```
import subprocess

def install_package(package):
    try:
        subprocess.check_call(["pip", "install", package, "--upgrade", "
        print(f"Successfully installed/upgraded {package}")
    except subprocess.CalledProcessError as e:
        print(f"Error installing/upgrading {package}: {e}")
        print(e.stdout.decode() if e.stdout else "")
        print(e.stderr.decode() if e.stderr else "")

install_package("keras")
install_package("keras-hub==0.21.1") # Fix specific version for keras-nl
```

```
Successfully installed/upgraded keras
Successfully installed/upgraded keras-hub==0.21.1
```

```
import os
os.environ["KERAS_BACKEND"] = "jax"
```

```
# @title
import os
from IPython.core.magic import register_cell_magic

@register_cell_magic
def backend(line, cell):
    current, required = os.environ.get("KERAS_BACKEND", ""), line.split(
    if current == required:
        get_ipython().run_cell(cell)
    else:
        print(
            f"This cell requires the {required} backend. To run it, chan
            f"\"{required}\" at the top of the notebook, restart the run
        )
```

Text classification

A brief history of natural language processing

Preparing text data

↳ 32 cells hidden

Sets vs. sequences

Loading the IMDb classification dataset

```
import os, pathlib, shutil, random

zip_path = keras.utils.get_file(
    origin="https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar
    fname="imdb",
    extract=True,
)

imdb_extract_dir = pathlib.Path(zip_path) / "aclImdb"
```

```
for path in imdb_extract_dir.glob("*/*"):
    if path.is_dir():
        print(path)
```

```
/root/.keras/datasets/imdb/aclImdb/train/unsup
/root/.keras/datasets/imdb/aclImdb/train/pos
/root/.keras/datasets/imdb/aclImdb/train/neg
/root/.keras/datasets/imdb/aclImdb/test/pos
/root/.keras/datasets/imdb/aclImdb/test/neg
```

```
print(open(imdb_extract_dir / "train" / "pos" / "4077_10.txt", "r").read
```

```
I first saw this back in the early 90s on UK TV, i did like it then but i
```

```
train_dir = pathlib.Path("imdb_train")
test_dir = pathlib.Path("imdb_test")
val_dir = pathlib.Path("imdb_val")

# Remove directories if they already exist to prevent FileExistsError
if test_dir.exists():
    shutil.rmtree(test_dir)
if train_dir.exists():
    shutil.rmtree(train_dir)
if val_dir.exists():
    shutil.rmtree(val_dir)

shutil.copytree(imdb_extract_dir / "test", test_dir)

val_percentage = 0.4  # Validate on 10,000 samples.
for category in ("neg", "pos"):
    src_dir = imdb_extract_dir / "train" / category
    src_files = os.listdir(src_dir)
    random.Random(1337).shuffle(src_files)
    num_val_samples = int(len(src_files) * val_percentage)

    os.makedirs(val_dir / category)
    for file in src_files[:num_val_samples]:
        shutil.copy(src_dir / file, val_dir / category / file)
    os.makedirs(train_dir / category)
    for file in src_files[num_val_samples:]:
        shutil.copy(src_dir / file, train_dir / category / file)
```

```
from keras.utils import text_dataset_from_directory

batch_size = 32
train_ds = text_dataset_from_directory(train_dir, batch_size=batch_size)
val_ds = text_dataset_from_directory(val_dir, batch_size=batch_size)
test_ds = text_dataset_from_directory(test_dir, batch_size=batch_size)
```

Set models

Training a bag-of-words model

↳ 8 cells hidden

Training a bigram model

↳ 5 cells hidden

Sequence models

```
max_length = 150 # Cutoff reviews after 150 words.
max_tokens = 10_000 # Restrict to top 10,000 words
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    split="whitespace",
    output_mode="int",
    output_sequence_length=max_length,
)
text_vectorization.adapt(train_ds_no_labels)

sequence_train_ds = train_ds.map(
    lambda x, y: (text_vectorization(x), y), num_parallel_calls=8
)
sequence_val_ds = val_ds.map(
    lambda x, y: (text_vectorization(x), y), num_parallel_calls=8
)
sequence_test_ds = test_ds.map(
    lambda x, y: (text_vectorization(x), y), num_parallel_calls=8
)
```

```
x, y = next(sequence_test_ds.as_numpy_iterator())
x.shape
```

```
x
```

