Họ tên: Trần Vân Anh

MSV: B20DCCN075

1. Word Embedding là gì? Trình bày hiểu biết của mình và các ứng dụng của word embedding

1.1. Word Embedding là gì?

Word Embedding là một kỹ thuật quan trọng trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP) được sử dụng để biểu diễn từ vựng của ngôn ngữ dưới dạng các vector số học trong không gian đa chiều. Kỹ thuật này cho phép máy tính hiểu và làm việc với từ vựng ngôn ngữ một cách hiệu quả hơn bằng cách biểu diễn chúng dưới dạng số học.

1.2. Cách Hoạt Động của Word Embedding

Cách Word Embedding hoạt động dựa trên nguyên tắc rằng các từ có ý nghĩa tương tự thường xuất hiện gần nhau trong văn bản. Các mô hình Word Embedding học từ cách từ hoặc cụm từ xuất hiện trong ngữ cảnh của những từ khác. Cụ thể:

- Word2Vec: Word2Vec sử dụng mô hình skip-gram hoặc Continuous Bag of Words (CBOW) để dự đoán từ cục bộ (local context) của một từ. Nếu hai từ có ngữ cảnh gần giống nhau, thì chúng sẽ có vector biểu diễn gần nhau trong không gian vector.

- GloVe: GloVe kết hợp thông tin cục bộ và toàn cục. Nó sử dụng ma trận xác suất xuất hiện của các cặp từ trong văn bản để tạo ra các vector từ.

- FastText: FastText là biến thể của Word2Vec cho phép xử lý các n-gram của từ, giúp xử lý từ vựng một cách linh hoạt hơn.

2.1. Xử Lý Ngôn Ngữ Tự Nhiên (NLP)

Word Embedding là trụ cột của NLP và đã thay đổi cách chúng ta xử lý ngôn ngữ tự nhiên. Dưới đây là một số ứng dụng quan trọng:

- Phân loại văn bản: Word Embedding cho phép máy tính phân loại các văn bản vào các lớp khác nhau, như phân loại email spam hoặc tin tức.

- Dịch máy: Biểu diễn từ vựng dưới dạng vector giúp cải thiện chất lượng dịch máy.

- Phân tích tình cảm: Word Embedding được sử dụng để phân tích cảm xúc trong văn bản, như đoán xem một đoạn văn có tính tích cực hay tiêu cực.

- Tóm tắt văn bản: Giúp tạo ra tóm tắt tự động từ các đoạn văn.

2.2. Hệ Thống Gợi Ý

Word Embedding cung cấp khả năng hiểu sâu hơn về sở thích của người dùng. Điều này hữu ích trong:

- Hệ thống gợi ý sản phẩm: Cho phép gợi ý sản phẩm tùy chỉnh cho người dùng dựa trên lịch sử mua sắm và sở thích cá nhân.

- Gợi ý nội dung trực tuyến: Cung cấp nội dung tương tự hoặc liên quan đến nội dung người dùng đang xem.

3.1. Tiềm Năng Mở Rộng

Word Embedding không chỉ giới hạn trong tiếng Anh. Nó có thể mở rộng để áp dụng cho nhiều ngôn ngữ khác nhau, giúp cải thiện xử lý ngôn ngữ tự nhiên toàn cầu.

3.2. Kết Hợp Với Học Sâu

Word Embedding là một phần quan trọng của học sâu (Deep Learning), giúp tạo ra mô hình mạng neural mạnh mẽ trong NLP như BERT, GPT, và Transformer.

3.3. Ứng Dụng Mở Rộng

Word Embedding có tiềm năng trong các lĩnh vực khác nhau như:

- Xử lý hình ảnh: Biểu diễn ảnh dưới dạng vector, giúp kết hợp thông tin hình ảnh và văn bản.

- Xử lý âm thanh: Cải thiện xử lý và hiểu âm thanh trong ứng dụng như nhận diện giọng nói.

- Khoa học dữ liệu và tài chính: Sử dụng Word Embedding để phân tích và dự đoán dữ liệu thị trường và tài chính.

4.1. Thách Thức Trong Word Embedding

Mặc dù Word Embedding có nhiều ứng dụng mạnh mẽ, nhưng cũng đối mặt với một số thách thức, bao gồm:

- Hiệu suất tính toán: Để tạo ra các vector từ, cần đối mặt với việc tính toán trên các tập dữ liệu lớn, đòi hỏi sự tập trung vào hiệu suất tính toán.

- Ý nghĩa ngữ cảnh: Word Embedding không phân biệt ý nghĩa ngữ cảnh. Ví dụ, cùng một từ có thể có nhiều ý nghĩa khác nhau trong các ngữ cảnh khác nhau.

