```
%import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Input, Dense, Embedding, Lambda
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

print("TensorFlow Version:", tf.__version__)
```

```
TensorFlow Version: 2.19.0
```

```
 # Our sample text corpus
corpus = [
    'the cat sat on the mat',
    'the dog sat on the log',
    'the cat chased the dog'
]

# 1. Tokenization: Convert words to integers
tokenizer = Tokenizer()
tokenizer.fit_on_texts(corpus)

# The dictionary mapping words to integers
word_index = tokenizer.word_index
print("Word Index:", word_index)

# Vocabulary size is the number of unique words + 1 (for padding)
vocab_size = len(word_index) + 1

# Convert sentences to sequences of integers
sequences = tokenizer.texts_to_sequences(corpus)
print("Integer Sequences:", sequences)
```

```
Word Index: {'the': 1, 'cat': 2, 'sat': 3, 'on': 4, 'dog': 5, 'mat': 6, 'log': 7, 'chased': 8}
Integer Sequences: [[1, 2, 3, 4, 1, 6], [1, 5, 3, 4, 1, 7], [1, 2, 8, 1, 5]]
```

```
# Create context/target pairs
window_size = 1
X, y = [], []

for sequence in sequences:
    for i in range(window_size, len(sequence) - window_size):
        # Target word is the word at the center of the window
        target_word = sequence[i]

        # Context words are the words around the target
        context_words = sequence[i - window_size : i] + sequence[i + 1 : i + 1 + window_size]

        X.append(context_words)
        y.append(target_word)

# Convert to NumPy arrays for the model
X = np.array(X)
y = np.array(y)

# One-hot encode the target variable, as this is a classification task
y = to_categorical(y, num_classes=vocab_size)

print("Context (X) shape:", X.shape)
print("Target (y) shape:", y.shape)
print("\nSample Context:", X[0])
print("Sample Target (one-hot):", y[0])
```

```
Context (X) shape: (11, 2)
Target (y) shape: (11, 9)

Sample Context: [1 3]
Sample Target (one-hot): [0. 0. 1. 0. 0. 0. 0. 0. 0.]
```

```
# Define model parameters
embedding_dim = 10
context_size = 2 * window_size

# Define the model using the Keras Functional API
# Input layer for context words
input_layer = Input(shape=(context_size,))
```

```
# Embedding layer
embedding_layer = Embedding(input_dim=vocab_size, output_dim=embedding_dim, name='embedding_layer')(input_layer)

# Average the embeddings of the context words
avg_layer = Lambda(lambda x: tf.reduce_mean(x, axis=1))(embedding_layer)

# Output layer to predict the target word
output_layer = Dense(vocab_size, activation='softmax')(avg_layer)

# Create the CBOW model
cbow_model = Model(inputs=input_layer, outputs=output_layer)

# Compile the model
cbow_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
cbow_model.summary()

# Train the model
history = cbow_model.fit(X, y, epochs=200, verbose=1)
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 2)	0
embedding_layer (Embedding)	(None, 2, 10)	90
lambda (Lambda)	(None, 10)	0
dense (Dense)	(None, 9)	99

Total params: 189 (756.00 B)
Trainable params: 189 (756.00 B)
Non-trainable params: 0 (0.00 B)

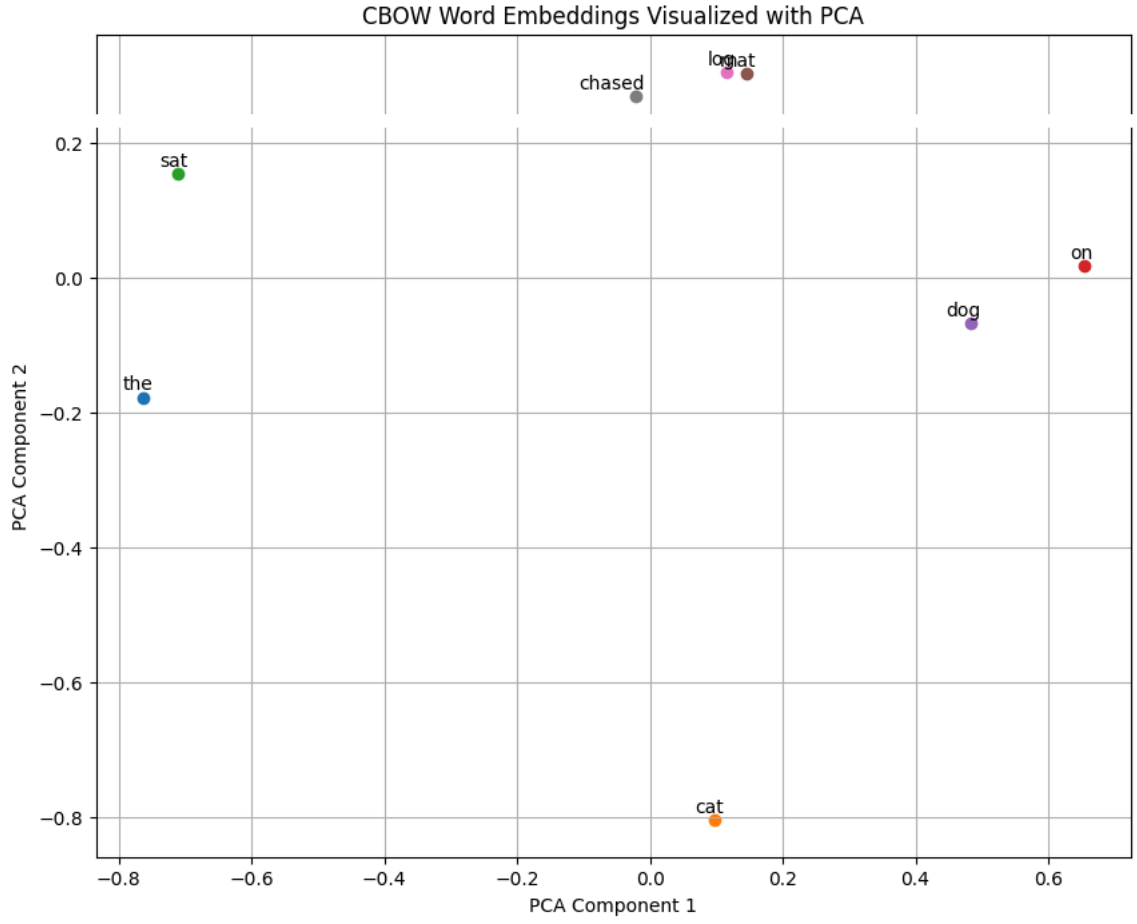
```
Epoch 1/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.0909 - loss: 2.1933
Epoch 2/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.1818 - loss: 2.1909
Epoch 3/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.1818 - loss: 2.1884
Epoch 4/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.1818 - loss: 2.1859
Epoch 5/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.1818 - loss: 2.1835
Epoch 6/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.1818 - loss: 2.1810
Epoch 7/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.3636 - loss: 2.1786
Epoch 8/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.3636 - loss: 2.1761
Epoch 9/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.3636 - loss: 2.1737
Epoch 10/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.3636 - loss: 2.1712
Epoch 11/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.3636 - loss: 2.1687
Epoch 12/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.3636 - loss: 2.1663
Epoch 13/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.3636 - loss: 2.1638
Epoch 14/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.3636 - loss: 2.1613
Epoch 15/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.3636 - loss: 2.1588
Epoch 16/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.3636 - loss: 2.1563
Epoch 17/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.3636 - loss: 2.1538
Epoch 18/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.4545 - loss: 2.1513
Epoch 19/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.5455 - loss: 2.1488
Epoch 20/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.5455 - loss: 2.1463
Epoch 21/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.5455 - loss: 2.1437
Epoch 22/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - accuracy: 0.5455 - loss: 2.1412
```

