Deep Learning, NLP, and Representations

Posted on July 7, 2014

neural networks (../../posts/tags/neural_networks.html), deep learning (../../posts/tags/deep_learning.html), representations (../../posts/tags/representations.html), NLP (../../posts/tags/NLP.html), recursive neural networks (../../posts/tags/recursive_neural_networks.html)

Introduction

In the last few years, deep neural networks have dominated pattern recognition. They blew the previous state of the art out of the water for many computer vision tasks. Voice recognition is also moving that way.

But despite the results, we have to wonder... why do they work so well?

This post reviews some extremely remarkable results in applying deep neural networks to natural language processing (NLP). In doing so, I hope to make accessible one promising answer as to why deep neural networks work. I think it's a very elegant perspective.

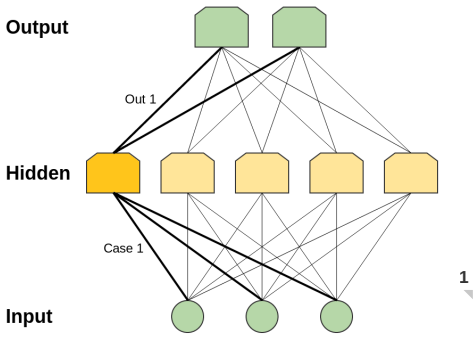
One Hidden Layer Neural Networks

A neural network with a hidden layer has universality: given enough hidden units, it can approximate any function. This is a frequently quoted – and even more frequently, misunderstood and applied – theorem.

It's true, essentially, because the hidden layer can be used as a lookup table.

For simplicity, let's consider a perceptron network. A perceptron (http://en.wikipedia.org/wiki/Perceptron) is a very simple neuron that fires if it exceeds a certain threshold and doesn't fire if it doesn't reach that threshold. A perceptron network gets binary (0 and 1) inputs and gives binary outputs.

Note that there are only a finite number of possible inputs. For each possible input, we can construct a neuron in the hidden layer that fires for that input,[1] and only on that specific input. Then we can use the connections between that neuron and the output neurons to control the output in that specific case. [2]



And so, it's true that one hidden layer neural networks are universal. But there isn't anything particularly impressive or exciting about that. Saying that your model can do the same thing as a lookup table isn't a very strong argument for it. It just means it isn't *impossible* for your model to do the task.

Universality means that a network can fit to any training data you give it. It doesn't mean that it will interpolate to new data points in a reasonable way.

No, universality isn't an explanation for why neural networks work so well. The real reason seems to be something much more subtle... And, to understand it, we'll first need to understand some concrete results.

Word Embeddings

I'd like to start by tracing a particularly interesting strand of deep learning research: word embeddings. In my personal opinion, word embeddings are one of the most exciting area of research in deep learning at the moment, although they were originally introduced by Bengio, *et al.* more than a decade ago.[3] Beyond that, I think they are one of the best places to gain intuition about why deep learning is so effective.

A word embedding $W : \text{words} \to \mathbb{R}^n$ is a paramaterized function mapping words in some language to high-dimensional vectors (perhaps 200 to 500 dimensions). For example, we might find:

$$W(\text{``cat"}) = (0.2,\ \text{-}0.4,\ 0.7,\ \ldots)$$

$$W(\text{``mat"}) = (0.0,\ 0.6,\ \text{-}0.1,\ \ldots)$$

(Typically, the function is a lookup table, parameterized by a matrix, θ, with a row for each word: $W_\theta(w_n) = \theta_n$.)

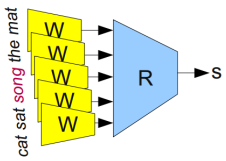
W is initialized to have random vectors for each word. It learns to have meaningful vectors in order to perform some task.

For example, one task we might train a network for is predicting whether a 5-gram (sequence of five words) is 'valid.' We can easily get lots of 5-grams from Wikipedia (eg. "cat sat on the mat") and then 'break' half of them by switching a word with a random word (eg. "cat sat **song** the mat"), since that will almost certainly make our 5-gram nonsensical.

