1. **Using two hidden layers. Try using one or three hidden layers, and see how doing so affects validation and test accuracy.**

This code builds a rudimentary sentiment analysis model on the IMDb dataset, which comprises movie reviews classified as positive or negative. The dataset is part of Keras, a deep learning package, and the model was created using the Sequential API. Here's an overview of the key steps:

```
from keras.datasets import imdb
from keras import models, layers
from keras.preprocessing import sequence

# Load the data
max_features = 10000  # Number of words to consider as features
maxlen = 500  # Cut texts after this number of words (among top max_features most common words)
batch_size = 32

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Pad sequences to ensure each input has the same length
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

# Model with one hidden layer
model = models.Sequential()
model.add(layers.Embedding(max_features, 32, input_length=maxlen))
model.add(layers.Flatten())
model.add(layers.Dense(32, activation='relu'))  # One hidden layer
model.add(layers.Dense(1, activation='sigmoid'))  # Output layer

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

# Train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=batch_size, validation_split=0.2)

# Evaluate on test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy with one hidden layer: {test_acc}')
```

The IMDb dataset is loaded with imdb.load_data() and num_words=max_features, which restricts the vocabulary to the top 10,000 most common words.
Each review in x_train and x_test is a series of numbers that represent words.
The reviews are then padded with sequence.pad_sequences() to ensure they are of the same length. This is required because neural networks require constant input dimensions.

Building the model:

A Sequential model is built, allowing layers to be layered consecutively.
The first layer is an Embedding layer, which converts the word indices (integers) to dense 32-dimensional vectors. This layer assists the model in learning word representations during training.
The Flatten layer turns the embedding's 2D output into a 1D vector that may be sent to the Dense layers.

The model consists of one Dense layer (hidden layer) with 32 units and a ReLU activation function. This layer aids in the discovery of patterns and linkages within the review data.
The last layer is a Dense output layer with one unit and a sigmoid activation function, which returns a probability value (between 0 and 1) indicating whether the review is positive or negative.

The model uses RMSprop as an optimizer to update weights and decrease training errors.
Binary cross-entropy as the loss function is appropriate for binary classification issues (positive/negative evaluation).
Accuracy is the evaluation statistic used to track how effectively the model performs.

Following training, the model is assessed on the test dataset (x_test, y_test) to determine its accuracy in predicting the sentiment of unseen reviews.
The final test accuracy is shown, indicating how well the model generalizes to new data.
To summarize, this code generates a basic neural network model with one hidden layer that classifies movie reviews as favorable or bad based on their text content.

Accuracy:

```
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 16s 22ms/step - accuracy: 0.6578 - loss: 0.5779 - val_accuracy: 0.8666 - val_loss: 0.3175
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.9227 - loss: 0.1994 - val_accuracy: 0.8488 - val_loss: 0.3638
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 18ms/step - accuracy: 0.9794 - loss: 0.0665 - val_accuracy: 0.8486 - val_loss: 0.4573
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 19ms/step - accuracy: 0.9967 - loss: 0.0152 - val_accuracy: 0.8376 - val_loss: 0.6558
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 18ms/step - accuracy: 0.9996 - loss: 0.0025 - val_accuracy: 0.8408 - val_loss: 0.7791
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - accuracy: 0.9999 - loss: 4.2081e-04 - val_accuracy: 0.8402 - val_loss: 0.8605
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - accuracy: 0.9999 - loss: 3.2229e-04 - val_accuracy: 0.8414 - val_loss: 0.9179
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - accuracy: 0.9998 - loss: 6.0706e-04 - val_accuracy: 0.8446 - val_loss: 0.9210
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - accuracy: 1.0000 - loss: 3.7749e-05 - val_accuracy: 0.8436 - val_loss: 0.9419
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - accuracy: 1.0000 - loss: 2.9471e-05 - val_accuracy: 0.8428 - val_loss: 0.9617
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8404 - loss: 0.9130
Test accuracy with one hidden layer: 0.8410400152206421
```

2. Try using layers with more hidden units or fewer hidden units: 32 units, 64 units

```python
# Example: Using 64 units in one hidden layer
model_units = models.Sequential()
model_units.add(layers.Embedding(max_features, 32, input_length=maxlen))
model_units.add(layers.Flatten())
model_units.add(layers.Dense(64, activation='relu'))  # 64 hidden units
model_units.add(layers.Dense(1, activation='sigmoid'))

# Compile and train as usual
model_units.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
history_units = model_units.fit(x_train, y_train, epochs=10, batch_size=batch_size, validation_split=0.2)
test_loss_units, test_acc_units = model_units.evaluate(x_test, y_test)
print(f'Test accuracy with 64 hidden units: {test_acc_units}')
```

Using the IMDb dataset, the algorithm creates a neural network model with 64 hidden units that classifies movie reviews as good or negative.

