4. 어탠션의 탄생

2장과 3장에 걸쳐서 언어 모델의 발전이 어떻게 발전해왔는지 알아봤다. 언어 모델은 N-Gram을 활용하는 확률적인 기법으로 발전하고 있었다. 그러다 2013년에 단어를 벡터로 표현하는 Word2Vec이 나오면서 언어 모델은 딥러닝의 범주로 들어오게 된다. Word2Vec을 계기로 기계가 마치 정말 사람처럼 문맥을 이해할 수 있다는 것을 알게 됐다. 그 이후 문맥을 조금 더 잘 이해하기 위해 단어의 연속적인 시퀀스를 이해하고자 RNN을 적용했고, 더 긴 시퀀스를 잘 기억할 수 있게 하기 위해 LSTM을 언어 모델에 적용했다.

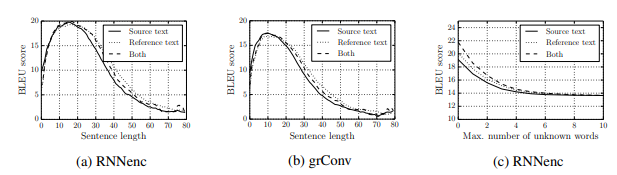
그렇다면 LSTM 기반의 언어 모델에는 어떤 문제가 있을까? 3장에서 설명했듯이 LSTM의 가장 큰 특징은 입력값 w1을 고정된 길이의 벡터로 출력하고(v1), 그 출력된 벡터를 다시 그 다음 입력(w2)와 같이 넣어서 v2를 출력하는 사이클을 반복한다는 것이다. 이것을 Auto-Regressive 하다고 한다.

[LSTM의 Auto-Regressive한 특징]

4.1. 하나의 벡터로 모든 정보를 담는 LSTM

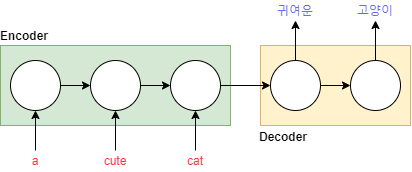
"6평짜리 방에서 살면 어떨까? 혼자 산다고 가정하면 그래도 어느 정도 괜찮을 것이다. 그런데 친구 5명이서 그 방에서 산다면 어떨까? 잠을 잘 때도 서로 엉겨붙어서 자야 할 것이다."

LSTM도 위와 유사한 단점을 갖는다. 입력 문장의 총 길이에 상관없이 LSTM은 그 입력을 고정된 길이의 벡터로 인코딩한다. 그렇다면 입력문장이 길어지면 길어질수록 많은 정보를 벡터에 넣어야 하기 때문에 인코딩이 힘들어질 수 밖에 없다. 실제로 문장의 길이가 길어질수록 LSTM의 성능이 떨어진다는 연구결과가 있다.



[그림1: 시퀀스 길이가 길어질수록 떨어지는 LSTM 성능]  
(https://arxiv.org/pdf/1409.1259.pdf)

LSTM으로 번역 시스템을(NMT) 만든다고 가정해보자. NMT는 기본적으로 입력 문장을(Sequence) 번역문(Sequence)으로 바꿔주는 Sequence-To-Sequence 구조를 갖게 된다. 입력 문장의 각 단어를 하나하나 LSTM으로 인코딩해서 나온 마지막 결과를 활용하여(h3) 번역문의 각 단어를 하나 하나 만드는 것이다.



[그림2: Seq2Seq 모델 예시]

(encoder/decoder)

