

ELMo

20182464
김승기



AI &
HEALTHCARE LAB

What is ELMo?

- ELMo
 - **E**MBEDDINGS from **L**ANGUAGE **M**ODELS
 - 언어 모델로부터 만들어진 임베딩



What makes ELMo special?

- Pre-trained word representations
 - Neural language understanding 모델의 핵심 요소
- High quality representation
 - 단어의 복잡한 특성
 - 구문 분석(syntax)
 - 의미 분석(semantics)
 - 문맥에 맞는 표현
 - 다의어(polysemy)
 - 눈 → eye? 
 - → snow? 

GloVe vs ELMo

Example

Source	Nearest Neighbors
GloVe play	playing, game, games, played, players, plays, player, Play, football, multiplayer
Chico Ruiz made a spectacular <u>play</u> on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent <u>play</u> .
biLM Olivia De Havilland signed to do a Broadway <u>play</u> for Garson {...}	{...} they were actors who had been handed fat roles in a successful <u>play</u> , and had talent enough to fill the roles competently , with nice understatement .

GloVe mostly learns *sport-related context*

Table 4: Nearest neighbors to “play” using GloVe and the context embedding from a biLM.

- **GloVe** : “play”라는 단어와 비슷한 단어를 뽑았을 때, 스포츠와 관련된 단어들이 있는 것을 알 수 있다.
- **ELMo** : “play”가 문장에서 어떤 의미로 사용 되었는지를 구분한다.
 - 첫 번째 “play” : GloVe가 판단한 것과 동일한 “(스포츠에서의) 경기”를 의미
 - 두 번째 “play” : “(연극에서의)play”를 의미
- 따라서, ELMo는 이를 문맥에 맞게 잘 판단하고 있는 것을 볼 수 있다.

ELMo can distinguish the word sense based on the context

bi-LSTM (bidirectional LSTM)

- ELMo는 전체 입력 문장을 가지고 단어 토큰의 임베딩 벡터를 만든다.
- 이런 점에서 단어나 *서브워드를 입력 받았던 Word2Vec, GloVe, Fasttext 등의 이전 모델과 차이점을 갖는다.

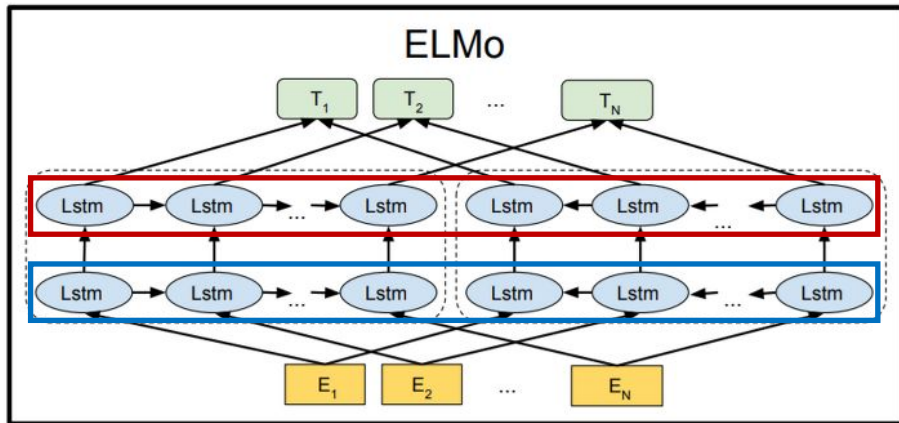
* 서브워드 분리(Subword segmenation)

하나의 단어는 더 작은 단위의 의미 있는 여러 서브워드들의 조합으로 구성된 경우가 많기 때문에,

하나의 단어를 여러 서브워드로 분리해서 단어를 인코딩 및 임베딩하겠다는 의도를 가진 전처리 작업

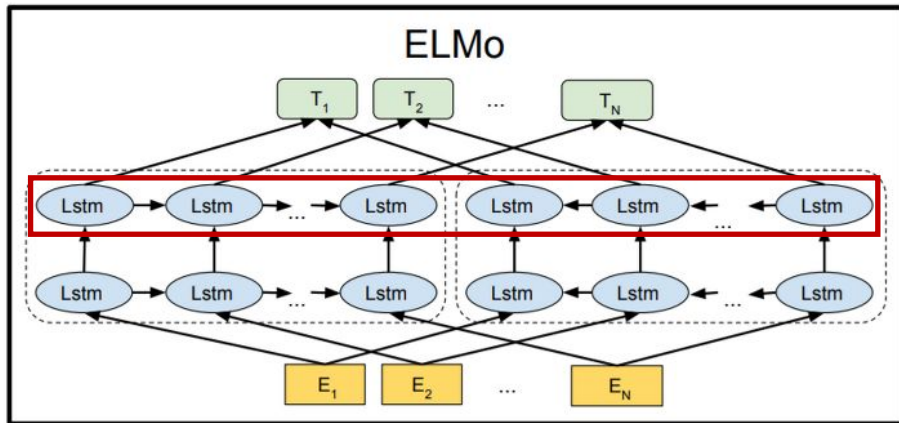
E.g. birthplace = birth + place

bi-LSTM (bidirectional LSTM)



