Họ và tên: Quách Xuân Phúc

MSV: B20DCCN513

6.1

Word Embedding là một cách biểu diễn các từ và tài liệu bằng các vector dày đặc. Điều này có nghĩa là mỗi từ được đại diện bởi một vector gồm các số, trong đó các số đại diện cho các mối quan hệ giữa từ đó và các từ khác trong từ vựng.

Word Embedding hữu ích cho các tác vụ xử lý ngôn ngữ tự nhiên (NLP) vì chúng cho phép chúng ta biểu diễn ý nghĩa của các từ theo cách có thể được hiểu bởi máy tính. Ví dụ, hai từ có ý nghĩa tương tự sẽ có biểu diễn vector tương tự, trong khi hai từ có ý nghĩa khác nhau sẽ có biểu diễn vector khác nhau.

Hai ví dụ phổ biến về phương pháp học word embedding từ văn bản bao gồm:

- Word2Vec.
- GloVe.

Có hai cách tiếp cận chính để tạo word embedding:

- Co-occurrence-based embedding: Dựa trên tần suất xuất hiện của các từ trong cùng một ngữ cảnh.
- Distributional embedding: Dựa trên phân phối xác suất của các từ trong văn bản.

*** Keras Embedding Layer**

Keras cung cấp một lớp Embedding có thể được sử dụng để học các word embedding. Lớp Embedding lấy dữ liệu đầu vào được mã hóa theo số và trả về một biểu diễn vector dày đặc cho mỗi từ trong chuỗi đầu vào.

Để sử dụng lớp Embedding, trước tiên chúng ta cần chuẩn bị dữ liệu đầu vào của mình. Điều này bao gồm mã hóa theo số các từ trong dữ liệu của chúng ta, sao cho mỗi từ được đại diện bởi một số duy nhất. Chúng ta có thể sử dụng API Tokenizer trong Keras để làm điều này.

Khi dữ liệu đầu vào của chúng ta đã được mã hóa theo số, chúng ta có thể tạo một lớp Embedding. Lớp Embedding lấy ba đối số:

- input_dim: Kích thước của từ vựng trong dữ liệu văn bản.
- output_dim: Kích thước của không gian vector trong đó các từ sẽ được nhúng.
- input_length: Độ dài của các chuỗi đầu vào.

Đầu ra của lớp Embedding là một vector 2D với một embedding cho mỗi từ trong chuỗi đầu vào.

Mã sau đây cho thấy cách học một embedding bằng lớp Embedding của Keras:

```
import keras

# Tạo một lớp Embedding với một từ vựng gồm 200 từ và một không gian vector gồm 32 chiều.
e = keras.layers.Embedding(200, 32)

# Tạo một mô hình với lớp Embedding là lớp đầu tiên.
model = keras.Sequential([e])

# Biên dịch mô hình.
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Huấn luyện mô hình trên tập dữ liệu gồm văn bản và nhãn.
model.fit(x_train, y_train, epochs=10)

# Đánh giá mô hình trên tập dữ liệu kiểm tra.
model.evaluate(x_test, y_test)
```

Trong ví dụ này, chúng ta đang sử dụng lớp Embedding để học một embedding cho tập dữ liệu gồm 200 từ. Chúng ta đang sử dụng một không gian vector gồm 32 chiều, có nghĩa là mỗi từ sẽ được đại diện bởi một vector gồm 32 số.

Khi mô hình được đào tạo, chúng ta có thể sử dụng nó để dự đoán nhãn của dữ liệu mới. Để làm điều này, chúng ta chỉ cần truyền dữ liệu mới cho mô hình và nó sẽ dự đoán nhãn.

Mã sau đây cho thấy cách sử dụng embedding GloVe đã được đào tạo sẵn trong Keras:

```
import keras

# Tải mô hình embedding GloVe.
embeddings = numpy.load('glove.6B.300d.txt', encoding='latin1')

# Tạo một lớp Embedding với mô hình embedding GloVe đã được đào tạo sẵn.
e = keras.layers.Embedding(embeddings.shape[0], embeddings.shape[1], weights=[embeddings])

# Tạo một mô hình với lớp Embedding là lớp đầu tiên.
model = keras.Sequential([e])

# Biên dịch mô hình.
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Huấn luyện mô hình trên tập dữ liệu gồm văn bản và nhãn.
model.fit(x_train, y_train, epochs=10)

# Đánh giá mô hình trên tập dữ liệu kiểm tra.
model.evaluate(x_test, y_test)
```

