딥러닝 프레임워크 Attention 조사

팀원
202184042 ICT공학부 이재성
201904192 ICT공학부 김성중
201904236 산업데이터사이언스 전병준
201904217 산업데이터사이언스 우사랑

매칭 기업
㈜테라텍

Summary

Attention은 RNN 기반 seq2seq 모델의 성능 향상을 위해 도입되었으며, 현재는 seq2seq를 대체하는 방법으로 사용되고 있습니다. Attention은 입력 데이터의 중요도를 동적으로 계산하여 출력에 반영하는 메커니즘입니다. Attention의 구조에는 Dot-Product, Scaled Dot-Product, Additive, Multi-Head 등 다양한 유형이 있습니다. Attention은 현재 기계 번역, 이미지 처리, 그래프 분석 등 다양한 분야에서 활용되고 있습니다.

Attention 탄생

어텐션 매커니즘이란?
'모히또 가서 몰디브나 한잔 하자'라는 대사는 어느 한 영화의 명대사입니다. 이 영화가 어느 영화인지 물어보는 질문에 답하기 위해 내부자들의 모든 장면과 대사를 숙지하고 있을 필요가 젼혀 없습니다. '모히또'와 '몰디부' 두 단어가 영화 <내부자들>이라는 사실을 쉽게 알 수 있습니다.

어텐션 메커니즘(Attention Mechanism)은 기계 학습 및 자연어 처리 분야에서 중요한 기술 중 하이며, 특히 seq2seq 작업에서 활용됩니다.

seq2seq는 입력된 시퀀스로부터 다른 도메인 시퀀스를 출력하는 다양한 분야에서 사용되는 모델인데, 번역기에서 대표적으로 사용되는 모델로, seq2seq 모델은 인코더에서 입력 시퀀스를 Context vector라는 하나의 고정된 크기의 벡터 표현으로 압축하고 Decoder는 Context vector를 통해 출력 시퀀스를 만듭니다. 하지만 두 가지 문제점이 발생됩니다.

첫 번째 문제점은 하나의 고정된 크기의 벡터에 모든 정보를 압축해야 하기 때문에 정보 손실이 발생
두 번째 문제점은 RNN계열의 고질적인 문제인 기울기 손실이 발생

Attention 구조

위 두 가지 문제의 해결책이 바로 Attention입니다. Decoder에서 출력 단어를 예측하는 매 시점마다 인코더에서의 전체 입력 문장을 다시 한 번 참고합니다. 여기서 예측해야 할 단어와 연관이 있는 입력 단어 부분을 좀 더 집중해서 참고한다.

Attention Model의 Encoder Layer은 seq2seq의 인코더 층과 동일하지만 Decoder Layer에서 차이가 난다. Decoder Layer에서 RNN계열 층을 지나고 Attention Layer을 지나는데 이때 Attention Layer에서는 시점마다 RNN계열 층의 은닉상태와 인코더에서의 전체 입력 시퀀스 정보(Key, Value)를 다시 한번 참고합니다.

이때 쿼리, 키, 값(Query, Key, Value)이라는 용어가 나옵니다.

쿼리(Query)는 어텐션 메커니즘에서 주목할 대상을 나타내는 정보로, 보통 출력 시퀀스의 현재 위치 또는 상태와 관련이 있고, 어텐션 메커니즘을 통해 중요한 입력 요소를 찾는데 사용됩니다. 기계번역에서 디코더의 현재 출력 단어에 해당하는 쿼리가 사용될 수 있으며, 이 쿼리는 입력 문장의 어떤 부분에 더 집중해야 하는지를 결정합니다.

키(Key)는 입력 시퀀스의 각 요소에 대한 정보를 나타내며 쿼리와 비교되어 유사성을 측정하고, 어떤 입력 요소가 현재 쿼리와 얼마나 관련 있는지를 결정하는 데 사용됩니다. 일반적으로 입력 시퀀스의 각 요소에 대한 벡터 또는 표현으로 나타냅니다.

