NLP Reading Group — Meeting 1

Matt Smith

UW Data Science Club

28 February, 2019

Welcome

Welcome to the NLP reading group! :) This is a weekly reading group hosted by UWDSC in an attempt to appeal to intermediate members, as well as to help members gain experience reading papers. I will try to cover a mixture of basic theory and applications.

The first week I will mostly be feeding you the information, but as we get more comfortable reading papers I will start presenting selections from the papers.

For this reason I encourage you to try and read the papers we cover. While I can't cover all of the background needed to read the papers, I will try to cover what is most essential.

The first week we will be covering more in-depth some of the ideas presented in the NLP Workshop. Partly because I glazed over the details in the workshop, and partly because these ideas are very important to know.

Coverage

Word Representations

An NLP model takes sequences of language units (usually words or letters) as an input and returns output(s).

We'll consider the case of supervised learning on word sequences.

We need a way to represent words as data.

A really naive way to do this is by using the binary representation of words.

Practice

Explain why this is a terrible idea.

Curse of Dimensionality

Definition: Vocabulary

A *vocabulary* is a set of words you want to handle.

If your vocabulary size is 40000 and you want to handle sequences of 10 words, you have

$$40000^{10} = \sim 10^{46}$$

possible sequences as your input.

It's very likely that we will encounter a sequence we have never seen before in our training data.

Why is this a problem?

One-hot Representations of Words

The traditional way of representing words is using a one-hot vector.

A vocabulary with N words is a set $\{w_1, w_2, w_3, \ldots, w_N\}$.

A one-hot representation converts your vocabulary to a N-dimensional vector space where each word is represented as the ith standard basis vector e_i.

Example

If $N = 5$, the second standard basis vector is $e_2 := \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$.

Problems with One-hot Words

What issues can you think of with this?

Problems with One-hot Words

What issues can you think of with this?

- The memory cost of such a representation is huge
- All of our words are orthogonal in our representation
- Our model needs to learn the relationships between words
- We might not have a lot of data to learn these representations

Orthogonal representations in particular hamper generalization when we consider the Curse of Dimensionality.

Unsupervised Learning

Thanks to the internet, we have access to lots of unlabeled text.

Can we learn something about relationships between words using this unlabeled data?

Distributed Representations of Words

Key idea of this paper:

Represent words as "embedded points" or "embeddings" in $\mathbb{R}^h$ for some h.

E.g. $h = 300$ would give us *300-dimensional word embeddings*.

In practice we can embed other things such as characters and n-grams.

Distributed Representations of Words

Wait a minute...... This is worse!

Instead of N^{10} possible inputs to our model, there are now infinite possible inputs.

It turns out in practice that this is not a problem because:

- The only inputs we care about are the N points we have embedded in $\mathbb{R}^N$
- The word representation has nice local properties that allow for good generalization

Nice local properties include things like smoothness, convexity, quadraticity and Lipschitz continuity of the gradient.

Distributed Representations of Words

This paper doesn't prove any of these nice local properties, only notes that they exist in practice.

We'll see in a later paper that the properties of word embeddings are very very nice if they're constructed in a natural way.

Language Modeling

Language Modeling is a set of tasks that we use to understand language in a very abstract way.

The most common kind of language model is a model that attempts to predict the next word given the previous words in a sentence.

Given a sentence $w_1 w_2 \dots w_k$, we write w_a^b to refer to the sentence fragment $w_a w_{a+1} \dots w_b$.

Language modeling wants to find the function $\Pr(w_i \mid w_1^{i-1})$.

We can find the probability of any sentence,

$$\Pr(w_1^k) = \prod_{i=1}^{k} \Pr(w_i \mid w_1^{i-1})$$

Language Modeling

The paper does this using a dense neural network, making the simplifying assumption that

