**REPORT**



과목명 딥러닝

학과 산업경영공학

이름 12140471 김성진

12160539 김성민

12180604 김민규

그룹/팀 2그룹 14팀

제출일자 2020-12-23

* 차

**1. 목적 3**

**2. 이론 3**

2.1 RNN

2.2 LSTM

2.3 GRU

2.4 Transformer

**3. 방법 14**

3.1 API를 활용한 경로예측

3.1.1 표본 선정

3.1.2 pandas 데이터 형태로 변환

3.1.3 데이터 전처리 및 주소 return 받기

3.2 경로예측

3.2.1 데이터 분리

3.2.2 시간 단위로 예측

3.2.3 LSTM모델 학습

3.2.4 GRU모델 학습

**4. 결과 24**

**5. 결론** **28**

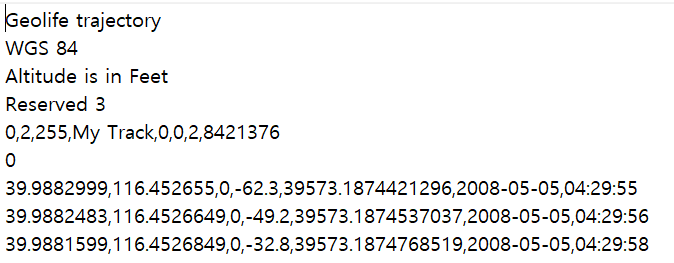
**6. 시행착오** **29**

**7. 참고문헌 33**

**1.** **목적**

인공지능 기술의 서비스 접목이 가속화되면서 개개인에게 특화된 서비스를 제공하는 ‘초개인화 시대’에 이르렀다. 초개인화’란 개인의 상황과 필요에 맞게 기업이 개별적인 맞춤 혜택을 제공하는 것을 뜻한다. 개인의 취향과 라이프 스타일을 중시하는 젊은 소비자들을 겨냥한 마케팅 방법이라고 말할 수 있다. 초개인화 시대에 어ᄄᅠᇂ게 고객을 사로잡을 것인가 그것이 우리가 생각한 문제다. 우리는 이러한 초개인화시대에 대응하기 위해서 사람들이 어떻게 움직이는지에 대한 관심이 필요하다고 생각한다. 사람들이 움직이는 경로를 분석하고 미래에 어디에 위치할 것인가를 예측하면 분명 이러한 초개인화시대에 많은 도움이 될 것이라고 생각했다. 따라서 우리는 위치(위도,경도),날짜와 시간이 포함되어있는 182명의 데이터를 통해 경로를 예측하는 실험을 진행하기로 한다. 진행 방법은 LSTM 모델, GRU, TRANSFORMER 모델을 사용하여 예측하기로 한다.

데이터 출처 : <https://www.microsoft.com/en-us/download/details.aspx?id=52367>



**2. 이론**

1) 순환 신경망(Recurrent Neural Network, RNN)

RNN(Recurrent Neural Network)은 시퀀스(Sequence) 모델입니다. 입력과 출력을 시퀀스 단위로 처리하는 모델입니다. 번역기를 생각해보면 입력은 번역하고자 하는 문장. 즉, 단어 시퀀스입니다. 출력에 해당되는 번역된 문장 또한 단어 시퀀스입니다. 이러한 시퀀스들을 처리하기 위해 고안된 모델들을 시퀀스 모델이라고 합니다. 그 중에서도 RNN은 딥 러닝에 있어 가장 기본적인 시퀀스 모델입니다.

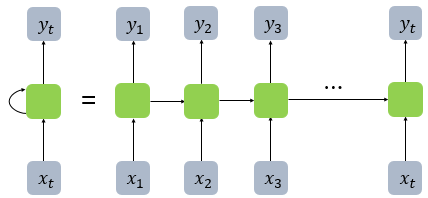
RNN은 은닉층의 노드에서 활성화 함수를 통해 나온 결과값을 출력층 방향으로도 보내면서, 다시 은닉층 노드의 다음 계산의 입력으로 보내는 특징을 갖고있습니다.



이를 그림으로 표현하면 위와 같습니다. x는 입력층의 입력 벡터, y는 출력층의 출력 벡터입니다. 실제로는 편향 b도 입력으로 존재할 수 있지만 앞으로의 그림에서는 생략합니다. RNN에서 은닉층에서 활성화 함수를 통해 결과를 내보내는 역할을 하는 노드를 셀(cell)이라고 합니다. 이 셀은 이전의 값을 기억하려고 하는 일종의 메모리 역할을 수행하므로 이를 메모리 셀 또는 RNN 셀이라고 표현합니다.

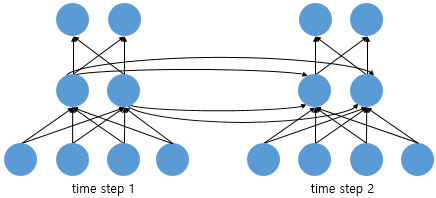
은닉층의 메모리 셀은 각각의 시점(time step)에서 바로 이전 시점에서의 은닉층의 메모리 셀에서 나온 값을 자신의 입력으로 사용하는 재귀적 활동을 하고 있습니다. 앞으로는 현재 시점을 변수 t로 표현하겠습니다. 이는 현재 시점 t에서의 메모리 셀이 갖고있는 값은 과거의 메모리 셀들의 값에 영향을 받은 것임을 의미합니다. 그렇다면 메모리 셀이 갖고 있는 이 값은 뭐라고 부를까요?

메모리 셀이 출력층 방향으로 또는 다음 시점 t+1의 자신에게 보내는 값을 은닉 상태(hidden state)라고 합니다. 다시 말해 t 시점의 메모리 셀은 t-1 시점의 메모리 셀이 보낸 은닉 상태값을 t 시점의 은닉 상태 계산을 위한 입력값으로 사용합니다.



RNN을 표현할 때는 일반적으로 위의 그림에서 좌측과 같이 화살표로 사이클을 그려서 재귀 형태로 표현하기도 하지만, 우측과 같이 사이클을 그리는 화살표 대신 여러 시점으로 펼쳐서 표현하기도 합니다. 두 그림은 동일한 그림으로 단지 사이클을 그리는 화살표를 사용하여 표현하였느냐, 시점의 흐름에 따라서 표현하였느냐의 차이일 뿐 둘 다 동일한 RNN을 표현하고 있습니다.

