EXPLANING MODEL BUILDING WITH EMPHASIS ON LEARNING EMBEDDING AND NOT USING THE PRETRAINED MODEL FOR EMBEDDING

```
import tensorflow as tf  # Import TensorFlow library
from tensorflow import keras  # Import Keras module from TensorFlow
from tensorflow.keras import layers  # Import layers module from Keras

# --- Data Preparation (Dummy Data for Demonstration) --- # Define example parameters for the data
max_tokens = 10000  # Example vocabulary size (number of unique words/tokens)
input_length = 50   # Example sequence length (maximum length of input sequences)
batch_size = 32     # Batch size for training

# Create dummy training dataset using tf.data.Dataset
int_train_ds = tf.data.Dataset.from_tensor_slices(
    (tf.random.uniform(shape=(1000, input_length), minval=0, maxval=max_tokens, dtype=tf.int64),  # Input sequences (random integers)
     tf.random.uniform(shape=(1000,), minval=0, maxval=2, dtype=tf.int64))  # Target labels (random 0 or 1)
).batch(batch_size)  # Batch the dataset into batches of size batch_size

# Create dummy validation dataset using tf.data.Dataset
int_val_ds = tf.data.Dataset.from_tensor_slices(
    (tf.random.uniform(shape=(200, input_length), minval=0, maxval=max_tokens, dtype=tf.int64),  # Input sequences (random integers)
     tf.random.uniform(shape=(200,), minval=0, maxval=2, dtype=tf.int64))  # Target labels (random 0 or 1)
).batch(batch_size)  # Batch the dataset into batches of size batch_size

# --- Model Definition ---
# Define the input layer, taking integer sequences as input
inputs = keras.Input(shape=(None,), dtype="int64")  # shape=(None,) allows variable sequence lengths

# Embedding layer: Converts integer sequences to dense vectors (embeddings)
embedded = layers.Embedding(input_dim=max_tokens, output_dim=256)(inputs)  # 256-dimensional embeddings

# Bidirectional LSTM layer: Processes the embedded sequences in both forward and backward directions
x = layers.Bidirectional(layers.LSTM(32))(embedded)  # 32 units in each LSTM layer

# Dropout layer: Applies dropout regularization to prevent overfitting
x = layers.Dropout(0.5)(x)  # Dropout rate of 0.5 (50%)

# Output layer: Produces a probability for binary classification (0 or 1)
outputs = layers.Dense(1, activation="sigmoid")(x)  # Sigmoid activation for binary output

# Create the Keras model by specifying the input and output layers
model = keras.Model(inputs, outputs)

# --- Model Compilation ---

# Compile the model by specifying the optimizer, loss function, and metrics
model.compile(optimizer="rmsprop",  # RMSprop optimizer
              loss="binary_crossentropy",  # Binary cross-entropy loss for binary classification
              metrics=["accuracy"])  # Track accuracy during training

# --- Model Summary ---

# Print a summary of the model architecture
model.summary()

# --- Callbacks ---

# Define callbacks to be used during training
callbacks = [
    keras.callbacks.ModelCheckpoint("embeddings_bidir_gru.keras",  # File path to save the model
                                    save_best_only=True)  # Save only the best model (based on validation loss)
]

# --- Model Training ---

# Train the model using the training data and validate on the validation data
model.fit(int_train_ds,  # Training dataset
          validation_data=int_val_ds,  # Validation dataset
          epochs=10,  # Number of epochs to train for
          callbacks=callbacks)  # Apply the defined callbacks

# --- Load Best Model ---

# Load the best saved model from the specified file path
model = keras.models.load_model("embeddings_bidir_gru.keras")

# --- Model Evaluation ---

# Evaluate the model on the validation dataset and print the results
loss, accuracy = model.evaluate(int_val_ds)
print(f"Validation loss: {loss}, Validation accuracy: {accuracy}")
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, None)	0
embedding_1 (Embedding)	(None, None, 256)	2,560,000
bidirectional_1 (Bidirectional)	(None, 64)	73,984
dropout_1 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

```
Total params: 2,634,049 (10.05 MB)
Trainable params: 2,634,049 (10.05 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 107ms/step - accuracy: 0.5360 - loss: 0.6928 - val_accuracy: 0.5000 - val_loss: 0.6943
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 71ms/step - accuracy: 0.5865 - loss: 0.6800 - val_accuracy: 0.5000 - val_loss: 0.6957
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.6674 - loss: 0.6505 - val_accuracy: 0.4600 - val_loss: 0.7008
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.9073 - loss: 0.5226 - val_accuracy: 0.5400 - val_loss: 0.7845
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 80ms/step - accuracy: 0.9677 - loss: 0.1454 - val_accuracy: 0.5150 - val_loss: 1.3259
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - accuracy: 1.0000 - loss: 0.0099 - val_accuracy: 0.5200 - val_loss: 1.5929
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.9987 - loss: 0.0070 - val_accuracy: 0.5100 - val_loss: 1.8466
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 71ms/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 0.5250 - val_loss: 2.0305
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 1.0000 - loss: 0.0013 - val_accuracy: 0.5350 - val_loss: 2.0792
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 80ms/step - accuracy: 1.0000 - loss: 7.1472e-04 - val_accuracy: 0.5250 - val_loss: 2.4489
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.4742 - loss: 0.6954
```

