File Edit View Run Kernel Tabs Settings Help

credit-card-default.ipynb

Markdown Python 3

Students Do: Preventing credit card defaults with neural networks

In this activity, you will train a neural network model to predict whether a credit card holder will default in the next month.

The dataset provided contains `30,000` anonymous records of credit default status with `23` features columns and one binary target column entitled *DEFAULT*, where `1` represents a defaulted credit card.

The `23` features include demographic info (age, gender, marital status, etc.), credit limit, past payment details, and other relevant information.

You are tasked to create a neural network model to predict if a credit card holder will default.

[1]:
```
# Initial imports
from path import Path
import pandas as pd
import hvplot.pandas
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

Instructions

1. Load the data in a Pandas DataFrame.

[2]:
```
# Read in data
file_path = Path("../Resources/cc_default.csv")
df = pd.read_csv(file_path)
df.head()
```

[2]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	...	"BILL_AMT4"	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT
0	20000	2	2	1	24	2	2	-1	-1	-2	...	0	0	0	0	689	0	0	0	
1	120000	2	2	2	26	-1	2	0	0	0	...	3272	3455	3261	0	1000	1000	1000	0	200
2	90000	2	2	2	34	0	0	0	0	0	...	14331	14948	15549	1518	1500	1000	1000	1000	500
3	50000	2	2	1	37	0	0	0	0	0	...	28314	28959	29547	2000	2019	1200	1100	1069	100
4	50000	1	2	1	57	-1	0	-1	0	0	...	20940	19146	19131	2000	36681	10000	9000	689	67

5 rows × 24 columns

2. Define the features set `X` by including all the columns of the DataFrame except the `DEFAULT` column.

[3]:
```
# Define features data
X = df.copy()
X = X.drop(columns=["DEFAULT"])
X.head()
```

[3]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT3	"BILL_AMT4"	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AM
0	20000	2	2	1	24	2	2	-1	-1	-2	...	689	0	0	0	0	689	0	0	
1	120000	2	2	2	26	-1	2	0	0	0	...	2682	3272	3455	3261	0	1000	1000	1000	
2	90000	2	2	2	34	0	0	0	0	0	...	13559	14331	14948	15549	1518	1500	1000	1000	10
3	50000	2	2	1	37	0	0	0	0	0	...	49291	28314	28959	29547	2000	2019	1200	1100	10
4	50000	1	2	1	57	-1	0	-1	0	0	...	35835	20940	19146	19131	2000	36681	10000	9000	6

5 rows × 23 columns

3. Create the target vector `y` by assigning the values of the `DEFAULT` column of the DataFrame.

[4]:
```
# Define target data
y = df["DEFAULT"].values
y = y.reshape(-1, 1)
y[:5]
```

[4]:
```
array([[1],
       [1],
       [0],
       [0],
       [0]])
```

4. Create the training an testing sets using the `train_test_split` function from `sklearn`.

[5]:
```
# Create training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=78)
```

5. Scale the features data using the `StandardScaler` from `sklearn`.

[6]:
```
# Create the scaler instance
X_scaler = StandardScaler()
```

[7]:
```
# Fit the scaler
X_scaler.fit(X_train)
```

[7]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

[8]:
```
# Scale the features data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

6. Create a neural network model with `23` inputs, one hidden layer with `69` units, and an output layer with a single output. Use the `relu` activation function for the first layer and `sigmoid` for the second layer.

[9]:
```
# Define the model
number_inputs = 23
number_hidden_nodes = 69

nn = Sequential()
nn.add(Dense(units=number_hidden_nodes, input_dim=number_inputs, activation="relu"))
nn.add(Dense(1, activation="sigmoid"))
```

7. Compile the neural network model using the `binary_crossentropy` loss function, the `adam` optimizer, and `accuracy` as additional metric.

[10]:
```
# Compile model
nn.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
```