값(Value)은 입력 시퀀스의 각 요소에 대한 정보를 담고 있습니다. 어텐션 메커니즘에서 가중 평균을 계산할 때 사용되며, 쿼리와 키의 유사성에 따라 값에 가중치가 부여됩니다. 주로 입력 시퀀스의 각 요소에 대한 정보를 포함하는 벡터 또는 표현으로 나타냅니다.

이러한 쿼리, 키, 값을 가지고 어텐션 층에서는 어텐션 함수가 사용되는데 다양한 함수를 통해서 어텐션 스코어가 만들어지고 보통 SoftMax함수를 통해 어텐션값이 생성됩니다. 이외에도 Sigmoid, ReLu함수가 있습니다.

어텐션 메커니즘은 기계 번역, 이미지 캡션 생성, 음성 인식 등 다양한 자연어 처리 및 컴퓨터 비전 작업에서 사용됩니다. 예를 들어, 기계 번역에서는 번역할 때 주어진 단어들 중에서 어떤 단어가 번역에 더 중요한지를 결정하는 데 어텐션 메커니즘이 사용될 수 있습니다.

어텐션 메커니즘은 다양한 형태로 구현될 수 있으며, 그중 가장 널리 사용되는 형태 중 하나는 "소프트 어텐션(Soft Attention)"입니다. 소프트 어텐션은 입력의 모든 요소에 대한 가중치를 계산하고, 이를 가중치의 합으로 가중 평균하여 출력을 생성합니다.

딥러닝에서 어텐션은 모델이 입력의 각 요소를 적절히 고려하여 출력을 생성할 수 있게 도와줍니다. 이를 통해 모델의 성능을 향상시키고, 입력의 일부가 무시되거나 손실되는 것을 방지할 수 있

습니다.

어텐션 메커니즘은 다양한 형태로 구현될 수 있으며, 주로 소프트 어텐션(Soft Attention)과 하드 어텐션(Hard Attention)으로 나뉩니다. 각각의 유형은 다른 방식으로 작동하며 특정 응용 분야나 모델 아키텍처에 따라 선택될 수 있습니다.

Attention 종류

Attention Mechanism에는 다음과 같은 다양한 유형이 있습니다.

1. Dot-Product Attention : 쿼리와 키의 내적을 계산하여 가중치를 구하는 방식

2. Scaled Dot-Product Attention : Dot-Product Attention에서 스케일링을 추가한 방식

3. Multi-Head Attention : 여러 개의 Attention Head를 병렬로 사용하여 다양한 관점에서 입력 데이터를 고려하는 방식

Attention 방식

1. Hard Attention(하드 어텐션) : 입력 요소 중 가장 중요한 부분에만 집중하여 출력을 계산하는 방식(주로 이미지 처리 분야에서 사용)

2. Soft Attention(소프트 어텐션) : 모든 입력 요소에 대해 가중치를 부여하여 출력을 계산하는 방식

Attention 메커니즘

Attention Mechanism은 다음과 같은 과정으로 동작합니다.

1. 쿼리(Query), 키(key), 값(Value) 벡터를 계산합니다

2. 쿼리와 키의 유사도를 계산하여 가중치를 구합니다.

3. 가중치와 값 벡터를 곱하여 최종 출력을 생성합니다.

위 과정을 통해 출력을 생성 시 입력 데이터의 중요도를 동적으로 반영할 수 있게 됩니다.

Attention 활용

1. 기계 번역 : Attention은 기계 번역 모델의 핵심 구성 요소로, 입력 문장의 중요 부분에 집중하여 번역 품질을 향상시킵니다.

2 이미지 처리 : Attention은 이미지 캡셔닝, 이미지 검색 등의 분야에서 활용되며, 이미지 내 중요 영역에 집중할 수 있습니다.

3. 그래프 분석 : 그래프 신경망 모델에서 Attention은 노드 간 관계를 효과적으로 모델링할 수 있습니다.

결론

현재 Attention은 다양한 딥러닝 모델에서 핵심적인 역할을 수행하고 있으며, 지속적인 발전을 통해서 딥 러닝의 성능 향상을 촉진하고 있습니다.

