YBIGTA 10기 노혜미

Concrete RNN Architectures

1. Simple RNN (SRNN)
2. Long Short-Term Memory (LSTM)
3. Gated Recurrent Unit (GRU).

1. Simple RNN

- 다음과 같은 형태를 취한다.

$$s_i = R_{SRNN}(s_{i-1}, x_i) = g(x_i W^X + s_{i-1} W^S + b)$$

$$y_i = O_{SRNN}(s_i) = s_i$$

- 즉, i에서의 state는 i에서의 input과 i-1에서의 state의 선형 결합*이다. 그리고 tanh 라든지 ReLU와 같은 non-linear activation에 넘겨진다.

*벡터들을 숫자배하고 벡터끼리의 덧셈을 통해 조합해 새로운 벡터를 얻는 연산이다. 예를 들면, $(7, 12)^T$ = $2(2, 3)^T + 3$이다. T는 행과 열을 바꾸는 것이다. 예에서는 1행 2열을 트랜스포즈 했으니 2행 1열짜리 벡터가 될 것이다.

- i에서의 output은 그 위치에서의 hidden state와 같다.
- simple RNN 이렇게 간단하게 보여도, 언어 모델링 뿐만 아니라 sequence tagging에서 좋은 결과를 낸다.

2. LSTM

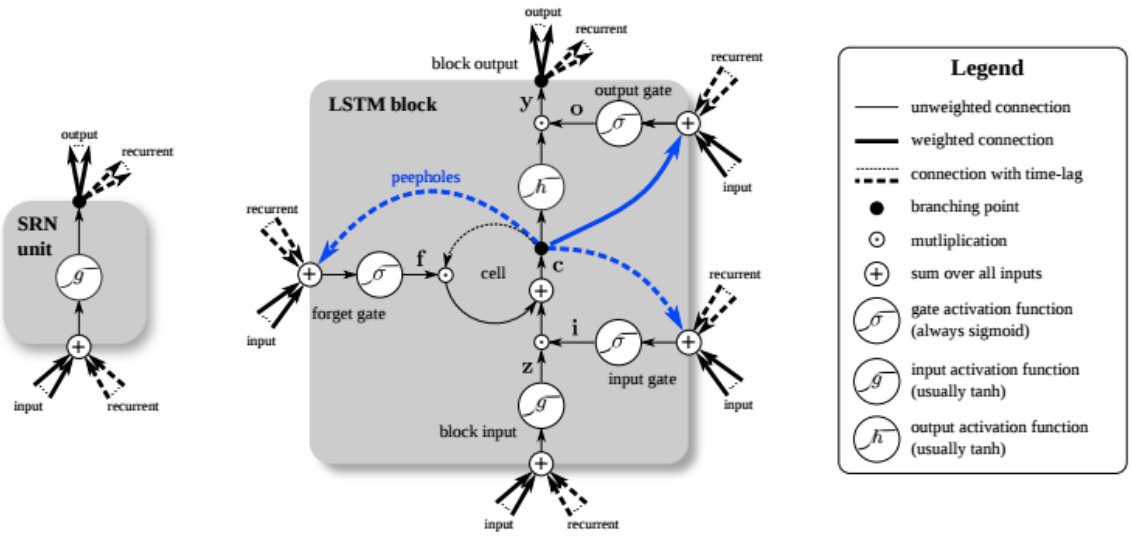


Figure 1. Detailed schematic of the Simple Recurrent Network (SRN) unit (left) and a Long Short-Term Memory block (right) as used in the hidden layers of a recurrent neural network.

이미지 출처: https://deeplearning4j.org/kr/lstm#long

- S-RNN은 vanishing gradient 문제때문에 효과적으로 학습시키기가 어렵다. Error signal인 gradient가 sequence를 이동할수록 back-prop 과정에서 빠르게 사라져서 초기 input signal까지 닿지 않는다. 그리고 이것은 S-RNN이 긴 문장에서 단어들의 밀접한 관계를 잡아내지 못한다.
- 이러한 문제를 해결하기 위해 등장한 것이 바로 LSTM이다.
- 주요한 아이디어는 state의 표현 또한 시간이 흘러도 gradient를 갖고 있을 수 있는 "memory cells"로 나타내는 것이다.
- memory cell로의 접근은 *gating components*에 의해 통제 받는다. gating component는 logical gates와 비슷하게 작용하는 완만한 모양을 가진 함수이다.
- 각각의 input state에서, gate는 얼마 만큼의 input이 memory cell에 쓰여져야 하고, 잊혀져야 하는지를 결정한다.
- gate, g는 0에서 1사이의 값을 가지는 n차원짜리 vector이다. 그리고 g는 또다른 n차원 vector인 v와 element-wise로 곱*해 진다. 그 결과는 또 다른 vector와 더해진다.

