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

– "Word embedding" trong xử lý ngôn ngữ tự nhiên (NLP) là một thuật ngữ quan trọng được sử dụng để biểu diễn từ bằng các vector có giá trị thực. Đây là một tiến bộ quan trọng trong lĩnh vực NLP đã cải thiện khả năng của máy tính hiểu nội dung dựa trên văn bản một cách tốt hơn. Nó được xem xét là một trong những bước tiến quan trọng nhất trong deep learning để giải quyết các vấn đề khó khăn trong xử lý ngôn ngữ tự nhiên.

+ Định nghĩa Word Embedding:

• Word Embedding là một cách tiếp cận việc biểu diễn những từ hoặc văn bản. Nó được định nghĩa là một vector số học đầu vào biểu diễn một từ trong không gian có số chiều thấp hơn. Nó cho phép các từ có ý nghĩa tương tự có một biểu diễn tương tự.

+ Mục tiêu của Word Embedding:

• Giảm chiều dữ liệu

• Sử dụng một từ để dự đoán các từ xung quanh nó

• Phải nắm bắt được ngữ nghĩa giữa các từ

+ Cách sử dụng Word Embedding:

• Chúng được sử dụng như đầu vào cho các mô hình học máy.

• Lấy các từ → Chuyển thành biểu diễn số học → Sử dụng trong quá trình huấn luyện hoặc dự đoán.

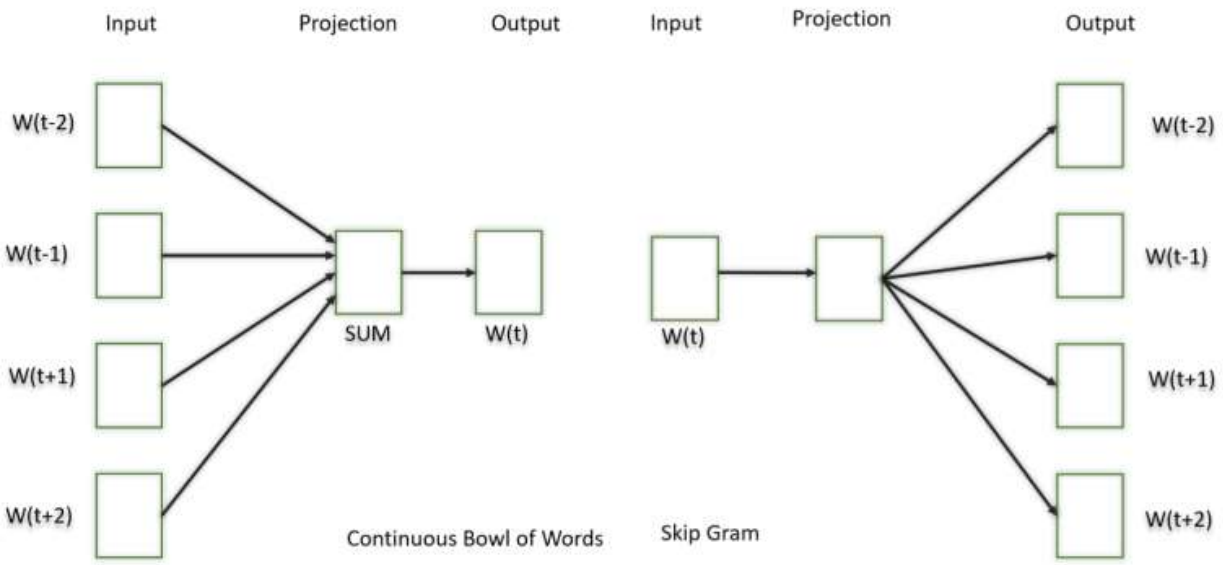
• Để biểu diễn hoặc trực quan hóa các dữ liệu trong tập văn bản đã được sử dụng để huấn luyện chúng.

+ Một số thuật toán Embedding:

1. Word2Vec:

- Word2Vec là một trong những thuật toán Word Embedding phổ biến nhất. Nó sử dụng mô hình neural network đơn giản để học biểu diễn từ, đặc biệt là Continuous Bag of Words (CBOW) và Skip-gram.

- CBOW cố gắng dự đoán từ tiền ngữ dựa trên ngữ cảnh xung quanh và Skip-gram cố gắng dự đoán ngữ cảnh dựa trên từ tiền ngữ. Cả hai đều tạo ra các biểu diễn Word Embedding rất mạnh.



2. GloVe (Global Vectors for Word Representation):

- GloVe là một phương pháp khác để học Word Embedding. Nó sử dụng ma trận thống kê xuất hiện của các từ trong văn bản để học các biểu diễn từ.
- GloVe tập trung vào việc học thông tin về tần suất xuất hiện của các từ và xây dựng biểu diễn từ dựa trên mối quan hệ tần số xuất hiện này.

In [5]:
```
import gensim.downloader as api

# Tải mô hình GloVe từ Gensim
glove_model = api.load("glove-wiki-gigaword-100")

# Truy cập biểu diễn của từ 'king'
word_vector = glove_model['king']

# Tìm các từ tương tự với 'king'
similar_words = glove_model.most_similar('king', topn=5)

# In biểu diễn của từ 'king' và các từ tương tự
print("Biểu diễn của từ 'king':\n", word_vector)
print("Các từ tương tự:", similar_words)
```

```
Biểu diễn của từ 'king':
 [-0.32307  -0.87616   0.21977   0.25268   0.22976   0.7388   -0.37954
 -0.35307  -0.84369  -1.1113   -0.30266   0.33178  -0.25113   0.30448
 -0.077491 -0.89815   0.092496 -1.1407   -0.58324   0.66869  -0.23122
 -0.95855   0.28262  -0.078848  0.75315   0.26584   0.3422   -0.33949
  0.95608   0.065641  0.45747   0.39835   0.57965   0.39267  -0.21851
  0.58795  -0.55999   0.63368  -0.043983 -0.68731  -0.37841   0.38026
  0.61641  -0.88269  -0.12346  -0.37928  -0.38318   0.23868   0.6685
 -0.43321  -0.11065   0.081723  1.1569    0.78958  -0.21223  -2.3211
 -0.67806   0.44561   0.65707   0.1045    0.46217   0.19912   0.25802
  0.057194  0.53443  -0.43133  -0.34311   0.59789  -0.58417   0.068995
  0.23944  -0.85181   0.30379  -0.34177  -0.25746  -0.031101 -0.16285
  0.45169  -0.91627   0.64521   0.73281  -0.22752   0.30226   0.044801
 -0.83741   0.55006  -0.52506  -1.7357    0.4751   -0.70487   0.056939
 -0.7132    0.089623  0.41394  -1.3363   -0.61915  -0.33089  -0.52881
  0.16483  -0.98878 ]
Các từ tương tự: [('prince', 0.7682329416275024), ('queen', 0.7507690787315369), ('son', 0.7020888328552246), ('brother', 0.698
5775828361511), ('monarch', 0.6977889537811279)]
```

+ Lợi ích của Word Embeddings:

  • Nó nhanh hơn rất nhiều trong quá trình huấn luyện so với việc xây dựng thủ công các mô hình như WordNet (sử dụng các lớp nhúng đồ thị).

