Lecture 09. Practical Tips for Final Projects

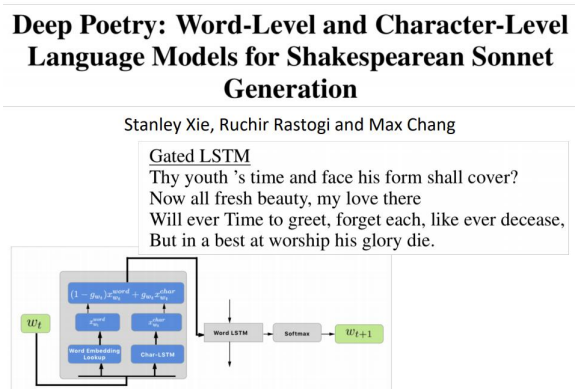
Ⅰ. Research

1. SQuAD : 스탠포드 대학의 NLP 그룹에서 크라우드 소싱을 통해 만든 의키피디아 article에 에 대한 107.785개의 질문-대답 데이터셋. 한국에느 KorQuAD가 있음. (https://rajpurkar.github.io/SQuAD-explorer/)
2. 연구에서 중요시하는 요소들 : Baseline, Benchmark, Evaluation, Error Analysis, Paper Writing

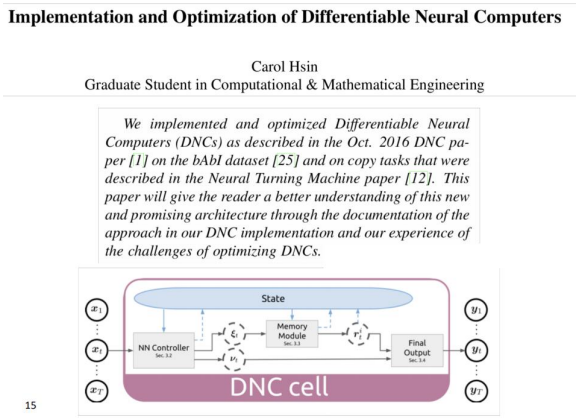
Ⅱ. NLP 연구

1. NLP 연구 분야

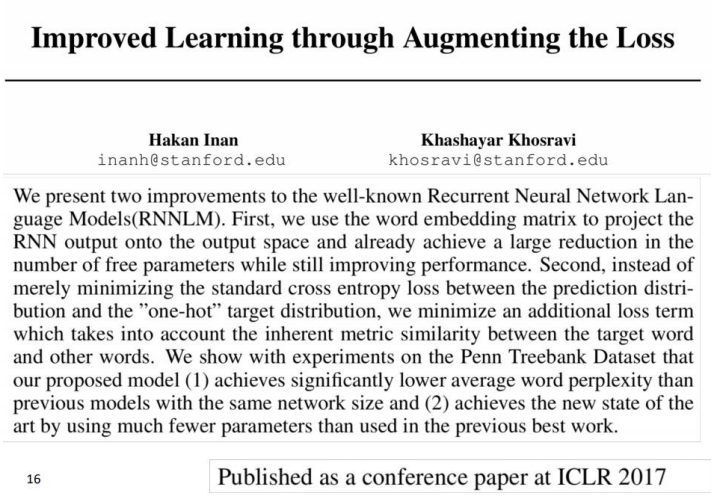
-모델의 application을 찾아보고 어떻게 효율적으로 적용할지 찾는 연구



-복잡한 neural architecture을 구현해보고 특정 데이터에 대한 성능을 측정하는 연구



-새롭거나 기존의 NN 모델을 구상한 후 구현하여 실험적인 데이터를 토대로 성능 향상을 보여주는 연구

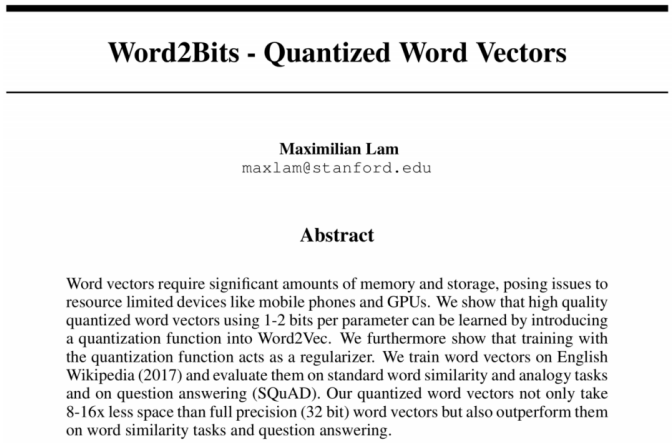


Improved Learning through Augmenting the Loss

Hakan Inan inanh@stanford.edu
Khashayar Khosravi khosravi@stanford.edu

We present two improvements to the well-known Recurrent Neural Network Language Models(RNNLM). First, we use the word embedding matrix to project the RNN output onto the output space and already achieve a large reduction in the number of free parameters while still improving performance. Second, instead of merely minimizing the standard cross entropy loss between the prediction distribution and the "one-hot" target distribution, we minimize an additional loss term which takes into account the inherent metric similarity between the target word and other words. We show with experiments on the Penn Treebank Dataset that our proposed model (1) achieves significantly lower average word perplexity than previous models with the same network size and (2) achieves the new state of the art by using much fewer parameters than used in the previous best work.

