Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

Choe Haein

2022-08-21

서울시립대학교 통계학과
석사과정
최적화 연구실

LSTM

Long-Short-Term Memory

- 전통적인 RNN 은 오래된 기억의 영향력이 사라지는 문제가 있다(Vanishing Gradient)
- LSTM 은 메모리 셀이라는 구조를 추가하여 보존해야 할 기억의 양은 결정함
- 메모리 셀을 계산하기 위하여 LSTM 에는

  1. `Input Gate`
  2. `Forget Gate`
  3. `Candidate`
  4. `Output Gate`

  라는 구조가 존재한다.

Long-Short-Term Memory

Input Gate

$$i_t = \sigma(W_{xi}x_t + Uh_{t-1} + V_i c_{t-1})$$

Candidate

$$\tilde{c_t} = tanh(W_c x_t + U_c h_{t-1})$$

Forget Gate

$$f_t = \sigma(W_{xf}x_t + U_f h_{t-1} + Vic_{t-1})$$

Memory Cell

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c_t}$$

Output Gate

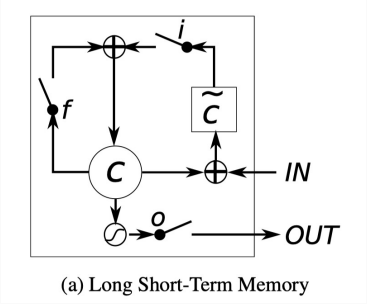
$$o_t = \sigma(Wx_t + U_o ht - 1 + V_o c_t)$$

Hidden State at t

$$h_t = o_t \circ tanh(c_t)$$

Long-Short-Term Memory

LSTM 의 구조도는 그림과 같다



(a) Long Short-Term Memory

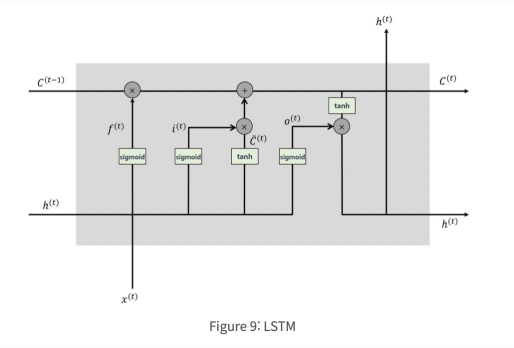


Figure 9: LSTM

(https://yjjo.tistory.com/17?category=881892)

GRU

Gated Recurrent Network Unit

- 본 논문에서는 LSTM와 비슷한 구조이며 더 간소화된 RNN 프레임워크가 제시됨
- LSTM 에서는 3개의 게이트가 존재하지만 GRU 에서는

    1. Reset Gate
    2. Update Gate

    두 개의 게이트만이 존재하여 계산량이 감소하였다.

Gated Recurrent Network Unit

Reset Gate

$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$

Update Gate

$$z_t = \sigma(W_x z x_t + U_z h_{t-1})$$

Candidate

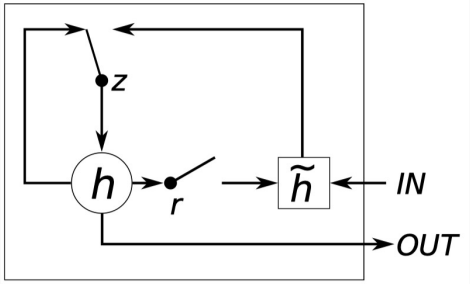
$$\tilde{h_t} = tanh(U(r_t \circ h_{t-1}) + W x_t)$$

Hidden state at t

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h_t}$$

Gated Recurrent Network Unit

GRu의 구조는 그림과 같다



(b) Gated Recurrent Unit

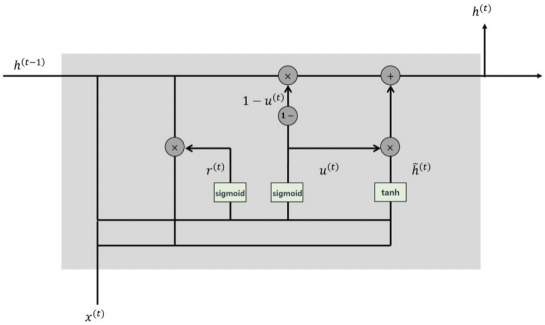


Figure 6: GRU

(https://yjjo.tistory.com/18?category=881892)

Seq2Seq

Seq2Seq

Unit	# of Units	# of Parameters
Polyphonic music modeling		
LSTM	36	$\approx 19.8 \times 10^3$
GRU	46	$\approx 20.2 \times 10^3$
tanh	100	$\approx 20.1 \times 10^3$
Speech signal modeling		
LSTM	195	$\approx 169.1 \times 10^3$
GRU	227	$\approx 168.9 \times 10^3$
tanh	400	$\approx 168.4 \times 10^3$

Table 1: The sizes of the models tested in the experiments.

			tanh	GRU	LSTM
Music Datasets	Nottingham	train	3.22	2.79	3.08
		test	**3.13**	3.23	3.20
	JSB Chorales	train	8.82	6.94	8.15
		test	9.10	**8.54**	8.67
	MuseData	train	5.64	5.06	5.18
		test	6.23	**5.99**	6.23
	Piano-midi	train	5.64	4.93	6.49
		test	9.03	**8.82**	9.03
Ubisoft Datasets	Ubisoft dataset A	train	6.29	2.31	1.44
		test	6.44	3.59	**2.70**
	Ubisoft dataset B	train	7.61	0.38	0.80
		test	7.62	**0.88**	1.26

Table 2: The average negative log-probabilities of the training and test sets.

Implementation

Source

이 게시물을 참고하여 연습하였습니다

Practice

```
class Encoder(nn.Module):
    def __init__(self, input_dim, emb_dim, hid_dim, dropout):
        super().__init__()

        self.hid_dim = hid_dim

        self.embedding = nn.Embedding(input_dim, emb_dim) #no dropout as only one layer!

        self.rnn = nn.GRU(emb_dim, hid_dim)

        self.dropout = nn.Dropout(dropout)

    def forward(self, src):

        #src = [src len, batch size]

        embedded = self.dropout(self.embedding(src))

        #embedded = [src len, batch size, emb dim]

        outputs, hidden = self.rnn(embedded) #no cell state!

        #outputs = [src len, batch size, hid dim * n directions]
        #hidden = [n layers * n directions, batch size, hid dim]

        #outputs are always from the top hidden layer

        return hidden
```

인코더 클래스

클래스 맴버

인코더 매소드

Practice