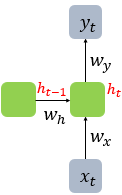
피드 포워드 신경망에서는 뉴런이라는 단위를 사용했지만, RNN에서는 뉴런이라는 단위보다는 입력층과 출력층에서는 각각 입력 벡터와 출력 벡터, 은닉층에서는 은닉 상태라는 표현을 주로 사용합니다. 그래서 사실 위의 그림에서 회색과 초록색으로 표현한 각 네모들은 기본적으로 벡터 단위를 가정하고 있습니다. 피드 포워드 신경망과의 차이를 비교하기 위해서 RNN을 뉴런 단위로 시각화해보겠습니다.



위의 그림은 입력 벡터의 차원이 4, 은닉 상태의 크기가 2, 출력층의 출력 벡터의 차원이 2인 RNN이 시점이 2일 때의 모습을 보여줍니다. 다시 말해 뉴런 단위로 해석하면 입력층의 뉴런 수는 4, 은닉층의 뉴런 수는 2, 출력층의 뉴런 수는 2입니다.

RNN은 입력과 출력의 길이를 다르게 설계 할 수 있으므로 다양한 용도로 사용할 수 있습니다. 위 그림은 입력과 출력의 길이에 따라서 달라지는 RNN의 다양한 형태를 보여줍니다. 위 구조가 자연어 처리에서 어떻게 사용될 수 있는지 예를 들어봅시다. RNN 셀의 각 시점 별 입, 출력의 단위는 사용자가 정의하기 나름이지만 가장 보편적인 단위는 '단어 벡터'입니다.

이제 RNN에 대한 수식을 정의해보겠습니다.



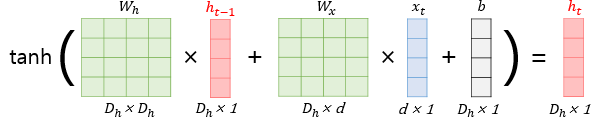
현재 시점 t에서의 은닉 상태값을 htht라고 정의하겠습니다. 은닉층의 메모리 셀은 htht를 계산하기 위해서 총 두 개의 가중치를 갖게 됩니다. 하나는 입력층에서 입력값을 위한 가중치 WxWx이고, 하나는 이전 시점 t-1의 은닉 상태값인 ht−1ht−1을 위한 가중치 WhWh입니다.

이를 식으로 표현하면 다음과 같습니다.  
은닉층 : ht=tanh(Wxxt+Whht−1+b)ht=tanh(Wxxt+Whht−1+b)  
출력층 : yt=f(Wyht+b)yt=f(Wyht+b)  
단, ff는 비선형 활성화 함수 중 하나.

RNN의 은닉층 연산을 벡터와 행렬 연산으로 이해할 수 있습니다. 자연어 처리에서 RNN의 입력 xtxt는 대부분의 경우에서 단어 벡터로 간주할 수 있는데, 단어 벡터의 차원을 dd라고 하고, 은닉 상태의 크기를 DhDh라고 하였을 때 각 벡터와 행렬의 크기는 다음과 같습니다.

xtxt : (d×1)(d×1)  
WxWx : (Dh×d)(Dh×d)  
WhWh : (Dh×Dh)(Dh×Dh)  
ht−1ht−1 : (Dh×1)(Dh×1)  
bb : (Dh×1)(Dh×1)

배치 크기가 1이고, dd와 DhDh 두 값 모두를 4로 가정하였을 때, RNN의 은닉층 연산을 그림으로 표현하면 아래와 같습니다.



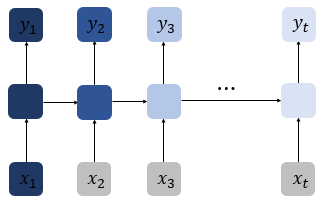
이때 htht를 계산하기 위한 활성화 함수로는 주로 하이퍼볼릭탄젠트 함수(tanh)가 사용되지만, ReLU로 바꿔 사용하는 시도도 있습니다.

위의 식에서 각각의 가중치 WxWx, WhWh, WyWy의 값은 모든 시점에서 값을 동일하게 공유합니다. 만약, 은닉층이 2개 이상일 경우에는 은닉층 2개의 가중치는 서로 다릅니다.

출력층은 결과값인 ytyt를 계산하기 위한 활성화 함수로는 상황에 따라 다를텐데, 예를 들어서 이진 분류를 해야하는 경우라면 시그모이드 함수를 사용할 수 있고 다양한 카테고리 중에서 선택해야하는 문제라면 소프트맥스 함수를 사용하게 될 것입니다.

2) 장단기 메모리(Long Short-Term Memory, LSTM)

**1. RNN의 한계**

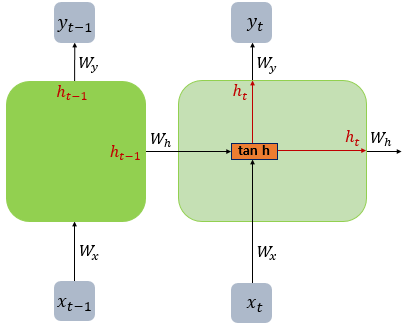


앞 챕터에서 RNN은 출력 결과가 이전의 계산 결과에 의존한다는 것을 언급한 바 있습니다. 하지만 RNN은 비교적 짧은 시퀀스(sequence)에 대해서만 효과를 보이는 단점이 있습니다. RNN의 시점(time step)이 길어질 수록 앞의 정보가 뒤로 충분히 전달되지 못하는 현상이 발생합니다. 위의 그림은 첫번째 입력값인 x1의 정보량을 짙은 남색으로 표현했을 때, 색이 점차 얕아지는 것으로 시점이 지날수록 x1의 정보량이 손실되어가는 과정을 표현하였습니다. 뒤로 갈수록 x1의 정보량은 손실되고, 시점이 충분히 긴 상황에서는 x1의 전체 정보에 대한 영향력은 거의 의미가 없을 수도 있습니다.

어쩌면 가장 중요한 정보가 시점의 앞 쪽에 위치할 수도 있습니다. RNN으로 만든 언어 모델이 다음 단어를 예측하는 과정을 생각해봅시다. 예를 들어 ''모스크바에 여행을 왔는데 건물도 예쁘고 먹을 것도 맛있었어. 그런데 글쎄 직장 상사한테 전화가 왔어. 어디냐고 묻더라구 그래서 나는 말했지. 저 여행왔는데요. 여기 \_\_\_'' 다음 단어를 예측하기 위해서는 장소 정보가 필요합니다. 그런데 장소 정보에 해당되는 단어인 '모스크바'는 앞에 위치하고 있고, RNN이 충분한 기억력을 가지고 있지 못한다면 다음 단어를 엉뚱하게 예측합니다.