Key steps:
Model Construction:

The first layer is an embedding layer, which converts word indices into 32-dimensional dense vectors.
A Flatten layer is introduced to reshape the embedding output in preparation for the following dense layer.
The hidden layer has 64 units and uses the ReLU activation function.
To forecast the emotion, the output layer uses a Dense layer with one unit and sigmoid activation (positive/negative).
Model Compilation:

The model is constructed using the RMSprop optimizer, with binary cross-entropy as the loss function and accuracy as the evaluation measure.
Training the model:

The model is trained using the IMDb.

Training data is collected for 10 epochs with a batch size of 32, with 20% of the data used for validation.
Model Evaluation:

Following training, the model is assessed on test data, and the final test accuracy is reported.
In summary, this method evaluates the performance of a model with 64 hidden units by training it on movie review data and comparing its accuracy to unseen test data.

Accuracy:

```
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 18s 26ms/step - accuracy: 0.6489 - loss: 0.5815 - val_accuracy: 0.8564 - val_loss: 0.3259
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 28ms/step - accuracy: 0.9292 - loss: 0.1848 - val_accuracy: 0.8566 - val_loss: 0.3490
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 15s 23ms/step - accuracy: 0.9863 - loss: 0.0502 - val_accuracy: 0.8468 - val_loss: 0.5097
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 25ms/step - accuracy: 0.9972 - loss: 0.0110 - val_accuracy: 0.8380 - val_loss: 0.6652
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 24ms/step - accuracy: 0.9994 - loss: 0.0025 - val_accuracy: 0.8384 - val_loss: 0.8079
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 15s 24ms/step - accuracy: 0.9999 - loss: 7.9073e-04 - val_accuracy: 0.8386 - val_loss: 0.8844
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 26ms/step - accuracy: 1.0000 - loss: 1.3351e-04 - val_accuracy: 0.8374 - val_loss: 0.9264
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 25ms/step - accuracy: 1.0000 - loss: 3.7540e-05 - val_accuracy: 0.8352 - val_loss: 0.9540
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 25ms/step - accuracy: 1.0000 - loss: 2.0144e-05 - val_accuracy: 0.8352 - val_loss: 0.9708
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 23ms/step - accuracy: 0.9998 - loss: 0.0010 - val_accuracy: 0.8416 - val_loss: 0.9552
782/782 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.8392 - loss: 0.9449
Test accuracy with 64 hidden units: 0.8386800289154053
```

3. Using the mse loss function instead of binary_crossentropy.

```
# Model with mse loss
model_mse = models.Sequential()
model_mse.add(layers.Embedding(max_features, 32, input_length=maxlen))
model_mse.add(layers.Flatten())
model_mse.add(layers.Dense(32, activation='relu'))
model_mse.add(layers.Dense(1, activation='sigmoid'))

# Compile with mse loss function
model_mse.compile(optimizer='rmsprop', loss='mse', metrics=['accuracy'])

# Train and evaluate
history_mse = model_mse.fit(x_train, y_train, epochs=10, batch_size=batch_size, validation_split=0.2)
test_loss_mse, test_acc_mse = model_mse.evaluate(x_test, y_test)
print(f'Test accuracy with mse loss: {test_acc_mse}')
```

The code builds a neural network model identical to the previous ones, but instead of utilizing binary cross-entropy as the loss function, it employs mean squared error (MSE). Here's the summary:

Key steps:
Model Definition:

The model includes an Embedding layer, which converts word indices into 32-dimensional dense vectors.
A Flatten layer reshapes the embedded output.
The hidden layer is a 32-unit dense layer activated using ReLU.
The output layer has one unit with sigmoid activation to determine sentiment (positive or negative).
Model Compilation:

