BE201 Artificial Intelligence Basics, Homework

202211167 장지원

1. Google Colaboratory

주의 사항: 런타임 유형을 GPU로 바꾸고 학습을 진행해야 한다 (속도 때문에).

2. Spam Mail Classification by Naive Bayes

Model Training: You should first complete the implementation in Part 2 of the skeleton code so that the class (NBModel) computes the required probability values for the training dataset.

(a) (5pts) Implement the function word exists() so that it returns True or False depending on the existence of the word in the training dataset.

```
def word_exists(self, word):
        if word in self.all_words_list: # 토큰화된 train_dataset 리스트에
word가 있다면
          return True
        else:
         return False
```

(b) (5pts) Implement the function spam prob() so that it returns the probability that an email was a spam in the training dataset.

```
def spam_prob(self):
        total_rows = len(self.train_dataset) # 전체 행 갯수
        spam_rows = 0 # spam 행 갯수
        for i in range(total_rows):
          is_spam = self.train_dataset[i][1]
          if is_spam == 1: # if is_spam이 1이면
            spam_rows += 1 # +1
        spam_probability = spam_rows/total_rows # spam 행/ 전체 행
        return spam_probability
```

(c) (10pts) Implement the two functions spam cond prob() and ham cond prob(). (The first function should return the conditional probability (0.0 ~ 1.0) that a spam email contains the given word. Similarly, the second function should return the conditional probability that a ham email contains the given word.)

```
def spam_cond_prob(self, word):
        # spam 확률 (분모)
        spam_probability = self.spam_prob()

        # word in spam 확률 (분자)
        total_rows = len(self.train_dataset) # 전체 행 갯수
        spam_rows = 0 # spam 행 갯수
        word_in_spam_rows = 0 # word가 있는 spam의 행 갯수
        for i in range(total_rows):
          content, is_spam = self.train_dataset[i]
          if is_spam == 1: # if is_spam이 1이면
```

```
            spam_rows += 1 # +1
            if word in tokenize(content): # is_spam이 1인 메세지 안에
word가 있는지 확인
              word_in_spam_rows += 1 # +1

        # 조건부 확률
        if spam_probability == 0:
          return 1 # 분모가 0이면 1 return

        word_in_spam_probability = word_in_spam_rows/total_rows #
word가 spam인 메세지에에 있을 확률
        spam_conditional_probability =
word_in_spam_probability/spam_probability
        return spam_conditional_probability

    """
    Task 4. P(word_i|S=ham)
    Return the conditional probability (0.0~1.0) that a ham email has
the given word
    """
    def ham_cond_prob(self, word):
        # ham 확률 (분모)
        ham_probability = 1-self.spam_prob()

        # word in ham 확률 (분자)
        total_rows = len(self.train_dataset) # 전체 행 갯수
        ham_rows = 0 # ham 행 갯수
        word_in_ham_rows = 0 # word가 있는 ham의 행 갯수
        for i in range(total_rows):
          content, is_spam = self.train_dataset[i]
          if is_spam == 0: # if is_spam이 0이면 (ham 이면)
            ham_rows += 1 # +1
            if word in tokenize(content): # is_spam이 0인 메세지 안에
word가 있는지 확인
              word_in_ham_rows += 1 # +1

        if ham_probability == 0:
          return 1  # 분모가 0이면 1 return

        word_in_ham_probability = word_in_ham_rows/total_rows # word가
ham인 메세지에에 있을 확률
        ham_conditional_probability =
word_in_ham_probability/ham_probability
        return ham_conditional_probability
```

Model Inference: In Part 3 of the skeleton code, you should complete the inference procedure (i.e., inference() function). A skeleton code is already given, so please take a look at the portion marked with comments to see where to edit.

(d) (10pts) Complete the code so that it computes the probability of either spam or ham, i.e., spam prob and ham prob variables using the Naive Bayes algorithm we learned.

```
def inference(model, test_dataset):
    n_correct = 0
    spam_prob = model.spam_prob()
    ham_prob = 1-spam_prob
    for content, is_spam in test_dataset:
        words_set = tokenize(content)  # The set of words in an tested
email (i.e., inference)

        ###########################################################
        spam_prob = model.spam_prob()
        ham_prob = 1-model.spam_prob()
        is_spam_prob = spam_prob # initial value
        is_ham_prob = ham_prob

        for word in words_set:
            if model.word_exists(word):
                is_spam_prob *= model.spam_cond_prob(word)
                is_ham_prob *= model.ham_cond_prob(word)
                pass

        spam_prob = is_spam_prob
        ham_prob = is_ham_prob
        ###########################################################


        # If the probability of spam is higher than that of ham, then
we predict it as a spam
        is_spam_prediction = spam_prob >= ham_prob
        if is_spam_prediction == is_spam:
            n_correct += 1

    n_samples = len(test_dataset)
    accuracy = n_correct / n_samples
    print("Accuracy {} ({} / {})\n".format(accuracy, n_correct,
n_samples))
```

(e) (10pts) Explain your implementation briefly and report the classification results.

a. 토큰화(tokenization)된 dataset에 word가 있는지 확인하는 코드를 작성해보았다. b. dataset의 row들을 interation하며 is_spam index가 1로 labeling 되어있는 갯수를 count 하였다. 이후 이를 전체로 나누어서 $P(S = spam)$을 구해보았다.

c. spam인 문자에 해당 word가 들어있을 확률을 구하고 전체로 나누어서 $P(word = i|S = spam)$을 구해보았다.

d. Bayes rule을 바탕으로 ($P(S = spam|dataset) = P(word = i1|S = spam) * P(word = i1|S =$

$spam) * P(word = i1|S = spam) \ldots * P(S = spam))$ inferencing을 구현해보았다.