```
"""inputs = keras.Input(shape=(None,), dtype="int64")
embedded = layers.Embedding(input_dim=max_tokens, output_dim=256)(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
 loss="binary_crossentropy",
 metrics=["accuracy"])
model.summary()
callbacks = [
 keras.callbacks.ModelCheckpoint("embeddings_bidir_gru.keras",
 save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,
 callbacks=callbacks)
model = keras.models.load_model("embeddings_bidir_gru.keras")"""
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-47fa6c3f1d30> in <cell line: 0>()
----> 1 inputs = keras.Input(shape=(None,), dtype="int64")
      2 embedded = layers.Embedding(input_dim=max_tokens, output_dim=256)(inputs)
      3 x = layers.Bidirectional(layers.LSTM(32))(embedded)
      4 x = layers.Dropout(0.5)(x)
      5 outputs = layers.Dense(1, activation="sigmoid")(x)

NameError: name 'keras' is not defined
```

HERE, EXPLAININ THE PRETRAINED EMBEDDING MODELS, WORD2VEC AND GLOVE

```
import gensim.downloader as api  # Import the Gensim downloader for pre-trained models
from sklearn.manifold import TSNE  # Import t-SNE for dimensionality reduction (visualization)
import matplotlib.pyplot as plt  # Import Matplotlib for plotting

# --- 1. Load Pre-trained Word2Vec Model ---

# Load the pre-trained Word2Vec model trained on Google News dataset (300-dimensional vectors)
# 'word2vec-google-news-300' is the model identifier in Gensim's downloader
wv = api.load('word2vec-google-news-300')  # wv stands for word vectors
```

```
[==================================================] 100.0% 1662.8/1662.8MB downloaded
```

```
# --- 2. Example 1: Semantic Similarity ---

# Calculate and print the cosine similarity between word vectors
# Cosine similarity measures how similar two vectors are (between -1 and 1)
print(wv.similarity('king', 'queen'))  # Similarity between 'king' and 'queen' (should be high)
print(wv.similarity('man', 'woman'))  # Similarity between 'man' and 'woman' (should be high)
print(wv.similarity('king', 'man'))  # Similarity between 'king' and 'man' (should be high)
print(wv.similarity('king', 'car'))  # Similarity between 'king' and 'car' (should be low)
```

```
0.6510957
0.76640123
0.22942673
0.061895393
```

```
# --- 3. Example 2: Analogy ---

# Find words most similar to 'king' + 'woman' - 'man' (should be close to 'queen')
# This demonstrates the ability of Word2Vec to solve analogy problems
result = wv.most_similar(positive=['king', 'woman'], negative=['man'])
print(result)
```

```
[('queen', 0.7118193507194519), ('monarch', 0.6189674139022827), ('princess', 0.5902431011199951), ('crown_prince', 0.5499460697174072),
```

```
# --- 3. Example 2: Analogy ---

# Find words most similar to 'king' + 'woman' - 'man' (should be close to 'queen')
# This demonstrates the ability of Word2Vec to solve analogy problems

result = wv.most_similar(positive=['king', 'girl', 'young'], negative=['man', 'adult'])
print(result)
```

```
[('princess', 0.47500330209732056), ('prince', 0.470218300819397), ('Prince_Paras', 0.46112126111984253), ('queen', 0.45939457416534424)
```

