Group 4
Deep Learning Project

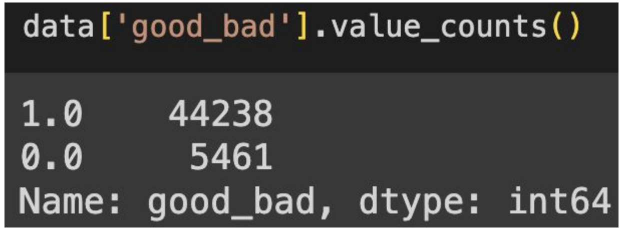
Group Members:
Eric Chaves
Kwame Sefa-Boateng
Jordyn Dolly
Han Zhang

The Problem

- The primary goal of the model is to use consumer credit features to predict if a borrower will repay the loan.
- This is useful in the area of finance when a financial institution like a bank or investment company is determining whether to finance a loan.
- We deploy the deep learning models such as Deep Neural Network (DNN), Multilayer Perceptron (MLP), and Wide & Deep Neural Network models to aid the prediction of possible default of loan.

The Dataset

- This is a publicly available dataset with loan information between the years of 2007 through 2014
- Our dataset consists of 395,932 loan samples and 31 parameters
- Parameter examples
  - Loan amount, annual income, debt to income ratio, number of open accounts, revolving balance, employment length, etc..
- Our target variable is ‘good_bad’
  - ‘good_bad’ = 1 when the borrower repays the loan
  - ‘good_bad’ = 0 when the loan is not repaid



```
data['good_bad'].value_counts()

1.0    44238
0.0     5461
Name: good_bad, dtype: int64
```

Deep Learning Models

Deep Neural Network Model

- A DNN can be extremely useful in determining if an individual will likely default on a loan repayment.
  - Ability to learn non-linear relationships
  - Automatically learn the hierarchical representations of features
  - Handles high dimensional data well
  - High flexibility by experimenting with number of layers, number of neurons, and activation functions
  - DNN's allow for end to end learning. Therefore the model can learn directly from the raw data to the final predictions.
- However, DNN's can be more difficult to interpret and can experience the vanishing gradient.

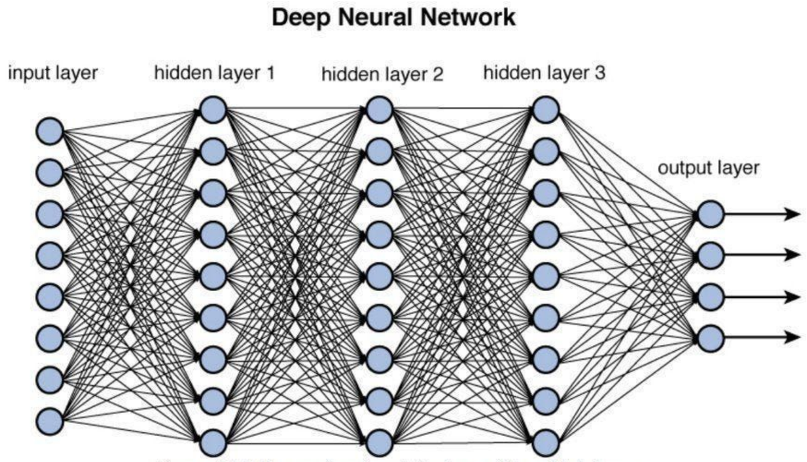
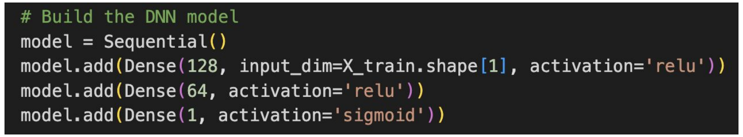


Figure 12.2 Deep network architecture with multiple layers.

