Supervised Learning: Intro to Natural Language Processing and Vector Space Embedding

Noah Simon & Ali Shojaie

July 14-16, 2021
Summer Institute in Statistics for Big Data
University of Washington

Vector Space Embedding

Sometimes we would like to use complex objects as predictors

We will focus, in particular, on words/sentences/documents

A Simple Question

Suppose we would like predict if a pt will be readmitted to the hospital within 1 month based on a doctor's discharge note

Suppose we had discharge notes, and followup on 1000 pts

How would we build a classifier for this?

The problem

How do we turn the document into a numeric feature-vector?

We can do this the easy way, or the hard way...

The Easy way

We consider all words that could have been included in the document

$$p \sim 200,000$$

And assign to each document, a binary p-vector with

- 0 indicating absence of a word
- 1 indicating presence of a word

We then use these binary vectors as our numeric features

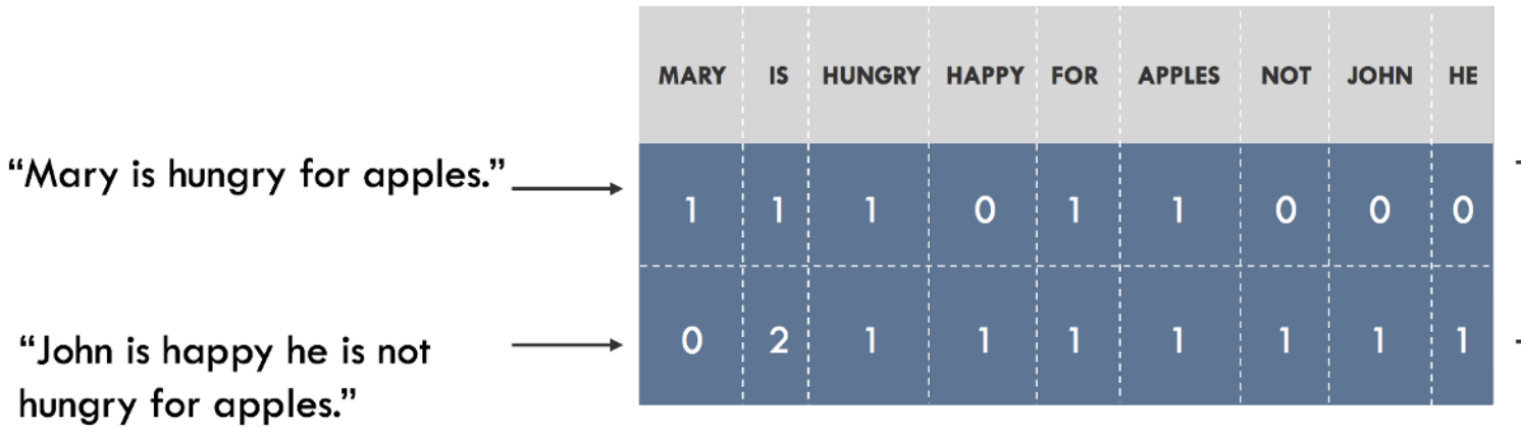
The Easy way extended

Rather than just presence/absence, can instead use

- count
- frequency
- normalized frequency

"term-frequency-inverse-document-frequency"

The Easy way example (count)



borrowed from "https://blog.insightdatascience.com/how-to-solve-90-of-nlp-problems-a-step-by-step-guide-fda605278e4e"

Issues - Similarity

This approach struggles with "similar words"

"Patient is in great health"

"Patient has excellent health"

No way to apriori know that "excellent" and "great" are similar.

Issues - Context

This approach loses context!

"Patient does have cancer; Not happy"

"Patient does not have cancer; Happy"

Both of these have the same feature vector... and yet would indicate opposite prognosis.

