

RNN, LSTM, Word Embedding, Attention, Transformer, BERT

🕒 Created	@October 15, 2021 9:31 PM
☰ Tags	post

RNN

LSTM

Word Embedding

Bag of Words, BoW

Seq2seq

Transformer

Encoder: self-attention + feed forward neural network

Decoder: self-attention + encoder-decoder attention + feed forward

BERT

Model Architecture: extension of Transformer

Related Works

BERT: from decoders to encoders

Questions

RNN

<https://www.youtube.com/watch?v=UNmqTiOnRfg>

- 일반적인 접근: Neural network as a matrix multiplication(linear mapping) + activation
- RNN: 모델의 이전 output에 의해 다음 output이 결정되는 구조
e.g. 어제 파이를 먹었으면, 내일은 버거를 먹자!
- 이전 output에 새로운 정보까지 고려하려면? matrix/vector concatenation
 - add concatenated vectors then pick the largest signal
 - split concatenated vectors and merge

LSTM

Understanding LSTM Networks

Posted on August 27, 2015 Humans don't start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You don't throw everything away and start thinking from scratch again. Your thoughts have persistence.

 <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

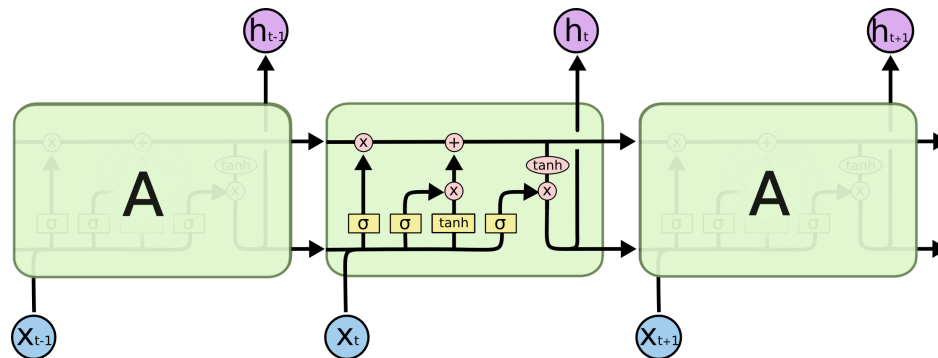
RNN: networks w/ loop, allowing info to persist, residual과 상반되는 느낌?

Challenge. gap between relevant info and current location

- if close? OK, BUT what if far, need more context, i.e., long-term dependency issue
- gradient vanishing and explosion; multiplication may lead to zero of infinity
 - 두 가지 상태(cell state, hidden state)를 보관, 덧셈을 함께 사용하여 이 문제를 해소

Long Short Term Memory, LSTM

Repeating module differs; not a single tanh, rather complex



From the left most σ ,

- forget gate layer(which to forget)
- input gate layer(which to update) + candidate cell state
- decide what to output

There exist multiple variants; peephole(consider cell state at sigmoid), coupled forget-input, GRU

Word Embedding

What Are Word Embeddings for Text? - Machine Learning Mastery

Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation. They are a distributed representation for text that is perhaps one of the key breakthroughs for the impressive performance of deep learning methods on

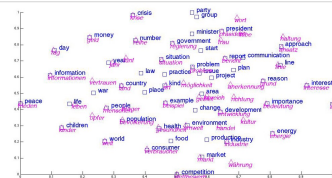
<https://machinelearningmastery.com/what-are-word-embeddings/>



Glossary of Deep Learning: Word Embedding

The Big Idea of Word Embedding is to turn text into numbers. This transformation is necessary because many machine learning algorithms (including deep nets) require their input to be vectors of continuous values; they just won't work on strings of plain text.