Image sourced from https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

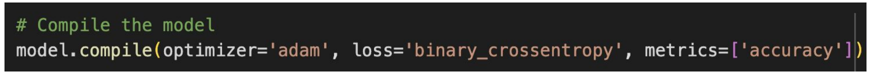
DNN Implementation



```python
# Build the DNN model
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

To gain a basic understanding of the model and to not overfit, a simple three layer model was used.
When building the DNN model:

1. The first layer introduces the Dense Layer with 128 neurons. Additionally, for the activation function we had chosen a rectified linear unit (ReLU). The activation function introduces nonlinearity to the data.
2. The second layer is another Dense Layer with 64 neurons to continue transforming the data.
3. The final layer has a single neuron, indicating that the network is designed for binary classification. The activation function used for the output layer is the sigmoid function, which places the output between 0 and 1.



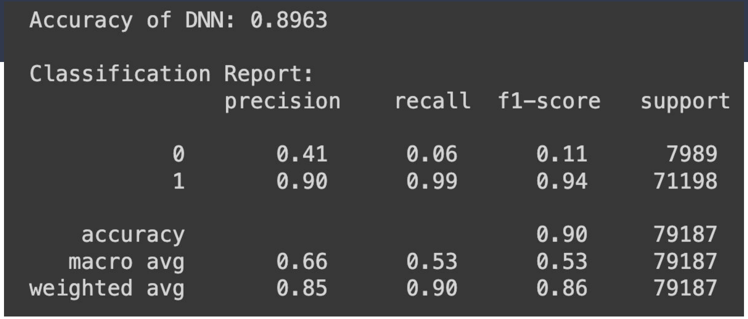
```python
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

When compiling the model for training:

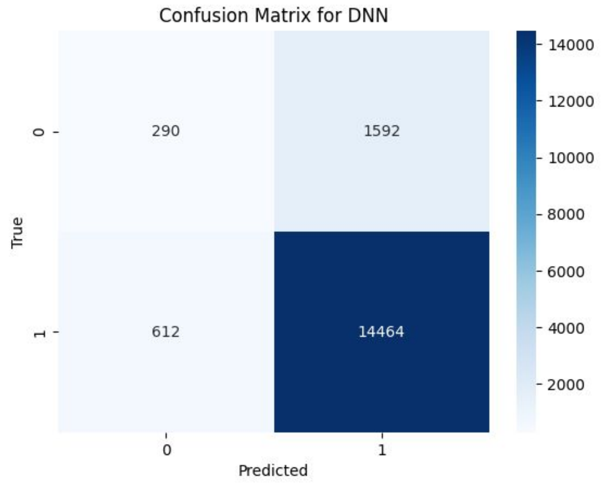
1. Optimizer (`optimizer='adam'`):
   a. The optimizer is responsible for updating the weights of the neural network during training. We used `'adam'` as it has an adaptive learning rate and is efficient.
2. Loss Function (`loss='binary_crossentropy'`):
   a. The loss function is a measure of how well the model is performing. We used `'binary_crossentropy'` as it calculates the binary cross-entropy between predicted probabilities and true labels.
3. Metrics (`metrics=['accuracy']`):
   a. We used `'accuracy'` to measure the proportion of correctly predicted instances as it is a simple metric to compare across multiple models.

DNN Evaluation

- The basic DNN model produced a relatively high accuracy of 0.8963
- The precision of 0 or when a loan is not repaid is low. This is one indication that our model has a high number of false positives.
- The recall appears very good for our loan repayment group showing almost no false negatives
- Knowing that false positives may be an issue we created a Confusion Matrix
  - False Negative: 290
  - False Positive: 1,592
  - True Negative: 612
  - True Positive 14,664



```
Accuracy of DNN: 0.8963

Classification Report:
              precision    recall  f1-score   support

           0       0.41      0.06      0.11      7989
           1       0.90      0.99      0.94     71198

    accuracy                           0.90     79187
   macro avg       0.66      0.53      0.53     79187
weighted avg       0.85      0.90      0.86     79187
```



MLP (Multilayer Perceptron)

1. Components of an MLP
    a. Input layer - receives the input signal to be processed
    b. Hidden layers - neurons that apply a weighted sum on their inputs
    c. Output layer - one neuron for binary classification
    d. Activation Functions - allow the model to learn more complex relationships
    e. Weights and Biases - neuron in hidden and output layer has weights and bias. Weight represent strength of connection between neurons

2. How MLP Works
    a. **Forward Propagation** - data fed into input layer. Each neuron in the subsequent layers takes the weighted sum of its inputs and applies an activation function. Continues until output layer is reached
    b. **Backpropagation and optimization** - the models prediction is compared with the target using a loss function. The weights and biases of the neurons are adjusted to minimize loss.
    c. **Learning -** The process of forward propagation, loss calculation, back propagation, and optimizing weights is repeated over the epochs.