*** Các ứng dụng của word embedding**

- Phân loại văn bản: Word embedding có thể được sử dụng để cải thiện độ chính xác của các mô hình phân loại văn bản. Ví dụ, một mô hình phân loại văn bản có thể sử dụng word embedding để tính toán sự tương tự giữa văn bản đầu vào và các văn bản được phân loại trước đó.
- Tìm kiếm thông tin: Word embedding có thể được sử dụng để cải thiện độ chính xác của các hệ thống tìm kiếm thông tin. Ví dụ, một hệ thống tìm kiếm thông tin có thể sử dụng word embedding để tính toán sự tương tự giữa truy vấn và các tài liệu được lưu trữ.
- Dịch ngôn ngữ: Word embedding có thể được sử dụng để cải thiện độ chính xác của các hệ thống dịch ngôn ngữ. Ví dụ, một hệ thống dịch ngôn ngữ có thể sử dụng word embedding để tìm các từ hoặc cụm từ có ý nghĩa tương tự trong hai ngôn ngữ.
- Công cụ gợi ý: Word embedding có thể được sử dụng để cải thiện độ chính xác của các công cụ gợi ý. Ví dụ, một công cụ gợi ý có thể sử dụng word embedding để đề xuất các sản phẩm hoặc dịch vụ liên quan đến các sản phẩm hoặc dịch vụ mà người dùng đã xem trước đó.
- Tự động hóa: Word embedding có thể được sử dụng để cải thiện độ chính xác của các hệ thống tự động hóa. Ví dụ, một hệ thống tự động hóa có thể sử dụng word embedding để hiểu các hướng dẫn của con người.

Ngoài ra, word embedding còn có thể được sử dụng trong các ứng dụng NLP khác, chẳng hạn như:

- Tóm tắt văn bản: Word embedding có thể được sử dụng để tạo các tóm tắt của văn bản dài.
- Tạo văn bản: Word embedding có thể được sử dụng để tạo văn bản mới.
- Trả lời câu hỏi: Word embedding có thể được sử dụng để trả lời các câu hỏi về văn bản.

*** Kết luận**

Word embedding là một kỹ thuật quan trọng trong NLP, có thể được sử dụng để thực hiện nhiều tác vụ NLP khác nhau. Các vector word embedding có thể giúp máy tính hiểu được ý nghĩa của ngôn ngữ tự nhiên, từ đó cải thiện hiệu quả của các hệ thống NLP.

*** Lưu ý khi sử dụng word embedding**

- Chuẩn hóa dữ liệu: Dữ liệu đầu vào cho lớp Embedding phải được chuẩn hóa, nghĩa là tất cả các từ phải được chuyển đổi thành một dạng nhất quán. Điều này có thể được thực hiện bằng cách sử dụng mã hóa một-hot hoặc mã hóa nhị phân.

- Kích thước của không gian vector: Kích thước của không gian vector cho embedding là một tham số quan trọng cần được xem xét. Kích thước không gian vector càng lớn thì mô hình càng có thể học được các mối quan hệ phức tạp giữa các từ. Tuy nhiên, kích thước không gian vector càng lớn thì mô hình càng cần nhiều dữ liệu để được đào tạo.
- Lựa chọn mô hình embedding: Có nhiều mô hình embedding khác nhau có sẵn, mỗi mô hình có ưu và nhược điểm riêng. Một số mô hình phổ biến bao gồm Word2Vec, GloVe và FastText.

6.2

*** Kiến thức về phân loại text trong word embedding**

Phân loại text là một nhiệm vụ NLP trong đó một mô hình được đào tạo để phân loại văn bản thành các lớp khác nhau. Ví dụ: một mô hình phân loại text có thể được đào tạo để phân loại email thành thư rác hoặc không phải thư rác, hoặc phân loại bài viết tin tức thành các chủ đề khác nhau.

*** Ứng dụng của word embedding trong phân loại text**

Word embedding có thể được sử dụng để cải thiện hiệu quả của các mô hình phân loại text bằng cách:

- Tạo các tính năng đại diện cho ý nghĩa của văn bản: Các vector word embedding có thể được sử dụng để tạo các tính năng đại diện cho ý nghĩa của văn bản. Điều này giúp cho mô hình phân loại dễ dàng học được mối quan hệ giữa các từ trong văn bản và giữa văn bản và nhãn.
- Giảm bớt sự phụ thuộc vào dữ liệu: Word embedding có thể giúp giảm bớt sự phụ thuộc vào dữ liệu. Điều này là do các vector word embedding có thể được học từ một tập dữ liệu lớn, giúp cho mô hình phân loại có thể hoạt động tốt hơn trên các dữ liệu mới.

*** Các kiến trúc mô hình phân loại text sử dụng word embedding**

Có nhiều kiến trúc mô hình phân loại text sử dụng word embedding. Một số kiến trúc phổ biến bao gồm:

- Kiến trúc 1-layer: Kiến trúc này bao gồm một lớp Embedding được tiếp theo bởi một lớp Dense. Lớp Dense được sử dụng để phân loại văn bản thành các lớp khác nhau.

- Kiến trúc nhiều layer: Kiến trúc này bao gồm nhiều lớp Embedding và Dense. Các lớp Embedding được sử dụng để tạo các tính năng từ văn bản, và các lớp Dense được sử dụng để phân loại văn bản thành các lớp khác nhau.

*** Các vấn đề cần lưu ý khi sử dụng word embedding trong phân loại text**

- Có một số vấn đề cần lưu ý khi sử dụng word embedding trong phân loại text:
- Kích thước không gian vector: Kích thước không gian vector là một tham số quan trọng cần được xem xét. Kích thước không gian vector càng lớn thì mô hình càng có thể học được các mối quan hệ phức tạp giữa các từ. Tuy nhiên, kích thước không gian vector càng lớn thì mô hình càng cần nhiều dữ liệu để được đào tạo.
- Lựa chọn mô hình embedding: Có nhiều mô hình embedding khác nhau có sẵn, mỗi mô hình có ưu và nhược điểm riêng. Cần lựa chọn mô hình embedding phù hợp với nhiệm vụ phân loại text cụ thể.
- Dữ liệu đào tạo: Dữ liệu đào tạo cần phải được chuẩn hóa và cân bằng. Điều này sẽ giúp mô hình phân loại học được các mối quan hệ chính xác giữa các từ và các lớp.

