Neural Network Performance Analysis

This notebook explores various neural network architectures to predict sentiment in the IMDB dataset. Different models were experimented with, varying in hidden layers, hidden units, and loss functions. Below is the analysis of these models.

In [41]:
```
%matplotlib inline
!jupyter nbconvert --to html Neural_Network_Assignment.ipynb
from google.colab import files
files.download('Neural_Network_Assignment.html')
```

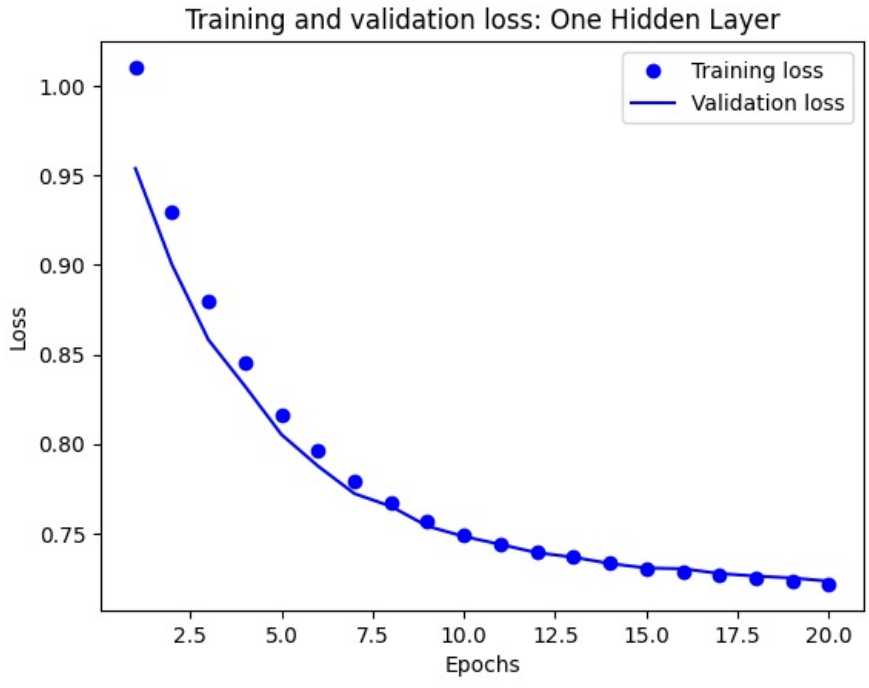
```
[NbConvertApp] Converting notebook Neural_Network_Assignment.ipynb to html
[NbConvertApp] Writing 593635 bytes to Neural_Network_Assignment.html
```