A quick fix

Rather than using words as "objects"...

and counting words,

Instead use pairs of words (2-grams) as "objects"

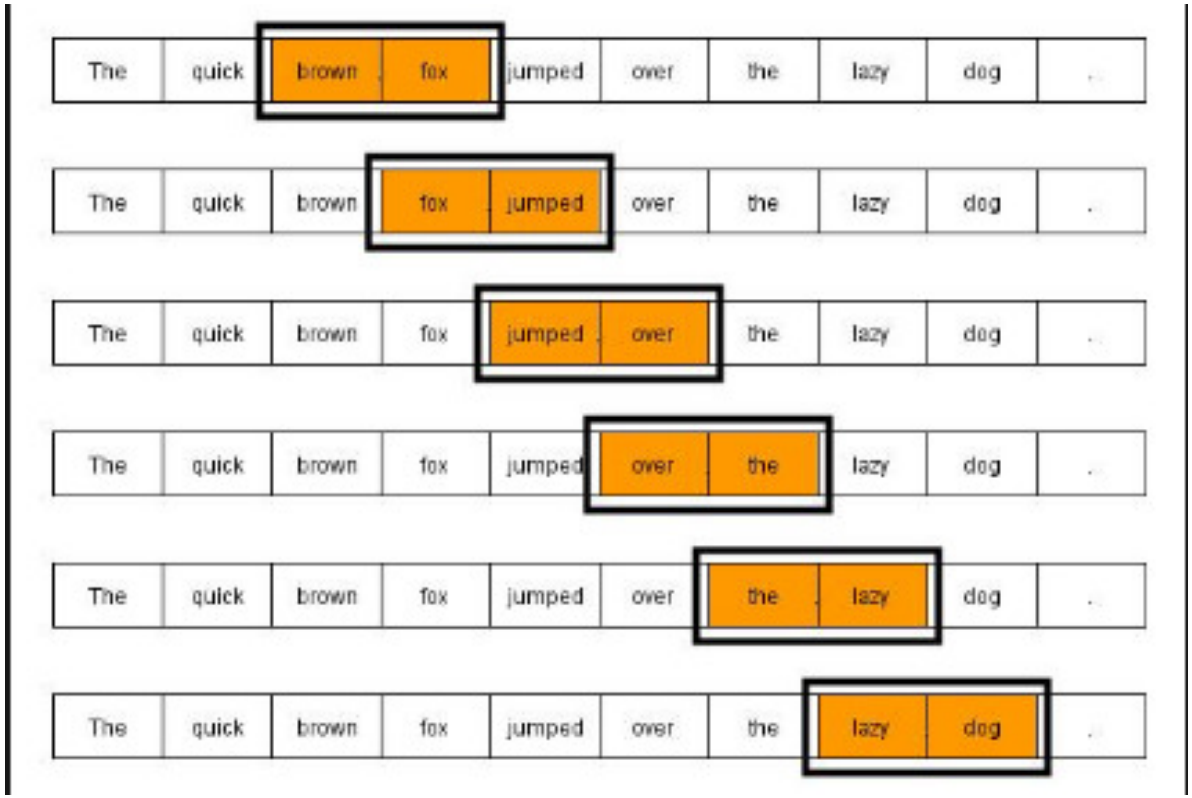
and count/identify presence of pairs of words.

"Patient does have cancer; Not happy"

"objects" are: "patient does", "does have", ..., "not happy"

n-grams

Example with 2 grams.



borrowed from "https://www.depends-on-the-definition.com/introduction-n-gram-language-models/"

Issue

We keep a bit more context...

However, now we have a HUGE number of potential "objects"

p^2 potential objects

If we do not have many samples, will have almost no shared non-zero features

Similar (but not identical) words still create problems.