8. Fit the model with `100` epochs.

[11]:
```
# Fit the model
model = nn.fit(X_train_scaled, y_train, epochs=100)
```

```
Train on 22500 samples
Epoch 1/100
22500/22500 [==============================] - 2s 101us/sample - loss: 0.4803 - accuracy: 0.8029
Epoch 2/100
22500/22500 [==============================] - 1s 61us/sample - loss: 0.4515 - accuracy: 0.8159
Epoch 3/100
22500/22500 [==============================] - 1s 62us/sample - loss: 0.4441 - accuracy: 0.8180
Epoch 4/100
22500/22500 [==============================] - 1s 64us/sample - loss: 0.4406 - accuracy: 0.8190
Epoch 5/100
22500/22500 [==============================] - 1s 65us/sample - loss: 0.4372 - accuracy: 0.8193
Epoch 6/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4356 - accuracy: 0.8189
Epoch 7/100
22500/22500 [==============================] - 2s 69us/sample - loss: 0.4344 - accuracy: 0.8200
Epoch 8/100
22500/22500 [==============================] - 2s 69us/sample - loss: 0.4333 - accuracy: 0.8181
Epoch 9/100
22500/22500 [==============================] - 2s 73us/sample - loss: 0.4331 - accuracy: 0.8194
Epoch 10/100
22500/22500 [==============================] - 2s 73us/sample - loss: 0.4312 - accuracy: 0.8195
Epoch 11/100
22500/22500 [==============================] - 2s 78us/sample - loss: 0.4304 - accuracy: 0.8201
Epoch 12/100
22500/22500 [==============================] - 2s 78us/sample - loss: 0.4300 - accuracy: 0.8192
Epoch 13/100
22500/22500 [==============================] - 2s 76us/sample - loss: 0.4289 - accuracy: 0.8199
Epoch 14/100
22500/22500 [==============================] - 2s 78us/sample - loss: 0.4284 - accuracy: 0.8200
Epoch 15/100
22500/22500 [==============================] - 2s 78us/sample - loss: 0.4274 - accuracy: 0.8197
Epoch 16/100
22500/22500 [==============================] - 2s 79us/sample - loss: 0.4269 - accuracy: 0.8196
Epoch 17/100
22500/22500 [==============================] - 2s 79us/sample - loss: 0.4263 - accuracy: 0.8208
Epoch 18/100
22500/22500 [==============================] - 2s 78us/sample - loss: 0.4269 - accuracy: 0.8214
Epoch 19/100
22500/22500 [==============================] - 2s 78us/sample - loss: 0.4255 - accuracy: 0.8211
Epoch 20/100
22500/22500 [==============================] - 2s 79us/sample - loss: 0.4250 - accuracy: 0.8204
Epoch 21/100
22500/22500 [==============================] - 2s 84us/sample - loss: 0.4245 - accuracy: 0.8212
Epoch 22/100
22500/22500 [==============================] - 2s 84us/sample - loss: 0.4249 - accuracy: 0.8214
Epoch 23/100
22500/22500 [==============================] - 2s 84us/sample - loss: 0.4239 - accuracy: 0.8220
Epoch 24/100
22500/22500 [==============================] - 2s 84us/sample - loss: 0.4240 - accuracy: 0.8204
Epoch 25/100
22500/22500 [==============================] - 2s 83us/sample - loss: 0.4234 - accuracy: 0.8217
Epoch 26/100
22500/22500 [==============================] - 2s 83us/sample - loss: 0.4228 - accuracy: 0.8223
Epoch 27/100
22500/22500 [==============================] - 2s 83us/sample - loss: 0.4221 - accuracy: 0.8223
Epoch 28/100
22500/22500 [==============================] - 2s 84us/sample - loss: 0.4218 - accuracy: 0.8215
Epoch 29/100
22500/22500 [==============================] - 2s 84us/sample - loss: 0.4212 - accuracy: 0.8212
Epoch 30/100
22500/22500 [==============================] - 2s 84us/sample - loss: 0.4217 - accuracy: 0.8222
Epoch 31/100
22500/22500 [==============================] - 2s 87us/sample - loss: 0.4215 - accuracy: 0.8226
Epoch 32/100
22500/22500 [==============================] - 2s 94us/sample - loss: 0.4210 - accuracy: 0.8215
Epoch 33/100
22500/22500 [==============================] - 2s 92us/sample - loss: 0.4208 - accuracy: 0.8231
Epoch 34/100
22500/22500 [==============================] - 2s 90us/sample - loss: 0.4208 - accuracy: 0.8226
Epoch 35/100
```