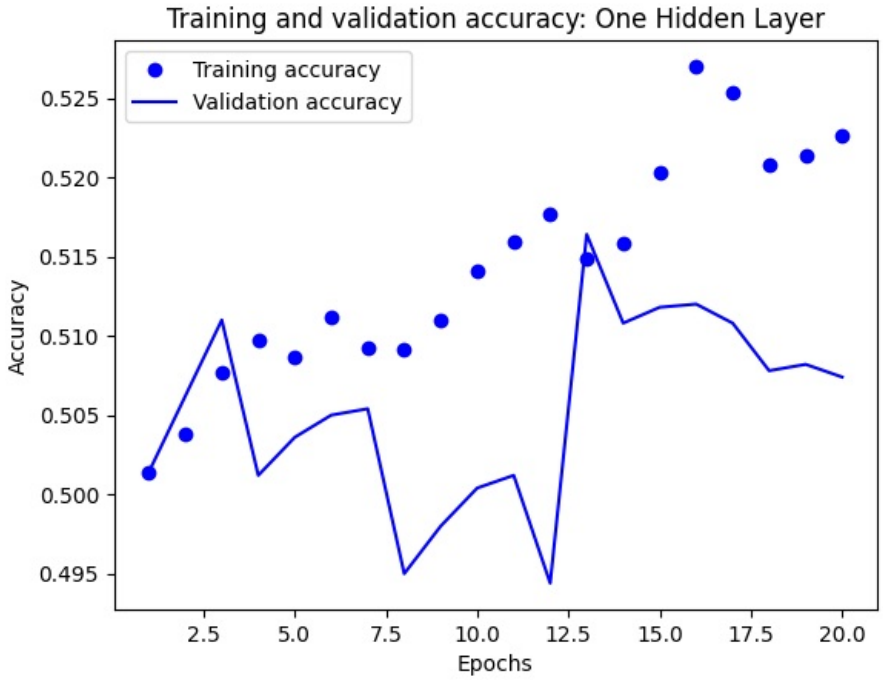
Model with One Hidden Layer

- **Overview**: The model was trained using a single hidden layer with 32 units and the `tanh` activation function.
- **Performance**: The training accuracy slowly improved but remained unstable across the epochs. Validation accuracy fluctuated and settled around 50%, indicating possible underfitting.
- **Conclusion**: Using only one hidden layer resulted in suboptimal performance, which might be due to insufficient model complexity.

In [ ]:
```
# Import necessary modules
from tensorflow.keras import models, layers, regularizers
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load the IMDB dataset
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)

# Preprocess the data (pad sequences to make them the same length)
partial_x_train = pad_sequences(train_data[:20000], maxlen=256)
partial_y_train = train_labels[:20000]
x_val = pad_sequences(train_data[20000:], maxlen=256)
y_val = train_labels[20000:]

# One hidden layer model
model_one_hidden_layer = models.Sequential([
    layers.Dense(32, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1, activation="sigmoid")
])

# Compile the model
model_one_hidden_layer.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["accuracy"])

# Train the model
history_one_hidden_layer = model_one_hidden_layer.fit(partial_x_train, partial_y_train, epochs=20, batch_size=5

import matplotlib.pyplot as plt

def plot_history(history, title):
    history_dict = history.history
    loss_values = history_dict["loss"]
    val_loss_values = history_dict["val_loss"]
    acc = history_dict.get("accuracy", None)
    val_acc = history_dict.get("val_accuracy", None)
    epochs = range(1, len(loss_values) + 1)

    # Plot training and validation loss
    plt.plot(epochs, loss_values, "bo", label="Training loss")
    plt.plot(epochs, val_loss_values, "b", label="Validation loss")
    plt.title(f"Training and validation loss: {title}")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()

    # If accuracy is available, plot training and validation accuracy
    if acc and val_acc:
        plt.clf()
        plt.plot(epochs, acc, "bo", label="Training accuracy")
        plt.plot(epochs, val_acc, "b", label="Validation accuracy")
        plt.title(f"Training and validation accuracy: {title}")
        plt.xlabel("Epochs")
        plt.ylabel("Accuracy")
        plt.legend()
        plt.show()
# Plot the results for one hidden layer
plot_history(history_one_hidden_layer, "One Hidden Layer")
```

```
Epoch 1/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.5035 - loss: 1.0487 - val_accuracy: 0.5014 - val_loss: 0.
9540
Epoch 2/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4985 - loss: 0.9489 - val_accuracy: 0.5062 - val_loss: 0.9
003
Epoch 3/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5090 - loss: 0.8887 - val_accuracy: 0.5110 - val_loss: 0.8
582
Epoch 4/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5113 - loss: 0.8534 - val_accuracy: 0.5012 - val_loss: 0.8
324
Epoch 5/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5071 - loss: 0.8221 - val_accuracy: 0.5036 - val_loss: 0.8
053
Epoch 6/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5102 - loss: 0.8016 - val_accuracy: 0.5050 - val_loss: 0.7
877
Epoch 7/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5132 - loss: 0.7800 - val_accuracy: 0.5054 - val_loss: 0.7
722
Epoch 8/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5107 - loss: 0.7690 - val_accuracy: 0.4950 - val_loss: 0.7
651
Epoch 9/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5102 - loss: 0.7590 - val_accuracy: 0.4980 - val_loss: 0.7
540
Epoch 10/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5131 - loss: 0.7505 - val_accuracy: 0.5004 - val_loss: 0.7
483
Epoch 11/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5188 - loss: 0.7445 - val_accuracy: 0.5012 - val_loss: 0.7
438
Epoch 12/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5203 - loss: 0.7405 - val_accuracy: 0.4944 - val_loss: 0.7
392
Epoch 13/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5158 - loss: 0.7374 - val_accuracy: 0.5164 - val_loss: 0.7
367
Epoch 14/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5174 - loss: 0.7336 - val_accuracy: 0.5108 - val_loss: 0.7
331
Epoch 15/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5156 - loss: 0.7311 - val_accuracy: 0.5118 - val_loss: 0.7
307
Epoch 16/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5213 - loss: 0.7297 - val_accuracy: 0.5120 - val_loss: 0.7
302
Epoch 17/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5219 - loss: 0.7273 - val_accuracy: 0.5108 - val_loss: 0.7
276
Epoch 18/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5261 - loss: 0.7244 - val_accuracy: 0.5078 - val_loss: 0.7
261
Epoch 19/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5239 - loss: 0.7232 - val_accuracy: 0.5082 - val_loss: 0.7
251
Epoch 20/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5207 - loss: 0.7225 - val_accuracy: 0.5074 - val_loss: 0.7
233
```