The model we train will run each word in the 5-gram through W to get a vector representing it and feed those into another 'module' called R which tries to predict if the 5-gram is 'valid' or 'broken.' Then, we'd like:

$$R(W(\text{``cat"}),\ W(\text{``sat"}),\ W(\text{``on"}),\ W(\text{``the"}),\ W(\text{``mat"})) = 1$$

$$R(W(\text{``cat"}),\ W(\text{``sat"}),\ W(\text{``song"}),\ W(\text{``the"}),\ W(\text{``mat"})) = 0$$



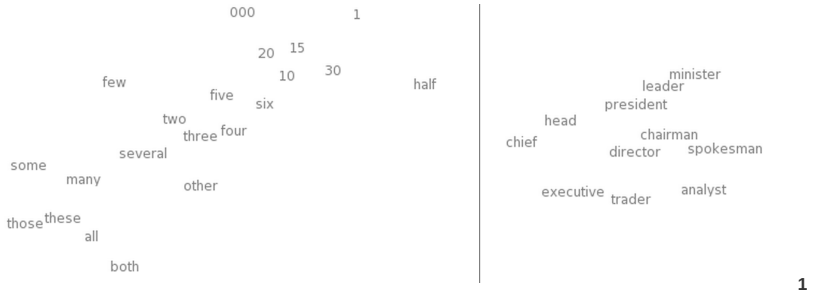
Modular Network to determine if a 5-gram is 'valid' (From Bottou (2011) (http://arxiv.org/pdf/1102.1808v

In order to predict these values accurately, the network needs to learn good parameters for both W and R.

Now, this task isn't terribly interesting. Maybe it could be helpful in detecting grammatical errors in text or something. But what is extremely interesting is W.

(In fact, to us, the entire point of the task is to learn W. We could have done several other tasks – another common one is predicting the next word in the sentence. But we don't really care. In the remainder of this section we will talk about many word embedding results and won't distinguish between different approaches.)

One thing we can do to get a feel for the word embedding space is to visualize them with t-SNE (http://homepage.tudelft.nl/19j49/t-SNE.html), a sophisticated technique for visualizing high-dimensional data.



t-SNE visualizations of word embeddings. Left: Number Region; Right: Jobs Region. From Turian *et al.* (2010) (http://www.iro.umontreal.ca/~lisa/pointeurs/turian-wordrepresentations-acl10.pdf), see complete image (http://metaoptimize.s3.amazonaws.com/cw-embeddings-ACL2010/embeddings-mostcommon.EMBEDDING_SIZE=50.png).

This kind of 'map' of words makes a lot of intuitive sense to us. Similar words are close together. Another way to get at this is to look at which words are closest in the embedding to a given word. Again, the words tend to be quite similar.

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

What words have embeddings closest to a given word? From Collobert *et al.* (2011) (http://arxiv.org/pdf/1103.0398v1.pdf)

It seems natural for a network to make words with similar meanings have similar vectors. If you switch a word for a synonym (eg. "a few people sing well" → "a *couple* people sing well"), the validity of the sentence doesn't change. While, from a naive perspective, the input sentence has changed a lot, if W maps synonyms (like "few" and "couple") close together, from R's perspective little changes.

This is very powerful. The number of possible 5-grams is massive and we have a comparatively small number of data points to try to learn from. Similar words being close together allows us to generalize from one sentence to a class of similar sentences. This doesn't just mean switching a word for a synonym, but also switching a word for a word in a similar class (eg. "the wall is blue" → "the wall is *red*"). Further, we can change multiple words (eg. "the wall is blue" → "the *ceiling* is *red*"). The impact of this is exponential with respect to the number of words.[4]

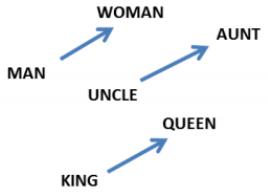
So, clearly this is a very useful thing for W to do. But how does it learn to do this? It seems quite likely that there are lots of situations where it has seen a sentence like "the wall is blue" and know that it is valid before it sees a sentence like "the wall is red". As such, shifting "red" a bit closer to "blue" makes the network perform better.

We still need to see examples of every word being used, but the analogies allow us to generalize to new combinations of words. You've seen all the words that you understand before, but you haven't seen all the sentences that you understand before. So too with neural networks.

