Recent Advancements in NLP (1/2)

Moiz Saifee
Dec 10 · 7 min read ★

2018 has been widely touted as the “Imagenet moment” in the field of NLP due to the release of pre-trained models and open source algorithms by big players like Google, Open AI, Facebook etc. I have tried to summarize some of the important advancements in the field of NLP in this blog.

I have spent a good fifteen years in the field of Data Science and AI and have worked a fair bit on NLP in this period, but the pace with which things have been moving in the last 2–5 years is unprecedented. If you have not caught up with all the exciting work, you may be lagging behind significantly on what’s possible these days.

NLP is one of my favorite areas given its wide applications. Some of the popular applications by which we interact every day are — search engines, machine translation, chatbots, home assistants etc.

This is my first blog in a series of two blogs to capture the exciting work in the field of NLP

NLP is Hard

In addition to being tremendously useful, NLP is also pretty challenging, consider the following few examples:

“Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo” is a grammatically correct sentence

“John saw the man on the mountain with a telescope” Who is on the mountain? John, the man, or both? Who has the telescope? John, the man, or the mountain?

These challenges in interpreting language can cause Siri to do something like this

I have summarized some of the standard techniques which were used in the past along with some popular recent ones.

Pre 2012:

- Using the bag of words representation which paid no heed to the ordering of words was the norm
- Latent Semantic Indexing (LSI), SVM (Support Vector Machine) and Hidden Markov Models (HMM) were the popular techniques
- Standard tricks of the trade were — POS tagging, NER etc

2013–2016

One of the more important developments in this period was work on “word embedding”.

Word Embedding

Word Embedding is nothing but useful numerical representation of words. While the early work on this concept was done as far back as 2003 (Bengio et al.), but it was really work by Tomas Nikolov et al (from Google) in 2013 on Word2Vec which gave it a real push.

One important point to note is, for many word embedding algorithms — while the representation is "learned" by training a model on certain task, example — to predict the next word given surrounding words — it turns out these learned representations give great performance in a variety of other NLP tasks

Word2Vec (2013) by Google

Uses Neural Network to come up with the word representations. There are two popular variants:

1. Continuous bag of words (CBOW) — In this approach we try to predict a given word given its neighboring words (context)
2. Skip Gram — The approach is similar, but we solve the reverse problem — given target words, try to come up with neighboring words

Word2Vec has a lot of nice and desirable benefits:

- The original model was trained on 1.6 B word dataset and with millions of words in the vocabulary
- The training is pretty fast
- Even people with modest amount of data could use these pre-calculated embeddings (generated by training on huge data) in their specific tasks and walk away with a lot of improvement
- The Word2Vec training itself was unsupervised, that is did not require pricey annotated data
- The numerical representation of words which were similar in meaning were close to each other (had small cosine distance)
- "Somewhat surprisingly, it was found that similarity of word representations goes beyond simple syntactic regularities. Using a word offset technique where simple

algebraic operations are performed on the word vectors, it was shown for example that *vector("King") — vector("Man") + vector("Woman")* results in a vector that is closest to the vector representation of the word *Queen"*

There was additional work which was done on Word Embedding after the impetus which it received with Word2Vec. The most notable variants are:

GloVe — Global Vector for Word Representation (2014) by Stanford

- Attempted to combine benefits of both — global matrix factorization methods (like LSA), which do well on capturing statistical structure and local context windows methods like Word2Vec, which do well on analogy tasks.
- Instead of extracting numerical representations from training a neural network for certain task like predicting the next word, GloVe vectors have inherent meaning, which is derived from word co-occurrences

FastText (2016) by Facebook

- Its conceptually somewhat similar to Word2Vec, but with a twist — instead using words for creating embeddings, it uses n-gram of characters. For example, fastText representation of word "test" with n=2 would be <t, te, es, st, t>
- One of the major benefits of FastText is, the above approach allows it to generalize to unknown words, or the words which were not part of vocabulary in training
- Requires lesser training data compared to Word2Vec.

That's it on embeddings — the key takeaway is you can find powerful numerical representations of words in almost any major language in the world, which are trained on huge text corpses and can give you a massive headstart in the NLP problem you are trying to solve, thanks to the algorithms listed above.