-그냥 새로운 연구



Word2Bits - Quantized Word Vectors

Maximilian Lam
maxlam@stanford.edu

Abstract

Word vectors require significant amounts of memory and storage, posing issues to resource limited devices like mobile phones and GPUs. We show that high quality quantized word vectors using 1-2 bits per parameter can be learned by introducing a quantization function into Word2Vec. We furthermore show that training with the quantization function acts as a regularizer. We train word vectors on English Wikipedia (2017) and evaluate them on standard word similarity and analogy tasks and on question answering (SQuAD). Our quantized word vectors not only take 8-16x less space than full precision (32 bit) word vectors but also outperform them on word similarity tasks and question answering.

2. 연구의 시작

-NLP논문에 대한 ACL Anthology를 참고한다.(https://aclanthology.info/)

-주요 ML 컨퍼런스들의 논문을 참조한다.(NeurIPS, ICML, ICLR,...)

-기존의 프로젝트들을 참조한다.

-출판 전 논문들을 본다.(https://arxiv.org/)

-제일 좋은 방법은 우리 세상 속에 존재하는 재미있는 문제들을 찾는 것이다.

-http://www.arxiv-sanity.com/

-https://paperswithcode.com/sota

3. 꼭 가져야할 것

(1) 적절한 데이터

성능, testing 등을 위해 최소 만 개의 레이블된 데이터가 필요하다. 데이터를 찾는 방법은 크게 직접 데이터를 구한 후 전처리하고 레이블링 하기, 기존의 프로젝트에서 구한 데이터/기업에서 갖고 있는 데이터, 공개되고 잘 관리된 dataset 활용하기(캐글, AIHub, AI데이터가공바우처사업 등등)

-https://linguistics.stanford.edu/resources/resources-corpora

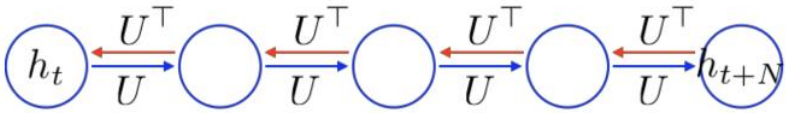
-http://statmt.org/

(2) Feasible task

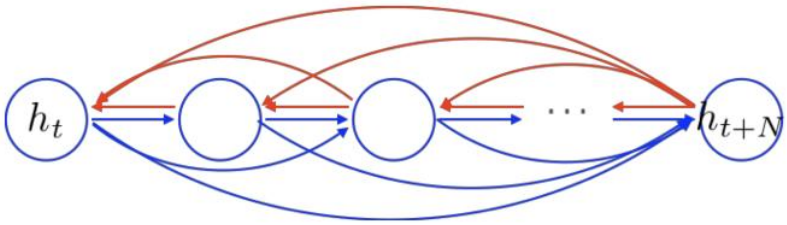
(3) Automatic evaluation metric

Ⅲ. Gated Recurrent Units & LSTM

It implies that the error must backpropagate through all the intermediate nodes:



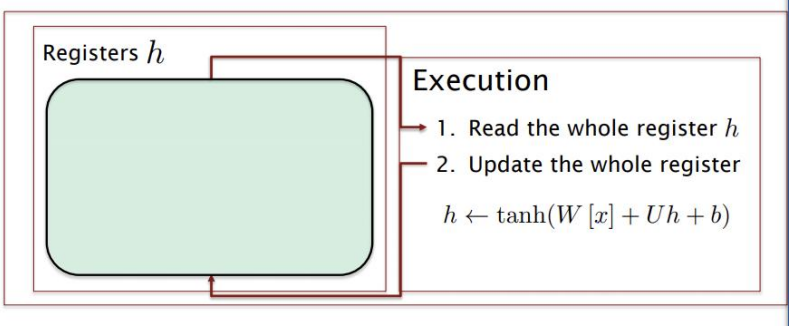
Perhaps we can create shortcut connections.



