딥러닝 및 응용 실습

기말 과제 1 - 문자를 이용한 딥러닝 실습 보고서

정보보호학과 2017111370

이름 : 이혜민 🐹

목차 (Index)

1. 과제 소개

"RNN과 LSTM을 활용하여 영화리뷰인 imdb나 뉴스 reuters 데이터에 대한 분류의 정확도를 분석하기"

✓ 파라미터 값 설정

✓ RNN과 LSTM을 활용하여 분석 비교

2. 실험 준비

- 데이터 소개

IMDB 리뷰 감성 분석하기 (IMDB Movie Review Sentiment Analysis)

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941,
4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480,
284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111,
17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4,
1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13,
1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8,
316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12,
16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12,
215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15,
256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46,
7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26,
141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144,
30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38,
1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19,
178, 32]
1
```

- 텍스트 분류 중, 감성 분류를 연습하기 위해 자주 사용되는 훈련데이터
  - 영화 사이트 IMDB의 리뷰 데이터
- IMDB 데이터는 리뷰에 대한 텍스트와 해당 **리뷰가 긍정인 경우는 '1' 부정인 경우는 '0'**으로 표시한 레이블로 구성된 데이터
- 스탠포드 대학교에서 2011년 낸 논문에서 이 데이터를 소개, 해당 당시에는 이 데이터를 훈련 데이터와 텍스트 데이터 50:50 비율로 분할하여 88.98% 정확도를 얻었다고 함

2. 실험 준비

- 설정 및 매개변수 값 소개

✔ sample_size : 256

✔ epochs : 10, 20, 30

✔ batch_size : 64, 128

✔ validation_split : 0.2 / 0.3 / 0.4

✔ unit : 32, 128 (마지막 실험)

3. 실험 과정

- 코드 소개

문자를 이용한 딥러닝 실습 보고서 제출

학번 : 201711370 / 이름 : 이혜민 (정보보호)

1. Tensorflow가 제공하는 imdb 데이터 불러오기

```
# 텐서플로가 제공하는 imdb 데이터 불러오기
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Embedding
from tensorflow.keras import preprocessing

dic_size = 10000 # 사전의 크기 (사전에 있는 단어 개수)
sample_size = 512

# tensorflow가 제공하는 간소한 버전의 IMDB 읽기
(x_train, y_train),(x_test, y_test) = imdb.load_data(num_words=dic_size)
print(x_train.shape, x_test.shape)
print(x_train[0])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17465344/17464789 [==============================] - 0s 0us/step
17473536/17464789 [==============================] - 0s 0us/step
(25000,) (25000,)
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 11
```

```
[2] # 단어를 숫자, 숫자를 단어로 변환하는데 사용하는 표 (표는 dictionary로 구현)
    word2id = imdb.get_word_index()
    id2word = {word:id for id, word in word2id.items()}

    for i in range(1, 21):
      print(id2word[i], end='/')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/i
1646592/1641221 [==============================] - 0s 0us/step
1654784/1641221 [==============================] - 0s 0us/step
the/and/a/of/to/is/br/in/it/i/this/that/was/as/for/with/movie/but/film/on/
```

- Tensorflow가 제공하는 IMDB 데이터 불러오기

3. 실험 과정

- 코드 소개

2. 학습에 필요한 하이퍼 파라미터 값 설정

- sample_size : 256
- epochs : 10, 20, 30
- batch_size : 64, 128
- validation_split : 0.2 / 0.3 / 0.4