*** Kết luận**

Word embedding là một kỹ thuật mạnh mẽ có thể được sử dụng để cải thiện hiệu quả của các mô hình phân loại text. Tuy nhiên, cần lưu ý một số vấn đề khi sử dụng word embedding để đảm bảo hiệu quả và công bằng.

*** Example of Learning an Embedding**

Chúng ta sẽ định nghĩa một vấn đề nhỏ, trong đó chúng ta có 10 tài liệu văn bản, mỗi tài liệu có một nhận xét về một bài tập mà học sinh nộp. Mỗi tài liệu văn bản được phân loại là tích cực "1" hoặc tiêu cực "0".

Đầu tiên, chúng ta sẽ xác định các tài liệu và nhãn lớp của chúng.

```
# define documents
docs = ['Well done!',
 'Good work',
 'Great effort',
 'nice work',
 'Excellent!',
 'Weak',
 'Poor effort!',
 'not good',
 'poor work',
 'Could have done better.']
# define class labels
labels = array([1,1,1,1,1,0,0,0,0,0])
```

Tiếp theo, chúng ta có thể mã hóa từng tài liệu thành số nguyên. Điều này có nghĩa là lớp Embedding sẽ có các chuỗi số nguyên làm đầu vào. Chúng ta có thể thử nghiệm với các cách mã hóa mô hình túi từ (bag of word) phức tạp hơn như số lượng hoặc TF-IDF.

Keras cung cấp hàm one_hot() tạo ra một giá trị băm của mỗi từ dưới dạng mã hóa số nguyên hiệu quả. Chúng ta sẽ ước tính kích thước từ vựng là 50, lớn hơn nhiều so với cần thiết để giảm khả năng xung đột từ hàm băm.

```
# integer encode the documents
vocab_size = 50
encoded_docs = [one_hot(d, vocab_size) for d in docs]
print(encoded_docs)
```

Các chuỗi có độ dài khác nhau và Keras thích các đầu vào được vectơ hóa và tất cả các đầu vào có cùng độ dài. Chúng ta sẽ đệm tất cả các chuỗi đầu vào để có độ dài là 4. Một lần nữa, chúng ta có thể làm điều này với một hàm Keras tích hợp, trong trường hợp này là hàm pad_sequences().

```
# pad documents to a max length of 4 words
max_length = 4
padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs)
```

Bây giờ chúng ta đã sẵn sàng để định nghĩa lớp Embedding của mình như một phần của mô hình mạng thần kinh.

Lớp Embedding có 50 từ vựng và độ dài đầu vào là 4. Chúng ta sẽ chọn một không gian embedding nhỏ gồm 8 chiều.

Mô hình là một mô hình phân loại nhị phân đơn giản. Điều quan trọng là đầu ra từ lớp Embedding sẽ là 4 vector, mỗi vector có 8 chiều, một vector cho mỗi từ. Chúng ta làm phẳng nó thành một vector 32 phần tử để truyền cho lớp Dense đầu ra.

```
# define the model
model = Sequential()
model.add(Embedding(vocab_size, 8, input_length=max_length))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# summarize the model
print(model.summary())
```

Cuối cùng, chúng ta có thể huấn luyện và đánh giá mô hình phân loại.

```
# fit the model
model.fit(padded_docs, labels, epochs=50, verbose=0)
# evaluate the model
loss, accuracy = model.evaluate(padded_docs, labels, verbose=0)
print('Accuracy: %f' % (accuracy*100))
```

* Chạy code:

```
from numpy import array
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Embedding
# define documents
docs = ['Well done!',
        'Good work',
        'Great effort',
        'nice work',
        'Excellent!',
        'Weak',
        'Poor effort!',
        'not good',
        'poor work',
        'Could have done better.']
# define class labels
labels = array([1,1,1,1,1,0,0,0,0,0])
# integer encode the documents
vocab_size = 50
encoded_docs = [one_hot(d, vocab_size) for d in docs]
print(encoded_docs)
# pad documents to a max length of 4 words
max_length = 4
padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs)
# define the model
model = Sequential()
model.add(Embedding(vocab_size, 8, input_length=max_length))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# summarize the model
print(model.summary())
# fit the model
model.fit(padded_docs, labels, epochs=50, verbose=0)
# evaluate the model
loss, accuracy = model.evaluate(padded_docs, labels, verbose=0)
print('Accuracy: %f' % (accuracy*100))
```

```
[[16, 47], [37, 13], [20, 48], [25, 13], [49], [11], [48, 48], [29, 37], [48, 13], [29, 4, 47, 19]]
[[16 47  0  0]
 [37 13  0  0]
 [20 48  0  0]
 [25 13  0  0]
 [49  0  0  0]
 [11  0  0  0]
 [48 48  0  0]
 [29 37  0  0]
 [48 13  0  0]
 [29  4 47 19]]
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 4, 8)              400

 flatten (Flatten)           (None, 32)                0

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 433 (1.69 KB)
Trainable params: 433 (1.69 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Accuracy: 80.000001
```