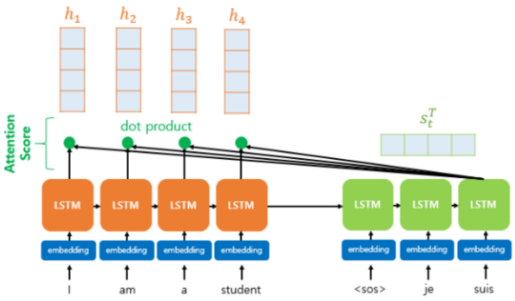
계산 과정

Dot-Product Attention

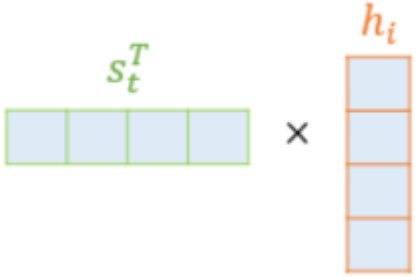
어텐션 구조에서 가장 기본이 되고 수식이 간단.

- Q = Query : t 시점의 디코더 셀에서의 은닉상태
- K = Keys : 모든 시점의 인코더 셀의 은닉상태들
- V = Values : 모든 시점의 인코더셀의 은닉상태들

(1) 어텐션 스코어 구하기



스코어 값을 구하기 위해 st를 전치(transpose)하고 각 은닉상태에 내적을 수행합니다. 즉, 모든 어텐션 스코어 값은 스칼라입니다. 왼쪽 그림은 st와 인코더의 i번째 은닉 상태의 어텐션 스코어의 계산법입니다.

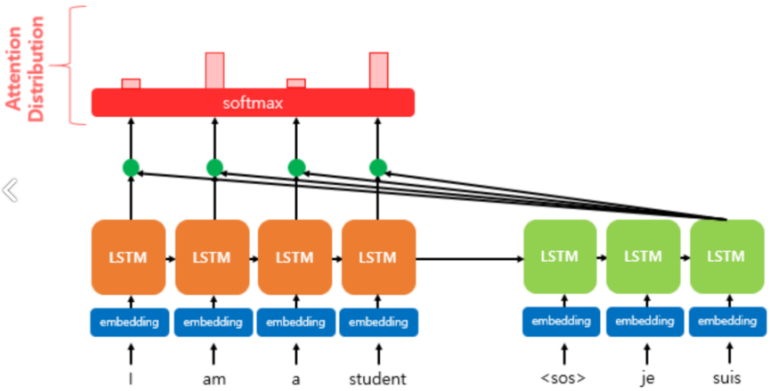


왼쪽의 그림처럼 내적을 통해 어텐션 스코어를 구합니다.

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N]$$

[어텐션 스코어 모음집]

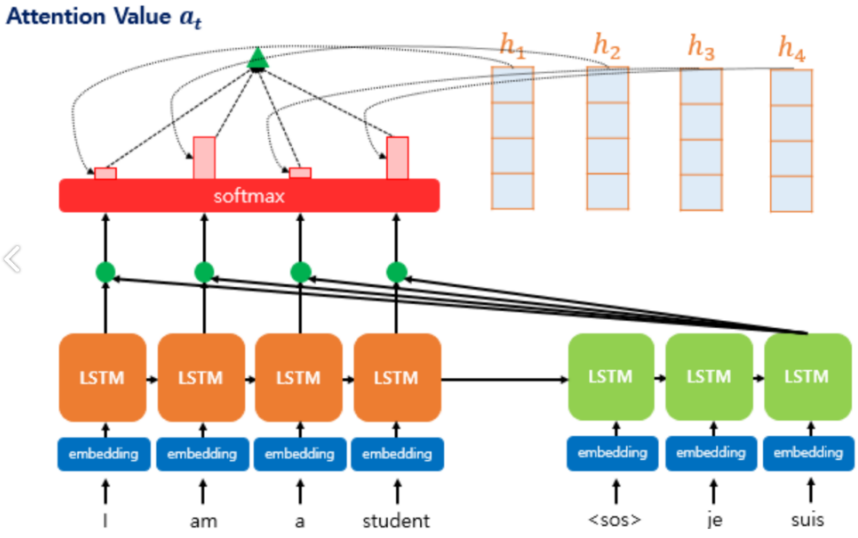
(2) 소프트맥스 함수를 취해 어텐션 분포를 구하기



어텐션 스코어 모음집을 소프트맥스 함수를 적용하여, 모든 값을 합하면 1이 되는 확률 분포를 얻는데, 이것을 어텐션 분포이고 각각의 값은 어텐션 가중치라고 합니다. 'I', 'am', 'a', 'student'입력에 소프트맥스를 적용하여 얻은 어텐션 가중치 출력값이 각각 0.1, 0.4, 0.1, 0.4이면 이들의 합은 1입니다.