Training a recurrent model

```
from keras import ops

class OneHotEncoding(keras.Layer):
    def __init__(self, depth, **kwargs):
        super().__init__(**kwargs)
        self.depth = depth

    def call(self, inputs):
        flat_inputs = ops.reshape(ops.cast(inputs, "int"), [-1])
        one_hot_vectors = ops.eye(self.depth)
        outputs = ops.take(one_hot_vectors, flat_inputs, axis=0)
        return ops.reshape(outputs, ops.shape(inputs) + (self.depth,))

one_hot_encoding = OneHotEncoding(max_tokens)
```

```
x, y = next(sequence_train_ds.as_numpy_iterator())
one_hot_encoding(x).shape
```

```
hidden_dim = 64
inputs = keras.Input(shape=(max_length,), dtype="int32")
x = one_hot_encoding(inputs)
x = layers.Bidirectional(layers.LSTM(hidden_dim))(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs, name="lstm_with_one_hot")
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
```

```
model.summary(line_length=80)
```

```
model.fit(
    sequence_train_ds,
    validation_data=sequence_val_ds,
    epochs=10,
    callbacks=[early_stopping],
)
```

```
test_loss, test_acc = model.evaluate(sequence_test_ds)
test_acc
```

Understanding word embeddings

Using a word embedding

```
hidden_dim = 64
inputs = keras.Input(shape=(max_length,), dtype="int32")
x = keras.layers.Embedding(
    input_dim=max_tokens,
    output_dim=hidden_dim,
    mask_zero=True,
)(inputs)
x = keras.layers.Bidirectional(keras.layers.LSTM(hidden_dim))(x)
x = keras.layers.Dropout(0.5)(x)
outputs = keras.layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs, name="lstm_with_embedding")
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
```

```
model.summary(line_length=80)
```

```
model.fit(
    sequence_train_ds,
    validation_data=sequence_val_ds,
    epochs=10,
    callbacks=[early_stopping],
)
test_loss, test_acc = model.evaluate(sequence_test_ds)
test_acc
```

Pretraining a word embedding

```
imdb_vocabulary = text_vectorization.get_vocabulary()
tokenize_no_padding = keras.layers.TextVectorization(
    vocabulary=imdb_vocabulary,
    split="whitespace",
    output_mode="int",
)
```

```
import tensorflow as tf

context_size = 4
window_size = 9

def window_data(token_ids):
    num_windows = tf.maximum(tf.size(token_ids) - context_size * 2, 0)
    windows = tf.range(window_size)[None, :]
    windows = windows + tf.range(num_windows)[:, None]
    windowed_tokens = tf.gather(token_ids, windows)
    return tf.data.Dataset.from_tensor_slices(windowed_tokens)

def split_label(window):
    left = window[:context_size]
    right = window[context_size + 1 :]
    bag = tf.concat((left, right), axis=0)
    label = window[4]
    return bag, label

dataset = keras.utils.text_dataset_from_directory(
    imdb_extract_dir / "train", batch_size=None
)
dataset = dataset.map(lambda x, y: x, num_parallel_calls=8)
dataset = dataset.map(tokenize_no_padding, num_parallel_calls=8)
dataset = dataset.interleave(window_data, cycle_length=8, num_parallel_c
dataset = dataset.map(split_label, num_parallel_calls=8)
```

```
hidden_dim = 64
inputs = keras.Input(shape=(2 * context_size,))
cbow_embedding = layers.Embedding(
    max_tokens,
    hidden_dim,
)
x = cbow_embedding(inputs)
x = layers.GlobalAveragePooling1D()(x)
outputs = layers.Dense(max_tokens, activation="sigmoid")(x)
cbow_model = keras.Model(inputs, outputs)
cbow_model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["sparse_categorical_accuracy"],
)
```

```
cbow_model.summary(line_length=80)
```

```
dataset = dataset.batch(1024).cache()
cbow_model.fit(dataset, epochs=4)
```

