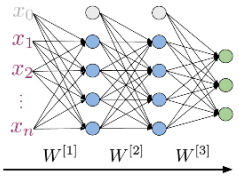
Week 1

Neural Networks with Sentiment Analysis

Forward propagation



$a^{[i]}$ Activations ith layer

$$a^{[0]} = X$$
$$z^{[i]} = W^{[i]} a^{[i-1]}$$
$$a^{[i]} = g^{[i]}(z^{[i]})$$

int matching?

Word	Number
a	1
able	2
about	3
...	...
hand	615
...	...
happy	621
...	...
zebra	1000

Tweet: This movie was almost good

↓

[700 680 720 20 55]

↓ Padding

[700 680 720 20 55 0 0 0 0 0 0]

To match size of longest tweet

(+) Trax로 구현

Week 2

RNN for Language Modeling

Traditional LM

Sequence	$P(\text{Sequence})$
I saw the game of soccer	4.5 e-5
I saw the soccer game	6.0 e-5
I saw the soccer match	4.6 e-5
Saw I the game of soccer	2.6 e-9

- Make use of probabilities to help identify which sentence is most likely to take place

$$P(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \longrightarrow \text{Bigrams}$$

$$P(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)} \longrightarrow \text{Trigrams}$$

$$P(w_1, w_2, w_3) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_2)$$

Bigram 고사용도.

Large N-grams capture dependencies between distant words and need a lot of space and RAM.

RNN

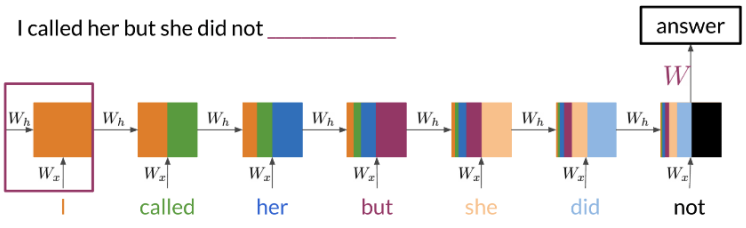
Nour was supposed to study with me. I called her but she did not _____

Trigram은 2개 단어로 다음단어 예측

want
respond
choose
want
have
ask
attempt
answer
know

RNNs look at every previous word

Similar probabilities with trigram



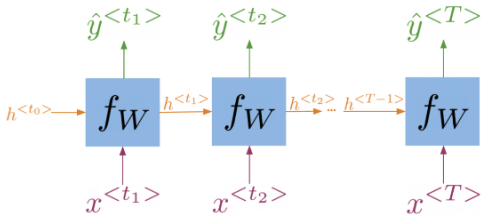
- 시간이 지날수록 과거 단어의 내용 작아짐. 그래도 존재!
- 같은 레이어 parameter를 공유함

Application of RNNs

Ex. machine translation, caption generation

- One to One: given some scores of a championship, you can predict the winner.
- One to Many: given an image, you can predict what the caption is going to be.
- Many to One: given a tweet, you can predict the sentiment of that tweet.
- Many to Many: given an english sentence, you can translate it to its German equivalent.

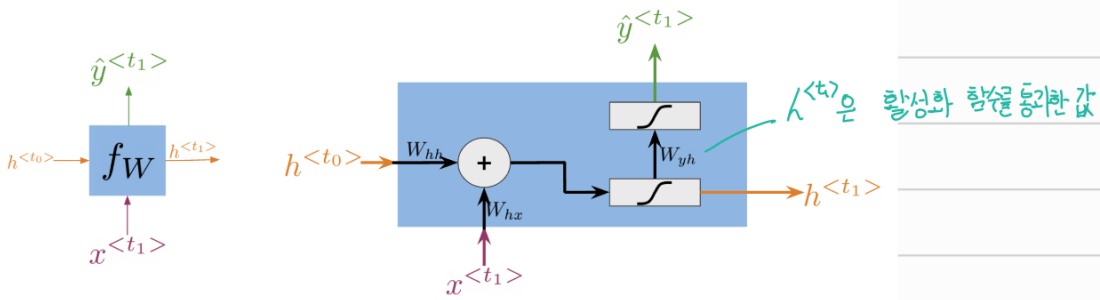
Math in Simple RNNs



$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$
$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$