결과는 밑과 같았다 (Accuracy : 94%).

```
Accuracy 0.9491626794258373 (1587 / 1672)
Accuracy (scikit-learn) 0.9796650717703349 (1638 / 1672)
```

(f) (10pts) Compare your results with three other Naive Bayes classification algorithms imple-mented in the scikit-learn library: MultinomialNB, BernoulliNB, ComplementNB. (You can test them by modifying the baseline code already implemented in Part 4.)

결과는 밑과 같았다 (Accuracy : 94% / MltunoamialNB-Accuracy : 97% / BernoulliNB-Accuracy : 97% / complementNB : 96%)

```
Accuracy 0.9491626794258373 (1587 / 1672)
Accuracy (scikit-learn : MultinomialNB) 0.9796650717703349 (1638 /
1672)
Accuracy (scikit-learn : BernoulliNB) 0.9766746411483254 (1633 / 1672)
Accuracy (scikit-learn : ComplementNB) 0.9617224880382775 (1608 / 1672)
```

(g) (10pts) Your first implementation would have a lower accuracy than the scikit-learn algorithms. Devise any strategy to improve the accuracy and discuss it in the report. Include your implementation in the submitted code by adding a new inference function and/or model class.

첫 번째 구현에서는 train 부분이 정확히 작성 되어있지 않았다. 이를 해결하기 위해 Model class에 training 함수를 추가해 주었다.

train은 '단어 빈도수'에 대하여 진행할 것이다. 이를 위해서 단어 빈도수를 저장하는 word_frequency dictionary를 init 함수에 선언해 주었다. training 함수는 NBModel instance가 생성되면 자동으로 실행되도록 구현하였다.

```
class NBModel:
    def __init__(self, train_dataset):
        self.train_dataset = train_dataset
       self.all_words_list = tokenize_dataset(train_dataset)

        self.word_frequency = {}  # 단어 빈도수 기록
        self.training(train_dataset)
```

다음으로는 training 함수를 작성해 보았다. training 함수에서는 dataset에서의 각 단어들이 얼마나 자주 나오는지 학습을 진행한다. 코드는 다음과 같다.

```
def training(self, train_dataset):
        spam_prob = self.spam_prob()
        ham_prob = 1 - spam_prob

        for content, is_spam in train_dataset:
            words_set = tokenize(content)
```

```
            is_spam_prob = spam_prob
            is_ham_prob = ham_prob

            for word in words_set:
                if self.word_exists(word):
                    # Update word frequency
                    if word in self.word_frequency:
                        self.word_frequency[word] += 1 # 단어 빈도수 count
                    else:
                        self.word_frequency[word] = 1
```

마지막으로 이러한 학습 결과를 바탕으로 추론을 진행하는 inference 함수를 구현해 보았다. 만약 해당 단어가 word_frquency dictionary에 저장되어 있다면 해당 단어의 빈도수(빈도수는 dictionary에 count되어 저장되어 있을 것이다)를 가중치(weight)로서 사용해서 추론을 진행하도록 구현해 보았다.

```
def inference2(model, test_dataset):
    n_correct = 0
    spam_prob = model.spam_prob()
    ham_prob = 1-spam_prob
    for content, is_spam in test_dataset:
        words_set = tokenize(content)  # The set of words in an tested
email (i.e., inference)

        ###########################################################
        spam_prob = model.spam_prob()
        ham_prob = 1-model.spam_prob()
        is_spam_prob = spam_prob # initial value
        is_ham_prob = ham_prob

        for word in words_set:
            if model.word_exists(word):
                if word in model.word_frequency:
                    # Apply word frequency information
                    model.word_frequency[word] += 1
                    is_spam_prob *= model.spam_cond_prob(word) *
model.word_frequency[word]
                    is_ham_prob *= model.ham_cond_prob(word) *
model.word_frequency[word]
                else:
                    # Use default conditional probabilities if word
frequency is not available
                    model.word_frequency[word] = 1
                    is_spam_prob *= model.spam_cond_prob(word)
                    is_ham_prob *= model.ham_cond_prob(word)

        spam_prob = is_spam_prob
```

```
        ham_prob = is_ham_prob
        ##########################################################


        # If the probability of spam is higher than that of ham, then
we predict it as a spam
        is_spam_prediction = spam_prob >= ham_prob
        if is_spam_prediction == is_spam:
            n_correct += 1

    n_samples = len(test_dataset)
    accuracy = n_correct / n_samples
    print("Accuracy {} ({} / {})\n".format(accuracy, n_correct,
n_samples))
```

