Tr.json, Te.json 데이터를 이용하여 문서 분류 모델 만들기

목차

Json 파일 읽어오기

제가 가장 먼저 한 작업은 Json 데이터를 읽어온 것입니다. 어떠한 데이터가 들어있는지 아는 것이 먼저였고 데이터 요소들마다 어떤 특징을 가지고 있는지를 알기 위해서 아래와 같이 Json 데이터를 읽었습니다. Json 파일을 그냥 읽을 경우에는 유니코드 에러가 발생하여 아래와 같이 인코딩과 디코딩 코드를 추가해줬습니다.

```
In [2]: #-*- coding:utf-8 -*-
import json
import pandas as pd
from pprint import pprint
fw = open("test.txt",'w',encoding = 'utf-8')

with open('Tr.json',encoding="iso-8859-1") as f:
    json_data = json.load(f,encoding = "utf-8")
    for i in range(len(json_data)):
        for j in json_data[i]['sentence']:
            print(j.encode('iso-8859-1').decode('euc-kr','ignore'))

with open('Te.json',encoding="iso-8859-1") as f:
    json_data2 = json.load(f,encoding = "utf-8")
    for i in range(len(json_data2)):
        for j in json_data2[i]['sentence']:
            print(j.encode('iso-8859-1').decode('euc-kr','ignore'))
```

```
Re/SL+:/SF [/SF+질문/NNG+]/SF+licq/SL+설치/NNG 방법/NNG+(/SF+RPM/SL+방법/NNG+이/JKS 아니/VCN+ㄴ/ETM+./SF+./SF+./SF+tar/SL+./SF+gz/SL+파일/NNG 받/VV+아/EC+서/JKB+)/SF

./SF+./SF

일반/NNG+적/XSN+으로/JKB 소스/NNG+를/JKO 이용/NNG+해/NNG+서/JKB 리눅스/NNP 프로그램/NNG+을/JKO 설치/NNG+하/XSV+ㄹ/ETM 때/NNG+는/JX

./SF+.//SF+configure/SL

make/SL

make/SL install/SL

이럴/VA+게/EC 세/MM 명령/NNG+을/JKO 차례/NNG+대/XSN+로/JKB 실행/NNG+해/NNG 주/VX+면/EC 대부분/NNG 되/VV+더군요/EF+./SF

그리/VV+고/EF+./SF+./SF+./SF
```

위와 같이 형태소 tagging 이 된 문장이 출력이 되었습니다. 그러나 'sentence 라는 컬럼에 해당하는 요소 말고도 answer 에 해당하는 요소도 한꺼번에 쉽게 볼 수 있는 방법이 없을까' 라고 고민하던 중 Pandas 라는 유용한 라이브러리를 찾게 됩니다.

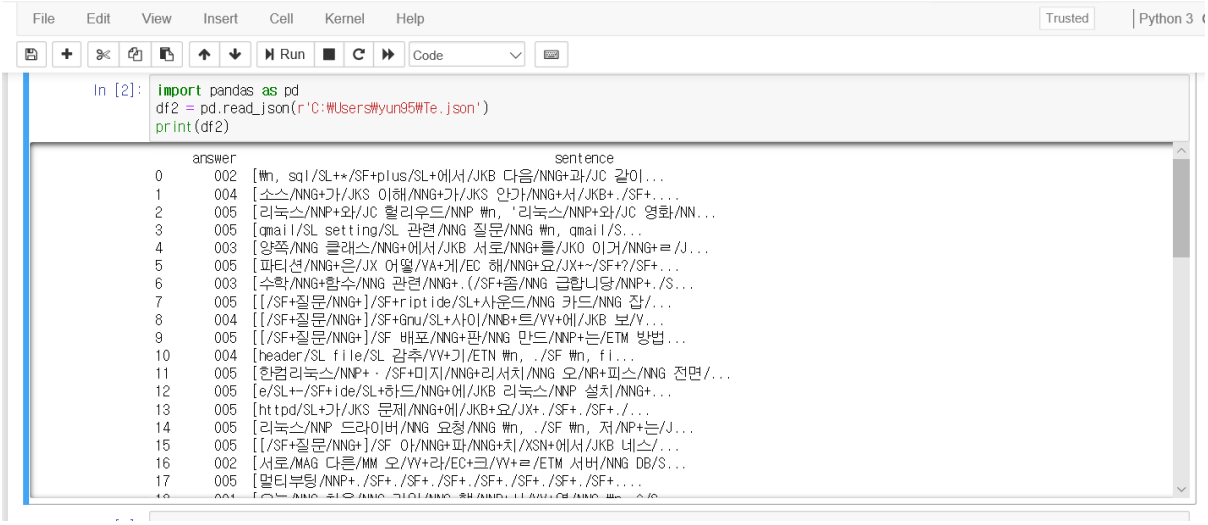
Pandas 란?

Pandas 는 파이썬에서 사용하는 데이터분석 라이브러리로, 행과 열로 이루어진 데이터 객체를 만들어 다룰 수 있는 매우 편리한 도구입니다.

Pandas 라이브러리를 이용해서 Tr.json 과 Te.json 을 읽어봤습니다.

```
In [1]: import pandas as pd
        df = pd.read_json(r'C:₩Users₩yun95₩Tr.json')

In [2]: print(df)
```

	answer	sentence
0	005	[Re/SL+:/SF [/SF+질문/NNG+]/SF+licq/SL+설치/NNG 방법...
1	004	[Re/SL+:/SF [/SF+질문/NNG+]/SF C/SL+로/JKB ORACLE...
2	005	[Re/SL+:/SF 나/NP+만/JX+의/JKG 배포/NNG+판/NNG+을/JKO...
3	004	[Re/SL+:/SF 너무/MAG 쉬/VV+운/NNG+것/NNB 같/VA+은데/EF...
4	004	[Perl/SL cgi/SL 에/NNG+서/JKB request/SL 쪽/NNB+의...
5	003	[Re/SL+:/SF CList/SL in/SL a/SL CList/SL+?/SF ...
6	004	[Re/SL+:/SF [/SF+질문/NNG+]/SF exit/SL 에/NNG 관하/...
7	003	[CreateWindow/SL+로/JKB 버튼/NNG+을/JKO 만드/NNP+는데/...
8	001	[[/SF+추천/NNG+부탁/NNG+]/SF 음식점/NNG+에서/JKB 들/XSN+...
9	005	[Re/SL+:/SF [부팅/NNP+에러/NNG+]/SF+루트/NNG 파일/NNG ...
10	005	[Re/SL+:/SF [/SF+문/NNG+]/SF Kernel/SL 2.4.4/SN...
11	001	[Re/SL+:/SF 이/MM 노래/NNG 아/NNG+시/XSN+는/JX 분/NNB...
12	004	[Re/SL+:/SF dos/SL+용/XSN c/SL++/SF++/SF 에/NNG+...
13	004	[Re/SL+:/SF [/SF+질문/NNG+]/SF 채팅/NNP+프로그램/NNG 중...
14	005	[Re/SL+:/SF gcc/SL+-/SF+3.0/SN+이/JKC 나왔/NNP+습니...
15	003	[[/SF+질문/NNG+]/SF 다음/NNG 처리/NNG+를/JKO 할/NNG+려고...
16	004	[Re/SL+:/SF 자바/NNP+에서/JKB+./SF+./SF+이거/NNG+ㄹ/J...
17	004	[Re/SL+:/SF c/SL+로/JKB 짜/VV+ㄴ/ETM cgi/SL+에서/JK...
18	005	[MD/SL device/SL+?/SF ₩n, ?/SF ₩n, MD/SL+가/JKS...
19	004	[Re/SL+:/SF [/SF+질문/NNG+]/SF+perl/SL+을/JKO cro...
20	005	[Re/SL+:/SF X/SL 설정/NNG+에/JKB 대/XPN+해서/NNG+./S...
21	001	[[/SF+질문/NNG+]/SF ram/SL+화일/NNG+의/JKG 음악/NNG+이...
22	005	[Re/SL+:/SF ip/SL+를/JKO 고정/NNG+ip/SL+에서/JKB dh...
23	003	[[/SF+질문/NNG+]/SF VC/SL++/SF++/SF+에/NNG+서/JKB ...
24	005	[Re/SL+:/SF 리눅스/NNP+에서/JKB 플래쉬/NNP 보/VV+는/ETM ...
25	004	[Re/SL+:/SF 이/VCP+게/EC 뭐/NP+가/JKS 틀린거져/NNP+?/S...
26	004	[Re/SL+:/SF [/SF+질문/NNG+]/SF+system/SL+(/SF+)/...
27	002	[Developer/SL+로/JKB 내/NP+컴퓨터/NNG+의/JKG 서버/NNG+...