4.2. Các Kỹ Thuật Liên Quan

Cùng với Word Embedding, có một số kỹ thuật liên quan:

- Doc2Vec: Tương tự Word2Vec nhưng áp dụng cho văn bản hoàn chỉnh thay vì từng từ.

- BERT (Bidirectional Encoder Representations from Transformers): Một mô hình học sâu tiên tiến trong NLP, kết hợp Word Embedding với kiến thức về ngữ cảnh từ cả hai hướng.

5.1. Tóm Lại

Word Embedding đã thay đổi cách chúng ta làm việc với ngôn ngữ tự nhiên và đã có ứng dụng rộng rãi trong NLP và nhiều lĩnh vực khác. Nó cung cấp khả năng biểu diễn từ vựng một cách hiệu quả, giúp máy tính hiểu và tương tác với ngôn ngữ tự nhiên một cách thông minh.

5.2. Triển Vọng Tương Lai

Triển vọng của Word Embedding là sự mở rộng và tích hợp với các lĩnh vực khác nhau, từ xử lý hình ảnh và âm thanh đến khoa học dữ liệu và tài chính. Sự tiến bộ trong học sâu và NLP sẽ tiếp tục tạo ra những ứng dụng mới và mạnh mẽ của Word Embedding trong tương lai.

2.Áp dụng cho phân loại text: Trình bày kiến thức (3 trang) và code, thêm phần show như Bài tập 5, chạy code với 3 kiến trúc khác nhau (thêm layer, neuron) và nêu nhận xét outputs trên Biểu đồ

2.1.Kiến thức

Phân loại văn bản mô tả một loại vấn đề chung như dự đoán cảm xúc của các dòng tweet và bài đánh giá phim cũng như phân loại email có phải là thư rác hay không.

Phương thức hoạt động để phân loại văn bản liên quan đến việc sử dụng tính năng nhúng từ để biểu thị các từ và Mạng thần kinh chuyển đổi (CNN) để tìm hiểu cách phân biệt tài liệu về các vấn đề phân loại.

Yoav Goldberg, trong cuốn sách sơ lược về học sâu để xử lý ngôn ngữ tự nhiên, nhận xét rằng mạng lưới thần kinh nói chung mang lại hiệu suất tốt hơn so với các bộ phân loại tuyến tính cổ điển, đặc biệt là khi được sử dụng với các từ nhúng được đào tạo trước.

Ông cũng nhận xét rằng mạng nơ ron tích chập có hiệu quả trong việc phân loại tài liệu, cụ thể là vì chúng có thể chọn ra các tính năng nổi bật (ví dụ: mã thông báo hoặc chuỗi mã thông báo) theo cách bất biến đối với vị trí của chúng trong chuỗi đầu vào. Yoav Goldberg nhấn mạnh vai trò của CNN như một mô hình trích xuất đặc điểm.

Nhúng từ + CNN = Phân loại văn bản

Kiến trúc bao gồm ba phần chính:

- Nhúng từ : Một cách biểu diễn phân tán của các từ trong đó các từ khác nhau có nghĩa tương tự (dựa trên cách sử dụng của chúng) cũng có cách biểu diễn tương tự.
- Mô hình tích chập : Một mô hình trích xuất tính năng học cách trích xuất các tính năng nổi bật từ các tài liệu được biểu diễn bằng cách nhúng từ.
- Mô hình được kết nối đầy đủ : Việc giải thích các tính năng được trích xuất dưới dạng đầu ra dự đoán.