  *똑같은 행과 열을 가진 원소끼리 곱하는 것.
- g의 값은 sigmoid 함수를 써서 0이나 1에 가깝게 설정되어 있다.
- v와 g가 곱해질때, g의 vector에서 1에 가까운 element와 곱해지면 해당 v의 element는 넘겨지게 되고, 0에 가까운 element와 곱해지면 차단된다.
- 수학적으로 LSTM의 구조는 다음과 같이 정의된다.

$$s_j = R_{LSTM}(s_{j-1}, x_j) = [c_j; h_j]$$

$$c_j = c_{j-1} \circledcirc f + g \circledcirc i$$

$$h_j = tanh(c_j) \circledcirc o$$

$$i = \sigma(x_j W^{xi} + h_{j-1} W^{hi})$$

$$f = \sigma(x_j W^{xf} + h_{j-1} W^{hf})$$

$$o = \sigma(x_j W^{xo} + h_{j-1} W^{ho})$$

$$y_j = O_{LSTM}(s_j) = h_j$$

*◎는 component-wise product를 나타낸다.

- j번째에서의 state는 c_j와 h_j 두 개의 vector로 구성된다.
- c_j는 meomory component이고 h_j는 output 혹은 state component이다.
- 3개의 gate가 있는데 **i**nput, **f**orget, **o**utput의 세 글자를 따서 i, f, o gate이다.
- gate의 값들은 현재 input인 x_j와 이전의 state인 h_{j-1}의 선형 결합으로 계산된다. 그리고 sigmoid activation 함수를 거치게 된다.
- candidate g의 update는 x_j와 h_{j-1}의 선형 결합으로 계산된 뒤, tanh activation 함수를 거친다.
- c_j는 그 다음에 update된다. forget gate는 $c_{j-1} \circledcirc f$를 통해 이전 memory를 얼마나 저장할지 결정하고, input gate는 $g \circledcirc i$를 통해 얼마나 update할지를 결정한다.

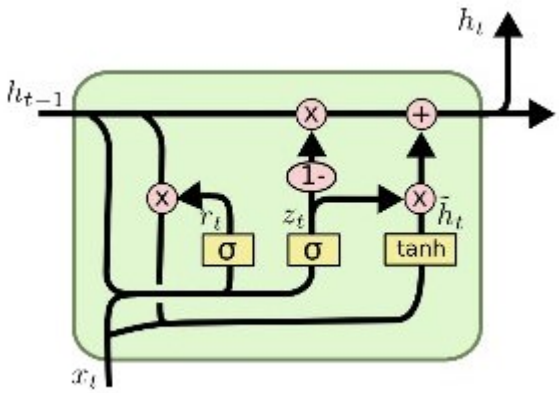
- 결국, h_i 혹은 y_j값은 non-linearity한 tanh를 거치고 output gate에 의해 통제받는 memory c_j의 내용에 따라서 결정된다.
- gating mechanism은 c_j와 관련된 gradient가 매우 긴 시간 동안 높게 유지하도록 해준다.

Practical Considerations

- LSTM networks를 훈련시킬 때, Jozefowicz(2015)는 forget gate의 bias를 1에 가깝게 초기화 시킬 것을 강추한다.
- Zaremba(2014)는 dropout을 적용하려면 recurrent connection이 아닌 곳, 즉 sequence position 사이가 아니라 layer 사이에 적용해야된다고 한다.

*누군지는 모르겠지만 논문까지 썼으니까 유명할 것 같다...