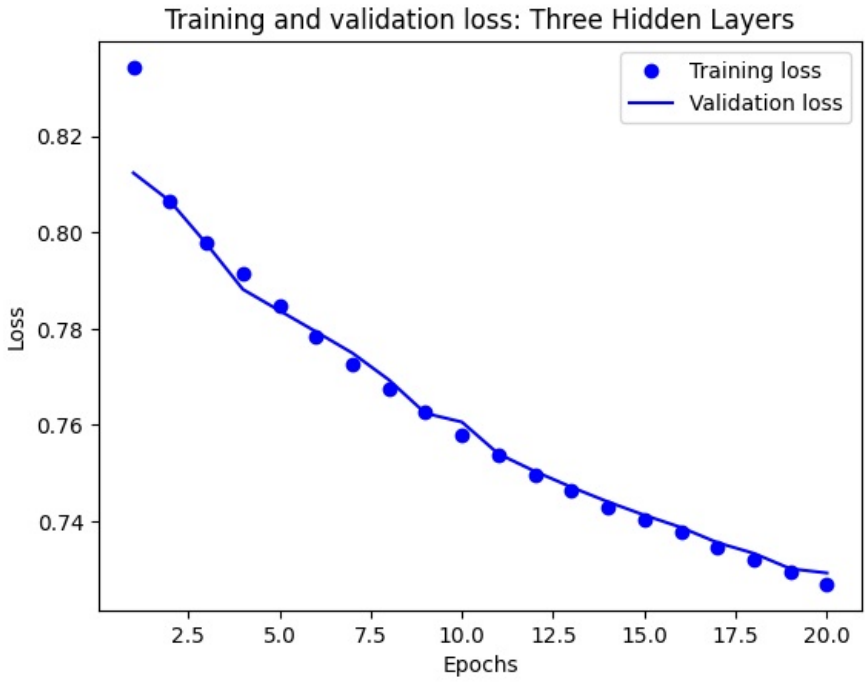


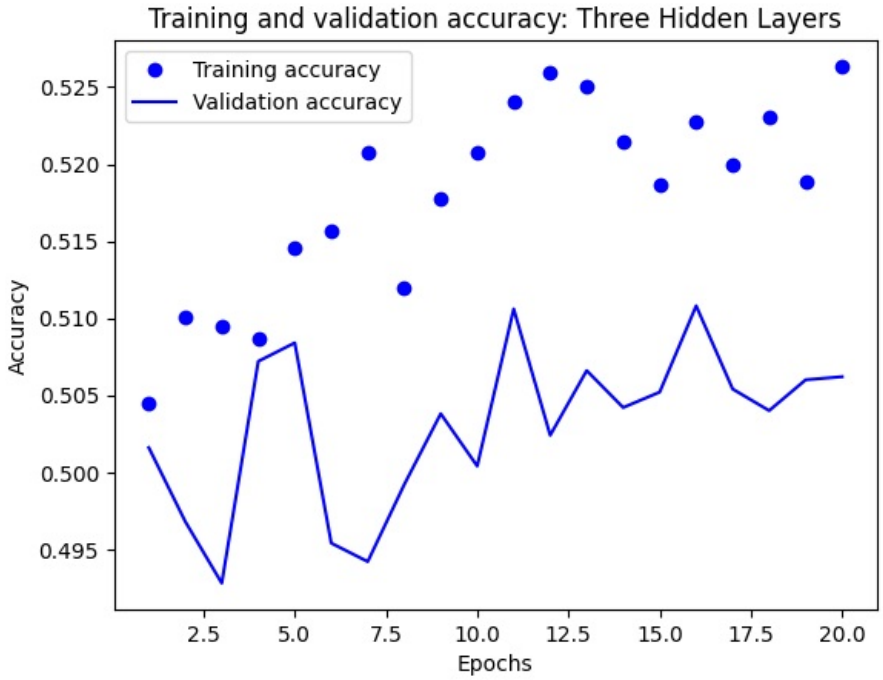
Model with Three Hidden Layers

- **Overview**: This model had three hidden layers with 32 units each.
- **Performance**: Training accuracy was slightly better than the one-hidden-layer model, but the validation accuracy hovered around 50%. Training loss consistently decreased, but validation loss did not improve much.
- **Conclusion**: Increasing the model complexity did not lead to better generalization, likely due to overfitting or a lack of sufficient regularization.

In [ ]:
```
# Three hidden layers model
model_three_hidden_layers = models.Sequential([
    layers.Dense(32, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(32, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(32, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1, activation="sigmoid")
])
model_three_hidden_layers.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["accuracy"])
history_three_hidden_layers = model_three_hidden_layers.fit(partial_x_train, partial_y_train, epochs=20, batch_

# Plot the results for three hidden layers
plot_history(history_three_hidden_layers, "Three Hidden Layers")
```

```
Epoch 1/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 4s 44ms/step - accuracy: 0.5028 - loss: 0.8572 - val_accuracy: 0.5016 - val_loss: 0.
8123
Epoch 2/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.5133 - loss: 0.8084 - val_accuracy: 0.4968 - val_loss: 0.
8065
Epoch 3/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.5141 - loss: 0.7985 - val_accuracy: 0.4928 - val_loss: 0.
7976
Epoch 4/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5110 - loss: 0.7922 - val_accuracy: 0.5072 - val_loss: 0.7
881
Epoch 5/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.5164 - loss: 0.7856 - val_accuracy: 0.5084 - val_loss: 0.7
837