  • Hầu hết tất cả các ứng dụng NLP hiện đại đều bắt đầu bằng một lớp nhúng (embedding layer).

  • Nó lưu trữ một xấp xỉ về ý nghĩa của từ.

+ Nhược điểm của Word Embeddings:

  • Nó có thể tiêu tốn nhiều bộ nhớ.

  • Nó phụ thuộc vào ngữ liệu văn bản nguồn. Bất kỳ thiên kiến cơ

  bản nào sẽ ảnh hưởng đến mô hình.

  • Nó không thể phân biệt giữa các từ đồng âm như "brake/break,""cell/sell," "weather/whether" , …

- Các ứng dụng quan trọng của word embedding:

  + Word embedding tạo ra các vector từ mà dựa vào đó ta có thể áp

  dụng chúng để thực hiện các thao tác về ngữ nghĩa như là tìm từ đồng

  nghĩa, trái nghĩa,…

+ Ngoài ra, chúng cũng là nguồn tài nguyên cho các hệ thống Machine Learning, Deep Learning nhằm thực hiện các mục đích cao hơn như là các hệ thống máy dịch, phân tích cảm xúc dựa trên ngôn từ, …

– Tiềm năng và thách thức của Word embedding

+ Tiềm năng:

- Nâng cao hiệu suất của mô hình NLP: Word embedding giúp cải thiện hiệu suất của các mô hình NLP bằng cách biểu diễn từng từ dưới dạng vectơ số học. Điều này cho phép mô hình hiểu được mối quan hệ giữa các từ và thực hiện các tác vụ như phân loại văn bản, dự đoán từ tiếp theo và dịch máy tốt hơn.
- Giảm chiều dữ liệu: Word embedding giúp giảm chiều dữ liệu, làm cho việc xử lý dữ liệu văn bản trở nên hiệu quả hơn. Thay vì sử dụng one-hot encoding, bạn có thể sử dụng biểu diễn vectơ có kích thước thấp hơn để đại diện cho từ.
- Khái quát hóa thông tin từ dữ liệu lớn: Word embedding có thể được huấn luyện trên dữ liệu lớn, do đó chúng có khả năng khái quát hóa thông tin từ tập dữ liệu lớn này. Điều này có nghĩa là bạn có thể sử dụng mô hình đã huấn luyện trên dữ liệu tự nhiên để thực hiện nhiều tác vụ khác nhau mà không cần phải tạo lại word embedding.

+ Thách thức:

- Khái quát hóa hạn chế: Mặc dù word embedding có khả năng khái quát hóa, nhưng chúng có thể không hoạt động tốt cho mọi loại dữ liệu hoặc ngôn ngữ. Mô hình có thể không hiểu được ý nghĩa của các từ đối với một lĩnh vực cụ thể hoặc ngôn ngữ không phổ biến.
- Ứng dụng hạn chế: Word embedding có thể không hoạt động tốt cho các ứng dụng đặc biệt, như NLP dựa trên dấu vết thời gian hoặc ngôn ngữ chuyên ngành. Đối với các trường hợp này, cần phải tùy chỉnh hoặc mở rộng word embedding.
- Phụ thuộc vào dữ liệu lớn: Để có word embedding hiệu quả, cần một lượng dữ liệu lớn để huấn luyện. Điều này có thể là một thách thức đối với các tác vụ với dữ liệu hạn chế.
- Giải quyết vấn đề ẩn số: Word embedding có thể lặp lại và tạo ra biểu diễn không tốt cho các từ hiếm. Điều này có thể yêu cầu các phương pháp xử lý đặc biệt để giải quyết vấn đề này.

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 ?

- **Áp dụng word embedding cho phân loại text:**

1. Biểu diễn từ bằng word embedding:

   • Trước tiên, bạn cần một bộ dữ liệu lớn để huấn luyện mô hình word embedding như Word2Vec, GloVe hoặc BERT.

   • Trong quá trình huấn luyện, các từ trong bộ dữ liệu sẽ được biểu diễn dưới dạng các vectơ số học, sao cho các từ có ý nghĩa tương tự sẽ có các vectơ gần nhau trong không gian vectơ.

2. Chuẩn bị dữ liệu huấn luyện:

   • Chuẩn bị dữ liệu huấn luyện văn bản với các văn bản đã được gán nhãn.

   • Tiền xử lý dữ liệu, bao gồm việc tách từ (tokenization), loại bỏ stop words và các ký tự không cần thiết, chuyển đổi các từ thành chữ thường hoặc chữ hoa, …

3. Biểu diễn văn bản bằng word embedding:

   • Sử dụng mô hình word embedding đã được huấn luyện để biểu diễn mỗi từ trong văn bản dưới dạng vectơ số học.

   • Cộng hoặc trung bình các vectơ từ để biểu diễn toàn bộ văn bản.

4. Xây dựng mô hình phân loại:

   • Sử dụng các thuật toán học máy như máy học tập sâu (deep learning) hoặc học máy cổ điển để xây dựng mô hình phân loại.

   • Mô hình có thể là một mạng nơ-ron (neural network) đơn giản hoặc phức tạp hơn, như mạng nơ-ron hồi quy (RNN), mạng nơron tái lặp (LSTM), hoặc các mô hình sử dụng kiến thức đại diện (pre-trained models) như BERT hoặc GPT.

5. Huấn luyện và đánh giá mô hình:

   • Sử dụng dữ liệu huấn luyện để đào tạo mô hình phân loại, sử dụng vectơ biểu diễn văn bản được tạo ra từ word embedding.

• Sử dụng dữ liệu kiểm tra để đánh giá hiệu suất của mô hình bằng các phép đo như độ chính xác, F1-score, hoặc ma trận nhầm lẫn.

– **Chạy code và giải thích:**

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

+ Thực hiện thêm các thư viện cần sử dụng.

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

+ Khởi tạo 2 ma trận, 1 ma trận chứa các câu và 1 ma trận chứa các giá trị 0 và 1 tương ứng với câu trên tiêu cực hay tích cực.