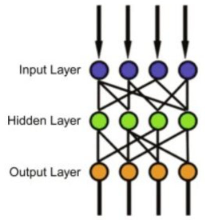


Image sourced from https://www.sciencedirect.com/topics/computer-science/multilayer-perceptron#:~:text=The%20input%20layer%20receives%20the,computational%20engine%20of%20the%20MLP.

MLP Implementation

Model Settings

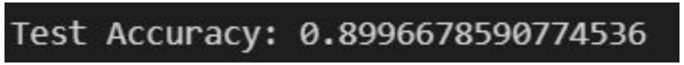
- Start with a dense layer of 64 neurons and uses relu activation function
- dropout layer with a dropout rate of 0.5, used to prevent overfitting
- Second dense layer of 64 neurons with relu activation
- Second dropout layer with dropout rate of 0.5
- The final dense layer with 1 neuron uses a sigmoid activation function suitable for binary classification
- Model is compiled with the adam optimizer

Training Strategy

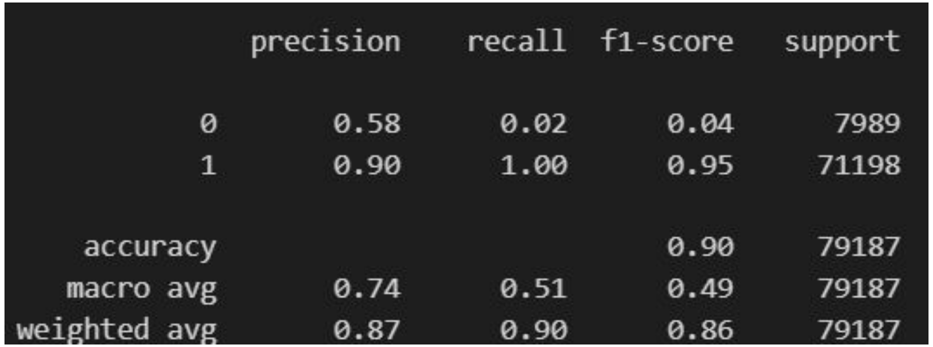
- Dataset is split into training and testing sets with an 80-20 split
- Trained for 10 epochs with a batch size of 32
- A validation split of 20% is used during training to monitor the model’s performance on a validation set

MLP Results

- Accuracy ≅ 0.90
- F1-score
  - Class 0 model has very low performance
  - Class 1 model has very high performance
- Low recall for class 0 indicates model fails to identify a significant portion of actual bad loans.
- Class 0 precision of .58 indicates the model is correct a little over half the time when predicting a loan will default.



```
Test Accuracy: 0.8996678590774536
```



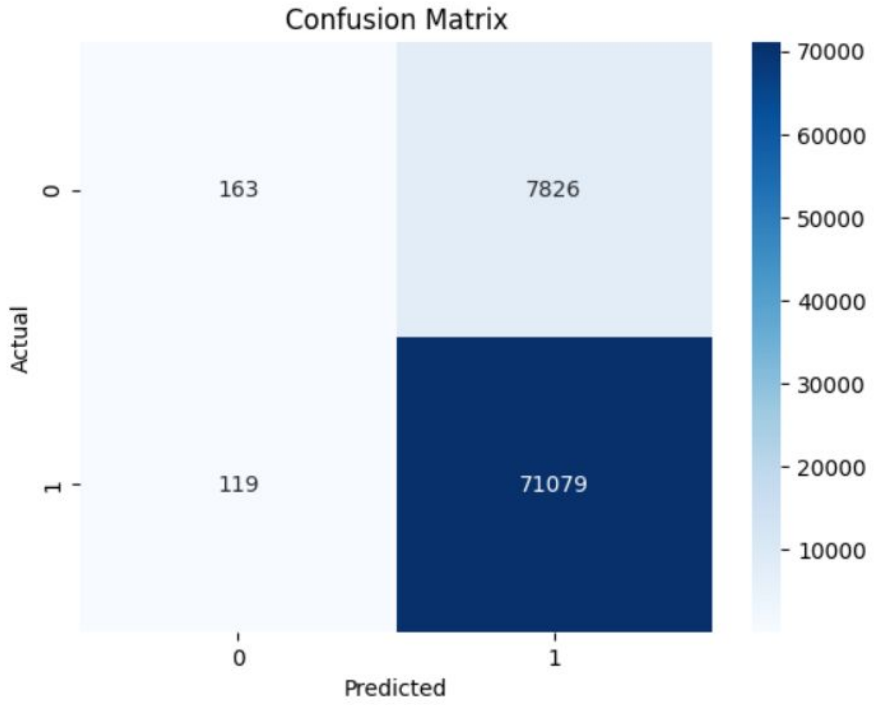
```
              precision    recall  f1-score   support

           0       0.58      0.02      0.04      7989
           1       0.90      1.00      0.95     71198

    accuracy                           0.90     79187
   macro avg       0.74      0.51      0.49     79187
weighted avg       0.87      0.90      0.86     79187
```