2.2. 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
```

2.3. 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

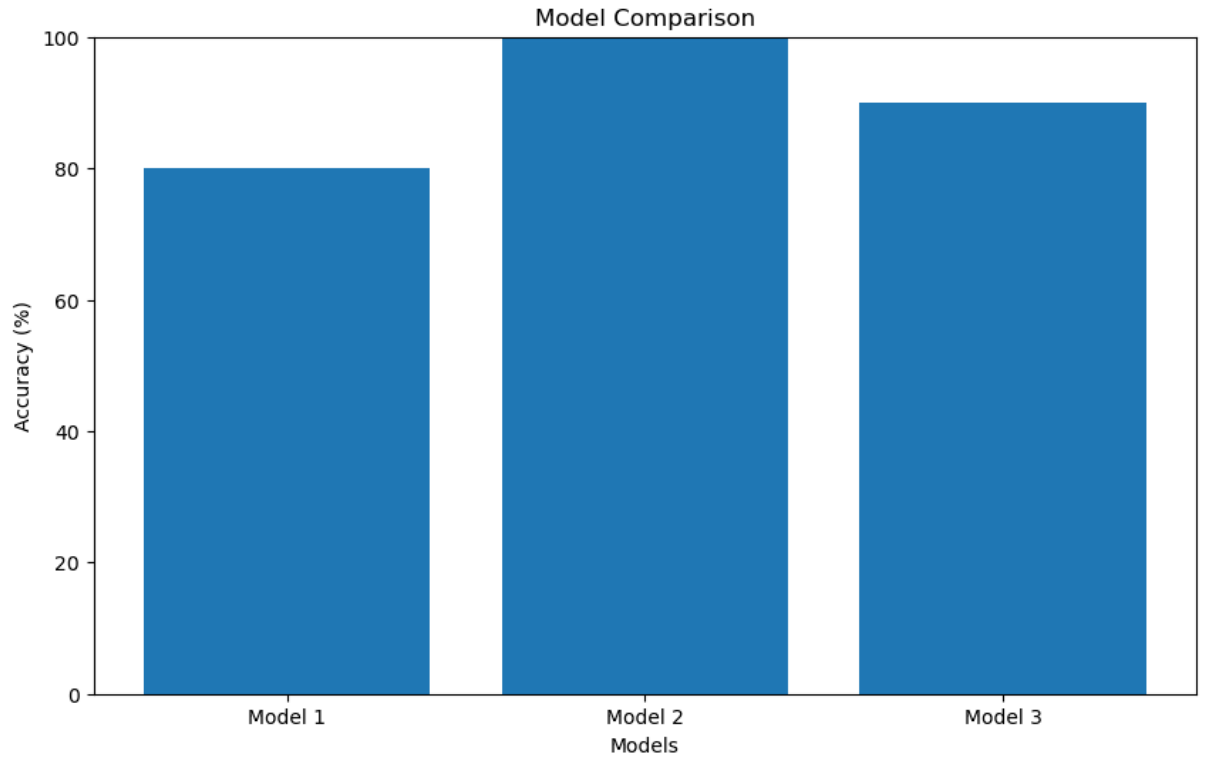
```
Model 1 Architecture:
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_10 (Embedding)    (None, 4, 8)              400

 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 hinh 2: Thêm 1 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.

1. **Áp dụng cho phân loại review restaurant: Trình bày kiến thức (3 trang) và code, thêm phần show như Bài tập 5, chạy code với 3 kiến trúc khác nhau (thêm layer, neuron) và nêu nhận xét outputs trên Biểu đồ**

3.1. Kiến thức

Trong học sâu, lớp nhúng nghe có vẻ bí ẩn cho đến khi bạn hiểu rõ về nó. Vì lớp nhúng là một phần thiết yếu của mạng lưới thần kinh nên điều quan trọng là phải hiểu hoạt động của nó. Trong bài viết này, tôi sẽ cố gắng giải thích lớp nhúng là gì, sự cần thiết của nó và cách thức hoạt động của nó, cùng với một số ví dụ về mã hóa. Vậy hãy bắt đầu.

Lớp nhúng là gì

Lớp nhúng là một trong những lớp có sẵn trong Keras. Điều này chủ yếu được sử dụng trong các ứng dụng liên quan đến Xử lý ngôn ngữ tự nhiên như mô hình hóa ngôn ngữ, nhưng nó cũng có thể được sử dụng với các tác vụ khác liên quan đến mạng lưới thần kinh. Trong khi giải quyết các vấn đề về NLP, chúng ta có thể sử dụng các phần nhúng từ được đào tạo trước như GloVe. Ngoài ra, chúng ta cũng có thể huấn luyện các phần nhúng của riêng mình bằng cách sử dụng lớp nhúng Keras.

Cần nhúng

Việc nhúng từ có thể được coi là một giải pháp thay thế cho mã hóa một lần cùng với việc giảm kích thước. Như chúng ta biết khi xử lý dữ liệu văn bản, chúng ta cần chuyển đổi nó thành số trước khi đưa vào bất kỳ mô hình học máy nào, bao gồm cả mạng thần kinh. Để đơn giản, các từ có thể được so sánh với các biến phân loại. Chúng tôi sử dụng mã hóa một lần để chuyển đổi các đặc điểm phân loại thành số. Để làm như vậy, chúng tôi tạo các đối tượng địa lý giả cho từng danh mục và điền vào chúng các số 0 và 1.

Tương tự, nếu chúng ta sử dụng mã hóa one-hot cho các từ trong dữ liệu văn bản, chúng ta sẽ có một đặc điểm giả cho mỗi từ, nghĩa là có 10.000 đặc điểm cho vốn từ vựng 10.000 từ. Đây không phải là phương pháp nhúng khả thi vì nó đòi hỏi không gian lưu trữ lớn cho các vectơ từ và làm giảm hiệu quả của mô hình. Lớp nhúng cho phép chúng ta chuyển đổi từng từ thành một vectơ có độ dài cố định có kích thước xác định. Vectơ kết quả là một vectơ dày đặc có giá trị thực thay vì chỉ 0 và 1.