The hard way

What we would like to do:

Assign a numeric vector to each word such that "nearby" words are similar with respect to

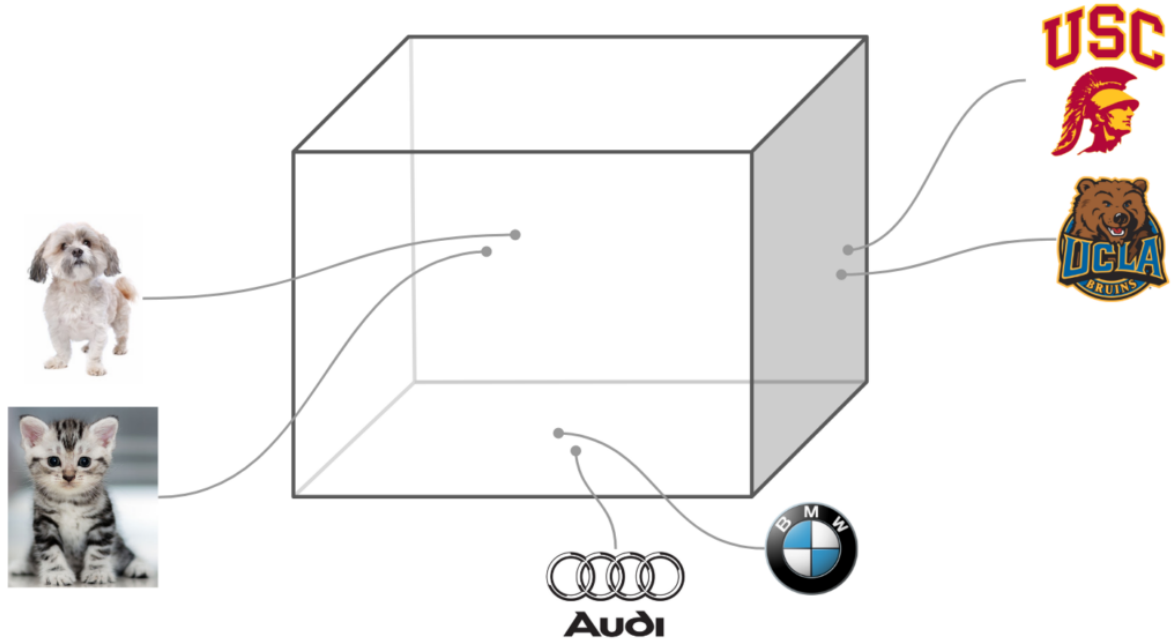
- meaning
- part of speech (or eg. tense)

Perhaps...

$$\texttt{cat} \to (1, 0.1, 0.3)$$
$$\texttt{dog} \to (1.1, 0.13, 0.34)$$
$$\texttt{typing} \to (-2.3, 3.1, -1.1)$$

Embedding Picture

A cartoon of embedding in $\mathbb{R}^2$



borrowed from "http://techblog.gumgum.com/articles/deep-learning-for-natural-language-processing-part-1-word-embeddings"

What does this do for us?

If we already have an "embedding"...

(way of assigning numeric vectors to words)