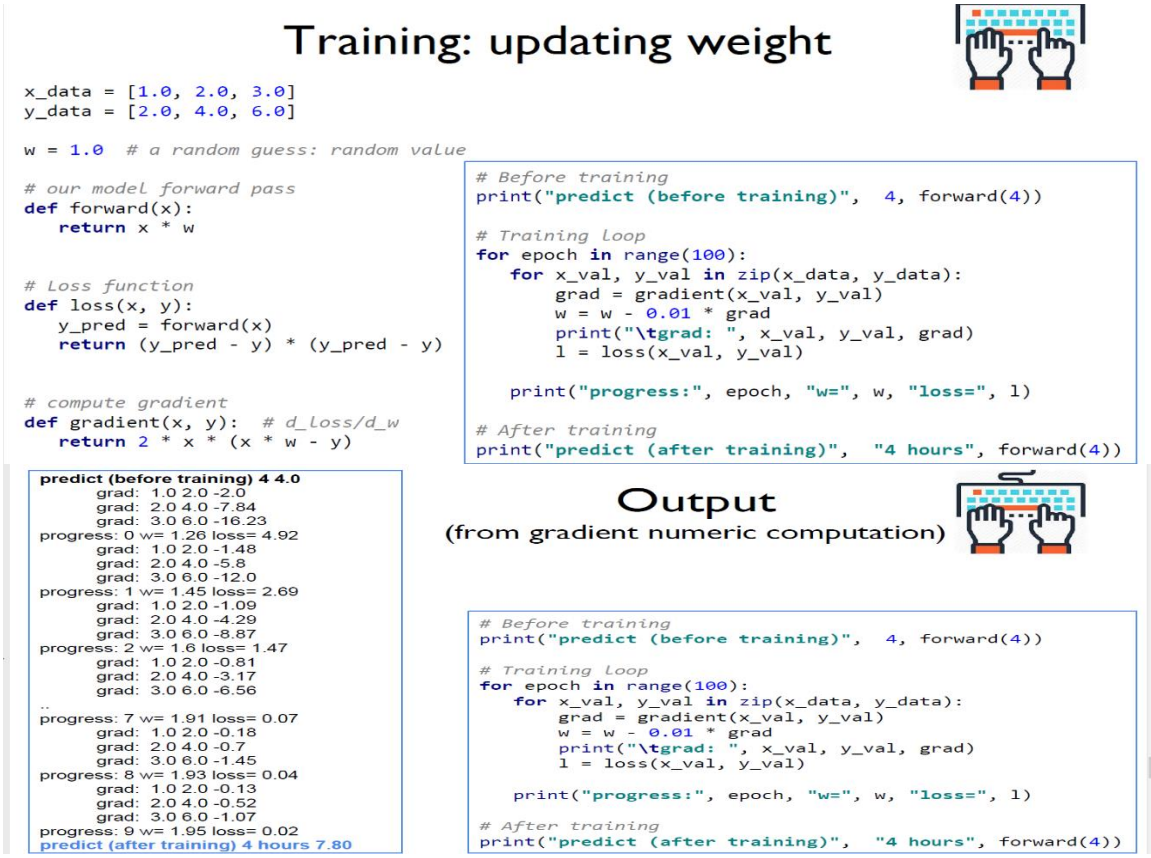
	answer	sentence
0	002	[₩n, sql/SL+*/SF+plus/SL+에서/JKB 다음/NNG+과/JC 같이...
1	004	[소스/NNG+가/JKS 이해/NNG+가/JKS 안가/NNG+서/JKB+./SF+....
2	005	[리눅스/NNP+와/JC 헐리우드/NNP ₩n, '리눅스/NNP+와/JC 영화/NN...
3	005	[qmail/SL setting/SL 관련/NNG 질문/NNG ₩n, qmail/S...
4	003	[양쪽/NNG 클래스/NNG+에서/JKB 서로/NNG+를/JKO 이거/NNG+ㄹ/J...
5	005	[파티션/NNG+은/JX 어떻/VA+게/EC 해/NNG+요/JX+~/SF+?/SF+...
6	003	[수학/NNG+함수/NNG 관련/NNG+.(/SF+좀/NNG 급합니당/NNP+./S...
7	005	[[/SF+질문/NNG+]/SF+riptide/SL+사운드/NNG 카드/NNG 잡/...
8	004	[[/SF+질문/NNG+]/SF+Gnu/SL+사이/NNB+트/VV+에/JKB 보/V...
9	005	[[/SF+질문/NNG+]/SF 배포/NNG+판/NNG 만드/NNP+는/ETM 방법...
10	004	[header/SL file/SL 감추/VV+기/ETN ₩n, ./SF ₩n, fi...
11	005	[한컴리눅스/NNP+ · /SF+미지/NNG+리서치/NNG 오/NR+피스/NNG 전면/...
12	005	[e/SL+-/SF+ide/SL+하드/NNG+에/JKB 리눅스/NNP 설치/NNG+...
13	005	[httpd/SL+가/JKS 문제/NNG+에/JKB+요/JX+./SF+./SF+./...
14	005	[리눅스/NNP 드라이버/NNG 요청/NNG ₩n, ./SF ₩n, 저/NP+는/J...
15	005	[[/SF+질문/NNG+]/SF 아/NNG+파/NNG+치/XSN+에서/JKB 네스/...
16	002	[서로/MAG 다른/MM 오/VV+라/EC+크/VV+ㄹ/ETM 서버/NNG DB/S...
17	005	[멀티부팅/NNP+./SF+./SF+./SF+./SF+./SF+./SF+....

위와 같이 각 sentence 마다 answer 의 값이 지정이 된 것을 알 수 있었고 sentence 의 경우 리스트의 형태로 담겨있습니다. 그리고 answer 의 경우 '001'부터 '005'까지 총 다섯가지로 분류되어 있습니다.

성킴 교수님 강의 분석 및 자연어 처리 분석

Json 파일을 읽고 난 후 두 가지 사실을 알게 되었습니다. 첫번째로 sentence 컬럼은 문자열로 구성된 리스트 요소를 가지고 있고 두번째로 answer 컬럼의 경우 총 다섯가지의 문자열로 분류되어 있습니다. 이를 통해서 sentence 의 문자열을 이용하여 다섯가지 answer 를 예측하는 것이 최종 목표라는 사실을 알게 되었습니다.

이를 수행하기 위해서 성킴 교수님의 강의를 우선 시청하였습니다.
성킴 교수님의 강의에서는 파이썬과 파이토치로 퍼셉트론을 구현하였다는 사실을 알게 되었습니다. 아래와 같이 파이썬의 경우 가중치를 직접 갱신하고 편미분도 식에 의해서 직접 구현한 것을 볼 수 있습니다. 학습하는 함수, loss 값을 구하는 함수, 편미분하는 함수 등 모두 책에서 봤던 식이라 그런지 이해하기 쉬웠으며 생각보다 학습도 잘되어서 loss 가 최소가 되는 가중치도 쉽게 찾을 수 있었습니다.