Word embeddings exhibit an even more remarkable property: analogies between words seem to be encoded in the difference vectors between words. For example, there seems to be a constant male-female difference vector:

$$W(\text{``woman''}) - W(\text{``man''}) \simeq W(\text{``aunt''}) - W(\text{``uncle''})$$

$$W(\text{“woman”}) - W(\text{“man”}) \simeq W(\text{“queen”}) - W(\text{“king”})$$



From Mikolov *et al.* (2013a) (https://www.aclweb.org/antholo 1090.pdf)

This may not seem too surprising. After all, gender pronouns mean that switching a word can make a sentence grammatically incorrect. You write, “*she* is the aunt” but “*he* is the uncle.” Similarly, “*he* is the King” but “*she* is the Queen.” If one sees “*she* is the *uncle*,” the most likely explanation is a grammatical error. If words are being randomly switched half the time, it seems pretty likely that happened here.

“Of course!” We say with hindsight, “the word embedding will learn to encode gender in a consistent way. In fact, there’s probably a gender dimension. Same thing for singular vs plural. It’s easy to find these trivial relationships!”

It turns out, though, that much more sophisticated relationships are also encoded in this way. It seems almost miraculous!

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

Relationship pairs in a word embedding. From Mikolov *et al.* (2013b) (http://arxiv.org/pdf/1301.3781.pdf).

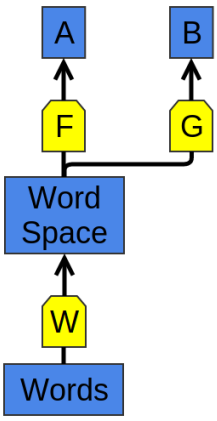
It’s important to appreciate that all of these properties of W are *side effects*. We didn’t try to have similar words be close together. We didn’t try to have analogies encoded with difference vectors. All we tried to do was perform a simple task, like predicting whether a sentence was valid. These properties more or less popped out of the optimization process.

This seems to be a great strength of neural networks: they learn better ways to represent data, automatically. Representing data well, in turn, seems to be essential to success at many machine learning problems. Word embeddings are just a particularly striking example of learning a representation.

Shared Representations

The properties of word embeddings are certainly interesting, but can we do anything useful with them? Besides predicting silly things, like whether a 5-gram is ‘valid’?

We learned the word embedding in order to do well on a simple task, but based on the nice properties we’ve observed in word embeddings, you may suspect that they could be generally useful in NLP tasks. In fact, word representations like these are extremely important:

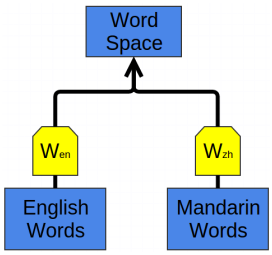
> *The use of word representations... has become a key "secret sauce" for the success of many NLP systems in recent years, across tasks including named entity recognition, part-of-speech tagging, parsing, and semantic role labeling. (Luong et al. (2013)*
>
> 
>
>
> W and F learn to perform task A. Later, G can learn to perform B based on W.
>
> *(http://nlp.stanford.edu/~lmthang/data/papers/conll13_morpho.pdf))*

This general tactic – learning a good representation on a task A and then using it on a task B – is one of the major tricks in the Deep Learning toolbox. It goes by different names depending on the details: pretraining, transfer learning, and multi-task learning. One of the great strengths of this approach is that it allows the representation to learn from more than one kind of data.

There's a counterpart to this trick. Instead of learning a way to represent one kind of data and using it to perform multiple kinds of tasks, we can learn a way to map multiple kinds of data into a single representation!

One nice example of this is a bilingual word-embedding, produced in Socher *et al.* (2013a) (http://ai.stanford.edu/~wzou/emnlp2013_ZouSocherCerManning.pdf). We can learn to embed words from two different languages in a single, shared space. In this case, we learn to embed English and Mandarin Chinese words in the same space.

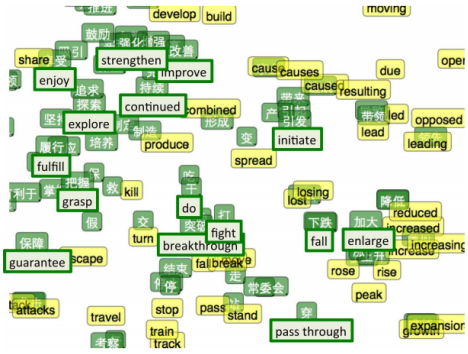
We train two word embeddings, W_{en} and W_{zh} in a manner similar to how we did above. However, we know that certain English words and Chinese words have similar meanings. So, we optimize for an additional property: words that we know are close translations should be close together.