```
# 파라미터 값 for문으로 자동으로 돌아가도록 설정

parameters =[] # list of (sample_size, epochs, batch_size, validation_split)
for s in [256]:
  for e in [10,20,30]:
    for b in [64, 128]:
      for v in [0.2,0.3,0.4]:
        parameters.append((s,e,b,v))
```

- for문을 돌리기 위해서 사용하여 각 파라미터 값들 리스트로 지정

3. 실험 과정

- 대표적인 워드 임베딩 기법

워드 임베딩 (Word Embedding) : 단어의 순서와 의미를 내포하는 벡터의 형태로 단어를 표현하는 기법

- 워드 임베딩 기법을 사용하여 단어들에 대한 신뢰성 높은 특징값을 부여
- 문장에 대한 문법적 해석이 가능
- 단어의 거리를 통해서 의미론적 추론이 가능

3. 실험 과정

- RNN과 LSTM의 차이

Recurrent Neural Network(RNN)

- 입력과 출력이 각각 독립적이라고 가정한 기존의 신경망 구조에서 벗어나
동일한 활성화 함수를 한 시퀀스의 모든 요소마다 적용하여 출력 결과가 이전의 계산 결과에 영향을 받는 알고리즘

Long Short Term Method (LSTM)

- RNN의 짧은 시퀀스만 효과적으로 처리하는 한계점을 장기적으로 해결하기 위한 RNN의 변형 알고리즘

3. 실험 과정

실험 1) 워드 임베딩을 사용하여 IMDB 데이터를 부정/긍정으로 평가 분류하는 학습 모델

3. 단어 임베딩을 사용하여 IMDB 데이터를 부정/긍정으로 평가 분류하는 학습 모델

```
# 단어 임베딩을 사용하여 IMDB 데이터를 부정/ 긍정으로 평가 분류하는 학습 모델

# 텐서플로가 제공하는 imdb 데이터 불러오기
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Embedding
from tensorflow.keras import preprocessing

def training1(ssize, eps, bsize, vsplit):
  dic_size = 10000 # 사전의 크기 (사전에 있는 단어 개수)
  sample_size = ssize

  # tensorflow가 제공하는 간소한 버전의 IMDB 읽기
  (x_train, y_train),(x_test, y_test) = imdb.load_data(num_words=dic_size)

  # 16차원의 임베딩 공간
  embed_space_dim = 16

  x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=sample_size)
  x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=sample_size)
```

```
  embed = Sequential()
  embed.add(Embedding(input_dim=dic_size, output_dim=embed_space_dim, input_length=sample_s
  embed.add(Flatten())
  embed.add(Dense(32,activation='relu'))
  embed.add(Dense(1,activation='sigmoid'))
  embed.compile(loss='binary_crossentropy',optimizer='Adam',metrics=['accuracy'])
  hist = embed.fit(x_train, y_train, epochs=eps, batch_size=bsize, validation_split=vsplit,
                   verbose=2)

  return embed, hist

def evaluate1(embed):
# 모델 평가
  res = embed.evaluate(x_test, y_test, verbose=0)
  #print("Accuracy:", res[1]*100)
  return res

def trainingcurve(hist):
  # 학습 곡선
  plt.plot(hist.history['accuracy'])
  plt.plot(hist.history['val_accuracy'])
  plt.title('Model accuracy')
  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(['Train','Validation'], loc='best')
  plt.grid()
  plt.show()
```

- Flatten( ) 사용 후, Dense(32, ‘relu’), Dense(1, “sigmoid”) 신경망 층을 쌓은 구조

3. 실험 과정

실험 2) 워드 임베딩을 사용하여 IMDB 데이터를 부정/긍정으로 평가 분류하는 LSTM 학습 모델

4. 단어 임베딩을 사용하여 IMDB 데이터를 부정/긍정으로 평가 분류하는 **LSTM** 신경망 학습

```
# 단어 임베딩을 사용하여 IMDB 데이터를 부정/ 긍정으로 평가 분류하는 LSTM 신경망 학습
# 텐서플로가 제공하는 imdb 데이터 불러오기

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Embedding
from tensorflow.keras import preprocessing
from tensorflow.keras.callbacks import EarlyStopping

def training2(ssize, eps, bsize, vsplit):
  dic_size = 10000 # 사전의 크기 (사전에 있는 단어 개수)
  sample_size = ssize

  # tensorflow가 제공하는 간소한 버전의 IMDB 읽기
  (x_train, y_train),(x_test, y_test) = imdb.load_data(num_words=dic_size)

  # 16차원의 임베딩 공간
  embed_space_dim = 16

  x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=sample_size)
  x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=sample_size)

  early = EarlyStopping(monitor='val_accuracy', patience=5,restore_best_weights=True)
```