```
# 1. Extract the learned embeddings
word_embeddings = cbow_model.get_layer('embedding_layer').get_weights()[0]

# 2. Create a dictionary to map words to their learned vectors
embedding_dict = {}
for word, index in word_index.items():
    embedding_dict[word] = word_embeddings[index]
```

```
# Print the embedding for a sample word
print(f"Embedding for the word 'cat':\n{embedding_dict['cat']}")

# 3. Visualize the embeddings in 2D space
# We use PCA to reduce the 10-dimensional embeddings to 2 dimensions
pca = PCA(n_components=2)
reduced_embeddings = pca.fit_transform(list(embedding_dict.values()))

plt.figure(figsize=(10, 8))
for i, word in enumerate(embedding_dict.keys()):
    x, y_coord = reduced_embeddings[i]
    plt.scatter(x, y_coord)
    plt.annotate(word, (x, y_coord), xytext=(5, 2), textcoords='offset points', ha='right', va='bottom')

plt.title("CBOW Word Embeddings Visualized with PCA")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.grid(True)
plt.show()
```

```
Epoch 38/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.6364 - loss: 2.0983
Epoch 39/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.6364 - loss: 2.0955
```

```
Embedding for the word 'cat':
[ 0.20320508 -0.2745773   0.05219927  0.03624204  0.02861902 -0.3063684
 -0.33978415  0.02899507  0.20597608  0.08294297]
```




```
Epoch 60/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - accuracy: 0.6364 - loss: 2.0315
Epoch 61/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.6364 - loss: 2.0282
Epoch 62/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step - accuracy: 0.6364 - loss: 2.0249
Epoch 63/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step - accuracy: 0.6364 - loss: 2.0216
Epoch 64/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 0.6364 - loss: 2.0182
Epoch 65/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - accuracy: 0.6364 - loss: 2.0148
Epoch 66/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step - accuracy: 0.6364 - loss: 2.0114
Epoch 67/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 0.6364 - loss: 2.0080
Epoch 68/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 0.6364 - loss: 2.0045
Epoch 69/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step - accuracy: 0.6364 - loss: 2.0011
Epoch 70/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step - accuracy: 0.6364 - loss: 1.9976
Epoch 71/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - accuracy: 0.6364 - loss: 1.9941
Epoch 72/200
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.6364 - loss: 1.9905
```