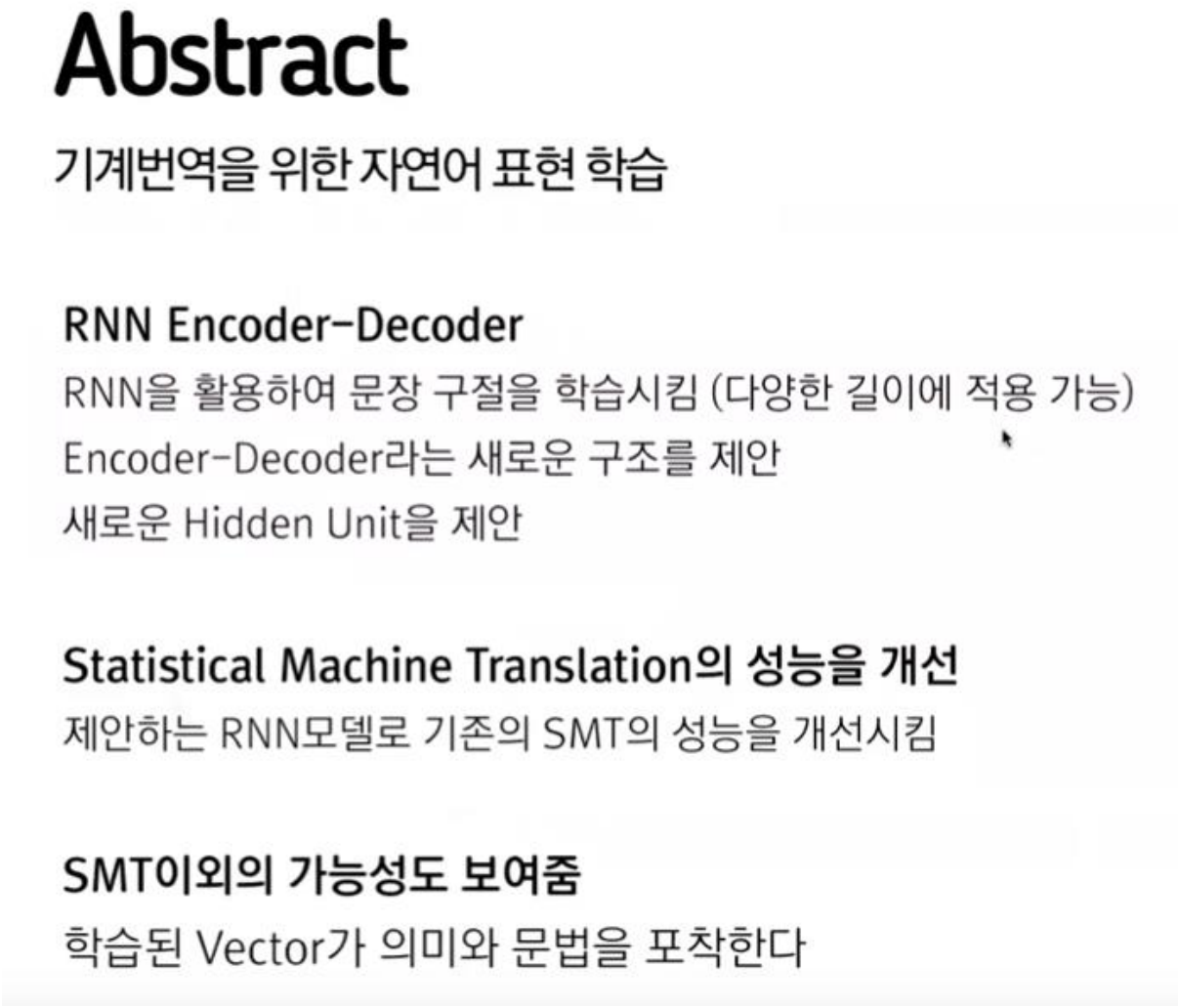
PR-003:Learning phrase representations using RNN encoder-decoder for statistical machine translation

RNN 인코더 디코더 2014년

논문 선택 이유: tensorflow tutorial을 따라하고 있는데 seq2seq 모델을 따라하고 있음.



