Lecture 13. Contextual Word Embeddings

Ⅰ. Reflections on word representations

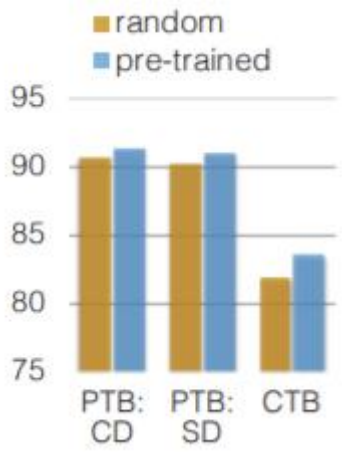
단어를 표현한다는 것은 word embedding을 통해 단어를 벡터로 표현함으로써 컴퓨터가 이해할 수 있도록 자연어를 적절히 변환해주는 방법을 말함. 단어를 벡터로 표현하는 방법으로는 Word2vec, GloVe, fastText 등이 있다.

1. 2012년 이전 word representation

	POS WSJ (acc.)	NER CoNLL (F1)
State-of-the-art*	97.24	89.31
Supervised NN	**96.37**	**81.47**
Unsupervised pre-training followed by supervised NN**	**97.20**	**88.87**
+ hand-crafted features***	97.29	89.59

State-of-the-art는 딥러닝 모델이 아닌 rule-based 혹은 일반적인 머신러닝 방법론이다. 세번째는 word2vec과 같은 unsupervised NN을 supervised NN 방법론과 결합한 방법으로 feature engineering과 같은 전처리 작업을 조금 거치게 되면 성능이 상승함을 볼 수 있다.

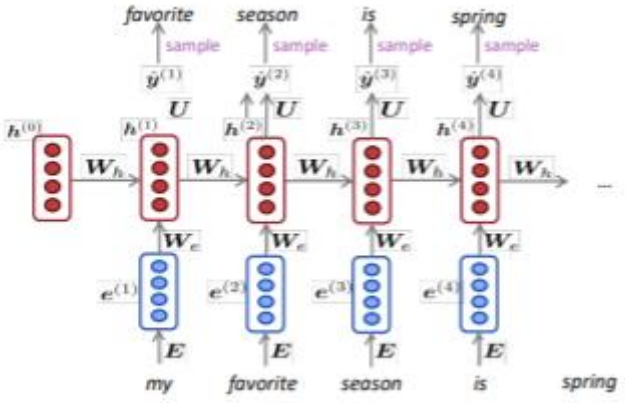
2. 2014년 이후 word representation



2014년 이후 단어를 표현하는 방법으로는 pre-trained된 word vector를 사용하는 방법이 일반적인 random initialize된 word vector보다 성능이 좋다는 것을 보여준다. 그 이유는 가지고 있는 데이터보다 방대한 양의 데이터로 학습된 결과를 활용하기 때문이다. Pre-trained된 word vector를 사용하는 것은 'unk' 토큰을 처리할 때보다 효과적이다. unk 토큰은 보통 data set에서 잘 나타나지 않는 단어를 일컫거나(5회 이하) test data set에만 존재

하는 단어를 unk 토큰으로 분류하기 된다. 하지만 단순히 횟수로 단어를 unk로 분류하게 되면 unk 토큰이 가지는 의미가 모호해지며 중요한 의미를 가지는 단어를 놓칠 수 있게 된다. unk 토큰을 처리하는 방법으로는 char-level model로 word vector를 생성하거나 pre-trained word vector를 생성하거나 random vector를 부여하여 vocab에 추가하는 방법이 있는데 그 중 pre-trained word vector를 사용하는 것이 가장 효과적이라고 한다.

3. Word embedding 시 고려해야할 점



1개의 단어를 1개의 vector로 표현하는 것은 동음이의어 문제를 발생시킬 수 있다. Vector가 문맥에 따라 달라지는 단어의 type을 고려하지 못하는 문제와 단어의 언어적/사회적 의미에 따라 달라지는 측면을 고려하지 못할 수 있다. 이러한 문맥을 고려하여 word vector를 생성할 수 있는 방법으로는 neural language model이 있다. LSTM구조가 적용된 언어 모델로 문장의 sequence를 고려하여 다음 단어를 예측하는 모델이다. Sequence 학습을 통해 문맥을 어느정도 유지할 수 있다. 이러한 특징을 활용한 모델의 대표적인 사례가 ELMo이다.