MLP Results

- True positive = 71079
- True negative = 163
- False Positive = 7826
- False Negative = 119

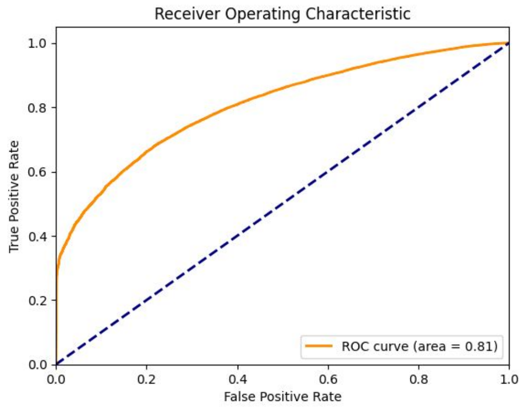
➔ Sensitivity 99.83%
➔ Specificity 2.04%
➔ Precision 90.08%
➔ Error Rate 10.03%

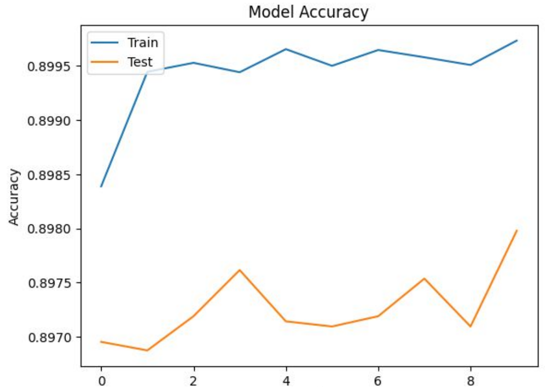
The model is very accurate in identifying positive cases (loan repayment), but struggles to identify negative cases (loan default).

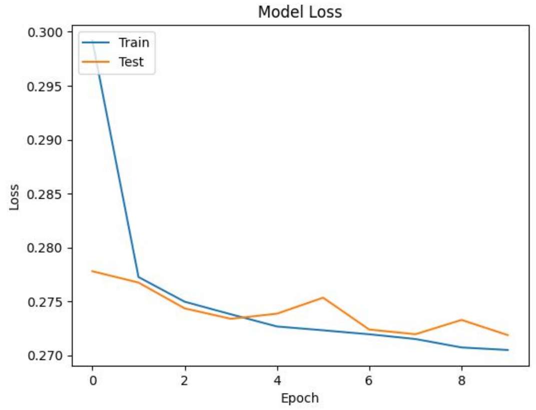


MLP Results

1. ROC Curve
   a. The area under the curve is 0.81 which suggests the model has a good discriminative ability
2. Model Loss & Accuracy
   a. Shows signs of possible overfitting
   b. The training metrics are consistently better than the testing metrics
   c. There is a variance in the testing accuracy which does not improve with more epochs







Wide & Deep Neural Network Model

The key idea is to combine the memorization capability of a wide model with the generalization ability of a deep model.

Deep Component:

Construct using Dense layers.

Utilize ReLU activation function for learning complex patterns in the deep neural network.

Wide Component:

Integrate a linear model for memorizing rule-based patterns.