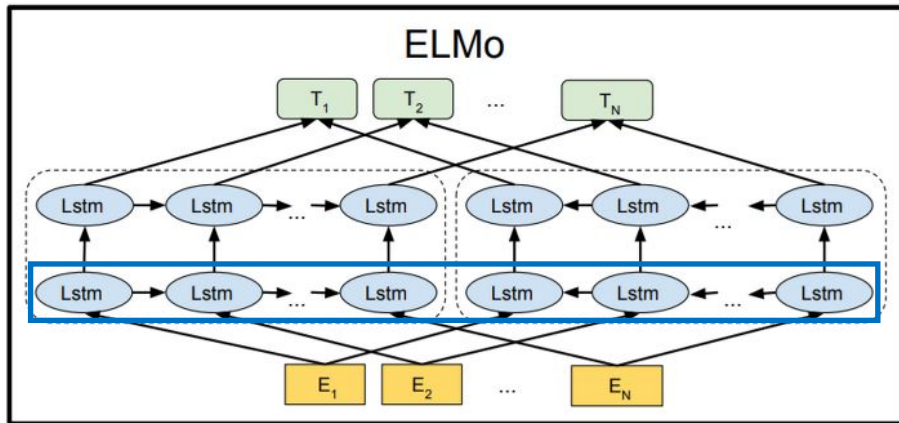
- 커다란 말뭉치를 2개 층으로 이루어진 bi-LSTM(bidirectional LSTM)이 만들어내는 임베딩 벡터를 사용한다.
- 이 벡터는 bi-LSTM의 모든 내부 층에 대한 은닉 벡터에 가중치를 부여한 뒤에 선형 결합하여 사용한다.
- 이렇게 선형 결합을 통해 나타나는 벡터는 단어가 가진 많은 특성에 대한 정보를 담고 있게 된다.

bi-LSTM (bidirectional LSTM)



- 각 은닉 벡터 중에서 위쪽에 위치한 LSTM은 단어의 **문맥적인 (Context-dependent)** 의미를 포착할 수 있다.
- 따라서, 의미적인 표현이 중요해지는 NLU(Natural Language Understanding)이나 QA(Question&Answering) 등의 태스크에는 위쪽 bi-LSTM층이 만들어 내는 은닉 벡터에 더 많은 가중치를 부여하게 된다.

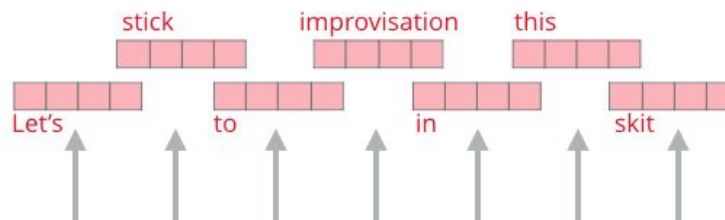
bi-LSTM (bidirectional LSTM)



- 아래쪽에 위치한 LSTM은 단어의 구조적인(Syntax) 의미를 포착할 수 있다.
- 따라서, 단어의 구조적인 표현이 중요해지는 구문 분석(Syntax Analysis)이나 품사 태깅(POS Tagging) 등의 태스크에는 아래쪽 bi-LSTM층이 만들어 내는 은닉 벡터에 가중치를 더 많이 주게 된다.

