Introduction to NLP

A review of historic and deep NLP goals and methods

Matt Smith

UW Data Science Club

13 February, 2019

Table of Contents

Table of Contents

Natural Language Processing

Natural Language Processing is the study of Natural (human) languages. What do people do with languages?

- Write
- Speak
- Organize their thoughts

If we can understand how people use language, we can design products or solutions that use natural language instead of machine language.

Alphabets and words and phrases and sentences and...

Some "facts" about languages:

- Languages have a finite, small alphabet
- Certain arrangements of letters compose the words of a language
- Certain arrangements of words make phrases, which convey an idea
- Phrases are joined together into simple or complex sentences

Many people should see at least one thing wrong here...

Alphabets and words and phrases and sentences and...

Some "facts" about languages:

- Languages have a finite, small alphabet
- Certain arrangements of letters compose the words of a language
- Certain arrangements of words make phrases, which convey an idea
- Phrases are joined together into simple or complex sentences

Many people should see at least one thing wrong here...

Not every natural language follows these rules For example, Mandarin doesn't have a small set of alphabets.

Nevertheless, we apply this English-centric lens and press on.

The primal decompositions of language

One of the early goals of NLP was to "factorize" language.

Ideally, language should be:

The primal decompositions of language

One of the early goals of NLP was to "factorize" language.

Ideally, language should be:

- Finitely factorizable

The primal decompositions of language

One of the early goals of NLP was to "factorize" language.

Ideally, language should be:

- Finitely factorizable
- Deterministically factorizable

The primal decompositions of language

One of the early goals of NLP was to "factorize" language.

Ideally, language should be:

- Finitely factorizable
- Deterministically factorizable
- Uniquely factorizable

The primal decompositions of language

One of the early goals of NLP was to "factorize" language.

Ideally, language should be:

- Finitely factorizable
- Deterministically factorizable
- Uniquely factorizable

Unfortunately, it seems like none of these are true without making significant assumptions.

Phones and Phonemes

An intuitive way to break down a word is into its sounds.

IPA: an alphabet that describes how words are pronounced. Useful for speech recognition – recognize fundamental sounds instead of letters.

e.g. cat == /kæt/

By using IPA we can normalize languages with weird pronunciation such as English — consider the words enough; though; through; plough; dough; cough?

All look really similar but don't have similar pronunciations.

Phones and Phonemes

Phones: It turns our phonemes are not sufficient to capture all pronounciation information. E.g. the /t/ in tea and trip. Phones are technically the "units" of spoken language but are not as often used.

Criticisms of IPA/Phonemes:

- Isn't complete – no epiglottal or palatal sounds
- People pronounce words in different ways
- Doesn't capture tonal information – 妈 (mom) vs 马 (horse)
- Doesn't capture pitch accent – 决行 (going) vs 欠航 (cancelled)
- Doesn't capture pitch semantics
- (Surprisingly) language-specific

Graphemes

If sound doesn't work, then what if we break down words into their fundamental writing units?

Graphemes are blocks of letters that represent the core "writing units" of words.

In English these are usually alphabets like <s>, <h> and <t> but other graphemes such as <th> and <oo> exist.

It was decided that graphemes should ideally be 1-to-1 with phonemes.

Graphemes in non-English languages

Graphemes are really broken for CJK.

Korean has an alphabet, but the letters are combined into blocks. E.g. ㅎ (h) + ㅓ (eo) + ㄹ (l) = 헐 (heol).

Are the letters graphemes, or are the blocks?

Chinese characters use a radical system to link words by idea or pronunciation. Consider 话/話 (speech) and 语/語 (language). Both use the mouth radical 口 to signify meaning (which is part of the speech radical 讠/言). 语/話 uses the pronunciation radical 五 (wǔ/go).

What are the graphemes – the whole character? The radicals? The "important" radicals?

Morphemes

Can we break down words into their semantic components?

