Lecture 20
Word Vector Embeddings

18 April 2016

Taylor B. Arnold
Yale Statistics
STAT 365/665

Yale

Notes

- Final two problem sets 8 and 9 are now posted
- You have all the tools to do 8 already; we'll cover what you need for 9 these last two weeks

What problems are neural networks good at?

We talked briefly about why neural networks are so powerful in certain settings, but have not circled back to this general topic since our deep-dive into computer vision and CNNs.

- neural networks can be very good for general learning tasks
- particularly useful for semi-supervised and multiclass problems with a high number of classes
- relative weakness comes when the variables are the correct representation for learning, such as looking at genetic data

What problems are neural networks good at?

We talked briefly about why neural networks are so powerful in certain settings, but have not circled back to this general topic since our deep-dive into computer vision and CNNs.

- neural networks can be very good for general learning tasks
- particularly useful for semi-supervised and multiclass problems with a high number of classes
- relative weakness comes when the variables are the correct representation for learning, such as looking at genetic data

You will have a chance to test these ideas on problem set 8, where you will apply neural networks to the Chicago Crime dataset.

A personal opinion

When we have general unstructured data, basically anything other than text, video, sound, or images, building predictive neural network models is relatively fast and easy. Expensive CNNs and RNNs (we'll see this next class) are not needed, and so the model structure is a much more straightforward.

I think there is a lot of potential for applying plain neural networks to these types of problems. Unfortunately, the majority of researchers outside of NLP or computer vision are not familiar enough with neural networks and the various platforms to do this type of analysis. *Hopefully, after this class, you will feel comfortable enough with NNs that this does not apply!*

A note about speech processing

Recall from our whirlwind overview of neural networks in image processing that outside of LeNet, neural networks did not become the go-to model for image classification until 2012.

The uses of deep neural networks for speech recognition, what I would describe as the third big application area, came quite a bit earlier. Most people would go back to the paper by Hinton, Osindero, and Teh as the start of this interest:

> *Hinton, G.E., Osindero, S. and Teh, Y.W., 2006. A fast learning algorithm for deep belief nets. Neural computation, 18(7), pp.1527-1554.*

A note about speech processing, cont.

And a really great read on the longer history is given in this paper:

> *Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N. and Kingsbury, B., 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. Signal Processing Magazine, IEEE, 29(6), pp.82-97.*

You will notice that many of the ideas they explored are already out of date compared to the approaches we have seen today.

A note about speech processing, cont.

And a really great read on the longer history is given in this paper:

> *Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N. and Kingsbury, B., 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. Signal Processing Magazine, IEEE, 29(6), pp.82-97.*

You will notice that many of the ideas they explored are already out of date compared to the approaches we have seen today.

I won't talk much more about speech processing, primarily because I think of it as a fairly niche problem. A lot of people will want to use the algorithms that come out of speech processing, but most practitioners and researchers will be simply using the transcriptions as a starting point.

Onwards to text processing!

We are now going to transition into talking about using neural networks for natural language processing.

I have found that there are far fewer resources for using neural networks for text analysis. The most comprehensive, are the excellent lecture notes by Richard Socher:

http://cs224d.stanford.edu/

I'll pull quite frequently from these over the next two weeks.

Straight to the point

As we are already quite familiar with neural networks, we can skip a lot of the introductory material from their notes and move directly into the two main problems with text and NNs.

The reference problem we'll start with is sentiment analysis. We assume we have a large corpus with small snippets of text tagged with either a 1 (positive sentiment) or 0 (negative sentiment). Our goal is to build a classifier that can predict sentiment on new snippets of text. More concretely, we can look at the IMDB Movie reviews dataset provided by keras.

I caught this movie about 8 years ago, and have never had it of my mind. surely someone out there will release it on Video, or hey why not DVD! The ford coupe is the star.....if you have any head for cars WATCH THIS and be blown away.

I think I will make a movie next weekend. Oh wait, I'm working..oh I'm sure I can fit it in. It looks like whoever made this film fit it in. I hope the makers of this crap have day jobs because this film sucked!!! It looks like someones home movie and I don't think more than $100 was spent making it!!! Total crap!!! Who let's this stuff be released?!?!?!

Would that more romantic comedies were as deftly executed as this one? I never thought anything as mundane as the simple sale of a music box could leave me catching my breath with excitement. Margaret Sullavan makes a marvellous saleswoman, and she and James Stewart always brought out the best in each other. This movie sports what I think is Frank Morgan's most winning performance, and with "The Wizard of Oz" and "Tortilla Flat" under his belt, that is saying a lot. The way he finds a Christmas dinner partner left me giddy with joy. Director Ernst Lubitsch might have thought "Trouble In Paradise" his favorite, but this one he must surely consider a triumph. With some of the wittiest dialogue American movies of the 30's has to offer.