ELMo Embeddings

ELMo
Embeddings

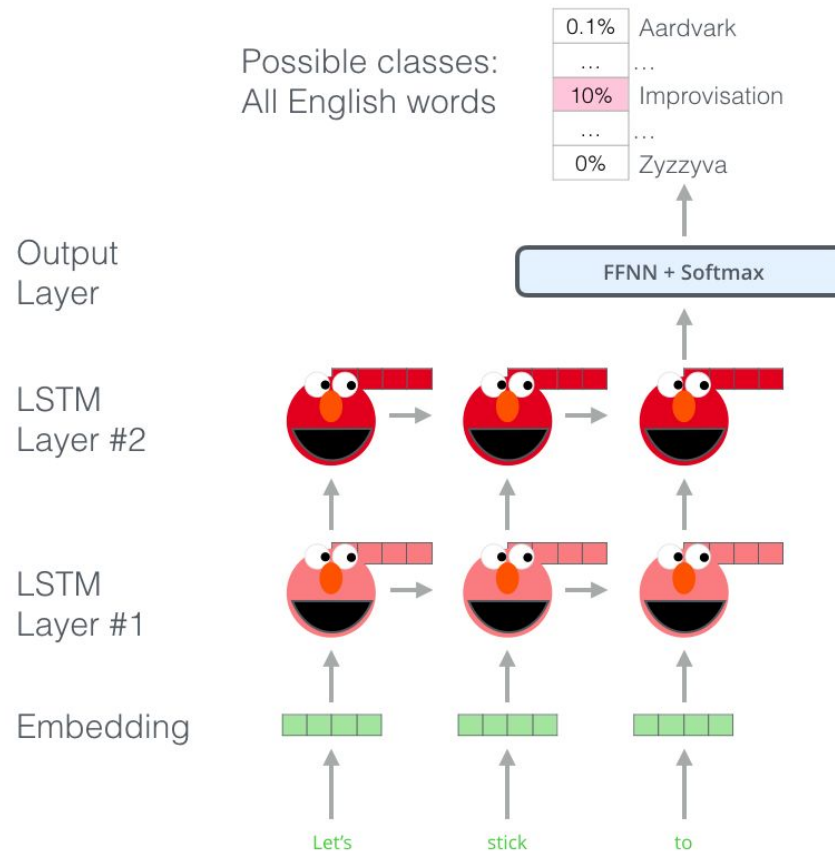


Words to embed



The Process of ELMo

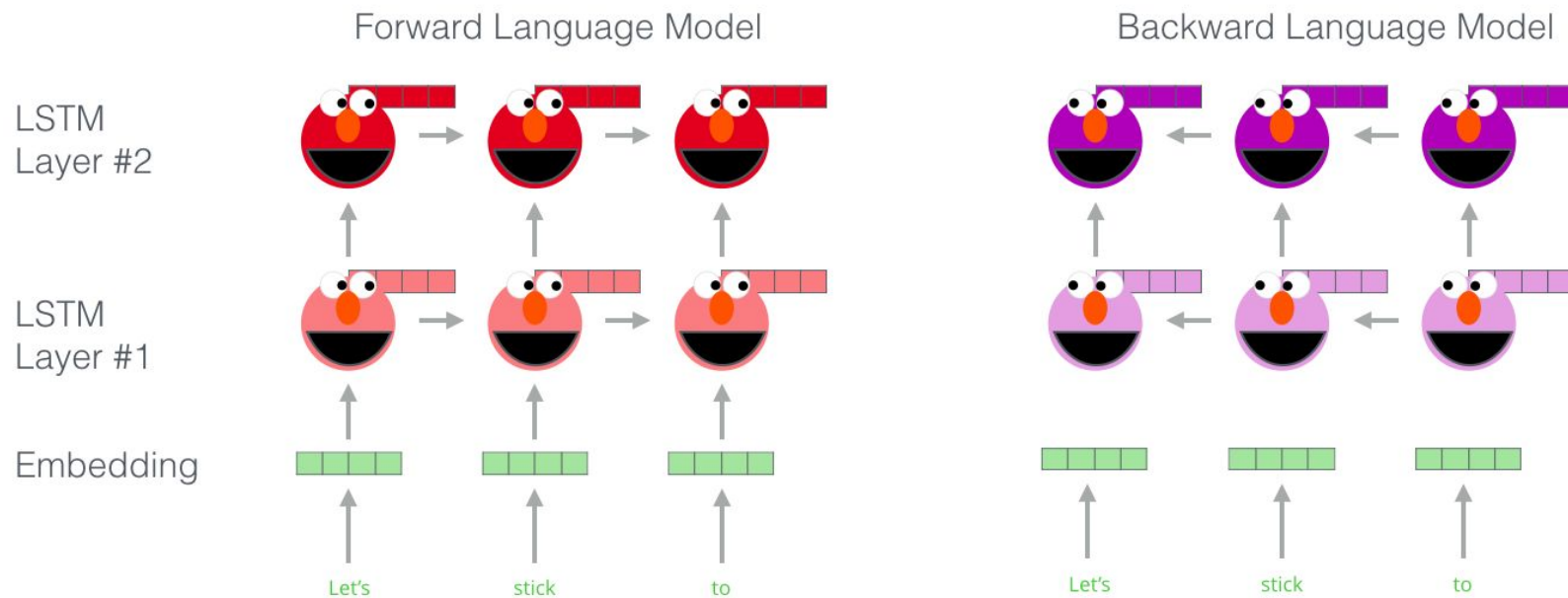
- ELMo는 기본적으로 언어 모델(Language Models)을 따르고 있기 때문에 타겟 단어 이전까지의 시퀀스로부터 타겟 단어를 예측한다.