결과는 다음과 같았다.

```
Accuracy 0.9491626794258373 (1587 / 1672)
Accuracy (scikit-learn : MultinomialNB) 0.9796650717703349 (1638 /
1672)
Accuracy (scikit-learn : BernoulliNB) 0.976 6746411483254 (1633 /
1672)
Accuracy (scikit-learn : ComplementNB) 0.9617224880382775 (1608 / 1672)
```

학습을 시켜 학습 결과를 가중치(weight)로써 활용해 추론을 진행해 보았지만, 정확도(Accuracy)가 달라지지 않았음을 확인할 수 있다. 문제의 원인으로는 세 가지를 생각해볼 수 있다. 먼저 각 단어에 대한 빈도수 차이가 크지 않았다면 가중치가 추론 과정에서 큰 영향을 끼치지 못했을 것이다. 만약 이것이 문제의 주요 원인이었다면, 다른 파라미터를 가중치로써 사용함으로써 해결할 수 있다. 두 번째로는 첫 번째 implementation에서 이미 모델이 충분히 학습했을 수도 있다. 이 경우에는 추가 training이 영향을 끼치지 못했을 것이다. 마지막으로 데이터가 충분하지 않았을 수 있다. 데이터가 충분하지 않았다면 model이 충분히 학습하지 못해 상대적으로 낮은 정확도를 얻을 수밖에 없었을 것이다.

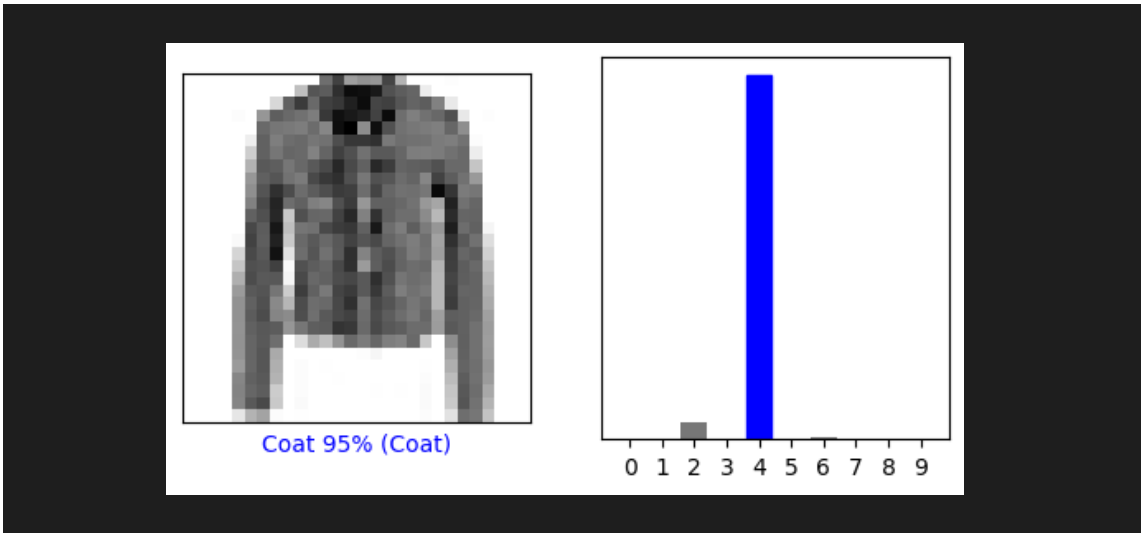
이 외에도 여러 문제들이 모델의 정확도 향상을 막았을 수 있다.

추가로 확률 소실을 막기 위한 로그 변환, 문맥 파악을 위한단어 시퀀스를 고려하는 모델 등을 사용하여 정확도를 높일 수도 있을 것이다.

3. Classification by deep neural networks

(a) (5pts) After training is done, plot output values of the network for an arbitrary image from test dataset.

```
i = 10
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```

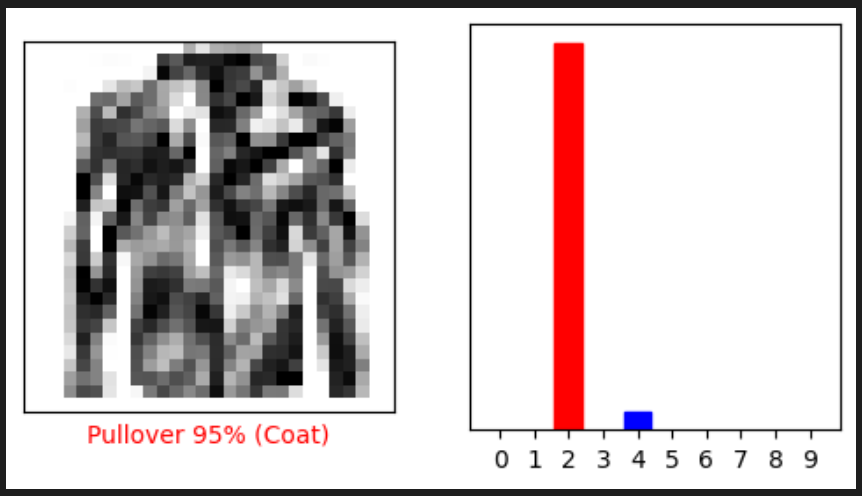


(b) (5pts) Explain the meaning of the output values.

Plot의 x축은 신뢰도(confidence)를 나타낸다. 왼쪽 image의 숫자는 신뢰도 퍼센트(100점 만점)을 나타낸다. (a)의 output에서는 4번 레이블 Coat의 신뢰도가 가장 높았고, 따라서 image를 Coat로 예측하였다.

(c) (5pts) Find a wrong prediction from test set and report values. Again, explain the meaning of output values.