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

+ Khởi tạo số lượng từ vựng ước tính là 50.

+ Sử dụng hàm one_hot trong keras biểu diễn mỗi từ dưới dạng mã hoá thành các các số nguyên.

```
# 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)
```

```
[[18 41  0  0]
 [47  7  0  0]
 [20 12  0  0]
 [20  7  0  0]
 [31  0  0  0]
 [ 1  0  0  0]
 [39 12  0  0]
 [36 47  0  0]
 [39  7  0  0]
 [ 5 11 41  1]]
```

+ Vì mỗi vector có độ dài khác nhau nên sử dụng lệnh pad_sequence để đưa độ dài mỗi câu có số lượng là 4 từ hay độ dài của vector là 4 để thuận tiện cho keras.

```
# Define the model
model = Sequential()
model.add(Embedding(vocab_size, 8, input_length=max_length))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
```

+ Khởi tạo mô hình

+ Lớp nhúng được sử dụng để ánh xạ các từ trong từ vựng thành các vector có số chiều nhất định.

• vocab_size: Đây là kích thước của từ vựng, số lượng từ khác nhau trong dữ liệu.

• 8: Đây là kích thước của vector nhúng cho mỗi từ. Mỗi từ sẽ được biểu diễn bằng một vector có 8 chiều.

• input_length=max_length: Đây chỉ định độ dài của dãy đầu vào.

+ Lớp tiếp theo thực hiện việc làm phẳng (flatten) các vector này thành một vector 1 chiều duy nhất.

+ Lớp cuối là lớp đầu ra của mô hình, là một lớp kết nối đầy đủ và có 1 đơn vị đầu ra.

• Hàm kích hoạt (activation function) được sử dụng ở đây là 'sigmoid', thường được sử dụng trong các mô hình phân loại nhị phân để đưa ra dự đoán xác suất dự đoán là giá trị 0 hoặc 1.

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

# Summarize the model
print(model.summary())
```

```
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, 1)                 33

=================================================================
Total params: 433 (1.69 KB)
Trainable params: 433 (1.69 KB)
Non-trainable params: 0 (0.00 Byte)
```

+ Chạy mô hình và in ra thông tin cơ bản.

+ Mô hình được biên dịch với trình tối ưu hóa 'adam', hàm mất mát nhị phân chéo, và sử dụng độ chính xác làm chỉ số đánh giá.

```
# 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)
print('Accuracy: %.2f%%' % (accuracy * 100))
```

+ Thực hiện huấn luyện mô hình.

+ In ra độ chính xác của mô hình.

```
# Plot training accuracy
plt.plot(history.history['accuracy'])
plt.title('Training Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train'], loc='upper left')
plt.show()

# Plot training loss
plt.plot(history.history['loss'])
plt.title('Training Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```

+ Tiến hành vẽ biểu đồ để xem quá trình huấn luyện được lưu trong history.

+ plt.plot(history.history['accuracy']): Vẽ đường cong biểu đồ cho độ chính xác của mô hình trên tập dữ liệu huấn luyện.

+ plt.title('model accuracy'): Đặt tiêu đề cho biểu đồ là "model accuracy" (độ chính xác của mô hình).

+ plt.ylabel('accuracy'): Đặt nhãn trục y là "accuracy" (độ chính xác).

+ plt.xlabel('epoch'): Đặt nhãn trục x là "epoch" (số vòng lặp huấn luyện).

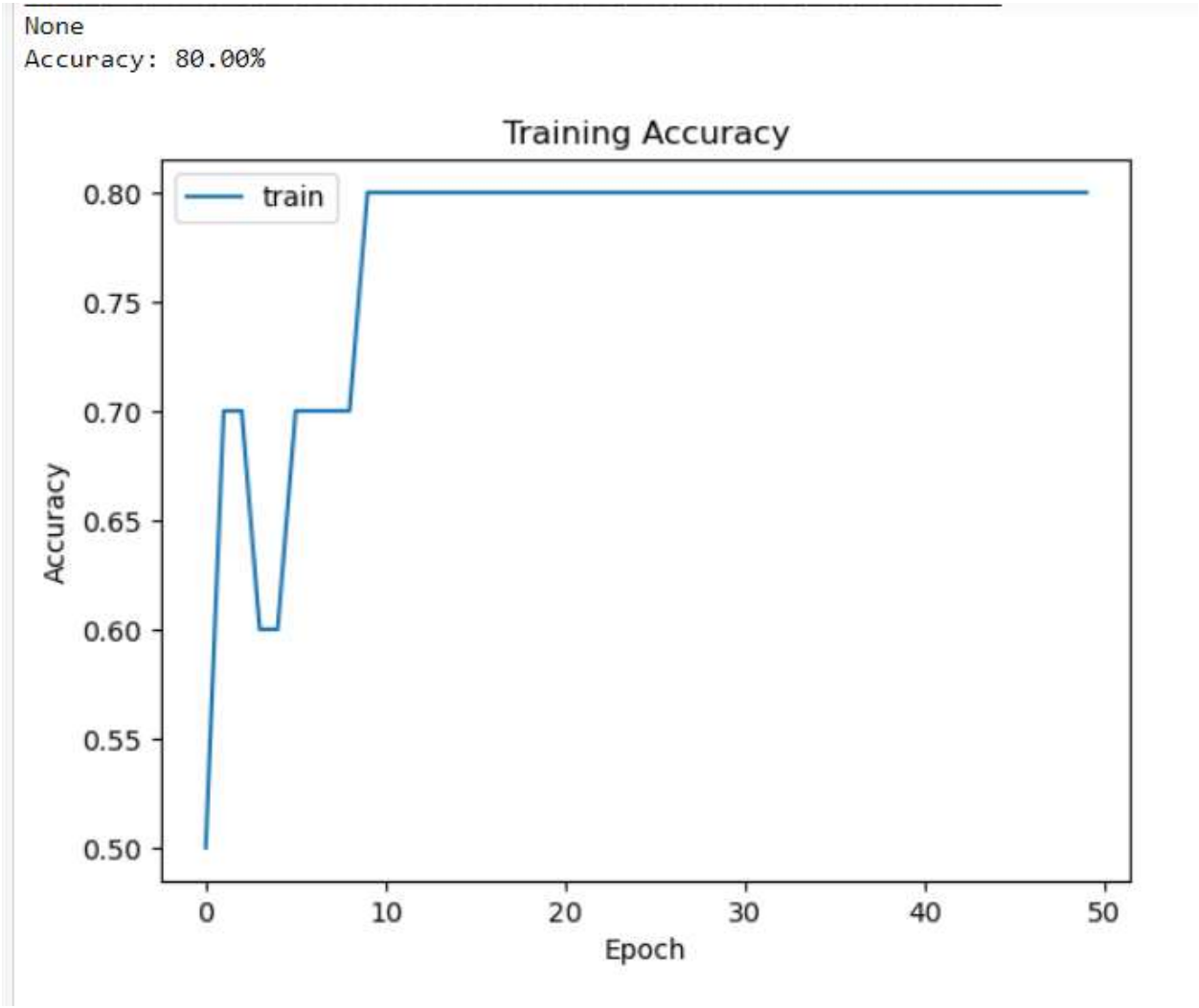
+ plt.legend(['train', 'test'], loc='upper left'):