%3CmxGraphModel%3E%3Croot%3E%3CmxCell%20id%3D%220%22%2F%3E%3CmxCell%20id%3D%221%22%20parent%3D%220%22%2F%3E%3CmxCell%20id%3D%222%22%20value%3D%22%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3BfillColor%3D%23d5e8d4%3BstrokeColor%3D%2382b366%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%2270%22%20y%3D%22165%22%20width%3D%22230%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%223%22%20value%3D%22%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3BfillColor%3D%23fff2cc%3BstrokeColor%3D%23d6b656%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22330%22%20y%3D%22165%22%20width%3D%22150%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%224%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%225%22%20target%3D%2220%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%225%22%20value%3D%22%22%20style%3D%22ellipse%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3Baspect%3Dfixed%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22240%22%20y%3D%22180%22%20width%3D%2250%22%20height%3D%2250%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%226%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BentryX%3D0%3BentryY%3D0.5%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%227%22%20target%3D%225%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%227%22%20value%3D%22%22%20style%3D%22ellipse%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3Baspect%3Dfixed%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22160%22%20y%3D%22180%22%20width%3D%2250%22%20height%3D%2250%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%228%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BentryX%3D0%3BentryY%3D0.5%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%229%22%20target%3D%227%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%229%22%20value%3D%22%22%20style%3D%22ellipse%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3Baspect%3Dfixed%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%2280%22%20y%3D%22180%22%20width%3D%2250%22%20height%3D%2250%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2210%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BentryX%3D0.5%3BentryY%3D1%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2211%22%20target%3D%229%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2211%22%20value%3D%22a%22%20style%3D%22text%3Bhtml%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3BwhiteSpace%3Dwrap%3Brounded%3D0%3BfontColor%3D%23FF3333%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%2285%22%20y%3D%22270%22%20width%3D%2240%22%20height%3D%2220%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2212%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%2213%22%20target%3D%227%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2213%22%20value%3D%22cute%22%20style%3D%22text%3Bhtml%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3BwhiteSpace%3Dwrap%3Brounded%3D0%3BfontColor%3D%23FF3333%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22165%22%20y%3D%22270%22%20width%3D%2240%22%20height%3D%2220%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2214%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BentryX%3D0.5%3BentryY%3D1%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2215%22%20target%3D%225%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2215%22%20value%3D%22cat%22%20style%3D%22text%3Bhtml%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3BwhiteSpace%3Dwrap%3Brounded%3D0%3BfontColor%3D%23FF3333%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22245%22%20y%3D%22270%22%20width%3D%2240%22%20height%3D%2220%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2216%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BentryX%3D0.5%3BentryY%3D1%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2217%22%20target%3D%2222%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2217%22%20value%3D%22%22%20style%3D%22ellipse%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3Baspect%3Dfixed%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22420%22%20y%3D%22180%22%20width%3D%2250%22%20height%3D%2250%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2218%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BentryX%3D0%3BentryY%3D0.5%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2220%22%20target%3D%2217%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2219%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BentryX%3D0.5%3BentryY%3D1%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2220%22%20target%3D%2221%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2220%22%20value%3D%22%22%20style%3D%22ellipse%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3Baspect%3Dfixed%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22340%22%20y%3D%22180%22%20width%3D%2250%22%20height%3D%2250%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2221%22%20value%3D%22%EA%B7%80%EC%97%AC%EC%9A%B4%22%20style%3D%22text%3Bhtml%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3BwhiteSpace%3Dwrap%3Brounded%3D0%3BfontColor%3D%236666FF%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22345%22%20y%3D%22120%22%20width%3D%2240%22%20height%3D%2220%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2222%22%20value%3D%22%EA%B3%A0%EC%96%91%EC%9D%B4%22%20style%3D%22text%3Bhtml%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3BwhiteSpace%3Dwrap%3Brounded%3D0%3BfontColor%3D%236666FF%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22425%22%20y%3D%22120%22%20width%3D%2240%22%20height%3D%2220%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2223%22%20value%3D%22%26lt%3Bb%26gt%3BEncoder%26lt%3B%2Fb%26gt%3B%22%20style%3D%22text%3Bhtml%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3BwhiteSpace%3Dwrap%3Brounded%3D0%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%2274%22%20y%3D%22145%22%20width%3D%2240%22%20height%3D%2220%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2224%22%20value%3D%22%26lt%3Bb%26gt%3BDecoder%26lt%3B%2Fb%26gt%3B%22%20style%3D%22text%3Bhtml%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3BwhiteSpace%3Dwrap%3Brounded%3D0%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22333%22%20y%3D%22246%22%20width%3D%2240%22%20height%3D%2220%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3C%2Froot%3E%3C%2FmxGraphModel%3E

직관적으로 생각해보자. 위의 Seq2Seq 구조의 모델이 훌륭한 번역기로 학습되려면 h3가 입력문장의 모든 단어를 표현하고 있어야 한다. 짧은 문장에서는 가능할지 모르겠지만 문장이 길수록 힘들어질 것이란 것을 직관적으로 이해할 수 있다.

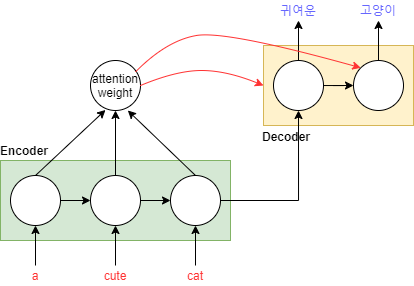
4.2. 왜 어텐션(Attention)하지 않지?

앞 절에서 LSTM은 입력 데이터를 하나의 벡터로 표현하는 구조를 가지고 있고, 이러한 특성 때문에 입력 데이터가 길어질 경우 입력을 하나의 벡터로 표현하기가 힘들어진다는 것을 설명했다. 그렇다면 어떻게 이 문제를 해결할 수 있을까? 필요한 부분만 집중해서(Attention) 보면 된다. 번역기에서 입력 문장을 번역문으로 바꿀 때 각 입력 단어를 해석할 때 집중해야 되는 부분이 정해져 있다. 아래의 빈 칸 채우기 문제를 봐보자.

ENG: My name is Dooli.

KOR: 내 이름은 둘리\_\_.

빈 칸에 들어갈 정답은 "입니다", "이다" 등이 적절하다. 저 빈 칸을 채우기 위해서 "My name" "Dooli" 등에 집중할 이유가 있을까? 물론 부분적으로 있지만, 가장 집중해야 되는 부분은 "is"이다. LSTM에서 집중을 해야하는 특정 부분에만 집중하도록 weight을 주는 것이 바로 어텐션 메커니즘이다. [그림2]에 어텐션 네트워크를 추가하면 아래와 같은 구조가 된다.



[그림3: Seq2Seq with Attention]

디코더에서 번역 단어를 하나 하나 출력할 때마다 어텐션 가중치를 추가해주는데, 이 어텐션 가중치는 인코더의 output 정보를 통해서 계산된다. [그림2]와 [그림3]을 비교해보자. [그림2]에서 디코더를 연산할 때 인코더의 히든스테이트 값만 들어간다. 하지만 [그림3]에서는 디코더에 인코더의 히든스테이트 값에 인코더의 아웃풋 값이 추가된다. 인코더의 아웃풋은 T개의 값으로 이루어져 있는데, 그 T개의 아웃풋이 가중치로 계산되어 디코더의 입력으로 사용된다. 어텐션 네트워크에 대한 자세한 구조는 다음 절에서 자세하게 설명할 예정이다.