이를 장기 의존성 문제(the problem of Long-Term Dependencies)라고 합니다

**2. RNN 내부 열어보기**

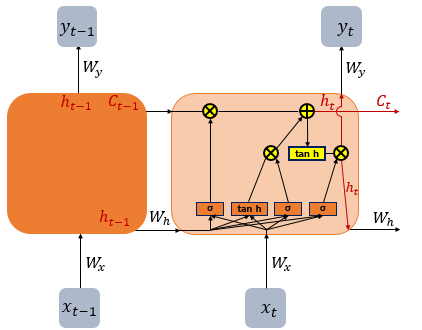


LSTM에 대해서 이해해보기 전에 RNN에 대해 언급하겠습니다. 위의 그림은 RNN의 내부 구조를 보여줍니다. 여기에서는 RNN 계열의 인공 신경망의 그림에서는 편향 bb를 생략합니다. 위의 그림에 편향 bb를 그린다면 xtxt 옆에 tanh로 향하는 또 하나의 입력선을 그리면 됩니다.

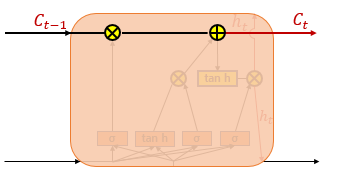
ht=tanh(Wxxt+Whht−1+b)ht=tanh(Wxxt+Whht−1+b)

RNN은 xtxt와 ht−1ht−1이라는 두 개의 입력이 각각의 가중치와 곱해져서 메모리 셀의 입력이 됩니다. 그리고 이를 하이퍼볼릭탄젠트 함수의 입력으로 사용하고 이 값은 은닉층의 출력인 은닉 상태가 됩니다.

**3. LSTM(Long Short-Term Memory)**



위의 그림은 LSTM의 전체적인 내부의 모습을 보여줍니다. 전통적인 RNN의 이러한 단점을 보완한 RNN의 일종을 장단기 메모리(Long Short-Term Memory)라고 하며, 줄여서 LSTM이라고 합니다. LSTM은 은닉층의 메모리 셀에 입력 게이트, 망각 게이트, 출력 게이트를 추가하여 불필요한 기억을 지우고, 기억해야할 것들을 정합니다. 요약하면 LSTM은 은닉 상태(hidden state)를 계산하는 식이 전통적인 RNN보다 조금 더 복잡해졌으며 셀 상태(cell state)라는 값을 추가하였습니다. 위의 그림에서는 t시점의 셀 상태를 CtCt로 표현하고 있습니다. LSTM은 RNN과 비교하여 긴 시퀀스의 입력을 처리하는데 탁월한 성능을 보입니다.

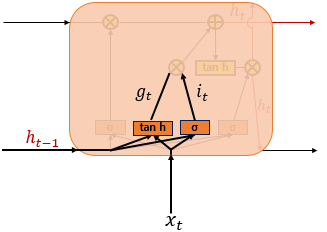


셀 상태는 위의 그림에서 왼쪽에서 오른쪽으로 가는 굵은 선입니다. 셀 상태 또한 이전에 배운 은닉 상태처럼 이전 시점의 셀 상태가 다음 시점의 셀 상태를 구하기 위한 입력으로서 사용됩니다.

은닉 상태값과 셀 상태값을 구하기 위해서 새로 추가 된 3개의 게이트를 사용합니다. 각 게이트는 삭제 게이트, 입력 게이트, 출력 게이트라고 부르며 이 3개의 게이트에는 공통적으로 시그모이드 함수가 존재합니다. 시그모이드 함수를 지나면 0과 1사이의 값이 나오게 되는데 이 값들을 가지고 게이트를 조절합니다. 아래의 내용을 먼저 이해하고 각 게이트에 대해서 알아보도록 하겠습니다.

* 이하 식에서 σ는 시그모이드 함수를 의미합니다.
* 이하 식에서 tanh는 하이퍼볼릭탄젠트 함수를 의미합니다.
* Wxi,Wxg,Wxf,WxoWxi,Wxg,Wxf,Wxo는 xtxt와 함께 각 게이트에서 사용되는 4개의 가중치입니다.
* Whi,Whg,Whf,WhoWhi,Whg,Whf,Who는 ht−1ht−1와 함께 각 게이트에서 사용되는 4개의 가중치입니다.
* bi,bg,bf,bobi,bg,bf,bo는 각 게이트에서 사용되는 4개의 편향입니다.

**(1) 입력 게이트**



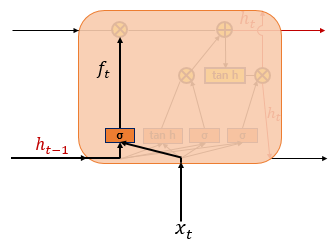
it=σ(Wxixt+Whiht−1+bi)it=σ(Wxixt+Whiht−1+bi)  
gt=tanh(Wxgxt+Whght−1+bg)gt=tanh(Wxgxt+Whght−1+bg)

입력 게이트는 현재 정보를 기억하기 위한 게이트입니다. 우선 현재 시점 t의 xx값과 입력 게이트로 이어지는 가중치 WxiWxi를 곱한 값과 이전 시점 t-1의 은닉 상태가 입력 게이트로 이어지는 가중치 WhiWhi를 곱한 값을 더하여 시그모이드 함수를 지납니다. 이를 itit라고 합니다.

그리고 현재 시점 t의 xx값과 입력 게이트로 이어지는 가중치 WxiWxi를 곱한 값과 이전 시점 t-1의 은닉 상태가 입력 게이트로 이어지는 가중치 WhgWhg를 곱한 값을 더하여 하이퍼볼릭탄젠트 함수를 지납니다. 이를 gtgt라고 합니다.

시그모이드 함수를 지나 0과 1 사이의 값과 하이퍼볼릭탄젠트 함수를 지나 -1과 1사이의 값 두 개가 나오게 됩니다. 이 두 개의 값을 가지고 이번에 선택된 기억할 정보의 양을 정하는데, 구체적으로 어떻게 결정하는지는 아래에서 배우게 될 셀 상태 수식을 보면 됩니다.