In [1]:

```
import torch
from torch.autograd import Variable
import torch.nn.functional as F

x_data = Variable(torch.Tensor([[1.0], [2.0], [3.0], [4.0]]))
y_data = Variable(torch.Tensor([[0.], [0.], [1.], [1.]]))


class Model(torch.nn.Module):

    def __init__(self):
        """
        In the constructor we instantiate nn.Linear module
        """
        super(Model, self).__init__()
        self.linear = torch.nn.Linear(1, 1)  # One in and one out

    def forward(self, x):
        """
        In the forward function we accept a Variable of input data and we must return
        a Variable of output data.
        """
        y_pred = F.sigmoid(self.linear(x))
        return y_pred

# our model
model = Model()


# Construct our loss function and an Optimizer. The call to model.parameters()
# in the SGD constructor will contain the learnable parameters of the two
# nn.Linear modules which are members of the model.
criterion = torch.nn.BCELoss(size_average=True)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

# Training loop
for epoch in range(1000):
        # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x_data)

    # Compute and print loss
    loss = criterion(y_pred, y_data)
    print(epoch, loss.data[0])

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# After training
hour_var = Variable(torch.Tensor([[1.0]]))
print("predict 1 hour ", 1.0, model(hour_var).data[0][0] > 0.5)
hour_var = Variable(torch.Tensor([[7.0]]))
print("predict 7 hours", 7.0, model(hour_var).data[0][0] > 0.5)
```

```
983 tensor(0.3768)
984 tensor(0.3767)
985 tensor(0.3767)
986 tensor(0.3766)
987 tensor(0.3765)
988 tensor(0.3764)
989 tensor(0.3763)
990 tensor(0.3762)
991 tensor(0.3761)
992 tensor(0.3760)
993 tensor(0.3759)
994 tensor(0.3758)
995 tensor(0.3758)
996 tensor(0.3757)
997 tensor(0.3756)
998 tensor(0.3755)
999 tensor(0.3754)
predict 1 hour  1.0 tensor(0, dtype=torch.uint8)
predict 7 hours 7.0 tensor(1, dtype=torch.uint8)
```

파이토치 코드를 직접 실행하였고 놀라운 사실을 알 수 있었습니다. 파이썬의 경우 모든 함수를 직접 수식으로 구현하였는데 파이토치의 경우 nn 패키지를 사용하여 모델을 구현하였고 가중치를 자동으로 갱신해주는 optimizer 패키지가 있었고 SGD 와 같은 최적화 알고리즘 또한 제공해주었습니다. 파이토치를

이용한다면 파이토치의 패키지와 API 를 이용하여 쉽고 빠르고 딥러닝 모델을 구현할 수 있었습니다.

성킴 교수님의 강의를 듣고 딥러닝 모델이 어떤 방법으로 구현되었는지 직접 경험할 수 있었지만 트레이닝 데이터로 들어간 값이 매우 간단한 숫자였고 이 과제의 가장 큰 산은 바로 자연어 처리라는 것을 알게 되었습니다. 그 이유는 컴퓨터는 0 과 1 로 구성되어 있어서 문자열을 숫자로 바꾼 후에 학습이 되기 때문입니다. 더군다나 트레이닝 데이터에는 영어 뿐만 아니라 특수문자, 숫자, 한글 등 다양한 문자열이 들어가 있어서 아스키코드로의 변환도 불가능 하였습니다. 다행히도 자연어 처리를 학습할 수 있는 다양한 사이트와 유튜브가 있었고 그를 토대로 차근차근 진행하였습니다.

먼저 KoNLPy 라는 한글 데이터를 전처리 해주는 패키지를 사용하여 실습을 진행해봤습니다. Pandas 를 이용하여 트레이닝 데이터의 answer 001 에 해당하는 모든 요소를 Tr_1_answer_content 에 담아주었고 Twitter 라는 Konlpy 의 패키지를 이용하여 문자열의 명사만 추출하는 함수인 nlpy.nouns 를 실행하였습니다.
아래와 같이 한글의 명사에 해당하는 단어가 모두 추출되었고
빅 데이터 분석에 많이 쓰이는 워드클라우드 패키지를 이용해봤습니다.
단어 빈도수가 높을수록 단어의 크기가 크게 나왔고 빈도수가 작을수록 단어의 크기는 작게 나오는 것을 알 수 있습니다.

In [1]:
```
import re
import pandas as pd
Tr = pd.read_json(r'C:₩Users₩yun95₩Tr.json')
Tr_1_answer = Tr.loc[(Tr['answer'] == '001')]
Tr_1_answer_content = str(Tr_1_answer['sentence'])
Tr_1_answer_content = re.sub('[-+=.#/?:$}A-Z₩n]', '',Tr_1_answer_content)
```

In [2]:
```
from konlpy.tag import Twitter

nlpy = Twitter()

nouns = nlpy.nouns(Tr_1_answer_content)

print(nouns)

from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
%matplotlib inline

def displayWordCloud(data = None, backgroundcolor = 'white', width=800, height=600 ):
    wordcloud = WordCloud(
                        font_path = '/Library/Fonts/NanumGothic.ttf',
                        stopwords = STOPWORDS,
                        background_color = backgroundcolor,
                         width = width, height = height).generate(data)
    plt.figure(figsize = (15 , 10))
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.show()
%time displayWordCloud(' '.join(nouns))
```

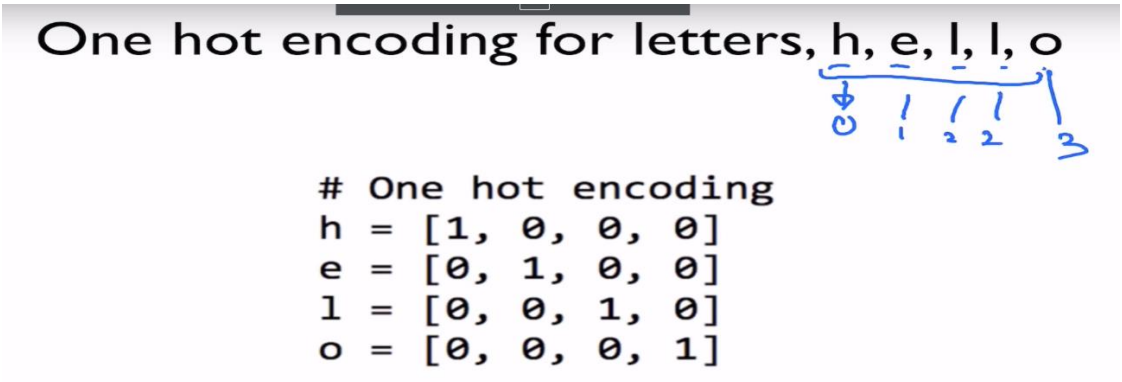
answer 001 클래스의 경우 노래, 음악, 질문에 관한 단어가 제일 많이 들어있다는 것을 알게 되었고 nlpy.phrase 함수를 이용하여 구(phrase)로 단어를 추출하여 워드클라우드를 추출하였습니다

```
In [3]: from konlpy.tag import Twitter

nlpy = Twitter()

phrases = nlpy.phrases(Tr_1_answer_content)

print(phrases)  #영어랑 숫자까지 되서 좋음

from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
%matplotlib inline

def displayWordCloud(data = None, backgroundcolor = 'white', width=800, height=600 ):
    wordcloud = WordCloud(
                        font_path = '/Library/Fonts/NanumGothic.ttf',
                        stopwords = STOPWORDS,
                        background_color = backgroundcolor,
                         width = width, height = height).generate(data)
    plt.figure(figsize = (15 , 10))
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.show()
%time displayWordCloud(' '.join(phrases))
```

숫자와 영어가 포함되니 처음에 했던 워드클라우드와는 전혀 다른 단어의 빈도를 보여주었습니다. 이를 통해서 텍스트로 이루어진 데이터를 어떤 방법으로 정제하는지에 따라서 결과가 많이 달라질 수 있다는 사실을 알게 되었습니다.