4.3. 어떻게 어텐션(Attention)하지?

이전의 절에서 어텐션을 왜 해야하는지에 대한 직관적인 이해를 해봤다. Seq2Seq 구조는 인코더와 디코더의 구조로 이루어져 있는데, 시퀀스의 길이가 길어질수록 인코더에서 시퀀스를 효과적으로 표현할 수 없기 때문에 무언가 플러스 알파가 필요한 것이다. 그것이 어텐션이며, 그 어텐션은 인코더의 값을 참고해서 가중치를 만든 후 디코더의 값을 만들어가는 것이다.

이번 절에서는 어텐션에 대한 구체적인 이야기를 해보려고 한다. 어텐션을 공부할 때 이해해야 하는 쿼리, 키, 벨류(Query, Key, Value)가 무엇인지 그리고 그것을 이용해서 어떻게 어탠션을 구현할 수 있는지 알아보자.

4.3.1. 묻고 참고하고 답하기

'''

영어 울렁증이 심한 윤우아빠가 길을 걸어가고 있는데, 갑자기 파란 눈의 외쿡인이 말을 걸었다.

외쿡인: I want to go to the N-Tower. Where can I take a bus?

윤우아빠: 음.... 엔타워 버스 스테이션... 인프론트오브 맬도날드... 쓰리투제로 버스...

'''

영어 울렁증이 심한 윤우아빠에게 언제라도 있을 수 있는 일이다. 외국인이 유창한 영어로 물어봐도 윤우아빠 귀에는 몇 단어 들리지 않았을 것이다.

[그림4: n-tower, go, bus, where 쿼리]

윤우아빠는 이 몇 단어를 조합해서 머리 속에 있는 몇 개 없는 영어 단어와 매칭해서 외국인이 뭘 원하는지 이해하려고 쩔쩔매고 있었을 것이다.

[그림5: 단어 키-밸류]

이렇게 저렇게 조합하여 윤우아빠는 드디어 외국인이 원하는 그림을 머리속으로 그려낸다.

[그림6: 컨택스트 그림]

이것이 쿼리, 키, 밸류의 기본적인 개념이다. 쿼리는 외국인이 말한 문장에서 들은 몇 개의 단어들(N-Tower, go, bus, where)이다. 그리고 윤우아빠가 외구고 있는 영어-한국어 단어가 각각 키, 밸류이다. 문법을 모르는 윤우아빠는 자기가 들은 몇 개의 단어만을 조합해서 상황을 판단해야 하는데, 최대한 자기가 이미 알고 있는 것과 유사한 조합으로 상황을 이해하려고 노력하게 된다.

[그림7: 확률그림]

[그림7]을 보면 리스닝한 영어 단어마다 연상되는 영어단어를 확률적으로 매칭했고, 윤우아빠는 그 모든 결과를 조합해서 드디어 그 상황(컨택스트)를 이해했다.

익숙한 생활 속 예시로 쿼리, 키, 밸류를 설명했으니 코드와 함꼐 이해해보자. 코드를 설명하면서 수식도 간단하게 살펴보려고 한다.

4.3.2. 어탠션 구현하기

어탠션을 이해했으니 간단하게 어탠션을 구현해보자. 실제 번역기 등을 만들어볼 수도 있지만 여기에서는 간단하게 알파벳을 거꾸로 뒤집는 Seq2Seq 모델을 만들어보려고 한다.

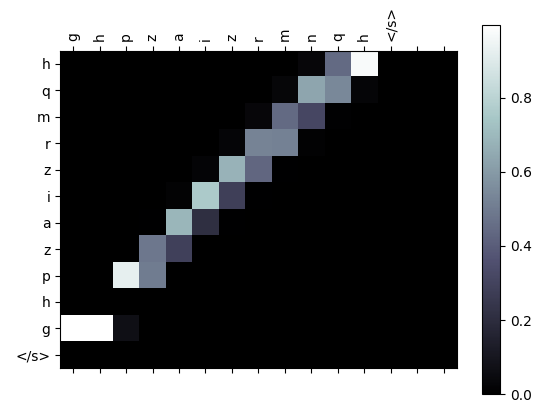
'''

ex-1) pqovqh → hqvoqp

ex-2) vnwqjkm → mkjqwnv

'''

위와 같은 모델을 어탠션을 활용해서 만들면 어떤 결과가 나오게 될까? ex-1의 제일 마지막 알파벳 h를 예측할때 출력값의 가장 처음 부분에 가중치를 많이 주면 된다. 마찬가지 원리로 ex-1의 제일 첫번째 알파벳인 p를 예측할 때는 출력값의 가장 마지막 부분에 가중치를 많이 주면 된다. 따라서 그림으로 그려보면 아래와 같이 어탠션 가중치를 시각화할 수 있다.



[그림8: 어탠션 가중치 시각화]

이제 구현을해보자. 전체 소스코드는 이 책의 레포지토리 /chapter4/attention 폴더에 있다.

### 데이터셋 생성하기

우선 랜덤 알파벳 스트링을 생성하는 함수를 만들어보자.

'''

def generate\_random\_alphabet\_index():

random\_length = np.random.randint(10, MAX\_LENGTH-2) # -2 because of <s> and </s>

random\_alphabet\_index = np.random.randint(0, 26, random\_length) + 3

return random\_alphabet\_index.tolist(), random\_length

'''

위의 함수를 실행하면 아래와 같이 랜덤한 길이의 알파벳 인덱스 리스트와 그 길이를 얻을 수 있다.