In English:

1. -s represents plurality
2. -ing represents infinitive conjugation
3. hyponatremia – hypo- (low), natrium (sodium), -emia (presence in blood)

Other languages use entire words/characters as morphemes. 小心 – literally "small heart" – means caution.

Morphemes may be represented by multiply phonemes/graphemes.

Phonemes, graphemes, morphemes and futility

Why bring all of this up?

Other than phonemes seeing occasional usage in voice understanding and generation, we don't care a lot about phonemes, graphemes or morphemes.

You should know:

- Language is complicated; assumptions are necessary and bad
- Historically these are interesting problems
- Factorization is an interesting problem to linguists
- This was the first application of deep NLP

Phonemes, graphemes, morphemes and futility

A common theme in phonemes, graphemes and morphemes is that they create a model of language that doesn't exactly "fit" with reality.

Part of the problem is that these constructs were decided on by humans in a way that was only partially data-driven.

It turns out that when we allow neural networks to pick what should be a phoneme or a grapheme, they do really well.

This was the first big win for deep NLP.

Table of Contents

Word Factorization

Classical NLP has been applied to many tasks.

You already saw word factorization: Given a sentence, decompose each word into its phonemes/graphemes/morphemes.

This is an example of a sequence-to-sequence task, as the input and the output are both sequences. Note that the sequences do not need to be 1-to-1 in the case of factorization; it will be clear, I think, when the sequences need to be the same length.

Let's look at a few other sequence-to-sequence tasks.

Stemming/Normalizing

Given a word, convert it into a normalized form. This is useful as it reduces the number of different words that we might encounter.

The most common form of this task is stemming. We can trim off a suffix of a word and expose the root word, leaving the meaning (hopefully) intact.

Stemming example

"data scienc is an interdisciplinari field that use scientif method , process , algorithm and system to extract knowledg and insight from data in variou form , both structur and unstructur , similar to data mine ."

Other normalization techniques include removing "inessential" parts of speech like stop words, capital letters and numbers.

POS tagging

Words don't have a fixed grammatical meaning withing a sentence. For example the word 'salt' can refer to the substance or the action of adding salt to something.

We can use Part of Speech tagging to find this grammatical information.

POS tagging example

[('And', 'Conj'), ('now', 'Adv'), ('for', 'Prep'), ('something', 'Noun'), ('completely', 'Adv'), ('different', 'Adj')]

Generic Tagging

POS models can be used to tag other things. For example, tagging the words in a sentence that are food items.

Lemmatization

We can combine POS tagging with stemming.

If we know the POS tag for a word, we can stem it more aggressively and correctly.

- We can "undo" conjugations of verbs
- We can distinguish between homographs that have different roots

Homographs

The word "entrance" can mean "an opening" and become "enter"; or it could be "make [someone] go into a trance" and become "trance".

Homographs are not as common in English as they are in, say, Telugu.

NER

Named Entity Recognition involves tagging proper nouns or other words whose meaning isn't obvious without context.

NER example

Mark bought 250 shares of Microsoft in 2015 => $[\text{Mark}]_{\text{Person}}$ bought $[250]_{\text{Number}}$ shares of $[\text{Microsoft}]_{\text{Company}}$ in $[2015]_{\text{Date}}$

Coreference/Anaphora resolution is part of NER. Often, we refer to proper nouns in multiple different ways, sometimes even implicitly.

Coreference example

International Business Machines == IBM

Anaphora example

"Susie went to the mall. She didn't buy anything." => Susie == She

Information extraction

Given a larger piece of text, extract some kind of symbolic representation. Can we represent the information contained within a text as some propositional or predicate logic?

This often requires some kind of NER to be done first.

IE example

"The Stark Enterprises and WayneCorp merger succeeded yesterday." => businessMerger(starkCo, wayneCo, 2019-02-12)

Relationship extraction is a special case of this.