* Chạy code với 3 kiến trúc khác nhau

```python
import numpy as np
import matplotlib.pyplot as plt
from numpy import array
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Embedding

# Define documents
docs = ['Well done!',
        'Good work',
        'Great effort',
        'nice work',
        'Excellent!',
        'Weak',
        'Poor effort!',
        'not good',
        'poor work',
        'Could have done better.']

# Define class labels
labels = array([1, 1, 1, 1, 1, 0, 0, 0, 0, 0])

# Integer encode the documents
vocab_size = 50
encoded_docs = [one_hot(d, vocab_size) for d in docs]
```

```
# Pad documents to a max length of 4 words
max_length = 4
padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')

# Create three different model architectures
models = []

# Model 1: Original model
model1 = Sequential()
model1.add(Embedding(vocab_size, 8, input_length=max_length))
model1.add(Flatten())
model1.add(Dense(1, activation='sigmoid'))
models.append(model1)

# Model 2: Add an additional dense layer
model2 = Sequential()
model2.add(Embedding(vocab_size, 8, input_length=max_length))
model2.add(Flatten())
model2.add(Dense(16, activation='relu'))  # Additional dense layer
model2.add(Dense(1, activation='sigmoid'))
models.append(model2)

# Model 3: Increase the number of neurons in the dense layer
model3 = Sequential()
model3.add(Embedding(vocab_size, 8, input_length=max_length))
model3.add(Flatten())
model3.add(Dense(32, activation='relu'))  # More neurons in the dense layer
model3.add(Dense(1, activation='sigmoid'))
models.append(model3)
```

```
# Lists to store accuracy for each model
accuracies = []

# Train and evaluate each model
for i, model in enumerate(models):
    print(f"Model {i + 1} Architecture:")
    print(model.summary())

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

    # Fit the model
    history = model.fit(padded_docs, labels, epochs=50, verbose=0)

    # Evaluate the model
    loss, accuracy = model.evaluate(padded_docs, labels, verbose=0)
    accuracies.append(accuracy * 100)
    print(f"Model {i + 1} Accuracy: {accuracy * 100:.2f}%")
    print("=" * 50)

# Plot the accuracies
plt.figure(figsize=(10, 6))
models_names = ["Model 1", "Model 2", "Model 3"]
plt.bar(models_names, accuracies)
plt.xlabel("Models")
plt.ylabel("Accuracy (%)")
plt.title("Model Comparison")
plt.ylim(0, 100)
plt.show()
```

- Mô hình 1: Mô hình gốc

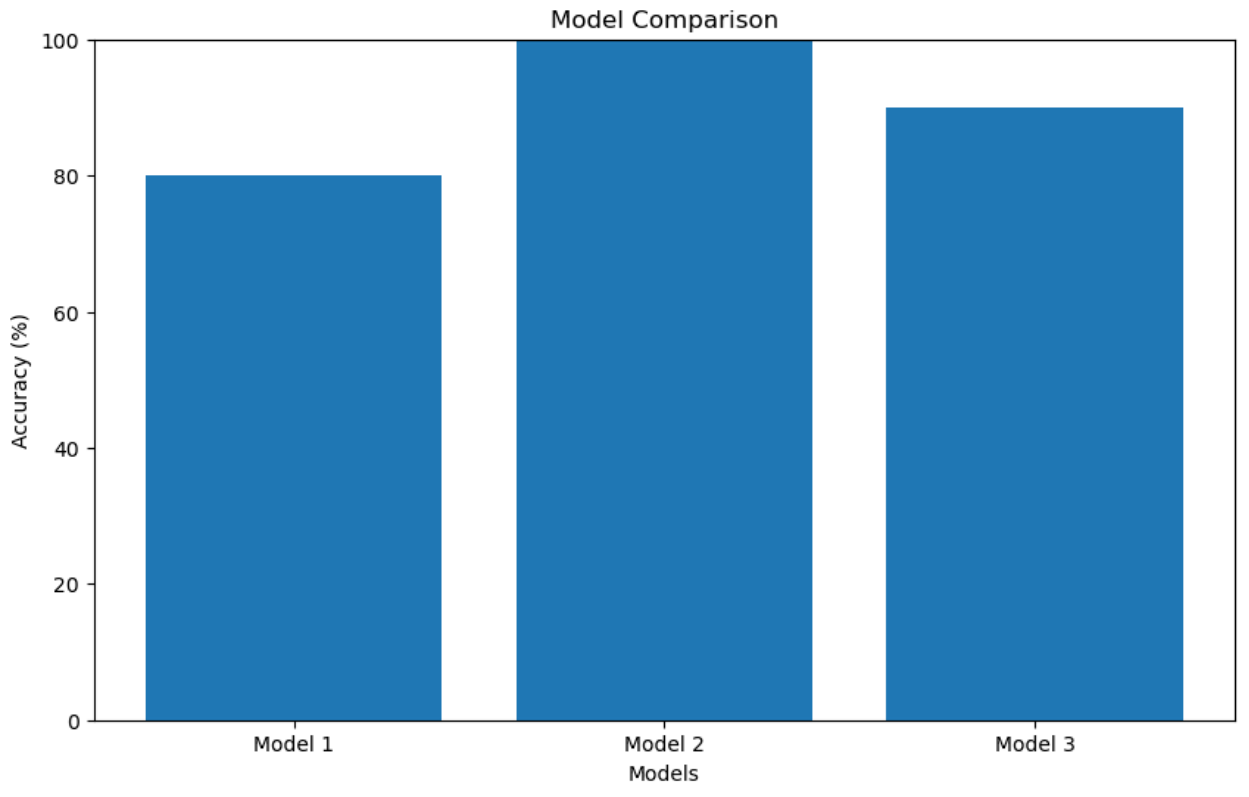
```
Model 1 Architecture:
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 4, 8)              400

 flatten_10 (Flatten)        (None, 32)                0

 dense_19 (Dense)            (None, 1)                 33

=================================================================
Total params: 433 (1.69 KB)
Trainable params: 433 (1.69 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Model 1 Accuracy: 80.00%
================================================
```