위의 실습을 통해서 데이터를 정제하는 연습을 하였고 데이터를 숫자로 바꾸는 방법이 어떠한 것이 있는지 분석하였습니다. 대표적으로 One-hot 인코딩, 워드넷, TF-IDF 방식이 있었고 그 중 가장 이해하기 쉬웠던 One-hot 방식을 택하게 되었습니다. One hot 인코딩 방식은 쉽게 말해 해당하는 값만 1 로 표시하고 나머지는 0 으로 표시하는 방식입니다.

아래 사진은 텍스트 -> 숫자로 바꾸는 One-hot 인코딩 예시입니다.
보시는 바와 같이 "hello"라는 문자열을 문자 형태로 쪼개서 h,e,l,l,o 에 각각 0 1 2 2 3 이라고 인덱스 번호를 붙인 후 h = 0 번이므로 0 번만 1 로 표시하고 0 번을 제외한 나머지는 다 0 으로 표시를 합니다. Ex) h=[1,0,0,0]
나머지 문자도 똑같은 방식으로 인코딩하면 아래와 같이 벡터로 변환된 것을 볼 수 있습니다.



자연어를 처리하고 벡터로 변환하는 것을 위의 실습들을 통해서 알아보았고 딥러닝 모델을 선정하고 구현하는 과정에 대해서 말씀드리겠습니다.

일단 첫번째 시도로 성킴 교수님이 구현하신 아래와 같은 기본 퍼셉트론 모델을 사용하려고 하였으나 문자가 아닌 간단한 숫자를 분류한 모델이라 변형하여 적용시키기 어려웠습니다. 그래서 문자열을 다룬 다른 모델을 찾아봤습니다.

In [1]:

```
import torch
from torch.autograd import Variable
import torch.nn.functional as F

x_data = Variable(torch.Tensor([[1.0], [2.0], [3.0], [4.0]]))
y_data = Variable(torch.Tensor([[0.], [0.], [1.], [1.]]))


class Model(torch.nn.Module):

    def __init__(self):
        """
        In the constructor we instantiate nn.Linear module
        """
        super(Model, self).__init__()
        self.linear = torch.nn.Linear(1, 1)  # One in and one out

    def forward(self, x):
        """
        In the forward function we accept a Variable of input data and we must return
        a Variable of output data.
        """
        y_pred = F.sigmoid(self.linear(x))
        return y_pred

# our model
model = Model()


# Construct our loss function and an Optimizer. The call to model.parameters()
# in the SGD constructor will contain the learnable parameters of the two
# nn.Linear modules which are members of the model.
criterion = torch.nn.BCELoss(size_average=True)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

# Training loop
for epoch in range(1000):
    # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x_data)

    # Compute and print loss
    loss = criterion(y_pred, y_data)
    print(epoch, loss.data[0])

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# After training
hour_var = Variable(torch.Tensor([[1.0]]))
print("predict 1 hour ", 1.0, model(hour_var).data[0][0] > 0.5)
hour_var = Variable(torch.Tensor([[7.0]]))
print("predict 7 hours", 7.0, model(hour_var).data[0][0] > 0.5)
```

문서 분류 모델 만들기

Tutorials > 문자-단위 RNN으로 이름 분류하기

NOTE

Click here to download the full example code

문자-단위 RNN으로 이름 분류하기

Author: Sean Robertson

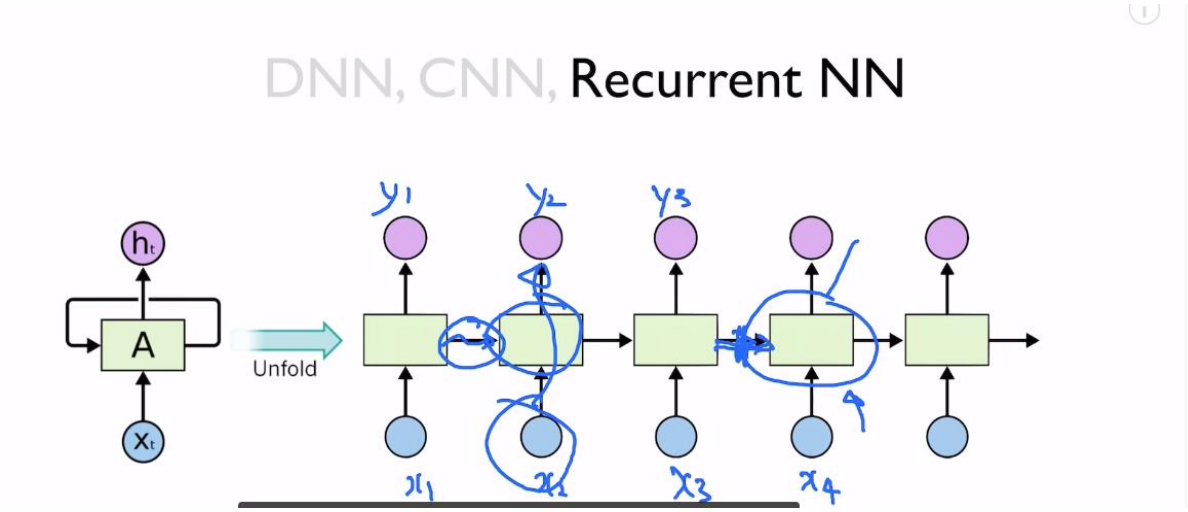
번역: 황성수

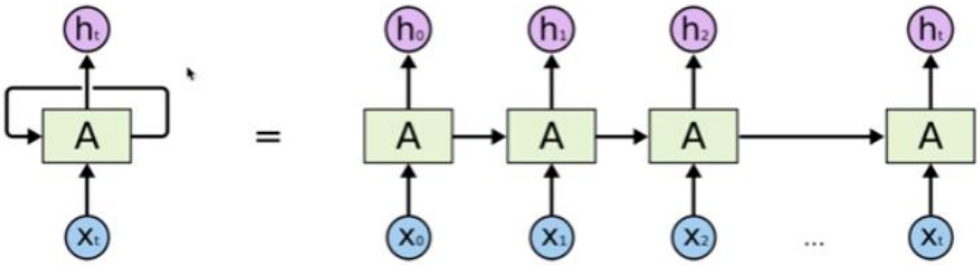
단어를 분류하기 위해 기본적인 문자-단위 RNN을 만들고 훈련 할 것입니다. 문자-단위 RNN은 문자의 연속을 읽어 들여서 각 단계의 예측과 "은닉 상태(Hidden State)" 출력하고 다음 단계에 이전 은닉 상태를 전달합니다. 단어가 속한 클래스와 같은 출력이 되도록 최종 예측으로 선택합니다.

구체적으로, 18개 언어로 된 수천 개의 성(姓)을 훈련시키고, 철자에 따라 이름이 어떤 언어인지 예측합니다:

다행히도 수업시간에 이용하였던 파이토치 튜토리얼 사이트에 위와 같은 RNN 모델 튜토리얼이 존재하였고 위 튜토리얼을 변형해서 만들기로 결정하였습니다. (나중에 알고 보니 성킴 교수님 강의에도 위와 같은 RNN 모델을 구현한 강의가 존재하였습니다)

튜토리얼을 따라하고 변형하기 전 RNN 이 무엇인지에 대해서 조사하였습니다.





RNN(Recurrent Neural Network)은 Neural Network 중 한 종류로서 연속된 데이터를 처리하는 모델입니다. 이런 데이터에는 음성인식, 자연어처리 등이 포함됩니다. 자연어의 경우 단어 하나만 안다고 해서 처리될 수 없고, 앞뒤 문맥을 함께 이해해야 해석이 가능합니다.
위 그림은 RNN 모델을 표현한 것이며 매번 들어오는 입력들을 처리한 후 예측하며 ht 라는 은닉층에 각 단계의 상태를 저장하고 이전의 은닉 상태를 다음단계로 전달하고 다시 예측하는 것을 반복합니다.
즉 연속된 데이터를 반복해서 처리하고 은닉층에 각각의 상태를 저장하고 이를 토대로 예측하는 모델입니다.