Ⅱ. Pre-ELMo and ELMO

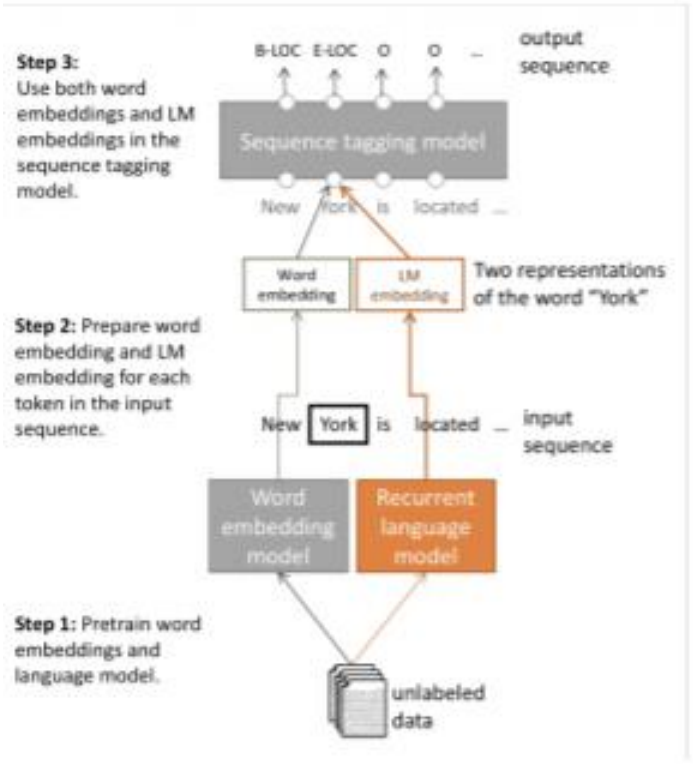
1. Pre-ELMo (TagLM)

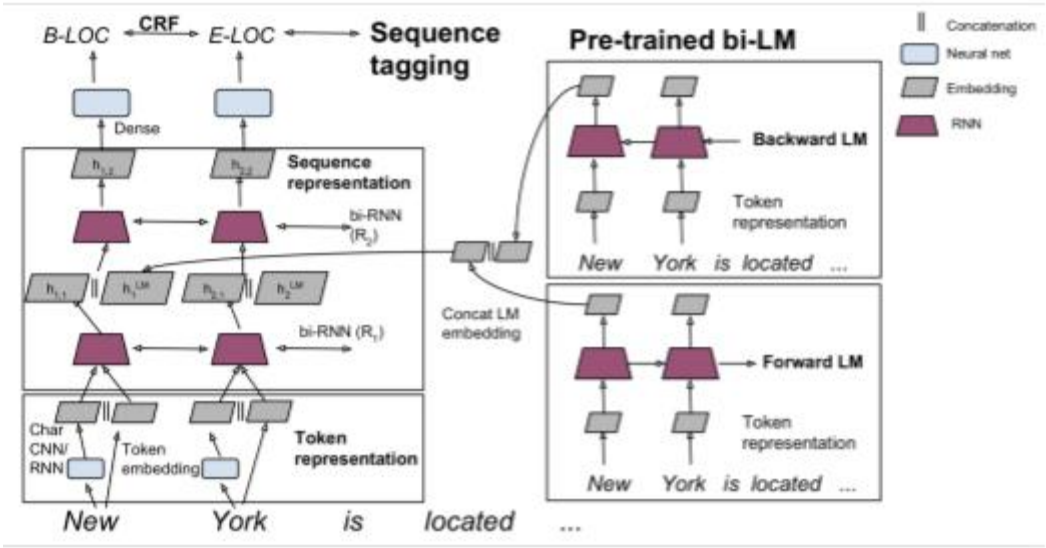
ELMo를 발표한 저자가 ELMo를 발표 이전에 제시했던 TagLM을 Pre-ELMo라고 한다. Pre-ELMo는 ELMo의 기본적인 구조와 굉장히 유사하며 RNN을 통해 문맥이 유지되는 word vector를 찾기 위한 가정이 적용되어 있다. 또한, Neural language Model로 word vector를 pre-trained한 뒤에 추가로 학습이 진행되는 semi-supervised 접근법이 활용되었다.

[Step1] Word2vec과 같은 conventional word embedding model로 word vector를 학습하고 동시에 bi-LSTM과 같은 NLM model로 word vector를 학습한다.

[Step2] 2가지 방법으로 학습된 word embedding을 input으로 사용한다.