```
i = 17
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```



마찬가지로 Plot의 x축은 신뢰도(confidence)를 나타낸다. 왼쪽 image의 숫자는 신뢰도 퍼센트(100

점 만점)을 나타낸다. 이 output에서는 2번 레이블 Pullover의 신뢰도가 가장 높았고, 따라서 image를 Pullover로 예측하였다. 하지만 실제 결과는 Coat였고, '잘못 예측된 것'임을 나타내기 위한 붉은색이 labeling된 것을 확인할 수 있다.

(d) (5pts) Please explain Flatten, Dense layer, Adam optimizer, and SparseCategoricalCross-entropy.

Flatten layer: 입력 데이터를 1차원으로 평탄화(flatten) 시켜주는 layer이다. 2차원, 혹은 3차원 feature map을 1차원 vector로 변환하여 이후 layer에서의 처리를 돕는다.

Dense layer: 입력 데이터와 출력 데이터를 모두 연결해주는 layer이다. 각각의 입력 데이터와 출력 데이터를 연결해주는 선은 가중치(weight)를 포함하고 있고, 이에 따라 출력이 결정된다.

Adam optimizer: saddle point에서의 문제를 해결하기 위한 Momentum과 step size 문제를 해결하기 위한 RMSdrop, 두 가지 개념이 합쳐진 optimizer이다.

Sparse/Categorical Cross-entropy: Sparse Cross-entropy는 데이터 lable이 int(정수형)로 되어 있을 때 사용하는 손실 함수(loss function)이다. 함수 내부에서 알아서 one-hot-vector encoding을 진행하여 연산을 수행한다. Categorical Cross-entropy는 데이터 lable이 one-hot-vector로 되어 있을 때 사용하는 손실 함수(loss function)이다.

(e) (10pts) Please replace MLP layers to convolutional layers of 3 × 3 support with the same channels, then retrain your network. How does the test accuracy change when the number of layers is equal to 1,2, and 3? Please analyze the result. Please save your model and submit it.

checkpoint는 모델의 가중치나 파라미터를 저장하고 있다. 학습 모델의 진행 상황을 특정 지점에서 보존하기 위해서 사용한다. 이번 assignment에서는 trained model을 저장하기 위해서 checkpoint를 활용하였다.

먼저 밑과 같은 코드를 작성하였다. checkpoint_path는 checkpoint가 저장될 path를 의미한다.

layer는 karas API의 Conv2D함수를 이용해서 정의해보았다. filter의 개수는 6개, kernel(filter)의 크기는 3X3, activation function은 relu로 설정하였고 마지막으로 padding은 적용하지 않았다. 코드는 밑과 같다.

(위의 파라미터들은 수업 PPT 23page를 참고해서 설정하였다)

```
checkpoint_path = r'C:\Users\User\Desktop\대학과제\2-2\인지기\HW1'
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create a callback that saves the model's weights
cp_callback =
tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                save_weights_only=True,
                                                verbose=1)

"layer == 1"
model_1 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters = 6, kernel_size = 3,
activation='relu', padding="same", input_shape=(28, 28, 1)),
    tf.keras.layers.Flatten(input_shape=(28, 28)),
```

```
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)])
#  # filter PPT 참고, kernel 3X3, 1D로 바뀌었으므로 shape 다시 지정
model_1.summary()

model_1.compile(optimizer='adam',
             loss=tf.keras.losses.SparseCategoricalCrossentropy(from_l
ogits=True),
             metrics=['accuracy'])

model_1.fit(train_images, train_labels, epochs=10,
callbacks=[cp_callback])

# save your model
model_1.save('3-(e)_1_layer_conv . hdf5')
```

결과는 다음과 같았다.

```
 _________________________________________________________________
 Epoch 1/10
 1875/1875 [==============================] - 7s 3ms/step - loss: 0.3943 - accuracy: 0.8586
 Epoch 2/10
 1875/1875 [==============================] - 6s 3ms/step - loss: 0.2688 - accuracy: 0.9022
 Epoch 3/10
 1875/1875 [==============================] - 7s 4ms/step - loss: 0.2194 - accuracy: 0.9181
 Epoch 4/10
 1875/1875 [==============================] - 6s 3ms/step - loss: 0.1825 - accuracy: 0.9326
 Epoch 5/10
 1875/1875 [==============================] - 7s 4ms/step - loss: 0.1504 - accuracy: 0.9439
 Epoch 6/10
 1875/1875 [==============================] - 6s 3ms/step - loss: 0.1223 - accuracy: 0.9554
 Epoch 7/10
 1875/1875 [==============================] - 6s 3ms/step - loss: 0.0990 - accuracy: 0.9635
 Epoch 8/10
 1875/1875 [==============================] - 6s 3ms/step - loss: 0.0819 - accuracy: 0.9699
 Epoch 9/10
 1875/1875 [==============================] - 6s 3ms/step - loss: 0.0656 - accuracy: 0.9775
 Epoch 10/10
 1875/1875 [==============================] - 6s 3ms/step - loss: 0.0547 - accuracy: 0.9808
```