Độ dài cố định của vectơ từ giúp chúng ta biểu diễn các từ theo cách tốt hơn cùng với kích thước được giảm bớt. Bằng cách này, lớp nhúng hoạt động giống như một bảng tra cứu. Các từ là các khóa trong bảng này, trong khi các vectơ từ dày đặc là các giá trị. Để hiểu rõ hơn, chúng ta hãy xem cách triển khai lớp Keras Embedding

3.2. 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)
```

3.3. Chạy code với 3 kiến trúc khác nhau

-Import thư viện và phân loại text:

```
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
```

```
# 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.

```
# 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

```
#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
```

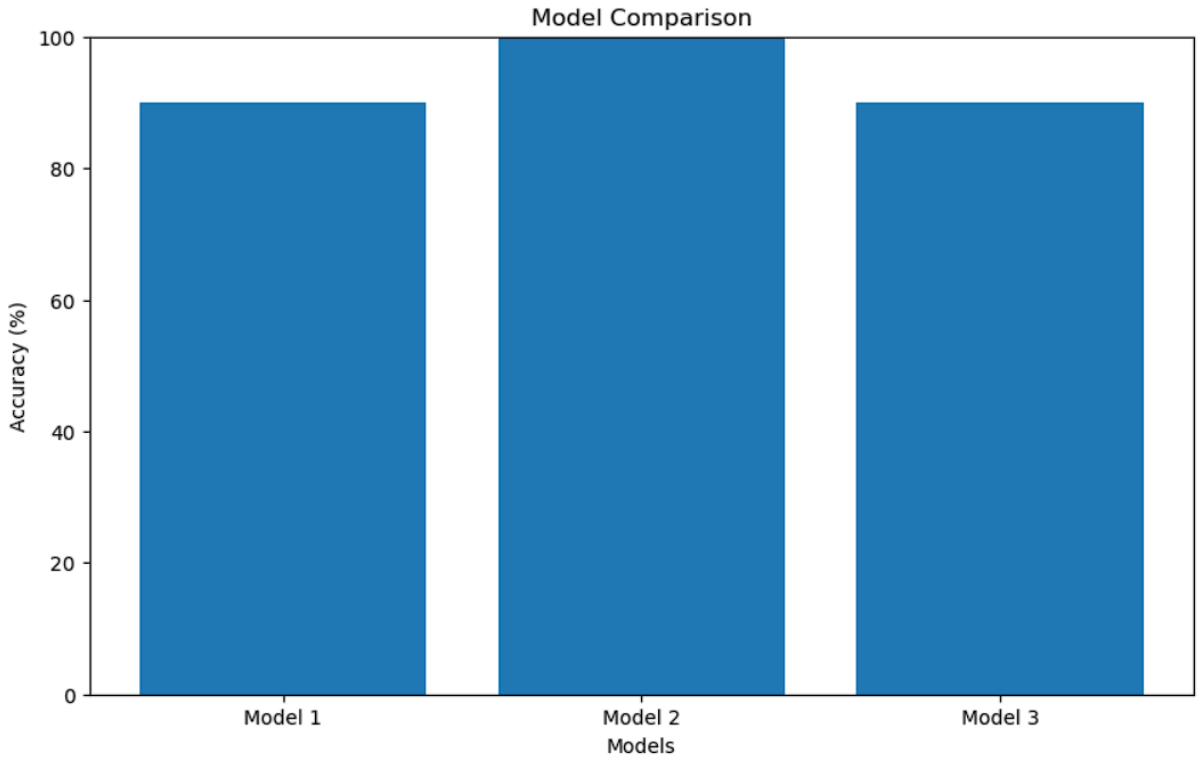
-Mô hình 3: Embedding -> Dense 4 neuron -> ReLU activation -> Dense 2 neuron -> Softmax Activattion

```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
```

-Biểu đồ so sánh:

```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 và Model 2 có độ chính xác tương tự nhau, lần lượt là 89.9% và 100%. Tuy nhiên, cả hai mô hình đều có dấu hiệu overfitting.

-Overfitting là một vấn đề phổ biến trong học máy, xảy ra khi mô hình học quá nhiều thông tin từ dữ liệu huấn luyện và trở nên quá phức tạp. Điều này có thể dẫn đến việc mô hình hoạt động kém hiệu quả trên dữ liệu mới.

-Để giảm overfitting, có thể thực hiện các biện pháp sau:

+Sử dụng kỹ thuật cross-validation để đánh giá mô hình trên dữ liệu chưa thấy trước.

+Giảm số lượng tham số của mô hình.

+Sử dụng kỹ thuật regularization để hạn chế sự phức tạp của mô hình.