Transformer

19/11/10

목차

• **Seq2Seq**

• **Attention Mechanism**

• **Transformer** - Representaion - Encoder - Self-Attention - Multi-head Attention - Residual Connection - Decoder

Seq2Seq

• 입력된 시퀀스로부터 다른 도메인의 시퀀스를 출력하는 모델

• 챗봇, 기계 번역에 사용됨

기존의 Recurrent 모델의 문제점

• Long-term dependency problem

• 분명 순차적으로 Sequence를 처리하지만, Time Step이 늘어날 수록 성능이 떨어진다.

• Parallelization

• T번째 hidden state를 얻기 위해서 t-1번째 hidden state가 필요하다. -> 순서대로 계산되어야 한다. -> 병렬처리 불가능 -> 계산 느림

So we suggest Transformer

• Long-term dependency problem

• 분명 순차적으로 Sequence를 처리하지만, Time Step이 늘어날 수록 성능이 떨어진다

• Parallelization

• T번째 hidden state를 얻기 위해서 t-1번째 hidden state가 필요하다. -> 순서대로 계산되어야 한다. -> 병렬처리 불가능 -> 계산 느림

• Attention

• Recurrece를 사용하지 않고 input과 outpu의 dependency 포착

• 행렬 계산으로 한 번에 병렬처리 가능

일단 Attention이 뭐죠?

• Seq2Seq에서의 Attention

• 디코더에서 출력 단어를 예측하는 매 time step마다, 인코더의 전체 입력을 다시 참고

• 이 때, 특정 시점에서 예측해야할 단어와 연관이 있는 단어에 좀 더 “집중”하게 됨

h1 h2 h3

h3 I love you -> 난 널 사랑해

I Iove you

FC( tanh( FC(h1,h2,h3) + FC(h3) ) )

h1 h2 h3

h3

I Iove you

h1\*0.9 + h2\*0.1 + h3\*0.1

(context vector)

CV1

0.8 0.1 0.1

Attention weight

s1 s2 s3 Softmax

FC( tanh( FC(h1,h2,h3) + FC(h3) ) )

h1 h2 h3

h3

I Iove you

h1\*0.9 + h2\*0.1 + h3\*0.1

<Start>

(context vector)

CVtanh( concat(cv, <Start>) )

1

0.8 0.1 0.1

Attention weight

s1 s2 s3 Softmax

FC( tanh( FC(h1,h2,h3) + FC(h3) ) )

h1 h2 h3

h3

I Iove you

h1\*0.9 + h2\*0.1 + h3\*0.1

<Start>

(context vector)

CV1

dh1 난 0.8 0.1 0.1

Attention weight

s1 s2 s3 Softmax

FC( tanh( FC(h1,h2,h3) + FC(h3) ) )

h1 h2 h3

h3

I Iove you

FC( tanh( FC(h1,h2,h3) + FC(dh1) ) )

h1 h2 h3

I Iove you

dh1

dh1 h1\*0.1 + h2\*0.1 + h3\*0.8

(context vector)

CV2

난 dh2 널

0.1 0.1 0.8

Attention weight

s1 s2 s3 Softmax

FC( tanh( FC(h1,h2,h3) + FC(dh1) ) )

h1 h2 h3

I Iove you

dh1 h1\*0.1 + h2\*0.7 + h3\*0.2

(context vector)

CV3

난 dh2 널 dh3 사랑해

0.1 0.7 0.2

Attention weight

s1 s2 s3 Softmax

FC( tanh( FC(h1,h2,h3) + FC(dh2) ) )

h1 h2 h3

I Iove you

dh1 h1\*0.3 + h2\*0.3 + h3\*0.4

난 dh2 널 dh3 사랑해 dh4 <End>

(context vector)

CV4

0.3 0.3 0.4

Attention weight

s1 s2 s3 Softmax

FC( tanh( FC(h1,h2,h3) + FC(dh3) ) )

h1 h2 h3

I Iove you

서론이 길었습니다. 이제 Transformer에요.

• RNN기반 Seq2Seq는 문제점이 꽤 많았다.

• 아예 RNN을 빼버리고 Attention으로만 인코더와 디코더를 만들어볼까?

Transformer Architecture

Transformer Architecture

1. Representation

• Transformer는 RNN과 달리 단어의 위치 정보를 가질 수 없다.

• 위치 정보를 알려주기 위하여 Positional Encoding을 하자.

1. Representation

• How? Pos encoding값과 Embedding Vector를 더하면 된다.

• 같은 단어라고 하더라도 문장 내 위치에 따라서 임베딩 벡터값이 달라진다.

• Transformer 논문에서는 위치정보 값을 만들기 위해 두 개의 함수 사용

1. Representation

• 왜 Sine, Cosine 함수를 쓰지?

It, however, gives the advantage of being able to scale to unseen lengths of sequences

(e.g. if our trained model is asked to translate a sentence longer than any of those in our training set).