  • Tạo chú thích (legend) trên biểu đồ để xác định các đường cong. Trong trường hợp này, nó tạo hai mục trong hộp chú thích là "train" và "test" để biểu thị độ chính xác trên tập dữ liệu huấn luyện và tập dữ liệu kiểm tra.

  • loc='upper left' chỉ định vị trí của hộp chú thích là ở góc trên bên trái của biểu đồ.

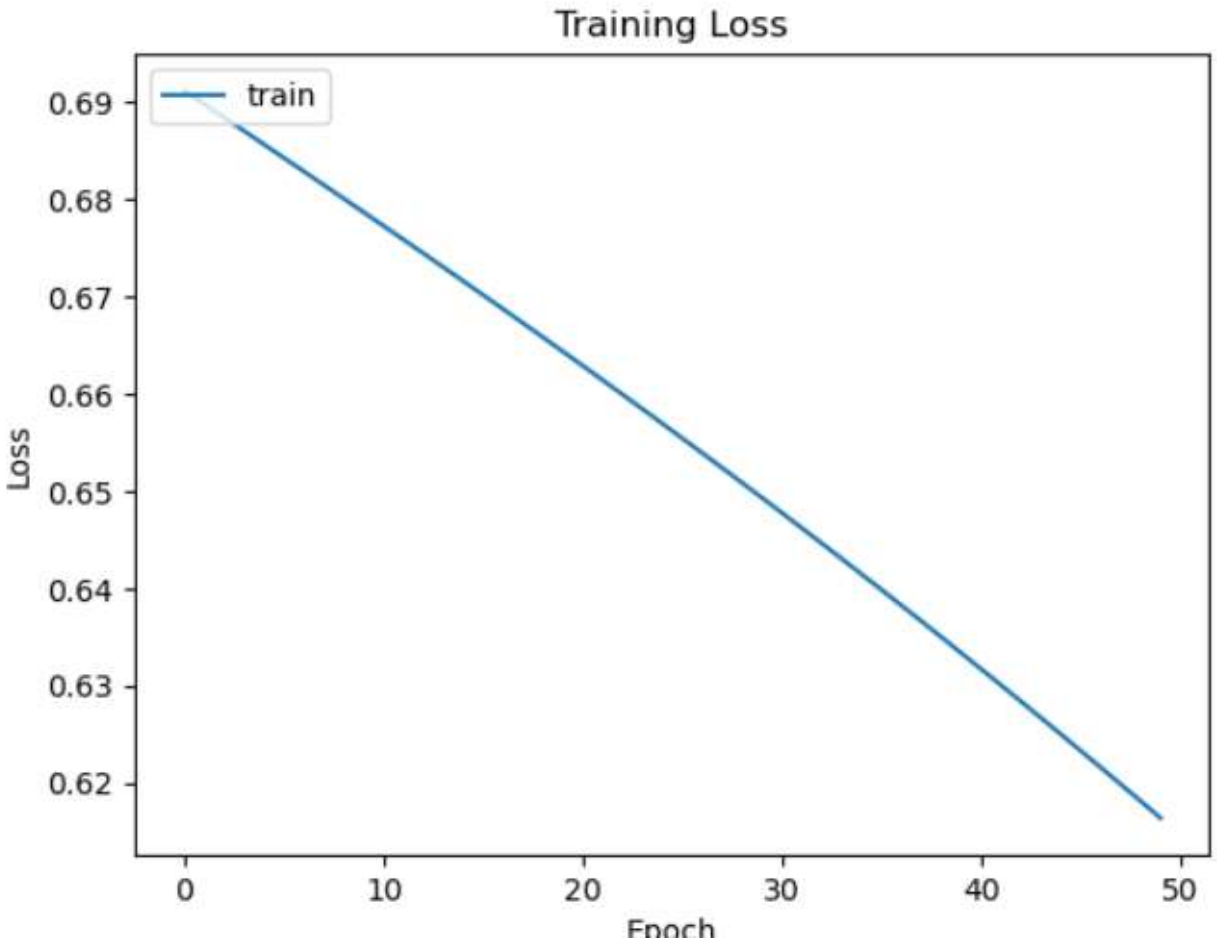
+ plt.show(): Lệnh này hiển thị biểu đồ trên màn hình.

+ Tương tự với “loss”



+ Độ chính xác của mô hình tăng lên sau mỗi lần lặp mà đạt giá trị cao nhất là 80%.

+ Trong nhiều vòng lặp, độ chính xác của mô hình không thay đổi và tăng cao sau một số vòng nhất định.



+ Độ sai lệch của giữa dự đoán của mô hình và giá trị thực tế giảm liên tục sau mỗi vòng huấn luyện.