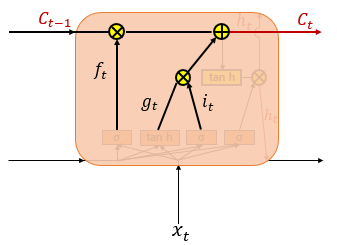
**(2) 삭제 게이트**



ft=σ(Wxfxt+Whfht−1+bf)ft=σ(Wxfxt+Whfht−1+bf)

삭제 게이트는 기억을 삭제하기 위한 게이트입니다. 현재 시점 t의 xx값과 이전 시점 t-1의 은닉 상태가 시그모이드 함수를 지나게 됩니다. 시그모이드 함수를 지나면 0과 1 사이의 값이 나오게 되는데, 이 값이 곧 삭제 과정을 거친 정보의 양입니다. 0에 가까울수록 정보가 많이 삭제된 것이고 1에 가까울수록 정보를 온전히 기억한 것입니다. 이를 가지고 셀 상태를 구하게 되는데, 구체적으로는 아래의 셀 상태 수식을 보면 됩니다.

**(3) 셀 상태(장기 상태)**



Ct=ft∘Ct−1+it∘gtCt=ft∘Ct−1+it∘gt

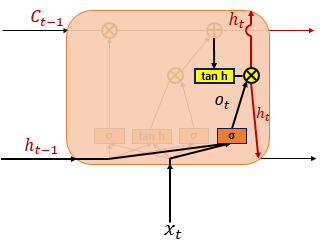
셀 상태 CtCt를 LSTM에서는 장기 상태라고 부르기도 합니다. 그렇다면 셀 상태를 구하는 방법을 알아보겠습니다. 삭제 게이트에서 일부 기억을 잃은 상태입니다.

입력 게이트에서 구한 itit, gtgt 이 두 개의 값에 대해서 원소별 곱(entrywise product)을 진행합니다. 다시 말해 같은 크기의 두 행렬이 있을 때 같은 위치의 성분끼리 곱하는 것을 말합니다. 여기서는 식으로 ∘ 로 표현합니다. 이것이 이번에 선택된 기억할 값입니다.

입력 게이트에서 선택된 기억을 삭제 게이트의 결과값과 더합니다. 이 값을 현재 시점 t의 셀 상태라고 하며, 이 값은 다음 t+1 시점의 LSTM 셀로 넘겨집니다.

삭제 게이트와 입력 게이트의 영향력을 이해해봅시다. 만약 삭제 게이트의 출력값인 ftft가 0이 된다면, 이전 시점의 셀 상태값인 Ct−1Ct−1은 현재 시점의 셀 상태값을 결정하기 위한 영향력이 0이 되면서, 오직 입력 게이트의 결과만이 현재 시점의 셀 상태값 CtCt을 결정할 수 있습니다. 이는 삭제 게이트가 완전히 닫히고 입력 게이트를 연 상태를 의미합니다. 반대로 입력 게이트의 itit값을 0이라고 한다면, 현재 시점의 셀 상태값 CtCt는 오직 이전 시점의 셀 상태값 Ct−1Ct−1의 값에만 의존합니다. 이는 입력 게이트를 완전히 닫고 삭제 게이트만을 연 상태를 의미합니다. 결과적으로 삭제 게이트는 이전 시점의 입력을 얼마나 반영할지를 의미하고, 입력 게이트는 현재 시점의 입력을 얼마나 반영할지를 결정합니다.

**(4) 출력 게이트와 은닉 상태(단기 상태)**



ot=σ(Wxoxt+Whoht−1+bo)ot=σ(Wxoxt+Whoht−1+bo)  
ht=ot∘tanh(ct)ht=ot∘tanh(ct)

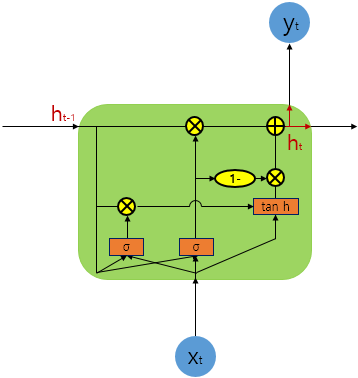
출력 게이트는 현재 시점 t의 xx값과 이전 시점 t-1의 은닉 상태가 시그모이드 함수를 지난 값입니다. 해당 값은 현재 시점 t의 은닉 상태를 결정하는 일에 쓰이게 됩니다.

은닉 상태를 단기 상태라고 하기도 합니다. 장기 상태의 값이 하이퍼볼릭탄젠트 함수를 지나 -1과 1사이의 값이 되고, 해당 값은 출력 게이트의 값과 연산되면서, 값이 걸러지는 효과가 발생하여 은닉 상태가 됩니다. 단기 상태의 값은 또한 출력층으로도 향합니다.

3) 게이트 순환 유닛(Gated Recurrent Unit, GRU)

1. GRU(Gated Recurrent Unit)

LSTM에서는 출력, 입력, 삭제 게이트라는 3개의 게이트가 존재했습니다. 반면, GRU에서는 업데이트 게이트와 리셋 게이트 두 가지 게이트만이 존재합니다. GRU는 LSTM보다 학습 속도가 빠르다고 알려져있지만 여러 평가에서 GRU는 LSTM과 비슷한 성능을 보인다고 알려져 있습니다.



반드시 LSTM 대신 GRU를 사용하는 것이 좋지는 않습니다. GRU와 LSTM 중 어떤 것이 모델의 성능면에서 더 낫다라고 단정지어 말할 수 없으며, 기존에 LSTM을 사용하면서 최적의 하이퍼파라미터를 찾아낸 상황이라면 굳이 GRU로 바꿔서 사용할 필요는 없습니다.

데이터 양이 적을 때는, 매개 변수의 양이 적은 GRU가 조금 더 낫고, 데이터 양이 더 많으면 LSTM이 더 낫다고 알려져 있습니다. GRU보다 LSTM에 대한 연구나 사용량이 더 많은데, 이는 LSTM이 더 먼저 나온 구조이기 때문입니다.

Massive Exploration of Neural Machine Translation Architectures, 2017 논문 中

In our experiments, LSTM cells consistently outperformed GRU cells

**4) 트랜스포머(Transformer)**

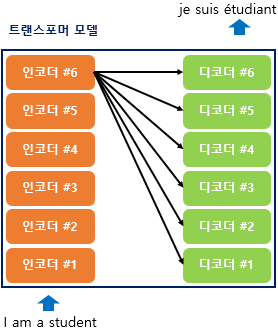
트랜스포머(Transformer)는 2017년 구글이 발표한 논문인 "Attention is all you need"에서 나온 모델로 기존의 seq2seq의 구조인 인코더-디코더를 따르면서도, 논문의 이름처럼 어텐션(Attention)만으로 구현한 모델입니다. 이 모델은 RNN을 사용하지 않고, 인코더-디코더 구조를 설계하였음에도 성능도 RNN보다 우수하다는 특징을 갖고 있습니다.