The Process of ELMo

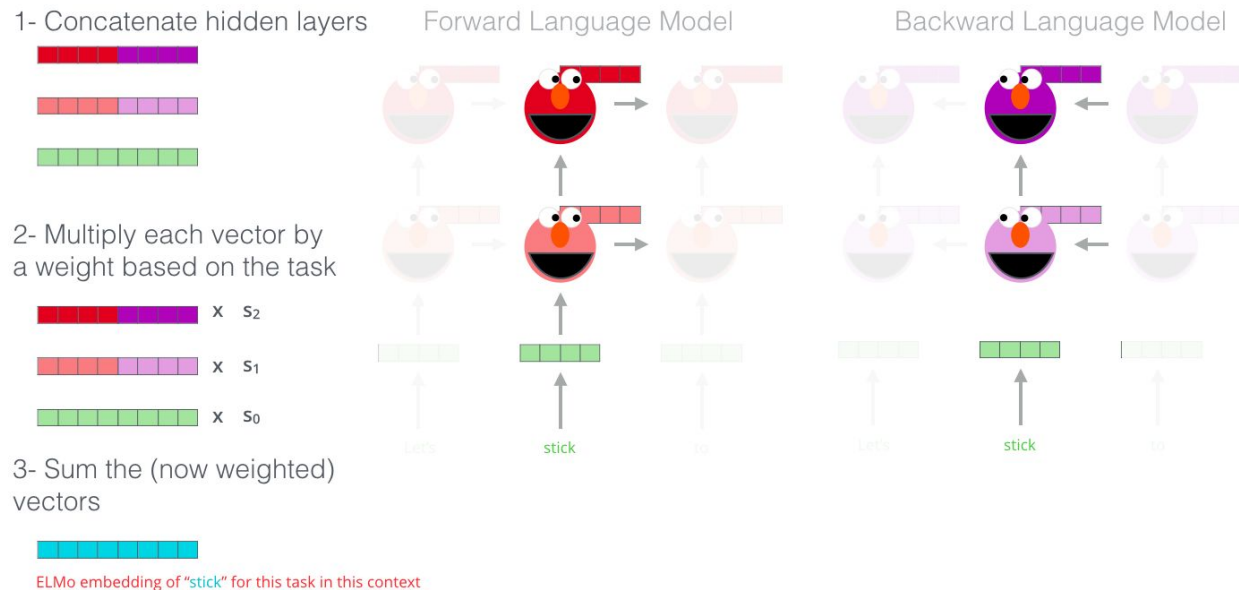
- ELMo는 단방향으로의 LSTM이 아니라 bi-LSTM, 즉 양방향으로 진행되는 LSTM을 사용하여 학습한다.

Embedding of “stick” in “Let’s stick to” - Step #1



The Process of ELMo

- 이후에는 단어마다 순방향(Forward) LSTM의 은닉 벡터 및 토큰 임베딩 벡터와 역방향(Backward) LSTM의 은닉 벡터 및 토큰 임베딩 벡터를 **Concatenate**한다.
 - 그리고 이어 붙인 벡터에 각각 **가중치** s_0, s_1, s_2 를 **곱해 준다**.
 - 마지막으로 세 벡터를 **더해주** 벡터를 ELMo 임베딩 벡터로 사용한다.
- Embedding of "stick" in "Let's stick to" - Step #2



The Process of ELMo

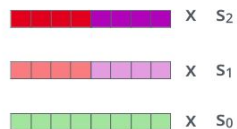
- s_0 , s_1 , s_2 는 학습 되는 파라미터로, 수행하고자 하는 태스크에 따라 달라진다.
- 단어의 **문맥적인 의미**가 중요한 태스크에서는 **상위 레이어**에 곱해주는 s_2 가 커지게 되고,
- **구조 관계**가 중요한 태스크에서는 **하위 레이어**에 곱해주는 s_1 이 커지게 된다.