- Mô hình 2: Thêm một lớp dense bổ sung

```
Model 2 Architecture:
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_11 (Embedding)    (None, 4, 8)              400

 flatten_11 (Flatten)        (None, 32)                0

 dense_20 (Dense)            (None, 16)                528

 dense_21 (Dense)            (None, 1)                 17

=================================================================
Total params: 945 (3.69 KB)
Trainable params: 945 (3.69 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Model 2 Accuracy: 100.00%
================================================
```

- Mô hình 3: Tăng số neuron trong lớp dense

```
Model 3 Architecture:
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_12 (Embedding)    (None, 4, 8)              400

 flatten_12 (Flatten)        (None, 32)                0

 dense_22 (Dense)            (None, 32)                1056

 dense_23 (Dense)            (None, 1)                 33

=================================================================
Total params: 1489 (5.82 KB)
Trainable params: 1489 (5.82 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Model 3 Accuracy: 90.00%
================================================
```



- Nhận xét:

- Model 1: Đây là mô hình gốc với kiến trúc Embedding và một lớp Dense. Độ chính xác của mô hình này có thể thấp hơn so với các biến thể khác, đặc biệt là khi tập dữ liệu phức tạp hơn.
- Model 2: Mô hình này đã thêm một lớp Dense bổ sung với 16 đơn vị. Điều này có thể cải thiện độ chính xác của mô hình so với Model 1, đặc biệt là trên tập dữ liệu có tính phức tạp hơn.
- Model 3: Mô hình này tăng số lượng neuron trong lớp Dense lên 32. Điều này có thể tạo ra sự cải thiện đáng kể về độ chính xác so với cả Model 1 và Model 2, đặc biệt là trong các tình huống phức tạp.

Sự giảm độ chính xác của Model 3 so với Model 2 có thể do một số nguyên nhân sau:

- Overfitting: Model 3 có một lớp Dense với 32 neuron, điều này làm cho nó có khả năng học cực kỳ phức tạp và mạnh mẽ trên tập dữ liệu huấn luyện. Tuy nhiên, nếu tập dữ liệu huấn luyện không đủ lớn hoặc không đại diện cho dữ liệu thực tế, mô hình có thể dễ dàng trang bị quá nhiều cho tập dữ liệu huấn luyện (overfitting). Khi overfitting xảy ra, mô hình sẽ không tổng quát hóa tốt trên dữ liệu kiểm tra mới, dẫn đến việc độ chính xác trên tập kiểm tra thấp hơn so với Model 2.
- Sự phức tạp thừa: Sự phức tạp của Model 3 có thể không cần thiết cho tập dữ liệu cụ thể bạn đang làm việc. Dữ liệu của bạn có thể không đủ phức tạp để cung cấp sự hưởng lợi từ việc sử dụng 32 neuron trong lớp Dense. Điều này có thể dẫn đến việc mô hình bị quá phức tạp và không thể tìm ra các quy luật đơn giản để phân loại dữ liệu.
- Không đủ dữ liệu huấn luyện: Nếu tập dữ liệu huấn luyện không đủ lớn, Model 3 có thể gặp khó khăn trong việc học các đặc trưng tổng quát và có thể nói quá nhiều về nhiễu trong tập dữ liệu, dẫn đến hiện tượng overfitting.

Để giải quyết vấn đề này, bạn có thể thử những biện pháp sau:

- Giảm độ phức tạp của mô hình: Thay vì có 32 neuron trong lớp Dense, bạn có thể thử với số lượng neuron ít hơn để giảm sự phức tạp của mô hình.
- Thêm dữ liệu huấn luyện: Cố gắng có thêm dữ liệu huấn luyện nếu có thể. Dữ liệu lớn hơn có thể giúp mô hình tổng quát hóa tốt hơn.
- Sử dụng kỹ thuật regularization: Bạn có thể sử dụng kỹ thuật regularization như L1 hoặc L2 regularization để giảm nguy cơ overfitting.
- Kiểm tra với tập kiểm tra độc lập: Hãy kiểm tra độ chính xác của mô hình trên một tập dữ liệu kiểm tra độc lập để xác minh xem sự giảm độ chính xác có xuất phát từ overfitting hay không.