RE example

"The Stark Enterprises and WayneCorp merger succeeded yesterday." => friends(tonyStark, bruceWayne, 2019-02-12)

Data augmentation

This is a combination of the previous techniques. Use classical NLP techniques to annotate data. For example, you might replace each word in your training set with a triple of the word, its POS tag and some NER data.

Rules-based NLP pipelines are understandable (if the design principles are public) and faster than ML ones, which makes classical NLP a good choice for this kind of task.

However, the information present in these tags are often present in embeddings. Some kind of data augmentation, such as stemming, is still done in word-based models.

Classification

Departing from many-to-many sequence methods, we'll now look at many-to-one methods. Given a sequence, we want to output a probability or a predicted value.

Spam detection

Given a document, tell me if it's spam or not.

Document classification

Given a document, tell me what kind of a document is it. Is it an email, an invoice, a review for a restaurant, etc.?

Sentiment/Sarcasm detection

How positive is a given sentence? Often this is trained on review data from Amazon, Yelp or IMDB datasets. Sarcasm detection is the new hotness IMO.

Question answering

We move into the domain of tasks that classical NLP struggled to succeed in, but deep NLP has had great successes in.

Question answering networks take as input a text and a multiple choice question, and output the correct answer to the question. The question is usually a reading comprehension question like you'd expect to see on the SAT.

A more difficult task is the "Story Cloze Task", where a model reads a short story and outputs the ending. See the ROCStories dataset.

Story Cloze example

Karen was assigned a roommate her first year of college. Her roommate asked her to go to a nearby city for a concert. Karen agreed happily. The show was absolutely exhilarating. => Karen became good friends with her roommate.

Text summarization

Very related to question answering.

Read a piece of text like an article or a legal document and summarize it in a brief paragraph.

There are bots on Reddit that do a very good job of this. Autotldr uses smmry.com to create summaries.

I grabbed some random NYT articles and reduced them to 7 sentences long. It does a pretty good job, and the article is pared down to about 1/10 of its original size.

See the next page for a summary of the NYT article *"Tech Is Splitting the U.S. Work Force in Two"*.

Text summarization

Automation is changing the nature of work, flushing workers without a college degree out of productive industries, like manufacturing and high-tech services, and into tasks with meager wages and no prospect for advancement. Recent research has concluded that robots are reducing the demand for workers and weighing down wages, which have been rising more slowly than the productivity of workers. Workers' jobs were saved because the company brought other manufacturing work back from Mexico. The 58 most productive industries in Phoenix employed only 162,000 people in 2017, 14,000 more than in 2010. Employment in the 58 industries with the lowest productivity grew 10 times as much over the period, to 673,000. Adair Turner, a senior fellow at the Institute for New Economic Thinking in London, argues that the economy today resembles what would have happened if farmers had spent their extra income from the use of tractors and combines on domestic servants. Mr. Acemoglu and Pascual Restrepo of Boston University argue that businesses are not even reaping large rewards for the money they are spending to replace their workers with machines.

Text-to-speech-to-text

Given some text, can you say it out loud?

Given some speech, can you understand what is said?

Speech understanding is not strictly NLP, but it's a language problem that involves sequence modeling so I would call it at least "applied NLP".

We can get some pretty good results if we just do text-to-phonemes and speech-to-phonemes, as there are rule-based tools for turning phonemes into speech (although such speech is kind of robotic) and decoding phonemes into text.

As you may expect, deep NLP models can do speech generation from phonemes much better than classical methods. I believe phoneme decoding is still done by classical statistical models such as CRFs.

Trigger word detection

Can you detect when someone says “Okay Alexa/Google/Cortana/Siri”?

Can you only activate on the voices of your owners?

More speech-to-text tasks include diarisation (chunking audio based on the speaker) and automatic captioning of videos.

Image annotation

Given a representation of an image (could be from a classical image processing algorithm or from a CNN encoder), output a caption that describes the image.

Often defining a good loss function for these is difficult. Something called the BLEU score is often used for both image annotation and machine translation.