Embedding of "stick" in "Let's stick to" - Step #2

1- Concatenate hidden layers



2- Multiply each vector by a weight based on the task

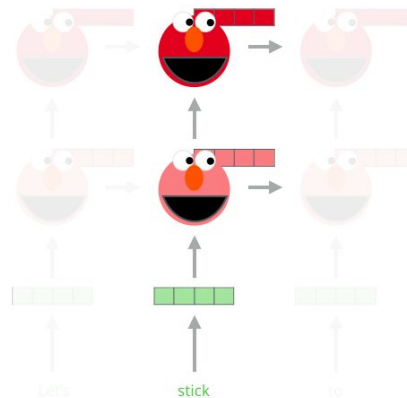


3- Sum the (now weighted) vectors

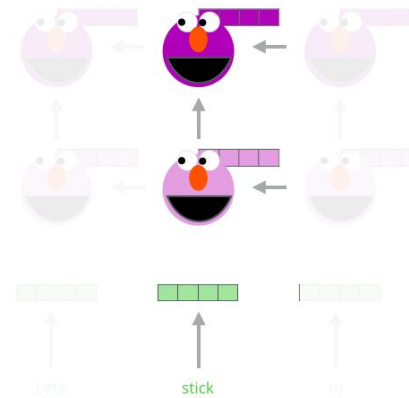


ELMo embedding of "stick" for this task in this context

Forward Language Model

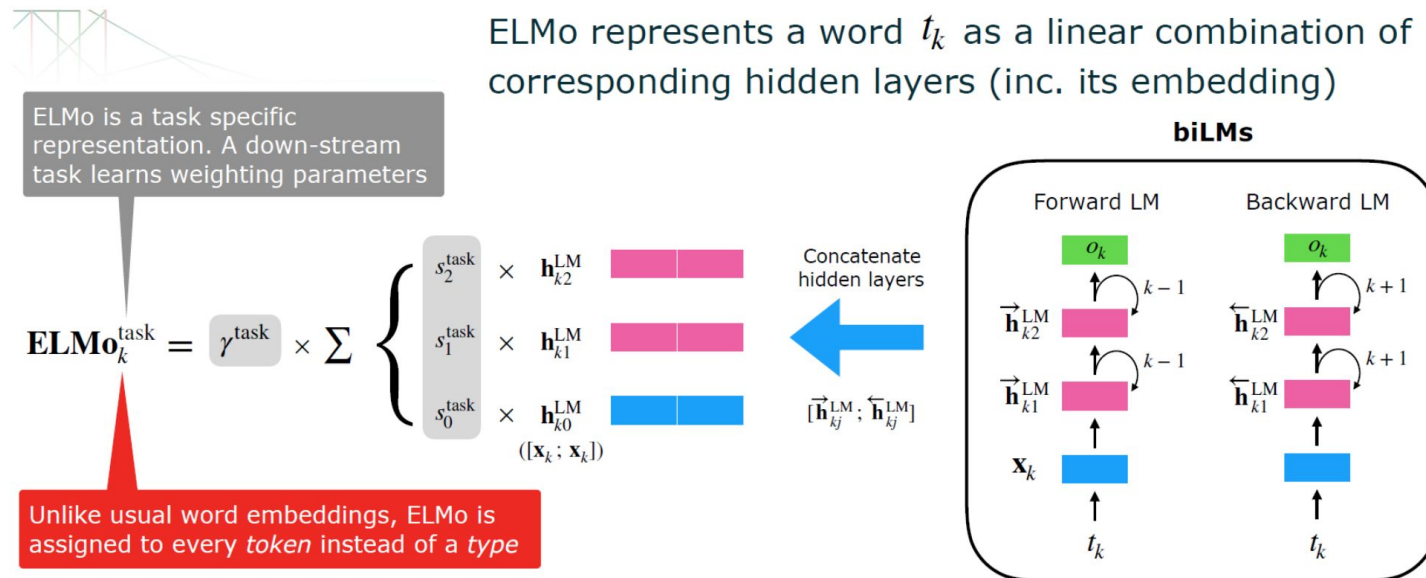


Backward Language Model



The Process of ELMo

- $\overrightarrow{h}_k^{LM}, \overleftarrow{h}_k^{LM}$ 는 각각 k번째 토큰에 대한 순방향 및 역방향 LM의 은닉 벡터를 나타내며 둘을 Concatenate하여 $[\overrightarrow{h}_k^{LM}; \overleftarrow{h}_k^{LM}] = h_k^{LM}$ 이 된다.
- γ 는 세 벡터를 가중합하여 나온 최종 벡터의 요소를 얼마나 증폭 혹은 감소시킬 것인지에 대한 Scale factor이며 태스크에 따라서 달라지게 된다.