Using the pretrained embedding for classification

```
inputs = keras.Input(shape=(max_length,))
lstm_embedding = layers.Embedding(
    input_dim=max_tokens,
    output_dim=hidden_dim,
    mask_zero=True,
)
x = lstm_embedding(inputs)
x = layers.Bidirectional(layers.LSTM(hidden_dim))(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs, name="lstm_with_cbow")
```

```
lstm_embedding.embeddings.assign(cbow_embedding.embeddings)
```

```
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
model.fit(
    sequence_train_ds,
    validation_data=sequence_val_ds,
    epochs=10,
    callbacks=[early_stopping],
)
```

```
test_loss, test_acc = model.evaluate(sequence_test_ds)
test_acc
```

Task

Perform a comprehensive text classification on the IMDb dataset by:

1. Initializing and preparing the IMDb dataset for training, validation, and testing.
2. Training a bag-of-words linear classifier.
3. Training a bigram linear classifier.
4. Preparing data for sequence models.
5. Training a recurrent neural network (LSTM) using one-hot encoding.

6. Training a recurrent neural network (LSTM) with a learned embedding layer.
7. Pretraining a word embedding using a Continuous Bag-of-Words (CBOW) model on the IMDb dataset.
8. Fine-tuning the recurrent neural network with the pre-trained CBOW embedding. Finally, confirm all steps are executed and provide observations or results.

Initialize IMDb Datasets

Subtask:

Load the IMDb dataset using `text_dataset_from_directory` to create training, validation, and test datasets for model training.

Reasoning: The first instruction is to download and extract the IMDb dataset using `keras.utils.get_file`. The provided notebook already has code for this, which I will reuse.

```
import os, pathlib, shutil, random

zip_path = keras.utils.get_file(
    origin="https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar
    fname="imdb",
    extract=True,
)

imdb_extract_dir = pathlib.Path(zip_path) / "aclImdb"

print("IMDb dataset downloaded and extracted to:", imdb_extract_dir)
```

```
IMDb dataset downloaded and extracted to: /root/.keras/datasets/imdb/aclI
```

```
train_dir = pathlib.Path("imdb_train")
test_dir = pathlib.Path("imdb_test")
val_dir = pathlib.Path("imdb_val")

# Remove directories if they already exist to prevent FileExistsError
if test_dir.exists():
    shutil.rmtree(test_dir)
if train_dir.exists():
    shutil.rmtree(train_dir)
if val_dir.exists():
    shutil.rmtree(val_dir)

shutil.copytree(imdb_extract_dir / "test", test_dir)
```

```
val_percentage = 0.4  # Validate on 10,000 samples.
for category in ("neg", "pos"):
    src_dir = imdb_extract_dir / "train" / category
    src_files = os.listdir(src_dir)
    random.Random(1337).shuffle(src_files)
    num_val_samples = int(len(src_files) * val_percentage)

    os.makedirs(val_dir / category)
    for file in src_files[:num_val_samples]:
        shutil.copy(src_dir / file, val_dir / category / file)
    os.makedirs(train_dir / category)
    for file in src_files[num_val_samples:]:
        shutil.copy(src_dir / file, train_dir / category / file)
print("IMDb dataset prepared with train, validation, and test directorie
```

```
IMDb dataset prepared with train, validation, and test directories.
```

```
from keras.utils import text_dataset_from_directory

batch_size = 32
train_ds = text_dataset_from_directory(train_dir, batch_size=batch_size)
val_ds = text_dataset_from_directory(val_dir, batch_size=batch_size)
test_ds = text_dataset_from_directory(test_dir, batch_size=batch_size)
print("Train, validation, and test datasets created.")
```

```
Found 15000 files belonging to 2 classes.
Found 10000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.
Train, validation, and test datasets created.
```

Train Bag-of-Words Classifier

Subtask:

Vectorize the text data using a bag-of-words approach, build a linear classifier, and train the model on the prepared datasets.