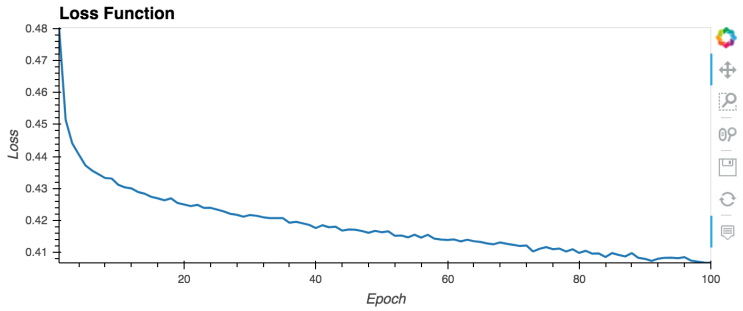
```
22500/22500 [==============================] - 2s 90us/sample - loss: 0.4207 - accuracy: 0.8220
Epoch 36/100
22500/22500 [==============================] - 2s 90us/sample - loss: 0.4193 - accuracy: 0.8235
Epoch 37/100
22500/22500 [==============================] - 2s 91us/sample - loss: 0.4196 - accuracy: 0.8229
Epoch 38/100
22500/22500 [==============================] - 2s 90us/sample - loss: 0.4191 - accuracy: 0.8234
Epoch 39/100
22500/22500 [==============================] - 2s 89us/sample - loss: 0.4186 - accuracy: 0.8230
Epoch 40/100
22500/22500 [==============================] - 2s 90us/sample - loss: 0.4177 - accuracy: 0.8228
Epoch 41/100
22500/22500 [==============================] - 2s 89us/sample - loss: 0.4185 - accuracy: 0.8228
Epoch 42/100
22500/22500 [==============================] - 2s 89us/sample - loss: 0.4179 - accuracy: 0.8219
Epoch 43/100
22500/22500 [==============================] - 2s 89us/sample - loss: 0.4181 - accuracy: 0.8234
Epoch 44/100
22500/22500 [==============================] - 2s 86us/sample - loss: 0.4168 - accuracy: 0.8239
Epoch 45/100
22500/22500 [==============================] - 2s 82us/sample - loss: 0.4172 - accuracy: 0.8230
Epoch 46/100
22500/22500 [==============================] - 2s 83us/sample - loss: 0.4171 - accuracy: 0.8223
Epoch 47/100
22500/22500 [==============================] - 2s 83us/sample - loss: 0.4167 - accuracy: 0.8232
Epoch 48/100
22500/22500 [==============================] - 2s 81us/sample - loss: 0.4162 - accuracy: 0.8232
Epoch 49/100
22500/22500 [==============================] - 2s 77us/sample - loss: 0.4168 - accuracy: 0.8231
Epoch 50/100
22500/22500 [==============================] - 2s 77us/sample - loss: 0.4163 - accuracy: 0.8238
Epoch 51/100
22500/22500 [==============================] - 2s 72us/sample - loss: 0.4166 - accuracy: 0.8232
Epoch 52/100
22500/22500 [==============================] - 2s 72us/sample - loss: 0.4152 - accuracy: 0.8248
Epoch 53/100
22500/22500 [==============================] - 2s 72us/sample - loss: 0.4153 - accuracy: 0.8253
Epoch 54/100
22500/22500 [==============================] - 2s 72us/sample - loss: 0.4148 - accuracy: 0.8240
Epoch 55/100
22500/22500 [==============================] - 2s 72us/sample - loss: 0.4156 - accuracy: 0.8255
Epoch 56/100
22500/22500 [==============================] - 2s 72us/sample - loss: 0.4147 - accuracy: 0.8241
Epoch 57/100
22500/22500 [==============================] - 2s 73us/sample - loss: 0.4155 - accuracy: 0.8245
Epoch 58/100
22500/22500 [==============================] - 2s 72us/sample - loss: 0.4144 - accuracy: 0.8248
Epoch 59/100
22500/22500 [==============================] - 2s 72us/sample - loss: 0.4141 - accuracy: 0.8248
Epoch 60/100
22500/22500 [==============================] - 2s 72us/sample - loss: 0.4139 - accuracy: 0.8261
Epoch 61/100
22500/22500 [==============================] - 2s 71us/sample - loss: 0.4141 - accuracy: 0.8257
Epoch 62/100
22500/22500 [==============================] - 2s 72us/sample - loss: 0.4135 - accuracy: 0.8247
Epoch 63/100
22500/22500 [==============================] - 2s 72us/sample - loss: 0.4140 - accuracy: 0.8247
Epoch 64/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4135 - accuracy: 0.8245
Epoch 65/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4133 - accuracy: 0.8251
Epoch 66/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4129 - accuracy: 0.8249
