Credit Approval Prediction using Neural Network with Keras & Tensorflow

z5197402 Di Mao on 16/10/2021

Introduction:

For companies like banks, predicting credit approval with high accuracy is very important. Using the historical credit screening data, build a neural network model to predict the chance of approval for credit.

Analysis to be done:

Perform data preprocessing, exploratory data analysis, and feature engineering. Build a neural network model to predict credit approval using the historical public data.

Dataset:

The data set used here can be downloaded from https://archive.ics.uci.edu/ml/datasets/Credit+Approval (https://archive.ics.uci.edu/ml/datasets/Credit+Approval). The data file contains complete credit data, including 15 features, like the education level and income information. Additional features include age, employement status, credit score, and others. The response variable was ApprovalStatus which is binary classification problem.

Gender	Age	Debt	Married	BankCustomer	EducationLevel	Ethnicity	YearsEmployed	PriorDefault	Employed	CreditScore	DriversLicense	Citizen	ZipCode	Income	ApprovalStatus
b	30.83	0.0	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
a	58.67	4.46	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
a	24.50	0.5	u	g	q	h	1.5	t	f	0	f	g	00280	824	+
b	27.83	1.54	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+

Data Processing: Data Cleaning , One Hot Encoding & Normalization

Please note that the dataset includes both numerical and categorical data types as well as missing values.

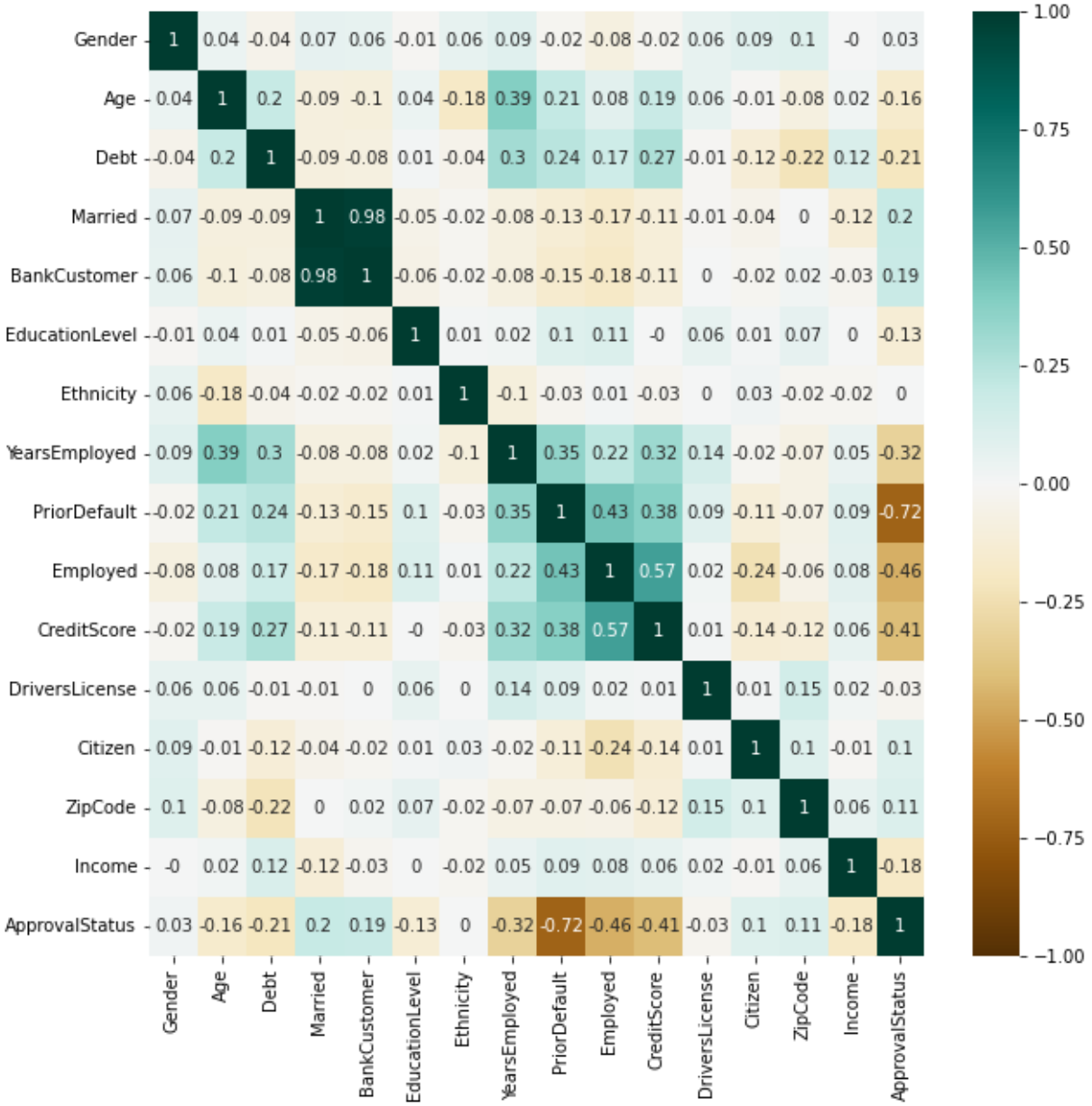
- Using pandas to load the data because it easily handles strings, whereas attempting to load the data directly training model would be more difficult.
- Converting all strings in the features to numbers is important at this stage, as well as converting response variable into integer values 0 and 1 as binary classification problem.
- By using the MinMaxScaler andLabelEncoder class from scikit-learn, all features are normalised between 0 and 1, and the target calsses are 0 or 1, respectively.
- Note that the features like "DriversLicense" and "ZipCode" are not as important as the other features in the dataset for predicting credit approvals, it is davised to drop these two features. But it will not happen in this assignment.