The model is constructed with the RMSprop optimizer and mean squared error (MSE) as the loss function.
The accuracy measure is used to monitor the model's performance throughout both training and assessment.
Training the model:

The model is trained using IMDb training data for 10 epochs with a batch size of 32, with 20% of the data used for validation.
Evaluation:

Following training, the model is assessed using test data, and the test accuracy is reported.
The MSE loss function is often used for regression, however here it is applied to a binary classification problem. While it can be useful, MSE is less suited than binary cross-entropy since it fails to account for probability outputs and classification-specific mistakes.
In conclusion, this code examines a model that uses MSE loss to predict movie review emotions and assesses its effectiveness by evaluating accuracy.

Accuracy:

```
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 12s 18ms/step - accuracy: 0.6336 - loss: 0.2130 - val_accuracy: 0.8494 - val_loss: 0.1066
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - accuracy: 0.9088 - loss: 0.0709 - val_accuracy: 0.8388 - val_loss: 0.1150
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 16ms/step - accuracy: 0.9677 - loss: 0.0283 - val_accuracy: 0.8440 - val_loss: 0.1198
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - accuracy: 0.9894 - loss: 0.0108 - val_accuracy: 0.8412 - val_loss: 0.1240
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - accuracy: 0.9960 - loss: 0.0044 - val_accuracy: 0.8354 - val_loss: 0.1314
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - accuracy: 0.9965 - loss: 0.0035 - val_accuracy: 0.8386 - val_loss: 0.1325
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - accuracy: 0.9976 - loss: 0.0024 - val_accuracy: 0.8360 - val_loss: 0.1344
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 16ms/step - accuracy: 0.9976 - loss: 0.0023 - val_accuracy: 0.8262 - val_loss: 0.1390
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - accuracy: 0.9982 - loss: 0.0019 - val_accuracy: 0.8286 - val_loss: 0.1376
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 19s 15ms/step - accuracy: 0.9989 - loss: 0.0012 - val_accuracy: 0.8302 - val_loss: 0.1376
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8262 - loss: 0.1409
Test accuracy with mse loss: 0.828279972076416
```

4. Using the tanh activation (an activation that was popular in the early days of neural networks) instead of relu.

```python
# Model with tanh activation
model_tanh = models.Sequential()
model_tanh.add(layers.Embedding(max_features, 32, input_length=maxlen))
model_tanh.add(layers.Flatten())
model_tanh.add(layers.Dense(32, activation='tanh'))  # tanh activation
model_tanh.add(layers.Dense(1, activation='sigmoid'))

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

# Train and evaluate
history_tanh = model_tanh.fit(x_train, y_train, epochs=10, batch_size=batch_size, validation_split=0.2)
test_loss_tanh, test_acc_tanh = model_tanh.evaluate(x_test, y_test)
print(f'Test accuracy with tanh activation: {test_acc_tanh}')
```

This code creates a neural network model for sentiment analysis that use the tanh activation function in the hidden layer rather than the more popular ReLU activation. Here is a breakdown:

Key Steps:
Model definition:

The model begins with an embedding layer that converts word indices into 32-dimensional dense vectors.
A Flatten layer reshapes the embedding output to be used in the Dense layer.
The hidden layer has 32 units and uses the tanh activation function. The tanh function returns values between -1 and 1, allowing it to capture a wider range of inputs than ReLU (which only returns non-negative values).
The output layer is a Dense layer with one unit and sigmoid activation, producing a probability for binary classification (positive or Negative emotion).
Model Compilation:

As this is a binary classification problem, the RMSprop optimizer is used to generate the model, and the loss function is binary cross-entropy.

Accuracy is used as an assessment metric.
Training the model:

The model is trained on the IMDb dataset for 10 epochs with a batch size of 32, using 20% of the training data for validation.
Evaluation:

Following training, the model's performance on the test set is assessed, and the test accuracy is reported.
Key distinction:
The most significant modification is the use of the tanh activation function in the hidden layer. Tanh, unlike ReLU, may produce negative values, which may allow the model to learn more sophisticated representations.
To summarize, this code evaluates a model using tanh activation. The hidden layer is trained on sentiment data from movie reviews and its accuracy is evaluated on unseen test data.