개념을 익히고 난 후 다시 파이토치 튜토리얼로 돌아와서 RNN 모델을 실습하였습니다. 사이트에 구현된 RNN 모델은 18 개 언어로 된 수천개의 이름을 훈련시키고, 이름이 어떤 언어인지 예측하는 모델이었으나 이 모델을 변형하여 각각의 sentence 모든 단어를 가지고 answer 클래스를 예측하는 것을 만들어봤습니다.

먼저 Json 파일을 읽었고 이 과정에서 all_answers 라는 리스트에는 answer 의 다섯 가지 클래스를 모두 담았습니다. 이 다섯가지 클래스인 '001,'002','003','004', '005' 등이 트레이닝 데이터의 label(정답) 역할을 하게 됩니다.

그리고 트레이닝 데이터를 answer 에 따라서 분류하였고 형태소 태그 중 'NN ,VV, SL, SN'에 해당하는 문자열만 추출하였습니다.

In [13]: %matplotlib inline

In [12]:
```
from __future__ import unicode_literals, print_function, division
from io import open
import glob
import os
import json
import pandas as pd
from konlpy.tag import Twitter
from itertools import chain
from collections import defaultdict
import re
import unicodedata
import string

n_letters = 128

# 각 클래스 별 단어 목록인 answer_lines 사전 생성
answer_lines = {}
all_answers = []
fw = open("test.txt",'w',encoding = 'utf-8')

with open('Tr.json',encoding="iso-8859-1") as f:
    json_data = json.load(f,encoding = "utf-8")
    for i in range(len(json_data)):
        for j in json_data[i]['sentence']:
            all_answers.append(json_data[i]['answer'])
            all_answers = list(set(all_answers))  #중복요소 제거해서 answer 모든 정답 담기
            json_data[i]['sentence'] = j.encode('iso-8859-1').decode('euc-kr', 'ignore')


Tr = pd.DataFrame(json_data)
Tr_1_answer = Tr.loc[(Tr['answer'] == '001')]
print(Tr_1_answer)
Tr_2_answer = Tr.loc[(Tr['answer'] == '002')]
Tr_3_answer = Tr.loc[(Tr['answer'] == '003')]
Tr_4_answer = Tr.loc[(Tr['answer'] == '004')]
Tr_5_answer = Tr.loc[(Tr['answer'] == '005')]
#print(Tr_1_answer)
Tr_1_answer_co = ' '.join(re.findall(r'([ㄱ-힣]+/NN|[ㄱ-힣]+/VV|[a-z]+/SL|[0-9]+/SN)+',str(Tr_1_answer['sentence'])))
Tr_2_answer_co = ' '.join(re.findall(r'([ㄱ-힣]+/NN|[ㄱ-힣]+/VV|[a-z]+/SL|[0-9]+/SN)+',str(Tr_2_answer['sentence'])))
Tr_3_answer_co = ' '.join(re.findall(r'([ㄱ-힣]+/NN|[ㄱ-힣]+/VV|[a-z]+/SL|[0-9]+/SN)+',str(Tr_3_answer['sentence'])))
Tr_4_answer_co = ' '.join(re.findall(r'([ㄱ-힣]+/NN|[ㄱ-힣]+/VV|[a-z]+/SL|[0-9]+/SN)+',str(Tr_4_answer['sentence'])))
Tr_5_answer_co = ' '.join(re.findall(r'([ㄱ-힣]+/NN|[ㄱ-힣]+/VV|[a-z]+/SL|[0-9]+/SN)+',str(Tr_5_answer['sentence'])))
```

(n_letters 의 경우 나올 수 있는 문자의 개수로서 한글 + 영어_소문자 + 숫자를 다 포함할 정도의 사이즈입니다.) One-hot 인코딩시 열의 개수로 사용될 예정입니다.)

```
print(Tr_1_answer_co)
Tr_1_nouns = re.sub('[A-Z/]','',Tr_1_answer_co)
Tr_2_nouns = re.sub('[A-Z/]','',Tr_2_answer_co)
Tr_3_nouns = re.sub('[A-Z/]','',Tr_3_answer_co)
Tr_4_nouns = re.sub('[A-Z/]','',Tr_4_answer_co)
Tr_5_nouns = re.sub('[A-Z/]','',Tr_5_answer_co)

answer_lines['001']= Tr_1_nouns.split()
answer_lines['002'] = Tr_2_nouns.split()
answer_lines['003'] = Tr_3_nouns.split()
answer_lines['004'] = Tr_4_nouns.split()
answer_lines['005'] = Tr_5_nouns.split()
print(answer_lines['001'])

n_answers = len(all_answers)
```

마지막으로 형태소 태깅을 제거한 후 split()함수로 각각의 단어들로 나눈 후 answer_lines 라는 딕셔너리에 모두 담았습니다. 딕셔너리를 활용한 이유는 이후에 학습할 때 클래스별로 단어들을 편하게 추출하기 위해서 이 자료형을 활용하였습니다.

```
[315 rows x 2 columns]
음식점/NN 을/VV 좋을만한/NN 앨범/NN http/SL news/SL le/SL step/SL by/SL step/SL win/SL http/SL news/SL le/SL http/SL news/SL le/SL 뮤직/NN
비디오/NN 음악/NN 을/NN 좋아하/VV 답/NN wind/SL 4/SN u/SL unitel/SL 조지/NN 윈스턴/NN 유/NN 키/NN 구라/NN 모토/NN http/SL docs/SL yahoo/SL
http/SL news/SL le/SL 아/NN 분/NN 가르/VV http/SL news/SL le/SL 아시/NN 분/NN 답변/NN 주세/NN /SL 싸이/VV 트/VV 가르/VV 가사/NN 찾/VV 난/N
N 모르/VV http/SL news/SL le/SL 분/NN 구입하셔서/NN /SL ses/SL 일본/NN 노래/NN 한국어/NN jhlee/SL 827/SN hanmail/SL net/SL http/SL news/SL
le/SL http/SL news/SL le/SL http/SL my/SL netian/SL 부탁드립니다/NN 계십시/NN ony/SL 레코드/NN 사/NN 계약/NN 협상/NN http/SL news/SL le/SL
들/VV 느낌/NN http/SL news/SL le/SL http/SL news/SL le/SL http/SL news/SL le/SL http/SL news/SL le/SL http/SL news/SL le/SL http/SL news/SL
le/SL e/SL /SL http/SL news/SL le/SL
['음식점', '을', '좋을만한', '앨범', 'http', 'news', 'le', 'step', 'by', 'step', 'win', 'http', 'news', 'le', 'http', 'news', 'le', '뮤직',
'비디오', '음악', '을', '좋아하', '답', 'wind', '4', 'u', 'unitel', '조지', '윈스턴', '유', '키', '구라', '모토', 'http', 'docs', 'yahoo',
'http', 'news', 'le', '아', '분', '가르', 'http', 'news', 'le', '아시', '분', '답변', '주세', '싸이', '트', '가르', '가사', '찾', '난', '모
르', 'http', 'news', 'le', '분', '구입하셔서', 'ses', '일본', '노래', '한국어', 'jhlee', '827', 'hanmail', 'net', 'http', 'news', 'le', 'ht
tp', 'news', 'le', 'http', 'my', 'netian', '부탁드립니다', '계십시', 'ony', '레코드', '사', '계약', '협상', 'http', 'news', 'le', '들', '느
낌', 'http', 'news', 'le', 'http', 'news', 'le', 'http', 'news', 'le', 'http', 'news', 'le', 'http', 'news', 'le', 'http', 'news', 'le',
'e', 'http', 'news', 'le']
```