```
# --- 4. Example 3: Getting the Vector for a Word ---

# Retrieve the vector representation of the word 'king'
vector_king = wv['king']
print(vector_king)  # Print the vector (300 numbers)
```

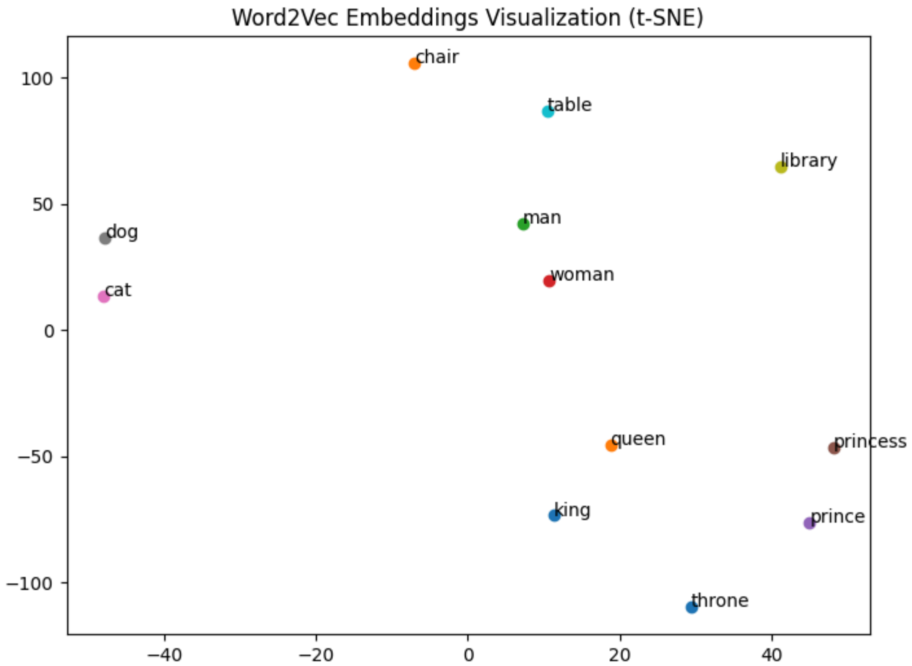
```
[ 1.25976562e-01  2.97851562e-02  8.60595703e-03  1.39648438e-01
 -2.56347656e-02 -3.61328125e-02  1.11816406e-01 -1.98242188e-01
  5.12695312e-02  3.63281250e-01 -2.42187500e-01 -3.02734375e-01
 -1.77734375e-01 -2.49023438e-02 -1.67968750e-01 -1.69921875e-01
  3.46679688e-02  5.21850586e-03  4.63867188e-02  1.28906250e-01
  1.36718750e-01  1.12792969e-01  5.95703125e-02  1.36718750e-01
  1.01074219e-01 -1.76757812e-01 -2.51953125e-01  5.98144531e-02
  3.41796875e-01 -3.11279297e-02  1.04492188e-01  6.17675781e-02
  1.24511719e-01  4.00390625e-01 -3.22265625e-01  8.39843750e-02
  3.90625000e-02  5.85937500e-03  7.03125000e-02  1.72851562e-01
  1.38671875e-01 -2.31445312e-01  2.83203125e-01  1.42578125e-01
  3.41796875e-01 -2.39257812e-02 -1.09863281e-01  3.32031250e-02
 -5.46875000e-02  1.53198242e-02 -1.62109375e-01  1.58203125e-01
 -2.59765625e-01  2.01416016e-02 -1.63085938e-01  1.35803223e-03
 -1.44531250e-01 -5.68847656e-02  4.29687500e-02 -2.46582031e-02
  1.85546875e-01  4.47265625e-01  9.58251953e-03  1.31835938e-01
  9.86328125e-02 -1.85546875e-01 -1.00097656e-01 -1.33789062e-01
 -1.25000000e-01  2.83203125e-01  1.23046875e-01  5.32226562e-02
 -1.77734375e-01  8.59375000e-02 -2.18505859e-02  2.05078125e-02
 -1.39648438e-01  2.51464844e-02  1.38671875e-01 -1.05468750e-01
  1.38671875e-01  8.88671875e-02 -7.51953125e-02 -2.13623047e-02
  1.72851562e-01  4.63867188e-02 -2.65625000e-01  8.91113281e-03
  1.49414062e-01  3.78417969e-02  2.38281250e-01 -1.24511719e-01
 -2.17773438e-01 -1.81640625e-01  2.97851562e-02  5.71289062e-02
 -2.89306641e-02  1.24511719e-02  9.66796875e-02 -2.31445312e-01
  5.81054688e-02  6.68945312e-02  7.08007812e-02 -3.08593750e-01
 -2.14843750e-01  1.45507812e-01 -4.27734375e-01 -9.39941406e-03
  1.54296875e-01 -7.66601562e-02  2.89062500e-01  2.77343750e-01
 -4.86373901e-04 -1.36718750e-01  3.24218750e-01 -2.46093750e-01
 -3.03649902e-03 -2.11914062e-01  1.25000000e-01  2.69531250e-01
  2.04101562e-01  8.25195312e-02 -2.01171875e-01 -1.60156250e-01
 -3.78417969e-02 -1.20117188e-01  1.15234375e-01 -4.10156250e-02
 -3.95507812e-02 -8.98437500e-02  6.34765625e-03  2.03125000e-01
  1.86523438e-01  2.73437500e-01  6.29882812e-02  1.41601562e-01
 -9.81445312e-02  1.38671875e-01  1.82617188e-01  1.73828125e-01
  1.73828125e-01 -2.37304688e-01  1.78710938e-01  6.34765625e-02
  2.36328125e-01 -2.08984375e-01  8.74023438e-02 -1.66015625e-01
 -7.91015625e-02  2.43164062e-01 -8.88671875e-02  1.26953125e-01
 -2.16796875e-01 -1.73828125e-01 -3.59375000e-01 -8.25195312e-02
 -6.49414062e-02  5.07812500e-02  1.35742188e-01 -7.47070312e-02
 -1.64062500e-01  1.15356445e-02  4.45312500e-01 -2.15820312e-01
 -1.11328125e-01 -1.92382812e-01  1.70898438e-01 -1.25000000e-01
  2.65502930e-03  1.92382812e-01 -1.74804688e-01  1.39648438e-01
  2.92968750e-01  1.13281250e-01  5.95703125e-02 -6.39648438e-02
  9.96093750e-02 -2.72216797e-02  1.96533203e-02  4.27246094e-02
 -2.46093750e-01  6.39648438e-02 -2.25585938e-01 -1.68945312e-01
```