'''

>>> from dataset import generate\_random\_alphabet\_index

>>> generate\_random\_alphabet\_index()

([21, 6, 26, 23, 28, 15, 18, 4, 21, 15, 19, 19], 12)

>>> generate\_random\_alphabet\_index()

([13, 22, 19, 4, 10, 19, 25, 27, 6, 23, 25, 14, 18], 13)

'''

보통 AI 모델에 입력을 넣을 때는 데이터의 길이가 고정이어야 한다. 여기에서는 최대 길이를 15로 정의하자. 그러면 최대 길이 15 이내에 대해서 </s>와 <pad> 등을 추가해줘야 한다. 이러한 작업을 AlphabetToyDataset 클래스를 통해서 해보자.

'''

from torch.utils.data import Dataset

# 데이터셋 만들기

class AlphabetToyDataset(Dataset):

def \_\_init\_\_(self, n\_dataset=1000):

bos = 0

eos = 1

pad = 2

self.inputs = []

self.labels = []

self.length = []

for \_ in range(n\_dataset):

# make input example

aindex, alen = generate\_random\_alphabet\_index()

# index to alphabet

#alphabet = list(map(lambda a: i2a[a], aindex))

# inversing

#inversed\_alphabet = list(map(lambda a: inverse\_map[a], alphabet))

# alphabet to index

#iindex = list(map(lambda ia: a2i[ia], inversed\_alphabet))

iindex = aindex[::-1]

# add bos, eos and pad

n\_pad = MAX\_LENGTH - len(aindex) - 1

aindex = aindex + [eos] + [pad]\*n\_pad

iindex = iindex + [eos] + [pad]\*n\_pad

# add to examples

self.inputs.append(aindex)

self.labels.append(iindex)

self.length.append(alen)

def \_\_len\_\_(self):

return len(self.inputs)

def \_\_getitem\_\_(self, index):

return [

torch.tensor(self.inputs[index], dtype=torch.long),

torch.tensor(self.labels[index], dtype=torch.long),

torch.tensor(self.length[index], dtype=torch.long)

]

'''

위의 AlphabetToyDataset을 클래스는 generate\_random\_alphabet\_index 함수를 통해서 생성된 랜덤 알파벳 a부터 z까지를 각각 3부터 28까지 치환하여 inputs을 만들고, 그것을 역순으로 배열하여 labels를 만든 것이다. 즉, 랜덤하게 생성된 알파벳을 역순으로 배열하는 간단한 AI 모델을 만들기 위한 데이터셋이다.

AlphabetToyDataset을 아래와 같이 사용해서 데이터셋을 만들수 있다.

'''

train\_dataset = AlphabetToyDataset(n\_dataset=3000)

valid\_dataset = AlphabetToyDataset(n\_dataset=300)

'''

### 데이터로더 생성하기

AlphabetToyDataset으로 train\_dataset과 valid\_dataset을 만들고, 그것을 DataLoader를 이용해서 AI 모델에 입력할 수 있도록 배치 사이즈도 설정해줄 수 있다.

'''

import torch

from torch.nn.utils.rnn import pad\_sequence

from torch.utils.data import DataLoader

def collate\_fn(batch):

inputs = pad\_sequence([b[0] for b in batch], batch\_first=True)

targets = pad\_sequence([b[1] for b in batch], batch\_first=True)

lengths = torch.stack([b[2] for b in batch])

lengths, indice = torch.sort(lengths, descending=True)

inputs = inputs[indice]

targets = targets[indice]

return inputs, targets, lengths

train\_dataloader = DataLoader(train\_dataset, collate\_fn=collate\_fn, batch\_size=16)

valid\_dataloader = DataLoader(valid\_dataset, collate\_fn=collate\_fn, batch\_size=1)

'''

for문을 통해서 DataLoader가 내뱉어내는 데이터를 검토해보자.

'''

tensor([[ 6, 24, 18, 24, 6, 16, 20, 27, 20, 18, 24, 19, 1, 2, 2],

[25, 14, 18, 19, 23, 7, 13, 28, 3, 24, 25, 21, 1, 2, 2],

[ 4, 7, 25, 18, 16, 17, 27, 27, 21, 10, 6, 19, 1, 2, 2],

[12, 9, 12, 16, 7, 20, 18, 19, 14, 18, 24, 1, 2, 2, 2],

[18, 16, 19, 14, 4, 18, 8, 4, 7, 10, 7, 1, 2, 2, 2],

[20, 7, 20, 20, 5, 7, 21, 20, 14, 10, 25, 1, 2, 2, 2],

[18, 25, 20, 7, 6, 24, 25, 22, 20, 14, 10, 1, 2, 2, 2],

[27, 3, 28, 28, 4, 18, 26, 28, 22, 21, 16, 1, 2, 2, 2],

[16, 25, 27, 15, 20, 25, 11, 10, 13, 14, 16, 1, 2, 2, 2],

[16, 14, 13, 3, 3, 20, 9, 14, 16, 3, 1, 2, 2, 2, 2],

[20, 26, 27, 23, 26, 15, 19, 24, 8, 23, 1, 2, 2, 2, 2],

[13, 27, 19, 21, 12, 11, 4, 11, 8, 21, 1, 2, 2, 2, 2],

[ 6, 28, 3, 27, 4, 14, 11, 23, 5, 22, 1, 2, 2, 2, 2],

[17, 16, 5, 8, 6, 12, 28, 17, 27, 14, 1, 2, 2, 2, 2],

[22, 26, 20, 19, 10, 8, 18, 17, 17, 10, 1, 2, 2, 2, 2],

[25, 20, 28, 13, 20, 4, 11, 19, 24, 5, 1, 2, 2, 2, 2]])