Epoch 6/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.5217 - loss: 0.7788 - val_accuracy: 0.4954 - val_loss: 0.
7794
Epoch 7/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5234 - loss: 0.7735 - val_accuracy: 0.4942 - val_loss: 0.
7749
Epoch 8/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.5081 - loss: 0.7690 - val_accuracy: 0.4992 - val_loss: 0.
7693
Epoch 9/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5203 - loss: 0.7640 - val_accuracy: 0.5038 - val_loss: 0.7
624
Epoch 10/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5248 - loss: 0.7582 - val_accuracy: 0.5004 - val_loss: 0.7
606
Epoch 11/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5215 - loss: 0.7551 - val_accuracy: 0.5106 - val_loss: 0.7
540
Epoch 12/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5270 - loss: 0.7505 - val_accuracy: 0.5024 - val_loss: 0.7
503
Epoch 13/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5241 - loss: 0.7468 - val_accuracy: 0.5066 - val_loss: 0.7
471
Epoch 14/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5216 - loss: 0.7433 - val_accuracy: 0.5042 - val_loss: 0.7
440
Epoch 15/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5219 - loss: 0.7403 - val_accuracy: 0.5052 - val_loss: 0.7
412
Epoch 16/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5239 - loss: 0.7379 - val_accuracy: 0.5108 - val_loss: 0.7
387
Epoch 17/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5255 - loss: 0.7344 - val_accuracy: 0.5054 - val_loss: 0.7
355
Epoch 18/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5264 - loss: 0.7322 - val_accuracy: 0.5040 - val_loss: 0.7
333
Epoch 19/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5195 - loss: 0.7298 - val_accuracy: 0.5060 - val_loss: 0.7
301
Epoch 20/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5281 - loss: 0.7270 - val_accuracy: 0.5062 - val_loss: 0.7
292
```