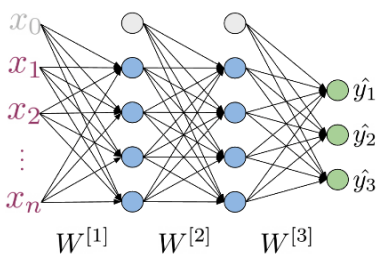
$W_h = (W_{hh}, W_{hx})$

$$h^{<t>} = g(W_{hh}h^{<t-1>} \oplus W_{hx}x^{<t>} + b_h)$$



$h^{<t>}$은 활성화 함수를 통과한 값

Cost Function for RNNs



K - classes or possibilities

$$J = -\sum_{j=1}^{K} y_j \log \hat{y}_j$$

Either 0 or 1

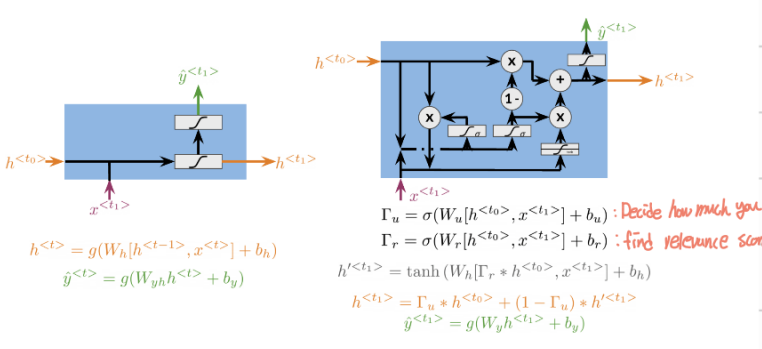
Looking at a single example (x, y)

Cross entropy loss

$$J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{K} y_j^{<t>} \log \hat{y}_j^{<t>}$$

↳ loss over several timesteps

Gated Recurrent Units



$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$
$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$

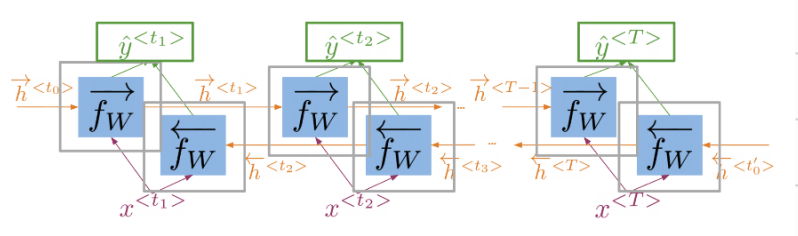
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$: Decide how much you want to update the weights by

$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$: find relevance score

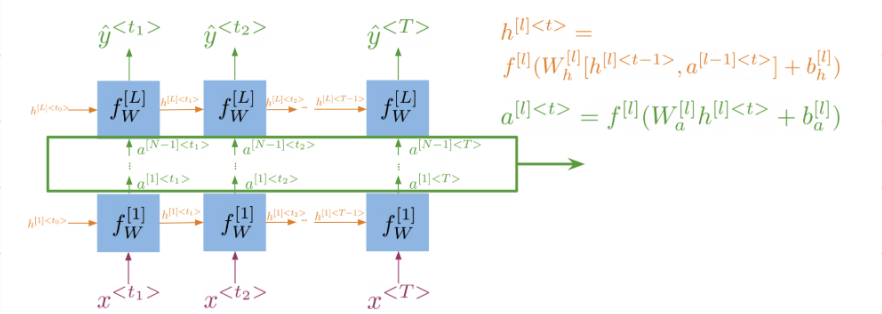
$$h'^{<t_1>} = \tanh(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h)$$
$$h^{<t_1>} = \Gamma_u * h^{<t_0>} + (1 - \Gamma_u) * h'^{<t_1>}$$
$$\hat{y}^{<t_1>} = g(W_y h^{<t_1>} + b_y)$$