새로운 가능성을 많이 보여줌

RNN 간단개요

Sequence 데이터가 들어왔을 때 많이 쓰임.

단순히 Input x를 받는 게 아니고 그 이전 input이 중요할 때 사용

Input이 들어왔을 때 state를 구하고 이전 state를 input과 함께 넣음으로서 새로운 state를 만드는 구조

x라는 seq data가 들어왔을 때 hidden unit에 들어와서 hidden state를 뱉어내는데 이거 자체가 input으로 다음 unit에 들어가는 것.

기계 번역(Machine Translation)

Rule-based MT
Dictionary, 문법 기반의 번역 (Parser, Analyzer, Generator, Transfer Lexicon)

Statistical MT
이미 번역된 문서들을 바탕으로한 통계 기반의 번역

Hybrid MT
Rule-based와 Statistical을 함께 사용

Neural MT
Deep Learning을 활용한 번역

Rule-based로 하다 보면 표현의 제약이 있음.

특히나 한국어가 복잡하고, 언어 간 교환시엔 부적합.

어떤 source가 왔을 때 target string이 자주 나온다 하는 것들을 통계

두 개를 합치기도 함

이제는 딥러닝이 트랜드

이번 논문이 그런 것의 시작

Statistical Machine Translation

$$p(e|f)$$

f가 주어졌을 때 e가 나올 확률

$$p(e|f) \propto p(f|e)p(e)$$

e가 주어졌을 때 f가 나올 확률과
e가 나올 확률의 곱과 비례한다

$$\tilde{e} = arg\max_{e \in e^*} p(e|f) = arg\max_{e \in e^*} p(f|e)p(e)$$

가장 높은 확률이 나오는 표현들을 골라서 최종 번역 $\tilde{e}$ 를 찾는다

입력 문장이 왔을 때 영어 문장이 나올 확률을 구하는 것

주로 SMT에서 기본 이론을 많이 쓰는 이유가, 이런 두개의 문제로 분할해서 성능 향상 시킬 여지를 둘 수 있으므로.

RNN Encoder-Decoder 구조 소개

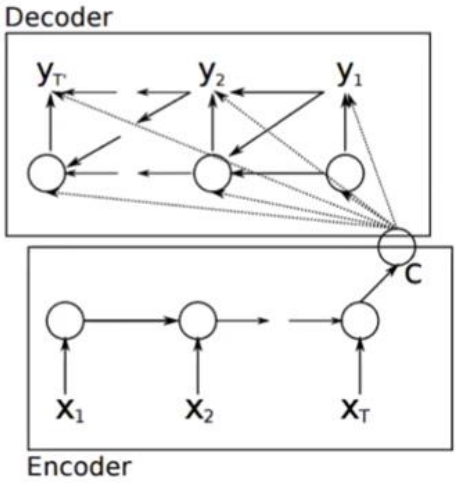


Figure 1: An illustration of the proposed RNN Encoder–Decoder.

$$\mathbf{h}_{\langle t \rangle} = f\left(\mathbf{h}_{\langle t-1 \rangle}, y_{t-1}\right.$$

$$\mathbf{h}_{\langle t \rangle} = f\left(\mathbf{h}_{\langle t-1 \rangle}, x_t\right)$$

이런 히든 유닛이 좀 독특한 방식을 가짐

말 그대로 seq를 받아서 하나의 c 벡터를 뽑는 모델

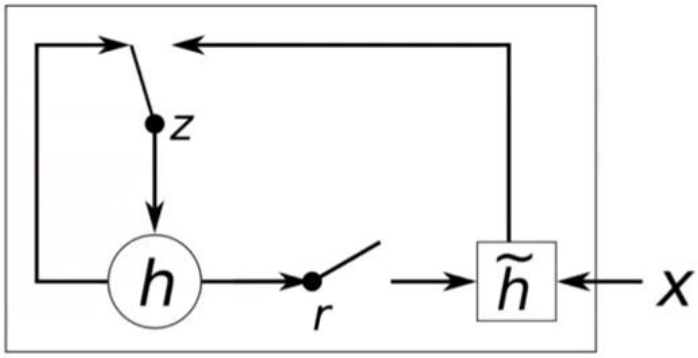
디코더는 반면에 Input을 c 벡터를 받아서 출력은 target language로 번역을 내놓는 generator 역할을 함.