Reasoning: The first instruction is to initialize a `layers.TextVectorization` layer with specified parameters and then adapt it to the training dataset without labels. This will prepare the text data for a bag-of-words model.

```
from keras import layers

max_tokens = 20_000
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    split="whitespace",
    output_mode="multi_hot",
)
# Create a dataset without labels for adaptation
train_ds_no_labels = train_ds.map(lambda x, y: x)
text_vectorization.adapt(train_ds_no_labels)
print("TextVectorization layer initialized and adapted.")
```

```
TextVectorization layer initialized and adapted.
```

```
bag_of_words_train_ds = train_ds.map(
    lambda x, y: (text_vectorization(x), y), num_parallel_calls=8
)
bag_of_words_val_ds = val_ds.map(
    lambda x, y: (text_vectorization(x), y), num_parallel_calls=8
)
bag_of_words_test_ds = test_ds.map(
    lambda x, y: (text_vectorization(x), y), num_parallel_calls=8
)
print("Bag-of-words datasets created for training, validation, and testi
```

```
Bag-of-words datasets created for training, validation, and testing.
```

```
def build_linear_classifier(max_tokens, name):
    inputs = keras.Input(shape=(max_tokens,))
    outputs = layers.Dense(1, activation="sigmoid")(inputs)
    model = keras.Model(inputs, outputs, name=name)
    model.compile(
        optimizer="adam",
        loss="binary_crossentropy",
        metrics=["accuracy"],
    )
    return model

print("build_linear_classifier function defined.")
```

```
build_linear_classifier function defined.
```

```
model = build_linear_classifier(max_tokens, "bag_of_words_classifier")
print("Bag-of-words linear classifier model created.")
```

```
Bag-of-words linear classifier model created.
```

```
early_stopping = keras.callbacks.EarlyStopping(
    monitor="val_loss",
    restore_best_weights=True,
    patience=2,
)
print("EarlyStopping callback created.")
```

```
EarlyStopping callback created.
```

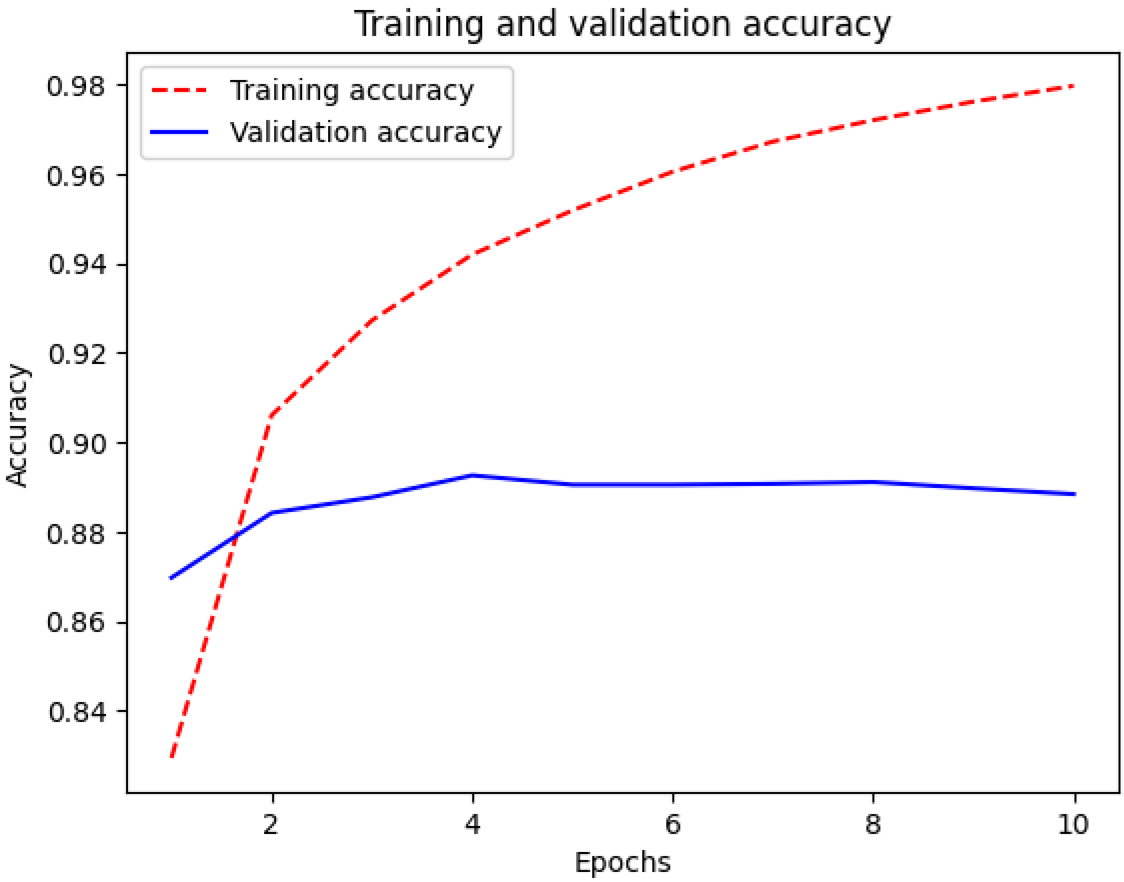
```
history = model.fit(
    bag_of_words_train_ds,
    validation_data=bag_of_words_val_ds,
    epochs=10,
    callbacks=[early_stopping],
)
print("Bag-of-words model training initiated.")
```