That's not the sound of bees, that's the effect induced by watching this extremely long, extremely boring, badly acted movie. How I ever made it through all 3 1/2 hours without falling asleep I'll never know. The plot is simple...3 thoroughly unlikable morons talk about sex for 3 1/2 hours. And you thought Rohmer was deadly. This is even worse, if that's possible. I must really be a masochist if I could watch this entire movie without turning it off...or killing someone.

Linear estimators

To solve this problem using regression, we could simply define:

$$X_{i,j} = \begin{cases} 0 & \text{word}_j \notin \text{doc}_i \\ 1 & \text{word}_j \in \text{doc}_i \end{cases}$$

And then fit a linear model:

$$y = X\beta + \epsilon$$

Or a logistic alternative.

Linear estimators

To solve this problem using regression, we could simply define:

$$X_{i,j} = \begin{cases} 0 & \text{word}_j \notin \text{doc}_i \\ 1 & \text{word}_j \in \text{doc}_i \end{cases}$$

And then fit a linear model:

$$y = X\beta + \epsilon$$

Or a logistic alternative.

A penalized variant is typically used, given the dimensions of X, we could shift to 'term-frequency' (counts) instead of indicators, and scale the values in several different ways. Though, the basic idea remains the same.

Penalized estimators, cont.

What subtleties are lost here?

- word placement in the text
- word forms
- negation
- context
- semantic meaning

Neural network structure

How can we use a neural network to solve this problem? The output structure is easy; we'll just use a single output neuron and compare this to to the 0/1 representation of the data using cross entropy (we could also have two outputs that act as dummy variables, as you did on problem set 7).

Neural network structure

How can we use a neural network to solve this problem? The output structure is easy; we'll just use a single output neuron and compare this to to the 0/1 representation of the data using cross entropy (we could also have two outputs that act as dummy variables, as you did on problem set 7).

What about the input layer though? How do we feed a block of text into a neural network? We can use a similar structure to the linear model.

A single word

Let's first simplify the problem and think about just a single word at a time. How can we represent even a single word as an input to a neural network? One approach is to determine a vocabulary of terms, these are all of the words that we want to support in the classification task. Usually we construct this by looking at all of the snippets of text and taking the N-most commonly occurring words. Any words in the texts not in this vocabulary are removed before further training.

A single word, cont.

Once we have this vocabulary, we can represent a single word by an N-length binary vector with exactly one 1:

$$\text{apple} \to [0, 0, 0, \ldots, 0, 1, 0, \ldots, 0]$$

This is called a one-hot representation, from the idea that just one of the bytes is 'hot' (i.e., turned on).

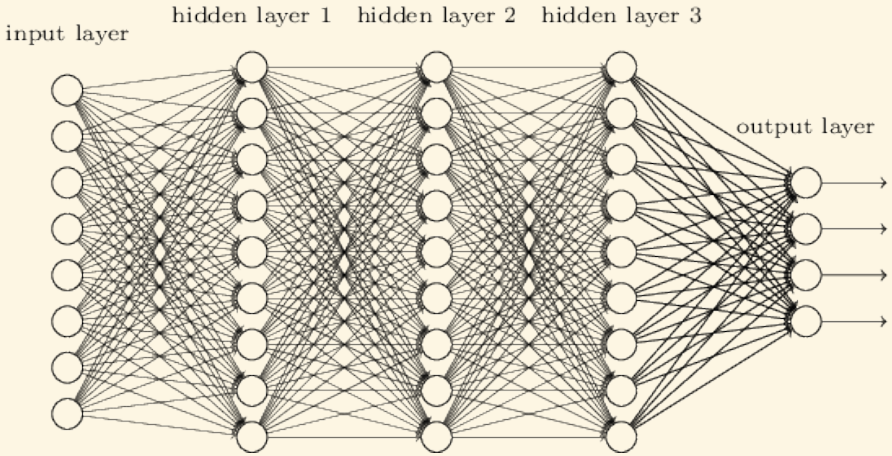
One-hot in neural networks

Suppose we use a one-hot representation as the first layer in a neural network. If this is followed directly by a dense layer with p hidden neurons, the weights in the layer can be defined as an N by p matrix W. In this special case we do not need a bias term, because we already know the scale of the previous layer (0's and 1's).

One-hot in neural networks

Suppose we use a one-hot representation as the first layer in a neural network. If this is followed directly by a dense layer with p hidden neurons, the weights in the layer can be defined as an N by p matrix W. In this special case we do not need a bias term, because we already know the scale of the previous layer (0's and 1's).