tensor([[19, 24, 18, 20, 27, 20, 16, 6, 24, 18, 24, 6, 1, 2, 2],

[21, 25, 24, 3, 28, 13, 7, 23, 19, 18, 14, 25, 1, 2, 2],

[19, 6, 10, 21, 27, 27, 17, 16, 18, 25, 7, 4, 1, 2, 2],

[24, 18, 14, 19, 18, 20, 7, 16, 12, 9, 12, 1, 2, 2, 2],

[ 7, 10, 7, 4, 8, 18, 4, 14, 19, 16, 18, 1, 2, 2, 2],

[25, 10, 14, 20, 21, 7, 5, 20, 20, 7, 20, 1, 2, 2, 2],

[10, 14, 20, 22, 25, 24, 6, 7, 20, 25, 18, 1, 2, 2, 2],

[16, 21, 22, 28, 26, 18, 4, 28, 28, 3, 27, 1, 2, 2, 2],

[16, 14, 13, 10, 11, 25, 20, 15, 27, 25, 16, 1, 2, 2, 2],

[ 3, 16, 14, 9, 20, 3, 3, 13, 14, 16, 1, 2, 2, 2, 2],

[23, 8, 24, 19, 15, 26, 23, 27, 26, 20, 1, 2, 2, 2, 2],

[21, 8, 11, 4, 11, 12, 21, 19, 27, 13, 1, 2, 2, 2, 2],

[22, 5, 23, 11, 14, 4, 27, 3, 28, 6, 1, 2, 2, 2, 2],

[14, 27, 17, 28, 12, 6, 8, 5, 16, 17, 1, 2, 2, 2, 2],

[10, 17, 17, 18, 8, 10, 19, 20, 26, 22, 1, 2, 2, 2, 2],

[ 5, 24, 19, 11, 4, 20, 13, 28, 20, 25, 1, 2, 2, 2, 2]])

tensor([12, 12, 12, 11, 11, 11, 11, 11, 11, 10, 10, 10, 10, 10, 10, 10])

torch.Size([16, 15]) torch.Size([16, 15]) torch.Size([16])

'''

### 모델링 구현하기

이 데이터를 이용해서 학습할 모델을 만들어보자. 기본적으로 Seq2Seq 모델을 어텐션과 함께 구현했다. 레포지토리 상에서 /chapter4/research/attention/seq2seq.py에 구현돼 있다.

'''

class EncoderRNN(nn.Module):

def \_\_init\_\_(self, input\_size, hidden\_size):

super(EncoderRNN, self).\_\_init\_\_()

self.hidden\_size = hidden\_size

self.embedding = nn.Embedding(input\_size, hidden\_size)

self.gru = nn.GRU(hidden\_size, hidden\_size, batch\_first=True)

def forward(self, inputs, hidden):

'''

Input Parameters

- inputs: (B,M)

- hidden: (1,B,H)

Output returns

- output: (B,1,O)

- hidden: (B,1,H)

- attn\_weights: (B,1,M)

Logging outputs

\*\* output: torch.Size([16, 1])

\*\* hidden: torch.Size([16, 1])

'''

#print('\*\* inputs: {}'.format(inputs.shape))

#print('\*\* hidden: {}'.format(hidden.shape))

embedded = self.embedding(inputs) # (B,M,H)

#print('\*\* embedded: {}'.format(embedded.shape))

output, hidden = self.gru(embedded, hidden) # (B,M,H), (1,B,H)

return output, hidden

def initHidden(self, batch\_size):

return torch.zeros(1, batch\_size, self.hidden\_size, device=device)

# with attention

class AttnDecoderRNN(nn.Module):

def \_\_init\_\_(self, hidden\_size, output\_size, dropout\_p=0.1, max\_length=MAX\_LENGTH):

super(AttnDecoderRNN, self).\_\_init\_\_()

self.hidden\_size = hidden\_size

self.output\_size = output\_size

self.dropout\_p = dropout\_p

self.max\_length = max\_length

self.embedding = nn.Embedding(self.output\_size, self.hidden\_size)

self.attn = nn.Linear(self.hidden\_size \* 2, self.max\_length)

self.attn\_combine = nn.Linear(self.hidden\_size \* 2, self.hidden\_size)

self.dropout = nn.Dropout(self.dropout\_p)

self.gru = nn.GRU(self.hidden\_size, self.hidden\_size, batch\_first=True)

self.out = nn.Linear(self.hidden\_size, self.output\_size)

def forward(self, inputs, hidden, encoder\_outputs):

embedded = self.embedding(inputs) # (B,1,H)

embedded = self.dropout(embedded)

# query: embedded

# key: hidden

# value: encoder\_outputs

attn\_weights = F.softmax(

self.attn(

torch.cat((embedded, hidden.transpose(0, 1)), -1) # (B,1,2H)

), # (B,1,M)

dim=-1) # (B,1,M)

#print('\*\* attn\_weights: {}'.format(attn\_weights.shape))

#print('\*\* attn\_weights: {}'.format(attn\_weights))

#print('\*\* attn\_weights.sum(): {}'.format(attn\_weights.sum().shape))

#print('\*\* attn\_weights.sum(): {:.4f}'.format(attn\_weights.sum()))

attn\_applied = torch.bmm(attn\_weights, encoder\_outputs) # (B,1,H)

#print('\*\* attn\_applied: {}'.format(attn\_applied.shape))

