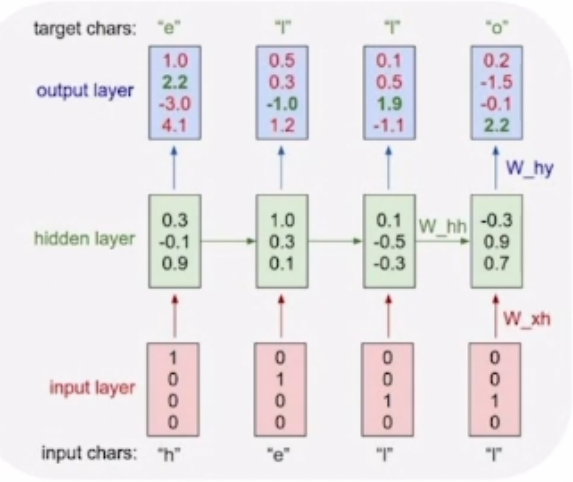
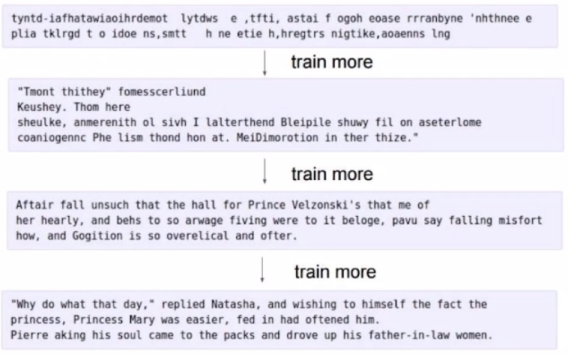
EURON 19주차



맨 처음에 "h"가 주어지면 "e"를 예측하고, "e"가 주어지면 "l"을 예측하고, "l"이 주어지면 다음 "l"을 예측하도록 hidden state가 학습



1. 더 많은 학습이 진행될수록 완전한 형태의 문장을 출력
2. 전 타임스텝까지의 주식값을 활용하여 다음날 주식값을 예측하는 형태로도 가능
3. 그 외 인물 별 대사, Latex로 쓰여진 논문, C 프로그래밍 언어와 같은 경우 해당

언어 모델: 이전에 등장한 문자열을 기반으로 다음 단어를 예측하는 태스크

- 그중에서도 캐릭터 레벨 언어 모델(character-level Language Model)은 문자 단위로 다음에 올 문자를 예측하는 언어 모델
- 이때 각 타임스텝별로 output layer를 통해 차원이 4(유니크한 문자의 개수) 벡터를 출력해주는데 이를 logit이라고 부른다. softmax layer를 통과시키면 원-핫 벡터 형태의 출력값이 나온다.

RNN 모델이 학습하는 방법 : Truncation , BPTT

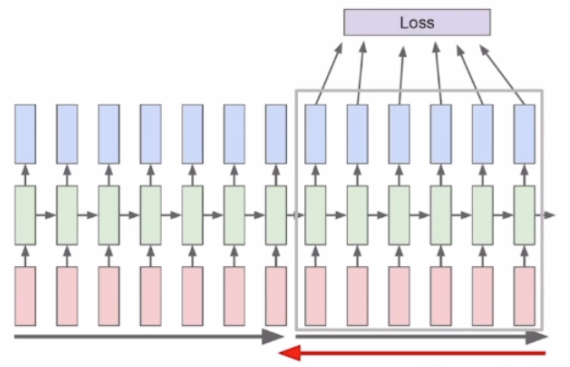
Truncation: 제한된 리소스(메모리) 내에서 모든 시퀀스를 학습할 수 없기때문에 아래 사진과 같이 잘라서 학습에 사용하는 것

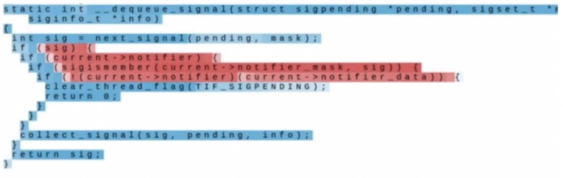
- 딥러닝 모델은 forward propagation을 통해 계산된 W를, backward propagation을 지나면서 W를 미분한 값인 gradient를 통해 학습

- BPTT(Backpropagation through time의 줄임말): RNN에서 타임스텝마다 계산된 weight를 backward propagation을 통해 학습하는 방식

하지만 기존의 Vanilla RNN은 Long-Term-Dependency의 발생으로 인해 위와 같이 학습될 수 없다.

Long-Term-Dependency: gradient가 전파되면서 소실되거나 증폭되면서 멀리까지 학습정보를 잘 전달하지 못하는 현상





특정 hidden state를 시각화한 그림으로, BPTT를 반복하게되면 다음과 같이 빨강(긍정)과 파랑(부정)으로 해당 time step에서의 중요한 부분을 잘 학습한다.

위의 hidden state 시각화는 RNN의 Long-Term-Dependency 문제를 보완한 LSTM 모델로 학습한 결과이다.