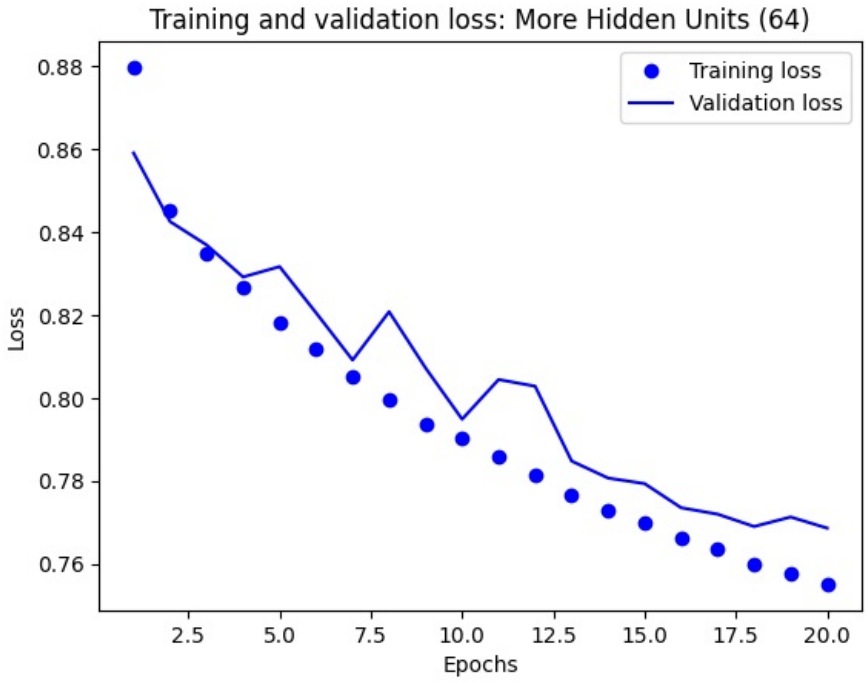


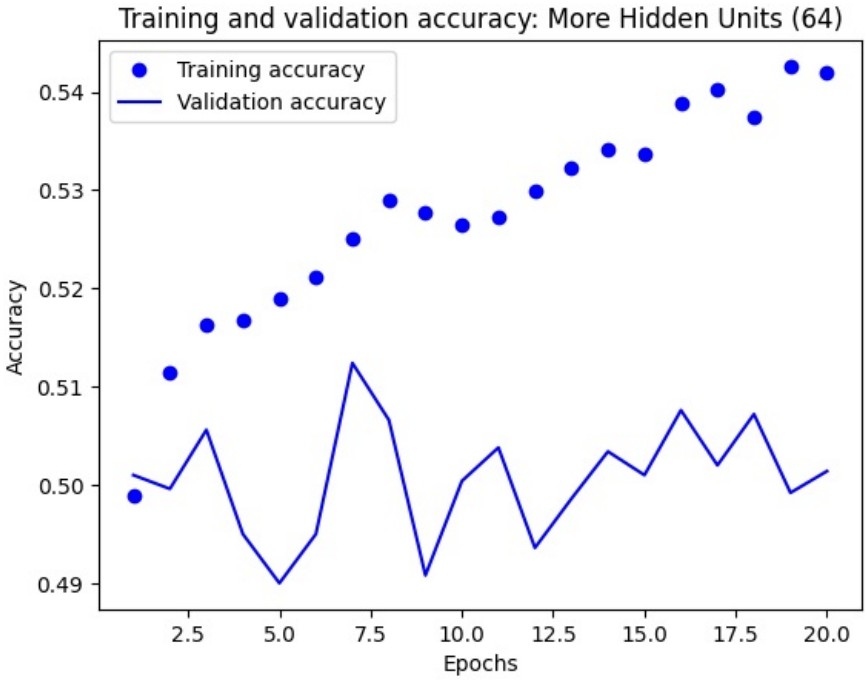
Model with More Hidden Units (64)

- **Overview**: This model used 64 units per layer, increasing the model capacity.
- **Performance**: Training accuracy improved, but validation accuracy showed fluctuations. The model seemed to generalize slightly better than previous models.
- **Conclusion**: Increasing the number of units per layer showed marginal improvements, though more epochs or regularization could enhance generalization.

In [ ]:
```
# Model with 64 hidden units
model_more_hidden_units = models.Sequential([
    layers.Dense(64, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(64, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1, activation="sigmoid")
])
model_more_hidden_units.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["accuracy"])
history_more_hidden_units = model_more_hidden_units.fit(partial_x_train, partial_y_train, epochs=20, batch_size

# Plot the results for 64 hidden units
plot_history(history_more_hidden_units, "More Hidden Units (64)")
```

```
Epoch 1/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.4977 - loss: 0.9042 - val_accuracy: 0.5010 - val_loss: 0.
8591
Epoch 2/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5100 - loss: 0.8479 - val_accuracy: 0.4996 - val_loss: 0.8
426
Epoch 3/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5172 - loss: 0.8366 - val_accuracy: 0.5056 - val_loss: 0.8
369
Epoch 4/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5206 - loss: 0.8271 - val_accuracy: 0.4950 - val_loss: 0.
8292
Epoch 5/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.5243 - loss: 0.8177 - val_accuracy: 0.4900 - val_loss: 0.
8317
Epoch 6/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.5258 - loss: 0.8124 - val_accuracy: 0.4950 - val_loss: 0.