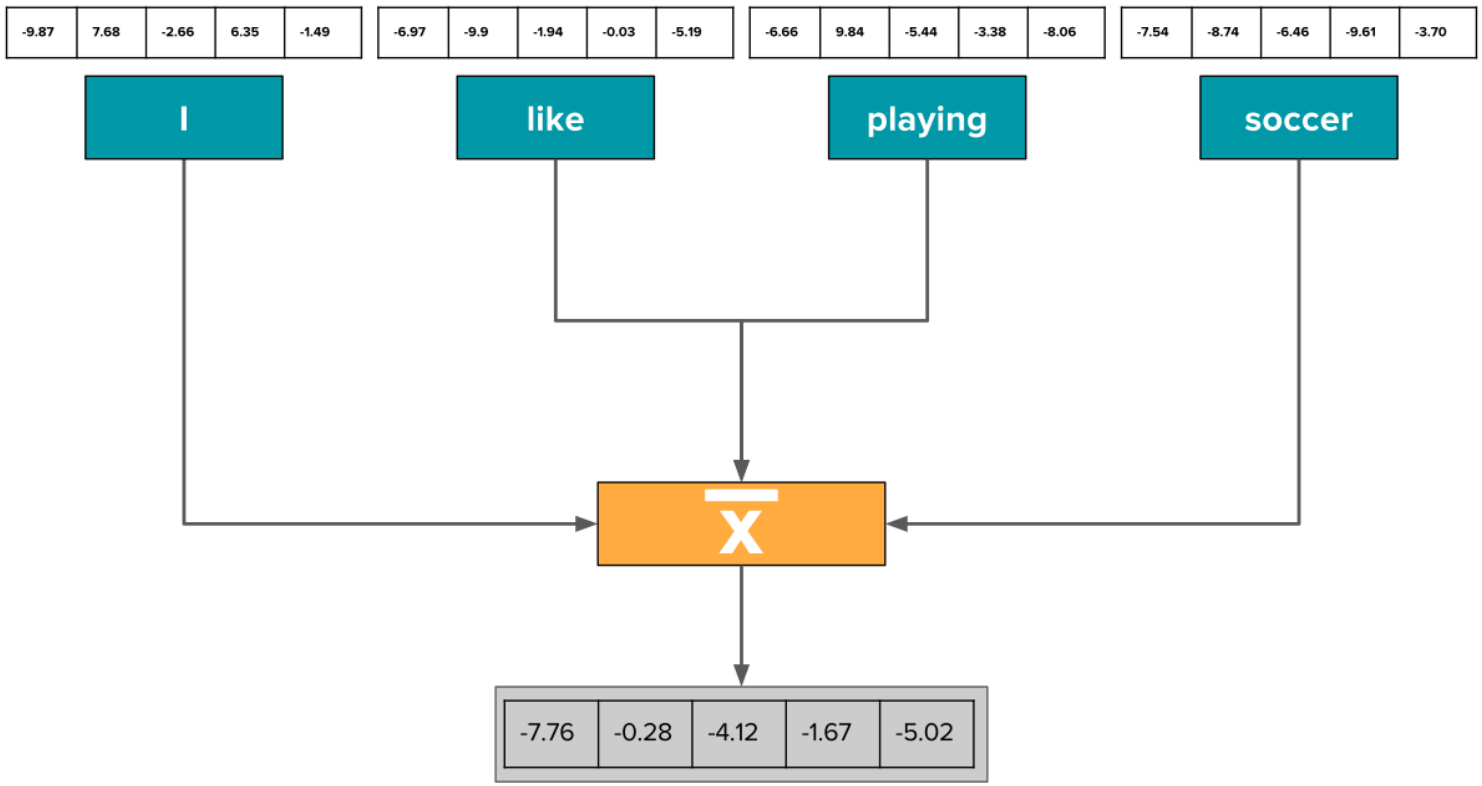
Can use these vectors as features in a predictive model...

Simplest version uses "average" of word-vectors from a sentence as input features to a predictive model

In some cases this is good enough (eg. search engines)

Example

Picture of "averaging" to get input features



borrowed from "https://techblog.gumgum.com/articles/deep-learning-for-natural-language-processing-part-2-rnns"

A More Nuanced Approach

That is still not how language works...