Of course, we observe that the words we knew had similar meanings end up close together. Since we optimized for that, it's not surprising. More interesting is that words we *didn't know* were translations end up close together.

In light of our previous experiences with word embeddings, this may not seem too surprising. Word embeddings pull similar words together, so if an English and Chinese word we know to mean similar things are near each other, their synonyms will also end up near each other. We also know that things like gender differences tend to end up being represented with a constant difference vector. It seems like forcing enough points to line up should force these difference vectors to be the same in both the English and Chinese embeddings. A result of this would be that if we know that two male versions of words translate to each other, we should also get the female words to translate to each other.

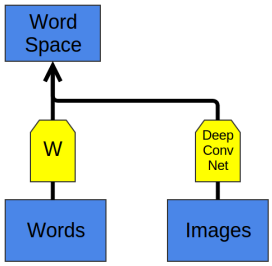
Intuitively, it feels a bit like the two languages have a similar 'shape' and that by forcing them to line up at different points, they overlap and other points get pulled into the right positions.



t-SNE visualization of the bilingual word embedding. Green is Chinese, Yellow is English. (Socher *et al.* (2013a) (http://ai.stanford.edu/~wzou/emnlp2013_ZouSocherCerManning.pdf))

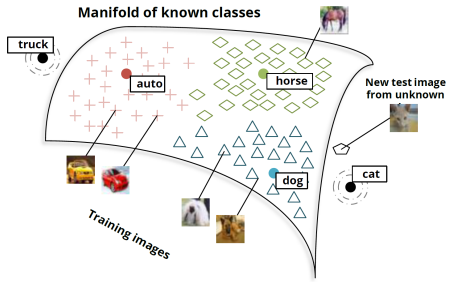
In bilingual word embeddings, we learn a shared representation for two very similar kinds of data. But we can also learn to embed very different kinds of data in the same space.

Recently, deep learning has begun exploring models that embed images and words in a single representation.[5]



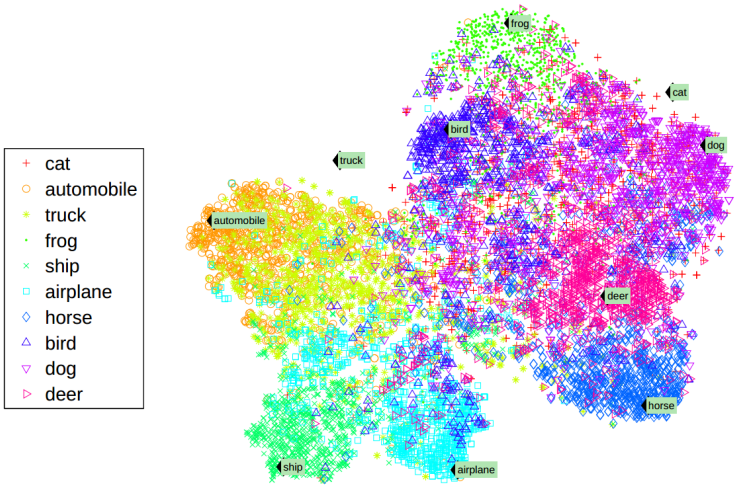
The basic idea is that one classifies images by outputting a vector in a word embedding. Images of dogs are mapped near the "dog" word vector. Images of horses are mapped near the "horse" vector. Images of automobiles near the "automobile" vector. And so on.

The interesting part is what happens when you test the model on new classes of images. For example, if the model wasn't trained to classify cats – that is, to map them near the "cat" vector – what happens when we try to classify images of cats?



(Socher *et al.* (2013b) (http://nlp.stanford.edu/~socherr/SocherGanjooManningNg_NIPS2013.pdf))

It turns out that the network is able to handle these new classes of images quite reasonably. Images of cats aren't mapped to random points in the word embedding space. Instead, they tend to be mapped to the general vicinity of the "dog" vector, and, in fact, close to the "cat" vector. Similarly, the truck images end up relatively close to the "truck" vector, which is near the related "automobile" vector.