For a given set of weights W, because of the one-hot representation, the values of the outputs from the first hidden layer will simply be row j of the matrix W, where j is the index of the input word in the vocabulary.


input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer

Word embeddings

A word embedding is nothing more than a compression of a one-hot representation and a dense hidden layer in a neural network. There is not need to actually create the one-hot vector, and multiply by all of W. We can just go directly from the index of the word in the vocabulary, and read off of the j-th row of W.

Word embeddings

A word embedding is nothing more than a compression of a one-hot representation and a dense hidden layer in a neural network. There is not need to actually create the one-hot vector, and multiply by all of W. We can just go directly from the index of the word in the vocabulary, and read off of the j-th row of W.

What are we doing here, really? The whole idea is to map a word as a vector in a p-dimensional space. Much like the bottleneck layer in an autoencoder, the representation in this space often has a semantic meaning.

Multiple words

This is great, but most of the time we want to work with a collection of words (a document) all as one entity. A simple way to do this is to apply the word embedding to each term, and the collapse (flatten) these into a single long vector.

So if we have T terms in a document and a word embedding with p terms, the output from the embedding layer will be of size $T \times p$.

To be clear, the embedding step is agnostic to the position of the word, much like the shared weights in a convolutional neural network.

Fixed width text

We will persist in one simplification today: assuming that each document has the same number of terms. This is done by truncating at some small number of words (less than what most of the movie reviews are) and filling in any trailing space by a special embedding of zeros.

I. Learning word embeddings with IMDB

Convolutions, again

It turns out we can think of the output of the word embeddings as being similar to the multidimensional tensors in image processing.

For example, consider a word embedding with $p = 3$. We can see this as three parallel 1-dimensional streams of length T, much in the way that a color image is a 2d-dimensional combination of parallel red, green, and blue channels.

Convolutions, again

It turns out we can think of the output of the word embeddings as being similar to the multidimensional tensors in image processing.

For example, consider a word embedding with $p = 3$. We can see this as three parallel 1-dimensional streams of length T, much in the way that a color image is a 2d-dimensional combination of parallel red, green, and blue channels.

In this context, we can apply 1-D constitutional layers just as before: shared weights over some small kernel. Now, however, the kernel has just a single spatial component.

II. Word embedding with 1D Convolutions

II. Reuters classification

Transfer learning

Much like with the lower convolution layers for image classification, there is a great benefit to being able to use a large corpus to learn the word embedding step. And again, we can either treat this embedding as fixed (transfer learning) or just a good starting point (pre-training).

Autoencoders?

We saw that we can either use autoencoders or another classification task to learn the weights in the levels of a convolutional neural network.

It turns out that for text, transfer learning with another classification task does not work quite as well. We instead need something akin to an autoencoder.

Autoencoders?

We saw that we can either use autoencoders or another classification task to learn the weights in the levels of a convolutional neural network.

It turns out that for text, transfer learning with another classification task does not work quite as well. We instead need something akin to an autoencoder.

Does that seem surprising?

How to autoencode

Using a direct autoencoder on a single not make a lot of sense; it would essentially just boil down to predicting the most frequent words as well as possible in the bottleneck layer. The latent structure only comes in when we view a sequence of terms.

How to autoencode

Using a direct autoencoder on not make a lot of sense; it would essentially just boil down to predicting the most frequent words as well as possible in the bottleneck layer.

Instead, we want to use the context of a word to predict other similar words that tend to co-occur with it. We'll look at a set of popular approaches to doing this.

Words in context

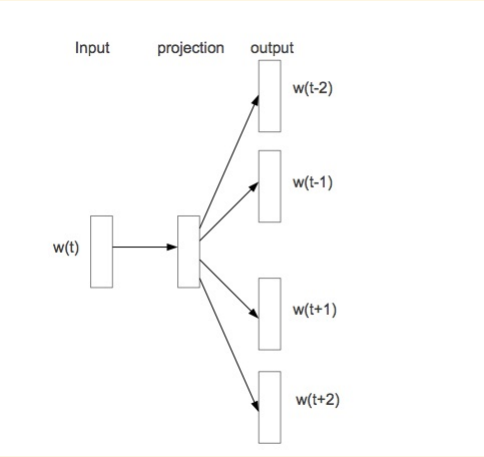
A key idea from the landmark paper

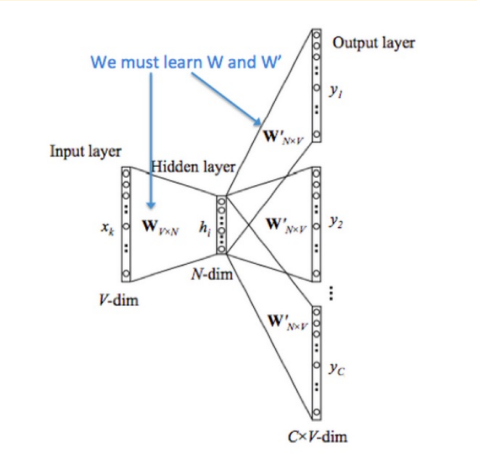
> *Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." In Advances in neural information processing systems, pp. 3111-3119. (2013).*