기존의 RNN과 유사한데, c를 포함한다는 다른 점이 있음.

Q. 그림에서 C가 입력의 화살표로 간 거는 말이 되는데 y로 바로 간 거는 … ?

A. 많이 헛갈린다. 사실 C가 y로 바로 가지는 않는다.

Hidden Unit 소개



Reset Gate $$r_j = \sigma\left(\left[\mathbf{W}_r\mathbf{x}\right]_j + \left[\mathbf{U}_r\mathbf{h}_{\langle t-1\rangle}\right]_j\right)$$

Update Gate $$z_j = \sigma\left(\left[\mathbf{W}_z\mathbf{x}\right]_j + \left[\mathbf{U}_z\mathbf{h}_{\langle t-1\rangle}\right]_j\right)$$

R이 완전히 열러버리면 이전 state는 버리고 새로운 state만 옴

Z는 새로운 state를 얼마나 반영할 것인가를 결정

다양한 Time scale을 좀 더 유연하게 대응하기 위해서는 이렇게 분리해 놓는 구조가 필요하더라.

Ex) 짧은 표현들이 많이 있는 문장이라고 한다면 r도 활발히 열림. (다음 표현 영향 X)

긴 표현은 Z가 활발히 작동

이것들을 분리해서 학습하기 때문에 훨씬 유연함.

RNN Encoder

$$h_j^{\langle t \rangle} = z_j h_j^{\langle t-1 \rangle} + (1 - z_j)\tilde{h}_j^{\langle t \rangle}$$

$$\tilde{h}_j^{\langle t \rangle} = \tanh\left(\left[\mathbf{W} e(\mathbf{x}_t) \right]_j + \left[\mathbf{U} \left(\mathbf{r} \odot \mathbf{h}_{\langle t-1 \rangle} \right) \right]_j \right)$$

$$\mathbf{c} = \tanh\left(\mathbf{V}\mathbf{h}^{\langle N \rangle} \right)$$

인코더의 hidden state는 update gate (z) 를 얼마나 반영할 것인가, 그리고 새롭게 계산한 state를 얼마나 반영할 것인가를 계산

새로운 state는 입력을 받아서 사전 index값 x_t 을 embedding해서 실제 vector 값을 가지게 됨. 이전 state에 reset gate를 element 곱 한 다음에 다시 weight를 곱해서 구하고 이것들을 더하면 새로운 state

인코더는 입력을 받아서 c를 구하는데 c는 새로운 state를 N번 구하고 weight를 구하고 tanh를 씌워서 구할 수 있음.

Q. LSTM과의 차이점은? Unit 안에 여러 gate가 있으면 훨씬 유연해야 되는데

A. 논문에서는 그러한 설명은 없음. LSTM과 유사하기 위해서 hidden unit을 적용했다고 함. 그것보다 좀 더 간단하게 하기 위해서 바꿨다. 퍼포먼스 이슈로 예측. LSTM 적용 시 성능 궁금함.

Q. LSTM가지고 굉장히 많은 실험을 해봄. Update 같은 경우는 LSTM은 3개가 있고 Input이랑 forget을 합쳐놓은 컨셉 (z, 1-z)임. LSTM 했을 때 Input, forget 값이 다 0~1 사이 값을 가질 수 있는데, 그거를 두개 합해서 1이 되는 쪽으로 여기 잇는 것처럼 했을 때 성능이 크게 차이가 없다는 실험 결과를 어디서 본 것 같다.

