Lecture 11. ConvNets for NLP

Ⅰ. Intro to CNNs

기존 RNN 계열의 모델은 the, of,... 등의 필요 없는 단어들을 포함하고 마지막 단어(벡터)의 영향을 많이 받는 문제가 있었다.(softmax를 맨 마지막에만 계산함.) 이를 해결하고자 CNN의 개념을 text에도 도입하였다.

-Text CNN의 필터는 텍스트이 지역적인 정보(단어의 등장순서/문맥정보) 보존

-문법적으로 맞는지 확인 불가

-언어학적으로 맞는 것 같지 않음

1. 1D Convolution 계산

Convolution 계산 -> pooling 계산 -> stride, local max pooling, dilation 계산

-dilated convolution : 필터 내부에 zero padding을 추가하여 강제로 receptive field를 늘리는 방법. Pooling을 수행하지 않고도 receptive field를 크게 가져갈 수 있음.

-필터 크기를 조정해서 n-gram의 다양한 feature를 만들 수 있음.

-다양하고 많은 filter를 사용해서 output의 차원을 늘릴수록, feature을 많이 사용할수록, 기존 문장에 대한 정보량이 커진다.

-pooling : max pooling, average pooling, k-max pooling 등등. NLP에서는 max pooling 선호하는데 정보가 모든 token에 골고루 있는 것이 아니라 sparse하게 있기 때문이다.

Ⅱ. Simple CNN for sentence classification

Convolution 연산에서는 각 단어들의 matrix 구조가 아니라 단어 벡터를 단순히 concat해서 연산.

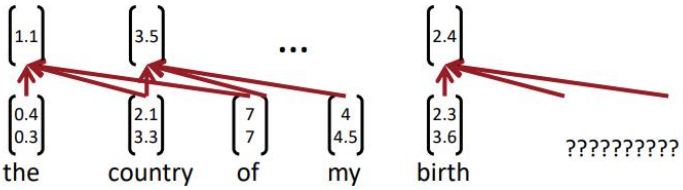
n개의 단어로 이루어진 문장이고 h 크기의 window를 가질 때의 결과는 아래와 같다.

Single layer CNN

- Filter **w** is applied to all possible windows (concatenated vectors)
- To compute feature (one *channel*) for CNN layer:

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$

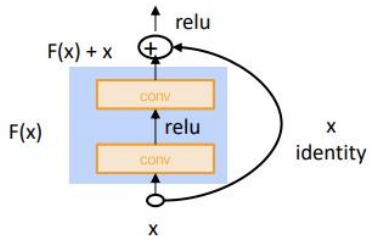
- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$
- All possible windows of length *h*: $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$
- Result is a feature map: $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



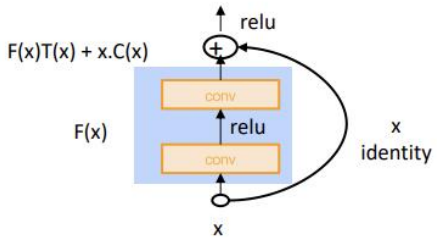
Pooling 연산은 feature map 결과인 c를 max pooling한다. Max pooling함으로써 filter들의 변화와 다양한 문장 길이에 대해 대처 가능하다. input으로 임베딩 된 벡터를 넣고 기존의 pre-trained word vectors(word2vec of GloVe)로 초기화시켰고, fine-tuning 한 것(static)과 fine-tuning하지 않은 것(non-static)을 둘 다 사용하였다. CNN layer를 통과시킨 후 pooling한 최종 벡터(z)를 soft max를 통해 classification을 수행하였다. Regularization으로 dropout을 사용하였다.

Ⅲ. CNN potpourri

1. Gated units used vertically



Residual block
(He et al. ECCV 2016)



Highway block
(Srivistava et al. NeurIPS 2015)

Note: pad **x** for conv so same size when add them

ResNet에서 나온 Residual block과 Highway block 모두 shortcut Connection의 개념이다. 즉, 같은 역할로 x에 대한 정보를 얼마나 넘겨줄지의 개념이다. Residual block에서는 F(x)+x를 사용하고 Highway block은 F(x)T(x)+x·C(x)를 사용하는 점만 차이가 있다.(T는 transform gate, C는 carry gate)

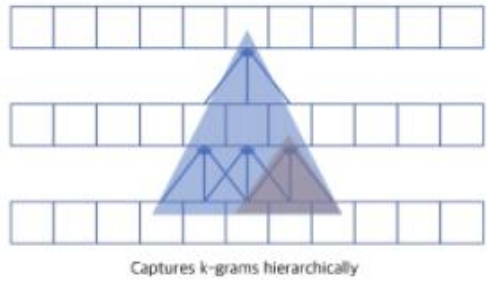
2. Batch Normalization & 1 x 1 Convolutions

   보통 CNN에 쓰이고 Convolution 연산의 output을 batch 별로 평균 0, 분산 1로 정규화 시킨다. Z-transfrom 같이 모델의 안정적인 학습을 가능하게 한다. Batch 별 업데이트는 변동이 클 수 있기 때문에 많이 사용하면 안된다. Parameter initialization에 덜 민감하고 learning rate에 대한 tuning이 쉬워진다.

   1 x 1 Convolutions은 1D convolution으로 kernel size는 1이다. 적은 파라미터로 channel 축소가 가능하고(작은 필터 사용) fully connected layer의 input으로 사용 가능하다.