(시퀀스-투-시퀀스(Sequence-to-Sequence)는 입력된 시퀀스로부터 다른 도메인의 시퀀스를 출력하는 다양한 분야에서 사용되는 모델입니다. 예를 들어 챗봇(Chatbot)과 기계 번역(Machine Translation)이 그러한 대표적인 예인데, 입력 시퀀스와 출력 시퀀스를 각각 질문과 대답으로 구성하면 챗봇으로 만들 수 있고, 입력 시퀀스와 출력 시퀀스를 각각 입력 문장과 번역 문장으로 만들면 번역기로 만들 수 있습니다. 그 외에도 내용 요약(Text Summarization), STT(Speech to Text) 등에서 쓰일 수 있습니다.)

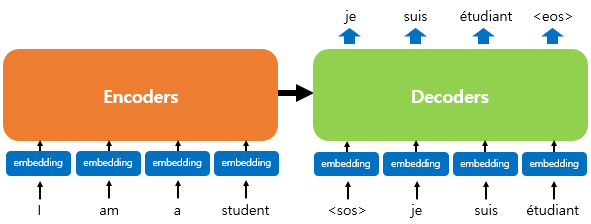
(어텐션의 기본 아이디어는 디코더에서 출력 단어를 예측하는 매 시점(time step)마다, 인코더에서의 전체 입력 문장을 다시 한 번 참고한다는 점입니다. 단, 전체 입력 문장을 전부 다 동일한 비율로 참고하는 것이 아니라, 해당 시점에서 예측해야할 단어와 연관이 있는 입력 단어 부분을 좀 더 집중(attention)해서 보게 됩니다.)

트랜스포머는 RNN을 사용하지 않지만 기존의 seq2seq처럼 인코더에서 입력 시퀀스를 입력받고, 디코더에서 출력 시퀀스를 출력하는 인코더-디코더 구조를 유지하고 있습니다. 다만 다른 점은 인코더와 디코더라는 단위가 N개가 존재할 수 있다는 점입니다.

이전 seq2seq 구조에서는 인코더와 디코더에서 각각 하나의 RNN이 t개의 시점(time-step)을 가지는 구조였다면 이번에는 인코더와 디코더라는 단위가 N개로 구성되는 구조입니다. 트랜스포머를 제안한 논문에서는 인코더와 디코더의 개수를 각각 6개를 사용하였습니다.



위의 그림은 인코더와 디코더가 6개씩 존재하는 트랜스포머의 구조를 보여줍니다. 이 책에서는 인코더와 디코더가 각각 여러 개 쌓여있다는 의미를 사용할 때는 알파벳 s를 뒤에 붙여 encoders, decoders라고 표현하겠습니다.



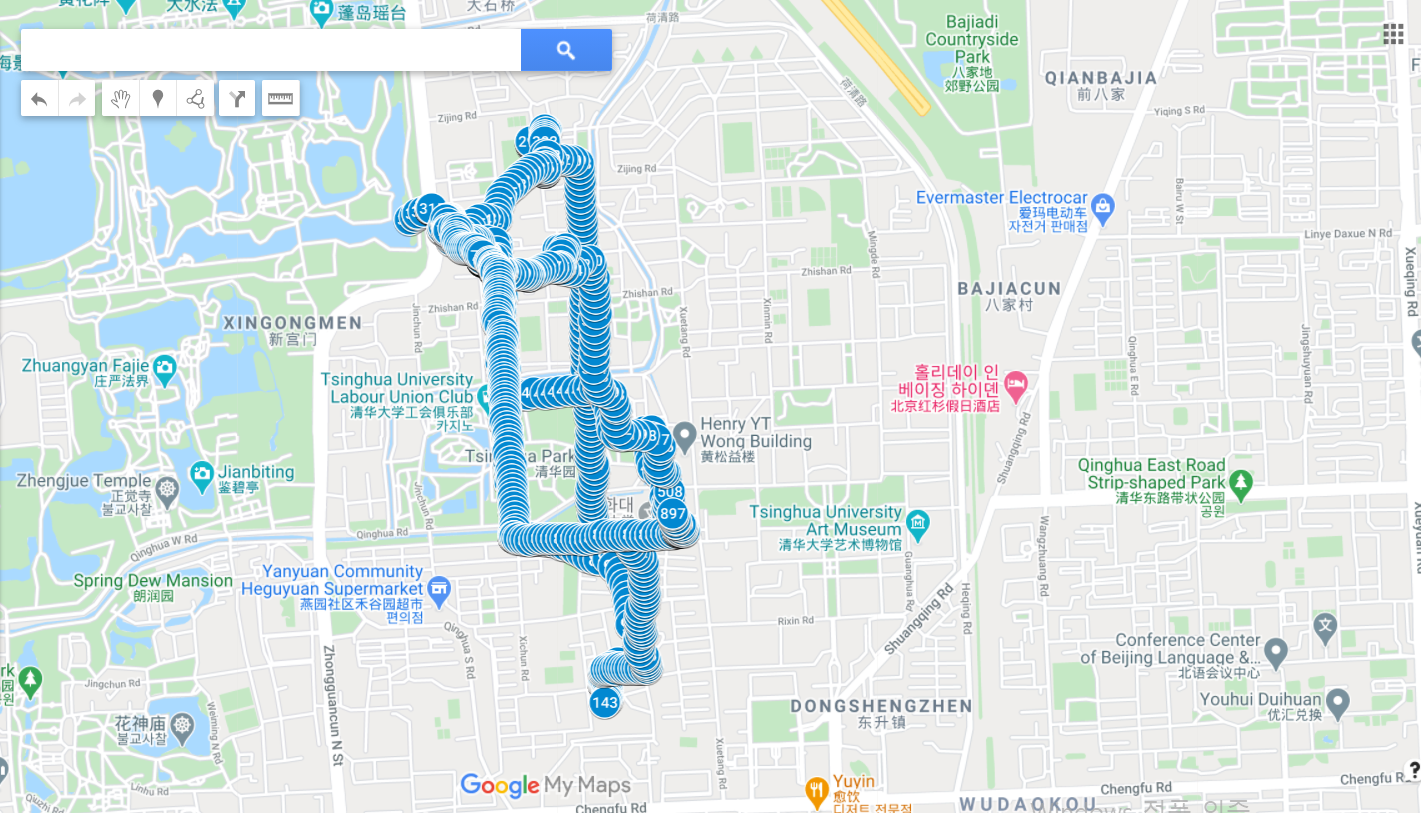
위의 그림은 인코더로부터 정보를 전달받아 디코더가 출력 결과를 만들어내는 트랜스포머 구조를 보여줍니다. 디코더는 마치 기존의 seq2seq 구조처럼 시작 심볼 <sos>를 입력으로 받아 종료 심볼 <eos>가 나올 때까지 연산을 진행합니다. 이는 RNN은 사용되지 않지만 여전히 인코더-디코더의 구조는 유지되고 있음을 보여줍니다.