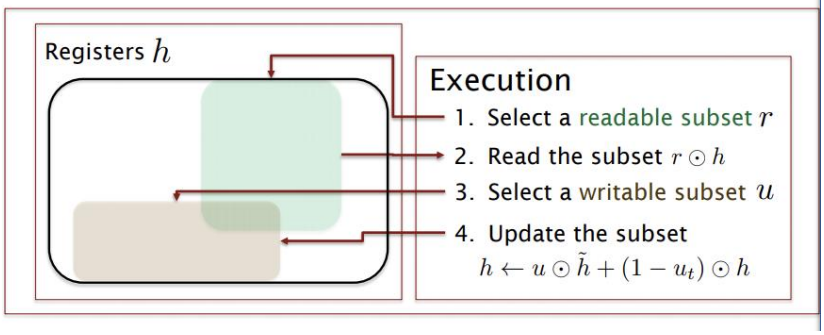
RNN을 이해하는게 중요하고 vanishing gradient는 문제가 많은데 gradient가 0이 되면 뭐가 문제인지 모르기 때문이다.(dependency 문제인지, 파라미트 configuaration문제인지 등등) Backpropagation을 위해서 shortcut connection을 만들거나 adaptive하게 만들 수 있다.

Tanh-RNN의 경우 h레지스터 전체를 읽고 전체 레지스터를 업데이트 하지만 GRU의 경우 읽을 수 있는 subset r을 선택하고 subset r⊙h를 읽은 다음 쓸 수 있는 subset u를 선택한다음 subset을 아래와 같이 update한다. GRU가 더 현실적이지만 attention에 약간의 overlap이 있을 수 있다.

tanh-RNN ….



GRU …



Gated Recurrent Unit
[Cho et al., EMNLP2014; Chung, Gulcehre, Cho, Bengio, DLUFL2014]

$$h_t = u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1}$$
$$\tilde{h}_t = \tanh(W\,[x_t] + U(r_t \odot h_{t-1}) + b)$$
$$u_t = \sigma(W_u\,[x_t] + U_u h_{t-1} + b_u)$$
$$r_t = \sigma(W_r\,[x_t] + U_r h_{t-1} + b_r)$$

Long Short-Term Memory
[Hochreiter & Schmidhuber, NC1999; Gers, Thesis2001]

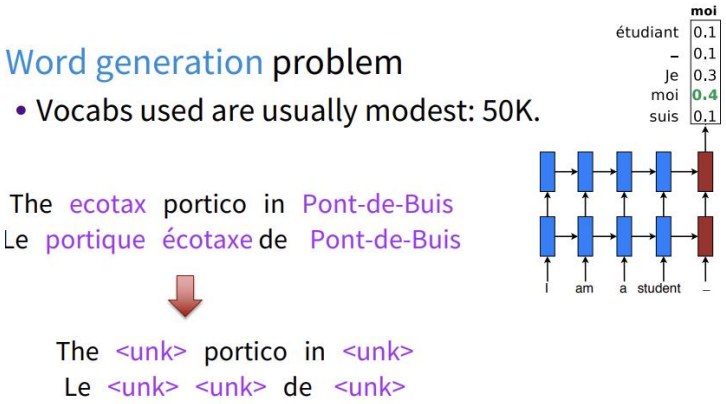
$$h_t = o_t \odot \tanh(c_t)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$\tilde{c}_t = \tanh(W_c\,[x_t] + U_c h_{t-1} + b_c)$$
$$o_t = \sigma(W_o\,[x_t] + U_o h_{t-1} + b_o)$$
$$i_t = \sigma(W_i\,[x_t] + U_i h_{t-1} + b_i)$$
$$f_t = \sigma(W_f\,[x_t] + U_f h_{t-1} + b_f)$$

LSTM gate들은 모든 연산들이 모든 것들의 맨 위에 밀어 넣어지기 보다는 잊어지거나 무시될 수 있다. 다음 단계를 위한 비선형 업데이트는 RNN과 같다. 곱하기보다, 비선형적인 것과 c_t-1를 더해서 c_t를 얻을 수 있다. c_t와 c_t-1사이에 직접적이고 선형적인 연결이 생긴다.

IV. The largest output vocabulary problem NMT(or all NLG)

Word generation problem : softmax가 expensive해서 일어난다.

해결 방법으로는 hierarchical softmax, large vocabulary set을 몇 개의 모델들로 나눠서 train한 후에 알맞은 번역 고르기, attention(단순 사전 검색), 추후에 배울 word pieces, char models 같은 것들이 있다.



MT 수동 평가방법으로는 adequacy and fluency(5점/7점 척도), 오류 분석, 번역 순위 메기기가 있고 자동 방법으로는 BLUE(Bilingual Evaluation Understudy), TER, METEOR가 있다.

Ⅴ. BLUE Evaluation Metric

BLUE score란 성과지표로 데이터의 X가 순서정보를 가진 단어(문장)들로 이루어져 있고, y 또한 단어들의 시리즈(문장)로 이루어진 경우에 사용되며 번역을 하는 모델에 주로 사용된다. 아래는 BLUE의 3가지 요소이다.