(Socher *et al.* (2013b)
(http://nlp.stanford.edu/~socherr/SocherGanjooManningNg_NIPS2013.pdf))

This was done by members of the Stanford group with only 8 known classes (and 2 unknown classes). The results are already quite impressive. But with so few known classes, there are very few points to interpolate the relationship between images and semantic space off of.

The Google group did a much larger version – instead of 8 categories, they used 1,000 – around the same time (Frome *et al.* (2013) (http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/41473.pdf)) and has followed up with a new variation (Norouzi *et al.* (2014) (http://arxiv.org/pdf/1312.5650.pdf)). Both are based on a very powerful image classification model (from Krizehvsky *et al.* (2012) (http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf)), but embed images into the word embedding space in different ways.

The results are impressive. While they may not get images of unknown classes to the precise vector representing that class, they are able to get to the right neighborhood. So, if you ask it to classify images of unknown classes and the classes are fairly different, it can distinguish between the different classes.

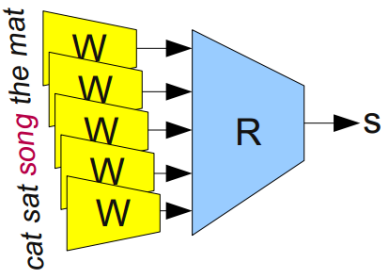
Even though I've never seen a Aesculapian snake or an Armadillo before, if you show me a picture of one and a picture of the other, I can tell you which is which because I have a general idea of what sort of animal is associated with each word. These networks can accomplish the same thing.

(These results all exploit a sort of "these words are similar" reasoning. But it seems like much stronger results should be possible based on relationships between words. In our word embedding space, there is a consistent difference vector between male and female version of words. Similarly, in image space, there are consistent features distinguishing between male and female. Beards, mustaches, and baldness are all strong, highly visible indicators of being male. Breasts and, less reliably, long hair, makeup and jewelery, are obvious indicators of being female.[6] Even if you've never seen a king before, if the queen, determined to be such by the presence of a crown, suddenly has a beard, it's pretty reasonable to give the male version.)

Shared embeddings are an extremely exciting area of research and drive at why the representation focused perspective of deep learning is so compelling.

Recursive Neural Networks

We began our discussion of word embeddings with the following network:

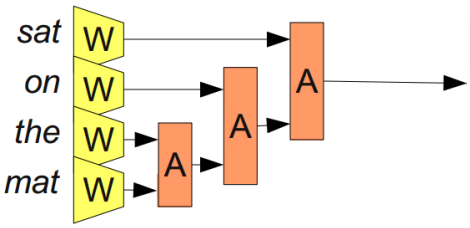


Modular Network that learns word embeddings
(From Bottou (2011)
(http://arxiv.org/pdf/1102.1808v3.pdf))

The above diagram represents a *modular* network, $R(W(w_1), W(w_2), W(w_3), W(w_4), W(w_5))$. It is built from two modules, W and R. This approach, of building neural networks from smaller neural network "modules" that can be composed together, is not very wide spread. It has, however, been very successful in NLP.

Models like the above are powerful, but they have an unfortunate limitation: they can only have a fixed number of inputs.

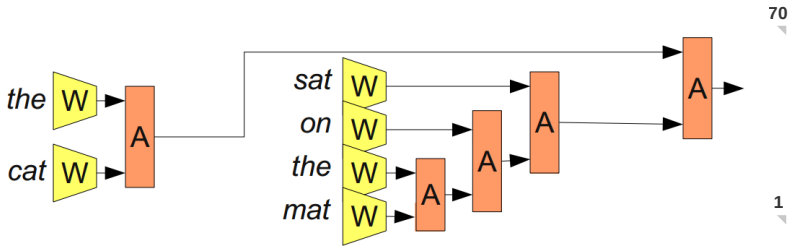
We can overcome this by adding an association module, A, which will take two word or phrase representations and merge them.



(From Bottou (2011)
(http://arxiv.org/pdf/1102.1808v3.pdf))

By merging sequences of words, A takes us from representing words to representing phrases or even representing whole *sentences*! And because we can merge together different numbers of words, we don't have to have a fixed number of inputs.