$\alpha^t = softmax(e^t)$ [디코더의 시점 t에서 어텐션 가중치의 모음값이 어텐션 분포 수식]

(3) 각 인코더의 어텐션 가중치와 은닉 상태를 가중합하여 어텐션 값을 구한다.



$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i$$

지금까지의 정보들을 하나로 합칩니다. 어텐션의 최종 결과값을 얻기 위해 각 인코더의 은닉 생태와 어텐션 가중치 값을 곱하고, 모두 더합니다. 이를 가중합 이라고 합니다.

[어텐션 값(Attention Value)]

모델 예시

Seq2Seq attention 모델 예시

1. " I love you" – > "난 너를 사랑해"

2. I love you 라는 인코더를 제시해보자 그러면 디코더 값으로는 난 너를 사랑해 라는 답이 나와야 한다.

3. 그러면 Seq2Seq +attention으로 모델을 진행하면 각각 I ,LOVE, YOU 에 할당값 h1, h2, h3가 들어올 것이다.

4. 각 3개의 인코더을 FC(tanh(FC(h1, h2, h3) + FC(h3)))에 넣어 score1, score 2, score 3값을 추출했다.

5. 그것들을 Softmax를 사용하면 확률 값을 지정하여 3개의 값이 각각 0.9, 0.0, 0.1이 나오는데 이것을 Attention Weight라고 하며 이것은 단어에 얼마만큼 우리가 중요도를 둘 것인지 숫자를 정하는 것이다.

6. 이렇게 진행이 되면 첫 번째 문맥 벡터가 제작이 된다.

7.식은 h1*0.9 +h2*0.0 +H3*0.1이다.

8. 또한 cv1을 디코더 첫번째에 넣어준다.

9. 그 후 디코더 부분의 번역된 값이 아직 없으므로 start라는 시그널(tanh(concat (cv,<start>)))을 넣어준다.

10 .그리고 이 시그널에 Attention Weight를 시행하면 Output값인 dh1이 나온다.

11. 두번째 단어는 현재에 있는 이 디코더에 스테이트 값이 중간의 플릿 컨넥티드 네트워크 값('FC(tanh(FC(h1, h2, h3) + FC(h3)))')에 들어간다.

12. 그래서 이 두번째 문맥 벡터는 h1*0.1 + h2*0.0+h3*0.9를 계산한 값이 된다.

13. 이 두번째 디코더 컨텍스트 백터의 RNN 셀에 들어간다.

14. 그래서 OUTPUT인 dh2가 나올 수 있었다.

15. 마지막으로 동일하게 dh2가 CV3에 들어갔고 다시한번 h1, h2, h3가 들어갔다.

16. 이번엔 다르게 진행하여 Attention Weight을 h1-0.03, h2 – 0.95 , h3 – 0.02를 배정한다.

17. 그러면 컨텍스트 벡터가 만들어지고 결과적으로 dh1('사랑해 ') 라는 값이 나왔다.

18. 마지막으로 END값이 나올 때 까지 진행을 한다.

> 결과 : 기존 h1, h2, h3값은 항상 쓰이고 있다. 즉 인코더에서는 state값이 항상 사용된다. 왜냐하면 이값들 중에 어느 값을 포커스 해야 하는가에 우리가 계산을 하기 위해서 이다. 이는 단순히 한개의 컨테스트 벡터 만 사용한 과거의 seq2seq모델 보다 직접적으로 컨텍스트 벡터를 만드는 어텐션 메커니즘이 좀더 방대한 양의 정보를 함축하는데 있어서 더 효율적이다.라는 것을 논문에서 증명되었다.

