```
Start coding or generate with AI.
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import imdb
import tensorflow as tf
import random

# Set seed for reproducibility
def set_seed(seed=42):
    np.random.seed(seed)
    tf.random.set_seed(seed)
    random.seed(seed)

set_seed(42)

# Load and preprocess the IMDB dataset
print("Loading IMDB dataset:")
(train_x, train_y), (test_x, test_y) = imdb.load_data(num_words=10000)

# Function to vectorize sequences
def vectorize(sequences, size=10000):
    results = np.zeros((len(sequences), size))
    for i, seq in enumerate(sequences):
        results[i, seq] = 1.0
    return results

# Vectorize the data
print("Vectorizing data:")
x_train = vectorize(train_x)
x_test = vectorize(test_x)

y_train = np.asarray(train_y).astype("float32")
y_test = np.asarray(test_y).astype("float32")

# Split training data into train and validation sets
x_val = x_train[:10000]
x_train = x_train[10000:]

y_val = y_train[:10000]
y_train = y_train[10000:]

# Function to build model
def build_model(layers_count=2, units=16, activation='relu', dropout=0.0, loss='binary_crossentropy'):
    model = keras.Sequential()
    model.add(layers.Dense(units, activation=activation, input_shape=(10000,)))

    for _ in range(layers_count - 1):
        model.add(layers.Dense(units, activation=activation))
        if dropout > 0.0:
            model.add(layers.Dropout(dropout))

    model.add(layers.Dense(1, activation='sigmoid'))

    model.compile(optimizer='rmsprop',
                  loss=loss,
                  metrics=['accuracy', 'precision', 'recall', 'auc'])
    return model

# Dictionary to store results
results = {
    'Layers': {},
    'Units': {},
    'Loss': {},
    'Activation': {},
    'Dropout': {}
}

# Function to plot metrics
def plot_metrics(history_dict, title, ylabel):
    plt.figure(figsize=(10, 6))
    for label, history in history_dict.items():
```

```
            epochs = range(1, len(history.history['accuracy']) + 1)
            plt.plot(epochs, history.history['accuracy'], label=f'Train {ylabel}')
            plt.plot(epochs, history.history['val_accuracy'], label=f'Val {ylabel}')
        plt.title(title)
        plt.xlabel('Epochs')
        plt.ylabel(ylabel)
        plt.legend()
        plt.show()

    # Experiment 1: Varying Hidden Layers
    layers_options = [1, 2, 3, 4, 5]
    layers_history = {}
    for layers_count in layers_options:
        print(f"\nTraining model with {layers_count} layers...")
        model = build_model(layers_count=layers_count, units=16)
        history = model.fit(x_train, y_train, epochs=20, batch_size=512, validation_data=(x_val, y_val), verbose=1)
        layers_history[layers_count] = history
        val_acc = history.history['val_accuracy'][-1]
        test_acc = model.evaluate(x_test, y_test, verbose=0)[1]
        results['Layers'][layers_count] = {'Val Acc': val_acc, 'Test Acc': test_acc}

    plot_metrics(layers_history, 'Experiment 1: Hidden Layers - Accuracy', 'Accuracy')
    plot_metrics(layers_history, 'Experiment 1: Hidden Layers - Loss', 'Loss')

    # Experiment 2: Varying Hidden Units
    units_options = [32, 64, 128, 256]
    units_history = {}
    for units in units_options:
        print(f"\nTraining model with {units} units:")
        model = build_model(layers_count=2, units=units)
        history = model.fit(x_train, y_train, epochs=20, batch_size=512, validation_data=(x_val, y_val), verbose=1)
        units_history[units] = history
        val_acc = history.history['val_accuracy'][-1]
        test_acc = model.evaluate(x_test, y_test, verbose=0)[1]
        results['Units'][units] = {'Val Acc': val_acc, 'Test Acc': test_acc}

    plot_metrics(units_history, 'Experiment 2: Hidden Units - Accuracy', 'Accuracy')
    plot_metrics(units_history, 'Experiment 2: Hidden Units - Loss', 'Loss')

    # Experiment 3: Different Loss Functions
    loss_history = {}
    print("\nTraining model with MSE loss:")
    model_mse = build_model(layers_count=2, units=16, loss='mse')
    history_mse = model_mse.fit(x_train, y_train, epochs=20, batch_size=512, validation_data=(x_val, y_val), verbose=1)
    loss_history['MSE'] = history_mse
    val_acc_mse = history_mse.history['val_accuracy'][-1]
    test_acc_mse = model_mse.evaluate(x_test, y_test, verbose=0)[1]
    results['Loss']['MSE'] = {'Val Acc': val_acc_mse, 'Test Acc': test_acc_mse}

    plot_metrics(loss_history, 'Experiment 3: Loss Function - Accuracy', 'Accuracy')
    plot_metrics(loss_history, 'Experiment 3: Loss Function - Loss', 'Loss')

    # Experiment 4: Different Activation Functions
    activation_history = {}
    print("\nTraining model with tanh activation:")
    model_tanh = build_model(layers_count=2, units=16, activation='tanh')
    history_tanh = model_tanh.fit(x_train, y_train, epochs=20, batch_size=512, validation_data=(x_val, y_val), verbose=1)
    activation_history['tanh'] = history_tanh
    val_acc_tanh = history_tanh.history['val_accuracy'][-1]
    test_acc_tanh = model_tanh.evaluate(x_test, y_test, verbose=0)[1]
    results['Activation']['tanh'] = {'Val Acc': val_acc_tanh, 'Test Acc': test_acc_tanh}

    plot_metrics(activation_history, 'Experiment 4: Activation - Accuracy', 'Accuracy')
    plot_metrics(activation_history, 'Experiment 4: Activation - Loss', 'Loss')

    # Experiment 5: Dropout Regularization
    dropout_history = {}
    dropout_options = [0.3, 0.5, 0.7]
    for dropout in dropout_options:
        print(f"\nTraining model with {dropout} dropout:")
        model = build_model(layers_count=2, units=16, dropout=dropout)
        history = model.fit(x_train, y_train, epochs=20, batch_size=512, validation_data=(x_val, y_val), verbose=1)
        dropout_history[dropout] = history
        val_acc_dropout = history.history['val_accuracy'][-1]
        test_acc_dropout = model.evaluate(x_test, y_test, verbose=0)[1]
        results['Dropout'][dropout] = {'Val Acc': val_acc_dropout, 'Test Acc': test_acc_dropout}

    plot_metrics(dropout_history, 'Experiment 5: Dropout - Accuracy', 'Accuracy')
```

```
    plot_metrics(dropout_history, 'Experiment 5: Dropout - Loss', 'Loss')

    # Summary of results
    print("\nSummary of Results: ")
    for exp, outcome in results.items():
        print(f"\n{exp}:")
        df = pd.DataFrame(outcome).T
        df.columns = ["Val Acc", "Test Acc"]
        print(df)
```

```
Loading IMDB dataset:
Vectorizing data:

Training model with 1 layers...