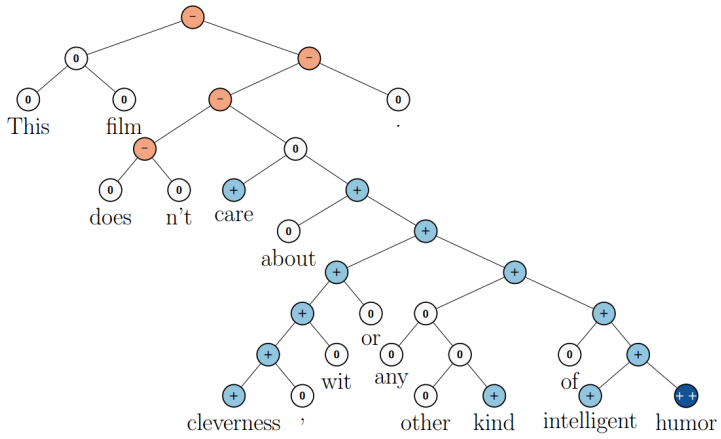
It doesn't necessarily make sense to merge together words in a sentence linearly. If one considers the phrase "the cat sat on the mat", it can naturally be bracketed into segments: "((the cat) (sat (on (the mat))))". We can apply A based on this bracketing:



(From Bottou (2011) (http://arxiv.org/pdf/1102.1808v3.pdf))

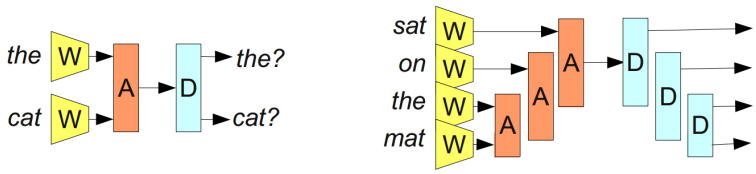
These models are often called "recursive neural networks" because one often has the output of a module go into a module of the same type. They are also sometimes called "tree-structured neural networks."

Recursive neural networks have had significant successes in a number of NLP tasks. For example, Socher *et al.* (2013c) (http://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf) uses a recursive neural network to predict sentence sentiment:



(From Socher *et al.* (2013c)
(http://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf))

One major goal has been to create a *reversible* sentence representation, a representation that one can reconstruct an actual sentence from, with roughly the same meaning. For example, we can try to introduce a disassociation module, D, that tries to undo A:

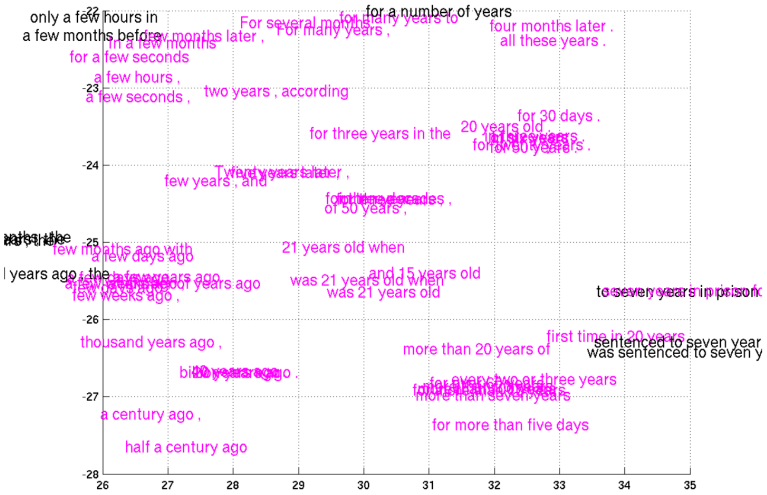


(From Bottou (2011) (http://arxiv.org/pdf/1102.1808v3.pdf))

If we could accomplish such a thing, it would be an extremely powerful tool. For example, we could try to make a bilingual sentence representation and use it for translation.

Unfortunately, this turns out to be very difficult. Very very difficult. And given the tremendous promise, there are lots of people working on it.

Recently, Cho *et al.* (2014) (http://arxiv.org/pdf/1406.1078v1.pdf) have made some progress on representing phrases, with a model that can encode English phrases and decode them in French. Look at the phrase representations it learns!