코드로 구현 (논문에서 나온 것으로 시행)

```
 !sudo apt-get install -y fonts-nanum
!sudo fc-cache -fv
!rm ~/.cache/matplotlib -rf
```

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

plt.rc('font', family='NanumBarunGothic')

df = pd.DataFrame([["i love you", "ich liebe dich"],
                   ["i love myself", "ich liebe mich"],
                   ["i like you", "ich mag dich"],
                   ["he love you", "er liebt dich"]],
                    columns=['src', 'tar'])
Df
```

결과 값

	src	Tar
0	I love you	Ich liebe dich
1	I love myself	Ich liebe mich
2	I like you	Ich mag dich
3	He love you	Er liebt dich

```
SOS_token = 0
EOS_token = 1
```

```
class Tokenizer:
    def __init__(self):
        self.vocab2index = {"": SOS_token, "": EOS_token}
        self.index2vocab = {SOS_token: "", EOS_token: ""}
        self.n_vocab = len(self.vocab2index)

    def add_vocab(self, sentence):
        for word in sentence.split(" "):
            if word not in self.vocab2index:
                self.vocab2index[word] = self.n_vocab
                self.index2vocab[self.n_vocab] = word
                self.n_vocab += 1

    def to_seq(self, sentence):
        l = []
        for s in sentence.split(" "):
            l.append(self.vocab2index[s])
        return l



src_tok = Tokenizer()
for s in df['src'].values:
    for v in s.split(' '):
        src_tok.add_vocab(v)

tar_tok = Tokenizer()
for s in df['tar'].values:
    for v in s.split(' '):
        tar_tok.add_vocab(v)

print(src_tok.vocab2index)
print(tar_tok.vocab2index)
```

결과값

```
{'<SOS>': 0, '<EOS>': 1, 'i': 2, 'love': 3, 'you': 4, 'myself': 5, 'like': 6, 'he': 7}
{'<SOS>': 0, '<EOS>': 1, 'ich': 2, 'liebe': 3, 'dich': 4, 'mich': 5, 'mag': 6, 'er': 7, 'liebt': 8}
```

```
src_data = [src_tok.to_seq(s) for s in df['src'].values]
tar_data = [[SOS_token] + tar_tok.to_seq(s) + [EOS_token] for s in df['tar'].values]
print(src_data)
print(tar_data)
```

결과값

```
[[2, 3, 4], [2, 3, 5], [2, 6, 4], [7, 3, 4]]
[[0, 2, 3, 4, 1], [0, 2, 3, 5, 1], [0, 2, 6, 4, 1], [0, 7, 8, 4, 1]]
```

```
# hparam
hparam = {}
hparam['embed_size'] = 4
```

```
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        self.embed = nn.Embedding(src_tok.n_vocab, hparam['embed_size']) # embed_size = 4
        self.rnn = nn.LSTM(input_size=hparam['embed_size'], hidden_size=hparam['embed_size'])

    def forward(self, x, h, c):
        # (1)
        x = self.embed(x)
        # (embed_size)
        x = x.view((1, 1, -1))
        # (1,1,embed_size)
        x, (h, c) = self.rnn(x, (h, c))
        # (1,1,embed_size) (1,1,embed_size) (1,1,embed_size)
```

```
        return h, c



class Decoder(nn.Module):
    def __init__(self):
        super(Decoder, self).__init__()
        self.embed = nn.Embedding(tar_tok.n_vocab, hparam['embed_size']) # 
embed_size = 4
        self.rnn = nn.LSTM(input_size=hparam['embed_size'], 
hidden_size=hparam['embed_size'])


    def forward(self, x, h, c):
        # (1)
        x = self.embed(x)
        # (embed_size)
        x = x.view((1, 1, -1))
        # (1,1,embed_size)
        x, (h, c) = self.rnn(x, (h, c))
        # (1,1,embed_size) (1,1,embed_size) (1,1,embed_size)
        return h, c



class Attention(nn.Module):
    def __init__(self):
        super(Attention, self).__init__()
        self.wc = nn.Linear(hparam['embed_size'] * 2, hparam['embed_size']) 
# (embed_size * 2, embed_size) = (8, 4)
        self.tanh = nn.Tanh()
        self.wy = nn.Linear(hparam['embed_size'], tar_tok.n_vocab) # 
(embed_size, word_cnt)
    def forward(self, x):
        # (1,1,embed_size * 2)
        x = self.wc(x)
        # (1,1,embed_size)
        x = self.tanh(x)
```