```
  # 신경망 모델의 설계와 학습 (LSTM층 포함)
  embed = Sequential()
  embed.add(Embedding(input_dim=dic_size, output_dim=embed_space_dim, input_length=sample_s
  embed.add(LSTM(units=32))
  embed.add(Dense(1,activation='sigmoid'))
  embed.compile(loss='binary_crossentropy',optimizer='Adam',metrics=['accuracy'])
  hist = embed.fit(x_train, y_train, epochs=eps, batch_size=bsize, validation_split=vsplit,
                   verbose=2, callbacks=[early])

  return embed, hist

# 모델 평가
def evaluate2(embed):
  res = embed.evaluate(x_test, y_test, verbose=0)
  #print("정확률은", res[1]*100)
  return res

# 학습 곡선
def trainingcurve(hist):
  plt.plot(hist.history['accuracy'])
  plt.plot(hist.history['val_accuracy'])
  plt.title('Model accuracy')
  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(['Train','Validation'], loc='best')
  plt.grid()
  plt.show()
```

- EarlyStopping(조기멈춤)을 사용하여 과잉 적합을 방지하는 코드 적용

- 이전 실험과는 다르게 신경망에 LSTM 모델과 units을 적용하여 모델 학습

3. 실험 과정

실험 3) LSTM 학습 모델의 성능 개선을 목적으로, Unit 값을 128로 조정

5. LSTM의 성능 개선을 위한 unit수 조정

```
[ ]  # 단어 임베딩을 사용하여 IMDB 데이터를 부정/ 긍정으로 평가 분류하는 LSTM 신경망 학습 - Unit 128
     # 텐서플로가 제공하는 imdb 데이터 불러오기

     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     from tensorflow.keras.datasets import imdb
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, LSTM, Embedding
     from tensorflow.keras import preprocessing
     from tensorflow.keras.callbacks import EarlyStopping

     def training2(ssize, eps, bsize, vsplit):
       dic_size = 10000 # 사전의 크기 (사전에 있는 단어 개수)
       sample_size = ssize

       # tensorflow가 제공하는 간소한 버전의 IMDB 읽기
       (x_train, y_train),(x_test, y_test) = imdb.load_data(num_words=dic_size)

       # 16차원의 임베딩 공간
       embed_space_dim = 16

       x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=sample_size)
       x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=sample_size)

       early = EarlyStopping(monitor='val_accuracy', patience=5,restore_best_weights=True)
```

```
       # 신경망 모델의 설계와 학습 (LSTM층 포함)
       embed = Sequential()
       embed.add(Embedding(input_dim=dic_size, output_dim=embed_space_dim, input_length=sample_s
       embed.add(LSTM(units=128))
       embed.add(Dense(1,activation='sigmoid'))
       embed.compile(loss='binary_crossentropy',optimizer='Adam',metrics=['accuracy'])
       hist = embed.fit(x_train, y_train, epochs=eps, batch_size=bsize, validation_split=
                        vsplit, verbose=2, callbacks=[early])

       return embed, hist

     # 모델 평가
     def evaluate2(embed):
       res = embed.evaluate(x_test, y_test, verbose=0)
       #print("정확률은", res[1]*100)
       return res

     # 학습 곡선
     def trainingcurve(hist):
       plt.plot(hist.history['accuracy'])
       plt.plot(hist.history['val_accuracy'])
       plt.title('Model accuracy')
       plt.ylabel('Accuracy')
       plt.xlabel('Epoch')
       plt.legend(['Train','Validation'], loc='best')
       plt.grid()
       plt.show()
```

- LSTM 신경망 모델의 units를 128로 조정 후, 각 파라미터 동일하게 실험해보기

3. 실험 과정

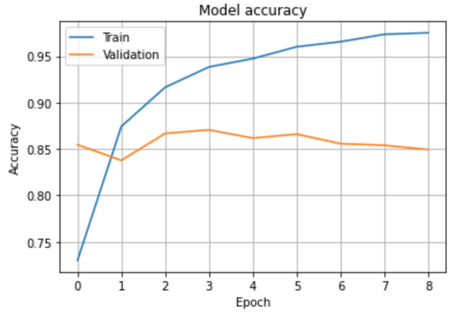
- 코드 소개

```
# 각 파라미터별로 학습 후, models2(dictionary에)에 저장
models2 = {}

for parameter in parameters:
  embed, hist = training2(parameter[0], parameter[1], parameter[2], parameter[3])
  models2[parameter] = (embed, hist)
```