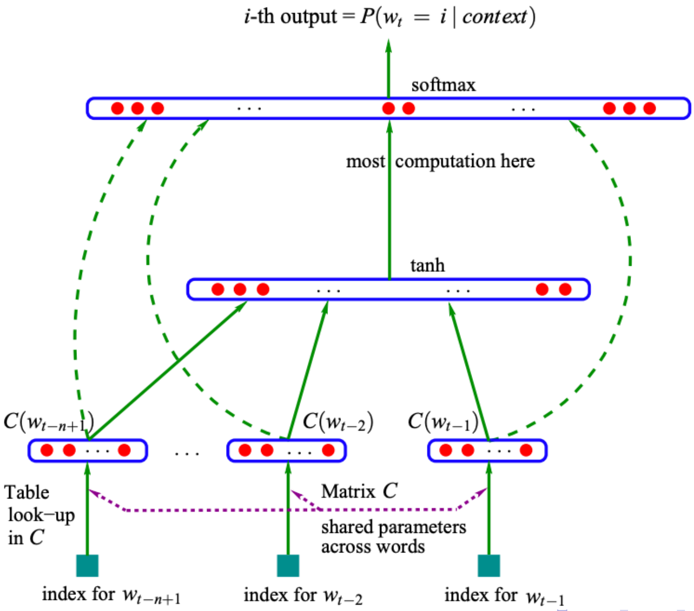
$$\Pr(w_i \mid w_1^{i-1}) \approx \Pr(w_i \mid w_{i-1-n}^{i-1})$$

for some n sufficiently large (they use $n = 5$).

This bounds the size of their input, which is necessary when using a regular neural network.

It turns out this simplifying assumption stops us from capturing dependencies in language. Nowadays people just use RNNs and can avoid this problem, although RNNs still have difficulties with long-term dependencies.

Model


i-th output = $P(w_t = i \mid context)$
softmax
most computation here
tanh
$C(w_{t-n+1})$
$C(w_{t-2})$
$C(w_{t-1})$
Table look–up in C
Matrix C
shared parameters across words
index for w_{t-n+1}
index for w_{t-2}
index for w_{t-1}

Word Embeddings

Not presented in this paper, but it turns out that the matrix C serves as a very good initialization for other language tasks.

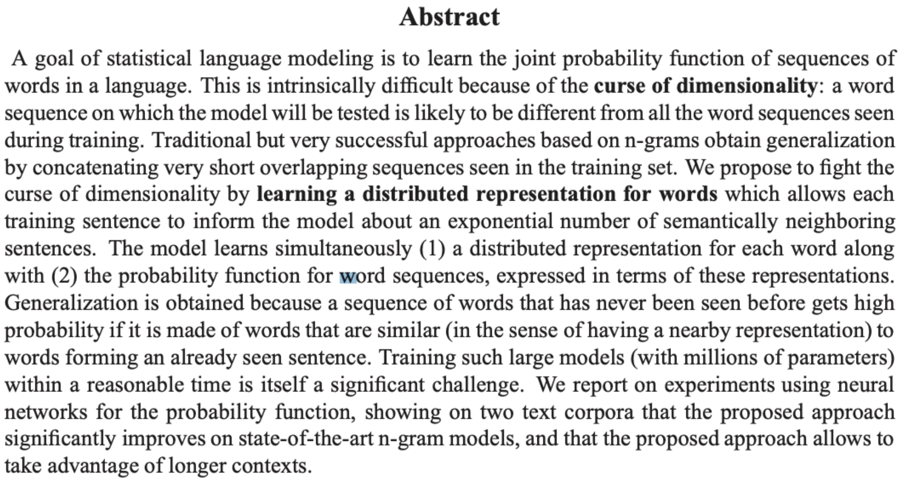
This is the concept of word embeddings, and it is one of the most important topics in NLP.

Usually lots of data is required before a model starts understanding language.

We can learn word embeddings from unsupervised data and then transfer that understanding to a supervised model.

This means that we don't need tons of supervised data to understand language, we only need to understand our task.

Abstract



Abstract

A goal of statistical language modeling is to learn the joint probability function of sequences of words in a language. This is intrinsically difficult because of the **curse of dimensionality**: a word sequence on which the model will be tested is likely to be different from all the word sequences seen during training. Traditional but very successful approaches based on n-grams obtain generalization by concatenating very short overlapping sequences seen in the training set. We propose to fight the curse of dimensionality by **learning a distributed representation for words** which allows each training sentence to inform the model about an exponential number of semantically neighboring sentences. The model learns simultaneously (1) a distributed representation for each word along with (2) the probability function for word sequences, expressed in terms of these representations. Generalization is obtained because a sequence of words that has never been seen before gets high probability if it is made of words that are similar (in the sense of having a nearby representation) to words forming an already seen sentence. Training such large models (with millions of parameters) within a reasonable time is itself a significant challenge. We report on experiments using neural networks for the probability function, showing on two text corpora that the proposed approach significantly improves on state-of-the-art n-gram models, and that the proposed approach allows to take advantage of longer contexts.