```
 2.89916992e-03  8.20312500e-02  3.41796875e-01  4.32128906e-02
 1.32812500e-01  1.42578125e-01  7.61718750e-02  5.98144531e-02
-1.19140625e-01  2.74658203e-03 -6.29882812e-02 -2.72216797e-02
-4.82177734e-03 -8.20312500e-02 -2.49023438e-02 -4.00390625e-01
-1.06933594e-01  4.24804688e-02  7.76367188e-02 -1.16699219e-01
 7.37304688e-02 -9.22851562e-02  1.07910156e-01  1.58203125e-01
 4.24804688e-02  1.26953125e-01  3.61328125e-02  2.67578125e-01
-1.01074219e-01 -3.02734375e-01 -5.76171875e-02  5.05371094e-02
 5.26428223e-04 -2.07031250e-01 -1.38671875e-01 -8.97216797e-03
-2.78320312e-02 -1.41601562e-01  2.07031250e-01 -1.58203125e-01
 1.27929688e-01  1.49414062e-01 -2.24609375e-02 -8.44726562e-02
 1.22558594e-01  2.15820312e-01 -2.13867188e-01 -3.12500000e-01
```

```python
# --- 5. Example 4: Checking if a Word Exists in the Vocabulary ---

# Check if the word 'cat' exists in the Word2Vec vocabulary
if 'cat' in wv:
    print("Cat is in the vocabulary")
else:
    print("Cat is not in the vocabulary")
```

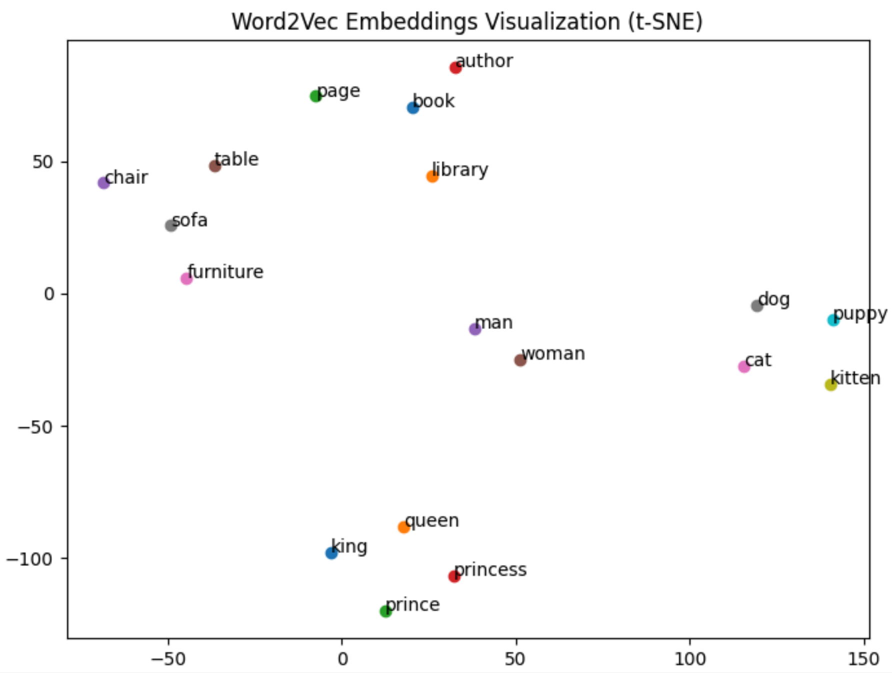
```
Cat is in the vocabulary
```

```python
# --- 6. Example 5: Visualizing Embeddings (using t-SNE) ---

# List of words to visualize
words = ['king', 'queen', 'man', 'woman', 'prince', 'princess', 'cat', 'dog', 'library', 'table', 'throne', 'chair']

# Get the vector representations for the words (only if they are in the vocabulary)
embeddings = [wv[word] for word in words if word in wv]

# Convert the list of embeddings to a 2D NumPy array
import numpy as np  # Import NumPy for array manipulation
embeddings = np.array(embeddings) # Convert the list of embeddings to a NumPy array

# Reduce the dimensionality of the vectors to 2D for visualization using t-SNE
# t-SNE (t-Distributed Stochastic Neighbor Embedding) is a technique for visualizing high-dimensional data
# Set perplexity to a value less than the number of samples (8 in this case)
tsne = TSNE(n_components=2, perplexity=5, random_state=42)  # Reduce to 2 dimensions, perplexity=5
embeddings_2d = tsne.fit_transform(embeddings)

