TAVE 서기

|  |  |  |  |
| --- | --- | --- | --- |
| **서기 내용** | | | |
| **서기 일자** | 21.11.26 | **서기** | 이아현 |
| **주제** | 순환 신경망 RNN | | |
| **시간** | 20:30 – 22:30 | **장소** | Zoom 미팅 |
| **스터디**  **인원** | 고성호, 권기호, 이아현, 서가을 : 시작    고성호, 권기호, 이아현, 서가을 : 종료 | | |
| **내용** | | | |
| **배운 내용** | **[목차]**  **Chapter 15. RNN과 CNN을 사용해 시퀀스 처리하기**  **15.2 RNN 훈련하기**  **Chapter 16. RNN과 어텐션을 사용한 자연어 처리**  **16.1 Char-RNN을 사용해 셰익스피어 문체 생성하기**  **16.2 감성 분석**  **16.3 신경망 기계 번역을 위한 인코더-디코더 네트워크**  **16.4 어텐션과 트랜스포머**  **16.6 연습문제** | | |
| **15.2 RNN 훈련하기**  **- RNN을 훈련하기 위한 기법은 타임 스텝으로 네트워크를 펼치고 보통의 역전파를 사용하는 것 => BPTT**    **16.1 Char-RNN을 사용해 셰익스피어 문체 생성하기**  **16.1.1 훈련 데이터셋 만들기**    **- 모든 글자를 정수로 인코딩하기**    **- 숫자 -> 문자, 문자 -> 숫자로 바꾸는 법**  **16.1.2 순차 데이터셋을 나누는 방법**  **- 시계열을 훈련 셋, 검증 셋, 테스트 셋으로 나누는 것은 간단한 작업 X**  **- 주어진 문제에 따라 달라짐**  **16.1.3 순차 데이터를 윈도 여러 개로 자르기**    **- window() -> flat\_map() -> shuffle() -> batch() -> map()**  **- 범주형 변수는 원핫 벡터 이용해서 인코딩**  **- 프리페칭 추가**  **16.1.4 Char-RNN 모델 만들고 훈련하기**    **- 이전 글자 100개를 기반으로 다음 글자를 예측**  **- 유닛 128개를 가진 GRU 층 2개**  **- 입력과 은닉 상태에 20% 드롭아웃 이용**  **16.1.5 Char-RNN 모델 사용하기**    **- 새로운 텍스트를 넣기 위해 전처리 함수 생성**  **- model에 “How are yo” 입력 -> ‘u’ 반환**  **16.1.6 가짜 셰익스피어 텍스트 생성하기**    **- 새로운 텍스트 생성?**  **1. 초기 텍스트 입력**  **2. 모델이 가장 가능성 있는 다음 글자 예측**  **3. 예측된 글자를 가지고 늘어난 텍스트를 모델에 전달하여 다음 글자 예측 But, 같은 단어가 반복되는 경우가 많음**  **=> tf.random.categorical() 함수를 사용해 모델이 추정한 확률을 기반으로 다음 글자를 무작위로 선택 O**  **- 조금 더 좋은 성능을 내려면?**  **- GRU 층과 층의 뉴런 수를 늘리고 더 오래 훈련하거나 규제(recurrent\_dropout=0.3) 추가**  **- 현재 모델은 100보다 긴 패턴 학습 X**  **- 윈도를 크게 할 수 있지만 학습 어려워짐**  **- LSTM과 GRU 셀이라도 매우 긴 시퀀스는 다룰 수 X**  **아니면 상태가 있는 RNN 사용**  **16.1.7 상태가 있는 RNN**  **- RNN이 한 훈련 배치를 처리한 후에 마지막 상태를 다음 훈련 배치의 초기 상태로 사용? -> 역전파는 짧은 시퀀스에서 일어나지만 모델이 장기간 패턴을 학습할 수 있음**  **=> 상태가 있는 RNN**      **- 상태가 있는 RNN 만드는 방법**  **1. 순차적이고 겹치지 않는 입력 시퀀스 만들기**  **Why? 