6.3

*** Kiến thức về Embedding Layer**

1. Embedding Layer là gì?

Lớp Embedding là một trong những lớp có sẵn trong thư viện Keras và thường được sử dụng trong các ứng dụng liên quan đến xử lý ngôn ngữ tự nhiên (NLP). Nó giúp biểu diễn từng từ trong văn bản thành các vector số thực có kích thước cố định.

2. Tại sao cần sử dụng Embeddings?

Giảm chiều dữ liệu: Trong khi xử lý dữ liệu văn bản, cần chuyển đổi từng từ thành dạng số để đưa vào mô hình. Một cách thường được sử dụng là one-hot encoding, tạo ra một vector có độ dài bằng số từ vựng, với hầu hết các giá trị là 0. Điều này tạo ra một ma trận lớn và thưa thớt, gây lãng phí bộ nhớ và giảm hiệu suất mô hình.

Tích hợp thông tin từ các từ liên quan: Embedding Layer giúp biểu diễn từng từ dưới dạng các vector có giá trị thực, cho phép mô hình học được mối quan hệ và sự tương quan giữa các từ trong không gian vector.

3. Cách hoạt động của Embedding Layer

Embedding Layer hoạt động như một bảng tra cứu (lookup table), trong đó các từ là các khóa (keys), và các vector từ tương ứng là các giá trị (values). Cụ thể:

- Mỗi từ được biểu diễn bởi một số nguyên duy nhất (index) trong từ điển từ vựng.
- Embedding Layer sử dụng index của từ để tìm vector từ tương ứng trong ma trận trọng số (embedding matrix).
- Vector từ này được sử dụng như một biểu diễn số học của từ trong không gian vector.

*** Restaurant Review Classification**

Trong phần này, chúng ta sẽ thực hiện các bước sau để giải quyết vấn đề phân loại đánh giá nhà hàng:

1. Chia các câu thành các từ riêng biệt
2. Tạo vector mã hóa one-hot cho từng từ
3. Sử dụng padding để đảm bảo tất cả các chuỗi có cùng độ dài
4. Đưa các chuỗi mã hóa được đệm vào lớp nhúng (Embedding Layer)
5. Duỗi và áp dụng lớp Dense để dự đoán nhãn

Chúng ta bắt đầu bằng việc nhập các thư viện cần thiết

```
from numpy import array
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten,Embedding,Dense
```

Để làm cho ví dụ đơn giản, chúng ta sẽ sử dụng tổng cộng 10 đánh giá. Một nửa trong số chúng là tích cực, được biểu thị bằng 0, và nửa còn lại là tiêu cực, được biểu thị bằng 1.

```
# Define 10 restaurant reviews
reviews =[
          'Never coming back!',
          'horrible service',
          'rude waitress',
          'cold food',
          'horrible food!',
          'awesome',
          'awesome services!',
          'rocks',
          'poor work',
          'couldn\'t have done better'
]

#Define labels
labels = array([1,1,1,1,1,0,0,0,0,0])
```

Chúng ta sẽ sử dụng kích thước từ vựng là 50 và mã hóa one-hot cho các từ bằng cách sử dụng hàm one_hot từ thư viện Keras.

```
Vocab_size = 50
encoded_reviews = [one_hot(d,Vocab_size) for d in reviews]
print(f'encoded reviews: {encoded_reviews}')
```

Bây giờ chúng ta cần áp dụng padding để đảm bảo tất cả các đánh giá được mã hóa có cùng độ dài. Hãy xác định 4 là độ dài tối đa và thêm các vector mã hóa bằng giá trị 0 vào cuối.

```
max_length = 4
padded_reviews =
pad_sequences(encoded_reviews,maxlen=max_length,padding='post')
print(padded_reviews)
```

Chúng ta tạo một mô hình Keras đơn giản. Chúng ta sẽ xác định độ dài của các vector nhúng cho mỗi từ là 8 và độ dài đầu vào sẽ là độ dài tối đa mà chúng ta đã xác định trước đó là 4.

```
model = Sequential()
embedding_layer =
Embedding(input_dim=Vocab_size,output_dim=8,input_length=max_length)
model.add(embedding_layer)
model.add(Flatten())
model.add(Dense(1,activation='sigmoid'))
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=
['acc'])

print(model.summary())
```

Tiếp theo, chúng ta sẽ huấn luyện mô hình trong 100 epochs (vòng lặp). Điều này có nghĩa rằng chúng ta sẽ đào tạo mô hình trên dữ liệu đào tạo 100 lần, cố gắng cải thiện mô hình theo thời gian.

```
model.fit(padded_reviews,labels,epochs=100,verbose=0)
```

Khi việc huấn luyện hoàn thành, lớp nhúng (embedding layer) đã học được các trọng số, chúng chính là các biểu diễn vector của từng từ. Hãy kiểm tra kích thước của ma trận trọng số.

```
print(embedding_layer.get_weights()[0].shape)
```

Vậy đó là cách chúng ta huấn luyện một lớp nhúng (embedding layer) trên ngữ liệu văn bản của chúng ta và thu được các vector nhúng cho từng từ. Sau đó, các vector này được sử dụng để biểu diễn các từ trong một câu. Lớp nhúng giúp chúng ta chuyển đổi từng từ thành các biểu diễn số thực có chiều dài cố định, giúp mô hình học cách hiểu và làm việc với dữ liệu văn bản một cách hiệu quả hơn.