**3. 방법**

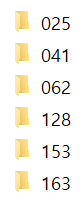
3.1. 표본 선정

1) 수집 일 수가 많은 데이터를 선별해서 골라낸다

2) 일정하게 데이터가 존재해야 한다.

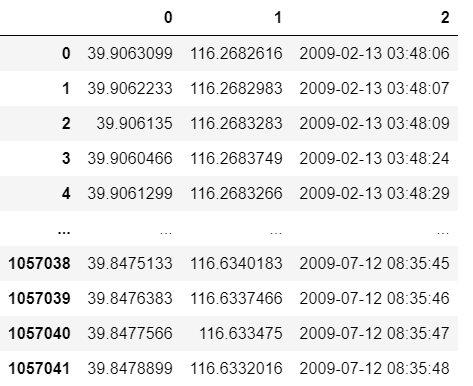


(구글맵을 통한 시각화)위의 기준에 부합하는 6명을 선정해서 학습할 예정이다.



길이가 500인 사람 이상 중에서 총 6인이 선정되었다. 하지만 시간적인 제약으로 인해 가장 긴 128번 사람을 대상으로 실험해보았다

3.1.2 pandas dataframe 형태로 변환



3.1.3 데이터 전처리 및 주소 return 받기

구글 API를 활용해 좌표에 수반한 주소를 리턴 받고, 이것을 파파고 API를 활용해서 번역하였다.



3.2 경로예측

3.2.1 데이터 분리하기

데이터의 문제점들을 종합해보면 아래와 같다.

1. 데이터 구성하고 있는 사람들이 많다는 점

2. 어느 곳에 방문할지에 대한 정보가 좀 더 중요하다는 점

3. 사람이 걸을 수 있는 속도가 한정되어 있다는 점

세가지를 모두 고려할 때 모두를 학습시키기 보다는 일부분을 쪼개 학습시키는 것이 합리 적이다.

3.2.2 시간 단위로 예측

간격 사이의 평균 시간이 1분 27초 정도였기에 약 30정도를 곱해서 30분 단위로 예측을 해본다.



3.2.3 API로 불러온 데이터 중에 쓸 수 있는 데이터 추출

한 논문에서 에서 장소를 모두 매핑시켜서 학습시키는 방법을 사용하였기에 다음과 같은 방법론을 만들어 냈다. 하지만 이후에 좌표 하나를 페어링 함수를 활용해 하나의 수로 매핑시키는 방법도 충분히 가능하다는 것을 알게 되었다.



3.2.3 LSTM, GRU 모델 학습

X, Y 좌표로 이루어져 있다는 점을 고려했을 때 X와 Y 각각을 하나의 모델로 두고 2개의 모델을 학습시키려고 하였음 LOSS를 X와 Y 로스의 합으로 구성되도록 하였다.

3.2.4 TRANSFORMER 모델 학습

X,Y 좌표를 페어링 함수를 활용해서 한 좌표로 들어가도록 한다. 이후 MSE LOSS, L1 LOSS를 활용해서 학습하도록 구성하였다. TRANSFORMER에 추출된 단어를 넣고자 하였으나 역량의 부족으로 인해서 그 부분까지는 시도해보지 못했다. 즉, 상가 -> 톨게이트 -> 집 -> 학교 이라고 했을 때 사람이 상가 -> 톨게이트 -> 집을 방문한다면, 학교가 RETURN되는 TRANSFORMER을 구성하고자 하였지만 역량의 부족으로 충분히 구성하지 못하였다. 그러나 만약 이 자체를 협업 필터링을 사용해서도 충분히 가능할 것이라 생각한다.

**4. 결과**

Train Set : 0분 부터 시작하여 30분 단위로 결과 값이 나오도록 함 (ex 0, 30, 60 분 ...)

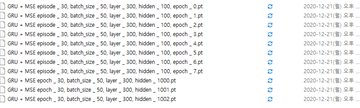
Validation Set : 10분 부터 시작하여 30분 단위로 값이 나오도록 함 (ex 10 40 분)

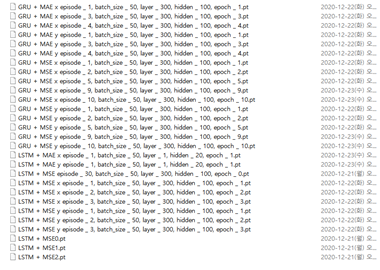
**WEIGHT LINK**

GRU : <https://drive.google.com/file/d/1Y4pwqJZvS5v-u1ZpMv4w15cgcyoP4K4F/view?usp=sharing>

LSTM

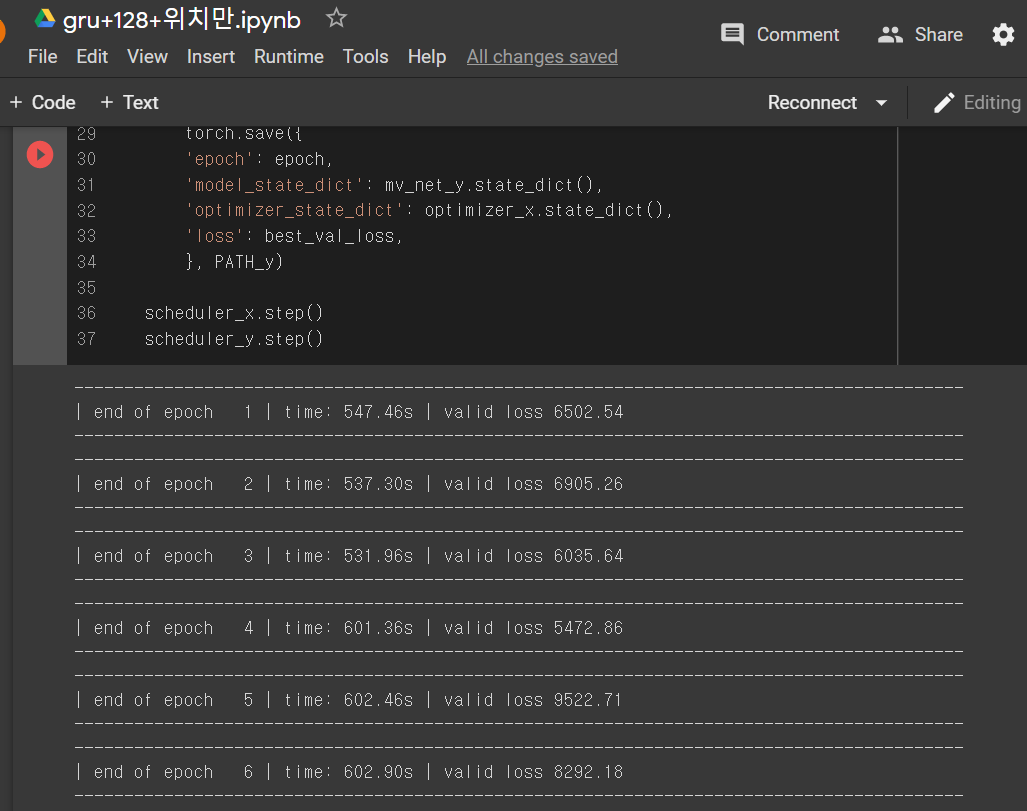
<https://drive.google.com/file/d/1xdSlsfMMTjPC50h8WimKMmn6lZ6pUcqU/view?usp=sharing>