Epoch 67/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4126 - accuracy: 0.8244
Epoch 68/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4132 - accuracy: 0.8254
Epoch 69/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4127 - accuracy: 0.8248
Epoch 70/100
22500/22500 [==============================] - 1s 65us/sample - loss: 0.4124 - accuracy: 0.8262
Epoch 71/100
22500/22500 [==============================] - 1s 64us/sample - loss: 0.4120 - accuracy: 0.8261
Epoch 72/100
22500/22500 [==============================] - 1s 64us/sample - loss: 0.4122 - accuracy: 0.8258
Epoch 73/100
22500/22500 [==============================] - 1s 64us/sample - loss: 0.4103 - accuracy: 0.8259
Epoch 74/100
22500/22500 [==============================] - 1s 63us/sample - loss: 0.4112 - accuracy: 0.8256
Epoch 75/100
22500/22500 [==============================] - 1s 63us/sample - loss: 0.4117 - accuracy: 0.8250
Epoch 76/100
22500/22500 [==============================] - 1s 64us/sample - loss: 0.4110 - accuracy: 0.8251
Epoch 77/100
22500/22500 [==============================] - 1s 64us/sample - loss: 0.4112 - accuracy: 0.8244
Epoch 78/100
22500/22500 [==============================] - 1s 65us/sample - loss: 0.4103 - accuracy: 0.8250
Epoch 79/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4110 - accuracy: 0.8252
Epoch 80/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4099 - accuracy: 0.8271
Epoch 81/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4105 - accuracy: 0.8281
Epoch 82/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4097 - accuracy: 0.8262
Epoch 83/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4097 - accuracy: 0.8255
Epoch 84/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4086 - accuracy: 0.8264
Epoch 85/100
22500/22500 [==============================] - 2s 68us/sample - loss: 0.4098 - accuracy: 0.8249
Epoch 86/100
22500/22500 [==============================] - 2s 75us/sample - loss: 0.4093 - accuracy: 0.8272
Epoch 87/100
22500/22500 [==============================] - 2s 72us/sample - loss: 0.4088 - accuracy: 0.8264
Epoch 88/100
22500/22500 [==============================] - 2s 72us/sample - loss: 0.4098 - accuracy: 0.8268
Epoch 89/100
22500/22500 [==============================] - 2s 78us/sample - loss: 0.4083 - accuracy: 0.8258
Epoch 90/100
22500/22500 [==============================] - 2s 78us/sample - loss: 0.4080 - accuracy: 0.8262
```

```
Epoch 91/100
22500/22500 [==============================] - 2s 84us/sample - loss: 0.4074 - accuracy: 0.8266
Epoch 92/100
22500/22500 [==============================] - 2s 81us/sample - loss: 0.4081 - accuracy: 0.8259
Epoch 93/100
22500/22500 [==============================] - 2s 76us/sample - loss: 0.4084 - accuracy: 0.8257
Epoch 94/100
22500/22500 [==============================] - 2s 77us/sample - loss: 0.4084 - accuracy: 0.8256
Epoch 95/100
22500/22500 [==============================] - 2s 77us/sample - loss: 0.4082 - accuracy: 0.8273
Epoch 96/100
22500/22500 [==============================] - 2s 82us/sample - loss: 0.4085 - accuracy: 0.8263
Epoch 97/100
22500/22500 [==============================] - 2s 84us/sample - loss: 0.4075 - accuracy: 0.8266
Epoch 98/100
22500/22500 [==============================] - 2s 83us/sample - loss: 0.4072 - accuracy: 0.8260
Epoch 99/100
22500/22500 [==============================] - 2s 84us/sample - loss: 0.4069 - accuracy: 0.8278
Epoch 100/100
22500/22500 [==============================] - 2s 81us/sample - loss: 0.4068 - accuracy: 0.8277
```