#return

output = torch.cat((embedded, attn\_applied), -1) # (B,1,2H)

#print('\*\* output: {}'.format(output.shape))

output = self.attn\_combine(output) # (B,1,H)

#print('\*\* output: {}'.format(output.shape)) # (B,1,H)

output = F.relu(output)

#print('\*\* output: {}'.format(output.shape)) # (B,1,H)

#print('\*\* hidden: {}'.format(hidden.shape)) # (B,1,H)

output, hidden = self.gru(output, hidden) # (B,1,H) (1,B,H)

#print('\*\* gru-output: {}'.format(output.shape))

#print('\*\* gru-hidden: {}'.format(hidden.shape))

output = F.log\_softmax(self.out(output), dim=-1)

#print('\*\* final-output: {}'.format(output.shape))

#print('\*\* final-hidden: {}'.format(hidden.shape))

#print('\*\* final-attn\_weights: {}'.format(attn\_weights.shape))

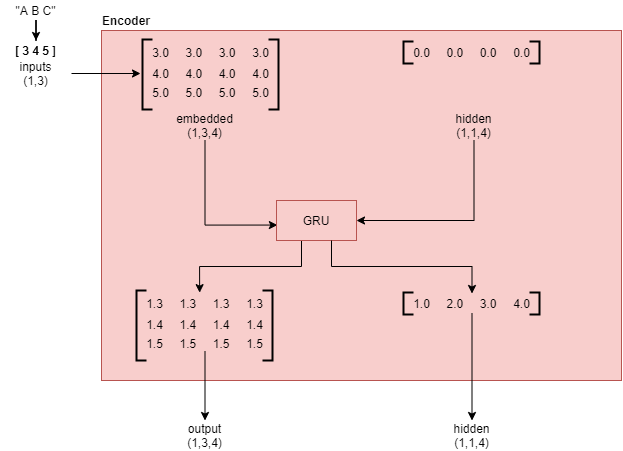
return output, hidden, attn\_weights

def initHidden(self, batch\_size):

return torch.zeros(1, batch\_size, self.hidden\_size, device=device)

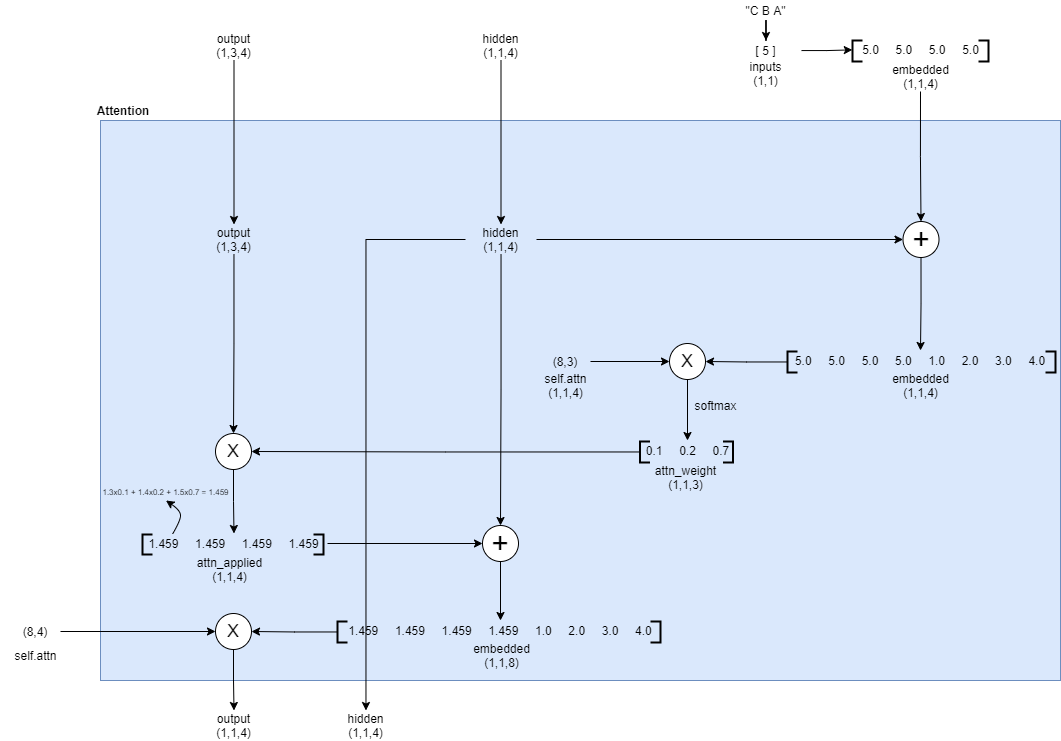
'''

모델링 부분은 조금 더 자세하게 설명해보려고 한다. 이 모델은 알파벳 시퀀스를 역순으로 배열하는 Seq2Seq 모델임을 상기하며 아래의 그림을 보라.



