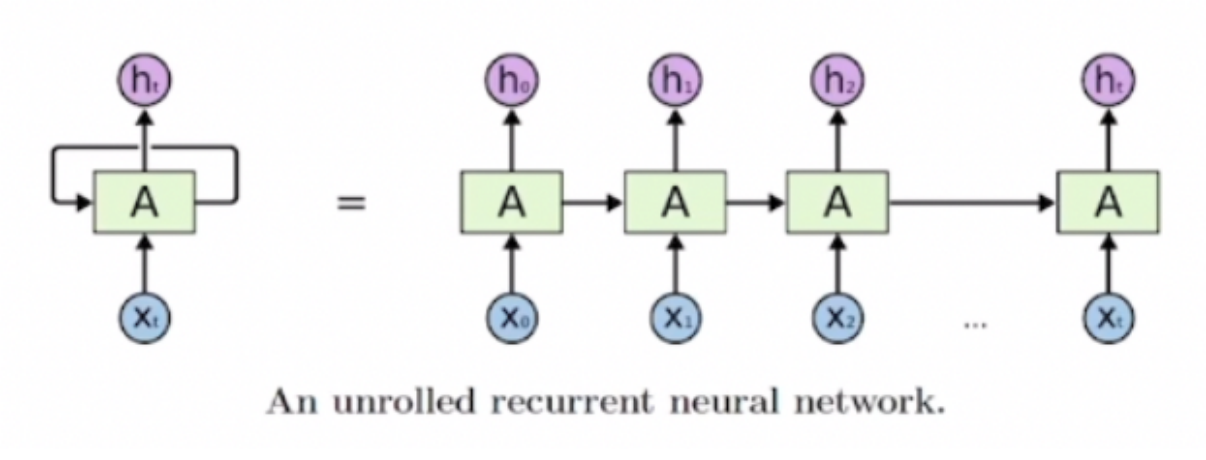
[자연어 처리의 모든 것] 2. 자연어 처리와 딥러닝

1. Recurrent Neural Network (RNN)

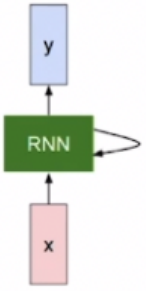
RNN



An unrolled recurrent neural network.

- 시퀀스 데이터가 입력, 출력 벡터
- 이전 스텝까지의 정보(x_t, h_(t-1))를 기반으로 현재 타임스텝 예측값(h_t)을 산출하는 구조의 딥러닝 모델
- 매 타임스텝마다 동일한 파라미터를 가진 모듈(A)을 반복적으로 사용
  = 재귀적인 호출
  -> Recurrent Neural Network
  RNN 계산 방법

$$h_t = f_W(h_{t-1}, x_t)$$



- t: 현재 타임스텝
- w: weight

- x_t: input vector at some time step
- h_t: new hidden-state vector
- f_W: RNN function with parameters W
- y_t: output vector at time step t
    - 매 타입스텝마다 구해야 할 수도 있고 마지막에만 구하는 경우도 있음
- 파라미터 W는 모든 타임스텝에서 동일한 값을 공유

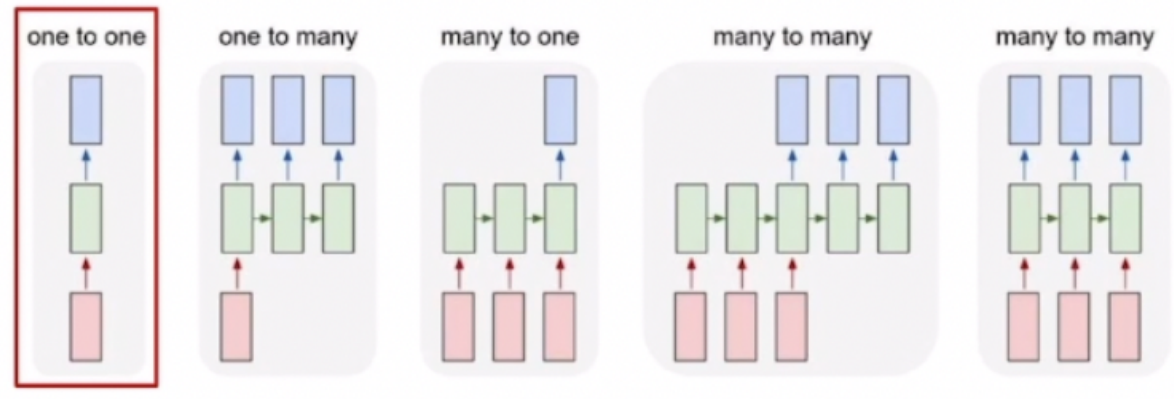
$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