Vanishing/Exploding Gradient Problem in RNN

Toy Example

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

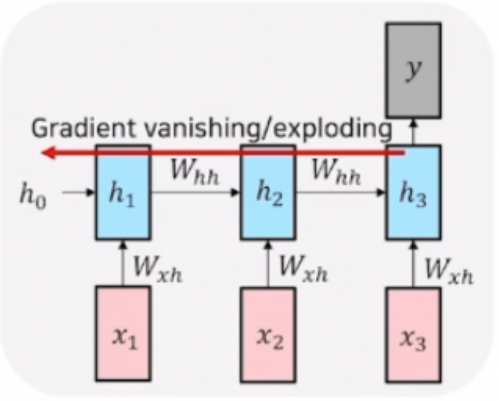
$h_3 = \tanh(2x_3 + 3h_2 + 1)$

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

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

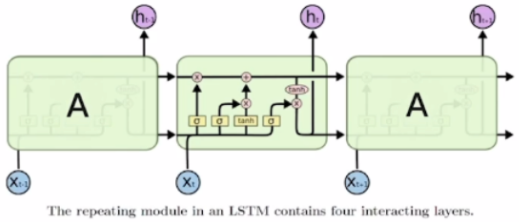
...

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



- 3번째 time step의 hidden state 인 *h*3 를 *h*1 으로 표현하면 맨 아랫줄의 식과 같이 표현된다. BPTT를 통해 gradient를 계산해주면, *tanh* 로 감싸진 괄호안의 값들 중에 3 값이 속미분되어 나오게 된다. (자세한 과정은 chain rule 참고)
- 위 예시는 time step이 3이므로 3이 2번 곱해지지만, 만약 길이가 더 길어진다면 미분값은 기하급수적으로 커질 것이고, 만약 속미분되어 나오는 W의 값이 1보다 작다면 미분값은 기하급수적으로 작아질 것이다.
- 이 계산과정을 통해 Gradient Vanishing/Exploding 문제가 발생하고 이 문제가 Long-Term-Dependency를 일으키게 된다.

LSTM : Long Short-Term Memory



LSTM의 중심 아이디어는 단기 기억으로 저장하여 이걸 때에 따라 꺼내 사용함으로 더 오래 기억할 수 있도록 개선하는 것이다.

LSTM의 여러 gate 설명

위 그림을 살펴보면 input으로 x_t, h_t-1이 들어오게 되고 이를 W에 곱해준 후 각각 sigmoid or tanh로 연산해줍니다.

Ifog 로 불리는 게이트들은 그림에서 순서대로 나타납니다.

Cell state에는 핵심 정보들을 모두 담아두고, 필요할 때마다 Hidden state를 가공해 time step에 필요한 정보만 노출하는 형태로 정보가 전파된다.

I : Input gate로 불리며, cell 에 쓸지 말지를 결정하는 게이트입니다. 즉, 들어오는 input에 대해서 마지막에 sigmoid를 거쳐 0-1 사이 값으로 표현해줍니다. 이 값은 cell state와 hidden state 두 갈래로 흐르게 됩니다.

- 표현식 : *sigmoid*(*W*(*xt*,*ht*−1)

f : Forget gate 로 불리며, 정보를 어느정도로 지울지를 0~1사이의 값으로 나타냅니다.

- 표현식 : *sigmoid*(*W*(*xt*,*ht*−1)

o : Output gate로 불리며, Cell 정보를 어느정도 hidden state에서 사용해야할 지를 0~1사이 값으로 나타냅니다.

- 표현식 : *sigmoid*(*W*(*xt*,*ht*−1)

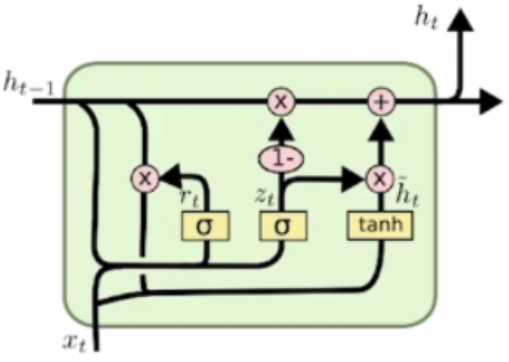
g : Gate gate로 불리며, 어느정도로 Cell state에 반영해야할 지를 -1 ~ 1 사이의 값으로 나타냅니다.

- 표현식 : *tanh*(*W*(*xt*,*ht*−1)

LSTM이 RNN과 다른점

- LSTM은 time step마다 필요한 정보를 단기 기억으로 hidden state에 저장하여 관리되도록 학
- 오차역전파(backpropagation) 진행시 가중치(W)를 계속해서 곱해주는 연산이 아니라, forget gate를 거친 값에 대해 필요로하는 정보를 덧셈을 통해 연산하여 그레디언트 소실/증폭 문제를 방지합니다.

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



- LSTM과 전체적인 동작원리는 유사하지만, Cell state, Hidden state를 일원화하여 경량화한 모델
- GRU에서 사용되는 $ht-1$ 은 LSTM에서의 ct 와 비슷한 역할을 합니다.
- forget gate 대신 (1-input gate)를 사용하여 ht 를 구할때 가중평균의 형태로 계산하게 됩니다.
- 계산량과 메모리 요구량을 LSTM에 비해 줄여준 모델이면서 동시에 성능면에서도 LSTM과 비슷하거나 더 좋은 성능을 내는 모델입니다.

RNN , LSTM , GRU 요약

- RNN은 들어오는 입력값에 대해서, 많은 유연성을 가지고 학습되는 딥러닝 모델입니다.
- RNN에서는 그레디언트 소실/증폭 문제가 있어 실제로 많이 사용되지는 않지만, RNN 계열의 LSTM, GRU 모델은 현재도 많이 사용되고 있습니다.
- LSTM과 GRU 모델은 RNN과 달리 가중치를 곱셈이 아닌 덧셈을 통한 그레디언트 복사로 그레디언트 소실/증폭 문제를 해결했습니다.