BLEU measures the similarity between a human-generated translation or annotation and the machine-generated one. It tries to do this is a way that it doesn't penalize making trivial changes to the sentence.

Warning

Many people think that BLEU is a bad metric. Read *"Re-evaluating the Role of Bleu in Machine Translation Research"*.

Machine translation

Really similar to image annotation!

Instead of receiving a representation of an image, we instead encode a representation of a sentence and then decode it in a different language.

Bias warning

I don't want you to think all image annotation and machine translation has to use this encoder/decoder architecture. However, it's what everyone is using right now: Google, Microsoft, Yandex and StanfordNLP all use this kind of Neural Machine Translation (NMT) model.

Tools

These tools all apply classical NLP techniques:

- spaCy
- Stanford NLP
- NLTK

Many of these are being augmented/rewritten to use deep NLP (e.g. spaCy >= 2 is entirely ML).

Some people pi the non-ML versions of these toolkits to train their models. The ML versions are slower, and people may not want to retrain their models after tools change.

Table of Contents

Motivating deep NLP

The same motivations for applying deep learning to any field:

- We want to remove human assumptions from our models
- Our current best solutions are terribly complex rule-based systems
- We don't know much about linguistics but want to make solutions
- We want to be paid a lot to look at TensorBoard and go "Hmm"
- We want to put buzzwords on our resumes

Deep learning in particular is good at making expert systems without expert knowledge.

Deep NLP terms

We usually have a collection of text to use as training data. If our individual texts are large bodies of texts, we call them **documents** and call our collection of text a **corpus**.

We can break our documents apart into paragraphs, sentences, words, etc. The most common paradigm is to deal with documents one sentence at a time, and to handle sentences by sequentially evaluating words or characters.

Sometimes **bigrams** are used instead of words. Bigrams are pairs of adjacent words in a sentence. For example the sentence "The quick black cat" has bigrams (The quick), (quick black) and (black cat).

N-grams

The general form of a bigram is called an **n-gram**. Words are sometimes called **unigrams** or 1-grams. n-grams are rare in practice for $n \geq 3$.

Deep NLP terms

Word-based models usually have a set of words that they understand. This is called the model's **vocabulary**. We often pre-process the input to the model and replace out-of-vocabulary words with a special word UNK, pronounced "unk".

The NUM token

Many models also replace all numbers with a special NUM token. Get creative: you can replace all URLs with a URL token if you don't think URLs are semantically relevant.

Because we want to make the most of our vocabulary, we usually lemmatize our text for both training and deployment. The vocabulary is often fixed if you are using pre-trained embeddings, though you may always remove words if you want to save memory (more on embeddings later).

Deep NLP terms

We usually classify deep NLP models in a few different ways:

Which units of language do they operate on? There's character models that operate on individual characters, and word models which operate on entire words. There's also bigram/n-gram models.

What is the output type? One-to-many, many-to-one, many-to-many, encoder/decoder.

What model is used? LSTM/GRU/Transformer/CNN/etc.

Sequence models

How do we model sequences? We could add a limit to the sequence length and use a standard neural network.

This doesn't work well:

Sequence models

How do we model sequences? We could add a limit to the sequence length and use a standard neural network.

This doesn't work well:

- The input space of such a network will be huge

Sequence models

How do we model sequences? We could add a limit to the sequence length and use a standard neural network.

This doesn't work well:

- The input space of such a network will be huge
- The neural network will have to re-learn sequence-based properties for every position in the input sequence

Sequence models

How do we model sequences? We could add a limit to the sequence length and use a standard neural network.

This doesn't work well:

- The input space of such a network will be huge
- The neural network will have to re-learn sequence-based properties for every position in the input sequence
- Sequences can be arbitrarily long, sometimes

RNNs

Instead we use recurrent neural networks (RNNs): A RNN is a neural network that has an internal state h.

For each part of our input, our RNN will take an input x_i and combine it with its hidden state h_{i-1} to produce:

- An updated hidden state h_i
- Possibly some output y_i

Stack the layers

We can stack RNNs just like layers of regular neural networks. The final layer can output whatever shape you want, but the intermediate layers must produce an output on each input.