- W_{xh} : $W_{x_t->h_t}$ 로 변환
- W_{hh} : $W_{h_{t-1}->h_t}$ 로 변환
- W_{hy} : $W_{h_t->y_t}$ 로 변환

Type of RNNs



1. one-to-one

- 입출력 데이터의 타임스텝이 1개라 시퀀스가 아닌 일반적인 경우의 모델 구조
- [키, 몸무게, 나이] 벡터로 저혈압/고혈압인지 분류하는 형태의 테스크

1. one-to-many

- 입력은 1개의 타임스텝, 출력은 여러 개의 타임스텝

- 이미지 캡션, 입력: 이미지 -> 출력: 여러 설명

1. many-to-one

- 입력 시퀀스를 입력으로 받은 후 최종값을 마지막 스텝에서 출력
- 감성 분석과 같이 문장을 넣으면 긍정/부정 중 하나의 레이블로 분류하는 테스크

1. many-to-many

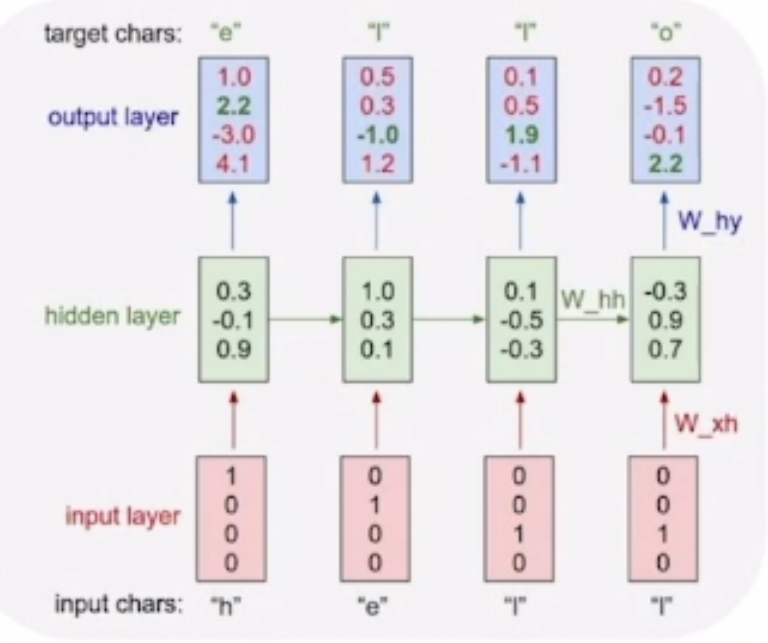
- 기계 번역
- 입력값을 끝까지 다 읽은 후 마지막 타임스텝에서 번역된 문장(예측값)을 출력해주는 테스크

1. many-to-many

- 딜레이가 존재하지 않고 입력할 때마다 예측값을 출력하는 task
- 단어별 품사 태깅 POS 테스크, 비디오 프레임별 분류 테스크

2. Character-level Language Model

character-level Language Model



- 언어 모델: 이전에 등장한 문자열을 기반으로 다음 단어를 예측하는 테스크
- 캐릭터 레벨 언어 모델(character-level Language Model): 문자 단위로 다음에 올 문자를 예측하는 언어 모델

- Example of training sequence "hello"
  - vocabulary: [h,e,l,o]
    -> one hot vector 형태의 input
- 맨 처음에 "h"가 주어지면 "e"를 예측하고, "e"가 주어지면 "l"을 예측하고, "l"이 주어지면 다음 "l"을 예측하도록 hidden state가 학습되어야함
- hidden layer:

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t + b)$$

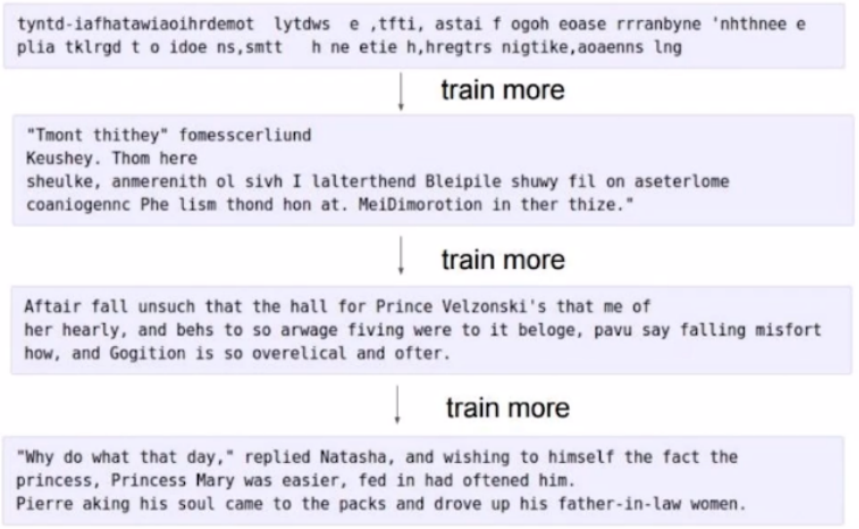
- output layer:

$$Logit = W_{hy}h_t + b$$

  - many to many task에 해당
  - 각 타임스텝별로 output layer를 통해 출력하는 차원 4(유니크한 문자의 개수) 벡터 - softmax layer를 통과시키면 one hot vector 형태의 출력값이 나옴
- ground truth vector와 가까워지도록 학습

다양한 언어모델의 예시

- 여러 단어, 문장으로 이루어진 문단 학습도 가능

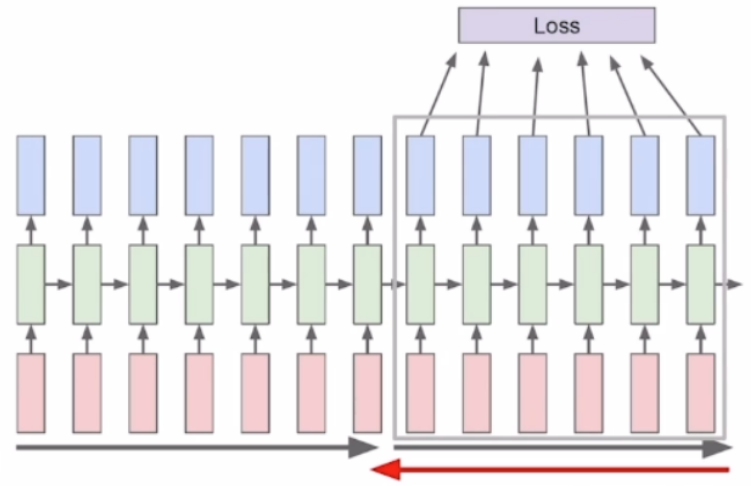


- 학습이 진행될수록 고품질의 문자열이 생성됨
- 인물별 대사
- Latex로 논문 작성

- C 언어

3. Backpropagation through time and Long-Term-Dependency

RNN 모델이 학습하는 방법 : Truncation , BPTT



- W_xh, W_hh, h_t, W_hy 학습 진행
- Truncation: 잘라서 제한된 길이의 시퀀스만으로 학습을 진행하는 것
- BPTT(Backpropagation through time): RNN에서 타임스텝마다 계산된 weight를 backward propagation을 통해 학습하는 방식