8205
Epoch 7/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5226 - loss: 0.8070 - val_accuracy: 0.5124 - val_loss: 0.
8091
Epoch 8/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.5233 - loss: 0.8017 - val_accuracy: 0.5066 - val_loss: 0.
8208
Epoch 9/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5331 - loss: 0.7942 - val_accuracy: 0.4908 - val_loss: 0.8
072
Epoch 10/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5283 - loss: 0.7913 - val_accuracy: 0.5004 - val_loss: 0.7
949
Epoch 11/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5271 - loss: 0.7866 - val_accuracy: 0.5038 - val_loss: 0.8
044
Epoch 12/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5265 - loss: 0.7843 - val_accuracy: 0.4936 - val_loss: 0.8
028
Epoch 13/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5327 - loss: 0.7777 - val_accuracy: 0.4986 - val_loss: 0.7
848
Epoch 14/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5395 - loss: 0.7728 - val_accuracy: 0.5034 - val_loss: 0.7
807
Epoch 15/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5411 - loss: 0.7697 - val_accuracy: 0.5010 - val_loss: 0.7
793
Epoch 16/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5452 - loss: 0.7653 - val_accuracy: 0.5076 - val_loss: 0.7
735
Epoch 17/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5448 - loss: 0.7628 - val_accuracy: 0.5020 - val_loss: 0.7
720
Epoch 18/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5395 - loss: 0.7598 - val_accuracy: 0.5072 - val_loss: 0.7
690
Epoch 19/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5446 - loss: 0.7577 - val_accuracy: 0.4992 - val_loss: 0.7
713
Epoch 20/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5374 - loss: 0.7566 - val_accuracy: 0.5014 - val_loss: 0.7
686
```