RNN shapes

RNNs don't have to output something at every timestep.

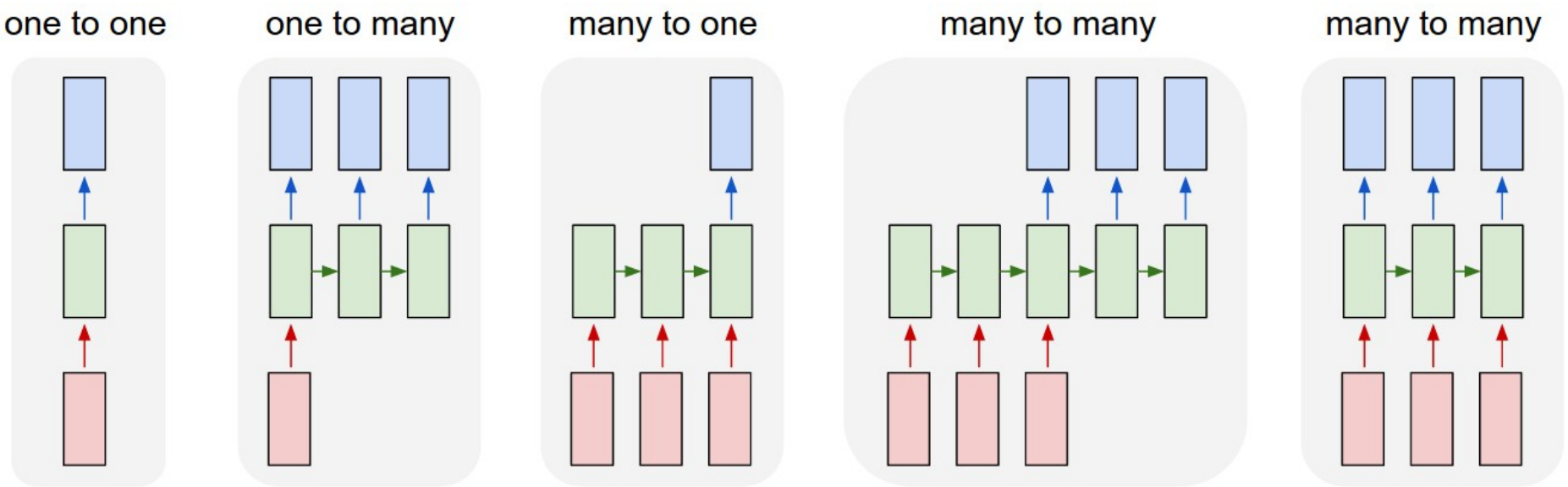


Figure: Examples of stacked RNNs with different output shapes.

Read *"The Unreasonable Effectiveness of Recurrent Neural Networks"*.