[Step3] 문맥과 상관없는 word embedding과 보유하고 있는 데이터의 문맥을 고려한 word embedding을 모두 사용할 수 있다.





$$\mathbf{h}_{k,1} = [\overrightarrow{\mathbf{h}}_{k,1}; \overleftarrow{\mathbf{h}}_{k,1}; \mathbf{h}_k^{LM}]$$

Pre-traine model로 학습한 word vector와 char-CNN model로 학습한 word vector를 concat하는 과정이 수식으로 나타나있다.

2. ELMo

Pre-ELMo의 저자가 해당 방법론을 보다 일반화하여 새롭게 지시한 모델이다. 모든 문장 전체를 사용해서 문맥이 반영된 word vector를 학습하는 모델이다. 보통 window를 지정함으로써 주변 단어를 통해서만 문맥을 고려하게 되는 기존 모델과 차이가 있다. char-CNN과 같은 모델로 단어의 특징을 고려한 word vector를 구하고 해당 vector를 LSTM의 input으로 사용해서 최종 word embedding 결과를 도출한다. Present를 문장에 따라 '선물'과 '현재'라는 다른 word vector로 표현할 수 있게 된다.

$$
\begin{aligned}
R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\
&= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\},
\end{aligned}
$$

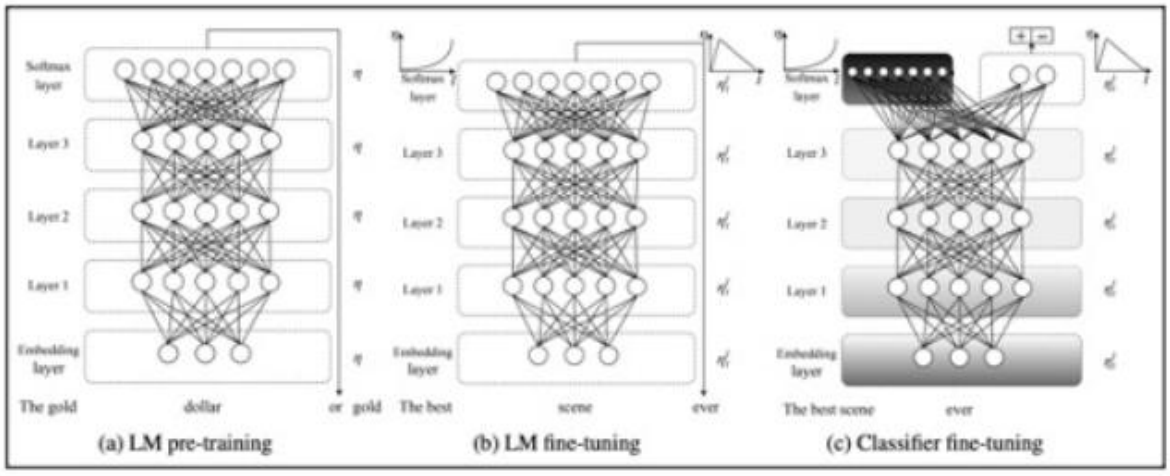
$$
\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}
$$

ELMo의 전체적인 구조는 순방향 language model과 역방향 language model이 모두 적용된 bidirectional language model이다. 순방향 모델은 문장 sequence에 따라 다음 단어를 예측하도록 학습되며 역방향 모델은 뒤에 있는 단어를 통해 앞 단어를 예측하도록 학습된다. ELMo는 모든 layer의 출력값을 활용해서 최종 word vector를 임베딩하는데 최종 layer의 값들로만 word vector로 사용했던 이전 모델과 차이가 있다. 첫번째 layer는 문맥으로부터 영향을 받지 않도록 설계하였지만 이후 layer인 LSTM layer에서는 문맥을 반영하도록 하였다. 첫번째 LSTM layer에서는 char-CNN의 결과에 residual connection을 적용하여 char-CNN으로 반영된 단어의 특징을 잃지않고 유지할 수 있고 학습 시 역전파를 통한 gradient vanishing 문제를 완화시킬 수 있었다. concat된 layer 별 출력에 정규화된 가중치를 곱해주며 학습을 최적화한다. 최종적으로 모든 layer의 벡터를 더해 하나의 임베딩 벡터라는 word vector를 만들게 된다. 이는 syntax 특징을 고려한 하위 layer의 정보와 semantics 특징을 고려한 상위 layer의 정보를 모두 활용함으로써 효과적인 word vector를 얻을 수 있다.