Coverage

Word2vec

word2vec is a series of 3 Google papers from 2013 about word embeddings.

They introduce two new, natural methods of generating word embeddings.

They then show that these methods generate word embeddings with very nice properties.

Specifically, semantic relationships are captured as linear relationships in the embedding space.

These results are summarized in the popular blog post "*The amazing power of word vectors*".

Language Modeling

We saw before one way of language modeling was to try and predict a word given the previous words in a sentence.

This intuitively requires some understanding about human languages. This is what I mean when I say a language modeling technique is ‘natural’.

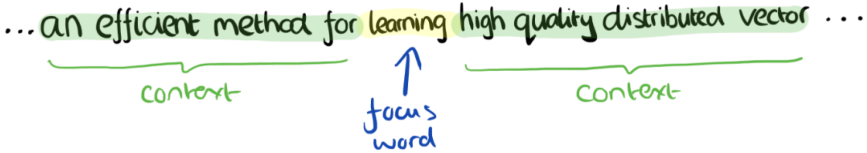
Mikolov et al. introduce two new techniques: Continuous Bag-of-Words and Continuous Skip-gram methods.

Bag-of-Words Models

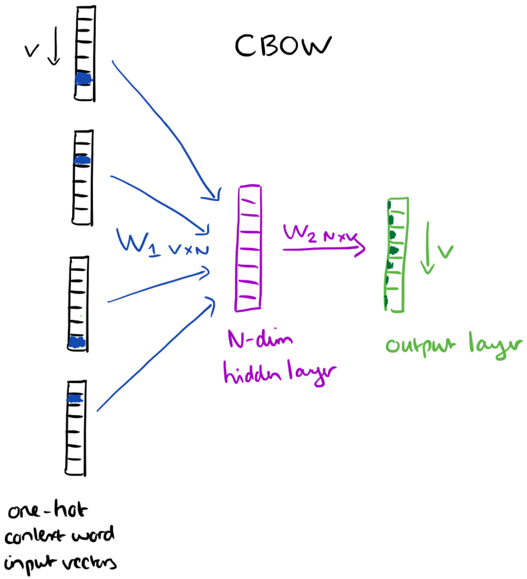
Generically, language models aim to predict a word or sequence of words given a *sentence context*.

In the traditional language modeling task, the context is the previous words in the sentence.

Bag-of-words uses a number of surrounding words as a context to predict the focus word.


...an efficient method for learning high quality distributed vector ...
context
focus word
context

Bag-of-Words Models



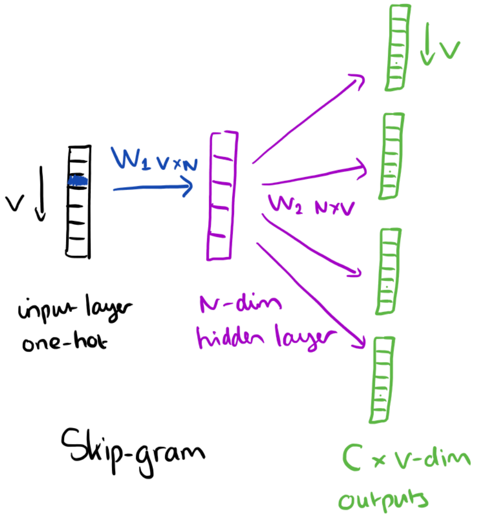
Skip-Gram Models

Skip-gram embeddings do the opposite of BoW.

Given a word, predict the surrounding context.