Gender	Age	Debt	Married	BankCustomer	EducationLevel	Ethnicity	YearsEmployed	PriorDefault	Employed	CreditScore	DriversLicense	Citizen	ZipCode	Income	ApprovalStatus
1.0	0.2568	0.0	0.5	0.0	0.9231	0.875	0.0439	1.0	1.0	0.0149	0.0	0.0	0.101	0.0	0.0
0.0	0.6755	0.1593	0.5	0.0	0.7692	0.375	0.1067	1.0	1.0	0.0896	0.0	0.0	0.0215	0.0056	0.0
0.0	0.1617	0.0179	0.5	0.0	0.7692	0.375	0.0526	1.0	0.0	0.0	0.0	0.0	0.14	0.0082	0.0
1.0	0.2117	0.055	0.5	0.0	0.9231	0.875	0.1316	1.0	1.0	0.0746	1.0	0.0	0.05	0.0	0.0
1.0	0.0965	0.2009	0.5	0.0	0.9231	0.875	0.06	1.0	0.0	0.0	0.0	1.0	0.06	0.0	0.0
1.0	0.2756	0.1429	0.5	0.0	0.6923	0.875	0.0877	1.0	0.0	0.0	1.0	0.0	0.18	0.0	0.0
1.0	0.292	0.0371	0.5	0.0	0.8462	0.375	0.2281	1.0	0.0	0.0	1.0	0.0	0.082	0.3128	0.0
0.0	0.1379	0.4138	0.5	0.0	0.1538	0.875	0.0014	1.0	0.0	0.0	0.0	0.0	0.04	0.0135	0.0
1.0	0.6116	0.0179	1.0	1.0	0.6154	0.375	0.1389	1.0	0.0	0.0	0.0	0.0	0.09	0.0031	0.0
1.0	0.4323	0.1755	1.0	1.0	0.9231	0.875	0.1111	1.0	0.0	0.0	1.0	0.0	0.026	0.0144	0.0

Now it is ready to create the neural network model using Keras.

Correlation Matrix

A correlation matrix can be used to represent the linear relationships between features as well as the correlations of the input features with the response variable.