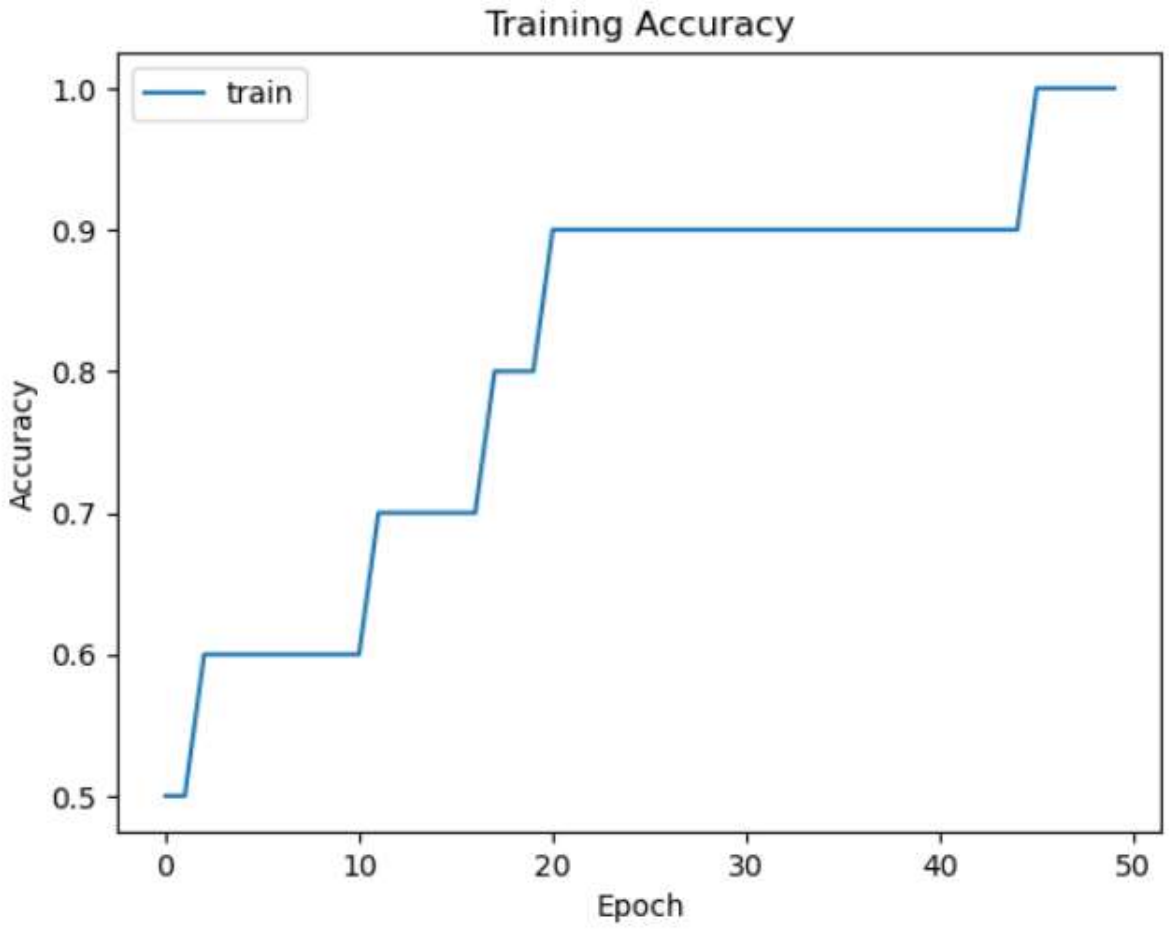
+ Giá trị giảm từ trên 0.69 và đạt giá trị cuối cùng gần 0.2.

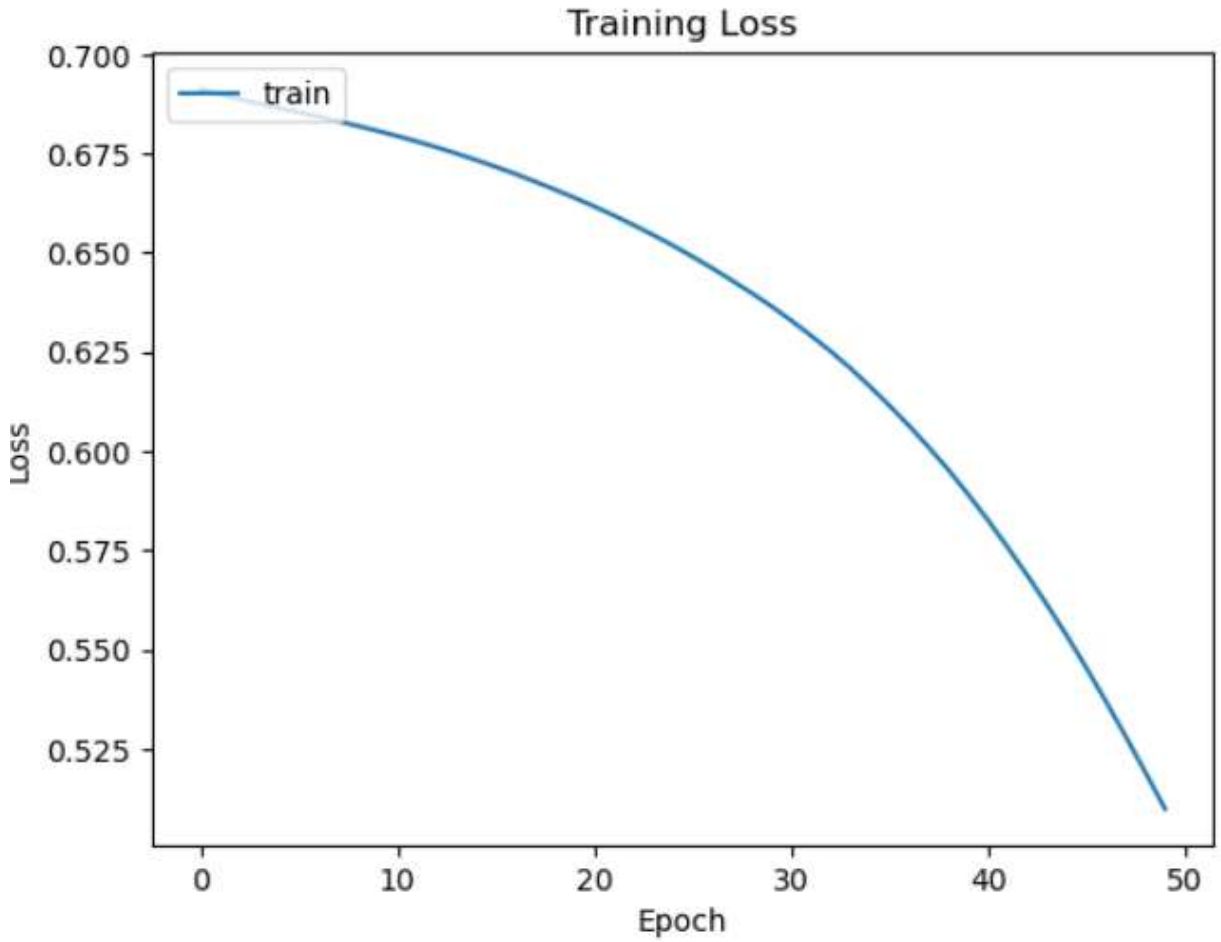
- Chạy code với kiến trúc 2:

```
# Define the model
model = Sequential()
model.add(Embedding(vocab_size, 8, input_length=max_length))
model.add(Flatten())

model.add(Dense(32, activation='relu'))  # Lớp ẩn với 12 đơn vị và hàm kích hoạt ReLU
model.add(Dense(32, activation='relu'))              # Lớp ẩn với 8 đơn vị và hàm kích hoạt ReLU
model.add(Dense(16, activation='relu'))

model.add(Dense(1, activation='sigmoid')) # Lớp đầu ra với hàm kích hoạt Sigmoid để đổi đầu ra c


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

+ Thực hiện thêm 2 lớp dense chưa 32 units và 1 lớp chứa 16 units. Các lớp được kích hoạt bởi hàm relu: f(x) = max(0, x).