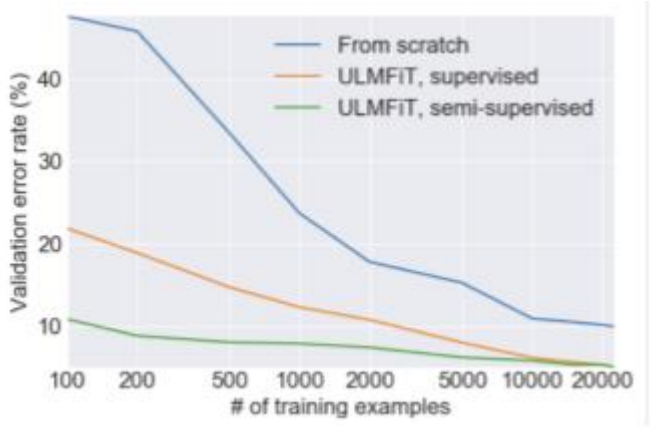
ELMo는 NER 뿐만 아니라 대부분의 task에서 SOTA의 성능을 보여주었다.

Ⅲ. ULMfit and onward

ULMfit는 NLP에서 본격적으로 transfer learning이 도입된 시점이다. Transfer learning이란 대량의 다른 데이터로 일반적인 학습을 진행한 뒤에 목적에 맞게 다시 튜닝을 통해 성능을 개선해 나가는 것을 의미한다. UMLfit은 text classification이라는 목적을 가지게 된다.



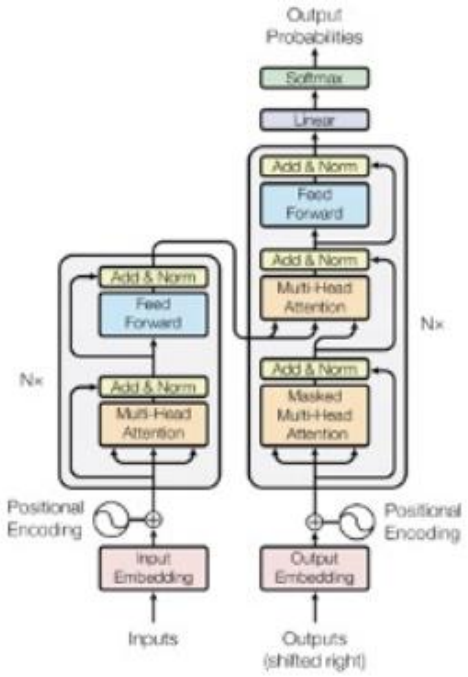
Transfer learning 부분에서는 language model이 대량의 corpus를 기반으로 학습하게 된다. Pre-trained된 언어 모델을 토대로 주어진 task에 맞게 언어 모델을 튜닝하면서 업데이트를 진행한다. 튜닝 시 사용하는 기법으로는 언어모델을 튜닝할 때 깊은 layer에 대해서는 상위 layer에 비해 더 작은 learning rate를 부여하는 discriminative fine-tuning과 튜닝 횟수에 따라서 초반에는 작은 learning rate를 적용하다가 점차 learning rate를 증가시키고 약 200번의 학습 후, 다시 learning rate를 점진적으로 감소시키는 slanted triangular learning rates가 있다. 최종적으로 classifier가 출력되는 부분으로 word vector가 결국 분류기로 출력된다.



Train data set이 굉장히 부족해도 pre-trained된 ULMfit 모델을 사용하게 되면 error rate가 굉장히 낮다. UMLfit 이후로는 모델의 parameter를 늘리고 pre-trained된 데이터 양을 늘려 기하급수적으로 리소스가 필요한 모델들이 등장하게 되는데 GPT, GERT, GTP2가 있다. 해당 모델들은 LSTM 모델이 아닌 모두 transformer 기반의 모델이다.

IV. Transformer architectures

Transformer 기반 모델이 등장하게 된 계기는 LSTM 모델이 가지는 RNN의 태생적인 한계가 존재했기 때문이다. RNN은 sequential한 특징을 고려하기 때문에 병렬적 계산이 불가능하여 매우 느리다. 또한 처음과 끝이 정해져있기 때문에 long-term dependency 문제가 항상 존재했는데 LSTM, GRU로 이러한 문제를 어느정도 해결할 수 있었지만 완전히는 불가능하였다. 모든 state를 참조할 수 있기 때문에 sequential한 특징에서 자유로운 attention 메커니즘만을 활용한다면 RNN 계열 모델의 한계를 해결할 수 있다고 생각해서 transformer 기반 모델이 등장하게 되었다.