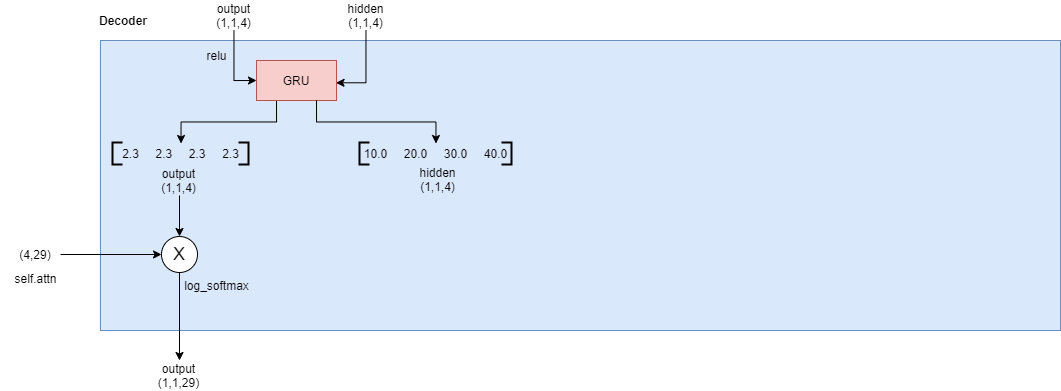
[그림9: 알파벳 시퀀스 어텐션 매커니즘 - 인코더]

[그림9]는 [3 4 5]가 `inputs`으로 들어갈 때 인코더에서 일어나는 연산 과정을 그림으로 나타낸 것이다. 인코더 부분은 간단하다. 우선 `inputs`를 임베딩하여 `embedded`가 됐다. (##각주1 임베딩한 값을 3.0 4.0 5.0 등으로 체웠는데, 이는 이해를 쉽게 하기 위해 친숙한 숫자로 표현한 것이다. 실제로는 랜덤한 값으로 임베딩된다.) `embedded`는 `hidden`과 함께 GRU에 입력된다. GRU는 `output`과 `hidden`을 리턴한다. 여기에서 `hidden`은 [3 4 5]라는 시퀀스를 하나의 벡터로 나타낸 것이다. 여기에서 `hidden`만으로는 `inputs` 시퀀스를 모두 표현하기 힘들기 때문에 디코더에서 `output`을 이용해서 어탠션을 적용한다.



[그림10: 알파벳 시퀀스 어텐션 매커니즘 - 디코더:어텐션]

인코더에서 [3 4 5]를 `inputs`으로 넣었을 때 기대하는 디코더의 `output`은 [5 4 3]이다. 디코더에서 어텐션이 적용되는 방법이 [그림10]에 표현돼 있다. Seq2Seq 모델의 디코더는 [5 4 3]의 값을 하나씩 타임스텝마다 넣어준다. [5] 하나가 [그림10]의 `inputs`이다. `inputs`을 임베딩하여 `embedded`를 만들고 그것을 인코더의 `hidden`과 이어붙인다. 그렇게 이은 벡터를 `attn`을 통해서 행렬곱을 해준다. (##각주2 행렬곱의 의미는 선형대수학에서의 projection이다. 즉 다른 차원의 점으로 이동시키는 것이다.) 여기에 softmax를 취하면 `attn\_weight`을 얻을 수 있다.

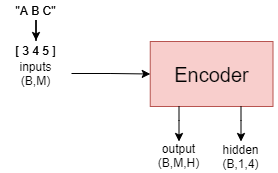


[그림11: 알파벳 시퀀스 어텐션 매커니즘 - 디코더:디코딩]

`attn\_weights`은 softmax를 취한 값이기 때문에 총 합이 1이고 길이는 `MAX\_LENGTH`이다. 이 값을 인코더의 `encoder\_outputs`에 곱해주면 인코더 `inputs` [3 4 5]에 가중치를 부여하는 셈이고 그것이 `attn\_applied`이다. `attn\_applied`와 `embedded`를 이어붙여보자. 이 두 값을 이어 붙인다는 것은 인코더의 입력 값과 디코더의 입력 값을 이어 붙인 셈이다. 이 값을 `attn\_combine`을 이용해 행렬 곱을 해서 relu 연산을 한 다음에 GRU 셀에 입력으로 넣는다. 핵심은 인코더와 디코더의 입력 값을 합쳐서 GRU에 넣는다는 것이고 인코더의 입력은 어텐션 가중치가 적용된 값이다.

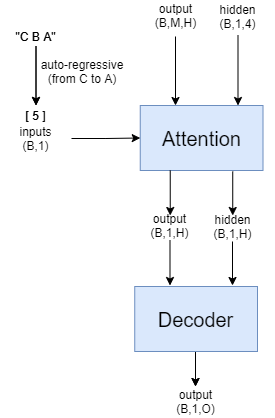
모델링하는 부분을 요약해보자.

우선 인코더를 만든다. 인코더의 입력인 `inputs`은 (batch\_size,max\_length) 형태의 벡터이다. 인코더의 출력은 `outputs`과 `hidden`인데, `outputs`은 (batch\_size,max\_length,hidden\_size)이며 `hidden`은 (batch\_size,1,hidden\_size)이다.



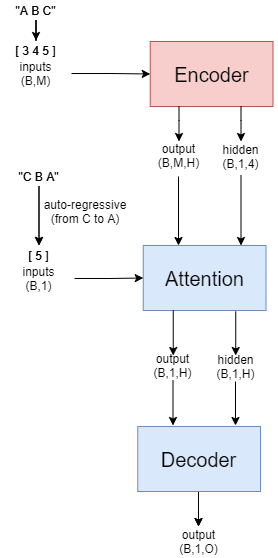
[그림12: 인코더 요약]

그 다음에 디코더를 만드는데 디코더는 인코더처럼 하나의 시퀀스를 통채로 처리하지 않고 하나 하나(auto-regressive) 시퀀스 길이만큼 반복하여 처리한다. 디코더가 입력으로 받아들이는 `inputs`은 (batch\_size, 1)과 앞서 계산했던 인코더의 출력인 `outputs`와 `hidden`이다. 이 과정을 디코더의 시퀀스 길이만큼 반복하게 된다.