Accuracy:

```
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 18ms/step - accuracy: 0.6984 - loss: 0.5521 - val_accuracy: 0.8626 - val_loss: 0.3213
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - accuracy: 0.9354 - loss: 0.1704 - val_accuracy: 0.8492 - val_loss: 0.3918
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - accuracy: 0.9904 - loss: 0.0380 - val_accuracy: 0.8462 - val_loss: 0.5672
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 20ms/step - accuracy: 0.9991 - loss: 0.0039 - val_accuracy: 0.8370 - val_loss: 0.8003
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 19s 17ms/step - accuracy: 1.0000 - loss: 5.0297e-04 - val_accuracy: 0.8390 - val_loss: 0.8884
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 22s 20ms/step - accuracy: 1.0000 - loss: 1.7158e-04 - val_accuracy: 0.8336 - val_loss: 0.9381
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 19ms/step - accuracy: 0.9997 - loss: 7.2065e-04 - val_accuracy: 0.8300 - val_loss: 0.9631
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 19ms/step - accuracy: 1.0000 - loss: 4.8476e-05 - val_accuracy: 0.8372 - val_loss: 0.9650
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 20ms/step - accuracy: 1.0000 - loss: 2.7985e-05 - val_accuracy: 0.8392 - val_loss: 0.9867
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 1.0000 - loss: 1.8938e-05 - val_accuracy: 0.8386 - val_loss: 0.9914
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8330 - loss: 0.9355
Test accuracy with tanh activation: 0.8335199952125549
```

5. Model with Dropout:

```
# Model with Dropout
model_dropout = models.Sequential()
model_dropout.add(layers.Embedding(max_features, 32, input_length=maxlen))
model_dropout.add(layers.Flatten())
model_dropout.add(layers.Dropout(0.5))  # Dropout layer
model_dropout.add(layers.Dense(32, activation='relu'))
model_dropout.add(layers.Dense(1, activation='sigmoid'))

# Compile and train
model_dropout.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
history_dropout = model_dropout.fit(x_train, y_train, epochs=10, batch_size=batch_size, validation_split=0.2)
test_loss_dropout, test_acc_dropout = model_dropout.evaluate(x_test, y_test)
print(f'Test accuracy with dropout: {test_acc_dropout}')
```

This code creates a neural network model with a Dropout layer to assist minimize overfitting during training, which can enhance generalization to new data.

Key steps:
Model Definition:

The first layer is an embedding layer, which transforms word indices into 32-dimensional dense vectors.
A flatten layer reshapes the embedding output for the following layer.
A dropout layer with a dropout rate of 0.5 is added, implying that 50% of the neurons are randomly discarded (set to zero) during each training cycle. This drives the model to learn more robust patterns while also preventing overfitting by limiting the reliance on any single neurons.
A dense layer with 32 units and ReLU activation follows, which aids in the discovery of complicated links in the data.
The output layer is: A dense layer with one unit and sigmoid activation produces a probability for binary classification (positive or negative sentiment).
Model Compilation:

The model is built using the RMSprop optimizer and the binary cross-entropy loss function, which is appropriate for binary classification.
Accuracy is used to measure model performance.
Training the model:

The model is trained on the IMDb dataset for 10 epochs with a batch size of 32, using 20% of the training data for validation.
Evaluation:

Following training, the model is assessed using test data, and the test accuracy is reported.
Key distinction:
The Dropout layer avoids overfitting by randomly deactivating 50% of neurons during training, allowing the model to generalize better to fresh input.

Accuracy:

```
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 17s 24ms/step - accuracy: 0.6197 - loss: 0.6132 - val_accuracy: 0.8524 - val_loss: 0.3440
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 15s 23ms/step - accuracy: 0.8923 - loss: 0.2637 - val_accuracy: 0.8788 - val_loss: 0.3052
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 24ms/step - accuracy: 0.9395 - loss: 0.1615 - val_accuracy: 0.8718 - val_loss: 0.3315
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 24ms/step - accuracy: 0.9611 - loss: 0.1055 - val_accuracy: 0.8644 - val_loss: 0.3895
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 19s 23ms/step - accuracy: 0.9708 - loss: 0.0791 - val_accuracy: 0.8684 - val_loss: 0.4275
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 23ms/step - accuracy: 0.9782 - loss: 0.0604 - val_accuracy: 0.8688 - val_loss: 0.4798
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 24ms/step - accuracy: 0.9814 - loss: 0.0522 - val_accuracy: 0.8728 - val_loss: 0.4566
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 23ms/step - accuracy: 0.9825 - loss: 0.0456 - val_accuracy: 0.8704 - val_loss: 0.4979
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 23ms/step - accuracy: 0.9855 - loss: 0.0373 - val_accuracy: 0.8574 - val_loss: 0.5828
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 23ms/step - accuracy: 0.9867 - loss: 0.0362 - val_accuracy: 0.8680 - val_loss: 0.5521
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8649 - loss: 0.5331
Test accuracy with dropout: 0.8656799793243408
```