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 109ms/step - accuracy: 0.7302 - auc: 0.8075 - loss: 0.5782 - precision: 0.7289 - recall: 0.7306 - val_a
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 38ms/step - accuracy: 0.8906 - auc: 0.9549 - loss: 0.3561 - precision: 0.8893 - recall: 0.8909 - val_ac
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9144 - auc: 0.9700 - loss: 0.2788 - precision: 0.9117 - recall: 0.9167 - val_ac
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9279 - auc: 0.9778 - loss: 0.2327 - precision: 0.9270 - recall: 0.9281 - val_ac
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9376 - auc: 0.9827 - loss: 0.2016 - precision: 0.9376 - recall: 0.9368 - val_ac
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9453 - auc: 0.9861 - loss: 0.1786 - precision: 0.9459 - recall: 0.9440 - val_ac
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9516 - auc: 0.9888 - loss: 0.1603 - precision: 0.9528 - recall: 0.9497 - val_ac
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 68ms/step - accuracy: 0.9572 - auc: 0.9910 - loss: 0.1449 - precision: 0.9583 - recall: 0.9556 - val_ac
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.9621 - auc: 0.9927 - loss: 0.1316 - precision: 0.9631 - recall: 0.9605 - val_ac
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9663 - auc: 0.9942 - loss: 0.1199 - precision: 0.9674 - recall: 0.9647 - val_ac
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9707 - auc: 0.9953 - loss: 0.1093 - precision: 0.9721 - recall: 0.9688 - val_ac
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9738 - auc: 0.9963 - loss: 0.1000 - precision: 0.9750 - recall: 0.9722 - val_ac
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9767 - auc: 0.9970 - loss: 0.0916 - precision: 0.9770 - recall: 0.9760 - val_ac
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9796 - auc: 0.9976 - loss: 0.0839 - precision: 0.9807 - recall: 0.9782 - val_ac
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9824 - auc: 0.9981 - loss: 0.0771 - precision: 0.9834 - recall: 0.9811 - val_ac
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9840 - auc: 0.9985 - loss: 0.0708 - precision: 0.9849 - recall: 0.9829 - val_ac
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9868 - auc: 0.9988 - loss: 0.0659 - precision: 0.9869 - recall: 0.9866 - val_ac
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9885 - auc: 0.9991 - loss: 0.0607 - precision: 0.9882 - recall: 0.9888 - val_ac
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.9904 - auc: 0.9993 - loss: 0.0553 - precision: 0.9901 - recall: 0.9907 - val_ac
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.9918 - auc: 0.9994 - loss: 0.0511 - precision: 0.9914 - recall: 0.9922 - val_ac

Training model with 2 layers...
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 103ms/step - accuracy: 0.7028 - auc: 0.7760 - loss: 0.5959 - precision: 0.7472 - recall: 0.5908 - val_a
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 38ms/step - accuracy: 0.8849 - auc: 0.9508 - loss: 0.3399 - precision: 0.8839 - recall: 0.8848 - val_ac
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9181 - auc: 0.9726 - loss: 0.2485 - precision: 0.9153 - recall: 0.9205 - val_ac
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9374 - auc: 0.9823 - loss: 0.1960 - precision: 0.9370 - recall: 0.9372 - val_ac
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9449 - auc: 0.9865 - loss: 0.1666 - precision: 0.9445 - recall: 0.9446 - val_ac
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9524 - auc: 0.9902 - loss: 0.1418 - precision: 0.9519 - recall: 0.9524 - val_ac
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9665 - auc: 0.9940 - loss: 0.1154 - precision: 0.9668 - recall: 0.9658 - val_ac
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9738 - auc: 0.9960 - loss: 0.0972 - precision: 0.9752 - recall: 0.9720 - val_ac
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.9772 - auc: 0.9970 - loss: 0.0844 - precision: 0.9774 - recall: 0.9768 - val_ac
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.9816 - auc: 0.9981 - loss: 0.0714 - precision: 0.9820 - recall: 0.9810 - val_ac
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.9833 - auc: 0.9985 - loss: 0.0643 - precision: 0.9845 - recall: 0.9819 - val_ac
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9873 - auc: 0.9990 - loss: 0.0547 - precision: 0.9869 - recall: 0.9875 - val_ac
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9895 - auc: 0.9993 - loss: 0.0467 - precision: 0.9900 - recall: 0.9888 - val_ac
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9933 - auc: 0.9997 - loss: 0.0362 - precision: 0.9937 - recall: 0.9930 - val_ac
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9952 - auc: 0.9998 - loss: 0.0300 - precision: 0.9945 - recall: 0.9960 - val_ac
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.9969 - auc: 0.9999 - loss: 0.0260 - precision: 0.9971 - recall: 0.9967 - val_ac
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.9968 - auc: 0.9999 - loss: 0.0233 - precision: 0.9964 - recall: 0.9970 - val_ac
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9978 - auc: 1.0000 - loss: 0.0204 - precision: 0.9971 - recall: 0.9985 - val_ac
```