배치에 있는 각 입력 시퀀스가 이전 배치의 시퀀스가 끝난 지점에서 시작해야 함**  **2. Dataset을 만들 때 window() 메서드에서 shift=n\_steps 사용**  **3. shuffle() 메서드 호출 X**  **16.2 감성 분석**  **- IMDb 리뷰 데이터셋 이용**    **- 전처리 함수 생성**      **- 어휘 사전 구축 : 좋은 성능을 내기 위해 사전에 있는 모든 단어를 모델이 알아야 할 필요는 X -> 가장 많이 등장하는 단어 10,000개만 남기고 삭제**    **- 최종 훈련 세트 만들기 및 모델 훈련**  **16.2.1 마스킹**  **- 마스킹 작업 : 차원 내 원소의 위치를 표시하는 작업**  **왜 필요할까?**  **Ex) I like Tave.**  **기대되는 입력층 : [~ ,~ ,~ ,~ ,~ ,~]**  **패딩이 포함된 입력층 : ['I', 'like', 'Tave', '0', '0', '0'] => [12 ,51 ,100002 , \ , \ , \]**  **이때 패딩은 의미가 없는 정보가 되므로 앞 3 단어가 의미 있는 단어라는 것을 알려줘야 함**  **- 마스크 텐서([True, True, True, False, False, False])가 생성되어 모든 층에 타임 스텝 차원이 유지되는 동안 자동으로 전파되어야 함**    **16.2.2 사전훈련된 임베딩 재사용하기**      **- 문장 인코더는 문자열을 입력받아 하나의 벡터로 인코딩**  **(문자열 파싱 후 대규모 코퍼스로 사전 훈련된 임베딩 행렬을 사용해 각 단어를 임베딩, 그 후 모든 단어 임베딩의 평균을 계산)**  **- 문장 임베딩 후, 두 개의 Dense층을 추가해 감성 분석 모델을 구성**  **- 모델을 구성한 후, 배치와 프리패치를 진행하고 모델을 훈련**  **16.3 신경망 기계 번역을 위한 인코더-디코더 네트워크**  **- 신경망 기계 번역 모델**     * **각각의 단어 -> 사전형태화 -> 적은 횟수 반복 : UNK, 문장의 시작 : SOS, 문장의 끝 : EOS** * **뒤집는 이유?**       **16.3.1 양방향 RNN**  **- 예시 문제를 통한 양방향 RNN 이해**     * **이전+이후 시점의 모든 데이터를 고려하기 위해 만든 것**   **=> 양방향 RNN**     * **동일한 입력에 대해 두개의 순환층을 실행** * **하나는 왼쪽에서 오른쪽으로 단어를 읽고 다른 하나는 오른쪽에서 왼쪽으로 읽음** * **타임 스텝마다 두 출력을 연결**   **16.3.2 빔(Beam) 검색**  **- K 개의 가능성 있는 문장의 리스트를 유지하고, 디코더 단계마다 해당 문장의 단어를 하나씩 생성하여 가능성 있는 k 개의 문장을 만듦**   1. **각 스텝에서 각각의 후보 시퀀스를 모든 가능한 다음 step으로 확장** 2. **확장된 후보 스텝에 대한 점수를 얻음. 이때 점수는 모든 확률 값을 곱하여 얻음** 3. **가능도가 높은 k개의 시퀀스만 남기고 나머지 후보들은 제거.** 4. **시퀀스가 끝날 때 까지 위 과정 반복** 5. **시퀀스가 끝나는 기준**   **- <eos> 시퀀스가 나온다.**  **- 설정한 최대 길이에 도달했다.**  **- Threshold likelihood 밑으로 가능도가 낮아짐.**    그림 1 K=5 를 사용한 beam search 과정의 시각화  **16.4 어텐션과 트랜스포머**  **16.4.1 Attention VS Seq2seq**  **- 왜 쓰나?**  **① seq2seq : 컨텍스트 벡터라는 하나의 고정된 크기의 벡터 표현으로 압축하고, 디코더는 이 컨텍스트 벡터를 통해서 출력 시퀀스**  **→ 하나의 고정된 크기의 벡터에 모든 정보를 압축하려고 하니까 정보 손실이 발생**  **→ RNN의 고질적인 문제인 기울기 소실(Vanishing Gradient) 문제가 존재**  **② 디코더에서 출력 단어를 예측하는 매 time step마다 인코더에서의 전체 입력 문장을 다시 한 번 참고, 예측해야 할 단어와 연관이 있는 입력 단어 부분을 좀 더 집중**  **→ 디코더 셀의 은닉 상태(Query)와 인코더 셀의 은닉 상태(Key)의 ‘유사도’를 구하고, 그 유사도를 각각의 ‘값(Value)’에 반영시켜 ‘어텐션 값(Attention Value)’을 구하는 것이 목표**  **- Dot Product (Luong)**  **① Attention Score (유사도) 구하기 : 하나의 디코더 은닉층에 대해 각 인코더의 은닉층 점곱**    **② Attention Distribution (유사도 분포) 구하기 : 소프트맥스 활용**    **③ Attention Value (어텐션 값) 구하기 : 각 어텐션 가중치와 인코더 은닉 상태를 가중합**    **④ Attention Value(어텐션 값) 활용 - 어텐션 값과 디코더 t시점의 은닉상태 연결 (Concatenate)**  **- 출력층 연산의 입력이 되는 s~t를 계산하고**  **출력층 입력으로 사용하기**    **- Bahdanau (연결 어텐션)**  **- Luong과 Bahdanau가 다른 점**  **① 어텐션 값 구할 때 ‘t 시점’이 아닌 ‘t-1 시점’을 사용하는 것이 다름**  **② Luong은 St로 S~t를 구했다면 Bahdanau는 St-1로 St를 구함**     1. **Attention Score (유사도) 구하기**      1. **Attention Distribution (어텐션 분포) 구하기 : Softmax 함수 사용** 2. **Attention Value (어텐션 값) 구하기 : 각 어텐션 가중치와 인코더 은닉 상태를 가중합** 3. **St 구하기 : Context Vector를 연결한 이전 디코더 Output과 St-1을 통해 St를 구함**     **16.4.2 Visual Attention**  **- Visual Attention을 사용하는 이유! 설명 가능성!**  **- 가중치를 직관적으로 확인해서 어디에 초점을 잡았는지를 쉽게 확인함**  **- 올바르지 못하게 예측하지 못한 부분에 대해 쉽게 교정할 수 있음**  **- Visual Attention 사용 예시 : 이미지 캡션 생성하기**  **① 이미지를 처리하여 일련의 특성 맵 출력**  **② 디코더 RNN이 한번에 한 단어씩 캡션 생성**  **16.4.3 Transformer : Attention is All you need**  **https://blog.promedius.ai/content/images/2021/03/t03-1.png**  **- 순환층이나 합성곱 층을 하나도 사용하지 않았음**  **- Attention Mechanism만 사용 (Attention is All you need)**  **- NMT(기계 번역)에서 Good! 훈련 속도가 빠르고 병렬화가 쉬움**  **- seq2seq 구조는 인코더와 디코더에서 각각 하나의 RNN이 하나의 시점(time-step)을 가지는 구조vs트랜스포머는 인코더와 디코더라는**  **단위가 N개로 구성되는 구조**      **- 위치 인코딩 : RNN 구조가 더이상 없기 때문에 순서 정보,**  **position 정보를 이해하기 위해,**  **임베딩 벡터에 위치정보를 더하여 입력으로 사용**    **■ Scaled Dot-Product Attention**  **: Q, K, V를 통해 입력 문장 내에서 유사도를 구함**  **(동사는 동사끼리, 주어는 주어끼리 등)**  **■ Multi Head Attention**  **: 그 Scaled Dot-Product를 여러 번 해서 단어의 여러 특징**  **을 조금 더 잘 추출할 수 있게 함**  **(ex. 동사인지 주어인지, 과거형인지 미래형인지 등)**  **■ Masked Multi Head Attention**  **: 빈 자리로 Softmax 함수가 혼동되는 것 방지**  **<pad>에 매우 작은 음수 투입**    **16.6 연습문제**  **1. 상태가 없는 RNN 대비 상태가 있는 RNN의 장단점은 무엇인가요?**  **- 장점**   * **모델이 장기간 패턴을 학습할 수 있음**   **- 단점**   * **데이터셋 처리의 복잡성** * **순차적이고 겹치지 않는 시퀀스를 만들어야 함**   **2. 