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 9s 17ms/step - accuracy: 0.8294 - loss: 0.48
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.9061 - loss: 0.317
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.9273 - loss: 0.25
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9420 - loss: 0.213
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9519 - loss: 0.185
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - accuracy: 0.9605 - loss: 0.16
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - accuracy: 0.9673 - loss: 0.146
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9721 - loss: 0.131
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.9762 - loss: 0.11
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.9797 - loss: 0.10
Bag-of-words model training initiated.
```

```
import matplotlib.pyplot as plt

accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, accuracy, "r--", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
print("Training and validation accuracy plotted.")
```



```
Training and validation accuracy plotted.
```

```
test_loss, test_acc = model.evaluate(bag_of_words_test_ds)
print(f"Bag-of-words test accuracy: {test_acc:.4f}")
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step - accuracy: 0.8829 - loss: 0.290
Bag-of-words test accuracy: 0.8829
```

```
max_tokens = 30_000 # Increase max_tokens for bigrams
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    split="whitespace",
    output_mode="multi_hot",
    ngrams=2, # Enable bigrams
)
text_vectorization.adapt(train_ds_no_labels)
print("TextVectorization layer for bigrams initialized and adapted.")
```

```
TextVectorization layer for bigrams initialized and adapted.
```

```
bigram_train_ds = train_ds.map(
    lambda x, y: (text_vectorization(x), y), num_parallel_calls=8
)
bigram_val_ds = val_ds.map(
    lambda x, y: (text_vectorization(x), y), num_parallel_calls=8
)
bigram_test_ds = test_ds.map(
    lambda x, y: (text_vectorization(x), y), num_parallel_calls=8
)
print("Bigram datasets created for training, validation, and testing.")
```

```
Bigram datasets created for training, validation, and testing.
```

```
model = build_linear_classifier(max_tokens, "bigram_classifier")
print("Bigram linear classifier model created.")
```