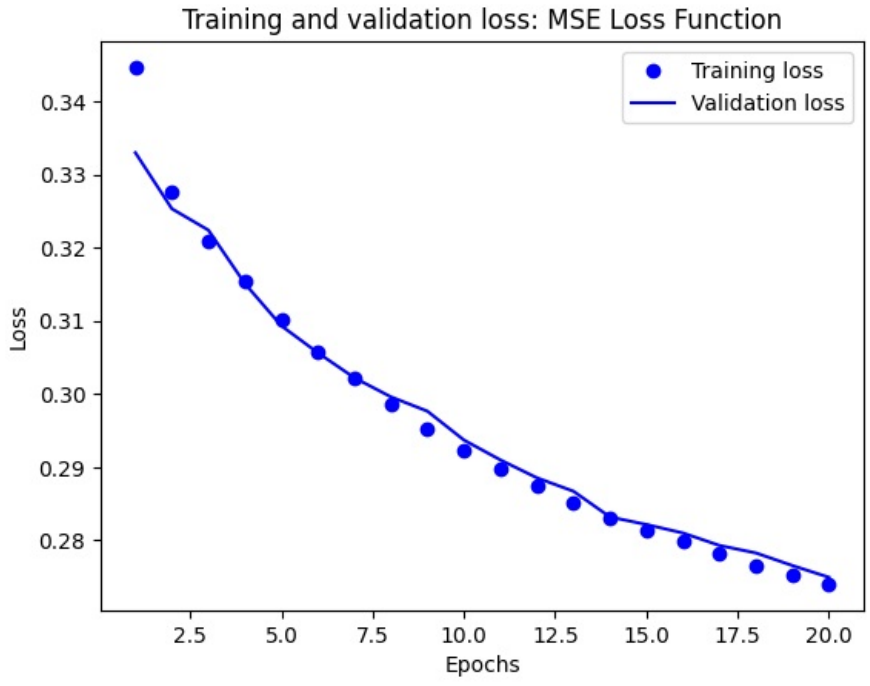


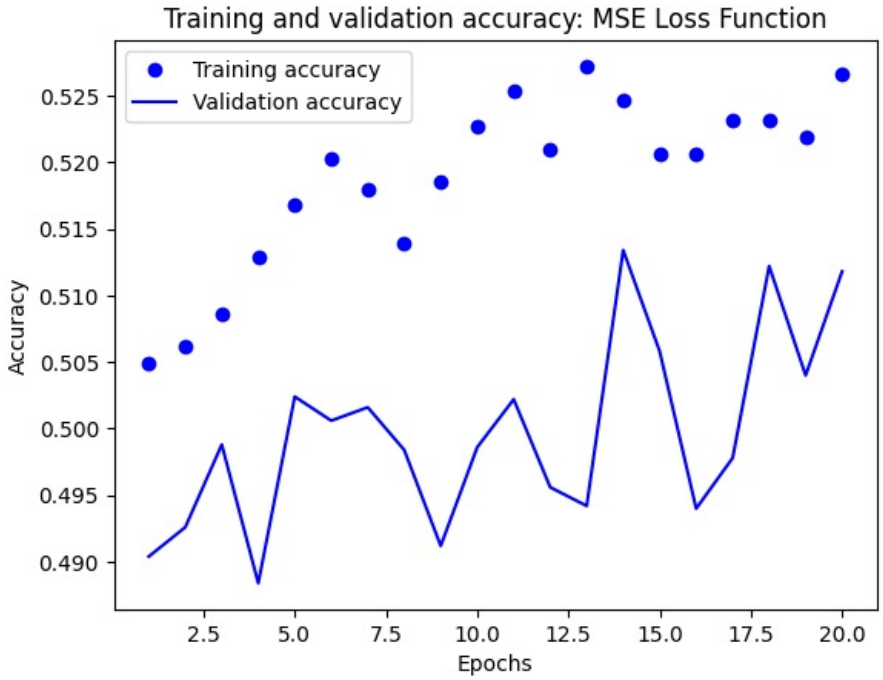
Model with MSE Loss Function

- **Overview**: This model replaced the default `binary_crossentropy` loss with `mse`.
- **Performance**: While training loss steadily decreased, validation accuracy was unstable. The model performed similarly to the `binary_crossentropy` model but did not show substantial improvements.
- **Conclusion**: MSE as a loss function did not significantly improve the results, likely because `binary_crossentropy` is more suitable for binary classification tasks.

In [ ]:
```
# Model with MSE loss function
model_mse_loss = models.Sequential([
    layers.Dense(32, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(32, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1, activation="sigmoid")
])
model_mse_loss.compile(optimizer="rmsprop", loss="mse", metrics=["accuracy"])
history_mse_loss = model_mse_loss.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512, validation_d

# Plot the results for MSE loss
plot_history(history_mse_loss, "MSE Loss Function")
```

```
Epoch 1/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.5037 - loss: 0.3549 - val_accuracy: 0.4904 - val_loss: 0.
3330
Epoch 2/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5037 - loss: 0.3300 - val_accuracy: 0.4926 - val_loss: 0.3
253
Epoch 3/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5125 - loss: 0.3215 - val_accuracy: 0.4988 - val_loss: 0.3
224
Epoch 4/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5179 - loss: 0.3164 - val_accuracy: 0.4884 - val_loss: 0.3
151
Epoch 5/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5190 - loss: 0.3109 - val_accuracy: 0.5024 - val_loss: 0.3
093
Epoch 6/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5195 - loss: 0.3065 - val_accuracy: 0.5006 - val_loss: 0.3
056
Epoch 7/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5165 - loss: 0.3027 - val_accuracy: 0.5016 - val_loss: 0.3