```
        # (1,1,embed_size)
        x = self.wy(x)
        # (1,1,word_cnt)
        x = F.log_softmax(x, dim=2)
        # (1,1,word_cnt)
        return x
```

모델 구동

```
encoder = Encoder()
decoder = Decoder()
attention = Attention()

enc_optimizer = optim.RMSprop(encoder.parameters(), lr=0.01)
dec_optimizer = optim.RMSprop(decoder.parameters(), lr=0.01)
att_optimizer = optim.RMSprop(attention.parameters(), lr=0.01)
criterion = nn.NLLLoss()

loss_hist = []
for epoch in range(500):
    loss_avg = []

    for batch in range(len(src_data)):

        loss = 0

        src_train = torch.LongTensor(src_data[batch])

        h, c = torch.zeros((1, 1, hparam['embed_size'])), torch.zeros((1, 1,
hparam['embed_size']))
        # (1,1,embed_size) (1,1,embed_size)

        enc_out = torch.Tensor([])
```

```
        # (src_len, 1, embed_size)
        for i in range(len(src_train)):
            # x = (1)
            h, c = encoder(src_train[i], h, c)
            # (1,1,embed_size) (1,1,embed_size)
            enc_out = torch.cat((enc_out, h))

        tar_train = torch.LongTensor(tar_data[batch])

        sent = []

        # teaching force rate
        rate = 0.5

        # teaching force
        if rate > np.random.rand():
            for i in range(len(tar_train[:-1])):
                h, c = decoder(tar_train[i], h, c)
                # (1,1,embed_size) (1,1,embed_size)
                score = enc_out.matmul(h.view((1,hparam['embed_size'],1)))
                # t 시점 state 의 encoder h attention score
                # (src_len, 1, 1) = score(hn, stT)
                att_dis = F.softmax(score, dim=0)
                # Attention Distribution
                # (src_len,1,1)
                att_v = torch.sum(enc_out * att_dis,
dim=0).view(1,1,hparam['embed_size'])
                # Attention Value
                # (1,1,embed_size)
                con = torch.cat((att_v, h), dim=2)
                # Concatinate
                out = attention(con)
                # (1,1,word_cnt)
                loss += criterion(out.view((1, -1)), tar_train[i+1].view(1))
```

```
                sent.append(tar_tok.index2vocab[out.argmax().detach().item()])

        # without teaching force
        else:
            dec_in = tar_train[0]
            # skalar
            for i in range(len(tar_train[:-1])):
                h, c = decoder(dec_in, h, c)
                score = enc_out.matmul(h.view((1,hparam['embed_size'],1)))
                att_dis = F.softmax(score, dim=0)
                att_v = torch.sum(enc_out * att_dis,
dim=0).view(1,1,hparam['embed_size'])
                con = torch.cat((att_v, h), dim=2)
                out = attention(con)
                topv, topi = out.squeeze().topk(1) # detach!
                # (1), (1)
                dec_in = topi[0].detach()
                # skalar
                loss += criterion(out.view((1, -1)), tar_train[i+1].view(1))

                sent.append(tar_tok.index2vocab[out.argmax().detach().item()])
                if dec_in == EOS_token:
                    break

        if (epoch + 1) % 50 == 0:
            print(epoch + 1, batch, loss.item())
            print(' '.join([tar_tok.index2vocab[t] for t in
tar_train.detach().numpy()[1: ]]))
            print(' '.join(sent))

        enc_optimizer.zero_grad()
        dec_optimizer.zero_grad()
        att_optimizer.zero_grad()
```

```
        loss = loss / len(df)
        loss.backward()

        enc_optimizer.step()
        dec_optimizer.step()
        att_optimizer.step()

        loss_avg.append(loss.item())

    loss_hist.append(sum(loss_avg))

    if (epoch + 1) % 50 == 0:
        print('avg loss', loss_hist[-1])
        print('==============================')
```