- 수식을 통해 ELMo를 구성하고 있는 양방향 언어 모델(Bidirectional Language Model, biLM)에 대해 알아보자.
- 지금까지의 순방향으로 진행되는 언어 모델에서 특정한 **N**개의 단어로 구성된 문장을 만드는 확률을 아래와 같이 나타낼 수 있다.

$$P(t_1, t_2, \dots, t_n) = \prod_{k=1}^N P(t_k | t_1, t_2, \dots, t_{k-1})$$

- 이를 통해, 새로운 단어의 등장은 앞 단어들에 의해서만 영향을 받는다는 것을 알 수 있다.

- ELMo에서 입력 토큰 벡터에 해당하는 x_k^{LM} 는 단어 단위의 임베딩 모델에서 Pre-trained 된 임베딩 벡터를 그대로 가져와서 사용해도 된다.
- 따라서, x_k^{LM} 는 context-independent한 토큰 representation이고,
- 각 은닉 벡터들 $\overrightarrow{h_{k,j}^{LM}}$ ($j = 1, \dots, L$)은 context-dependent한 representation이다.
- 하지만 LSTM이 토큰 임베딩을 학습하여 나타나는 은닉 상태 벡터인 $\overrightarrow{h_k^{LM}}$ 은 문맥을 고려할 수 있게 된다.
- 가장 윗 단의 출력, $\overrightarrow{h_{k,L}^{LM}}$ 은 다음 토큰인 t_{k+1} 를 소프트맥스 레이어를 거쳐 예측한다.

- 역방향에서도 마찬가지로, 역방향 모델을 수식으로 나타내면 다음과 같다.

$$P(t_1, t_2, \dots, t_n) = \prod_{k=1}^N P(t_k | t_{k+1}, t_{k+2}, \dots, t_N)$$

- 역방향 언어 모델에서는 타겟 토큰 t_k 보다 뒤에 위치하는 토큰이 조건으로 주어지며 이로부터 t_k 를 예측하게 된다.
- 역방향 LSTM에 의해 생성된 은닉 상태 벡터 \overleftarrow{h}_k^{LM} 역시 문맥을 고려할 수 있다.

- 이제 이 두 방향을 결합, 즉 두 식을 곱한 후에 t_k 가 위치할 최대 우도(Maximum Likelihood)를 구하면 된다.

$$\prod_{k=1}^N P(t_k | t_1, t_2, \dots, t_{k-1}) \times \prod_{k=1}^N P(t_k | t_{k+1}, t_{k+2}, \dots, t_N)$$

- 위 식에 로그를 취해주면 다음과 같은 식이 나오게 된다.

$$\sum_{k=1}^N (\log P(t_k | t_1, t_2, \dots, t_{k-1}; \Theta_x, \vec{\Theta}_{LSTM}, \Theta_s) + \log P(t_k | t_{k+1}, t_{k+2}, \dots, t_N; \Theta_x, \vec{\Theta}_{LSTM}, \Theta_s))$$

- Θ_x, Θ_s 는 토큰 벡터 층과 소프트맥스 층에서 사용되는 파라미터를 가리키는 것으로, 두 방향 모두에 같은 파라미터가 적용된다.
- Θ_{LSTM} 은 각 방향 LSTM층에서 사용되는 파라미터로, 방향마다 다른 파라미터가 학습된다.

- L개의 층을 가진 양방향 언어 모델을 통해서 생성된 L개의 은닉 벡터 $\overrightarrow{h_k^{LM}}, \overleftarrow{h_k^{LM}}$ 를 이어 붙이고(Concatenate) 최초의 임베딩 토큰 벡터 x_k^{LM} 를 자기 자신과 이어 붙인 벡터의 집합을 R_k 라고 하겠다.

$$R_k = \{x_k^{LM}, \overrightarrow{h_k^{LM}}, \overleftarrow{h_k^{LM}} \mid j = 1, \dots, L\} = \{h_{k,j}^{LM} \mid j = 0, \dots, L\}$$

- 이때 R_k 는 $2L+1$ 개의 representation의 집합이다. ($1+L+L$)
- 이어 붙인 $2L+1$ 개의 토큰을 선형 결합한 뒤에 최종 ELMo 벡터가 만들어지게 된다.
- $(h_{k,0}^{LM} = [x_k^{LM}; x_k^{LM}])$