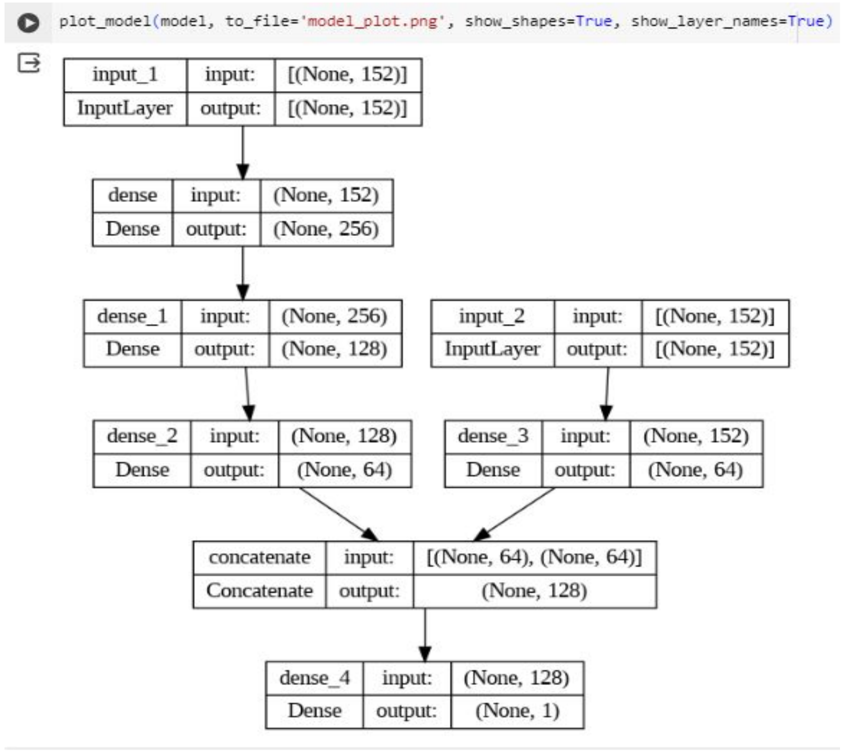
Combining Components:

Merge the deep and wide components to create a comprehensive model structure.

Output Layer:

Implement Sigmoid activation for binary classification tasks.

Wide & Deep Neural Network Model Implementation

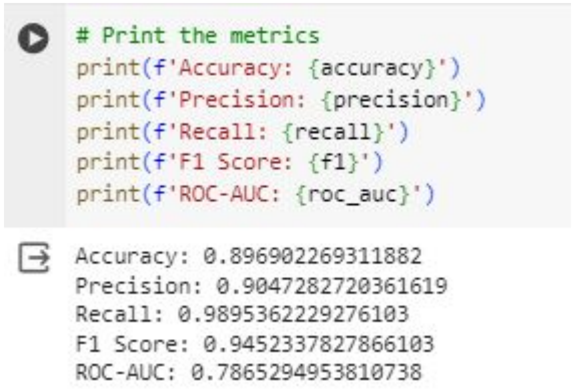


```
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

input_1 | InputLayer | input: [(None, 152)] | output: [(None, 152)]
dense | Dense | input: (None, 152) | output: (None, 256)
dense_1 | Dense | input: (None, 256) | output: (None, 128)
input_2 | InputLayer | input: [(None, 152)] | output: [(None, 152)]
dense_2 | Dense | input: (None, 128) | output: (None, 64)
dense_3 | Dense | input: (None, 152) | output: (None, 64)
concatenate | Concatenate | input: [(None, 64), (None, 64)] | output: (None, 128)
dense_4 | Dense | input: (None, 128) | output: (None, 1)

Overall, this model architecture diagram indicates a Deep & Wide model suitable for binary classification tasks with tabular data. The deep part of the model is designed to learn complex patterns through multiple layers of transformation, while the wide part can capture simple, rule-based information. The final output is structured to provide a binary classification decision.

Wide & Deep Neural Network Model Evaluation



```
# Print the metrics
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
print(f'ROC-AUC: {roc_auc}')
```

```
Accuracy: 0.896902269311882
Precision: 0.9047282720361619
Recall: 0.9895362229276103
F1 Score: 0.945233782786610З
ROC-AUC: 0.7865294953810738
```

The high F1 score implies that the model is performing well in terms of precision and recall, while the ROC-AUC value suggests that the model has a fair classifying ability. These results would typically be considered good for the model.

Conclusion

DNN

- A simple model that appeared to be overfitting the data due to the imbalance of 0’s and 1’s in the data
- Accuracy score of 0.8963 with a recall of 0.99

MLP

- Versatile and can be used for various tasks
- Particularly useful for structured data such as our dataset
- The model is highly accurate at 0.89967 with perfect recall

Deep & Wide NN

- Combined the memorization capability of a wide model with the generalization ability of a deep model
- The model accuracy was high at 0.8960 with a recall of 0.99

While all the models are highly accurate and have extremely high recall values, the MLP model is superior. The combination of its versatility and simplicity allowed us to reach the highest accuracy and recall scores.