Small section of the t-SNE of the phrase representation
(From Cho *et al.* (2014) (http://arxiv.org/pdf/1406.1078v1.pdf))

Criticisms

I've heard some of the results reviewed above criticized by researchers in other fields, in particular, in NLP and linguistics. The concerns are not with the results themselves, but the conclusions drawn from them, and how they compare to other techniques.

I don't feel qualified to articulate these concerns. I'd encourage someone who feels this way to describe the concerns in the comments.

Conclusion

The representation perspective of deep learning is a powerful view that seems to answer why deep neural networks are so effective. Beyond that, I think there's something extremely beautiful about it: why are neural networks effective? Because better ways of representing data can pop out of optimizing layered models.

Deep learning is a very young field, where theories aren't strongly established and views quickly change. That said, it is my impression that the representation-focused perspective of neural networks is presently very popular.

This post reviews a lot of research results I find very exciting, but my main motivation is to set the stage for a future post exploring connections between deep learning, type theory and functional programming. If you're interested, you can subscribe to my rss feed (../../rss.xml) so that you'll see it when it is published.

(I would be delighted to hear your comments and thoughts: you can comment inline or at the end. For typos, technical errors, or clarifications you would like to see added, you are encouraged to make a pull request on github (https://github.com/colah/NLP-RNNs-Representations-Post))

Acknowledgments

I'm grateful to Eliana Lorch, Yoshua Bengio, Michael Nielsen, Laura Ball, Rob Gilson, and Jacob Steinhardt for their comments and support.

1. Constructing a case for every possible input requires 2^n hidden neurons, when you have n input neurons. In reality, the situation isn't usually that bad. You can have cases that encompass multiple inputs. And you can have overlapping cases that add together to achieve the right input on their intersection.↩
2. (It isn't only perceptron networks that have universality. Networks of sigmoid neurons (and other activation functions) are also universal: give enough hidden neurons, they can approximate any continuous function arbitrarily well. Seeing this is significantly trickier because you can't just isolate inputs.)↩
3. Word embeddings were originally developed in (Bengio et al, 2001 (http://www.iro.umontreal.ca/~lisa/publications2/index.php/publications/show/64); Bengio et al, 2003 (http://machinelearning.wustl.edu/mlpapers/paper_files/BengioDVJ03.pdf)), a few years before the 2006 deep learning renewal, at a time when neural networks were out of fashion. The idea of distributed representations for symbols is even older, e.g. (Hinton 1986 (http://www.cogsci.ucsd.edu/~ajyu/Teaching/Cogs202_sp13/Readings/hinton86.pdf))."↩
4. The seminal paper, *A Neural Probabilistic Language Model* (Bengio, *et al.* 2003) (http://machinelearning.wustl.edu/mlpapers/paper_files/BengioDVJ03.pdf) has a great deal of insight about why word embeddings are powerful.↩
5. Previous work has been done modeling the joint distributions of tags and images, but it took a very different perspective.↩
6. I'm very conscious that physical indicators of gender can be misleading. I don't mean to imply, for example, that everyone who is bald is male or everyone who has breasts is female. Just that these often indicate such, and greatly adjust our prior.↩

70 Comments (/posts/2014-07-NLP-RNNs-Representations/#disqus_thread)

better, without even knowing the sentence parses nor using recursion. http://arxiv.org/pdf/1405.4...
14 • Reply • Share ›

chrisolah Mod ↗ rrenaud • 3 years ago
Thank you! Technical exposition is a skill I'm working very hard on developing right now.

I hadn't seen the new Mikolov paper, but it looks extremely exciting! :)
1 • Reply • Share ›

Eiso Kant ↗ chrisolah • 2 years ago
Hi Christopher,

First of all thank you for such well written articles, I've been a reader for quite some time but hadn't checked out the comments before. I was curious what your thoughts are on paragraph vectors. Especially since it seems that Mikolov is distancing himself from Quoc's results (https://groups.google.com/d....