3. NLP에서의 CNN

   - Neural network가 feature를 포착하는 순서는 tokens->multi-word->expressions ->phrases->sentences 순이다.(Capures k-grams hierarchically)

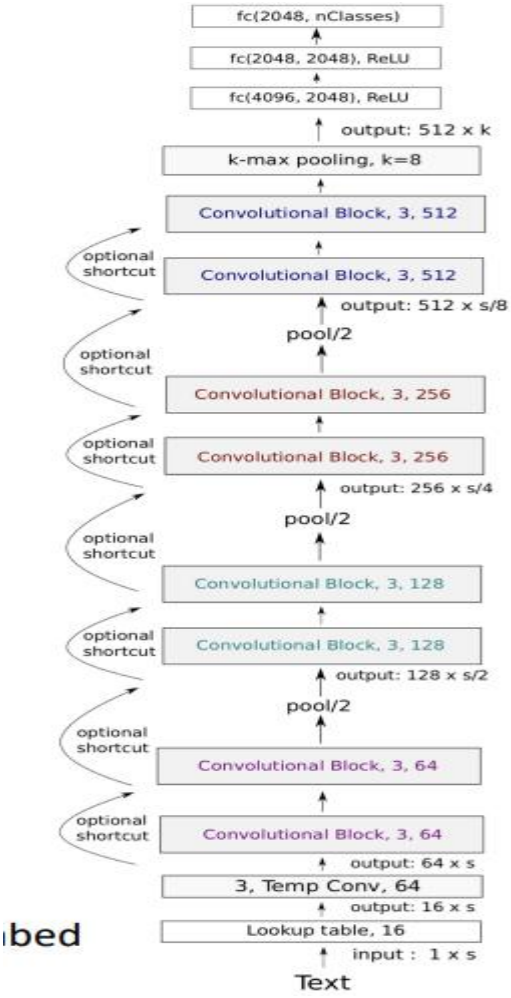


   - CNN의 최대 장점은 구현이 굉장히 잘 되어있어서 사용하기 편한다는 것이다.
   - CNN으로만 layer를 쌓고자 한다면 첫 단어와 마지막 단어를 잘 포착하기 위해서 많은 convolution layer가 필요하다.
   - 너무 전체를 보는 Realation Network와 너무 local하게 포착하는 CNN의 장점을 합쳐 나오게 된 것이 Self Attention이다.

4. 모델 분류

   - Bag of Vectors : CBoW, Bag of words 모델들...
   - Window Model : Relation Network(Skip Bigrams)
   - CNNs
   - RNNs
   - 어떻게 문장을 표현하는 지(average하는 CBoW 모델을 제외하면 나머지 개념들은 stack한다. 각 token 별로 토큰 위치에 맞는 lookup을 보고 벡터들을 뽑아낸다. 지금까지 배운 token 표현 방법들은 혼합해서 같이 쓰인다. Task가 classification이면 궁극적으로 average가 가장 일반적이다.)

VI. Deep CNN for sentence Classification : Conneau et al



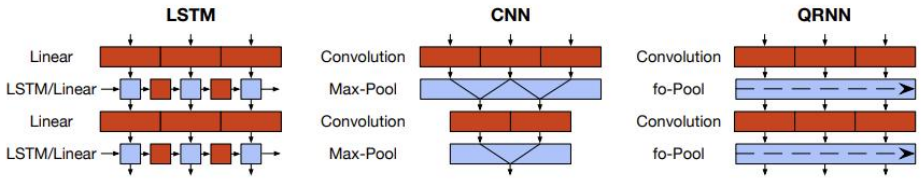
이미지나 다른 분야에서는 layer를 정말 deep하게 쌓는다. VD-CNN은 text classification을 위해 깊은 CNN 네트워크를 사용한 논문이다. 이미지 분야의 VGG-Net이나 ResNet을 닮았지만 그렇게 깊지는 않다.

Ⅴ. Quasi-Recurrent Neural Networks

RNN은 deep NLP의 매우 표준적인 building block이지만 병렬성이 안좋고 매우 느리다. 그래서 RNNs과 CNNs의 가장 좋고 가장 병렬가능한 파트를 가져오자는 아이디어가 나왔고 그것이 Quasi-Recurrent Neural Networks이다.

LSTMs보다 종종 더 낫고 더 빠르다. 하지만 LSTMs과 마찬가지로 character-level LMs에서는 작동하지 않았고 더 긴 dependency에서 modeling에 문제가 있다. 또한 LSTM만큼 좋은 성능을 가지려면 종종 더 깊은 network가 필요하다. 깊을 때도 여전히 빠르므로 true recurrence에 대한 대체로 깊이를 사용하면 효과적이다.

- Tries to combine the best of both model families



- Convolutions for parallelism across time:

$$\begin{aligned}\mathbf{z}_t &= \tanh(\mathbf{W}_z^1 \mathbf{x}_{t-1} + \mathbf{W}_z^2 \mathbf{x}_t) \\ \mathbf{f}_t &= \sigma(\mathbf{W}_f^1 \mathbf{x}_{t-1} + \mathbf{W}_f^2 \mathbf{x}_t) \\ \mathbf{o}_t &= \sigma(\mathbf{W}_o^1 \mathbf{x}_{t-1} + \mathbf{W}_o^2 \mathbf{x}_t).\end{aligned} \quad \rightarrow \quad \begin{aligned}\mathbf{Z} &= \tanh(\mathbf{W}_z * \mathbf{X}) \\ \mathbf{F} &= \sigma(\mathbf{W}_f * \mathbf{X}) \\ \mathbf{O} &= \sigma(\mathbf{W}_o * \mathbf{X}),\end{aligned}$$

Convolutions compute candidate, forget & output gates

- Element-wise gated pseudo-recurrence for parallelism across channels is **done in pooling layer**: $\mathbf{h}_t = \mathbf{f}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t,$