Deep and Bi-directional RNNs

- Bi-directional RNNs : knowing what is next in the sentence could give you more context about the sentence itself



↳ g 만들때 양쪽 방향의 h 사용



$$h^{[l]<t>} = f^{[l]}(W_h^{[l]}[h^{[l]<t-1>}, a^{[l-1]<t>}] + b_h^{[l]})$$
$$a^{[l]<t>} = f^{[l]}(W_a^{[l]}h^{[l]<t>} + b_a^{[l]})$$

Week 3

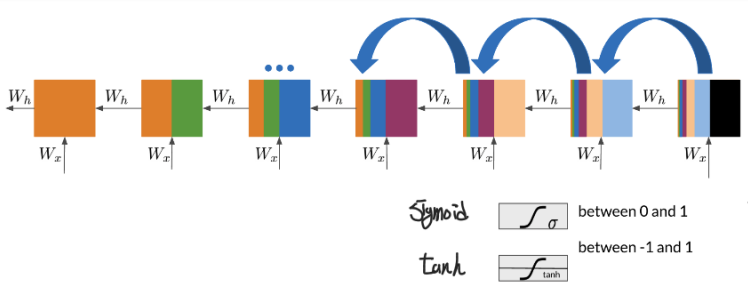
LSTMs and Named Entity Recognition

RNNs and Vanishing Gradient

- RNN은 짧은 범위의 dependency를 capture할 수 있고 전통적 LM보다 RAM을 덜 사용함
- 단점

① long-term dependency에 취약

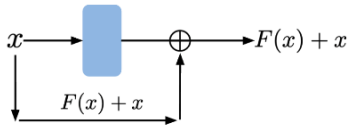
② BPTT시에 vanishing/exploding gradient에 취약



1보다 작은 값이 여러 layer를 거치며 곱해지면 결국 기울기는 0에 가까워짐

Solutions to Vanishing Gradient Problems

- Identity RNN with ReLU activation $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ -1 → 0
- Gradient clipping 32 → 25
- Skip connections



Introduction to LSTMs

- cell state + hidden state + 3 gates

Input gate: tells you how much information to input at any time point.

Forget gate: tells you how much information to forget at any time point.

Output gate: tells you how much information to pass over at any time point.