As you read a document, your state of understanding changes

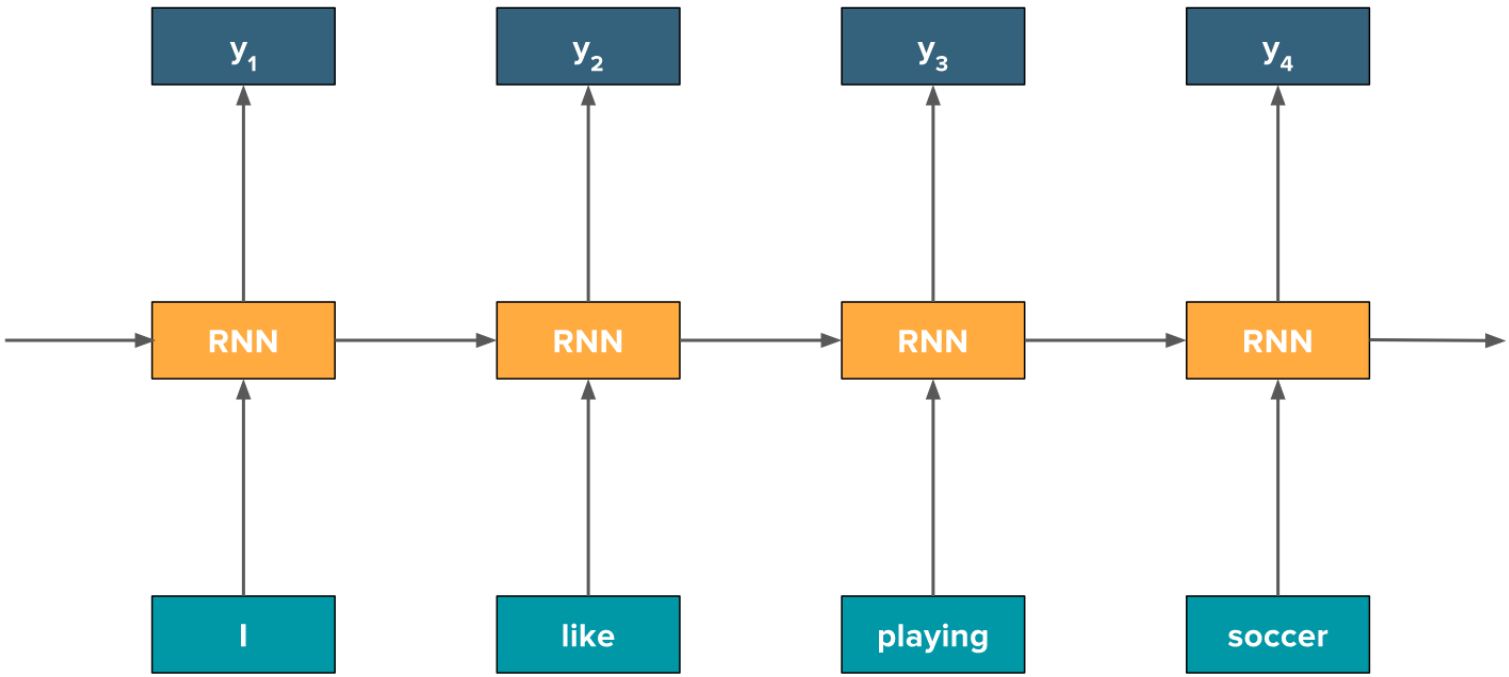
At each new word/sentence one updates their current state based on the new word/sentence

This update involves a complex interaction between the current state and the new word/sentence.

A More Nuanced Approach — RNN edition

This is precisely what a recurrent neural network models!

Combo of RNN with embeddings, allows powerful predictive model!