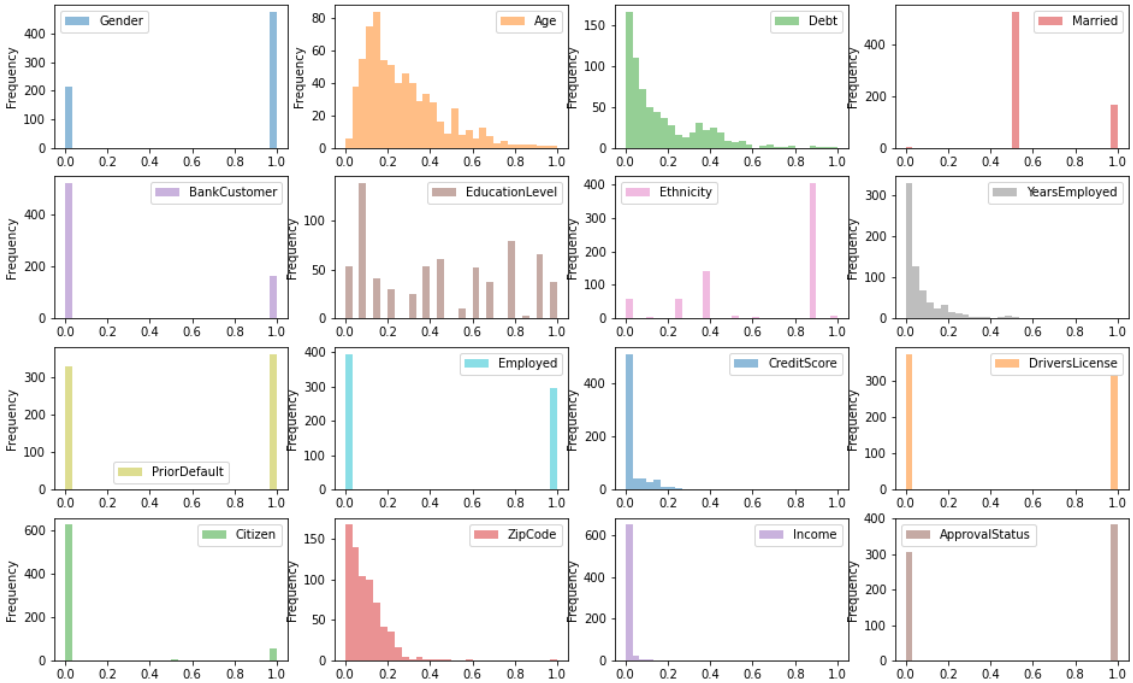


Observations:

- There are both positive and negative correlations in the heatmap.
- Features, i.e. "BankCustomer" and "Married" have high positive correlations, while "PriorDefault" and the target have high negative correlations. This makes sense as banks tend to develop couples to open joint bank accounts, and bad credit history increase the chance of rejection of credit application.

Histogram of selected features

The histogram gives the distribution of features.