A. 예. 알겠습니다.

RNN Decoder

$$\mathbf{h'}^{\langle 0 \rangle} = \tanh\left(\mathbf{V'c}\right)$$

$$h'^{\langle t \rangle}_j = z'_j h'^{\langle t-1 \rangle}_j + (1 - z'_j)\tilde{h}'^{\langle t \rangle}_j$$

$$\tilde{h}'^{\langle t \rangle}_j = \tanh\left(\left[\mathbf{W'}e(\mathbf{y}_{t-1})\right]_j + r'_j\left[\mathbf{U'h'}_{\langle t-1 \rangle} + \mathbf{Cc}\right]\right)$$

$$z'_j = \sigma\left(\left[\mathbf{W'}_z e(\mathbf{y}_{t-1})\right]_j + \left[\mathbf{U'}_z \mathbf{h'}_{\langle t-1 \rangle}\right]_j + \left[\mathbf{C}_z \mathbf{c}\right]_j\right)$$

$$r'_j = \sigma\left(\left[\mathbf{W'}_r e(\mathbf{y}_{t-1})\right]_j + \left[\mathbf{U'}_r \mathbf{h'}_{\langle t-1 \rangle}\right]_j + \left[\mathbf{C}_r \mathbf{c}\right]_j\right)$$

하나 다른 점은 Cc를 추가하게 됨. z, r을 구할 때에도.

RNN Encoder-Decoder 학습

$$P(y_t|y_{t-1}, y_{t-2}, \dots, y_1, \mathbf{c}) = g\left(\mathbf{h}'_{\langle t \rangle}, y_{t-1}, \mathbf{c}\right)$$

$$\max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} \log p_{\boldsymbol{\theta}}(\mathbf{y}_n \mid \mathbf{x}_n).$$

각 표현에 관한 확률을 알고 싶음.

t번째 확률은 h, y, c의 activation function (softmax 같은 것)으로 구하게 됨.

실제 학습은 en/de 따로 아니고 같이 시키고 x 줬을때 y 나올 확률의 평균값의 최대.

활용

SMT에 적용

$$\log p(\mathbf{f} \mid \mathbf{e}) = \sum_{n=1}^{N} w_n f_n(\mathbf{f}, \mathbf{e}) + \log Z(\mathbf{e})$$

Branch가 나올 확률.

F가 feature 함수: Src, target data 넣은 함수에 weight 곱해서 구하고 norm 상수 더해서 구함.

확률값이 하나의 피쳐 함수로 들어가게 되죠. 그래서 기존 SMT에 새로운 feature를 추가하는 방식으로 scoring.

성능

Models	BLEU	
	dev	test
Baseline	30.64	33.30
RNN	31.20	33.87
CSLM + RNN	31.48	34.64
CSLM + RNN + WP	31.50	34.54

BLEU (bilingual evaluation understudy)

기계 번역의 품질을 측정하는데 사용하는 지표. 실제 사람이 한 번역과 기계 번역의 유사성을 계산하는 방식으로 구함. 간단하고 쉽게 구할 수 있다는 장점이 있음.

베이스라인==SMT

CLSM(continuous Space Language Model) : 2006년에 신경망을 활용해서 확률을 구한 것.

WP(Word Penalty): Unknown(자주 안나오는 단어) 개수 자체를 SMT의 새로운 feature로 넣은 것.

사실은 크게 차이가 없음.