데이터가 위와 같이 정제되는 과정입니다. (Tr_1_answer)만 출력하였습니다.

```
In [3]: print(answer_lines)
{'001': ['음식점', '을', '좋을만한', '앨범', 'http', 'news', 'le', 'step', 'by', 'step', 'win', 'http', 'news', 'le', 'http', 'news', 'l
e', '뮤직', '비디오', '음악', '을', '좋아하', '답', 'wind', '4', 'u', 'unitel', '조지', '윈스턴', '유', '키', '구라', '모토', 'http', 'd
ocs', 'yahoo', 'http', 'news', 'le', '아', '분', '가르', 'http', 'news', 'le', '아시', '분', '답변', '주세', '싸이', '트', '가르', '가
사', '찾', '난', '모르', 'http', 'news', 'le', '분', '구입하셔서', 'ses', '일본', '노래', '한국어', 'jhlee', '827', 'hanmail', 'net', 'h
ttp', 'news', 'le', 'http', 'news', 'le', 'http', 'my', 'netian', '부탁드립니다', '계십시', 'ony', '레코드', '사', '계약', '협상', 'htt
p', 'news', 'le', '들', '느낌', 'http', 'news', 'le', 'http', 'news', 'le', 'http', 'news', 'le', 'http', 'news', 'le', 'http', 'news',
'le', 'http', 'news', 'le', 'e', 'http', 'news', 'le'], '002': ['도움', '부탁', '드리', '아시', '분', '계시', '친절', '도움', '되셨',
'길', '감사', '테이블', '선언', '문장', '아시', '있', 'http', 'news', 'le', 'http', 'www', '있', 'http', 'news', 'le', '감사', '감사드
리', '가지', '악', '따르', '되', 'monsteer', 'hanmail', 'net', '답변', '감사', '몰', '부탁', '드리', '으', '질문', '주세', 'at', 'java',
'lang', 'hread', 'form', '절', 'tbl', 'name', '예제', '있', '답', '글', '좀', '부탁', 'ip', '막', 'tomatuli', 'hotmail', 'com', 'halleng
e', 'is', 'success'], '003': ['방법', '좀', '알', '식', '연결', '해', '하', '답변', '부탁', '드리', '감사', 'settings', '설정', '수', 's
weetlie', 'cais', 'kaist', '모르', 'mail', 'honma', '3', 'hanmail', '답', '되', '감사', '정보', '나눠', '주세', 'rogrammer', 'epublic',
'of', '수고', '하', '세요', 'endif', '도', '주십시', '감사', 'broadroad', 'empal', 'com', 'visual', '잘', '해', '단계', '알', '주십사하
는', 'ygkgwg', 'semicontech', 'co', 'rogrammer', 'epublic', 'of', 'orea', 'mail', 'honma', '3', 'hanmail', 'iew', 'iew', 'iew', 'o', 'oo
p', '의', '차이점', '가는거져'], '004': ['답변', '기다리', '넘겨줍니다', '라', '고', '쓰', '되', 'and', 'standard', 'error', 'of', 'on
e', '친절', '답변', '부탁', '드리', '사용', '감사드리', 'http', 'news', 'le', '누르', '되', '참', '하시', '길', '것', 'han', 'ping', '프
로그램', '잘못쳤', '엽', '이해', '하', '거', '아시는분들', '답변', '부탁', '드리', 'http', 'news', 'le', 'rogrammer', 'epublic', 'of',
'orea', 'http', 'news', 'le', '책', '생각', 'http', 'news', 'le', '생각', 'http', 'news', 'le', 'blueegg', 'hanmail', 'net', 'and', 'g
o', 'together', 'in', 'http', 'news', 'le', 'http', 'news', 'le', 'rogrammer', 'epublic', 'of', 'orea', 'and', 'go', 'together', 'in',
```

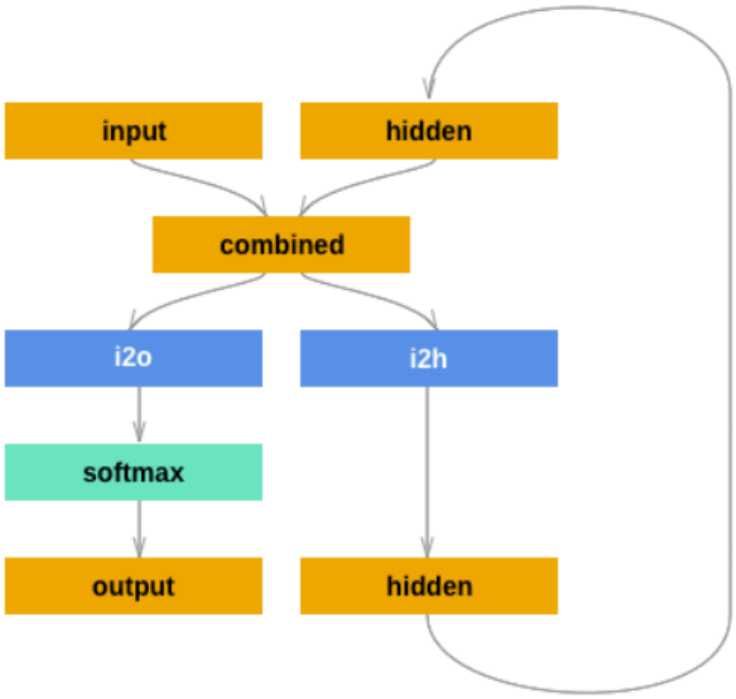
answer_lines 라는 딕셔너리를 출력하였습니다. 클래스 단어 사전이라고 보시면 됩니다.

```
In [28]: import torch

# all_letters 로 문자의 주소 찾기, 예시 "a" = 0
def letterToIndex(letter):
    return all_letters.find(letter)

# 검증을 위해서 한 문자를 <1 x n_letters> Tensor로 변환  #ONE-HOT벡터로 만드는 과정
def letterToTensor(letter):
    tensor = torch.zeros(1, n_letters)
    tensor[0][letterToIndex(letter)] = 1
    return tensor

# 한 줄(이름)을  <line_length x 1 x n_letters>,
# 또는 문자 벡터의 Array로 변경
def lineToTensor(line):
    tensor = torch.zeros(len(line), 1, n_letters)
    for li, letter in enumerate(line):
        tensor[li][0][letterToIndex(letter)] = 1
    return tensor

print(letterToTensor('가'))

print(lineToTensor('가나').size())

tensor([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
          0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
          0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
          0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
          0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
          0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
          0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
          0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
          0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
          0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
          0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.]])
torch.Size([2, 1, 128])
```

One hot-인코딩을 구현하였습니다. letterToTensor 는 한 문자를 Tensor 로 변환한다면 lineToTensor(line); 한 줄을 Tensor 로 변환해주는 함수입니다. 빠른 속도를 위해서 lineToTensor 를 사용할 것입니다.

```
In [5]: import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_answers)
```