# Create a scatter plot of the 2D embeddings
plt.figure(figsize=(8, 6))  # Set the figure size
for i, word in enumerate(words):
    if word in wv:  # Only plot if the word is in the vocabulary
        plt.scatter(embeddings_2d[i, 0], embeddings_2d[i, 1])  # Plot the point
        plt.annotate(word, (embeddings_2d[i, 0], embeddings_2d[i, 1]))  # Add the word label
plt.title('Word2Vec Embeddings Visualization (t-SNE)')  # Set the plot title
plt.show()  # Display the plot
```




```python
# --- 6. Example 5: Visualizing Embeddings (using t-SNE) ---

# List of words to visualize
words = ['king', 'queen', 'prince', 'princess', 'man', 'woman',
         'cat', 'dog', 'kitten', 'puppy',
         'book', 'library', 'page', 'author',
```

```
        'chair', 'table', 'furniture', 'sofa']

# Get the vector representations for the words (only if they are in the vocabulary)
embeddings = [wv[word] for word in words if word in wv]

# Convert the list of embeddings to a 2D NumPy array
import numpy as np  # Import NumPy for array manipulation
embeddings = np.array(embeddings) # Convert the list of embeddings to a NumPy array

tsne = TSNE(n_components=2, perplexity=5, random_state=42)  # Reduce to 2 dimensions, perplexity=5
embeddings_2d = tsne.fit_transform(embeddings)

# Create a scatter plot of the 2D embeddings
plt.figure(figsize=(8, 6))  # Set the figure size
for i, word in enumerate(words):
    if word in wv:  # Only plot if the word is in the vocabulary
        plt.scatter(embeddings_2d[i, 0], embeddings_2d[i, 1])  # Plot the point
        plt.annotate(word, (embeddings_2d[i, 0], embeddings_2d[i, 1]))  # Add the word label
plt.title('Word2Vec Embeddings Visualization (t-SNE)')  # Set the plot title
plt.show()  # Display the plot
```



NOW LETS USE GLOVE TECHNIQUE

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip
```

```
--2025-02-24 14:25:18--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2025-02-24 14:25:18--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2025-02-24 14:25:18--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.00MB/s    in 2m 39s

2025-02-24 14:27:57 (5.19 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]

Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# ... (Your load_glove_vectors, sentences_train, sentences_val, text_vectorization, embedding_dim, etc. code) ...

# 1. Prepare your training and validation sentences
sentences_train = ["This is a sample sentence.", "Another sentence with more words.", "Train data example one", "Train data example two"]
sentences_val = ["Validation sentence one.", "Validation sentence two."]
all_sentences = sentences_train + sentences_val  # Combine for consistent vocabulary

# 2. Create a TextVectorization layer with a larger vocabulary size
max_tokens = 10000  # Increase the maximum size of the vocabulary to match the training data
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=10  # Adjust according to your data
)

# 3. Adapt the TextVectorization layer to all sentences
text_vectorization.adapt(all_sentences)

# 4. Load GloVe vectors
path_to_glove_file = "glove.6B.100d.txt"
embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs
print(f"Found {len(embeddings_index)} word vectors.")

# 5. Create the embedding matrix using the consistent vocabulary
embedding_dim = 100
vocabulary = text_vectorization.get_vocabulary()
word_index = dict(zip(vocabulary, range(len(vocabulary))))
embedding_matrix = np.zeros((max_tokens, embedding_dim))
for word, i in word_index.items():
    if i < max_tokens:
        embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

# 6. Create the Embedding layer
embedding_layer = layers.Embedding(
    max_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
    mask_zero=True,
)
# 7. Vectorize your training and validation data
int_train_ds = tf.data.Dataset.from_tensor_slices(
    (text_vectorization(sentences_train), labels_train)  # Use text_vectorization to transform text to indices
).batch(2)

int_val_ds = tf.data.Dataset.from_tensor_slices(
    (text_vectorization(sentences_val), labels_val)  # Use text_vectorization to transform text to indices
).batch(2)

# 8. Build, compile, and train your model (rest of the code remains the same)
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = embedding_layer(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()
callbacks = [
    keras.callbacks.ModelCheckpoint("glove_embeddings_sequence_model.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,
          callbacks=callbacks)
model = keras.models.load_model("glove_embeddings_sequence_model.keras")
# Now 'int_train_ds' and 'int_val_ds' have indices matching the embedding layer

# --- Modified Functions using GloVe Vocabulary ---

def get_glove_vector(word, embeddings_index):
    """Retrieves the GloVe vector for a word."""
    return embeddings_index.get(word)
```

```
def glove_similarity(word1, word2, embeddings_index):
    """Calculates cosine similarity between two words."""
    vec1 = get_glove_vector(word1, embeddings_index)
    vec2 = get_glove_vector(word2, embeddings_index)
    if vec1 is not None and vec2 is not None:
        return cosine_similarity([vec1], [vec2])[0, 0]
    else:
        return None

def find_most_similar(word, embeddings_index, top_n=5):
    """Finds the top N most similar words to a given word."""
    word_vector = get_glove_vector(word, embeddings_index)
    if word_vector is None:
        return None

    similarities = []
    for vocab_word, vector in embeddings_index.items():
        if vocab_word != word:
            similarity = cosine_similarity([word_vector], [vector])[0, 0]
            similarities.append((vocab_word, similarity))

    similarities.sort(key=lambda x: x[1], reverse=True)
    return similarities[:top_n]

def check_word_vector(word, embeddings_index):
    """Checks if a word has a GloVe vector."""
    if get_glove_vector(word, embeddings_index) is not None:
        print(f"'{word}' has a GloVe vector.")
    else:
        print(f"'{word}' does not have a GloVe vector.")

def get_word_vector(word, embeddings_index):
    """Retrieves the GloVe vector for a word."""
    vector = get_glove_vector(word, embeddings_index)
    if vector is not None:
        print(f"Vector for '{word}': {vector}")
    else:
        print(f"'{word}' has no vector.")

def visualize_embeddings(words, embeddings_index):
    """Visualizes GloVe embeddings using t-SNE."""
    embeddings = [get_glove_vector(word, embeddings_index) for word in words if get_glove_vector(word, embeddings_index) is not None]
    words_filtered = [word for word in words if get_glove_vector(word, embeddings_index) is not None]

    if not embeddings:
        print("No embeddings to visualize.")
        return

    tsne = TSNE(n_components=2, random_state=42)
    embeddings_2d = tsne.fit_transform(embeddings)

    plt.figure(figsize=(8, 6))
    for i, word in enumerate(words_filtered):
        plt.scatter(embeddings_2d[i, 0], embeddings_2d[i, 1])
        plt.annotate(word, (embeddings_2d[i, 0], embeddings_2d[i, 1]))
    plt.title('GloVe Embeddings Visualization (t-SNE)')
    plt.show()
```

```
Found 400000 word vectors.
Model: "functional_11"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_11 (InputLayer)	(None, None)	0	-
embedding_11 (Embedding)	(None, None, 100)	1,000,000	input_layer_11[0][0]
not_equal_11 (NotEqual)	(None, None)	0	input_layer_11[0][0]
bidirectional_11 (Bidirectional)	(None, 64)	34,048	embedding_11[0][0], not_equal_11[0][0]
dropout_11 (Dropout)	(None, 64)	0	bidirectional_11[0][0]
dense_11 (Dense)	(None, 1)	65	dropout_11[0][0]