5.2: Model with L2 regularisation:

```
from tensorflow import keras
from tensorflow.keras import layers, models, regularizers  # Import regularizers

# Model with L2 regularization
model_l2 = models.Sequential()
model_l2.add(layers.Embedding(max_features, 32, input_length=maxlen))
model_l2.add(layers.Flatten())
model_l2.add(layers.Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.001)))  # L2 regularization
model_l2.add(layers.Dense(1, activation='sigmoid'))

# Compile and train
model_l2.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
history_l2 = model_l2.fit(x_train, y_train, epochs=10, batch_size=batch_size, validation_split=0.2)
test_loss_l2, test_acc_l2 = model_l2.evaluate(x_test, y_test)
print(f'Test accuracy with L2 regularization: {test_acc_l2}')
```

This code creates a neural network model using L2 regularization to minimize overfitting by penalizing big weights during training.

Key steps:
Model Definition:

The model starts with an Embedding layer, which transforms word indices into 32-dimensional dense vectors.
A Flatten layer reshapes the embedding output of the Dense layer.
The main change occurs in the Dense layer with 32 units and ReLU activation, where L2 regularization (regularizers.l2(0.001)) is used. This penalizes big weight values by adding a modest penalty proportionate to the square of the weights, hence reducing overfitting.
The last layer is a Dense layer with one unit and sigmoid activation, which predicts the likelihood of positive or negative emotion.
Model Compilation:

The model is compiled using the RMSprop optimizer and binary cross-entropy. as a loss function. Accuracy is used to monitor performance.
Training the model:

The model is trained for 10 epochs with a batch size of 32, using 20% of the training data for validation.
Evaluation:

The model is assessed on the test dataset, and the test accuracy is reported.
L2 regularization, also known as weight decay, prevents overfitting by penalizing the loss function depending on the sum of squared weights. This discourages using high weight values, encouraging the model to learn simpler patterns.
In conclusion, our model employs L2 regularization to increase generalization, trains on the IMDb sentiment dataset, and assesses its performance on the test set.

Accuracy:

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
625/625 ━━━━━━━━━━━━━━━━━━━━ 15s 22ms/step - accuracy: 0.6686 - loss: 0.6269 - val_accuracy: 0.8596 - val_loss: 0.3951
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 19s 19ms/step - accuracy: 0.9071 - loss: 0.3047 - val_accuracy: 0.8680 - val_loss: 0.3771
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 20ms/step - accuracy: 0.9604 - loss: 0.2014 - val_accuracy: 0.8558 - val_loss: 0.4299
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 12s 19ms/step - accuracy: 0.9819 - loss: 0.1399 - val_accuracy: 0.8522 - val_loss: 0.4800
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 19ms/step - accuracy: 0.9918 - loss: 0.0968 - val_accuracy: 0.8444 - val_loss: 0.5362
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 12s 19ms/step - accuracy: 0.9950 - loss: 0.0770 - val_accuracy: 0.8494 - val_loss: 0.5545
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 19ms/step - accuracy: 0.9957 - loss: 0.0685 - val_accuracy: 0.8450 - val_loss: 0.5915
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 19ms/step - accuracy: 0.9960 - loss: 0.0657 - val_accuracy: 0.8430 - val_loss: 0.6151
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 20s 19ms/step - accuracy: 0.9965 - loss: 0.0593 - val_accuracy: 0.8422 - val_loss: 0.6551
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 19ms/step - accuracy: 0.9960 - loss: 0.0625 - val_accuracy: 0.8456 - val_loss: 0.6822
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8420 - loss: 0.6626
Test accuracy with L2 regularization: 0.8417199850082397
```