팀 소개

kozistr

한기대 컴퓨터 공학부 3학년

김형찬 (https://github.com/kozistr, http://kozistr.tech) / (kozistr@gmail.com)

전략

Character-Level CNN 기반 모델만으로 최대 성능 뽑아 보기!

핵심 모델 및 시도

- Pre-processing
  - 기본으로 제공 된 kor_char_parser.py, dataset.py 그대로 사용
  - konlpy 나 gensim 같은 한국어 전처리 패키지나 한국어 pre-trained 모델을 포함한 포함된 custom docker를 만들지 못해서 다른 시도는 못해 봄
  - 아마 지식인, 영화 모두 w2c 이나 음절 단위로 pre-processing을 거치면 기존 음소 단위 전처리 보다 성능 향상 가능 할 거 같다
- Models
  - CharCNN, CharCNN-RNN, (CharRNN, ...)
  - TextCNN, VDCNN, PD? CNN, ... (깊은 모델 별로...)
  - Simemse?
  - CharCNN + Highway Net + LSTM (not good...)
  - Simple CNN with B-Residual-Blocks (not good...)
- Futher...
  - Model Ensemble 하기

- 좀 큰(일반화된)? 한국어 w2c pre-trained model 사용해보기
- 기존에 사용한 CharCNN 을 w2c 로 전처리 방식으로 바꾸고 CNN 앞 단에 RNN(GRU, LSTM 등등...) 쌓아 보기
- 각 Conv-Layer 에 Self-Attention 추가해 보기
- 지식 인 같은 경우 2개의 문장으로 split 후 Cosine / Angular / Euclid Distance 시도 해 보기
- TF-IDF 등등... 해 보기
- Data Augmentation 해 보기

References

- https://medium.com/@zake7749/top-1-solution-to-toxic-comment-classification-challenge-ea28dbe75054 - Kaggle Toxic Comment

지식인

- Input 은 두 문장을 tab 기준으로 나누지 않고 그대로 한 문장으로 feed (아마 tab으로 구분 후 진행을 하면 더 좋은 성능 기대 가능)
- RNN 은 사용 하지 않았고 CharCNN 베이스 모델로 진행,
- 전 처리는 하나도 하지 않았고 max_seq_length 도 기본 값인 251 이다.
- Dropout 을 regularize를 위해 사용함 (각 conv/dense layer 뒤에 dropout 추가, rate : 0.7) (0.7 이후부터는 퍼포먼스가 떨어짐). BN 은 오히려 좋지 못했다.
- Weight Initializer 는 HE Initilizer를 사용, Reguliarizer 는 l2 사용 (퍼포먼스에 큰 영향을 안 줌)
- Embeddings 은 100 ~ 600 시도, 378 ~ 400 이 최적이라고 판단. (378 사용)
- CharCNN 커널 사이즈는 1 ~ 10 사이에서 시도 해 보면 좋음. 결론으로는 10, 9, 7, 5, 3 이 좋은 결과가 나온다는 걸 알아냄. Word Vector 라면 1 ~ 5 사이를 시도 해 보면 좋을 것 같다.
- Conv1d-ThresholdReLU-DropOut-kMaxPool1d 이 하나의 block. Pooling 방식은 max-pool, topk-max-pool, avg-pool, 짬뽕하기 다 해보았는데, max-pooling 하나 쓰는게 제일 좋은 결과가 나옴. (top-k max-pooling 사용, k=3)
- 마지막은 다 concat-flatten 후 dense layer 2개에 통과 시켜서 bce 함.
- Batch Size 는 64 / 128 시도. 본선에서는 128.
- Embedding_lookup 다음에 SpatialDropout1D를 적용 (퍼포먼스 향상 good, rate 는 물론 0.7)
- Adam Optimizer 에 gradident 가 exploision 하는 걸 방지하기 위해 clip_grad_norm 을 5 로 설정함 (5 or 7)
- 모델이 70~80 epoch 에서 거의 converge 했고, 학습을 시킬 수록 성능은 유지 또는 향상. (over-fitting 문제는 거의 발생하지 않았음).
- 해당 모델로 예선1, 2차 본선에 사용. 예선1,2 차에는 아마 9등, 15등으로 상대적으로 본선(4등)보다는 좋지 못한? 결과를 보임. 아마 일반화 된 데이터 셋에 강한 모델인 것 같다
- Back-End 는 tensorflow + keras(tf.contrib 에 있는 거)을 사용

영화 평점

- 이것도 전 처리를 하나도 하지 않았다.
- 모델도 물론 CharCNN 기반으로 제작을 하였다.
- 같은 입력에 다른 random 값으로? Embeddings 을 3개 생성해서 했다.
- Conv Kernel Size 는 대신에 3, 5, 7 로 나름 tricky 하게 생각 해 보니 결과가 괜춘했다! 예선 때는 3개 이상 conv 사용하면 오히려 퍼포먼스가 좋지 못했다.
- Weight Initilizer 는 xavier를 사용했다.
- LR decay 는 사용하지 않았고, 영화 평점 같은 경우엔 LR 에 엄청 민감해서 LR 을 2e-4 이하로 설정을 해 주었다.
- Optimizer 는 Adam 을 사용하였고 fine-tuning 시에는 SGD 1e-5 로 진행을 하였다.
- 이 주제 경우엔 Conv Block 마다 Drop Out을 넣어보니 더 느리고 결과도 안 좋아서 뺐다. 마지막 Dense Layer 에만 Drop Out 이 붙어있다. (이 경우엔 0.7 도 안 좋고 0.6 이 제일 좋았다)
- 본선 때 매번 epoch를 짧게 줘서 fork나 resume 이 안되는 문제가 생겨서 어느 epoch 쯤에서 converge 하는지는 확인을 못해봤지만 epoch 20 쯤 에서 최고 test result 가 나오는 것 같다.
- 이번에 예제 코드가 pytorch 이기도 하고 한번도 pytoch를 안 써봐서 공부겸사겸사 해서 Back-End 는 그대로 pytorch를 사용 했다.

후기

- 전 처리를 하나도 하지 않아서, 만약 전 처리를 조금이라도 했으면 결과가 어떻게 나왔을 지가 궁금하다! (예를 들어 지식인에서 간단히 tab으로 문장 구분 등...)
- 만들었던 모델이 나름 over-fitting에 강한 이유가 DropOut 이외에 다른 요인이 뭐가 있는지도 궁금하다.
- 영화같은 경우는 실제로 데이터가 1, 10점에 많이 분포 되어 있어서 그냥 sigmoid 에 넣고 돌려 버린 게 아쉽다.
- 본선 때 resume 이나 fork 가 작동을 제대로 안한 부분이 조금 아쉽다 ㅠ (휴먼에러 일 수도 ㅠㅠ)