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *
        siginfo_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig) {
        if (current->notifier) {
            if (sigismember(current->notifier_mask, sig)) {
                if (!(current->notifier)(current->notifier_data)) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
        }
        collect_signal(sig, pending, info);
    }
    return sig;
}
```

- 특정 dimension의 hidden state를 빨강은 긍정, 파랑은 부정으로 시각화한 그림
- 조건문에 해당하는 부분이 빨강으로 나타냄 (하나의 cell)

Vanishing/Exploding Gradient Problem in RNN

- 앞에서 본 것은 오리지널 RNN보다 진보된 LSTM, GRU를 사용했을 때의 결과
  -> vanilla RNN은 많이 사용하지는 않음
- gradient가 전파되면서 기하급수적으로 소실되거나 증폭되면서 멀리까지 학습정보를 잘 전달하지 못하는 Long-Term-Dependency가 발생하기 때문

Toy Example

- $h_t = \tanh(w_{xh}x_t + w_{hh}h_{t-1} + b), t = 1,2,3$
- For $w_{hh} = 3, w_{xh} = 2, b = 1$

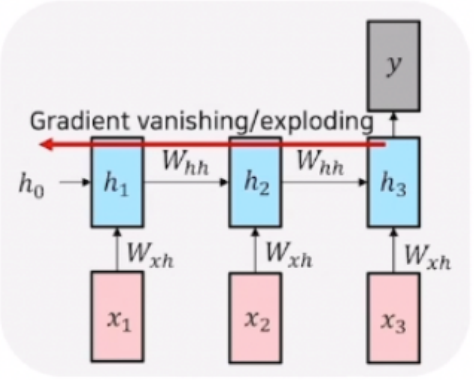
$h_3 = \tanh(2x_3 + 3h_2 + 1)$

$h_2 = \tanh(2x_2 + 3h_1 + 1))$

$h_1 = \tanh(2x_1 + 3h_0 + 1))$

...

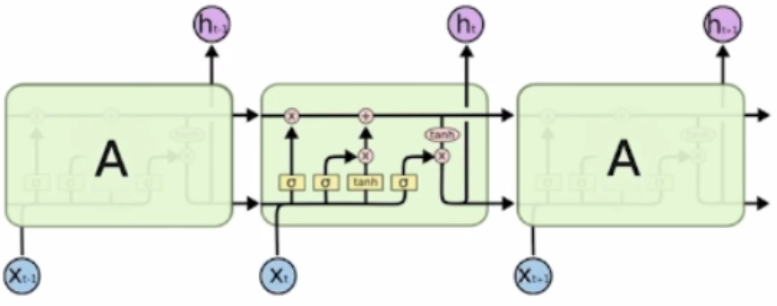
$h_3 = \tanh(2x_3 + 3\tanh(2x_2 + 3h_1 + 1) + 1)$



- back propagation 과정에서 h3에 대한 h1 편미분 값을 구할 때, 접선의 기울기들을 곱해 3이 거듭 곱해지며 값이 증폭됨
  -> 모델 자체가 전체적으로 학습이 잘 되지 않는 현상 발생

4. Long Short-Term Memory (LSTM)

LSTM : Long Short-Term Memory

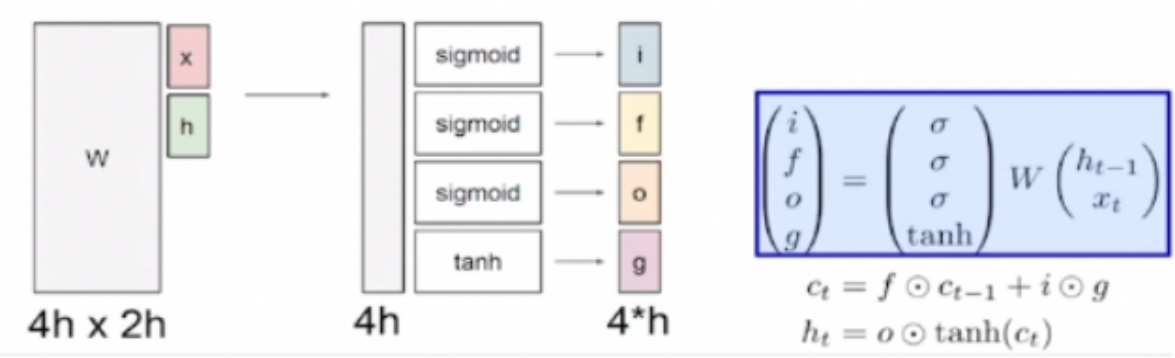


The repeating module in an LSTM contains four interacting layers.

- vanilla RNN을 완벽하게 대체 가능하고 더 좋은 성능을 보임

- gradient vanishing/exploding과 Long-Term-Dependency를 해결
- 중심 아이디어: 단기 기억으로 저장하여 이걸 때에 따라 꺼내 사용함으로 더 오래 기억할 수 있도록 개선
- $C_t, h_t = LSTM(x_t, C_{t-1}, h_{t-1})$
  - C_{t-1}: cell state vector