```
Bigram linear classifier model created.
```

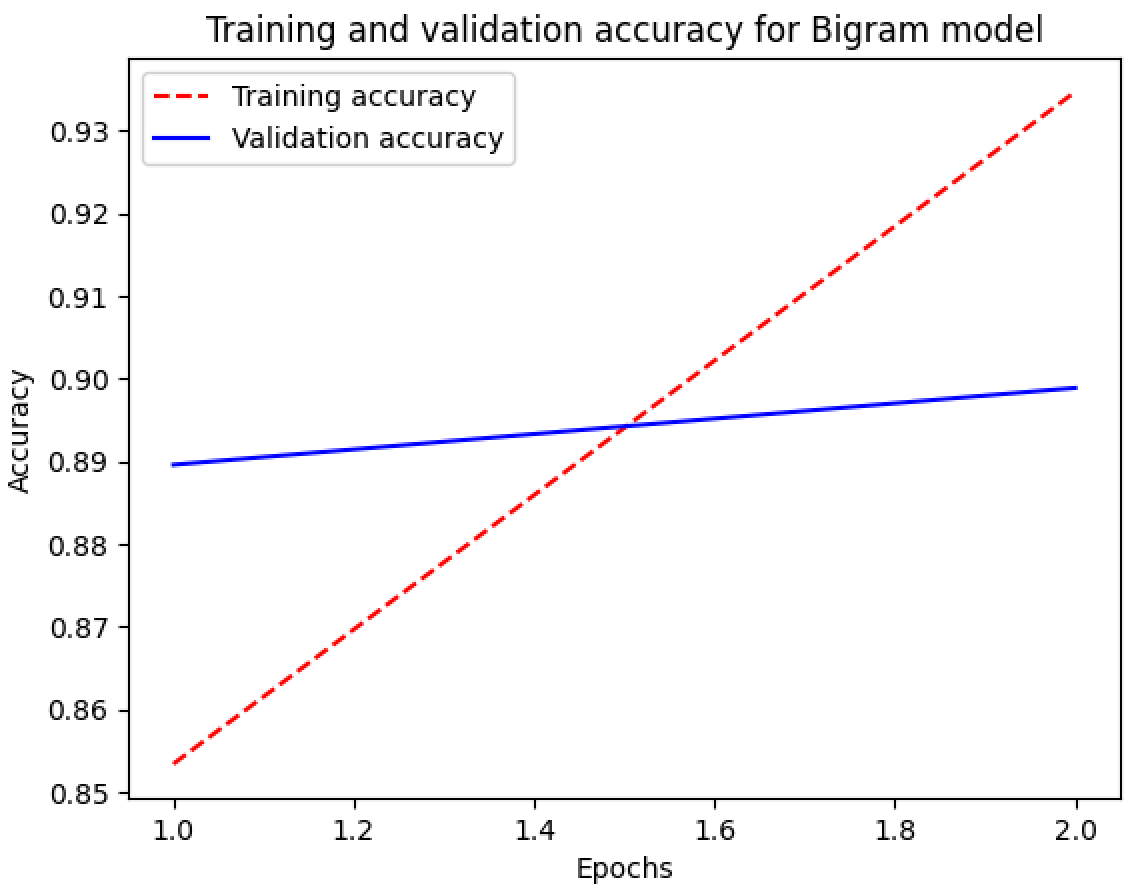
```
history = model.fit(
    bigram_train_ds,
    validation_data=bigram_val_ds,
    epochs=10,
    callbacks=[early_stopping],
)
print("Bigram linear classifier model training initiated.")
```

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 16ms/step - accuracy: 0.8534 - loss: 0.43
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9347 - loss: 0.24
Bigram linear classifier model training initiated.
```

```
import matplotlib.pyplot as plt

accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, accuracy, "r--", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy for Bigram model")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
print("Bigram model training and validation accuracy plotted.")
```



```
Bigram model training and validation accuracy plotted.
```

```
test_loss, test_acc = model.evaluate(bigram_test_ds)
print(f"Bigram test accuracy: {test_acc:.4f}")
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - accuracy: 0.8904 - loss: 0.33
Bigram test accuracy: 0.8904
```

```
max_length = 150 # Cutoff reviews after 150 words.
max_tokens = 10_000 # Restrict to top 10,000 words
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    split="whitespace",
    output_mode="int", # Output integer sequences
    output_sequence_length=max_length,
)
text_vectorization.adapt(train_ds_no_labels)
print("TextVectorization layer for sequence models initialized and adapt
```

```
TextVectorization layer for sequence models initialized and adapted.
```

```
sequence_train_ds = train_ds.map(
    lambda x, y: (text_vectorization(x), y), num_parallel_calls=8
)
sequence_val_ds = val_ds.map(
    lambda x, y: (text_vectorization(x), y), num_parallel_calls=8
)
sequence_test_ds = test_ds.map(
    lambda x, y: (text_vectorization(x), y), num_parallel_calls=8
)
print("Sequence datasets created for training, validation, and testing."
```

```
Sequence datasets created for training, validation, and testing.
```