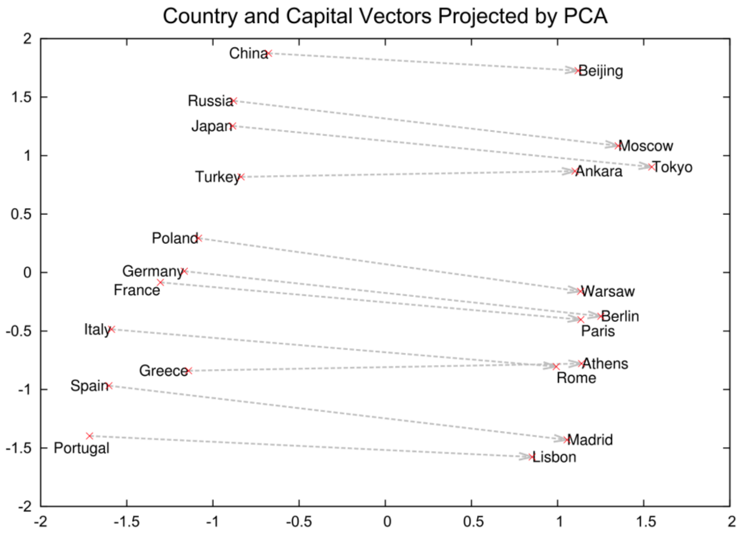
Both models include a “projection” layer that maps words from one-hot representations to a continuous representation. We can use this as the embedding matrix for other projects.

Skip-Gram Models



Properties of the Embedded Space

Semantic relationships are linear relationships in the embedded space.



Properties of the Embedded Space

We can use this to perform “semantic math”.

If e_X is the word embedding for the word X, then we might expect that

$$e_{king} + (e_{woman} - e_{man}) \approx e_{queen}$$

The idea is that the difference between $e_{king} - e_{queen}$ is the same difference between man and woman.

Properties of the Embedded Space

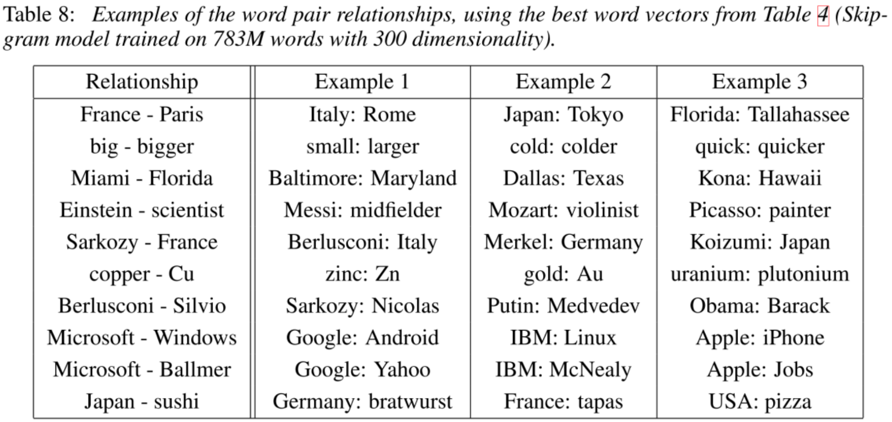


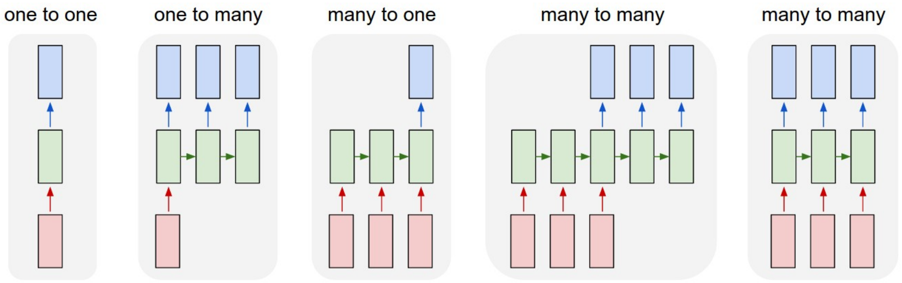
Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Coverage

Review of RNNs

There's many things we want to do with sequence models that make traditional neural networks insufficient.



We instead use RNNs, which process inputs sequentially while passing a hidden state forward in time.

A Modern Language Model

You might know that RNNs suffer from vanishing/exploding gradients.

$$u^{(t)} = W_h h^{(t-1)} + W_x x^{(t)} + b$$

$$h^{(t)} = \tanh(u^{(t)})$$

$$\frac{\partial h^{(t)}}{\partial h^{(1)}} = \prod_{i=2}^{t} \frac{\partial \tanh}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial h^{(i-1)}} = \prod_{i=2}^{t} \tanh'(u^{(i)})\, W_h$$

This is exponential blowup/vanishing dependent on W_h and the sequence length.