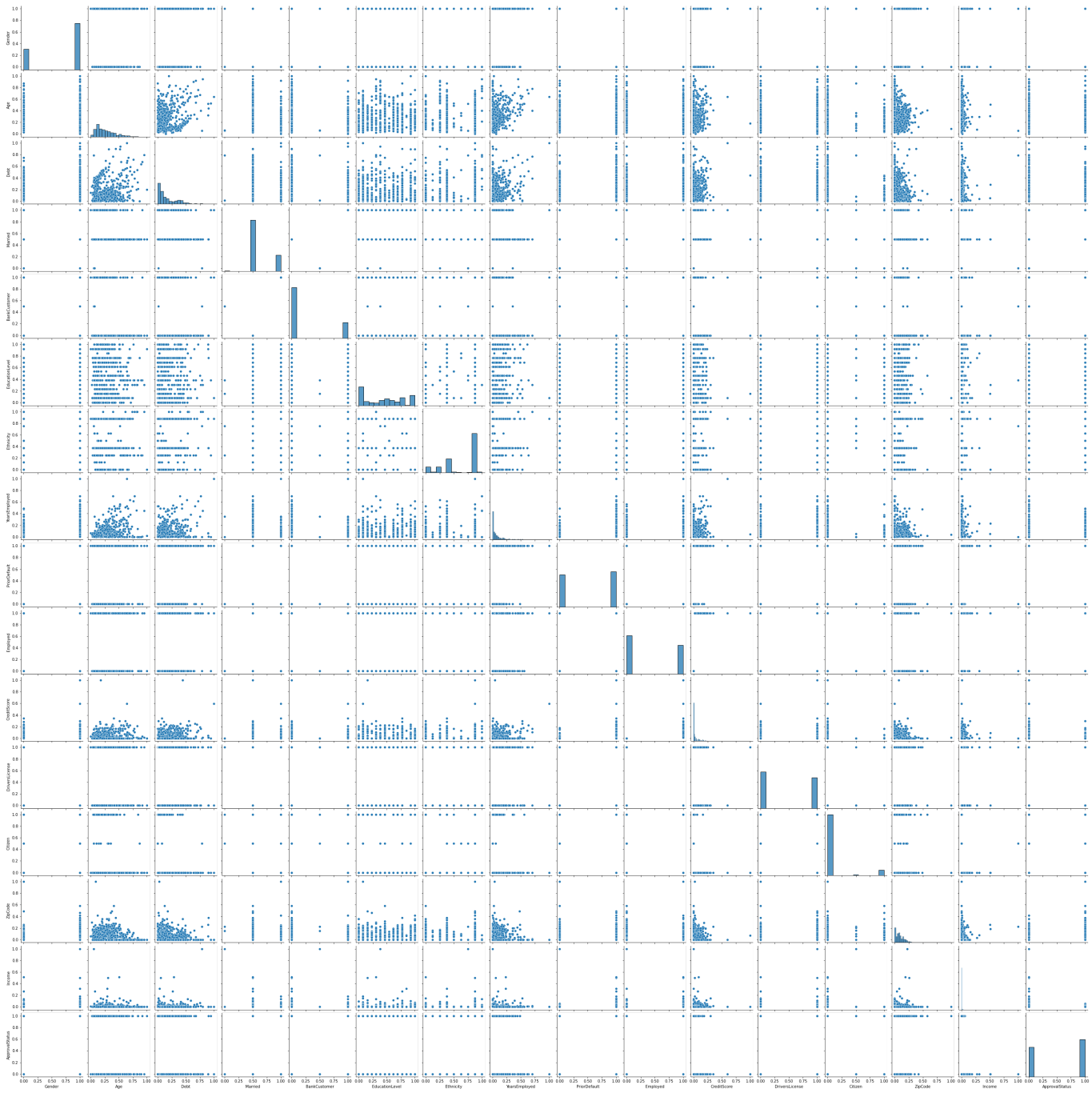


Obeservations:

- The histograms for all original numerical features display positive skewness.
- Note that almost all variables by themselves do not account for the approval status, which means that the target variable depends on a non-linear combination of all variables.

Additional Visualisations

Using pair scatterplots gives an overall visual summary of the dataset.



Modelling: 1-hidden-layer Neural Network (NN) using Keras

Step 1: Design the architecture of NN

Creating the NN architecture therefore means coming up with values for the number of layers of each type and the number of nodes in each of these layers.

- The input layer: With respect to the number of neurons comprising this layer, this parameter is completely and uniquely determined by the shape of the training data. Specifically, the number of neurons comprising this layer is equal to the number of features in the data.
- The hidden layer: There's one additional rule of thumb that helps for supervised learning problems. It can usually prevent over-fitting if keeping the number of neurons as below. Here the parameter alpha is set as 2.

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

N_i = number of input neurons.
N_o = number of output neurons.
N_s = number of samples in training data set.
α = an arbitrary scaling factor usually 2-10.

Source: [https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw (https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw)](https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw)

- The output layer: Every NN has exactly one output layer. It is completely determined by the chosen model configuration. Since the NN running for this problem is a classifier, it has one single node.

Step 2: Create the NN Model using Keras

To use Keras models with scikit-learn, we will use the KerasClassifier wrapper.

- Our model will have a single fully connected hidden layer with the number of neurons selected as demonstrated in step 1.
- The output layer contains a single neuron in order to make predictions. It uses the sigmoid activation function in order to produce a probability output in the range of 0 to 1 that can easily and automatically be converted to crisp class values.
- Since the logarithmic loss function is the preferred loss function for binary classification problems, we will use binary_crossentropy during training. The model also uses the efficient Adam optimization algorithm for gradient descent, comparing with SGD, and accuracy metrics will be collected when the model is trained.