2014–2016

While a lot of work was happening around 2014 on embeddings, with the onset of Deep Learning era, another old technique was staging a comeback — the RNN

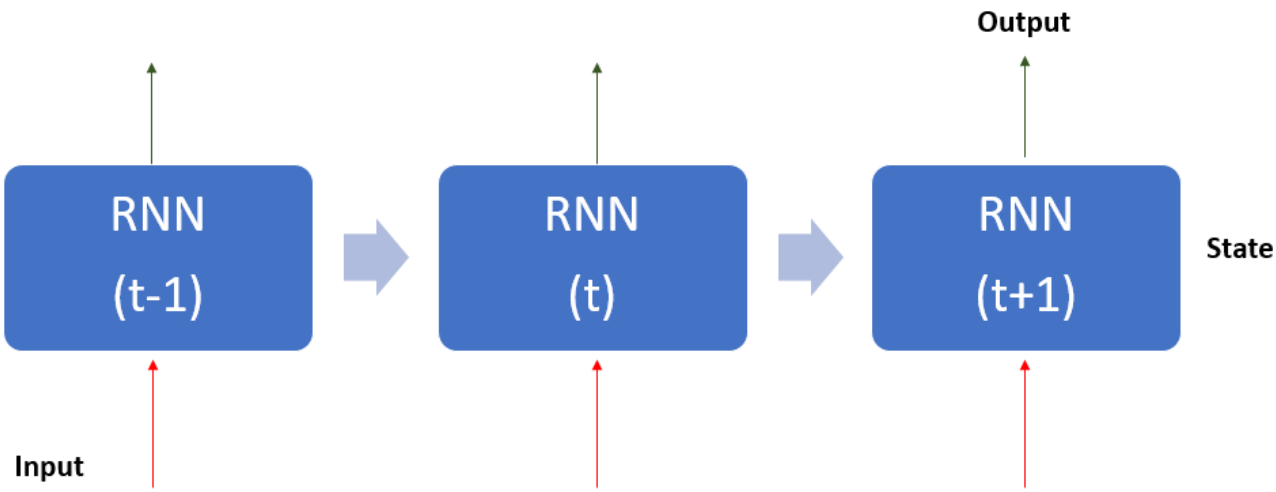
RNN — Recurrent Neural Network

RNN are a type of neural network that specialize in dealing with problems whose input / output are sequential in nature. The text data is inherently sequential given the fact that words are not placed randomly in a sentence and are informed by the grammar of the language. The idea of RNN was proposed as far back as 1980s, but they didn't really take off given the challenges in training the model

Why RNN?

The basic challenge of classic feedforward neural network is that it has no memory, that is, each training example given as input to the model is treated independent of each other. In order to work with sequential data with such models — you need to show them the entire sequence in one go as one training example. This is problematic because number of words in a sentence could vary and more importantly this is not how we tend to process a sentence in our head.

When we read a sentence, we read it word by word, keep the prior words / context in memory and then update our understanding based on the new words which we incrementally read to understand the whole sentence. This is the basic idea behind the RNNs — they iterate through the elements of input sequence while maintaining a internal "state", which encodes everything which it has seen so far. The "state" of the RNN is reset when processing two different and independent sequences.



While all this sounds great, but the reality is simplistic RNN described above are rarely used in practice. The reason is — while in theory the RNN should be able to retain the information seen many timesteps before, in practice this is almost impossible to do due to the *vanishing gradient* problem. Vanishing gradient is the problem due to which

deeper networks take forever to train as the updates in parameters of the network becomes painfully slow. Two variants of RNN — LSTM and GRU were designed to solve this problem.

LSTM — Long Short Term Memory

LSTM is a variants of RNN with a way for carrying over information across many timestamps without getting diluted (vanished) due to a lot of processing.

Without getting into too much mathematics, the way LSTM achieves this is by means of three gates — *Forget, Input* and *Output* Gate. The *Input* gate determines the extent to which the current timestamp input should be used , the *Forget* gate determines the extent to which output of the previous timestamp state should be used, and the *Output* gate determines the output of the current timestamp

GRU — Gated Recurrent Unit

The GRU introduced in 2014 by Kyunghyun Cho et al was a simplified version of LSTM with just two gates instead of 3 and with a far fewer parameters vs LSTM.

GRU have been shown to have better performance on certain tasks and smaller datasets, but LSTM tend to outperform GRU in general.

There were some additional variants like bidirectional LSTM which don't just process text left to right, but also do it right to left to give additional performance boost.

Closing Remarks

There was a lot of important work which happened off late on Word Embedding and Neural Network algorithms like RNN. These developments led to some rapid progress in the field of NLP and significantly improved the performance on various benchmark datasets.

But ..

The last few years have seen the rise of a new family of models called ***Transformers —*** These *attention* based models have completely swamped most of the other things. Many of the leading players have open sourced their version of Transformer architecture based models — BERT (Google), Transformer Xl (Google), GPT-2 (OpenAI), XL Net

(CMU), ERNIE (Baidu). I'll try to cover these new exciting developments in my next blog.

)

NLP Word Embeddings Rnn Artificial Intelligence Naturallanguageprocessing