```
313/313 - 15s - loss: 0.1468 - accuracy: 0.9474 - val_loss: 0.3646 - val_accuracy: 0.8618 -
Epoch 6/10
313/313 - 15s - loss: 0.1171 - accuracy: 0.9602 - val_loss: 0.4100 - val_accuracy: 0.8660 -
Epoch 7/10
313/313 - 15s - loss: 0.1012 - accuracy: 0.9657 - val_loss: 0.4349 - val_accuracy: 0.8558 -
Epoch 8/10
313/313 - 15s - loss: 0.0786 - accuracy: 0.9736 - val_loss: 0.5305 - val_accuracy: 0.8540 -
Epoch 9/10
313/313 - 15s - loss: 0.0718 - accuracy: 0.9753 - val_loss: 0.5277 - val_accuracy: 0.8494 -
Epoch 1/10
274/274 - 17s - loss: 0.4780 - accuracy: 0.7537 - val_loss: 0.3367 - val_accuracy: 0.8605 -
Epoch 2/10
274/274 - 14s - loss: 0.2499 - accuracy: 0.9031 - val_loss: 0.3125 - val_accuracy: 0.8693 -
```

```
# 모델 Accuracy 결과 출력

cnt = 0
for parameter in parameters:
  cnt += 1
  embed, hist = models2[parameter]
  res = evaluate2(embed)
  print( "[",cnt,"]"," sample_size:",parameter[0]," epochs:", parameter[1]," batch_size
  print("정확률: ", round(res[1]*100,2),"%")
  trainingcurve(hist)
```

```
[ 1 ]  sample_size: 256  epochs: 10  batch_size: 64  validation_split: 0.2
정확률:  86.13 %
```




```
[ 2 ]  sample_size: 256  epochs: 10  batch_size: 64  validation_split: 0.3
정확률:  85.78 %
```

- 실험에 대한 각 파라미터 별 학습 후에 dictionary 구조인 models2에 저장

- 실험한 결과에 대해서 분류 모델의 정확도(Accuracy)를 출력

4. 실험 결과 및 분석

실험 1) 워드 임베딩을 사용하여 IMDB 데이터를 부정/긍정으로 평가 분류하는 학습 모델

Batch_size	Validation_Split	Epoch : 10	Epoch : 20	Epoch : 30
64	0.2	86.65%	86.76%	86.63%
64	0.3	86.57%	86.48%	86.24%
64	0.4	86.45%	86.21%	86.2%
128	0.2	86.75%	86.57%	86.71%
128	0.3	86.49%	86.4%	86.38%
128	0.4	86.59%	86.21%	86.39%

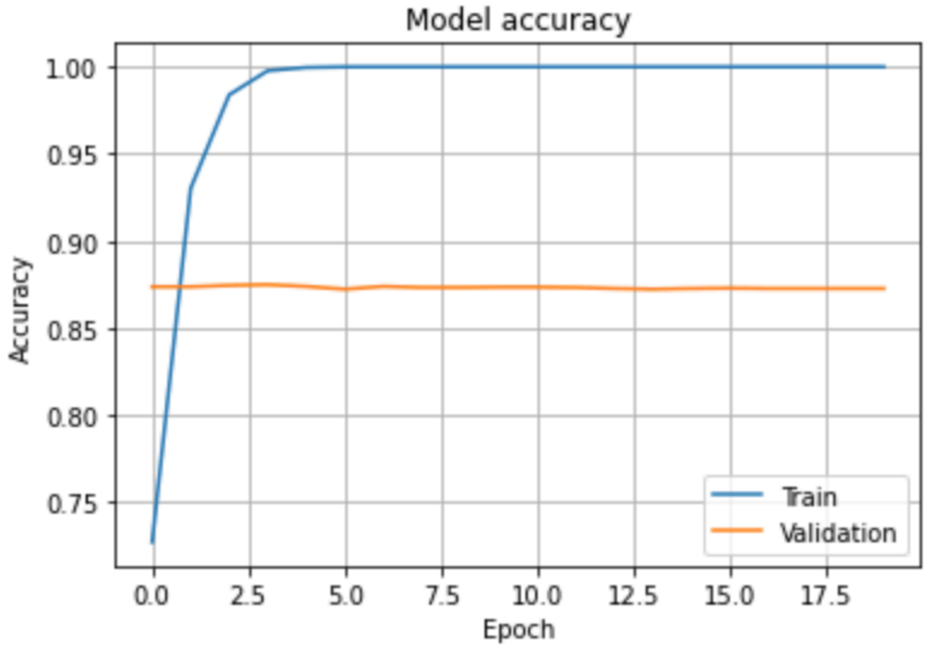
- **가장 좋은 성능** : batch_size : 64, Validation_split : 0.2, Epoch : 20
- **평균적으로 Batch_size : 128, Validation_split : 0.2인 경우**가 Epoch에 상관없이 Accuracy가 높음

4. 실험 결과 및 분석

실험 1) 워드 임베딩을 사용하여 IMDB 데이터를 부정/긍정으로 평가 분류하는 학습 모델

Epoch