Step 3: Training and Evaluating the NN Model

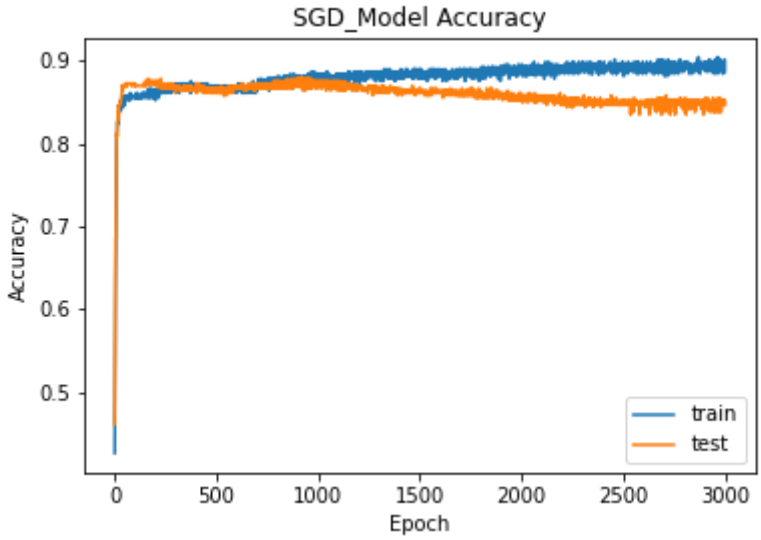
- Split the dataset into train and test subsets by using 50/50 percent
- Use SGD, Adam and Adam with combination of L2 regularisation (weight decay) to dropout for selected hyper-parameters.

Reporting Results

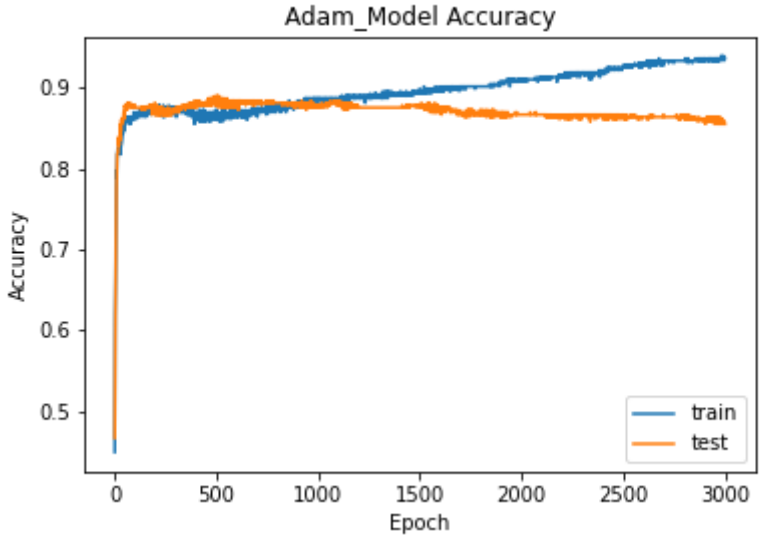
The below table presents the accuracy of the test data by using different combinations of hyper-parameters in our NN.

Combination	Optimizer	Learning rate	dropout_rate	weight_decay	Accuracy
SGD_all	SGD	0.1	0	0	87.25%
Adam_all	Adam	0.1	0	0	86.09%
Adam_l2	Adam	0.1	0	0.01	87.83%
Adam_dropout	Adam	0.1	0.4	0	86.38%
Adam_hybrid_1	Adam	0.1	0.1	0.01	87.54%
Adam_hybrid_2	Adam	0.1	0.25	0.01	87.54%
Adam_hybrid_2	Adam	0.1	0.5	0.02	87.27%

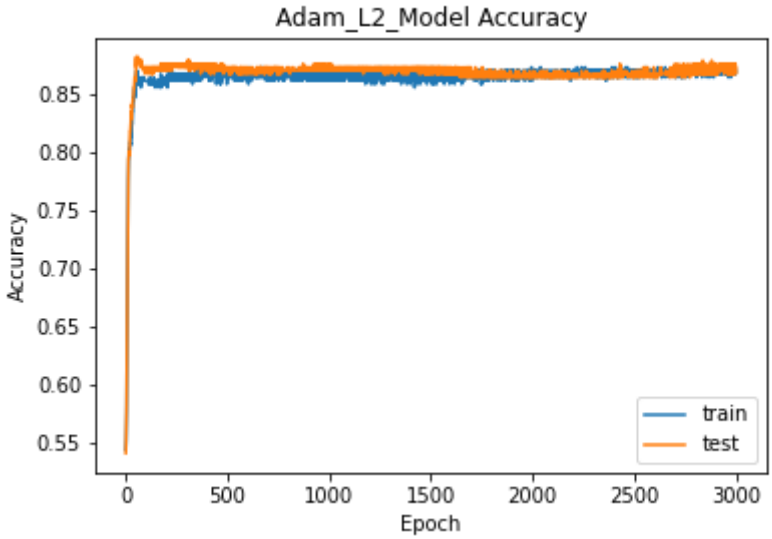
In terms of accuracy, SGD on train dataset was overfitting after 800 epochs, whilst underfitting on the test dataset. The possible reason is that SGD-based algorithms rely upon scalar and uniform learning of gradients in all the directions.