* GRU 모델 + MAE LOSS

GRU(batch size 50, layer 300, hidden : 100)와 L1 LOSS를 조합하였고, 여러 파라미터를 실험해본 결과 가장 적은 밸리데이션 로스를 만들어내는 모델을 만들어 냈다. loss가 4인 경우에 가장 적다는 점을 고려할 때 모든 배치를 다 돌리는 것은 무의미하다는 생각을 가져 학습을 멈췄다.



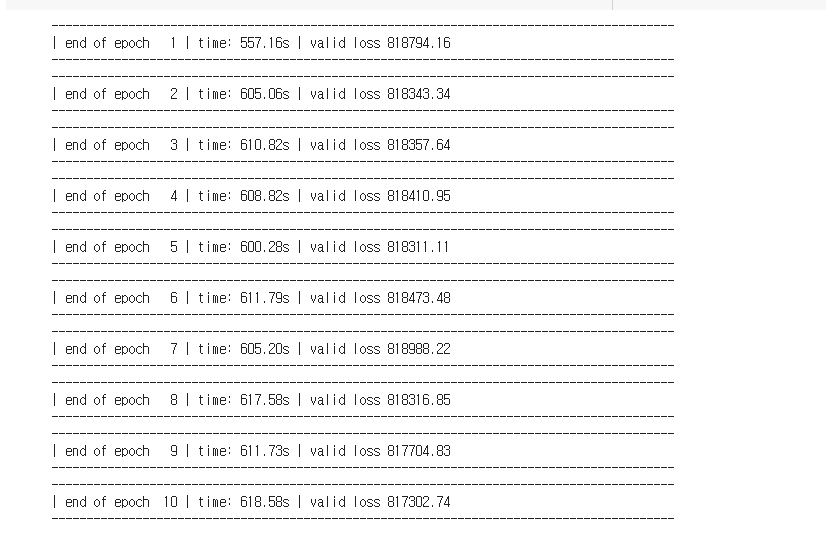
* GRU모델 + MSE LOSS

GRU(batch size 50, layer 300, hidden : 100)와 L1 LOSS를 조합하였고, 여러 파라미터를 실험해본 결과 배치 2인 경우에 가장 LOSS가 작게 나왔다. 그 이상은 더이상 로스가 줄어들지 않아 7에서 학습을 중단시켰다.



* LSTM모델 + MAE LOSS

학습이 잘 되지 않았다.



* TRANSFORMER 모델

학습하려고 코드를 모두 구성해 놓았으나 차원을 맞추는 것에 실패하여 코드로만 남아있다.

**5. 시행착오**

지도 데이터를 받아 오는 과정에서 시행착오

1. SELENIUM을 활용한 크롤링을 통한 주소 받아오기

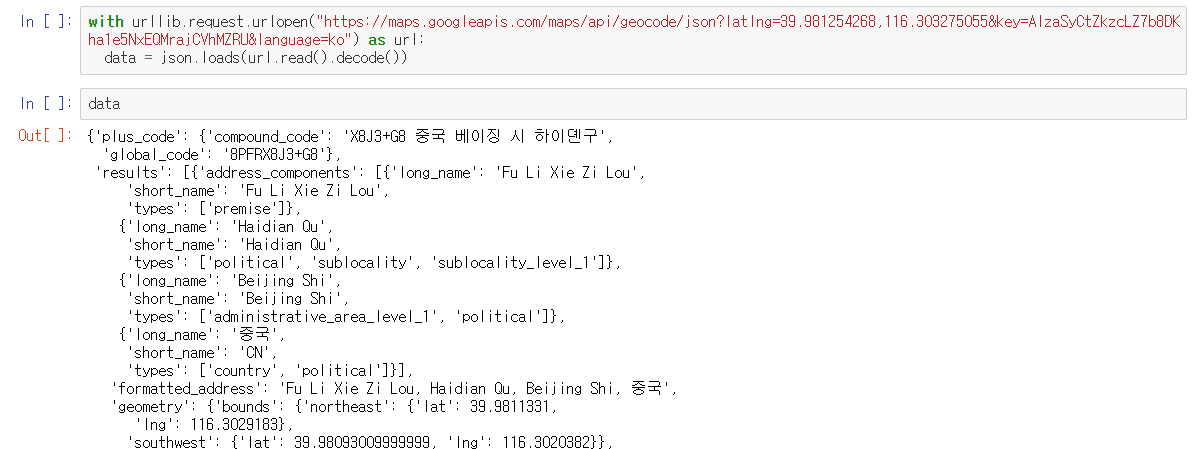
크롤링 하려는 시도를 하였지만, 버튼 자체를 클릭함에 있어서 막아 놓아 다른 방법을 고민하는 시간이 많이 흘렀다. 다음과 같이 버튼을 누르려고 하면은 거부 반응이 일어나서 API를 사용하는 방식을 사용하도록 하였다.