Train Recurrent Model with One-Hot Encoding

Subtask:

Build and train a recurrent neural network (LSTM) using one-hot encoded input sequences.

Reasoning: The first instruction is to define the `hidden_dim` for the LSTM layer, which is a prerequisite for building the model. This step will set the dimensionality for the LSTM units.

```
hidden_dim = 64
print(f"Hidden dimension for LSTM set to: {hidden_dim}")
```

```
Hidden dimension for LSTM set to: 64
```

```
inputs = keras.Input(shape=(max_length,), dtype="int32")
one_hot_encoding = OneHotEncoding(max_tokens)
x = one_hot_encoding(inputs)
print("Input layer and OneHotEncoding layer created and connected.")
```

```
Input layer and OneHotEncoding layer created and connected.
```

```
x = layers.Bidirectional(layers.LSTM(hidden_dim))(x)
print("Bidirectional LSTM layer added.")
```

```
Bidirectional LSTM layer added.
```

```
x = layers.Dropout(0.5)(x)
print("Dropout layer added.")
```

```
Dropout layer added.
```

```
outputs = layers.Dense(1, activation="sigmoid")(x)
print("Dense output layer added.")
```

```
Dense output layer added.
```

```
model = keras.Model(inputs, outputs, name="lstm_with_one_hot")
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
print("LSTM with one-hot encoding model created and compiled.")
```

```
LSTM with one-hot encoding model created and compiled.
```

```
history = model.fit(
    sequence_train_ds,
    validation_data=sequence_val_ds,
    epochs=10,
    callbacks=[early_stopping],
)
print("LSTM with one-hot encoding model training initiated.")
```

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 59s 120ms/step - accuracy: 0.7096 - loss: 0.
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 55s 117ms/step - accuracy: 0.8725 - loss: 0.
LSTM with one-hot encoding model training initiated.
```

```
test_loss, test_acc = model.evaluate(sequence_test_ds)
print(f"LSTM with one-hot encoding test accuracy: {test_acc:.4f}")
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 26ms/step - accuracy: 0.8112 - loss: 0.4
LSTM with one-hot encoding test accuracy: 0.8112
```

Train Recurrent Model with Learned Embedding

Subtask:

Build and train a recurrent model (LSTM) using a trainable embedding layer to learn word representations.

Reasoning: The next step is to create a Keras `Input` layer, which is the starting point for defining the recurrent model with a learned embedding.

```
inputs = keras.Input(shape=(max_length,), dtype="int32")
print("Input layer created.")
```

```
Input layer created.
```

```
x = keras.layers.Embedding(
    input_dim=max_tokens,
    output_dim=hidden_dim,
    mask_zero=True,
)(inputs)
print("Embedding layer created and connected to input.")
```

```
Embedding layer created and connected to input.
```

```
x = keras.layers.Bidirectional(keras.layers.LSTM(hidden_dim))(x)
print("Bidirectional LSTM layer added.")
```

```
Bidirectional LSTM layer added.
```

```
x = keras.layers.Dropout(0.5)(x)
print("Dropout layer added.")
```

```
Dropout layer added.
```

```
outputs = keras.layers.Dense(1, activation="sigmoid")(x)
print("Dense output layer added.")
```

```
Dense output layer added.
```

```
model = keras.Model(inputs, outputs, name="lstm_with_embedding")
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
print("LSTM with embedding model created and compiled.")
```