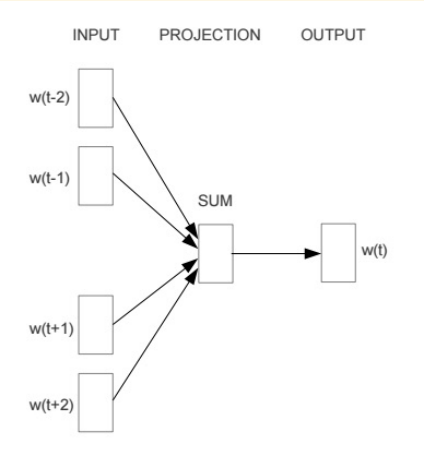
And the closely related follow-up:

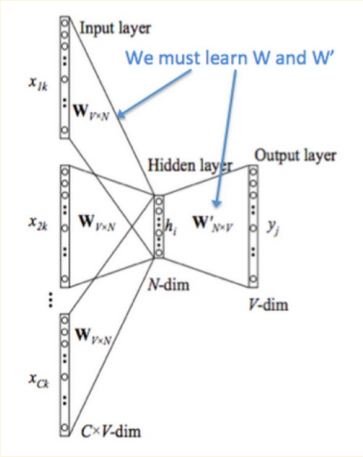
> *Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).*

Is to try to predict a word given its context. Specifically, we want to use the context of a word to predict other similar words that tend to co-occur with it.


Input
projection
output
w(t-2)
w(t-1)
w(t)
w(t+1)
w(t+2)


We must learn W and W'
Output layer
Input layer
Hidden layer
y_1
$\mathbf{W}'_{N\times V}$
x_k
$\mathbf{W}_{V\times N}$
h_i
$\mathbf{W}'_{N\times V}$
y_2
N-dim
V-dim
$\mathbf{W}'_{N\times V}$
y_C
$C\times V$-dim


INPUT
PROJECTION
OUTPUT
w(t-2)
w(t-1)
SUM
w(t)
w(t+1)
w(t+2)


Input layer
We must learn W and W'
x_{1k}
$\mathbf{W}_{V\times N}$
Hidden layer
Output layer
x_{2k}
$\mathbf{W}_{V\times N}$
h_i
$\mathbf{W}'_{N\times V}$
y_j
N-dim
V-dim
$\mathbf{W}_{V\times N}$
x_{Ck}
C×V-dim

You'll notice that these all yield two representations of word embeddings. Typically, they are averaged to get the final set of embeddings to use.

While we have generally steered away from the more theoretical papers, I want to draw your attention to one very interesting one that puts a surprisingly neat theory around word embeddings:

> *Levy, Omer, and Yoav Goldberg. "Neural word embedding as implicit matrix factorization." In Advances in Neural Information Processing Systems, pp. 2177-2185. 2014.*

They show that word embeddings essentially amount to a low-rank factorization of a co-occurrence matrix.

The specific implementation of these ideas by the Mikolov-headed group is known as word2vec. They supply pre-coded version of this. Let's load an example into python:

IV. word2vec

Another, very closely related technique to word2vec is given by GloVe:

> *Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation." In EMNLP, vol. 14, pp. 1532-1543. (2014).*

The differences are subtle at the level we are talking about, but you may see references to this model as well in some of the cited papers.

Finally, there is still some open questions in the world of word embeddings. For example, the ideas started with this paper:

> *Huang, Eric H., Richard Socher, Christopher D. Manning, and Andrew Y. Ng. "Improving word representations via global context and multiple word prototypes." In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1, pp. 873-882. Association for Computational Linguistics, 2012.*

Of what to do with the fact that words often have multiple meanings.

Finally, there is still some open questions in the world of word embeddings. For example, the ideas started with this paper:

> *Huang, Eric H., Richard Socher, Christopher D. Manning, and Andrew Y. Ng. "Improving word representations via global context and multiple word prototypes." In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1, pp. 873-882. Association for Computational Linguistics, 2012.*

Of what to do with the fact that words often have multiple meanings.

Does Anne Hathaway News Drive Berkshire Hathaway's Stock?

676

TEXT SIZE

ALEXIS C. MADRIGAL | MAR 18, 2011 | TECHNOLOGY

Given the awesome correlating powers of today's stock trading computers, the idea may not be as far-fetched as you think.