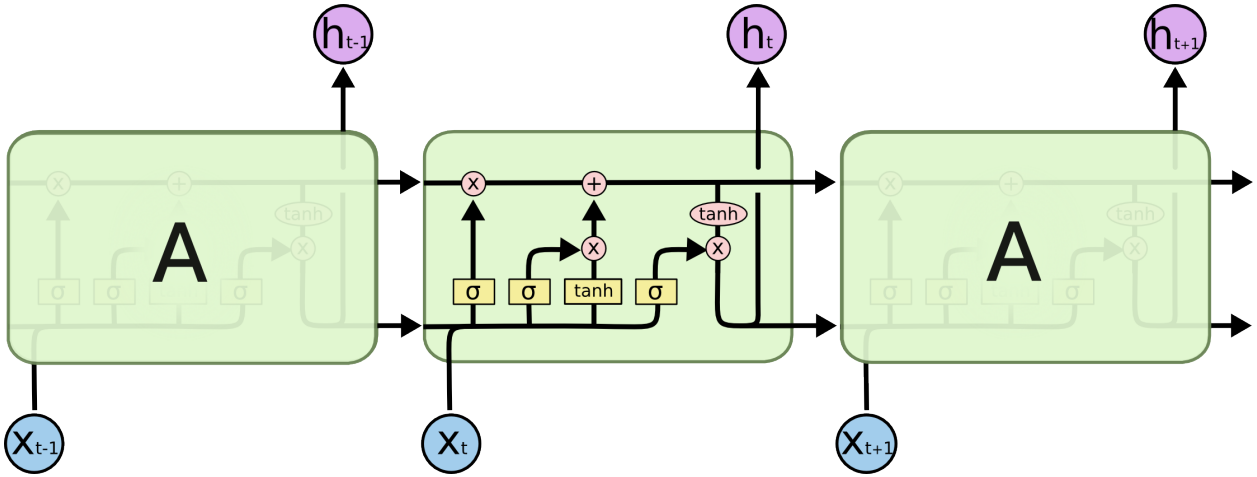
LSTMs/GRUs

A problem with RNNs is that suffer from vanishing/exploding gradient. It turns out that $\frac{\partial h_t}{\partial h_1} = \prod_{i=2}^{t} \frac{\partial h_i}{\partial h_{i-1}}$.

LSTMs and GRUs are modifications to the RNN architecture that enable medium-term dependency learning.

They have a gated residual hidden state called the "cell state". The cell state gates are controlled by the hidden layer and the input. The cell state determines the output and is managed by gates.

LSTMs/GRUs



- The forget gate lets the cell state know what it can forget
- The update gate lets the cell state know what parts of the input to remember
- The output gate restricts which parts of the cell state are updated
- The next hidden state is equal to the output

It turns out that when the forget gate is 1, the long-term gradient dependency disappears.

LSTMs/GRUs

GRUs are a modification of LSTMs. They combine the cell state and hidden state into one internal state, and also merge the input and forget gates into an "update gate".

Choosing a RNN

There are many other variants of LSTMs. Picking one is often treated as a hyperparameter that you can search over. GRUs have recently become very popular.

Read *"Understanding LSTM Networks"*.

Attention

Attention networks were designed to work like a special case of a "question-answering network".

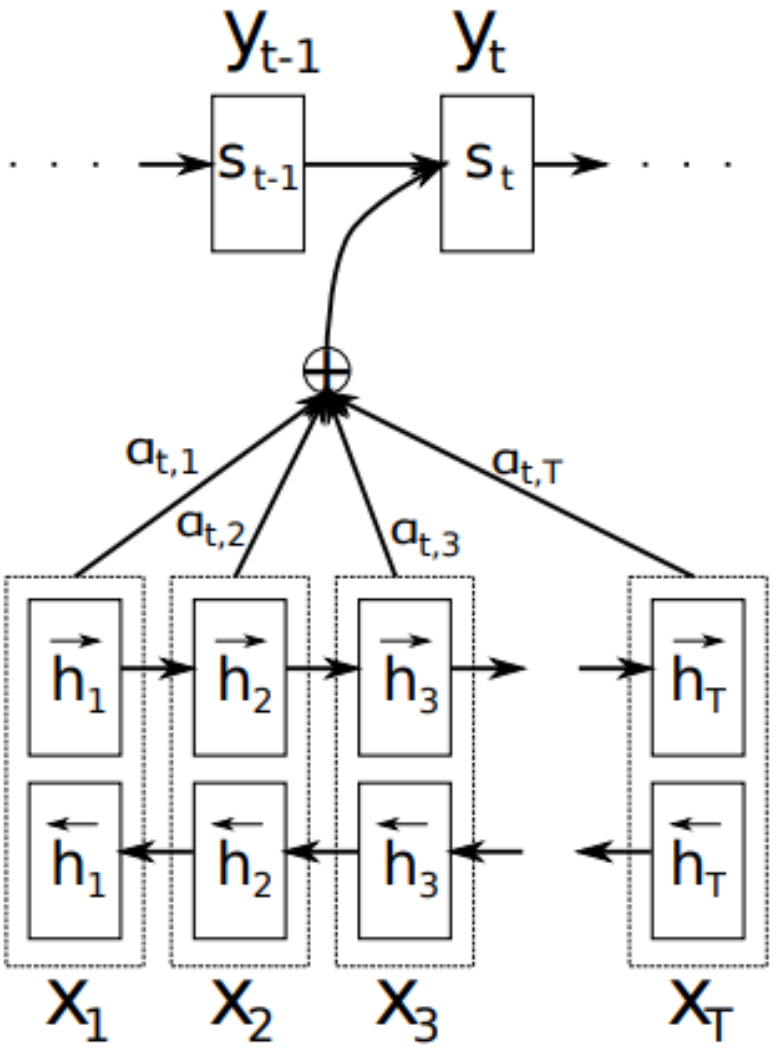
Question-answering networks have some internal knowledge and are able to answer questions about it. Attention networks use some sequential RNN output as their internal knowledge and ask the question "what are the important parts of that sentence?"