* Chạy code

```python
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Flatten, Dense

# Define 10 restaurant reviews
reviews = [
    'Never coming back!',
    'horrible service',
    'rude waitress',
    'cold food',
    'horrible food!',
    'awesome',
    'awesome services!',
    'rocks',
    'poor work',
    "couldn't have done better"
]

# Define labels
labels = np.array([1, 1, 1, 1, 1, 0, 0, 0, 0, 0])

# Create a tokenizer
tokenizer = Tokenizer(num_words=50)
tokenizer.fit_on_texts(reviews)

# Convert text to sequences
sequences = tokenizer.texts_to_sequences(reviews)

# Pad sequences
max_length = 4
padded_reviews = pad_sequences(sequences, maxlen=max_length, padding='post')

# Create the model
model = Sequential()
model.add(Embedding(input_dim=50, output_dim=8, input_length=max_length))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

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

# Print model summary
print(model.summary())

# Train the model
model.fit(padded_reviews, labels, epochs=100, verbose=0)

# Check the shape of the embedding weights
print(model.layers[0].get_weights()[0].shape)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 4, 8)              400

 flatten (Flatten)           (None, 32)                0

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 433 (1.69 KB)
Trainable params: 433 (1.69 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
(50, 8)
```

* Chạy code với 3 kiến trúc khác nhau

- Import thư viện:

```
from numpy import array
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Embedding, Dense
```

- Phân loại text:

```
# Define 10 restaurant reviews
reviews =[
          'Never coming back!',
          'horrible service',
          'rude waitress',
          'cold food',
          'horrible food!',
          'awesome',
          'awesome services!',
          'rocks',
          'poor work',
          'couldn\'t have done better'
]
#Define labels
labels = array([0,0,0,1,0,1,1,1,0,1])

def tokenize_and_pad(docs, vocab_size=50):
    tokenized_docs = [one_hot(d, vocab_size) for d in docs]
    max_len = max([len(doc) for doc in tokenized_docs])
    padded_docs = pad_sequences(tokenized_docs, maxlen=max_len, padding='post')

    return padded_docs

vocab_size = 50
X = tokenize_and_pad(reviews, vocab_size)
y = labels
```

- Tạo chuỗi lưu models và accuracies:

```
# Create three different model architectures
models = []

# Lists to store accuracy for each model
accuracies = []
```

- Mô hình 1: Embedding -> Dense 1 neuron -> Sigmoid Activation. Hàm mục tiêu: Binary Cross Entropy

```
# Model 1
model1 = Sequential()
model1.add(Embedding(vocab_size, 8, input_length=X.shape[-1]))
model1.add(Flatten())
model1.add(Dense(1, activation='sigmoid'))

model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
print(model1.summary())

model1.fit(X, y, epochs=20, verbose=0)

loss, accuracy = model1.evaluate(X, y, verbose=0)
print('Accuracy: %f' % (accuracy*100))

models.append(model1)
accuracies.append(accuracy * 100)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 4, 8)              400

 flatten (Flatten)           (None, 32)                0

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 433 (1.69 KB)
Trainable params: 433 (1.69 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Accuracy: 89.999998
```

- Mô hình 2: Embedding -> Dense 2 neuron -> Softmax Activation. Hàm mục tiêu: Cross Entropy Loss

```python
#Model 2
model2 = Sequential()
model2.add(Embedding(vocab_size, 8, input_length=X.shape[-1]))
model2.add(Flatten())
model2.add(Dense(2, activation='softmax'))

model2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
print(model2.summary())

model2.fit(X, y, epochs=20, verbose=0)

loss, accuracy = model2.evaluate(X, y, verbose=0)
print('Accuracy: %f' % (accuracy*100))

models.append(model2)
accuracies.append(accuracy * 100)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 4, 8)              400

 flatten_1 (Flatten)         (None, 32)                0

 dense_1 (Dense)             (None, 2)                 66

=================================================================
Total params: 466 (1.82 KB)
Trainable params: 466 (1.82 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Accuracy: 100.000000
```