applications

Next-character prediction

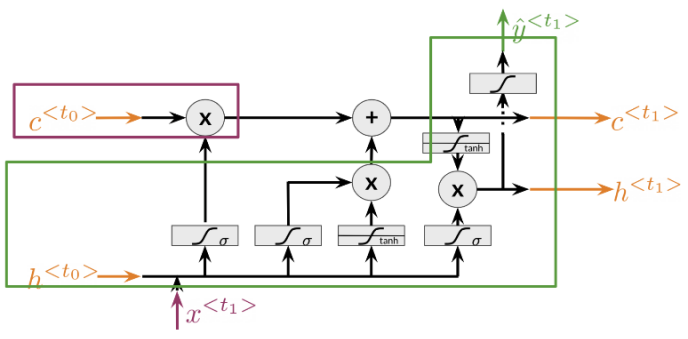
Chatbots

Music composition

Image captioning

Speech recognition

LSTM Architecture



The forget gate: $f = \sigma\left(W_f\left[h_{t-1}; x_t\right] + b_f\right)$

The input gate: $i = \sigma\left(W_i\left[h_{t-1}; x_t\right] + b_i\right)$

The gate gate: $g = \tanh\left(W_g\left[h_{t-1}; x_t\right] + b_g\right)$

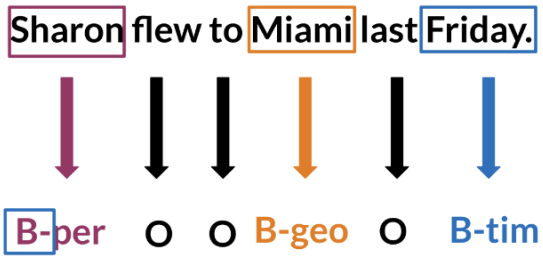
The cell state: $c_t = f \odot c_{t-1} + i \odot g$

The output gate: $o = \sigma\left(W_o\left[h_{t-1}; x_t\right] + b_o\right)$

Introduction to Named Entity Recognition (NER)

- locates and extracts predefined entities from text.

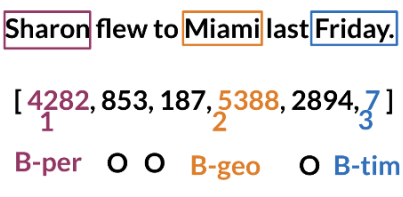
Ex. places, organizations, names, time, dates



Usage: Search efficiency, recommendation engines, customer service, automatic trading

Training NERs: Data Processing

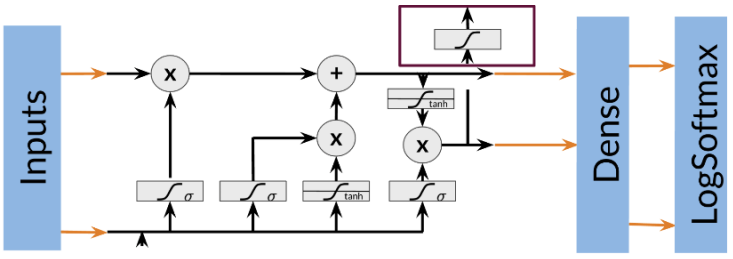
- Convert words and entity classes into arrays:
- Pad with tokens: Set sequence length to a certain number and use the <PAD> token to fill empty spaces
- Create a data generator:



Training an NER system:

1. Create a tensor for each input and its corresponding number
2. Put them in a batch ==> 64, 128, 256, 512 ...
3. Feed it into an LSTM unit
4. Run the output through a dense layer
5. Predict using a log softmax over K classes

Here is an example of the architecture:



Computing Accuracy

- Pass test set through the model
- Get arg max across the prediction array
- Mask padded tokens
- Compare with the true labels.

Week 4

Siamese Networks

How old are you? = What is your age?

Where are you from? ≠ Where are you going?

Siamese Networks: learns what makes two inputs the same

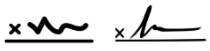
Here are a few applications of siamese networks:

Handwritten checks

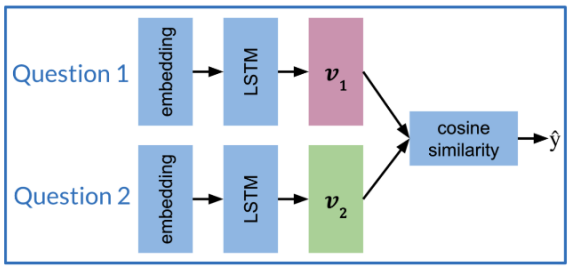
What is your age?
How old are you?