9. Plot the loss function and accuracy using `hvplot`.

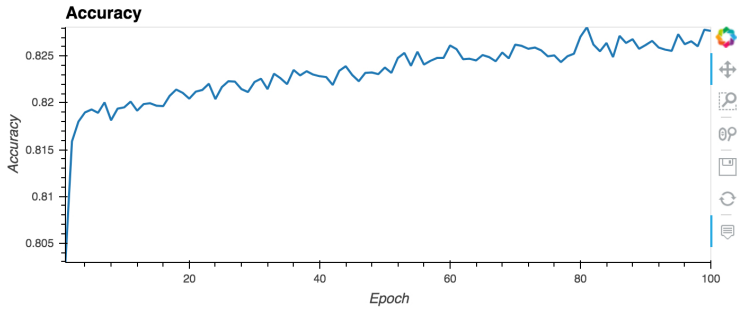
[12]:
```
# Create a dataframe with the history dictionary
df_plot = pd.DataFrame(model.history, index=range(1, len(model.history["loss"]) + 1))

# Plot the loss
df_plot.hvplot(title="Loss Function", y="loss", xlabel="Epoch", ylabel="Loss")
```

[12]:



[13]:
```
# Plot the accuracy
df_plot.hvplot(title="Accuracy", y="accuracy", xlabel="Epoch", ylabel="Accuracy")
```

[13]:



10. Evaluate the model using testing data and the `evaluate` method.

[14]:
```
# Evaluate the model fit with linear dummy data
model_loss, model_accuracy = nn.evaluate(X_test_scaled, y_test, verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
7500/1 - 1s - loss: 0.4306 - accuracy: 0.8127
Loss: 0.4452823066711426, Accuracy: 0.812666654586792
```

Challenge

For this challenge section, you have to figure out how the model you created could be modified to improve the accuracy.

1. Review the Keras documentation about activation functions, and decide if there is anyone that could be used instead of `sigmoid`.
2. Add a second hidden layer with `69` units and use a different activation function than `sigmoid`.
3. Change any other parameter that you believe could improve the model's accuracy.
4. Evaluate the model's accuracy and loss and write down your conclusions.

[15]:
```
# Define the model
number_inputs = 23
number_hidden_nodes = 69

nn_2 = Sequential()
nn_2.add(Dense(units=number_hidden_nodes, input_dim=number_inputs, activation="relu"))
nn_2.add(Dense(units=number_hidden_nodes, activation="tanh"))
nn_2.add(Dense(units=1, activation="tanh"))
```

[16]:
```
# Compile model
nn_2.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
```