```
 Total params: 1,034,113 (3.94 MB)
 Trainable params: 34,113 (133.25 KB)
 Non-trainable params: 1,000,000 (3.81 MB)
Epoch 1/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 8s 3s/step - accuracy: 0.3333 - loss: 0.7824 - val_accuracy: 0.5000 - val_loss: 0.7404
Epoch 2/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.3333 - loss: 0.7216 - val_accuracy: 0.5000 - val_loss: 0.7398
Epoch 3/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.6667 - loss: 0.5809 - val_accuracy: 0.5000 - val_loss: 0.7455
Epoch 4/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.8333 - loss: 0.4488 - val_accuracy: 0.5000 - val_loss: 0.7528
Epoch 5/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.8333 - loss: 0.6447 - val_accuracy: 0.5000 - val_loss: 0.7609
Epoch 6/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 1.0000 - loss: 0.4763 - val_accuracy: 0.5000 - val_loss: 0.7651
Epoch 7/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.8333 - loss: 0.5213 - val_accuracy: 0.5000 - val_loss: 0.7757
Epoch 8/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - accuracy: 0.5000 - loss: 0.6479 - val_accuracy: 0.0000e+00 - val_loss: 0.7806
Epoch 9/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.8333 - loss: 0.3933 - val_accuracy: 0.0000e+00 - val_loss: 0.7905
Epoch 10/10
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.8333 - loss: 0.5151 - val_accuracy: 0.0000e+00 - val_loss: 0.8034
```

```python
# --- Usage Examples ---

# Examples of Semantic Similarity
print(f"Similarity ('king', 'queen'): {glove_similarity('king', 'queen', embeddings_index)}")
print(f"Similarity ('man', 'woman'): {glove_similarity('man', 'woman', embeddings_index)}")
print(f"Similarity ('king', 'man'): {glove_similarity('king', 'man', embeddings_index)}")
print(f"Similarity ('king', 'car'): {glove_similarity('king', 'car', embeddings_index)}")
print(f"Similarity ('cat', 'dog'): {glove_similarity('cat', 'dog', embeddings_index)}")
```