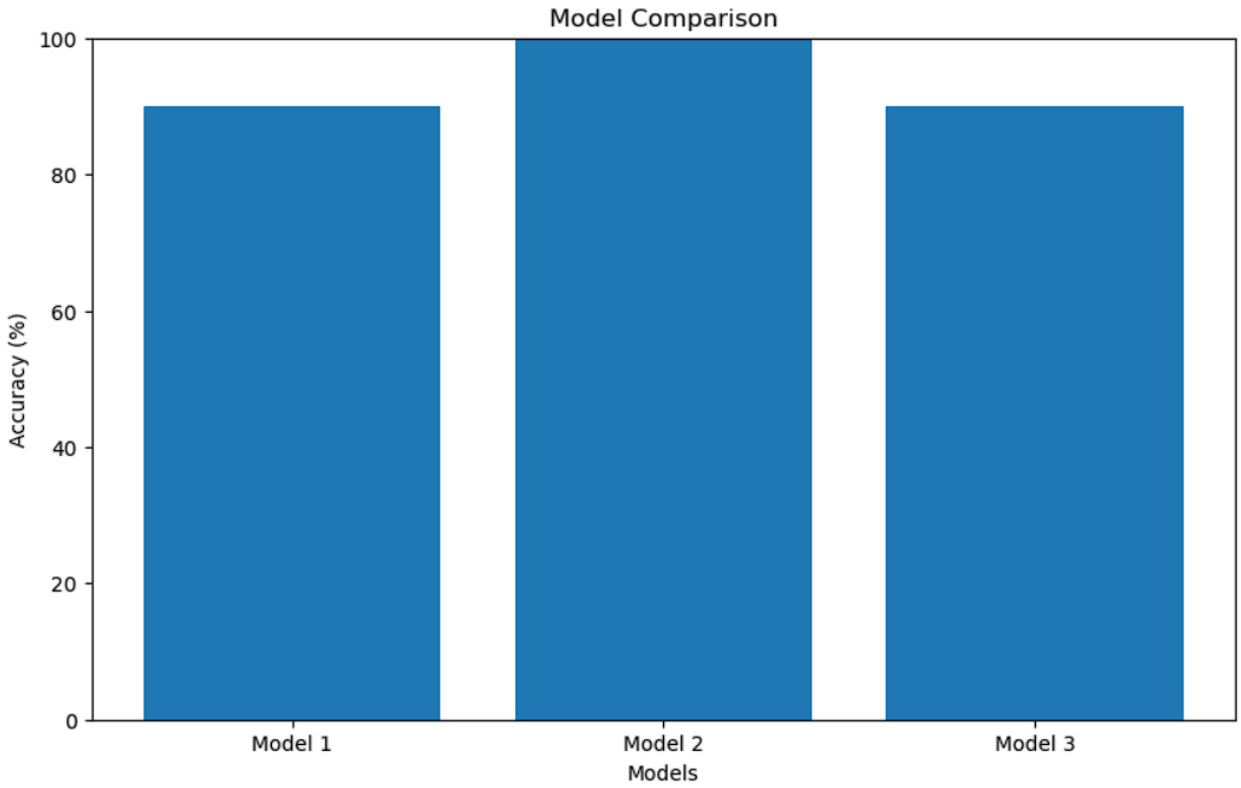
- Mô hình 3: Embedding -> Dense 4 neuron -> ReLU activation -> Dense 2 neuron -> Softmax Activattion. Hàm mục tiêu: Cross Entropy Loss

```python
# Model 3
model3 = Sequential()
model3.add(Embedding(vocab_size, 8, input_length=X.shape[-1]))
model3.add(Flatten())
model3.add(Dense(4, activation='relu'))
model3.add(Dense(2, activation='softmax'))

model3.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
print(model3.summary())

model3.fit(X, y, epochs=20, verbose=0)

loss, accuracy = model3.evaluate(X, y, verbose=0)
print('Accuracy: %f' % (accuracy*100))

models.append(model3)
accuracies.append(accuracy * 100)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 4, 8)              400

 flatten_2 (Flatten)         (None, 32)                0

 dense_2 (Dense)             (None, 4)                 132

 dense_3 (Dense)             (None, 2)                 10

=================================================================
Total params: 542 (2.12 KB)
Trainable params: 542 (2.12 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Accuracy: 89.999998
```

- Vẽ biểu đồ:

```python
# Plot the accuracies
plt.figure(figsize=(10, 6))
models_names = ["Model 1", "Model 2", "Model 3"]
plt.bar(models_names, accuracies)
plt.xlabel("Models")
plt.ylabel("Accuracy (%)")
plt.title("Model Comparison")
plt.ylim(0, 100)
plt.show()
```



- Nhận xét:

- Model 1 có độ chính xác 89.9%. Đây là một độ chính xác tốt, nhưng cần lưu ý rằng có dấu hiệu overfitting, điều này có nghĩa là mô hình có thể đã học cụ thể cho tập dữ liệu huấn luyện và không hoạt động tốt trên dữ liệu mới.
- Model 2 có độ chính xác 100%, điều này khá ấn tượng. Tuy nhiên, sự hoàn hảo này có thể là một dấu hiệu của overfitting. Nó cũng có thể do mô hình chưa được đánh giá trên dữ liệu kiểm tra hoặc dữ liệu kiểm tra rất giống với dữ liệu huấn luyện.
- Model 3 có độ chính xác 89.9%. Điều này tương tự như Model 1, và cũng có dấu hiệu overfitting.

Dựa trên những điểm này, có vẻ như Model 2 có thể đã học rất tốt dữ liệu huấn luyện, nhưng cần kiểm tra kỹ hơn để xem liệu nó hoạt động tốt trên dữ liệu kiểm tra và có bị overfitting không. Còn Model 1 và Model 3 cần được điều chỉnh để giảm overfitting và cải thiện độ chính xác trên dữ liệu mới.