```
LSTM with embedding model created and compiled.
```

```
history = model.fit(
    sequence_train_ds,
    validation_data=sequence_val_ds,
    epochs=10,
    callbacks=[early_stopping],
)
print("LSTM with embedding model training initiated.")
```

```
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 18s 31ms/step - accuracy: 0.7619 - loss: 0.4
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 11s 24ms/step - accuracy: 0.8807 - loss: 0.3
LSTM with embedding model training initiated.
```

```
test_loss, test_acc = model.evaluate(sequence_test_ds)
print(f"LSTM with learned embedding test accuracy: {test_acc:.4f}")
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - accuracy: 0.8200 - loss: 0.44
LSTM with learned embedding test accuracy: 0.8200
```

```
imdb_vocabulary = text_vectorization.get_vocabulary()
tokenize_no_padding = keras.layers.TextVectorization(
    vocabulary=imdb_vocabulary,
    split="whitespace",
    output_mode="int",
)
print("IMDb vocabulary extracted and tokenize_no_padding layer initializ
```

```
IMDb vocabulary extracted and tokenize_no_padding layer initialized.
```

```
import tensorflow as tf

context_size = 4
window_size = 9

def window_data(token_ids):
    num_windows = tf.maximum(tf.size(token_ids) - context_size * 2, 0)
    windows = tf.range(window_size)[None, :]
    windows = windows + tf.range(num_windows)[:, None]
    windowed_tokens = tf.gather(token_ids, windows)
    return tf.data.Dataset.from_tensor_slices(windowed_tokens)

def split_label(window):
    left = window[:context_size]
    right = window[context_size + 1 :]
    bag = tf.concat((left, right), axis=0)
    label = window[4]
    return bag, label

dataset = keras.utils.text_dataset_from_directory(
    imdb_extract_dir / "train", batch_size=None
)
dataset = dataset.map(lambda x, y: x, num_parallel_calls=8)
dataset = dataset.map(tokenize_no_padding, num_parallel_calls=8)
dataset = dataset.interleave(window_data, cycle_length=8, num_parallel_c
dataset = dataset.map(split_label, num_parallel_calls=8)
print("CBOW dataset prepared.")
```

```
Found 75000 files belonging to 3 classes.
CBOW dataset prepared.
```

```
hidden_dim = 64
print(f"Hidden dimension for CBOW embedding set to: {hidden_dim}")
```

```
Hidden dimension for CBOW embedding set to: 64
```

```
inputs = keras.Input(shape=(2 * context_size,))
cbow_embedding = layers.Embedding(
    max_tokens,
    hidden_dim,
)
x = cbow_embedding(inputs)
x = layers.GlobalAveragePooling1D()(x)
outputs = layers.Dense(max_tokens, activation="sigmoid")(x)
cbow_model = keras.Model(inputs, outputs)
cbow_model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["sparse_categorical_accuracy"],
)
print("CBOW model created and compiled.")
```

```
CBOW model created and compiled.
```

```
dataset = dataset.batch(1024).cache()
cbow_model.fit(dataset, epochs=4)
print("CBOW model training initiated and completed.")
```

```
Epoch 1/4
16503/16503 ━━━━━━━━━━━━━━━━━━━━ 424s 26ms/step - loss: 6.0188 - sparse_c
Epoch 2/4
   64/16503 ━━━━━━━━━━━━━━━━━━━━ 40s 2ms/step - loss: 5.6173 - sparse_cat
  self._interrupted_warning()
16503/16503 ━━━━━━━━━━━━━━━━━━━━ 46s 3ms/step - loss: 5.4980 - sparse_cat
Epoch 3/4
16503/16503 ━━━━━━━━━━━━━━━━━━━━ 50s 3ms/step - loss: 5.3780 - sparse_cat
Epoch 4/4
16503/16503 ━━━━━━━━━━━━━━━━━━━━ 52s 3ms/step - loss: 5.3222 - sparse_cat
CBOW model training initiated and completed.
```