022
Epoch 8/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5141 - loss: 0.2990 - val_accuracy: 0.4984 - val_loss: 0.2
996
Epoch 9/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5159 - loss: 0.2961 - val_accuracy: 0.4912 - val_loss: 0.2
976
Epoch 10/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5235 - loss: 0.2925 - val_accuracy: 0.4986 - val_loss: 0.2
937
Epoch 11/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5258 - loss: 0.2902 - val_accuracy: 0.5022 - val_loss: 0.2
910
Epoch 12/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5226 - loss: 0.2879 - val_accuracy: 0.4956 - val_loss: 0.2
885
Epoch 13/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5328 - loss: 0.2849 - val_accuracy: 0.4942 - val_loss: 0.2
867
Epoch 14/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5244 - loss: 0.2832 - val_accuracy: 0.5134 - val_loss: 0.2
832
Epoch 15/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5260 - loss: 0.2815 - val_accuracy: 0.5058 - val_loss: 0.2
822
Epoch 16/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5229 - loss: 0.2800 - val_accuracy: 0.4940 - val_loss: 0.2
810
Epoch 17/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5212 - loss: 0.2785 - val_accuracy: 0.4978 - val_loss: 0.2
793
Epoch 18/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5208 - loss: 0.2771 - val_accuracy: 0.5122 - val_loss: 0.2
783
Epoch 19/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5274 - loss: 0.2752 - val_accuracy: 0.5040 - val_loss: 0.2
765
Epoch 20/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.5311 - loss: 0.2741 - val_accuracy: 0.5118 - val_loss: 0.
2749
```





Overall Summary

In this experiment, altering the number of hidden layers, units, and loss function led to different outcomes. While increasing model complexity generally improved training performance, validation accuracy remained low, possibly indicating overfitting or insufficient regularization.