Question duplicates

Queries



Architecture



1) Embedding
2) LSTM
3) Vectors
4) Cosine Similarity

- Sub network share identical parameters
→ need to train one set of weights

Cost function

Triplet loss

How old are you? — Anchor

What is your age? — Positive

Where are you from? — Negative

$$\cos(v_1, v_2) = \frac{v_1 \cdot v_2}{||v_1||\,||v_2||}$$

$\cos(A, P) \approx 1$

$\cos(A, N) \approx -1$

$-\cos(A, P) + \cos(A, N) \leq 0$: We're trying to optimize

- $-2 \leq -\cos(A,P) + \cos(A,N) \leq 2$
  - ↑ good
  - ↑ bad

Triplets

How old are you? — A

What is your age? — P

Where are you from? — N

Simplified cost:

$$-\cos(A, P) + \cos(A, N) \leq 0$$

$$-0.5 + 0.5 + 0.2 \not\leq 0 \leq 0$$

(-1 + -1 ≤ 0)

Add a margin:

$$-\cos(A, P) + \cos(A, N) + \alpha \leq 0$$

Full cost:

$$\max(-\cos(A, P) + \cos(A, N) + \alpha, 0)$$

- Allows margin of safety
- Do not want negative number as cost

- **α:** controls how far cos(A,P) is from cos(A,N)
- **Easy** negative triplet: cos(A,N) < cos(A,P)
- **Semi-hard** negative triplet: cos(A,N) < cos(A,P) < cos(A,N) + α
- **Hard** negative triplet: cos(A,P) < cos(A,N)

Computing the cost

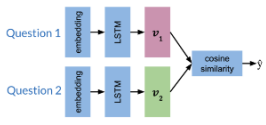
Prepare the batches as follows: ❌

What is your age?	How old are you? ✅
Can you see me?	Are you seeing me?
Where are thou?	Where are you?
When is the game?	What time is the game?

b = 4

· 각 row는 같은의미, 다른 row끼리는 다른의미

Hard Negative Mining



What is your age?
Can you see me?
Where are thou?
When is the game?

How old are you?
Are you seeing me?
Where are you?
What time is the game?

v_1 = (1, d_model)

v1_1	
v1_2	
v1_3	
v1_4	

v_2

v2_1	
v2_2	
v2_3	
v2_4	

· 각 row는 각 sentence의 encoding vector

$\cos(v_1, v_2)$

v_1

v_2	_1	_2	_3	_4
_1	0.9	-0.8	-0.3	-0.5
_2	-0.8	0.5	-0.1	-0.2
_3	-0.3	-0.1	0.7	-0.8
_4	-0.5	-0.2	-0.8	1.0

$$\text{Cost} = \max(-\cos(A, P) + \cos(A, N) + \alpha, 0)$$

· 대괄호: Score of similar sentences

· 1.은 anchor과 negative examples의 Similarity

$\cos(v_1, v_2)$

v_1

v_2	_1	_2	_3	_4
_1	0.9	-0.8	-0.3	-0.5
_2	-0.8	0.5	-0.1	-0.2
_3	-0.3	-0.1	0.7	-0.8
_4	-0.5	-0.2	-0.8	1.0

mean_neg: speeds up training ← mean negative of all the other off diagonals in the row

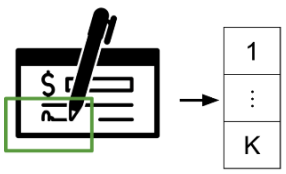
closest_neg: helps penalize the cost more ← highest number in the off diagonal

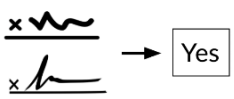
$$\text{Cost1} = \max(-\cos(A, P) + mean_neg) + \alpha, 0)$$

$$\text{Cost2} = \max(-\cos(A, P) + closest_neg + \alpha, 0)$$

The full cost is defined as: Cost 1 + Cost 2.

One shot learning





Classification: classifies input as 1 of K classes

One Shot Learning: measures similarity between 2 classes

Learn a similarity score!

$\cos(sig1, sig2) > \tau$ ✅

$\cos(sig1, sig2) \leq \tau$ ❌

Testing

Finally when **testing**:

1. Convert two inputs into an array of numbers
2. Feed it into your model
3. Compare $\boldsymbol{v}1,\boldsymbol{v}2$ using cosine similarity
4. Test against a threshold τ