자동 번역에 seq2seq RNN 대신 인코더-디코더 RNN을 사용하나요?**   * **문장을 번역할 때 단어를 하나씩 읽고 번역하는 것은 잘못된 번역의 위험이 있음** * **‘Je vous en prie’ = ‘You are welcome’을 번역할 경우를 가정. 한 단어씩 따로 번역해보면 ‘I you in pray’가 됨** * **sequence to sequence의 경우 전체 문장에 대한 이해 없이 첫 단어를 보고 바로 번역을 시작하기 때문에 잘못된 번역으로 이어질 가능성이 큼** * **인코더-디코더의 경우 인코더를 통해 문장에 쓰이는 단어를 모두 입력 받고 이에 대한 정보를 디코더에 넘겨주기 때문에 번역에 있어서 훨씬 유리함**   **3. 가변 길이 입력 시퀀스를 어떻게 다룰 수 있나요? 가변 길이 출력 시퀀스는 어떤가요?**  **- 배치에 있는 모든 시퀀스 길이가 동일하도록 짧은 시퀀스에 패딩을 추가하고 RNN이 패딩 토큰을 무시하도록 마스킹을 하여 처리할 수 있음**  **- 성능을 더 높이려면 크기가 비슷한 시퀀스를 모아 배치를 만드는 것이 좋음**  **- 래그드 텐서는 가변 길이 시퀀스를 담을 수 있고 tf.keras에서 지원할 계획 있음**  **- 시퀀스 마지막 다음에 오는 토큰을 무시하도록 손실 함수를 설정해야 함**  **- 시퀀스의 끝에 EOS 토큰을 출력하도록 모델을 훈련하는 것이 한 방법**  **4. 빔 검색이 무엇인가요? 왜 사용해야 하나요? 이를 구현하기 위해 어떤 도구를 사용할 수 있나요?**  **- 빔 검색이란?**   * **빔 검색은 훈련된 인코더-디코더 모델의 성능을 향상하기 위한 방법** * **기존의 인코더-디코더가 가장 확률이 높은 단어 1개만을 고려해 출력하였다면, 빔 검색을 사용할 땐 beam width 만큼의 단어의 출력 시퀀스를 만들어가는 방법을 사용**   **- 왜 사용해야 될까?**   * **더 많은 경우를 고려하기 때문에 더 정확한 번역이 가능**   **- 구현하기 위한 도구**   * **구현은 tensorflow addons를 활용하여 가능**   **5. 어텐션 메커니즘이 무엇인가요? 어떤 장점이 있나요?**     * **어텐션 메커니즘은 디코더가 입력 시퀀스에 직접 접근하기 위해 개발된 메커니즘** * **정렬 모델은 현재 디코더의 state와 인코더의 output들을 통해 정렬 점수를 출력 이를 softmax를 통해 0~1 사이의 확률값으로 변환한 뒤 가중치를 도출해냄** * **가중치와 인코더의 output을 곱해서 디코더에 주입함** * **디코더는 다음 디코더 state와 output을 만들어냄**   **- 장점**   * **긴 입력 시퀀스를 처리하기 용이함** * **정렬 점수가 인코더의 output의 어느 부분에 집중하고 있는지를 나타내기 때문에 모델의 오류 발생시 디버깅이 용이함**   **6. 트랜스포머 구조에서 가장 중요한 층이 무엇인가요? 이 층의 목적이 무엇인가요?**  **- Vector to Sequence**   * **각 time step에서 하나의 입력 벡터를 반복해서 주입하고 하나의 sequence를 출력할 수 있음** * **이미지를 입력하여 이미지 캡션을 출력하는 경우 사용할 수 있음**   **- Encoder-Decoder**   * **Sequence to Vector 뒤에 Vector to Sequence를 연결한 Network임** * **한 언어의 문장을 네트워크에 주입하면 이를 하나의 Vector로 변환하고, Decoder가 이 Vector를 다른 언어의 문장으로 Decoding함** * **Sequence to Sequence 보다 번역에 있어서 성능이 더 좋은데, 마지막 단어가 번역의 첫 번째 단어에 영향을 줄 수 있기 때문임**   **7. 샘플링 소프트맥스를 사용해야 할 때는 언제인가요?**     * **16.3 참고** * **인코더 디코더 네트워크에서 배웠듯이 디코더 층의 출력은 softmax층을 통과해 확률 값으로 바뀌게 됨. 