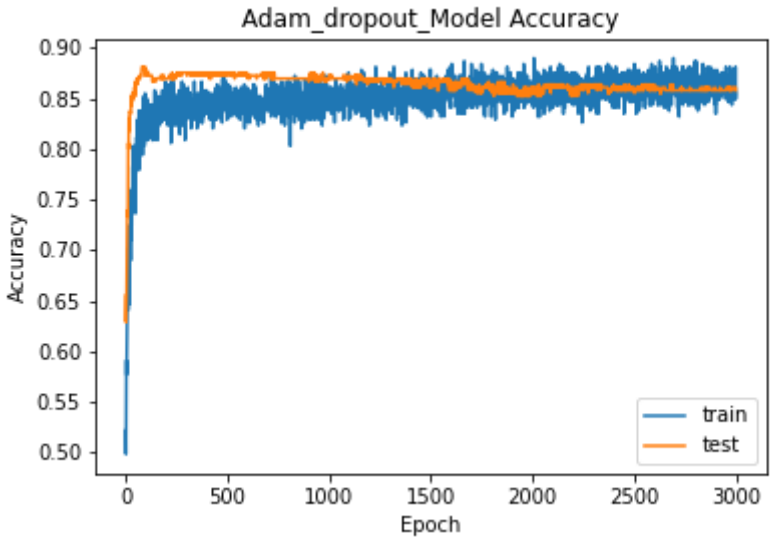
Comparing to SGD, Adam'performance was better as overfitting on train data and underfitting on test data started after about 1500 epoches.This is mainly because Adam is the extension to SGD, combining the best properties of the AdaGrad and RMSProp algorithms to handle sparse gradients on noisy problems.



In this experiment, L2 regularisation with weight decay = 0.01 applies penalties on layer parameters using Adam optimizer. These penalties are summed into the loss function that the network optimizes. As expected, we can see that the learning curve on both the train and test datasets rise and then plateau, indicating that the model may not have overfit the training dataset.