They construct a representation of the sentence as a weighted sum of their knowledge base, weighted by perceived importance.

We do this in reading order, so we end up with a sequence of the same length as our input.

Read Lilian Wang's blog post *"Attention? Attention!"*

Attention example



$\alpha_{t,i}$ represents the relation of word x_i to the hidden state s_{t-1}.

There's no definitive way to compute α, but often we compute some score for every $x_{t,i}$ with respect to s_{t-1} (say, using a feed-forward NN) and then apply softmax.

The hidden state at each timestep knows the previous hidden state s_{t-1}, the previous output y_{t-1} and the weighted sum

$$c_t := \sum_{i=1}^{T} \alpha_{t,i} x_i$$

Back to language models

How do we know if these models actually work?

An example of a very generic language task is language modeling: Given a sentence context, output a missing word. This is an unsupervized problem.

Language modeling example

"Today I'm giving a ____ on natural language processing."

The simplest formulation just asks you to guess the next word in a sentence.

RNNs perform very well on these tasks, even better than hidden Markov models or CRFs.

Embeddings

Much like the early layers of CNNs learn generic vision primitives, the early layer of word models are mostly concerned about learning what words mean.

We give words a dense representation by mapping each word to a trainable vector.

Embedding other things

You don't need to do this with words. You can also use dense layers on n-gram or character models.

You can then use embeddings trained on language modeling as a base for any sequence model. Pre-trained embeddings drastically increase performance in most cases.

Extra embedding fun

A brief justification for why embeddings work so well is because they capture semantic relationships as linear relationships.

For example, Warsaw - Poland + Germany $\approx$ Berlin.

The idea here is that (Warsaw - Poland) captures the idea of "capital city-ness", and that countries are all related to their capital by this relation.

Debiasing embeddings

Embeddings can sometimes learn problematic relationships. Read *"Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings"*.

Table of Contents

Problem statement

You're working for the NSA/CSA, spying on the Canadian company NeoTech. You received an anonymous leak of email printouts that detail plans to sell NeoTech technology to an unknown foreign government. Certain key details, such as the sender, were redacted from the printouts. You want to find out who sent these emails so you can "encourage" them to disclose more data. However, NeoTech recently upgraded to a secure email platform.

You have access to thousands of old NeoTech emails that you skimmed from before they secured their mail platform. Can you create a model on these emails to read an email and determine the sender from a list of NeoTech employees?

Model design

Some key facts about the problem: This is a document classification problem:

- Input data is entire emails
- Multiclass classification – one class for each NeoTech employee
- We'll need a many-to-one sequence model

I'm going to save you some trouble and design the model for you. We'll make a 3-layer GRU with word embeddings.

I'm also going to train the model for a few epochs and then give you the weights as initialization so you can get good results quickly.

Let's code!

LIVE CODING TIME