두 번째 layer도 같은 방식으로 정의하였다. 결과는 다음과 같았다.

```
Epoch 1/10
1875/1875 [==============================] - 7s 3ms/step - loss: 0.4275 - accuracy: 0.8471
Epoch 2/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2866 - accuracy: 0.8947
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2327 - accuracy: 0.9139
Epoch 4/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1908 - accuracy: 0.9279
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1552 - accuracy: 0.9417
Epoch 6/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1248 - accuracy: 0.9542
Epoch 7/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0982 - accuracy: 0.9643
Epoch 8/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0762 - accuracy: 0.9719
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0595 - accuracy: 0.9782
Epoch 10/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0461 - accuracy: 0.9834
```

마지막 layer도 같은 방식으로 정의하였다. 결과는 다음과 같았다.

```
Epoch 1/10
1875/1875 [==============================] - 9s 4ms/step - loss: 0.3998 - accuracy: 0.8579
Epoch 2/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2542 - accuracy: 0.9074
Epoch 3/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1984 - accuracy: 0.9273
Epoch 4/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1540 - accuracy: 0.9432
Epoch 5/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1153 - accuracy: 0.9564
Epoch 6/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0810 - accuracy: 0.9698
Epoch 7/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0589 - accuracy: 0.9783
Epoch 8/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0423 - accuracy: 0.9844
Epoch 9/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0332 - accuracy: 0.9883
Epoch 10/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0273 - accuracy: 0.9901
```

(Epoch 10/10) layer가 1개일 때는 98.0%, 2개일 때는 98.3%, 3개일 때는 99.0%의 정확도를 보이고 있다. layer를 쌓을수록 모델은 더 많은 특징들을 학습하는 것이다.

하지만 layer를 6개 이상 쌓으면 오히려 정확도가 내려가는 걸 확인할 수 있다 (98.4%). 이는 아마 과적합이나 gradient 손실 문제가 발생했기 때문일 것이다. 이를 해결하기 위한 방법은 drop이나 activation function변경, optimizer 변경 등이 있다.

결과 검증을 위해 한 번 더 학습시켜 보았다. layer1: 97.4%, layer2: 98.6%, layer3: 99.0%로 결과는 위와 같은 양상을 띄었다. (trained model은 해당 결과를 얻은 model로 제출하였다)

참고: CNN에서 filter의 개수는 보통 layer가 늘어날수록 증가한다. 보다 나은 결과를 위해서는 이 파라미터를 바꿀 수도 있다.

(f) (10pts) Please retrain your 3 convolution layers model with different optimizers: adadelta, sgd, and rmsprop. Please analyze the results.

adadelta optimizer를 이용한 결과는 다음과 같다.

```
_________________________________________________________________
Epoch 1/10
1875/1875 [==============================] - 11s 4ms/step - loss: 4.2879 - accuracy: 0.3510
Epoch 2/10
1875/1875 [==============================] - 9s 5ms/step - loss: 1.6862 - accuracy: 0.5994
Epoch 3/10
1875/1875 [==============================] - 8s 4ms/step - loss: 1.2806 - accuracy: 0.6686
Epoch 4/10
1875/1875 [==============================] - 9s 5ms/step - loss: 1.0869 - accuracy: 0.7030
Epoch 5/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.9689 - accuracy: 0.7266
Epoch 6/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.8859 - accuracy: 0.7443
Epoch 7/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.8244 - accuracy: 0.7568
Epoch 8/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.7766 - accuracy: 0.7673
Epoch 9/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.7376 - accuracy: 0.7769
Epoch 10/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.7057 - accuracy: 0.7833
```

sgd optimizer를 이용한 결과는 다음과 같다.

```
_________________________________________________________________
Epoch 1/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.5348 - accuracy: 0.8163
Epoch 2/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.3428 - accuracy: 0.8762
Epoch 3/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2896 - accuracy: 0.8930
Epoch 4/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2530 - accuracy: 0.9051
Epoch 5/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2189 - accuracy: 0.9197
Epoch 6/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1898 - accuracy: 0.9297
Epoch 7/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1617 - accuracy: 0.9394
Epoch 8/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1382 - accuracy: 0.9481
Epoch 9/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1175 - accuracy: 0.9564
Epoch 10/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1028 - accuracy: 0.9613
```

