epochs and batch size.

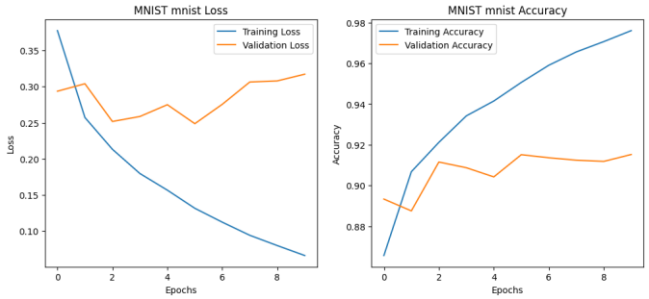
C. Tuning the hyperparameters

We tune the hyperparameters by creating a model using sequential and creating some layers with Dense function different activation method, we compile it with a learning rate and then we just train it. We could tune the model architecture as well as the optimizer, the learning rate, the epochs and the batch size also an earlystopping method.

D. Plotting loss and accuracy

We then plot the accuracy and loss, for that I asked chatgpt to plot for better figure using matplotlib and defining a function that will plot according to the model and we obtain those figure:





We can see that we have pretty terrible results on the MNIST dataset, the causes can be because of bad hyperparameters and too few epochs.

E. Save the weights of the layers and use callbacks during the training process.

We save the weight of the model using save_weight function and saving it on a file in .h5 such as iris_weight.h5.
We also create an early stopping and a checkpoint using Earlystopping function that allow us to tell how many epochs can pass with a falling in accuracy and a checkpoint using the best weight we got from before.

F. Practice saving and loading the trained model.

We save the model using save function which again create a .h5 file with the model on it and we then charge it using the laod_model function with the name of the file where the model is as parameters.

III. REGRESSION PROBLEM

A. Load California housing dataset and split it to training and testing datasets

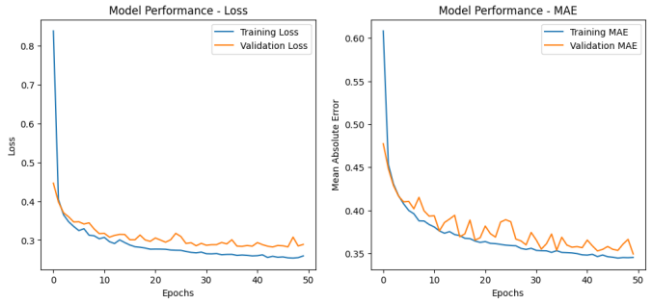
Firstly, we load the dataset using sklearn once again we split it in train and test set and then we scale it using StandardScaler function and fit_transform so we can use our model on similar data.

B. Build and train a neural network for price prediction

We create a neural network like in task 2 with Dense and Sequential with different activation and number of neurons. After that we compile it using mse as loss and mae as accuracy to fit to the regression. After compiling it we train it with the fit function once again and validation data, 50 epochs and 32 batch size.

C. plot the network history

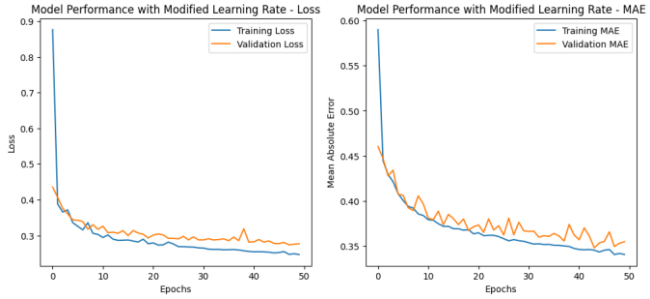
I also used chatgpt once again to plot for easier and better figure:



We don't have good performance but a few loss in overall the model can be improved.

D. Change the learning rate

We are changing the learning rate to train the model once again, we change it by adding in the optimizer a learning rate of 0.001 and we keep the others parameters and got this figure:



We can observe that with the learning rate the curve are smoother. There is more fluctuation in the one without the learning rate, both are learning well and but the learning rate help for the model to improve.

E. Save the weights of the layers and use callbacks during the training process

For this part we did the same as task 2. But we had different parameters for the early stopping and the training of the model.

F. Practice saving and loading the trained mode

For this part we also did like task 2.

I used chatgpt to help me to plot every graph, I also used it to correct me in case of coding error, also, I asked for help when putting hyperparameters and which value were the optimum one for the task we were using. I did the rest by myself and I already trained a lot of model on MNIST, IRIS and the California housing dataset which helped me a lot for this assignment and especially scaling, how many neurons to use in the input and output layers, and all the data cleaning. The rest of the neural network was done by me with knowledge from hands on machine learning. Also all the code is available on my github : https://github.com/Justine-IA/Deep_learning-