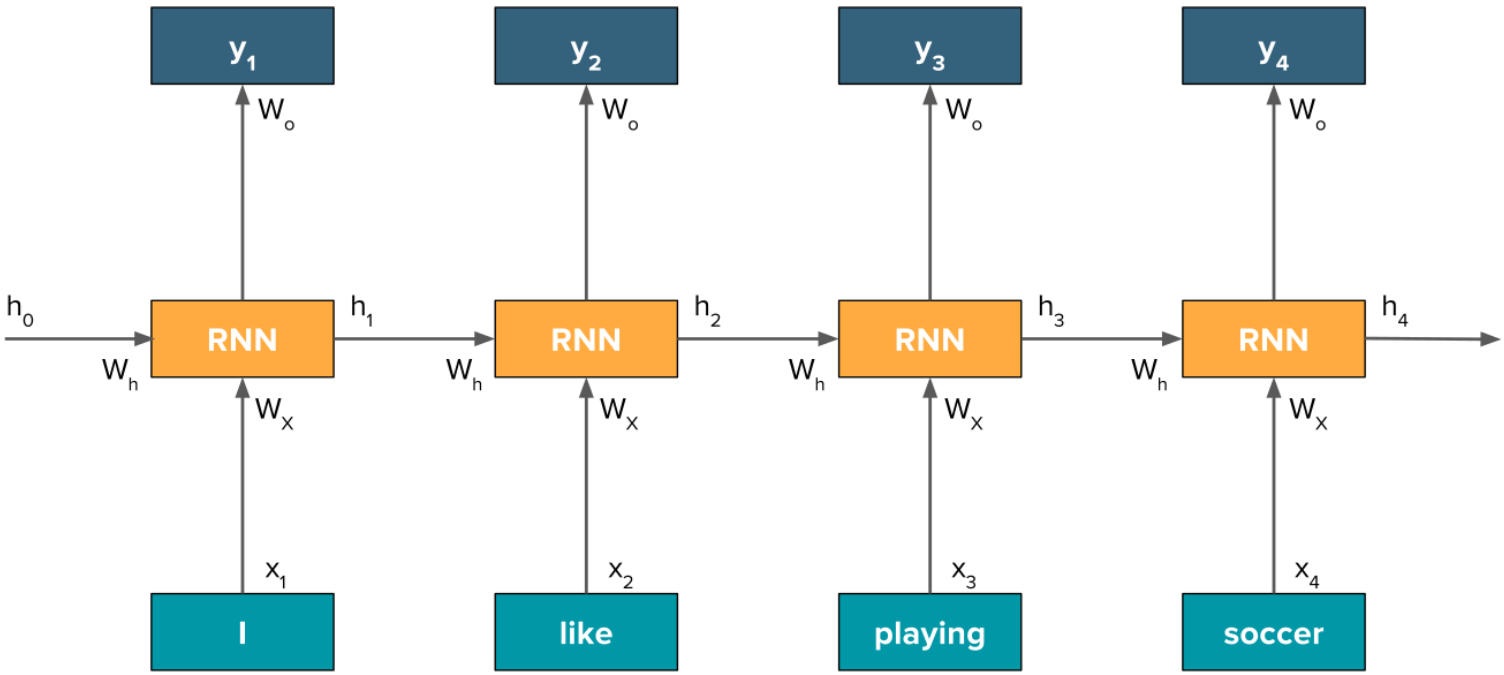


borrowed from "https://techblog.gumgum.com/articles/deep-learning-for-natural-language-processing-part-2-rnns"

A More Nuanced Approach — RNN edition

This is precisely what a recurrent neural network models!

Combo of RNN with embeddings, allows powerful predictive model!