RMSprop optimizer를 이용한 결과는 다음과 같다.

```
Epoch 1/10
1875/1875 [==============================] - 9s 4ms/step - loss: 0.6844 - accuracy: 0.8422
Epoch 2/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2965 - accuracy: 0.8939
Epoch 3/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2471 - accuracy: 0.9118
Epoch 4/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2139 - accuracy: 0.9232
Epoch 5/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1884 - accuracy: 0.9343
Epoch 6/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1668 - accuracy: 0.9420
Epoch 7/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1515 - accuracy: 0.9484
Epoch 8/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1393 - accuracy: 0.9532
Epoch 9/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1236 - accuracy: 0.9584
Epoch 10/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1161 - accuracy: 0.9617
```

RMSprop, sgd, adadelta 순으로 정확도가 높은 것을 확인할 수 있다. adadelta와 RMSprop은 adagrad(step size를 바꿔가며 탐색하는 기법)를 보완하기 위해 만들어진 기법(adadelta는 정지하는 것을 막도록, RMS drop은 맥락에 맞게 학습율을 조정하도록 설계되었다)이고, sgd는 기본 gradient descent의 비효율 문제(전체 데이터를 보고 loss를 계산하는 문제)를 해결하기 위해 만들어진 기법이다.

RMSprop과 sgd는 90%이상의 높은 정확도를 보이고 있다. 하지만 adadelta는 80%의 낮은 정확도를 보이고 있다. adadelta 함수내의 파라미터가 적합하지 않았을 수도 있고, 이 데이터에 adadelta 함수가 적합하기 않았을 수도 있다. 이 외에도 여러가지 문제가 낮은 정확도에 영향을 끼쳤을 수 있다. 우리는 다양한 optimizer를 적용해보며 상황에 따른 최적의 optimizer를 찾아야한다.

결과 검증을 위해 한 번 더 training시켜 보았다. RMSprop: 97.6%, sgd: 90.8%, adadelta: 77.8%로 결과는 위와 같은 양상을 띄었다. (trained model은 해당 결과를 얻은 model로 제출하였다)

4. Sentiment analysis by recurrent neural networks (RNN)

(a) (6pts) Explain the meaning of the output values

output은 정확도를 의미한다. 다시말해 train set을 바탕으로 학습한 model이 얼마나 test set을 잘 예측하는지를 말한다.

(b) (7pts) Please explain **the meaning of max words, embedding vector length, and in-put length** in 'Embedding(max words, embedding vector length, input length=max review length)'.

해당 용어들에 대한 정의를 하려면 word embedding 함수에 대해 알아야한다. word embedding 함수란 텍스트의 단어들을 model의 input 형태인 vector로 바꿔주는 함수이다.

max words: 이 argument는 몇 개에 단어에 대해 vector를 만들지를 정하는 역할을 한다. 이 코드는 5000개의 단어에 대해 embedding vector를 만든다.

embedding vector length: 이 argument는 함수의 출력 vector의 길이를 얼만큼 할 것인지 정해주는 역할을 한다. 이 코드는 32의 길이를 가진 vector로 단어들을 mapping해서 embedding vector를

만든다.

in-put length: 이 argument는 input의 길이를 얼마로 통일할 것인지를 정하는 역할을 한다. 이 코드는 500의 길이를 가진 input을 받는다.

(c) (7pts) Please **explain the meaning of in units, activation, and return sequences in** SimpleRNN(units=100, activation='tanh', return sequences=True).

SimpleRNN은 karas에서 제공해주는 함수로, 이전 출력이 다음 입력으로 완전히 연결된 model이다.

unit: 이 argument는 해당 layer가 가지는 뉴런의 개수를 의미한다.

activation: 이 arugument는 활성화 함수 종류를 의미한다. relu, tanh, sigmoid 등 다양한 비선형 함수들이 이 인자의 파라미터로써 사용될 수 있다.

sequences: 이 arugment는 마지막 은닉 상태만을 출력할지 모든 은닉 상태를 출력할지 결정하는 인자이다. True라면 모든 시점에서의 은닉 상태를 출력하고, False라면 마지막 시점에서의 은닉 상태만을 출력한다. RNN 층을 여러 개 쌓게 되었을 때는 이 인자를 True로 설정해야 한다. 만약 False로 설정한다면 이 전 RNN 층에서의 학습이 이 후 RNN 층으로 부분적으로만 전달되기 때문에 학습이 잘 되지 않는다.

(d) (8pts) Please change epoch of 3 to the epoch of 4, then retrain the model with one RNN(100) layer. How does the test accuracy change? and Why does the test accuracy increase (or decrease)? Please analyze the result. Please save your model and submit it.

해당 코드는 ipynp 파일에 첨부하였다.

epoch of 3에서는 64.3%였던 정확도가 epoch of 4에서는 83.9%로 상승하였다.