```
_________________________________________________________________
None
Accuracy: 100.00%
```





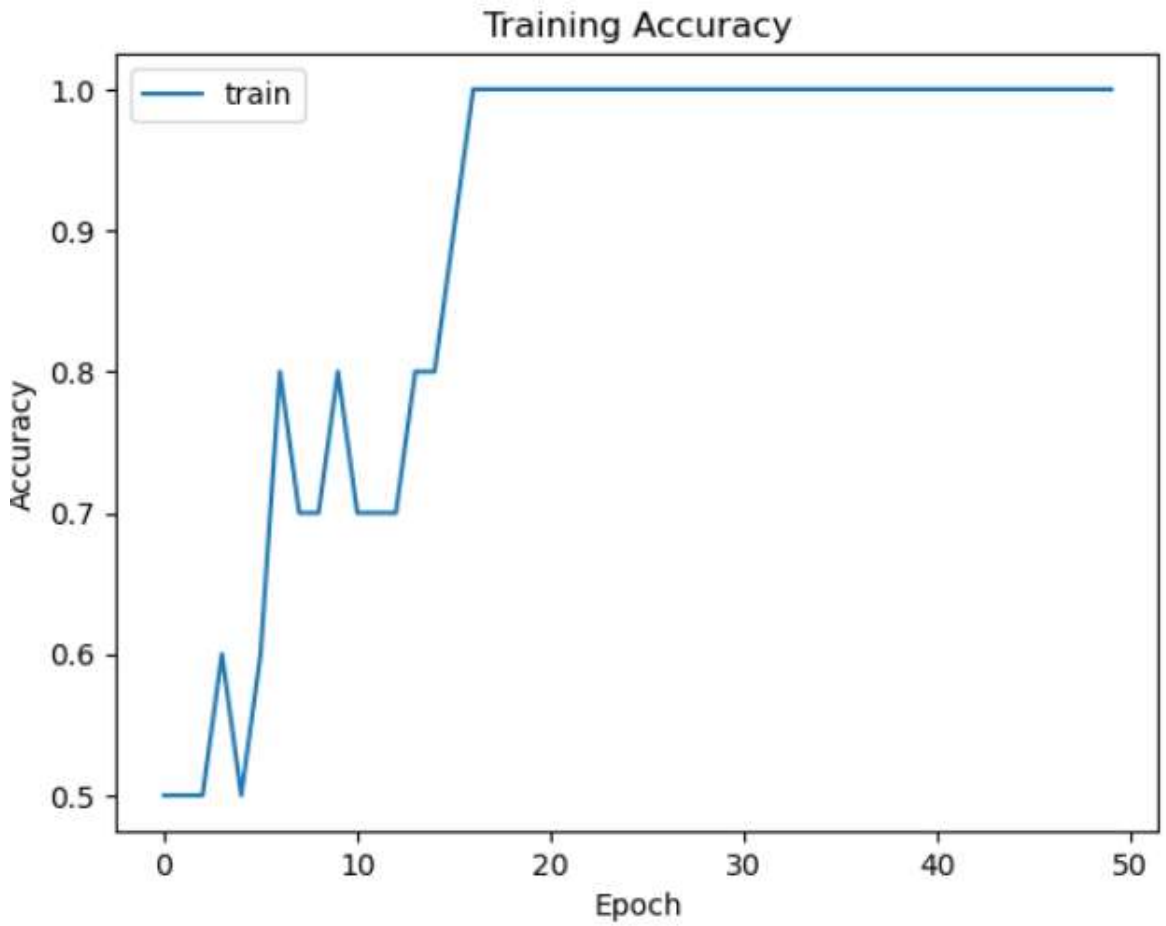
+ Sau khi thêm 3 lớp, độ chính xác của mô hình tăng lên thành 100%.

+ Chênh lệch dự đoán cũng được giảm hẳn so với mô hình ban đầu, kết quả cuối dưới 0.45

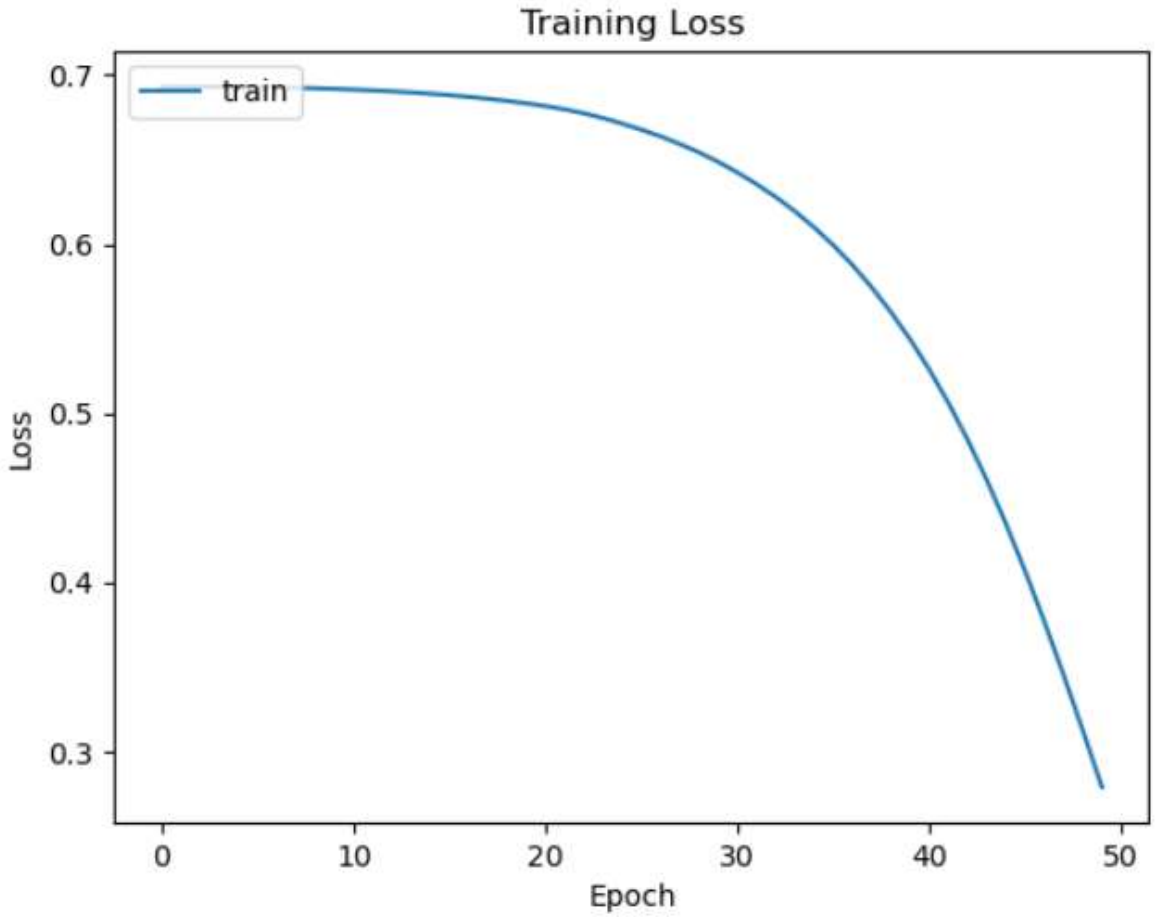
- Thực hiện chạy code với kiến trúc 3:

```
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
```

+ Tiếp tục thêm nhiều lớp ở giữa so với mô hình 2.



+ Sau khi thêm nhiều lớp, mô hình học tốt hơn và đạt độ chính xác là 100% nhanh hơn hẳn so với mô hình 1 và 2, cụ thể sau 10 vòng lặp.



+ Dự đoán chênh lệch cũng được giảm mạnh, kết quả cuối cùng gần bằng 0, cụ thể là ~ 0.2

3. 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?

Áp dụng cho phân loại review restaurant

+ Bước đầu tiên là tạo ra một bộ từ vựng (vocabulary) từ tập dữ liệu đánh giá nhà hàng.
+ Sau đó, mỗi từ trong từ vựng sẽ được ánh xạ vào một vector Word Embedding.
+ Đối với mỗi đánh giá nhà hàng, chúng ta có thể lấy trung bình (hoặc tổng) các vector Word Embedding của các từ trong đánh giá đó để tạo ra một vector biểu diễn cho toàn bộ đánh giá.

+ Vector biểu diễn này có thể được sử dụng làm đầu vào cho mô hình học máy như mạng nơ-ron.

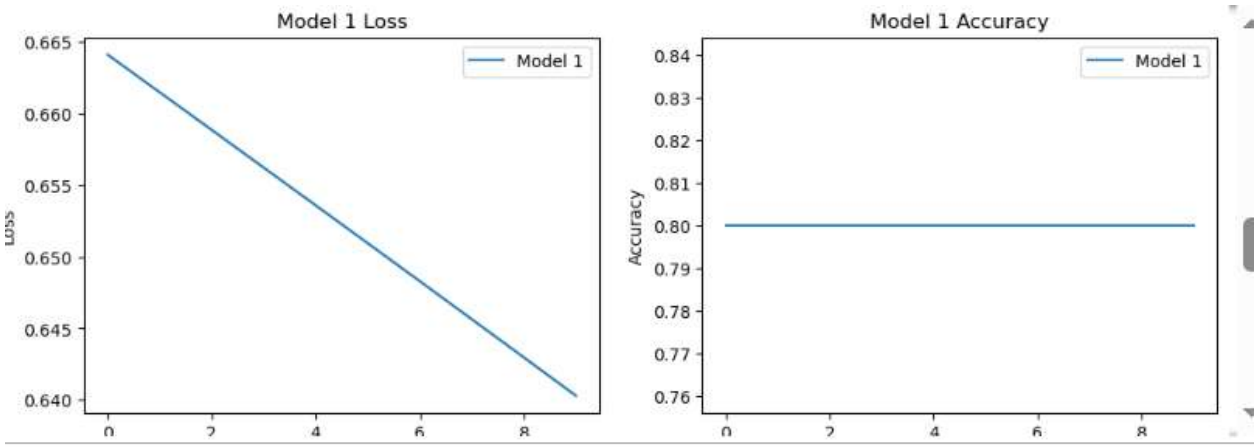
– Ví dụ code :

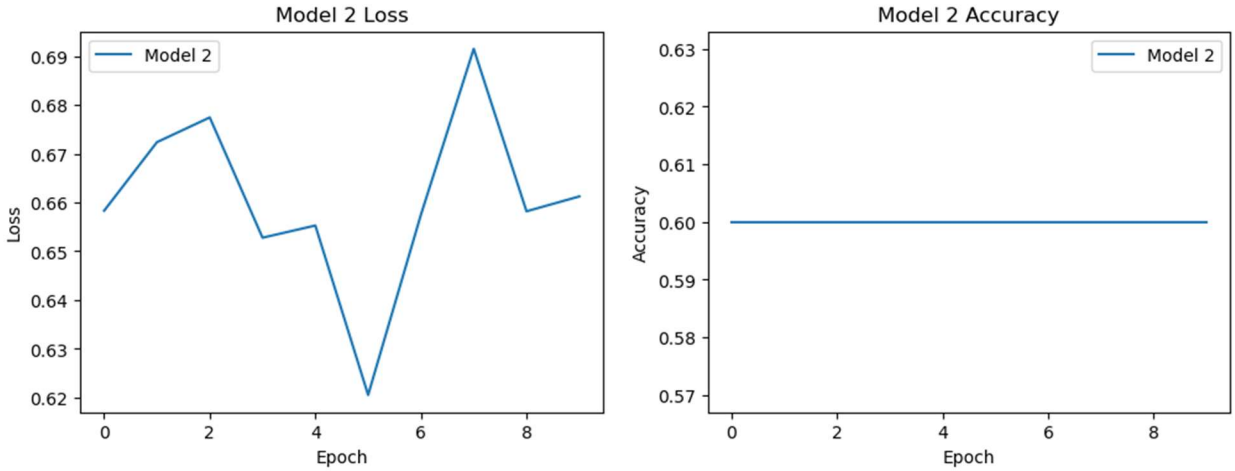
```
import numpy as np
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense, Dropout
import matplotlib.pyplot as plt

# Dữ liệu đánh giá nhà hàng và nhãn lớp (positive/negative)
reviews = ['The food was great!',
           'The service was terrible.',
           'I loved the ambiance.',
           'The food and service were excellent.',
           'Not a good experience.']