- 선형 결합시에 사용 되는 가중치 s 는 ELMo를 사용하고자 하는 태스크에 따라 달라진다.
- 이때, 이 가중치는 softmax-normalized 된다.
- γ 는 벡터 요소의 크기를 조정하는 Scale factor로 역시 태스크에 따라 변하게 된다.

$$\text{ELMO}_k^{\text{task}} = E(R_k; \Theta^{\text{task}}) = \gamma^{\text{task}} \sum_{j=0}^L s_j^{\text{task}} h_{k,j}^{LM}$$

Evaluation

- 아래는 다양한 태스크에서 ELMo가 보여준 성능을 기록한 것이다.
- 몇몇 태스크에 대하여 당시의 **SOTA** 모델보다 좋은 성능을 보였음을 알 수 있다.

- Performances

TASK	PREVIOUS SOTA		OUR BASELINE	ELMo + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
* SNLI	Chen et al. (2017)	88.6	88.0	88.7 \pm 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 \pm 0.19	90.15	92.22 \pm 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 \pm 0.5	3.3 / 6.8%

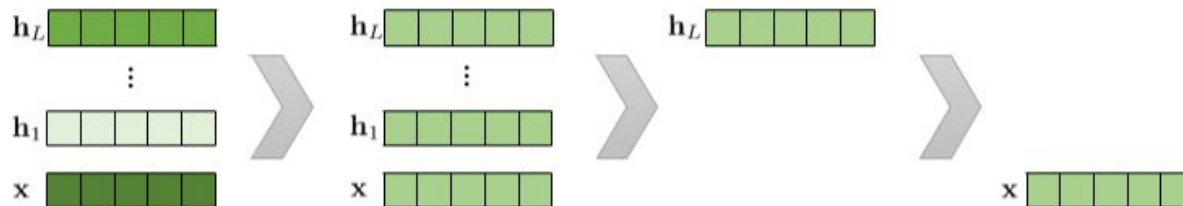
* **Natural Language Inference(NLI)** : 두 문장(Text, Hypothesis)이 주어졌을 때, entailment, contradiction, neutral 중 하나로 분류하는 태스크



How is ELMo best utilized?

- “임베딩 벡터와 은닉 벡터에 가중치를 어떻게 부여할 것인가?”
- 논문에서 적용했던 것과 같이 $L+1$ 개의 벡터에 모두 다른 가중치를 적용할 때의 성능이 가장 좋은 것을 보여준다.
- 이는 가중치를 적용하지 않을 때보다 더 좋은 성능을 보여준다.
- 벡터의 선형 결합을 사용하지 않고 하나의 벡터만 사용한다면 L 번째 층, 즉 최상단 은닉층이 생성한 벡터를 사용하는 것이 그냥 단어 임베딩 벡터를 사용하는 것보다 좋다는 연구 결과가 있다.

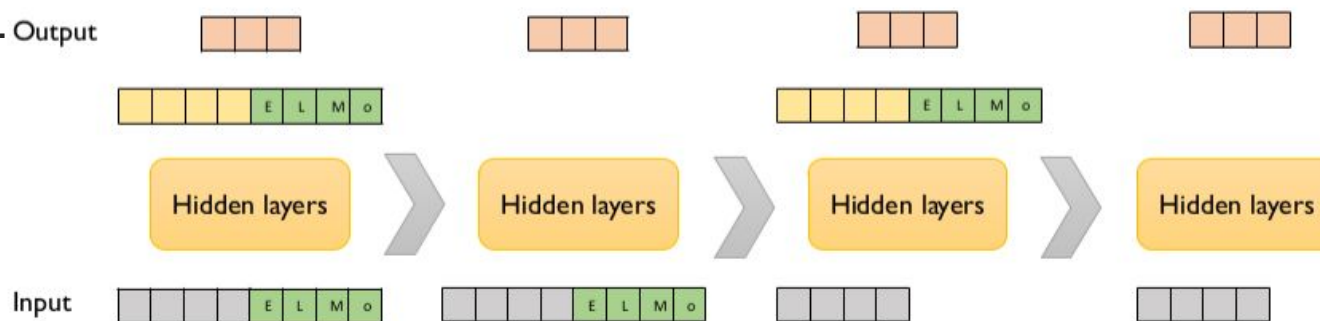
- Analysis: Alternate layer weighting scheme



How is ELMo best utilized?