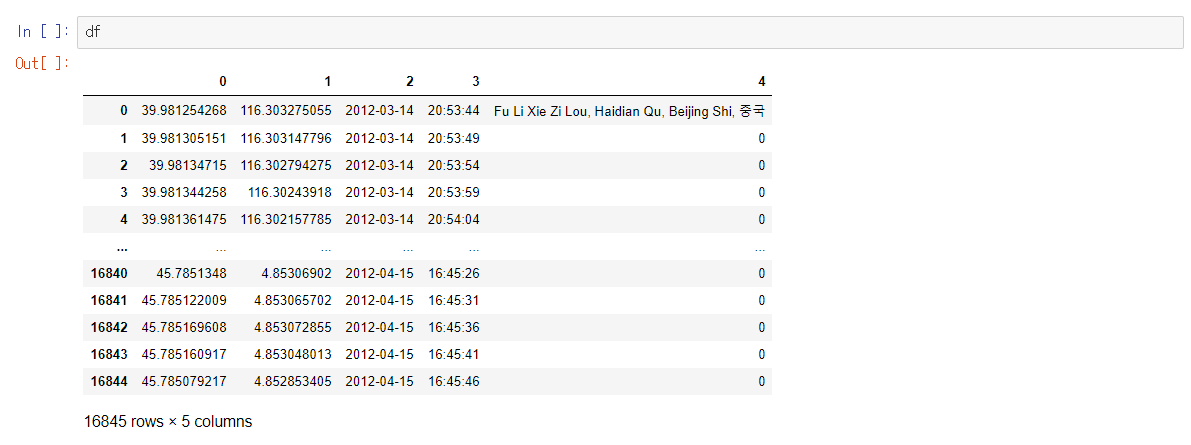


1. API 활성화

- 처음에 API를 활성화하기 위해서 “Google API Console - Google Cloud Platform”에 들어가서 사용자 인증 정보를 만들었다. 여기서 API 키를 새로 하나 생성했다. 여기까지만 하면 API가 자동으로 활성화된다고 생각하여 colab에서 코드를 실행시켜 보았으나 제대로 돌아가지 않았다. 여러 차례 코드를 수정했음에도 제대로 위치 정보를 불러올 수가 없었다. 코드 문제가 아닐 수도 있다는 생각이 들었다. 왜냐하면 구글 API 및 서비스에 들어가 대시보드를 살펴보니 트래픽이나 Geocoding API에 사용한 정보가 전혀 표시되지 않았기 때문이다. 이에 방향을 바꾸어 구글 API를 활성화시킬 수 있는 방법을 찾아보았고, 결국 라이브러리에 들어가 직접 구글 지도 관련 API를 활성화해 주어야 한다는 사실을 알게 되었다. 추가적으로 활성화를 하고 코드를 실행시켜보니 위치 정보를 획득할 수 있었다.

- 다음은 API 활성화를 통한 위치정보를 획득에 관한 결과이다. 성공적으로 위치 정보를 불러 올 수 있었음을 확인할 수 있다.





1. API를 불러오면서 코드 오류로 인한 시행착오들

엑셀에서 같은 그림에서 빈칸을 볼 수 있었다. 원래는 차있어야 하는 데이터라고 할 수 있다. 그러나 FOR 구문에서 철자 하나를 틀려서 10만건의 데이터를 모두 다 날려버린 경험을 했다. 이 데이터는 약 100만원 정도 육박하는 데이터 였으며, 이후에 사회에서 진출해서는 이 자체가 엄청난 손실을 입혔을 것이라고 생각이 된다. 돌다리도 두드려서 건너라는 말을 항상 세기면서 사회에 나가야겠다는 생각을 하게 하였다.



모델 학습시에 시행착오들

1. colab에서 사용할 때 gpu의 시간이 모두 다 되서 웨이트를 기록하지 못하는 어려움이 있었다. 이후에 저장되는 코드를 추가해서 모두 기록하도록 하였으며 파일의 인풋 아웃 풋에 엄청난 시간이 걸렸다.
2. gpu를 사용해보고 싶었지만 방법을 몰랐다.. 처음에는 파라미터에만 행렬을 집어넣으면 되는줄 알았지만 loss와 모델 자체에도 디바이스를 연결해야 한다는 것을 알게 되었다.
3. 인풋 사이즈를 맞춰주는 것에서 차원을 계산해주는 행위가 굉장한 어려움으로 다가왔다.
4. 처음에 트레인, 테스트, 벨리데이션 셋을 나누지 않고 모델을 만들었다. 결과를 만들어 냈지만 평가 지표가 없어서 모델을 잘 만들었는지 그렇지 않은지 알 수 없었다. 그러나 이후에 이 모델이 잘 만들어 졌는지를 확인하기 위해서 찾아보니 다음과 같은 방식으로 처리한다는 것을 알게 되었다. 이후에 코드를 수정하여 메트릭을 평가할 수 있었다. 그리고 10분, 20분, 30분 부터 시작하여 30분씩 미래 데이터를 예측해보고자 하였다.
5. 함수 내부에 텐서가 들어간다는 사실을 처음 알게 되었음
6. 텐서의 경우에 소수점 자리수가 줄어든다는 문제점이 있었다. 그렇기에 다음과 같은 코드를 찾는 것에 많은 시간이 들었다.
7. torch.set\_printoptions(precision=10)

**6. 결론**

이 한 개인의 예측 프로젝트를 하면서 가장 크게 느낀 것은 딥러닝은 엄청난 자원을 쏟아 붇고도 정답이 아닐 수 있다는 것이다. 특히 P100 GPU를 사용해 과거 12만건 정도의 행적을 한번 학습하는데 최소 10분이 걸렸는데 다수의 서비스를 하기에는 적합하지 못하다는 생각도 하였다. 그렇기에 구글 광고를 할 때는 로지스틱 regression 모델을 사용하는 것이라는 생각을 하였다. 또한 교수님께서 수업시간에 해주신 말씀 중 딥러닝과 머신러닝 둘 다 할 줄 알아야 한다는 점이 무슨 의미인지 깨달을 수 있었다. 즉, 산업공학적으로 생각했을 때, 딥러닝 모형을 한 개인에게 다음과 같이 구성한다는 것은 비합리적이라고 할 수 있다. 그렇기에 조금 더 가벼운 모델만으로 한 개인 혹은 차량의 이동 경로를 예측한다는 것이 훨씬 합리적인 선택이 될 수 있다는 것을 알게 되었다. 한편, pytorch라는 프레임 워크를 사용하는 역량과 복잡한 코드를 해석하는 역량 두가지를 키울 수 있었다. 팀원들과 교류하며 팀워크라는 측면도 많이 증가했다는 점은 이번 프로젝트를 통해서 소중한 깨닫음과 지혜를 주신 교수님께 감사드립니다.

**7. Reference**

[1] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[2]<https://www.quora.com/In-LSTM-how-do-you-figure-out-what-size-the-weights-are-supposed-to-be>

[3] <https://wikidocs.net/book/2155>

밑바닥부터 시작하는 딥러닝2,한빛미디어,p.238-p.286

[4] Alessandro Crivellari & Euro Beinat. (2020).LSTM-Based Deep Learning Model for Predicting Individual Mobility Traces of Short-Term Foreign Tourists.