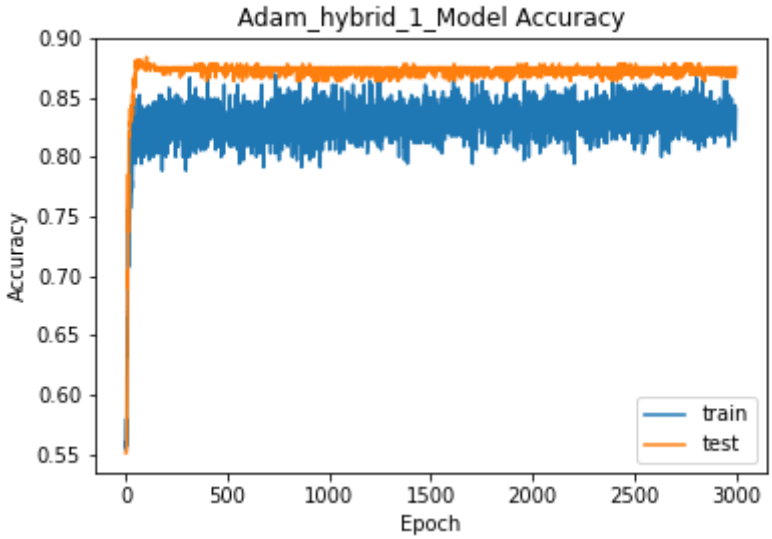


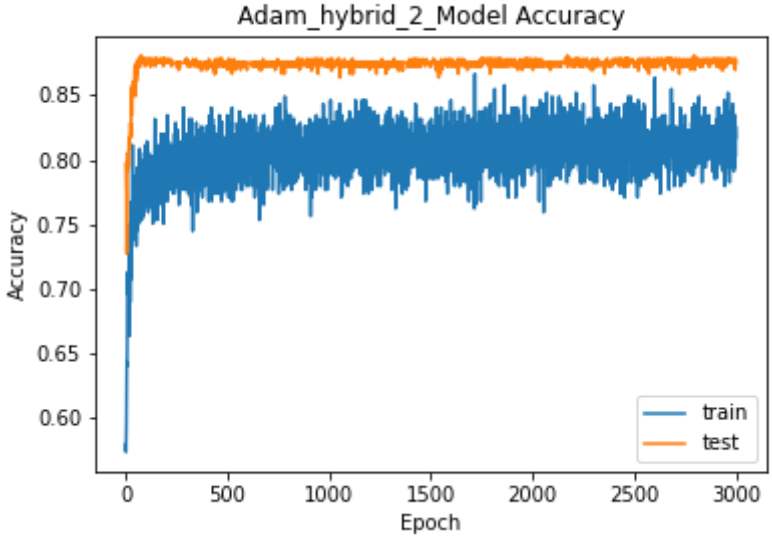
In this specific case, we introduce 40% drop rate of neurons in the hidden layer using Adam optimizer. We can see that dropout resulted in a slight drop in accuracy on the test dataset compared to Adam with L2 regularisation from last experiment, down from 87.83% to 86.38%. Model accuracy on both the train datasets saw a lot of noise given the use of dropout during training.

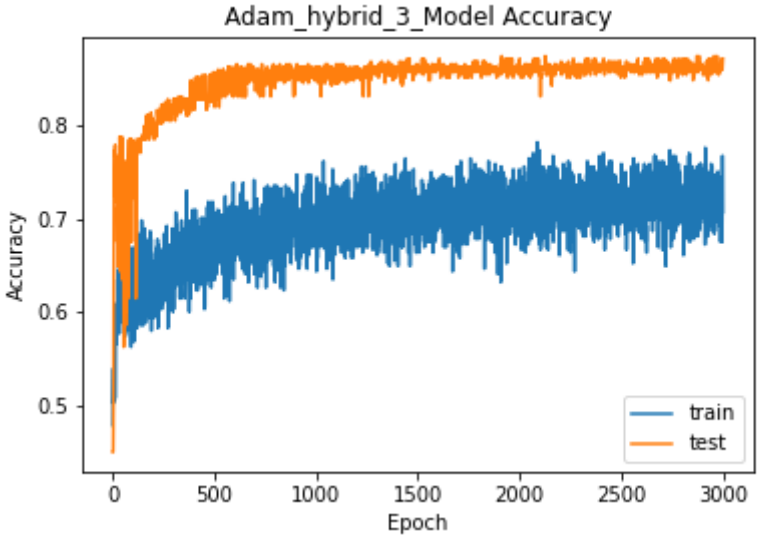


Three different combinations of L2 regularisation (weight decay) and dropout rate as proposed in question gives results as below. Holding other hyper-parameters constant, it is obvious that increasing dropout rate led to more noise added during training process, as well as in test dataset.

Combination	dropout_rate	weight_decay	Accuracy
Adam_hybrid_1	0.1	0.01	87.54%
Adam_hybrid_2	0.25	0.01	87.54%
Adam_hybrid_2	0.5	0.02	87.27%







Further development

Generally, in our experiment, as we used constant specified learning rate and just one hidden layer on a small dataset, there must be room to tune hyper-parameters to improve learning process.

- Keras supports learning rate schedules via callbacks, which operates separately from the optimization algorithm. This technique will adjust the learning rate when a plateau in model performance is detected, while to reduce the learning rate after the model stops improving with the hope of fine-tuning model weights.
- A small dropout value of 20%-50% of neurons with 20% will provide a good starting point. A probability too low has minimal effect and a value too high results in under-learning by the network. when dropout is used on a larger network, it is likely to get better performance, giving the model more of an opportunity to learn independent representations.
- A large learning rate can result in very large network weights. A large decay rate and momentum will constrain the size of network weights.

Conclusion

In this assignment problem, although we only use a simple neural network, we still got a good accuracy of over 87%. Many techniques could be implemented to improve model performance.