A Modern Language Model

A common solution is to use LSTMs. There are many variants, but the common idea is to use gated residual connections to control the gradient.

A typical LSTM has a residual connection called the cell state or c in addition to the normal hidden state h.

We define the forget gate

$$f^{(t)} := \sigma(W_f[h^{(t-1)}, x^{(t)}] + b_f)$$

the update gate

$$u^{(t)} := \sigma(W_u[h^{(t-1)}, x^{(t)}] + b_u)$$

and the output gate

$$o^{(t)} := \sigma(W_o[h^{(t-1)}, x^{(t)}] + b_o)$$

A Modern Language Model

At each timestep, we create a "candidate" cell state that we want to add to the cell state,

$$\tilde{C}^{(t)} = \tanh(W_C[h^{(t-1)}, x^{(t)}] + b_C)$$

and update according to the rule

$$C^{(t)} = f^{(t)} * C^{(t-1)} + u^{(t)} * \tilde{C}^{(t)}$$

Notice that $*$ here refers to element-wise multiplication.

We then output

$$y^{(t)} = h^{(t)} = o^{(t)} * \tanh(C^{(t)})$$

A Modern Language Model

I won't cover the math for this, but these gated residual connections allow us to shift the exponential growth we saw earlier to depend only on the forget gates.

That is,

$$\frac{\partial c_i^{(t)}}{c_i^{(1)}} = \prod_{j=2}^{t} f_i^{(t)}$$

We usually initialize the LSTM with a 'forget bias' to ensure the falues of $f^{(t)}$ are high enough. Most deep learning frameworks will do this for you.

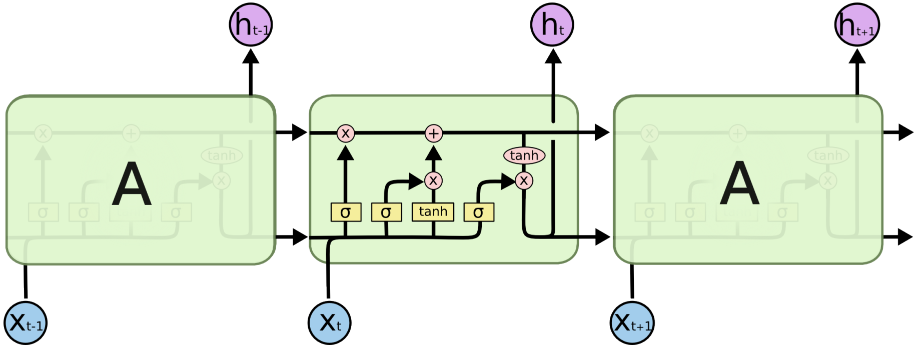
LSTM Variants

There are different kinds of LSTMs that are used in different scenarios.

Some add “peephole” connections that allow the gate computation to see the cell state. Other variants like the update/forget gates such that $u^{(t)} = 1 - f^{(t)}$.

GRUs are a popular variant that couple the update/forget gates and also merge the cell state and the hidden state.

LSTM



Coverage

Character Modeling

The paper mostly focuses on character modeling using LSTMs.

Character modeling looks similar to word modeling: given the previous characters in a sentence, we want to predict the next character.

Example

Say our training data has “I love RNNs.”. One training sample might be the input “I lov” and the expected output one-hot vector e_e.

While we train our model on one-hot targets, in reality the model ends up outputting probabilities for the next character. If we sample the next character according to this distribution, we can generate text.

Show and Tell

Let's look at some of the examples in the 'paper' together.

Practice and Exercises

As practice, find an implementation for char-rnn in your toolkit of choice (I use keras and found one by Googling "char-rnn keras") and feed it some interesting text. If your model does really well or you notice some cool patterns, let me know and I'll present your results next time (or you can... or I can just collect the results in a Facebook post).

Exercises, in order of best to worst:

1. Implement your own char-rnn. I don't care if you use a framework or do something/everything from scratch
2. Compare the performance of single-layer RNN/LSTM/GRUs to hidden Markov models. What about multi-layer networks?
3. Implement backpropagation through time in a RNN
4. Implement backpropagation through time in an LSTM