Q. baseline==CSLM?

A. !=. CSLM VS RNN 은 없다.

왜 성능이 잘 나올까

SMT에 적용

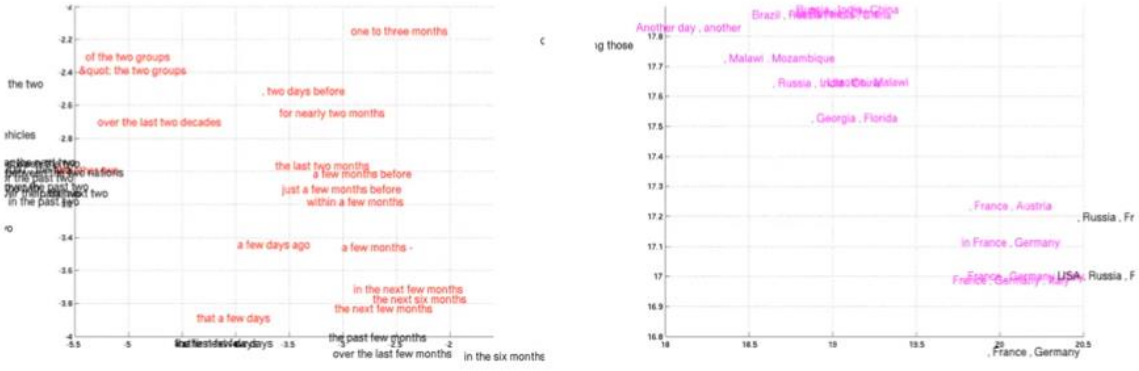
- CSLM의 contribution과 많이 겹치지 않음
- 통계적으로 도출된 Score가 아님
  - 출현 빈도가 많지 않은 표현들에 대해서 보완이 가능함
- RNN Encoder-Decoder가 짧은 표현을 선호함
  - 일반적으로 BLEU 값은 짧은 표현에서 높게 나옴

빈도수는 안넣다 보니까 보완됨

짧을수록 유추 가능성 높아짐.

이외의 활용 방안

- SMT를 완전히 대체
- 범용적인 Phrase Representation
  - 의미와 문법 모두를 담아냄



Word to Vector처럼 PR 의미가 있다. 평면에 Vector를 표현해봤더니 의미와 문법 모두 담아냄

결론

- SMT의 성능을 높일 수 있었다
- 기존 번역 시스템에 쉽게 적용 가능하다
- 완전히 SMT를 대체할 가능성을 보았다
- 범용 Phrase Representation으로 활용가능하다

Q. 디코더 쪽에서 y(t)를 구하는 식이 있었나요?

A. 논문에서는 나와있지 않지만 아마 일반적으로 h(t)에다가 weight를 곱해서 구할것으로 추정.

Q.Q. 근데 C가 화살표 있는게 y를 고려해서 같이 구하는게 아닐까요? 괜히 그려놓은 것 같지는 않은 것 같은데...

A. 그러네요, 단순히 weight를 곱하는 것 말고도 뭔가가 있을 것 같네요.

...열띤 토론...

Q. Softmax를 쓰면 k-Dim one hot vector를 쓰는 걸로 알고 있는데 k가 정해져 있나요?

A. 네. 그냥 k개를 분모로 두고 있음

Q. K개를 약간 하이퍼파라미터 같은 거네요

A. K개를 구할 때에는 아마 constant를 잡고 하는게 일반적인 것 같더라구요. 사전 사이즈를 몇 개로 나눌지를 결정을 해서, 그렇게 정하는 걸로 알고 있어요.

Q. RNN en/de coder 구조로 했을 때 장점이 있나요? seq가 나오면 rnn으로도 할 수 있을텐데

A. 가장 큰 장점: length. RNN을 활용하더라도 fixed된 length를 써야되는 경우가 많음. 인코더/디코더 길이다 달라도 됨. 아무리 길더라도 C 하나로 인코딩이 될꺼고, 디코딩 하면 다양한 length를 보완할 수 있다.

Q. 그냥 RNN 은 안되나요?

A. 이 논문은 하나의 C로 모은다음에 뱉는다는게, 중간부터 뽑아내기 시작하면 주술이 안끝났

는데 모르니까 다 듣고 어순에 따라 하나씩 내뱉을 때 이런 구조가 유리하다.

Q. 근데 RNN도 입력이 다 들어간 후에 출력해도 되잖아요

A. 그게 이거다

Q. 근데 Nonlinear function 한번 더 곱하잖아요? C로 갈 때. 사실 hidden에서 code로 갈 때 한 번 더 하는게 RNN 이랑 차이점 같은데.

A. Nonlinear function 은 엄청난 의미가 있는 건 아닌 것 같고, 가변 IO 길이가 다를 때, 어순이 다를 때를 고려하기 위해서 나온 게 인코더/디코더 구조라고 알고 있음.