3. GRU



이미지 출처: https://www.slideshare.net/xavigiro/recurrent-neural-networks-1-d2l2-deep-learning-for-speech-and-language-upc-2017

- LSTM 구조는 매우 효과적이지만 위에서 보듯이 상당히 복잡하다...그래서 분석하기도 힘들고 계산하는데도 많은 비용이 든다.
- GRU는 Cho(조경현) 교수가 LSTM의 대안으로써 만든 것이다. (신기신기)
- GRU 역시 gating mechanism에 근거했지만, gate가 상당히 적고 분리된 memory component가 없다.

$$s_j = R_{GRU}(s_{j-1}, x_j) = (1 - z) \circledcirc s_{j-1} + z \circledcirc h$$

$$z = \sigma(x_j W^{xz} + h_{j-1} W^{hz})$$

$$r = \sigma(x_j W^{xr} + h_{j-1} W^{hr})$$

$$h = tanh(x_j W^{xh} + (h_{j-1} \circledcirc r) W^{hg})$$

$$y_j = O_{LSTM}(s_j) = s_j$$

- gate (r)은 이전 state s_{j-1}으로의 접근을 통제하고 proposed update h를 계산하는데 사용된다. output y_j이기도 한 update된 state s_j는 이전 state s_{j-1}과 proposal h의 interpolation(보간)*을 기반으로 결정된다. interpolation의 proportion(비율?)은 gate z를 이용해 통제된다.

*어떤 함수에서 f(2.5)의 값을 알고싶다. 이때 f(2) = 1이고 f(3)= 3이라면 우리는 f(2.5)가 이 두 개 값의 중간이라고 추정할 수 있다. 이와 같은 방법을 interploation이라고 한다.

- GRU는 언어 모델링과 기계 번역에서 효과적으로 보인다. 하지만 GRU, LSTM, 가능한 대안적 RNN 구조 중에서 뭐가 더 나은지는 아직 잘 모른다.

Other variants

- Mikolov(2014)는 matrix 곱인 $s_{i-1}W^s$가 state vector인 s_i가 각각의 time step에서 큰 변화를 겪게 만든다고 한다. 그리고 이것은 긴 시간동안 정보를 기억하는 것을 방해한다.
- 그래서 state vector s_i를 느리게 변하는 component c_i("context units")과 빠르게 변하는 component인 h_i로 분리할 것을 제안한다.
- c_j는 input과 이전 component의 선형 보간법*에 따라 update된다. 이러한 update는 c_i가 이전의 input을 축적할 수 있게 해준다.

$$c_i = (1-\alpha)x_iW^{x1} + \alpha c_{i-1}$$

(α는 (0,1)에 속한다.)

*만약 어떤 함수 f(2.5)의 값을 알고 싶다고 하자. 이때 f(2)=1이고 f(3)=3이라면 우리는 f(2.5)=2이라고 추측할 수 있다. 이와 같이 양끝점 값이 주어졌을 때, 사이에 위치한 값을 추정하는 것을 선형 보간법이라고 한다.

- h_i의 update는 simple RNN의 update와 비슷하지만, c_i또한 계산에서 고려하는 게 차이이다.

$$h_i = \alpha(x_iW^{x2} + h_{i-1}W^h + c_iW^c)$$

- 아웃풋 y_i는 c_i와 h_i의 concatenation이 된다.

$$y_i = [c_i; h_i]$$

- 하지만 성능은 더 좋아질지라도 안 그래도 복잡한 LSTM이 더 복잡해 졌다는 게 흠이라면 흠이다.
- Le(2015)는 훨씬 더 간단한 접근을 제시했다. S-RNN의 activation function을 ReLU로 두고, bias b를 0으로 matrix W^s를 단위행렬로 초기화하라고 했다.
- 이러한 결과는 학습되지 않은 RNN이 현재 state에 이전 state를 카피하고 현재 input x_i의 영향을 더하고 음의 값은 0으로 만든다고 한다. 초기의 경향을 카피하는 쪽으로 설정한 후에는, 학습 과정이 W^s를 자유롭게 변하도록 만든다.
- Le는 이렇게 S-RNN을 수정하는 것이 언어 모델링을 포함한 몇몇 task에서 같은 parameter 개수임에도 S-RNN을 LSTM과 견줄 정도가 된다고 한다.

Reference

- Yoav Goldberg(2015), A Primer on Neural Network Models for Natural Language Processing - http://u.cs.biu.ac.il/~yogo/nnlp.pdf