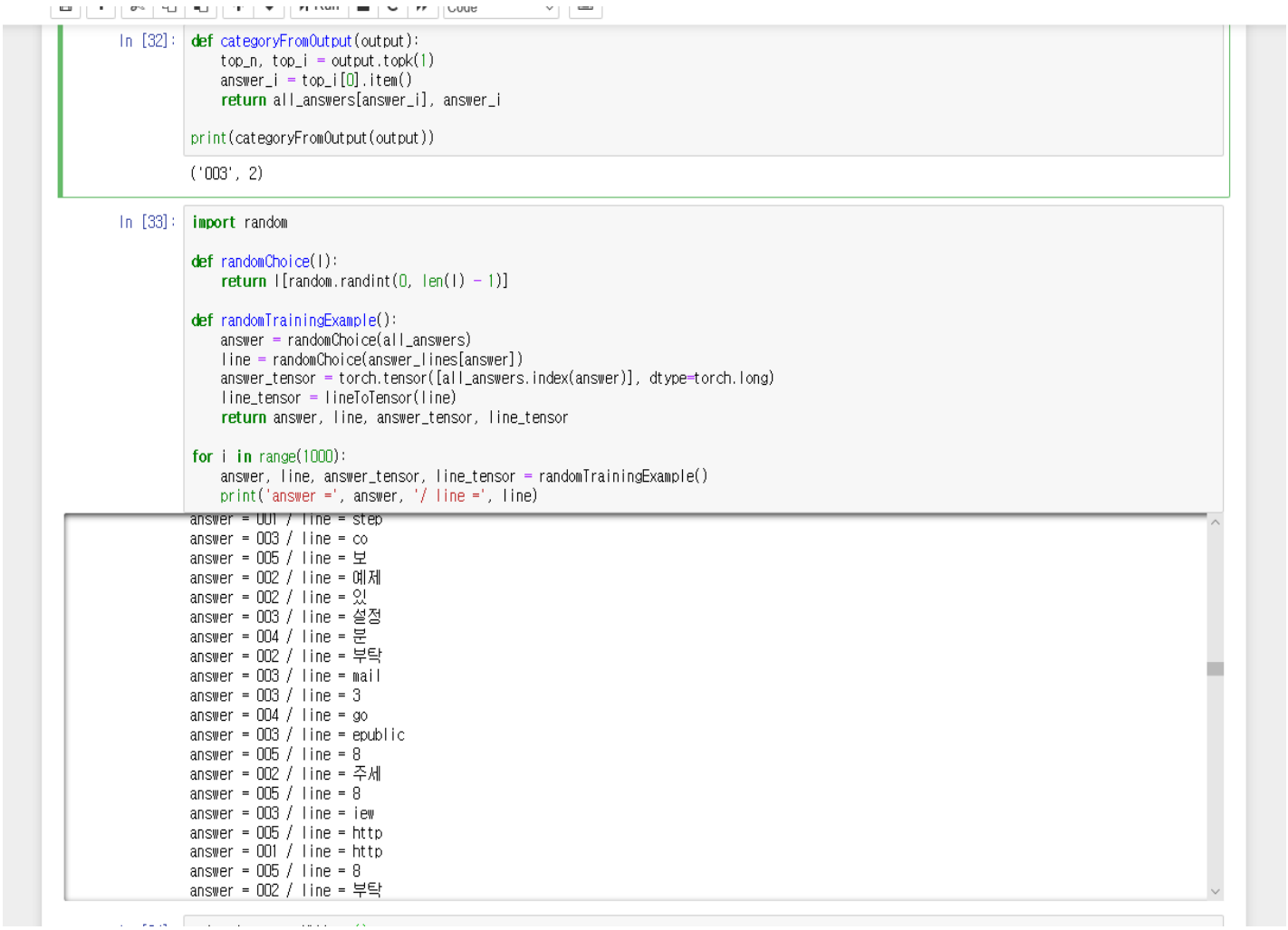
RNN 모델을 파이토치 패키지를 활용하여 구현한 코드입니다.



위 코드는 RNN 모델 답게 input 과 은닉층의 상태를 혼합하여 i2o 와 i2h 에 전달합니다. i2o 와 i2h 는 각각 다음단계의 출력과 은닉층에 저장이 되며 이 과정에서 출력층에 최종 저장되기 softmax 라는 활성함수를 거친 다음에 출력층에 저장이 됩니다.
은닉층의 상태는 재귀형태로 계속 반복하여 전달되어 다음 단계에도 영향을 미치는 것을 알 수 있습니다.

```
In [10]: input = letterToTensor('가')
         hidden =torch.zeros(1, n_hidden)

         output, next_hidden = rnn(input, hidden)
         print(output)

         tensor([[-1.7008, -1.5284, -1.5609, -1.7334, -1.5421]])

In [9]:  input = lineToTensor('음식점')
         hidden = torch.zeros(1, n_hidden)

         output, next_hidden = rnn(input[0], hidden)
         print(output)

         tensor([[-1.7008, -1.5284, -1.5609, -1.7334, -1.5421]])
```

앞서 설명한 것처럼 입력과 이전의 은닉 상태를 전달한 예시입니다. '가'라는 문자와 '음식점'이라는 단어가 tensor 로 잘 변환된 것을 볼 수 있습니다.
효율성을 위해서 lettertoTensor 대신 lineToTensor 를 사용할 것 입니다.



```
In [32]: def categoryFromOutput(output):
             top_n, top_i = output.topk(1)
             answer_i = top_i[0].item()
             return all_answers[answer_i], answer_i

         print(categoryFromOutput(output))

         ('003', 2)

In [33]: import random

         def randomChoice(l):
             return l[random.randint(0, len(l) - 1)]

         def randomTrainingExample():
             answer = randomChoice(all_answers)
             line = randomChoice(answer_lines[answer])
             answer_tensor = torch.tensor([all_answers.index(answer)], dtype=torch.long)
             line_tensor = lineToTensor(line)
             return answer, line, answer_tensor, line_tensor

         for i in range(1000):
             answer, line, answer_tensor, line_tensor = randomTrainingExample()
             print('answer =', answer, '/ line =', line)

         answer = 001 / line = step
         answer = 003 / line = co
         answer = 005 / line = 보
         answer = 002 / line = 예제
         answer = 002 / line = 있
         answer = 003 / line = 설정
         answer = 004 / line = 분
         answer = 002 / line = 부탁
         answer = 003 / line = mail
         answer = 003 / line = 3
         answer = 004 / line = go
         answer = 003 / line = epublic
         answer = 005 / line = 8
         answer = 002 / line = 주세
         answer = 005 / line = 8
         answer = 003 / line = iew
         answer = 005 / line = http
         answer = 001 / line = http
         answer = 005 / line = 8
         answer = 002 / line = 부탁
```

category 함수는 정답이라고 예측되는 값 중 가장 수치가 높은 값을 반환합니다 . random 패키지를 활용하여 빠르게 학습의 예시(클래스와 단어)를 얻어낼 수 있도록 구현하였습니다.

In [14]:
```
criterion = nn.NLLLoss()
```

In [15]:
```
learning_rate = 0.005 # 이것을 너무 높게 설정하면 폭발할 수 있고 너무 낮으면 학습이 되지 않을 수 있습니다.

def train(answer_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, answer_tensor)
    loss.backward()

    # learning rate를 곱한 파리미터의 경사도를 파리미터 값에 더합니다.
    for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)

    return output, loss.item()
```

학습 함수입니다. 학습률은 적당한 값을 넣었고 손실함수는 NLLloss 라는 함수를 활용하였습니다. One-hot 인코딩일 경우에 많이 쓰이며 softmax 활성함수랑 같이 쓰이는 경우가 많다고 합니다.
학습의 알고리즘은 다음과 같습니다.

1.입력과 tensor 생성
2. 0 으로 초기화된 은닉 상태 생성
3.각 문자 읽기
4.다음 문자를 위한 은닉 상태 유지
5.정답과 예상치 비교
6.backword 실행
7.예상 값과 손실 리턴

In [16]:
```
import time
import math

n_iters = 100000
print_every = 5000
plot_every = 5000



# 도식화를 위한 소실 추적
current_loss = 0
all_losses = []

def timeSince(since):
    now = time.time()
    s = now - since
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)

start = time.time()

for iter in range(1, n_iters + 1):
    answer, line, answer_tensor, line_tensor = randomTrainingExample()
    output, loss = train(answer_tensor, line_tensor)
    current_loss += loss

    # iter 숫자, 손실, 이름, 추측 출력
    if iter % print_every == 0:
        guess, guess_i = categoryFromOutput(output)
        correct = '✓' if guess == answer else 'X (%s)' % answer
        print('iterators : %d 퍼센테이지: %d%% 걸린시간: (%s) loss: %.4f 어구: %s  예측 클래스: %s 정답: %s' % (iter, iter / n_iters * 100,

    # 현재 평균 손실을 손실 리스트에 추가
    if iter % plot_every == 0:
        all_losses.append(current_loss / plot_every)
        current_loss = 0
```