확률 값 중 가장 큰 값을 prediction으로 선택하게 되는데, 만약 어휘 사전이 50,000개와 같이 방대하다면 50,000차원의 벡터를 출력해야 됨** * **이렇게 방대한 양의 연산을 수행하는 대신, target이 되는 단어(왼쪽의 예 에서는 Je, bois, du, lait)와 target 외의 단어 중 random하게 sample된 단어들만 고려함**   **8. 레버 문법을 따르는지 아닌지 구분하는 RNN 훈련**  **Reber grammar**   * **Reber Grammar는 왼쪽 상단의 그림의 규칙을 따르는 문법을 뜻함 B에서 출발해서 E까지 도달하며, 다시 뒤돌아가는 경로도 있고 자기 자신에서 출발해서 1번의 step만으로 자기 자신으로 되돌아오는 경로도 있으므로 무한한 경우의 단어가 만들어짐** * **주어진 단어가 Reber grammar를 따르는지 확인하기 위해서는 어떤 경로를 거쳐왔는지를 확인해야 함. 표의 Non-Reber 중 BTSSPXSE를 보면 P가 주어졌을 때 모델은 거쳐온 경로가 어떻게 되는지를 파악해야 함. P는 V(T나 X를 거쳐온)나 T(P, X, T를 거쳐온) 뒤에 와야만 함. 하지만, BTSSPXSE에서 P는 S뒤에 왔음. 때문에 이 단어는 Reber grammar를 따르고 있지 않다고 판단할 수 있음. 이 예에서 중요한 점은 모델이 이전 경로에 대한 정보를 계속 기억하고 있어야 한다는 점임. 우리가 배운 RNN으로 이러한 모델을 구성할 수 있음.**   Embedded Reber Grammar   * **하지만, 책의 연습문제에서는 보다 어려운 임베딩 된 레버 문법인지 구분하는 RNN모델을 요구하고 있음** * **바로 전에 살펴본 문법보다 더 어려운데, 이유는 다음과 같음. 모델은 단어를 검증할 때 위에 있는 그래프를 따랐는지 밑에 있는 그래프를 따랐는지까지 확인해야 함. 이를 위해서는 BT, 단어 ,TE 경로를 따랐는지 BP, 단어, PE 경로를 따랐는지를 확인해야 하는데 단어의 맨 앞 경로와 맨 뒤의 경로에 대해 확인하는 작업이 추가되므로 더 기억력이 좋은 모델을 요구함.**         **- RNN에 입력하기 위해 단어들을 임베딩으로 바꾸기**        **9. 날짜 문자열 포맷을 변환하는 인코더-디코더 모델을 훈련하세요(예를 들어, 'Aprill 22, 2019'에서 '2019-04-22'로 바꿉니다).**  **- 데이터 셋 준비**      **- version 1. basic Seq2seq**      **- version 2. 교사 강요 사용**        **version 3. TF Addons seq2seq API 사용**    **version4. TF-Addons seq2seq ScheduledEmbeddingSampler사용**        **version5. TFA seq2seq, Keras subclassing API, attention machanism 사용해 구현하기**    **10. 텐서플로의 Neural Machine Translation with Attention tutorial(어텐션을 사용한 신경망 기계 번역) 튜토리얼을 살펴보세요.**  [**https://www.tensorflow.org/tutorials/text/nmt\_with\_attention?hl=ko**](https://www.tensorflow.org/tutorials/text/nmt_with_attention?hl=ko)  **11. 최신 언어 모델 중 하나(예를 들어 BERT)로 셰익스피어가 쓴 것 같은 텍스트를 생성해보세요. => transformer 이용** | | |
| **과제할당** | 17.1 이아현  17.2 서가을  17.3 권기호  17.4 하정현  17.5 채원석  17.6 이문기  17.7 박제윤  17.8 허주희  17.9 안세윤  17.10 고성호 | | |
| **특이사항** | 없음 | | |
| **비고** | 없음 | | |