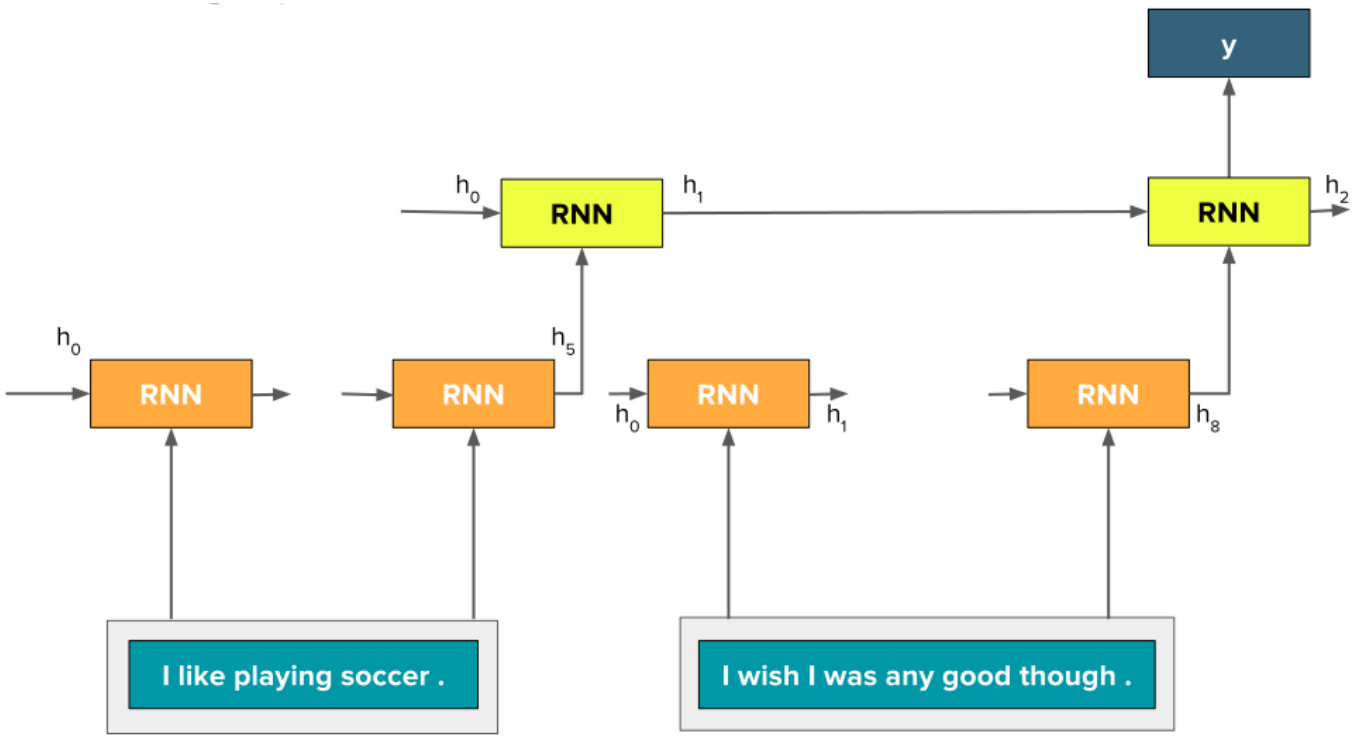


borrowed from "https://techblog.gumgum.com/articles/deep-learning-for-natural-language-processing-part-2-rnns"

A More Nuanced Approach — RNN edition

This is precisely what a recurrent neural network models!

Combo of RNN with embeddings, allows powerful predictive model!



borrowed from "https://techblog.gumgum.com/articles/deep-learning-for-natural-language-processing-part-2-rnns"

RNNs + VSE

The combination of RNN and VSE is extremely powerful

Has been used in (among other things)

- Machine Translation
- Speech Recognition
- Spell Checking
- Semantic Parsing + Question Answering
- Document Summarization

How to Calculate Embeddings

We have discussed what to do with embeddings...

But how do we possibly calculate embeddings???

Self Supervised Learning - I

The key is [what I call] Self Supervised Learning...

We would like the embeddings to preserve context

What does that mean?

Well, given the beginning and end of a sentence, it is not so hard for a human to roughly predict the middle word

We would like our embedding to preserve that:

Given embeddings of words in the beginning and end of the sentence, we would like to be able to predict the middle word in the sentence

Self Supervised Learning - II

Learning embeddings will involve two objects:

- Encoder: This maps words to vectors

  (Gives our embeddings)

- Decoder: This maps [sequences of] encoded vectors to a word

  (Predicts word from embeddings of surrounding words)

The decoder is a nuisance — but we cannot learn the encoder without it!

This is like in PCA — need to learn both PCs and Loadings simultaneously

Self Supervised Learning - III

We need to simultaneously learn both! Generally we solve a problem that looks like

$$\text{argmin}_{E,D} \sum_{i=2}^{N} \ell(W_i, D\left(E(W_{i-1}), E(W_{i+1})\right)$$

where

- $D\left(E(W_{i-1}), E(W_{i+1})\right)$ is a prediction for word i, based on words $i-1$ and $i+1$
- $\ell(W, \hat{W})$ is the loss from predicting $\hat{W}$ when the truth is W
- Sometimes more context is used

Self Supervised Learning - IV

Often the decoder doesn't just predict a *single* word, but gives a predicted probability to all possible words. Can use a fancy "multinomial" model here.

One version of most popular methodology, `word2vec`, uses a linear multinomial decoder [1]

[1] modulo some minor caveats!

Self Supervised Learning - IV

This methodology is powerful because we do not need "external supervision" to train it (eg. no one needs to label any documents or sentences).

By using later parts of the sentence to "supervise" the embedding and prediction process, we can learn embeddings from just a large corpus of text.

You can download pre-calculated embeddings using, eg. all of wikipedia!

Some care is required though — embeddings learned in one context may not do well in another!