<https://medium.com/deeper-learning/glossary-of-deep-learning-word-embedding-f90c3cec34ca>



- What
 - building a low-dimensional vector representation from corpus of text, which preserves the contextual similarity of words
 - type of word representation that allows words with similar meaning to have a similar representation
 - dense distributed representation, better generalization power
 - contrast to *bag of words* model
- How: algorithms
 - Embedding layer: learned jointly w/ NN on a specific NLP task
 - each word one-hot encoded
 - randomly initialized vectors → backprop
 - need large dataset, take long time

- Word2Vec: statistical method
 - efficiently learn standalone word embedding from text corpus
 - Continuous Bag-of-words, CBOW model: predicting current word based on its context
 - Continuous skip-gram model: predicting the surrounding words for given current word
 - context is defined by a window of neighboring words
- GloVe; Global Vectors for Word Representation, extension of word2vec
 - approach to marry global statistics with the local context-based learning
 - Rather than a window to define local context, constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus
- Using word embeddings
 - Learning: standalone vs jointly(specific task)
 - Reuse: static vs updated

Bag of Words, BoW

From a huge text corpus, count frequency of each word

- mainly for statistical analysis, bad at language generation or modeling
- cannot consider word ordering → n-gram; count on the unit of n-sequential words

Seq2seq

Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)

Translations: Chinese (Simplified), Japanese, Korean, Russian, Turkish Watch: MIT's Deep Learning State of the Art lecture referencing this post May 25th update: New graphics (RNN animation, word embedding graph), color coding, elaborated on the final attention example. Note: The animations below are videos.
<https://jalamar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>



- seq2seq: input sequence of items to output sequence of items
 - common models get fixed-size input, not a sequence
- Basic structure
 - Flow: input → encoder → vector of floats, called *context* → decoder → output
 - Both en-decoders are RNN(to be more specific, LSTM), last hidden state outputted from encoder is the context
 - Context vector was the bottleneck → *Attention!*
 - Even so, still an RNN - 과거의 정보를 context vector 하나에 온전히 담아낼 수 있는가
- Attention
 - encoder passes all the hidden states to decoder
 - preprocessing before actual decode
 - each hidden state most associated w/ one certain word
 - score each and multiply by softmaxed score, thus amplifying hidden states w/ high scores and down in other cases
 - not FIFO, but also learn how to align words

Transformer

The Illustrated Transformer

Discussions: Hacker News (65 points, 4 comments), Reddit r/MachineLearning (29 points, 3 comments) Translations: Chinese (Simplified), French, Japanese, Korean, Russian, Spanish, Vietnamese Watch: MIT's Deep Learning State of the Art lecture referencing this post In the previous post, we looked at Attention - a ubiquitous method in modern deep learning models.

<https://jalammar.github.io/illustrated-transformer/>

Transformer: also encoder-decoder based architecture, stacks of each whose weights not shared


Parallelism: word in each position flows through its own path in the encoder → speed up easily

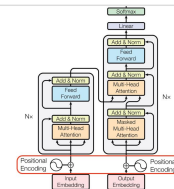
Encoder: self-attention + feed forward neural network

- self-attention: helps the encoder look at other words in the input sentence as it encodes a specific word
 - to bake the “understanding” of other relevant words into the one we’re currently processing e.g. 대명사 it 이 지칭하는 것은? 이러한 matching은 other words에 기반함
 - Query vector, Key vector, Value vector: matrix(trained) multiplication w/ embedding
 - calculate score of each word, against the target word - dot product of query v and key v
 - normalize by division and softmax
 - multiply value vector and softmax then add up weighted value v_i s
- word embedding only done at the very first layer, then its output fed to the next encoder
- multi-headed attention
 - maintain separate Q, K, V weight matrices for each head
 - concatenate all the outputs then condense down by multiplying with weight matrix
- Positional Encoding
 - vectors following a pattern to better determine position of each word

Transformer Architecture: The Positional Encoding

Transformer architecture was introduced as a novel pure attention-only sequence-to-sequence architecture by Vaswani et al. Its ability for parallelizable training and its general performance improvement made it a popular option among NLP (and recently CV)

 https://kazemnejad.com/blog/transformer_architecture_positional_encoding/



- due to parallelism, each word does not have sense of ordering → add piece of info
 - [0, 1]? how many words are there?
 - [1, # of words]? fixed length..
- voila! positional encoding vector!
 - d -dimensional vector, sin & cos terms w/ differing frequencies
 - encoding is not integrated into the model, but calculated and added separately

$$d_{word\ embedding} = d_{positional\ embedding}$$

represent a number in binary format, how will that be?