  - h_{t-1}: hidden state vector
- Cell state에는 핵심 정보들을 모두 담아두고, 필요할 때마다 Hidden state를 가공해 time step에 필요한 정보만 노출하는 형태로 정보 전파

LSTM의 연산 과정



- input으로 x_t, h_t-1이 들어오게 되고 이를 W에 곱해준 후 각각 sigmoid or tanh로 연산해줌
- 4개의 gate 존재
- i : Input gate이며, cell 에 쓸지말지를 결정하는 게이트
  - 들어오는 input에 대해서 마지막에 sigmoid를 거쳐 0-1 사이 값으로 표현해줌
  - 표현식 : sigmoid(W(xt, ht-1))
- f : Forget gate이며, 정보를 어느정도로 지울지를 sigmoid를 거쳐 0~1사이의 값으로 나타냄
  - 표현식 : sigmoid(W(xt, ht-1))
- o : Output gate이며, Cell 정보를 어느정도 hidden state에서 사용해야할 지를 sigmoid를 거쳐 0~1사이 값으로 나타냄
  - 표현식 : sigmoid(W(xt, ht-1))

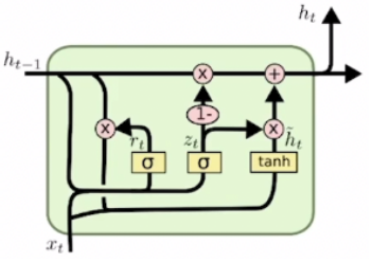
- g : Gate gate이며, 어느정도로 Cell state에 반영해야할 지를 tanh를 거쳐 -1~1 사이의 값으로 나타냄
    - 표현식 : tanh(W(xt, ht-1))

LSTM이 RNN과 다른점

- LSTEM은 각 time step마다 필요한 정보를 단기 기억으로 hidden state에 저장하여 관리되도록 학습함
- backpropagation 진행시 forget gate를 거친 값에 대해 필요로 하는 정보를 덧셈을 통해 연산하여 그레디언트 소실/증폭 문제를 방지

GRU : Gated Recurrent Unit

- $z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$
- $r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$
- $\tilde{h_t} = \tanh(W \cdot [r_t \cdot h_{t-1}, x_t])$
- $h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h_t}$
- c.f) $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C_t}$ in LSTM



- GRU: LSTM과 전체적인 동작원리는 유사하지만, Cell state, Hidden state를 일원화하여 경량화한 모델
- GRU에서 사용되는 h_{t-1}은 LSTM에서의 c_t와 비슷한 역할을 함
- forget gate 대신 1-input gate를 사용
- LSTM에 비해 계산량과 메모리 요구량을 줄이면서 성능도 더 좋음

RNN ,LSTM ,GRU 요약

- RNN은 다양한 길이의 시퀀스에 특화된 유연성을 가진 딥러닝 모델
- RNN에서는 gradient vanishing/exploding 문제가 있어 실제로 많이 사용되지는 않지만, LSTM, GRU 모델은 현재도 많이 사용되고 있음

해당글은 부스트코스의 [자연어 처리의 모든 것] 2. 자연어 처리와 딥러닝 강의를 듣고 작성한 글입니다.

velog 링크