[17]:
```
# Fit the model
model_2 = nn_2.fit(X_train_scaled, y_train, epochs=50)
```

```
Train on 22500 samples
Epoch 1/50
22500/22500 [==============================] - 3s 128us/sample - loss: 0.8266 - accuracy: 0.6902
Epoch 2/50
22500/22500 [==============================] - 2s 90us/sample - loss: 0.6876 - accuracy: 0.7368
Epoch 3/50
22500/22500 [==============================] - 2s 88us/sample - loss: 0.9363 - accuracy: 0.6593
Epoch 4/50
22500/22500 [==============================] - 2s 89us/sample - loss: 0.7005 - accuracy: 0.7391
Epoch 5/50
22500/22500 [==============================] - 2s 91us/sample - loss: 0.5604 - accuracy: 0.7979
Epoch 6/50
22500/22500 [==============================] - 2s 91us/sample - loss: 0.6411 - accuracy: 0.7546
Epoch 7/50
22500/22500 [==============================] - 2s 90us/sample - loss: 0.5761 - accuracy: 0.7746
Epoch 8/50
22500/22500 [==============================] - 2s 90us/sample - loss: 0.5179 - accuracy: 0.8068
Epoch 9/50
22500/22500 [==============================] - 2s 88us/sample - loss: 0.5591 - accuracy: 0.7750
Epoch 10/50
22500/22500 [==============================] - 2s 90us/sample - loss: 0.5371 - accuracy: 0.7756
Epoch 11/50
22500/22500 [==============================] - 2s 90us/sample - loss: 0.5888 - accuracy: 0.7567
Epoch 12/50
22500/22500 [==============================] - 2s 91us/sample - loss: 0.6875 - accuracy: 0.7176
Epoch 13/50
22500/22500 [==============================] - 2s 90us/sample - loss: 0.4886 - accuracy: 0.8110
Epoch 14/50
22500/22500 [==============================] - 2s 89us/sample - loss: 0.5313 - accuracy: 0.7751
Epoch 15/50
22500/22500 [==============================] - 2s 95us/sample - loss: 0.5078 - accuracy: 0.8070
Epoch 16/50
22500/22500 [==============================] - 2s 99us/sample - loss: 0.4865 - accuracy: 0.8156
Epoch 17/50
22500/22500 [==============================] - 2s 97us/sample - loss: 0.4813 - accuracy: 0.8034
Epoch 18/50
22500/22500 [==============================] - 2s 98us/sample - loss: 0.4748 - accuracy: 0.8108
Epoch 19/50
22500/22500 [==============================] - 2s 97us/sample - loss: 0.5542 - accuracy: 0.7571
Epoch 20/50
22500/22500 [==============================] - 2s 98us/sample - loss: 0.4770 - accuracy: 0.8129
Epoch 21/50
22500/22500 [==============================] - 2s 97us/sample - loss: 0.4842 - accuracy: 0.8044
Epoch 22/50
22500/22500 [==============================] - 2s 96us/sample - loss: 0.5231 - accuracy: 0.7710
Epoch 23/50
22500/22500 [==============================] - 2s 98us/sample - loss: 0.4770 - accuracy: 0.8104
Epoch 24/50
22500/22500 [==============================] - 2s 98us/sample - loss: 0.4554 - accuracy: 0.8182
Epoch 25/50
22500/22500 [==============================] - 2s 98us/sample - loss: 0.4485 - accuracy: 0.8168
Epoch 26/50
22500/22500 [==============================] - 2s 97us/sample - loss: 0.4679 - accuracy: 0.8028
Epoch 27/50
22500/22500 [==============================] - 2s 96us/sample - loss: 0.4931 - accuracy: 0.7891
Epoch 28/50
22500/22500 [==============================] - 2s 99us/sample - loss: 0.5142 - accuracy: 0.7821s - loss: 0.5201 - accuracy:
Epoch 29/50
22500/22500 [==============================] - 2s 96us/sample - loss: 0.4642 - accuracy: 0.8166
Epoch 30/50
22500/22500 [==============================] - 2s 98us/sample - loss: 0.4539 - accuracy: 0.8186
Epoch 31/50
22500/22500 [==============================] - 2s 98us/sample - loss: 0.5517 - accuracy: 0.7725
Epoch 32/50
22500/22500 [==============================] - 2s 97us/sample - loss: 0.4736 - accuracy: 0.8132
Epoch 33/50
22500/22500 [==============================] - 2s 98us/sample - loss: 0.4644 - accuracy: 0.8173
Epoch 34/50
22500/22500 [==============================] - 2s 97us/sample - loss: 0.4755 - accuracy: 0.8066
Epoch 35/50
22500/22500 [==============================] - 2s 97us/sample - loss: 0.4603 - accuracy: 0.8165
Epoch 36/50
22500/22500 [==============================] - 2s 98us/sample - loss: 0.4675 - accuracy: 0.8144
Epoch 37/50
22500/22500 [==============================] - 2s 99us/sample - loss: 0.5896 - accuracy: 0.7429
Epoch 38/50
22500/22500 [==============================] - 2s 97us/sample - loss: 0.5524 - accuracy: 0.7539
Epoch 39/50
22500/22500 [==============================] - 2s 98us/sample - loss: 0.4581 - accuracy: 0.8168
Epoch 40/50
22500/22500 [==============================] - 2s 99us/sample - loss: 0.4489 - accuracy: 0.8195
Epoch 41/50
22500/22500 [==============================] - 2s 97us/sample - loss: 0.4446 - accuracy: 0.8203
Epoch 42/50
22500/22500 [==============================] - 2s 96us/sample - loss: 0.4517 - accuracy: 0.8102
Epoch 43/50
22500/22500 [==============================] - 2s 97us/sample - loss: 0.4490 - accuracy: 0.8147
Epoch 44/50
22500/22500 [==============================] - 2s 97us/sample - loss: 0.4417 - accuracy: 0.8204
Epoch 45/50
22500/22500 [==============================] - 2s 97us/sample - loss: 0.4482 - accuracy: 0.8183
Epoch 46/50
22500/22500 [==============================] - 2s 97us/sample - loss: 0.4483 - accuracy: 0.8127
Epoch 47/50
22500/22500 [==============================] - 2s 95us/sample - loss: 0.4428 - accuracy: 0.8196
Epoch 48/50
22500/22500 [==============================] - 2s 91us/sample - loss: 0.4377 - accuracy: 0.8209
Epoch 49/50
22500/22500 [==============================] - 2s 90us/sample - loss: 0.4622 - accuracy: 0.8069
Epoch 50/50
22500/22500 [==============================] - 2s 89us/sample - loss: 0.4417 - accuracy: 0.8186
```

[18]:
```
# Evaluate the model fit with linear dummy data
model_loss, model_accuracy = nn_2.evaluate(X_test_scaled, y_test, verbose=2)
```

```
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
7500/1 - 1s - loss: 0.4364 - accuracy: 0.8240
Loss: 0.46841058138211566, Accuracy: 0.8240000009536743
```

Conclusions

Sample Answer: After reviewing the activation functions documentation, I decided to use the `tanh` function in both hidden layers. There is an improvement in the model's accuracy, but it's not as good as expected.

We may test with different functions or even perform PCA as part of the possible optimizations.