epoch를 늘리게 되면 model은 train set을 더 학습하게 된다. 보통 이에 따라 train set과 test set에 대한 학습률이 함께 증가하는 경향을 보인다. 밑의 사진에서는 epoch가 증가함에 따라 train set에 대한 accuaracy rate가 증가함을 확인할 수 있다. 보통의 경향처럼 test set에 대한 정확도도 함께 증가하는 모습을 보이고 있다.

```
Epoch 1/3
391/391 [==============================] - 77s 194ms/step - loss: 0.6467 - accuracy: 0.5963
Epoch 2/3
391/391 [==============================] - 74s 190ms/step - loss: 0.4933 - accuracy: 0.7626
Epoch 3/3
391/391 [==============================] - 73s 187ms/step - loss: 0.5182 - accuracy: 0.7414
<keras.src.callbacks.History at 0x7d3d61fd97e0>
```

```
Epoch 1/4
391/391 [==============================] - 81s 203ms/step - loss: 0.6179 - accuracy: 0.6297
Epoch 2/4
391/391 [==============================] - 79s 202ms/step - loss: 0.5380 - accuracy: 0.7380
Epoch 3/4
391/391 [==============================] - 78s 199ms/step - loss: 0.4233 - accuracy: 0.8137
Epoch 4/4
391/391 [==============================] - 80s 206ms/step - loss: 0.3673 - accuracy: 0.8420
<keras.src.callbacks.History at 0x7d3d630b3c10>
```

만약 train set에 대한 학습이 과하게 진행된다면 model이 train set의 noise까지 학습하게 된다. 이

문제를 overfitting(과적합)이라고 하는데, epoch 4까지는 이러한 문제가 발생하지 않았음을 확인할 수 있다. 만약 발생했다면 test set에 대한 정확도가 떨어졌을 것이다.

(e) (8pts) Please change epoch of 3 to the epoch of 5, then retrain the model with two RNN(100) layers. How does the test accuracy change? and Why does the test accuracy increase (or decrease)? Please analyze the result. Please save your model and submit it.

해당 코드는 ipynp 파일에 첨부하였다.

epoch of 5에서는 accuracy rate가 72.4%로 epoch of 4에 비해서 낮아진 것을 확인할 수 있다. 또한 밑 사진에서와 같이 train set에 대한 accuracy rate도 낮아졌음을 확인할 수 있다.

```
Epoch 1/5
391/391 [==============================] - 181s 464ms/step - loss: 0.7015 - accuracy: 0.5326
Epoch 2/5
391/391 [==============================] - 179s 457ms/step - loss: 0.6911 - accuracy: 0.5345
Epoch 3/5
391/391 [==============================] - 180s 462ms/step - loss: 0.6660 - accuracy: 0.5776
Epoch 4/5
391/391 [==============================] - 179s 459ms/step - loss: 0.5695 - accuracy: 0.6990
Epoch 5/5
391/391 [==============================] - 178s 456ms/step - loss: 0.5414 - accuracy: 0.7234
<keras.src.callbacks.History at 0x7d3d6561a9b0>
```

epoch, layer중 어느 파라미터가 문제의 원인인지 확인하기 위해 1 layer and epoch of 5 model과 2 layer and epoch of 3 model을 추가로 만들어보았다. 순서대로 1 layer and epoch of 5 model, 2 layer and epoch of 3 model의 train set 학습 결과이다. test set에 대한 accuracy는 각각 79.8%, 63.9%으로 나왔다.

```
Epoch 1/5
391/391 [==============================] - 74s 189ms/step - loss: 0.3362 - accuracy: 0.8581
Epoch 2/5
391/391 [==============================] - 90s 230ms/step - loss: 0.3243 - accuracy: 0.8604
Epoch 3/5
391/391 [==============================] - 84s 216ms/step - loss: 0.4519 - accuracy: 0.7779
Epoch 4/5
391/391 [==============================] - 78s 199ms/step - loss: 0.3455 - accuracy: 0.8546
Epoch 5/5
391/391 [==============================] - 78s 200ms/step - loss: 0.2825 - accuracy: 0.8867
<keras.src.callbacks.History at 0x7f9fa4a2af50>
```

```
Epoch 1/3
391/391 [==============================] - 162s 407ms/step - loss: 0.7067 - accuracy: 0.5054
Epoch 2/3
391/391 [==============================] - 160s 409ms/step - loss: 0.6458 - accuracy: 0.6119
Epoch 3/3
391/391 [==============================] - 159s 406ms/step - loss: 0.5586 - accuracy: 0.7050
<keras.src.callbacks.History at 0x7f9fa0ccfd90>
```

1 layer and epoch of 5 model은 1 layer and epoch of 4 model 보다 accuacy가 향상 되었다(epoch의 영향을 평가하기 위한 비교). 반면 2 layer and epoch of 3 model은 1 layer and epoch of 3 model 보다 accuracy가 떨어졌다(추가 layer의 영향을 평가하기 위한 비교). 위 결과는 epoch 5까지는 overfitting 문제가 일어나지 않음을 의미한다. 따라서 우리는 이를 통해 추가의 layer가 model의 학습에 영향을 끼쳤다고 예상해볼 수 있다.

2 layer and epoch of 3 model의 train set의 학습 결과를 살펴보자. epoch3/3 결과를 보면 70%인 것

을 확인할 수 있다. 이는 2 layer에서는 학습 자체가 잘 되지 않았음을 의미한다.

layer가 많아질수록 model은 더 많은 데이터를 필요로한다. 이 dataset은 layer 2개 만큼에 깊이로 학습하기에는 양이 충분하지 않았기 때문에 이러한 결과가 나오게 된 것이라 해석할 수 있다.