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

# Sử dụng Tokenizer để mã hóa văn bản thành các chuỗi số nguyên
max_words = 1000  # Số từ tối đa trong từ vựng
tokenizer = Tokenizer(num_words=max_words, oov_token='<OOV>')
tokenizer.fit_on_texts(reviews)
sequences = tokenizer.texts_to_sequences(reviews)

# Điền (padding) các chuỗi số nguyên để có độ dài tối đa
max_sequence_length = 10
padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length, padding='post')

# Xây dựng các mô hình mạng nơ-ron khác nhau

# Mô hình 1: Mô hình đơn giản
model1 = Sequential()
model1.add(Embedding(max_words, 8, input_length=max_sequence_length))
model1.add(Flatten())
model1.add(Dense(1, activation='sigmoid'))

# Mô hình 2: Mô hình với thêm lớp Dropout để tránh overfitting
model2 = Sequential()
model2.add(Embedding(max_words, 8, input_length=max_sequence_length))
model2.add(Flatten())
model2.add(Dense(4, activation='relu'))
model2.add(Dropout(0.5))
model2.add(Dense(1, activation='sigmoid'))
```

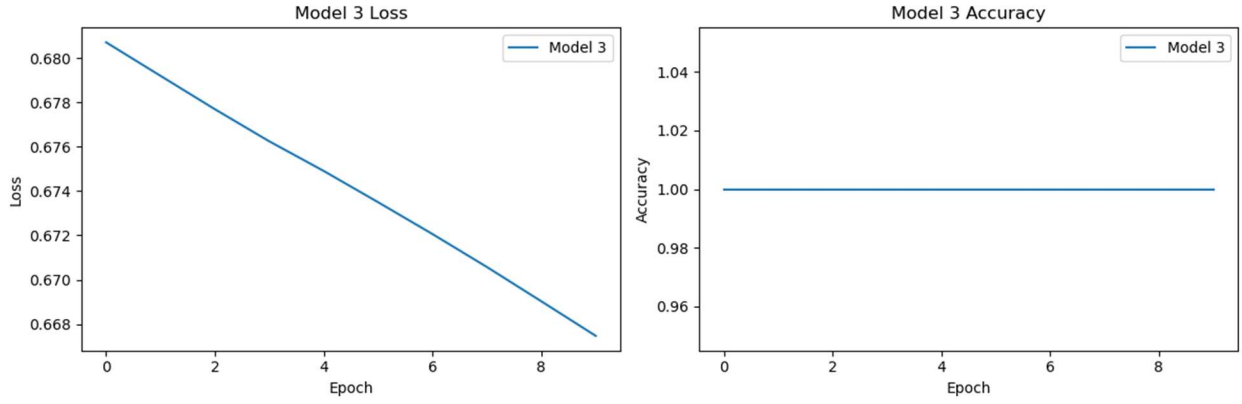
```python
# Mô hình 3: Mô hình với kiến trúc phức tạp hơn
model3 = Sequential()
model3.add(Embedding(max_words, 16, input_length=max_sequence_length))
model3.add(Flatten())
model3.add(Dense(8, activation='relu'))

model3.add(Dense(8, activation='relu'))
model3.add(Dense(1, activation='sigmoid'))

# Biên dịch và huấn luyện các mô hình
models = [model1, model2, model3]

for i, model in enumerate(models):
    print(f"Model {i + 1} Summary:")
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    print(model.summary())
    model.fit(padded_sequences, labels, epochs=10, verbose=0)
    loss, accuracy = model.evaluate(padded_sequences, labels, verbose=0)
    print(f"Accuracy (Model {i + 1}): {accuracy * 100}%")


# Số lượng mô hình
num_models = len(models)

# Biểu đồ loss và accuracy cho từng mô hình
for i, model in enumerate(models):
    # Huấn luyện mô hình (nếu chưa được huấn luyện)
    history = model.fit(padded_sequences, labels, epochs=10, verbose=0)

    # Lấy lịch sử (history) của mô hình
    loss = history.history['loss']
    accuracy = history.history['accuracy']

    # Vẽ biểu đồ loss
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(loss, label=f'Model {i + 1}')
    plt.title(f'Model {i + 1} Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    # Vẽ biểu đồ accuracy
    plt.subplot(1, 2, 2)
    plt.plot(accuracy, label=f'Model {i + 1}')
    plt.title(f'Model {i + 1} Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

plt.tight_layout()
plt.show()
```

```
Model 1 Summary:
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_14 (Embedding)    (None, 10, 8)             8000

 flatten_14 (Flatten)        (None, 80)                0

 dense_47 (Dense)            (None, 1)                 81

=================================================================
Total params: 8081 (31.57 KB)
Trainable params: 8081 (31.57 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
None
Accuracy (Model 1): 80.0000011920929%
Model 2 Summary:
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_15 (Embedding)    (None, 10, 8)             8000

 flatten_15 (Flatten)        (None, 80)                0

 dense_48 (Dense)            (None, 4)                 324

 dropout_1 (Dropout)         (None, 4)                 0

 dense_49 (Dense)            (None, 1)                 5

=================================================================
=================================================================
Total params: 8329 (32.54 KB)
Trainable params: 8329 (32.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Accuracy (Model 2): 60.00000238418579%
Model 3 Summary:
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_16 (Embedding)    (None, 10, 16)            16000

 flatten_16 (Flatten)        (None, 160)               0

 dense_50 (Dense)            (None, 8)                 1288

 dense_51 (Dense)            (None, 8)                 72

 dense_52 (Dense)            (None, 1)                 9

=================================================================
Total params: 17369 (67.85 KB)
Trainable params: 17369 (67.85 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
=================================================================
Total params: 17369 (67.85 KB)
Trainable params: 17369 (67.85 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Accuracy (Model 3): 100.0%
```


Model 1 Loss
Model 1
Loss
0.665
0.660
0.655
0.650
0.645
0.640
0
2
4
6
8
Model 1 Accuracy
Model 1
Accuracy
0.84
0.83
0.82
0.81
0.80
0.79
0.78
0.77
0.76
0
2
4
6
8


Model 2 Loss
Model 2
Loss
0.69
0.68
0.67
0.66
0.65
0.64
0.63
0.62
0
2
4
6
8
Epoch
Model 2 Accuracy
Model 2
Accuracy
0.63
0.62
0.61
0.60
0.59
0.58
0.57
0
2
4
6
8
Epoch


Model 3 Loss
Model 3
Loss
0.680
0.678
0.676
0.674
0.672
0.670
0.668
0
2
4
6
8
Epoch
Model 3 Accuracy
Model 3
Accuracy
1.04
1.02
1.00
0.98
0.96
0
2
4
6
8
Epoch