-n-gram을 통한 순서쌍들이 얼마나 겹치는지(precision)

-문장 길이에 대한 overfitting 보정(Brevity Penalty)

-같은 단어가 연속적으로 나올 때 overfitting 되는 것을 보정(Clipping)

$$BLEU = min(1, \frac{output\ length(예측\ 문장)}{reference\ length(실제\ 문장)})(\prod_{i=1}^{4} precision_i)^{\frac{1}{4}}$$

보통 BLUE4를 활용하여 n-gram길이가 4까지 계산한다. 3단어 미만 문장은 BLUE 0점이고 레퍼런스 번역 대비 번역의 정확성을 비교하는 지표이다.

VI. Research example

1. Task 정의 (예시 : 요약)
2. Dataset 정의

   -academic dataset을 찾는다. 그들은 이미 baseline이 짜여있다.

   -자신만의 data를 정의한다. 어렵고 새로운 baseline이 필요하다. (트위터, 블로그, 뉴스 등)

3. Dataset 정제(preprocessing) : test set을 미리 분리해둔다.
4. 평가 metric을 정한다. 온라인에서 활용 가능한 것을 찾고 요약의 경우 Rouge(Recall-Oriented Understudy for Gisting Evaluation) 같은 것도 쓸 수 있다.
5. Baseline을 정한다. 가장 간단한 모델부터 실행한다. 너무 잘나오면 문제가 쉬웠던 것이므로 다시한다.
6. 이미 존재하는 neural net model를 실행한다.
7. 항상 final test data를 제외하고는 data와 가깝게 한다. Dataset을 시각화하고 summary statistics를 모으며 에러를 본다. 다른 하이퍼파라미터들이 수행에 어떤 영향을 미치는지 분석한다.
8. 다양한 시도를 한다. 고정된 window neural model, recurrent neural network, recursive neural network, convolutional neural network, attention-based model 등

Ⅶ. Dataset

보통 dataset들은 train/dev/test으로 이루어져 있다. 개발이 끝난 후에만 test set을 사용한다. 만약 하나의 데이터 셋에만 이상 값들이 많이 존재하면 문제가 생길 수 있다.

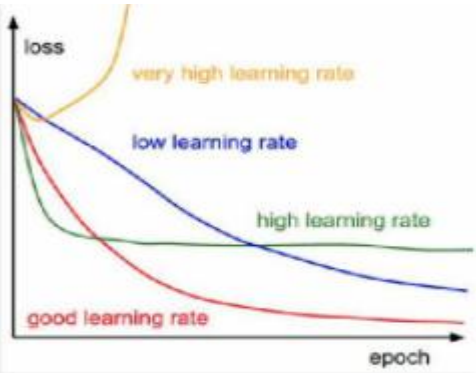
-Training set에서 훈련을 진행

-Tuning set에서 하이퍼파라마터 튜닝을 한다.

-Dev set에서 트레이닝이 잘 되었는지 확인한다. Dev set을 많이 쓰면 오버피팅이 발생할 수 있기 때문에 dev2 set도 두면 좋다.

-마지막에만 test set을 쓴다. 한두번만 쓰는게 좋다.

-모든 dataset들은 독립적이어야 한다.



트레이닝을 많이 하면 오버피팅이 일어날 수 있고 적절한 learning rate를 설정해주어야 한다.

Ⅷ. 연구방법론

1. 한단계씩 연구를 진행하는 것이 좋다. 처음에는 아주 간단한 모델로 시작해서 만약 잘 작동하면 간단하 모델에 많은 것들을 추가한다.

2. 데이터셋도 처음에는 아주 작은 데이터셋부터 시작하는 것이 좋다.(8개) 인위로 제작한 데이터도 좋다. 오류나 버그를 보기 쉽다. 작은 데이터로 시작하면 100%를 달성할 수 있도록 한다.

3. 데이터셋의 크기를 점점 키운다. 데이터셋을 키워도 100%에 가까운 정확도가 나오는 게 좋다. 안나오면 모델을 바꾼다. Traiing set오버피팅은 DL에서는 괜찮다.

4. 오류분석을 한다. Summary statistics, 모델의 출력값, 오류 분석을 한다. 하이퍼파라미터 튜닝이 성공적인 Nnets 모델에 제일 중요한 요소 중 하나이다.

5. 기타(RNN 트레이닝)

   -LSTM이나 GRU를 써본다.

   -orthogonal하게 recurrent matrices를 초기화한다.

   -다른 matrices들을 sensible scale로 만든다.

   -forget gate bias를 1로 둔다.(default to remembering)

   -adaptive learningm rate를 사용한다.

   -clip the norm of the gradient(1~5가 적당한 threshold이다.)

   -dropout을 vertically하게 적용시키거나 Bayesian Dropout을 사용한다.

   -침착하게 기다린다.