1. Transformer

   Transformer는 기계 번역을 위해 존재하는 input과 output이 encoder와 decoder로 구분되어 진행되는 seq2seq 모델이다. 이 때, decoder의 어떤 step과 encoder의 전체 step 중 가장 연관성있는 한 step을 찾아가면서 보다 효과적인 seq2eq 과정을 거치게 된다. Attention은 decoder에서 output으로 나오는 매 step마다 encoder의 전체 step을 참조하기 때문에 모든 state를 참조할 수 있다고 한다. Decoder가 encoder를 참고하는 과정에서 decoder의 단어와 step에 따라 attention value가 달라지게 된다.

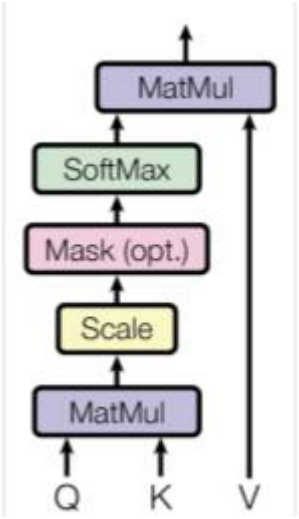
2. Attention value

   attention 구조에는 query, key, value가 존재하며 이 3가지 특성을 가지고 attention value를 도출한다. Query에 대한 모든 key와의 유사도를 각각 구한다. 구해낸 유사도를 key와 매핑되어있는 각각의 value에 반영해준다. 그리고 유사도가 반영된 value를 모두 더해서 반환한다. 이 값이 attention value이다.

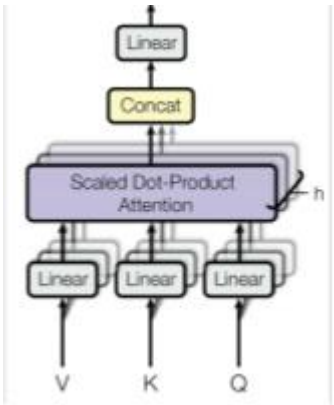
3. Self-attention

Self-attention이란 자기 자신의 query로 attention value를 구하는 것으로 특정 단어가 자신의 문장에서 다른 단어와의 attention value를 구함으로써 dependency와 같은 value의 높고 낮음을 구할 수 있게 되었다. Self-attention이 일어나는 부분을 'Scaled Dot Product Attention'이라고 하는데 마지막 softmax를 통해 어떠한 step과의 값을 뽑아낼 수 있다.

$$A(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$



Transformer의 전체 구조에서 multi-head attention을 확인할 수 있다. 이 부분이 self-attention이 발생되는 지점이다. Multi-head의 의미는 다양한 관점에서 바라봤을 때 보다 풍부한 정보를 얻을 수 있다는 것에 착안하여 attention을 다수로 진행함으로써 보다 많은 정보를 가져와 proejection하여 다음 layer에 전달하는 과정을 말한다.



$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Self-attention을 거친 두 문장은 multi-head attention 과정을 거치면서 보다 정확한 번역을 가능하게 해준다.

Ⅴ. BERT

BERT는 transforemr의 encoder 부분을 차용하여 bi-directional language model을 구축한 모델이다. Bi-direction의 순방향과 역방향은 각각 독립적으로 적용되기 때문에 순방향 혹은 역방향이라는 sequence를 가져 다음 단어 혹은 전 단어를 예측할 때 그 때까지 존재하는 단어만을 기반으로 예측에 활용할 수 있다. 즉, 단어를 예측할 때 전체 단어 모두를 활용할 수 없다. 양방향 예측이 한 번에 가능하다면 가운데 들어갈 단어를 예측할 때 많은 정보를 기반으로 할 수 있다. BERT는 이러한 문제를 Masked Language Model을 적용함으로써 해결하였다. 문장에 존재하는 단어 중 일부를 masked 처리한 뒤에 masked 처리된 단어를 나머지 모든 단어로 예측하는 과정이다. 이렇게 되면 masked된 단어를 제외한 모든 단어를 활용할 수 있게 되면서 순방향과 역방향의 모든 문맥을 한 번에 고려할 수 있게 된다. BERT는 2가지 목적 함수를 갖는데 첫번째는 masked된 단어를 예측하는 mask prediction이고 두번째는 문장과 문장 간의 연속성 여부에 대한 정보를 학습하는 next sentence prediction이다.