우리는 학습을 진행할 때 해당 model의 optimal depth(적절한 layer의 깊이)를 잘 찾아야한다.

(f) (8pts) Please replace the model with one RNN(100) layer to that with one RNN(50) layer, then retrain your network with the epoch of 3. How does the test accuracy change? and Why does the test accuracy increase (or decrease)? Please analyze the result. Please save your model and submit it.

해당 코드는 ipynp 파일에 첨부하였다.

결과 분석을 위해 RNN(100) layer와 RNN(50) layer의 출력 결과를 비교해보자. train set에 대한 accuracy(epoch 3/3기준)는 각각 74.1%와 70.6%로 RNN(100)과 RNN(50)이 비슷했고, 마찬가지로 test set에 대한 accuracy도 각각 64.3%와 65.1%로 RNN(100)과 RNN(50)이 비슷했다.

```
Epoch 1/3
391/391 [==============================] - 60s 153ms/step - loss: 0.6231 - accuracy: 0.6415
Epoch 2/3
391/391 [==============================] - 59s 150ms/step - loss: 0.4595 - accuracy: 0.7925
Epoch 3/3
391/391 [==============================] - 58s 149ms/step - loss: 0.5684 - accuracy: 0.7067
<keras.src.callbacks.History at 0x7d3d59202650>
```

일반적으로 뉴런의 개수가 많을수록 더 많은 정보를 학습할 수 있다. 그러나 뉴런의 수가 input data보다 월등히 많아진다면 model이 많은 정보를 학습하지 못하는 경향을 보일 수 있다. 이 코드에서도 마찬가지의 모습을 보인다. 이 model의 input은 32이다. RNN(100)은 input(32)에 비해 너무 많은 뉴런을 가지고 있다. 따라서 model은 이를 모두 학습하지 못하여 뉴런의 개수가 두 배 적은 RNN(50)과 비슷한 결과를 야기시켰을 것이다.

(g) (8pts) Please replace the model with one RNN(50) layer to that with two RNN(50) layers, then retrain your network. How does the test accuracy change? and Why does the test accuracy increase (or decrease)? Please analyze the result. Please save your model and submit it.

해당 코드는 ipynp 파일에 첨부하였다.

RNN(50) and 1 layer에서는 65.1%였던 정확도가 RNN(50) and 2 layer에서는 64.2%로 하락했음을 확인할 수 있다.

```
Epoch 1/3
391/391 [==============================] - 146s 335ms/step - loss: 0.5560 - accuracy: 0.6938
Epoch 2/3
391/391 [==============================] - 128s 327ms/step - loss: 0.4448 - accuracy: 0.7891
Epoch 3/3
391/391 [==============================] - 127s 324ms/step - loss: 0.4900 - accuracy: 0.7633
<keras.src.callbacks.History at 0x7d3d65c15de0>
```

우리는 (e) (RNN(100))에서 1 layer and epoch of 3 model과 2 layer and epoch of 3 model을 비교했을 때, 각 data set에 대한 1 layer and epoch of 3 model의 accuracy가 더 높았던 것을 확인할 수 있었다(추가 layer가 정확도 하락을 야기시키는 것을 확인할 수 있었다). RNN(50)에서도 이와 같은 경향을 확인할 수 있다. 해당 dataset은 layer 2개 만큼에 depth로 학습하기에는 양이 충분하지 않았다.

(h) (8pts) Please replace the model with one RNN(100) layer to that with one LSTM(100) layer with the epoch of 3, then retrain your network. How does the test accuracy change? and Why does the test accuracy increase (or decrease)? Please analyze the result. Please save your model and submit it.

해당 코드는 ipynp 파일에 첨부하였다.

LSTM(Long Short-Term Memory Network)은 simpleRNN 의 '장기 문맥 의존성' 문제를 해결하기 위해 고안된 model 이다. 정확도가 81.3%로 아주 높게 향상된 것을 확인할 수 있다.

```
Epoch 1/3
391/391 [==============================] - 282s 716ms/step - loss: 0.4648 - accuracy: 0.7727
Epoch 2/3
391/391 [==============================] - 279s 714ms/step - loss: 0.2839 - accuracy: 0.8878
Epoch 3/3
391/391 [==============================] - 277s 708ms/step - loss: 0.2369 - accuracy: 0.9070
<keras.src.callbacks.History at 0x7d3d63adb640>
```

5. Assignment Result (Realization)

지금까지 여러 model들을 training 시키고 test set으로 accuracy를 분석해보며 어떤 방법이 적합할지에 대해서 알아보았다. 정확도 향상과 경향성 검증을 위해 같은 model을 여러 번 학습시켜 보기도 하고, 다른 컴퓨터에서 학습시켜 보기도 하였는데, 정확도 뿐만 아니라 경향성도 매번 바뀌는 것을 확인할 수 있었다. 아마 model의 train set, test set, model의 black box에서 일어나는 일 등이 영향을 끼쳤을 것이다. model에 맞는 layer의 depth, number of unit 등을 선택하는 것도 중요하겠지만 계속 학습시키며 최선의 model을 구축하는 것도 인공지능을 제작하는데 중요한 스킬 중 하나이다.