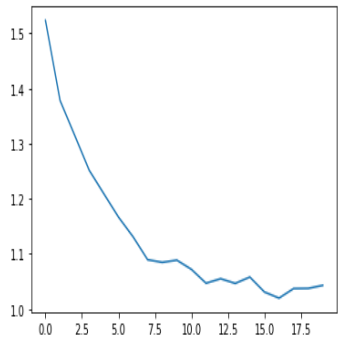
앞서 만들었던 모델을 직접 실행시킨 코드입니다. loss 값을 그래프로 나타내기 위해서 plot 함수를 사용하였으며 예측 값과 손실 값을 학습을 하면서 반환하였습니다.

```
iterators : 5000 퍼센테이지: 5% 걸린시간: (0m 3s) loss: 1.3707 어구: 감사드리  예측 클래스: 002 정답: ✓
iterators : 10000 퍼센테이지: 10% 걸린시간: (0m 6s) loss: 0.7654 어구: endif  예측 클래스: 003 정답: ✓
iterators : 15000 퍼센테이지: 15% 걸린시간: (0m 9s) loss: 1.4304 어구: 글  예측 클래스: 002 정답: ✓
iterators : 20000 퍼센테이지: 20% 걸린시간: (0m 16s) loss: 1.6211 어구: 연결  예측 클래스: 002 정답: X (003)
iterators : 25000 퍼센테이지: 25% 걸린시간: (0m 21s) loss: 2.2255 어구: 커널  예측 클래스: 002 정답: X (005)
iterators : 30000 퍼센테이지: 30% 걸린시간: (0m 23s) loss: 1.7297 어구: 구라  예측 클래스: 002 정답: X (001)
iterators : 35000 퍼센테이지: 35% 걸린시간: (0m 26s) loss: 0.0475 어구: honma  예측 클래스: 003 정답: ✓
iterators : 40000 퍼센테이지: 40% 걸린시간: (0m 29s) loss: 1.8460 어구: news  예측 클래스: 001 정답: X (005)
iterators : 45000 퍼센테이지: 45% 걸린시간: (0m 31s) loss: 0.4875 어구: http  예측 클래스: 001 정답: ✓
iterators : 50000 퍼센테이지: 50% 걸린시간: (0m 34s) loss: 2.1537 어구: news  예측 클래스: 001 정답: X (004)
iterators : 55000 퍼센테이지: 55% 걸린시간: (0m 37s) loss: 0.0827 어구: byte  예측 클래스: 004 정답: ✓
iterators : 60000 퍼센테이지: 60% 걸린시간: (0m 39s) loss: 1.2274 어구: 가지  예측 클래스: 002 정답: ✓
iterators : 65000 퍼센테이지: 65% 걸린시간: (0m 42s) loss: 0.9002 어구: 윈스턴  예측 클래스: 001 정답: ✓
iterators : 70000 퍼센테이지: 70% 걸린시간: (0m 44s) loss: 0.2849 어구: 부탁드립니다  예측 클래스: 001 정답: ✓
iterators : 75000 퍼센테이지: 75% 걸린시간: (0m 47s) loss: 0.0566 어구: kaist  예측 클래스: 003 정답: ✓
iterators : 80000 퍼센테이지: 80% 걸린시간: (0m 50s) loss: 2.6816 어구: 세계  예측 클래스: 003 정답: X (005)
iterators : 85000 퍼센테이지: 85% 걸린시간: (0m 53s) loss: 1.1795 어구: 부탁  예측 클래스: 002 정답: ✓
iterators : 90000 퍼센테이지: 90% 걸린시간: (0m 55s) loss: 0.3764 어구: http  예측 클래스: 001 정답: ✓
iterators : 95000 퍼센테이지: 95% 걸린시간: (0m 58s) loss: 0.0396 어구: 8  예측 클래스: 005 정답: ✓
iterators : 100000 퍼센테이지: 100% 걸린시간: (1m 0s) loss: 2.3066 어구: 모르  예측 클래스: 002 정답: X (005)
```

위와 같이 총 10 만번의 에폭 동안 손실 값과 예측 값을 5000 번마다 한 번씩 출력하였습니다.

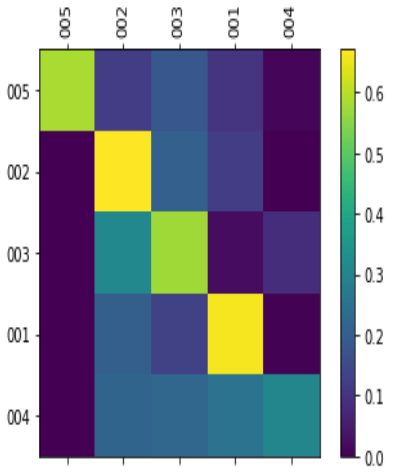
결과

```
In [37]: import matplotlib.pyplot as plt
         import matplotlib.ticker as ticker

         plt.figure()
         plt.plot(all_losses)

Out[37]: [<matplotlib.lines.Line2D at 0x1f032927588>]
```



위의 코드를 통해 저장되었던 loss 의 평균치를 plot 그래프로 나타냈고 학습을 할 수록 loss 값이 일정하게 감소하는 것을 볼 수 있습니다.
아래의 코드는 RNN 모델이 다른 클래스에서도 잘 작동하는지 보기 위해서 추측한 클래스와 실제 클래스를 나타내는 confusion matix 를 만들었습니다. confusion matrix 를 계산하기 위해 evaluate 함수를 사용하였고 evaluate 함수는 train 과 역전파를 제외하고는 동일하다는 것을 알 수 있습니다.

In [18]:
```
# 혼란 행렬에서 정확한 추측을 추적
confusion = torch.zeros(n_answers, n_answers)
n_confusion = 10000

# 주어진 라인의 출력 반환
def evaluate(line_tensor):
    hidden = rnn.initHidden()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    return output

# 올바르게 추측 된 예시와 기록을 살펴보십시오.
for i in range(n_confusion):
    answer, line, answer_tensor, line_tensor = randomTrainingExample()
    output = evaluate(line_tensor)
    guess, guess_i = categoryFromOutput(output)
    answer_i = all_answers.index(answer)
    confusion[answer_i][guess_i] += 1

# 모든 행을 합계로 나눔으로써 정규화하십시오.
for i in range(n_answers):
    confusion[i] = confusion[i] / confusion[i].sum()

# 도식 설정
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(confusion.numpy())
fig.colorbar(cax)

# 축 설정
ax.set_xticklabels(['']  + all_answers, rotation=90)
ax.set_yticklabels(['']  + all_answers)

# 모든 tick에서 강제로 레이블 지정
ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

# sphinx_gallery_thumbnail_number = 2
plt.show()
```



평가 그래프입니다. 예측한 클래스와 정답 클래스가 있는 대각선에서 밝은 색에 가까울수록 예측이 잘된 것을 알 수 있습니다. 대각선 성분을 이외의 성분들은 어두울수록 정답이 아닌 것을 잘 맞췄다는 것을 알 수 있습니다.
클래스 4 의 경우 비교적 예측이 잘 되지 않은 것으로 볼 수 있습니다.

느낀점

sentence 의 각각의 단어를 이용하여 answer 클래스를 예측하는 모델을 구현하면서 자연어 전처리과정과 RNN 을 직접 학습해 볼 수 있어 좋은 경험이었습니다. 아쉬운 점은 시간 관계상 교차검증을 해보지 못했다는 것과 트레이닝 데이터의 문장 속의 단어가 아닌 문장 전체를 학습하여 클래스를 예측하는 것을 설계하지 못 하였다는 것에 아쉬움을 느꼈습니다.
그러나 교수님께서 말씀하신 대로 방학 중에 더 공부하여 완벽한 딥러닝 모델을 만드는 것을 목표로 할 예정입니다.

감사합니다~!