```
[ 7 ]  sample_size: 256  epochs: 20  batch_size: 64  validation_split: 0.2
정확률:  86.76 %
```



- **가장 좋은 성능** : batch_size : 64, Validation_split : 0.2, Epoch : 20 인 경우

4. 실험 결과 및 분석

실험 2) 워드 임베딩을 사용하여 IMDB 데이터를 부정/긍정으로 평가 분류하는 LSTM 학습 모델

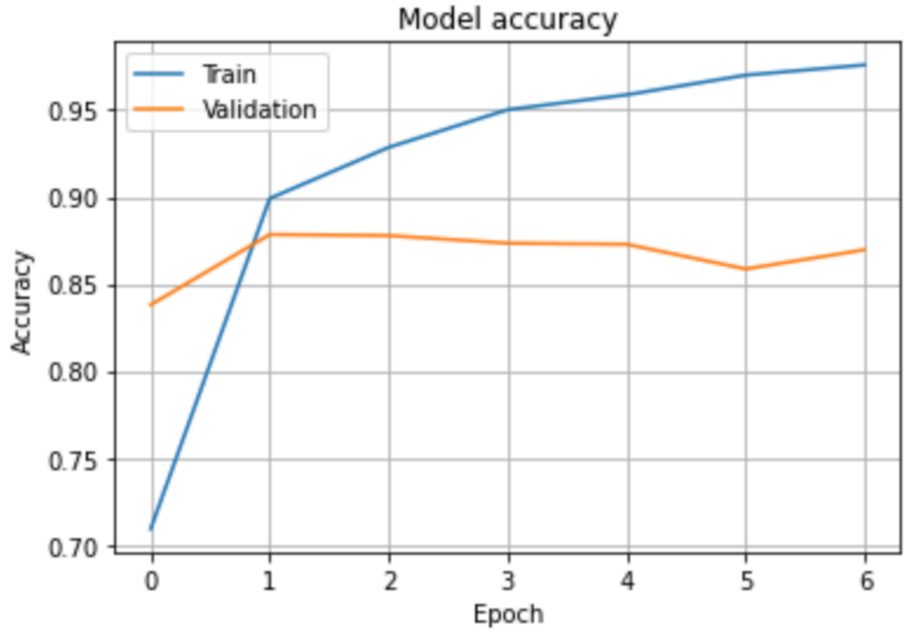
Batch_size	Validation_Split	Epoch : 10	Epoch : 20	Epoch : 30
64	0.2	86.82%	87.3%	86.96%
64	0.3	87.34%	86.57%	86.22%
64	0.4	86.35%	86.92%	86.68%
128	0.2	87.54%	87.07%	87.53%
128	0.3	87.15%	86.77%	87.42%
128	0.4	86.39%	86.96%	87.05%

- **가장 좋은 성능** : batch_size : 128, Validation_split : 0.2, Epoch : 30
- 실험 1과 마찬가지로 **평균적으로 Batch_size : 128, Validation_split : 0.2일 때가** Epoch에 상관없이 Accuracy가 전체적으로 높음

4. 실험 결과 및 분석

실험 2) 워드 임베딩을 사용하여 IMDB 데이터를 부정/긍정으로 평가 분류하는 LSTM 학습 모델