```
Similarity ('king', 'queen'): 0.7507690787315369
Similarity ('man', 'woman'): 0.8323494791984558
Similarity ('king', 'man'): 0.5118681192398071
Similarity ('king', 'car'): 0.28304237127304077
Similarity ('cat', 'dog'): 0.8798074722290039
```

```python
# Examples of Finding Similar Words (Continued)
print(f"Most similar to 'cat': {find_most_similar('cat', embeddings_index)}")
print(f"Most similar to 'book': {find_most_similar('book', embeddings_index)}")
```

```
Most similar to 'cat': [('dog', 0.8798075), ('rabbit', 0.74244267), ('cats', 0.7323004), ('monkey', 0.728871), ('pet', 0.71901405)]
Most similar to 'book': [('books', 0.84764856), ('novel', 0.81811666), ('published', 0.8023924), ('story', 0.7941391), ('author', 0.7937
```

```python
# Examples of Checking Word Vector Existence
check_word_vector("king", embeddings_index)
check_word_vector("randomword", embeddings_index)
```

```
'king' has a GloVe vector.
'randomword' does not have a GloVe vector.
```

```python
# Examples of Getting the Vector for a Word
get_word_vector("queen", embeddings_index)
get_word_vector("anotherword", embeddings_index)
```

```
Vector for 'queen': [-0.50045  -0.70826   0.55388   0.673     0.22486   0.60281  -0.26194
  0.73872  -0.65383  -0.21606  -0.33806   0.24498  -0.51497   0.8568
 -0.37199  -0.58824   0.30637  -0.30668  -0.2187    0.78369  -0.61944
 -0.54925   0.43067  -0.027348  0.97574   0.46169   0.11486  -0.99842
  1.0661   -0.20819   0.53158   0.40922   1.0406    0.24943   0.18709
  0.41528  -0.95408   0.36822  -0.37948  -0.6802   -0.14578  -0.20113
  0.17113  -0.55705   0.7191    0.070014 -0.23637   0.49534   1.1576
 -0.05078   0.25731  -0.091052  1.2663    1.1047   -0.51584  -2.0033
 -0.64821   0.16417   0.32935   0.048484  0.18997   0.66116   0.080882
```

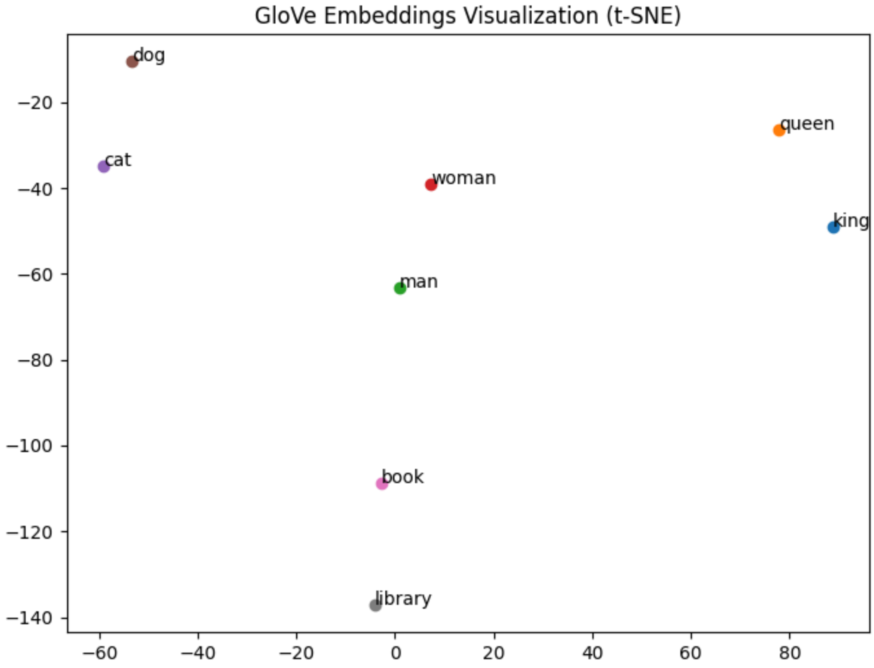
```
 0.3364    0.22758   0.1462   -0.51005   0.63777   0.47299  -0.3282
 0.083899 -0.78547   0.099148  0.039176  0.27893   0.11747   0.57862
 0.043639 -0.15965  -0.35304  -0.048965 -0.32461   1.4981    0.58138
-1.132    -0.60673  -0.37505  -1.1813    0.80117  -0.50014  -0.16574
-0.70584   0.43012   0.51051  -0.8033   -0.66572  -0.63717  -0.36032
 0.13347  -0.56075 ]
'anotherword' has no vector.
```