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

You can spot the rate of change between different bits. The LSB bit is alternating on every number, the second-lowest bit is rotating on every two numbers, and so on.

- Residual + Layer Normalization
 - residual; link original input to the output, to maintain original data's characteristics
 - layer normalization
 - batch norm; dependent on batch size and not intuitive to apply on RNN
 - compute mean & variance from all of the summed inputs to the neurons in a layer on a single training case, at each time step(fits RNN)

Decoder: self-attention + encoder-decoder attention + feed forward

- encoder-decoder attention: helps the decoder focus on relevant parts of the input sentence
- self-attention layer is only allowed to attend to earlier positions in the output sequence

BERT

Bidirectional Embedding Representations from Transformers

The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)

Discussions: Hacker News (98 points, 19 comments), Reddit r/MachineLearning (164 points, 20 comments) Translations: Chinese (Simplified), French, Japanese, Korean, Persian, Russian 2021 Update: I created this brief and highly accessible video intro to BERT The year 2018 has been an inflection point for machine learning models handling text (or more accurately, Natural Language Processing or NLP for short).

<https://jalammar.github.io/illustrated-bert/>

BERT provides a well, pre-trained model as an open source

- pre-training? semi supervised, Objective = predict masked word
- apply supervised learning step, *fine-tune* for your own goal

Model Architecture: extension of Transformer

- two model sizes - Base, Large
- BERT = transformer encoder stack, bigger than original one
- CLS(Classification) token in front of each input sentence

- Outputs vector at each position, which then fed to classifier(fc layer)
 - similar to what happens between a convolution part of a network and fc layers at the end
- **Embedding**; pretrained on huge dataset e.g. Word2Vec, GloVe

Related Works

ELMo: importance of context

"stick" == "n. long bar"? or "v. cling to"? meaning differs based on context

→ contextualized word-embeddings; uses a bi-directional LSTM

- pretrain on massive dataset and use it as a component
 - predict the next word in a sequence of words - *language modeling*
- can see hidden state of each unrolled-LSTM
- bi-directional; sense of next word + previous word
- embedding = concatenate hidden states from each layer + weighted sum

ULM-FIT: transfer learning in NLP, introduce language model and process to effectively fine-tune

Transformer: replace LSTM in that it can handle long-term dependencies better

But, how can we use it to classify sentences or to pre-train a language model?

OpenAI Transformer(GPT): pre-training a transformer decoder for language modeling

just w/ decoder; predict the next word, built to mask future tokens

BERT: from decoders to encoders

openAI transformer only trains a forward language model → backward?

- masked language model; randomly mask 15% of tokens
- two-sentences tasks; given two sentences A and B, is B coming next A?

May use it for fine-tuning or just to extract features like ELMo

Questions

- openAI에서 decoder만 쓰는 이유? forward prediction/generation이 목적이니까?

Arguments for multi-layer decoder-only Transformer · Issue #157 · openai/gpt-2

Hello, I recently started studying language modeling and GPT(-2) in particular. While I start to understand the way it is trained/fine-tuned, I do have some questions about its architecture. In OpenAI's paper it is stated that GPT (and G...

<https://github.com/openai/gpt-2/issues/157>

openai/gpt-2

#157 Arguments for multi-layer decoder-only Transformer

4 comments

ZheMann opened on July 14, 2019

- masked modeling의 장점?
 - labeling cost를 줄임, enable few shot learning
- fine-tuning하는 것과 feature extraction을 위한 component로 사용하는 것의 차이?
 - fine-tuning의 경우 모델을 일부 수정하여 학습 진행
 - feature extraction으로 활용할 때는 BERT 결과를 아예 별도로 집어넣게 됨?