```
[ 16 ]  sample_size: 256  epochs: 30  batch_size: 128  validation_split: 0.2
정확률:  87.53 %
```



- **가장 좋은 성능** : batch_size : 128, Validation_split : 0.2, Epoch : 30인 경우

4. 실험 결과 및 분석

실험 3) LSTM 학습 모델의 성능 개선을 목적으로, Unit 값을 128로 조정

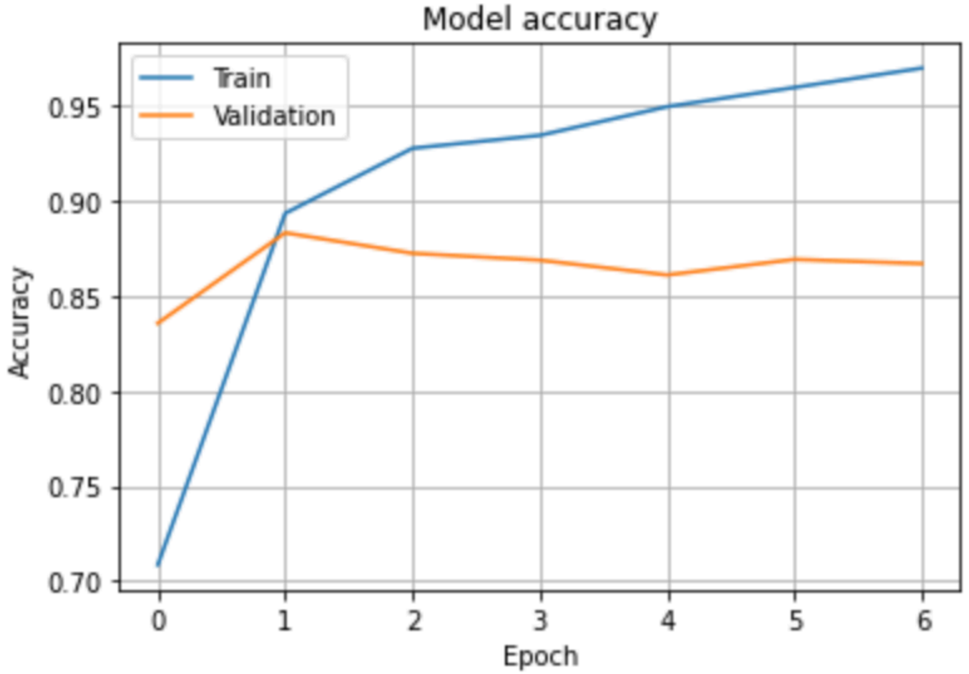
Batch_size	Validation_Split	Epoch : 10	Epoch : 20	Epoch : 30
64	0.2	86.13%	86.6%	87.35%
64	0.3	85.78%	85.61%	86.5%
64	0.4	86.1%	86.96%	85.69%
128	0.2	86.22%	86.91%	86.6%
128	0.3	86.49%	86.01%	86.89%
128	0.4	85.83%	86.72%	86.78%

- **가장 좋은 성능** : batch_size : 64, Validation_split : 0.2, Epoch : 30
- 이전 실험들에 비해서 전체적으로 성능이 낮음을 확인할 수 있음

4. 실험 결과 및 분석

실험 3) LSTM 학습 모델의 성능 개선을 목적으로, Unit 값을 128로 조정

```
[ 13 ]  sample_size: 256  epochs: 30  batch_size: 64  validation_split: 0.2
정확률:  87.35 %
```



- **가장 좋은 성능** : batch_size : 64, Validation_split : 0.2, Epoch : 30인 경우

5. 마무리 및 느낀점

✔ 평균적으로 **Batch_size : 128, Validation_split:0.2인 경우** Epoch에 상관없이 Accuracy가 높다.

✔ Units의 수를 늘리는 것은 성능 향상에 크게 영향을 미치지 않는다.

✔ 기본적인 분류 모델 보다는 RNN을 향상 시킨 LSTM 모델의 성능이 좋음을 확인할 수 있었다.

✔ **Early Stopping 적용은 과잉 적합**에 도움이 되었지만, 적용할 경우 굳이 Epoch을 많이 늘려서 실험할 필요는 없을 것 같다.

✔ LSTM 모델의 Validation Accuracy의 경우 Epoch이 증가함에 따라 중간에 **주춤 감소**하는 양상을 볼 수 있었다.

딥러닝 및 응용 실습

한 학기 동안 고생하셨습니다

- END -