```
# Example of Visualization
#words_to_visualize = ['king', 'queen', 'man', 'woman', 'cat', 'dog', 'book', 'library']
#visualize_embeddings(words_to_visualize, embeddings_index)
```

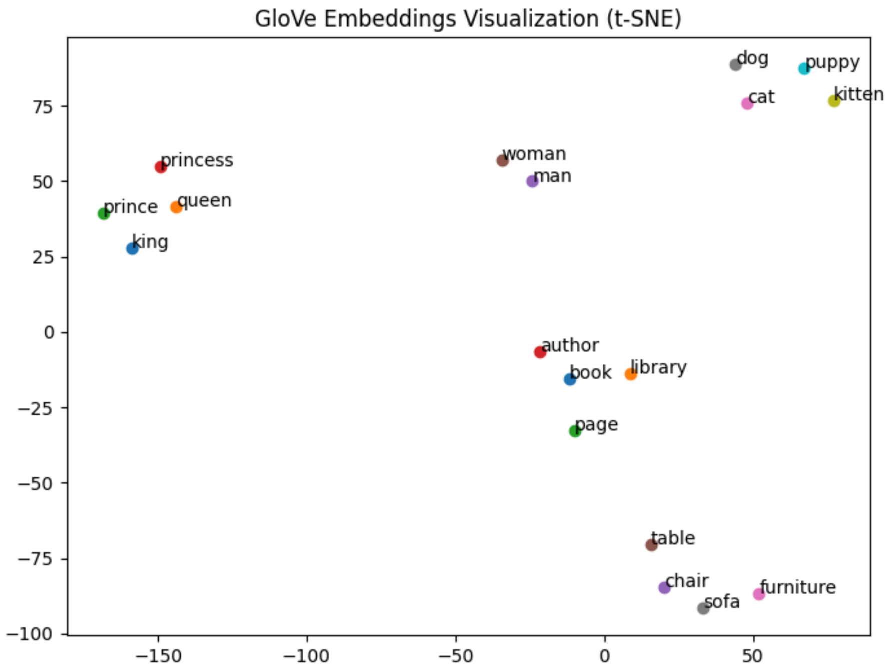
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-73-81beebf48764> in <cell line: 0>()
      1 # Example of Visualization
      2 words_to_visualize = ['king', 'queen', 'man', 'woman', 'cat', 'dog', 'book', 'library']
----> 3 visualize_embeddings(words_to_visualize, embeddings_index)

                         4 frames
/usr/local/lib/python3.11/dist-packages/sklearn/manifold/_t_sne.py in _check_params_vs_input(self, X)
    859
    860     def _check_params_vs_input(self, X):
--> 861         if self.perplexity >= X.shape[0]:
    862             raise ValueError("perplexity must be less than n_samples")
    863

AttributeError: 'list' object has no attribute 'shape'
```

Next steps: Explain error

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

def visualize_embeddings(words, embeddings_index):
    """Visualizes GloVe embeddings using t-SNE."""
    embeddings = [get_glove_vector(word, embeddings_index) for word in words if get_glove_vector(word, embeddings_index) is not None]
    words_filtered = [word for word in words if get_glove_vector(word, embeddings_index) is not None]

    if not embeddings:
        print("No embeddings to visualize.")
        return

    # Convert the list of embeddings to a NumPy array
    embeddings = np.array(embeddings)

    # Lower the perplexity to be significantly less than the number of samples
    tsne = TSNE(n_components=2, perplexity=3, random_state=42)  # Reduced perplexity to 3
    embeddings_2d = tsne.fit_transform(embeddings)

    plt.figure(figsize=(8, 6))
    for i, word in enumerate(words_filtered):
        plt.scatter(embeddings_2d[i, 0], embeddings_2d[i, 1])
        plt.annotate(word, (embeddings_2d[i, 0], embeddings_2d[i, 1]))
    plt.title('GloVe Embeddings Visualization (t-SNE)')
    plt.show()

words_to_visualize = ['king', 'queen', 'man', 'woman', 'cat', 'dog', 'book', 'library']
visualize_embeddings(words_to_visualize, embeddings_index)
```




```
words_to_visualize = ['king', 'queen', 'prince', 'princess', 'man', 'woman',
          'cat', 'dog', 'kitten', 'puppy',
          'book', 'library', 'page', 'author',
          'chair', 'table', 'furniture', 'sofa']

visualize_embeddings(words_to_visualize, embeddings_index)
```



--- Use Cases for Word2Vec and GloVe ---

1. Semantic Similarity and Relatedness:

- Finding synonyms and related words.
- Measuring the semantic distance between words or documents.
- Example: Building a search engine that understands the meaning of queries.

2. Analogy Tasks:

- Solving analogy problems like "king - man + woman = queen".
- Example: Building a question-answering system that can reason about relationships between words.

3. Feature Engineering for NLP Models:

- Using pre-trained embeddings as input features for deep learning models.
- Example: Improving the performance of sentiment analysis, text classification, or machine translation models.

4. Information Retrieval:

- Finding documents that are semantically similar to a query.
- Example: Building a document retrieval system that understands the meaning of documents.

5. Word Sense Disambiguation:

- Identifying the correct meaning of a word in a given context.