[그림13: 디코더 요약]

인코더와 디코더의 입력과 출력을 하나로 정리하면 [그림14]와 같다.



[그림14: 인코더/디코더 요약]

##팁: 모델링을 하기 전에 전체적인 입력과 출력의 형태를 정리한 후에 코딩을 하면 훨씬 간단하고 정리된 코드를 작성할 수 있다.

### 모델링 학습하기

`trainIters`함수를 이용해서 Seq2Seq 모델을 학습할 수 있다. `trainIters`내에는 `train`함수를 사용하고 있고 SGD 옵티마이져와 NLLLoss를 이용한다.

```

def train(input\_tensor, target\_tensor, encoder, decoder, encoder\_optimizer, decoder\_optimizer, criterion, max\_length=MAX\_LENGTH, with\_attention=True):

batch\_size = input\_tensor.size(0)

encoder\_hidden = encoder.initHidden(batch\_size)

encoder\_optimizer.zero\_grad()

decoder\_optimizer.zero\_grad()

input\_length = input\_tensor.size(1)

target\_length = target\_tensor.size(1)

encoder\_outputs = torch.zeros(max\_length, encoder.hidden\_size, device=device)

loss = 0

encoder\_outputs, encoder\_hidden = encoder(input\_tensor, encoder\_hidden)

decoder\_input = torch.tensor([bos]\*batch\_size, device=device)

decoder\_input = decoder\_input.unsqueeze(-1) # (B,1)

decoder\_hidden = encoder\_hidden

use\_teacher\_forcing = True if np.random.random() < teacher\_forcing\_ratio else False

for di in range(target\_length):

decoder\_output, decoder\_hidden, decoder\_attention = decoder(

decoder\_input, # (B,1)

decoder\_hidden, # (1,B,H)

encoder\_outputs # (B,M,H)

)

decoder\_output = decoder\_output.squeeze(1)

loss += criterion(decoder\_output, target\_tensor[:,di])

if use\_teacher\_forcing:

decoder\_input = target\_tensor[:,di].unsqueeze(-1) # (B,1)

else:

topv, topi = decoder\_output.topk(1)

decoder\_input = topi # (B,1)

loss.backward()

encoder\_optimizer.step()

decoder\_optimizer.step()

return loss.item() / target\_length

def trainIters(encoder, decoder, n\_iters, print\_every=1000, plot\_every=100, learning\_rate=0.01, with\_attention=True):

start = time.time()

plot\_losses = []

print\_loss\_total = 0 # print\_every 마다 초기화

plot\_loss\_total = 0 # plot\_every 마다 초기화

encoder\_optimizer = optim.SGD(encoder.parameters(), lr=learning\_rate)

decoder\_optimizer = optim.SGD(decoder.parameters(), lr=learning\_rate)

criterion = nn.NLLLoss()

for iter, batch in enumerate(train\_dataloader):

input\_tensor, target\_tensor, length\_tensor = batch

input\_tensor = input\_tensor.to(device)

target\_tensor = target\_tensor.to(device)

length\_tensor = length\_tensor.to(device)

loss = train(input\_tensor, target\_tensor, encoder,

decoder, encoder\_optimizer, decoder\_optimizer, criterion, with\_attention=with\_attention)

print\_loss\_total += loss

plot\_loss\_total += loss

if (iter+1) % print\_every == 0:

print\_loss\_avg = print\_loss\_total / print\_every

print\_loss\_total = 0

print('%s (%d %d%%) %.4f' % (timeSince(start, (iter+1) / n\_iters),

(iter+1), (iter+1) / n\_iters \* 100, print\_loss\_avg))

if (iter+1) % plot\_every == 0:

plot\_loss\_avg = plot\_loss\_total / plot\_every

plot\_losses.append(plot\_loss\_avg)

plot\_loss\_total = 0

showPlot(plot\_losses)

```

위의 코드를 자세하게 설명하지는 않을 것이다. 레포지토리에 있는 코드와 주피터 노트북을 통해서 스스로 학습해보도록 하자. `trainIters`함수는 아래와 같이 실행하면 된다.

```

for \_ in range(15):

trainIters(encoder1, decoder1, 75000, print\_every=30, with\_attention=True)

0m 0s (- 61m 34s) (10 0%) 3.1772

0m 0s (- 60m 42s) (20 0%) 3.1353

0m 1s (- 61m 26s) (30 0%) 2.9836

0m 2s (- 63m 3s) (40 0%) 2.9314

0m 2s (- 64m 14s) (50 0%) 2.8756

0m 3s (- 64m 57s) (60 0%) 2.7924

0m 3s (- 65m 27s) (70 0%) 2.9195

0m 4s (- 65m 41s) (80 0%) 2.8201

0m 4s (- 65m 34s) (90 0%) 2.8958

0m 5s (- 65m 24s) (100 0%) 2.7099

0m 5s (- 65m 15s) (110 0%) 2.7981

0m 6s (- 65m 20s) (120 0%) 2.7033

0m 6s (- 65m 24s) (130 0%) 2.7576

0m 7s (- 65m 39s) (140 0%) 2.7662

0m 7s (- 65m 47s) (150 0%) 2.7134

0m 8s (- 66m 11s) (160 0%) 2.6584

0m 9s (- 66m 19s) (170 0%) 2.7286

0m 9s (- 66m 26s) (180 0%) 2.6278

```

[코드1: 모델링 학습 코드 실행 예시]