- The Illustrated Transformer (Jay Alammar)

2. Encoder

• Encoder는 2개의 Sublayer로 구성

1. Multi-head Self-Attention

2. Position-wise Feed Forward Neural Network (걍 Fully-connected 신경망)

2.1 Self-Attention

• 기존 Seq2Seq의 Attention은 다른 두 문장이었다면 이번엔 자기 자신 에 Attention을 한다.

• Why?

• It이 무엇을 가르킬까?

• 기존 Seq2Seq 구조로는 알 수 없는 정보이다.

2.1 Self-Attention

• Q, K, V 벡터

• Attention에는 Query, Key, Value가 있다.

• 주어진 쿼리에 대해서 모든 키와의 유사도를 각각 구한다.

• 이 유사도를 가중치로 하여 키와 맵핑되어있는 각각의 값에 반영한다. 그리고 유사도가 반영된 값을 모두 가중합한다.

• 이해가 안된다면 일단 따라와주세요

2.1 Self-Attention

• Wq, Wk, Wv는 weight 파라미터로, 학습될 예정

• 각 Q,K,V는 독립된 Matrix, shape은 dmodel×(dmodel/num\_heads)

• 하이퍼파라미터 dmodel : 인코더와 디코더 입력과 출력 크기

Num\_layers : 레이어의 층 수

Num\_heads : 어텐션을 병렬로 하기 위한 어텐션 개수 dff : FC-layer 은닉층 크기

2.1 Self-Attention

• https://www.youtube.com/watch?v=mxGCEWOxfe8

2.1 Self-Attention

• Scaled dot product Attention

• 트랜스포머에서는 두 벡터의 내적값을 스케일링하는 값으로 dk에 루트를 씌운 √dk 사용함.

2.1 Self-Attention

• Scaled dot product Attention

• 트랜스포머에서는 두 벡터의 내적값을 스케일링하는 값으로 dk에 루트를 씌운 √dk 사용함.

2.1 Self-Attention

• Scaled dot product Attention

• Attention Value가 결국 Context Vector

2.1 Self-Attention

• Parallelization

2.1 Self-Attention

• Attention Score

• Self-Attention으로 Attention Score행렬을 구한다.

• Attetion Value

• 스코어 행렬로 어텐션 분포를 구하고, 이를 사용하여 모든 단어에 대한 어텐션 값을 구한다.

2.1 Self-Attention

• Attention Score

• Self-Attention으로 Attention Score행렬을 구한다.

• Attetion Value

• 스코어 행렬로 어텐션 분포를 구하고, 이를 사용하여 모든 단어에 대한 어텐션 값을 구한다.

2.1 Self-Attention

• Matrix 크기 정리

Q와 K의 크기 = **d**k, -> shape = (문장길이, dk)

V의 크기 = **d**v, , -> shape = (문장길이, dv)

Wq와 Wk -> (dmodel, dk)

Wv -> (dmodel, dv)

논문에서는dmodel/ num\_heads = dk = dv

결과적으로 Attention(Q,K,V) = a(seq\_len, dv)

입력 데이터의 shape와 동일하다!

2.2 Multi-head Attention

• 왜 임베딩 차원에 num\_heads로 나눴냐!

• Head수 만큼 각각의 어텐션을 따로 해준다.

• 각 head마다 Wq, Wk, Wv 값은 다르다.

2.2 Multi-head Attention

• 왜 임베딩 차원에 num\_heads로 나눴냐!

• Head수 만큼 각각의 어텐션을 따로 해준다.

• 각 head마다 Wq, Wk, Wv 값은 다르다.

-> 시각의 다양성

2.2 Multi-head Attention

• 병렬 어텐션 후 모든 어텐션 헤드를 연결(Concatenate)

• Concat 이후 shape = (seq\_len, dmodel)

2.2 Multi-head Attention

• Concatenated Matrix에 또 다른 가중치 행렬 Wo를 곱해서 최종 결과물이 나온다.

2.3 Residual connection

• Encoder에는 추가적으로 Residual Connection을 사용한다.

• Why? Layer를 거칠수록 gradient가 사라지는 vanishing gradient문제 완화

2.3 Residual connection

• https://www.youtube.com/watch?v=mxGCEWOxfe8

2.3 Residual connection

3. Decoder

• 비슷한 구조이지만, Encoder의 Self-Attention과는 약간 다른 방법이 적용된다.

• Masking -> position i 보다 이후에 있는 position에 attention을 주지 않음

• 1층에는 Masked Multi-Head Attention

• 2층에는 Encoder에서 온 Key, Value와 Decoder의 query가 input

• ‘지금 decoder에서 이런 값이 나왔는데 무엇이 output이 돼야 할까?’

• 3층에는 FFNN

• 모두 Residual Connection 존재

3.1 Masked Self-Attention

• Decoder에서는 i번째 ouput을 다시 i+1번째 input으로 사용하는 auto-regressive 한 특성을 유지하기 위해, masking out된 attention을 적용한다.

• How? Attention(Q,K,V)에서 softmax의 인풋값을 –무한대로 설정

3.2 Decoder flow

• Decoder의 흐름 예시

3.2 Decoder flow

• Decoder의 흐름 예시