- “ELMo 임베딩 벡터를 어떤 단계에 **Concatenate**하는 것이 좋은가?”
- 입출력 단계에 모두 ELMo 임베딩을 적용하는 것이 가장 좋은 것을 보여준다.
- 입, 출력 벡터 중 하나에만 적용하는 경우는 모두 적용한 경우보다는 떨어지지만,
- 아무것도 사용하지 않은 모델보다는 좋은 성능을 보여준다는 연구 결과가 있다.

• Analysis: Where to include ELMo?





Code

```
class ELMo(nn.Module):
    def __init__(self, vocab_size, output_dim: int, emb_dim: int, hid_dim: int, prj_dim, kernel_sizes: List[Tuple[int]],
                  seq_len: int, n_layers: int=2, dropout: float=0.):
        """
        Paramtrs:
            vocab_size: Character vocabulary size
            output_dim: Word vocabulary size
            emb_dim: Embedding dimension of chracter tokens
            hid_dim: hidden dimension for bi-directional language model
            kernel_sizes: `list`. Kernel_sizes for the convolution operations.
            seq_len: character sequence len
            n_layers: the number of layers of LSTM. default 2
            dropout: dropout of LSTM
        """
        super(ELMo, self).__init__()

        self.embedding = CharEmbedding(vocab_size, emb_dim, prj_dim, kernel_sizes, seq_len)
        self.bilms = BidirectionalLanguageModel(hid_dim, hid_dim, n_layers, dropout)

        self.predict = nn.Linear(hid_dim, output_dim)
```


Code

```
def forward(self, x: torch.Tensor):
    """
    Parameters:
        x: Sentence
    Dimensions:
        x: [batch, seq_len]
    """
    emb = self.embedding(x) # [Batch, Seq_len, Projection_layer (==Emb_dim==HId_dim)]
    _, last_output = self.bilms(emb) # [Batch, Seq_len, Hidden_size]
    y = self.predict(last_output) # [Batch, Seq_len, VOCAB_SIZE]

    return y # only use the output of the last LSTM of the biLM on training step
```

Code

```
class BidirectionalLanguageModel(nn.Module):
    def __init__(self, emb_dim: int, hid_dim: int, prj_emb: int, dropout: float=0.) -> None:
        """
        > We use dropout before and after every LSTM layer
        """
        super(BidirectionalLanguageModel, self).__init__()
        self.lstms = nn.ModuleList([nn.LSTM(emb_dim, hid_dim, bidirectional=True, dropout=dropout, batch_first=True),
                                     nn.LSTM(prj_emb, hid_dim, bidirectional=True, dropout=dropout, batch_first=True)])
        self.projection_layer = nn.Linear(2*hid_dim, prj_emb)
```

```
def forward(self, x: torch.Tensor, hidden: Tuple[torch.Tensor]=None):
    """
    Parameters:
        x: A sentence tensor that embeded
        hidden: tuple of hidden and cell. The initial hidden and cell state of the LSTM
    Dimensions:
        x: [Batch, Seq_len, Emb_size]
        hidden: [num_layers * num_directions, batch, hidden_size], [num_layers * num_directions, batch, hidden_size]
    """
    # > "...add a residual connection between LSTM layers"
    first_output, (hidden, cell) = self.lstms[0](x, hidden) # [Batch, Seq_len, # directions * Hidden_size]
    # TODO: [Batch, Seq_len, # directions * Hidden_size]를 그냥 넣는지,
    #         [Batch, Seq_len, # directions, Hidden_size]로 바꾼 후(nn.projection) 더해서 넣는지 확인이 필요
    projected = self.projection_layer(first_output) # [Batch, Seq_len, Projection_size]
    second_output, (hidden, cell) = self.lstms[1](projected, (hidden, cell)) # [Batch, Seq_len, # directions * Hidden_size]

    second_output = second_output.view(second_output.size(0), second_output.size(1), 2, -1) # [Batch, Seq_len, # directions, Hidden_size]
    second_output = second_output[:, :, 0, :] + second_output[:, :, 1, :] # [Batch, Seq_len, Hidden_size]
    return first_output, second_output # [Batch, Seq_len, Hidden_size]
```

Conclusion

- ELMo를 시작으로 대량의 말뭉치로부터 생성된 품질 좋은 임베딩 벡터를 만드는 모델이 많이 사용 되었다.
- ELMo는 이후에 등장하는 트랜스포머 기반의 BERT나 GPT보다 많이 사용되지는 않는다.
- 하지만, 좋은 품질의 임베딩 벡터를 바탕으로 적절한 Fine-tuning후에 여러 task에 적용하는 전이 학습(Transfer learning)의 시초격인 모델로서의 의의가 있다고 할 수 있다.