All the best,

Eiso (http://sourced.tech)
• Reply • Share ›

Abram Demski • 3 years ago
I appreciated this write-up, but one thing which bothers me is the all-too-common idea that word embeddings are "deep learning" in some sense. Word embeddings can be created by deep or shallow networks, and arguably, shallow is the dominant choice. As I understand it, a main advantage of word2vec over previous neural embeddings was to eliminate the single hidden layer that had been used previously. So, these representations were already fairly shallow, and got better as they were made more shallow.
10 • Reply • Share ›

Alex_Hirner ↗ Abram Demski • 2 years ago
Totally agree, I think much has to be done to determine an ideal depth of neural representations for any given upstream task. Or on the contrary, come up with a typology that eliminates this question (and associated parameter jungle) all together. Maybe good old SOM would suit that?
1 • Reply • Share ›

Abram Demski ↗ Abram Demski • 3 years ago
This mis-label is quite widespread, so I don't mean to accuse you personally! It just bothers me...
• Reply • Share ›

Yuan Wang • 2 years ago
Maybe, I will borrow such an impressive idea in Medical Project :-)
6 • Reply • Share ›

Max Gurewitz • 3 years ago
Great article! I appreciate the clear exposition and formatting.

A note on your interpretation of the universal approximation theorem,

"Universality means that a network can fit to any training data you give it. It doesn't mean that it will interpolate to new data points in a reasonable way."

. The universal approximation states not only that NN's can be fit to one's training data, but that they can approximate that data's underlying generator function!

A good way to think about this is in terms of Cybenko's original 1989 proof, where he was able to demonstrate that MLP's which are sums of sigmoids are equivalent to fourier series which are sums of sinusoidals.
5 • Reply • Share ›

chrisolah Mod ↗ Max Gurewitz • 3 years ago
Thank you for your kind criticism! I haven't read the original Cybenko paper, and am nervous I may have over-stepped my knowledge... (That said, it sounds like this is very similar to the continuous function approximation results for sigmoid networks I mention in footnote 2, so I'm hoping I'm not leading people too

far astray.)

I have an externally imposed deadline requiring me to get all the neural network blog posts I want to write out by Monday, so I'm quite pressed for time right now. I'll try and review the subject more carefully and make corrections soon!

Reply • Share ›

reasontruthandlogic • 3 years ago

The article is concise, enticing and readable and may encourage people who don't already know about machine learning to get involved. In that respect it is useful and probably fulfils its intended purpose.

However, as someone who already knows a bit about ML and was hoping to find out more about the use of "deep learning" in NLP, I was disappointed. For me it has two main problems. One is that it seems to understand by a "deep learning neural network" what is normally called an MLP. The term "deep learning" has recently been used to refer to several quite different models. These include MLPs but more often refer to other kinds of model, such as RBMs, which are quite different. Another problem is that, if you are talking about MLPs, then a lot of well theoretical results have been around for at least 15 years which explain quite simply not only their universality (without recourse to table look up), but also their ability to generalise, to reduce noise, optimally compress, and so on. For example, according to how they are set up, an MLP can do PCA, LDA, Bayes optimal classification (when the local optimum solution it finds happens to be global optimal, which it is often close to), or classification or compression with minimax information loss.

All that really comes across to me in the article is (1) that n-gram word statistics are an effective way of disambiguating different senses of the same word, because of the different grammatical contexts which these different senses permit, and (2) that representing each word by a numerical vector can be an effective way of permitting categorial word data to be successfully manipulated by what is an inherently distributed mechanism. These are important points, but as far as artificial neural networks go, understanding is concerned I feel that this

see more

3 Reply • Share ›

Roydell Clarke ➚ reasontruthandlogic • a year ago

Maybe this blog is just not for you. Olah is certainly preparing us to read and understand your more advance NN blog. What specifically should he be writing about?

2 Reply • Share ›

Nenad Živić • 2 years ago

This is a well written article, thanks! I would have an implementation question, though: what is the preferred way of implementing the W module? Is it a matrix that you update after each backpropagation iteration, or it's adding a 1-hot input layer and then have embeddings coded in the weight matrix of the first non-input layer?

Thanks in advance!

1 Reply • Share ›

chrisolah Mod ➚ Nenad Živić • 2 years ago

The two are equivalent. :)

In practice, you want to implement it as a matrix and look up the vector in the nth row -- multiplying super wide matrices is expensive!

There's lots of example code online. Here's some Theano (look at the self.emb variable): http://deeplearning.net/tut...

Reply • Share ›

Nenad Živić ➚ chrisolah • 2 years ago

Thank you very much, I'll give it a more detailed look tomorrow, but seems to answer my question. :)

Can you share some more of those lots of example code? :) I haven't been able to come up with much. Lasagne has an embedding layer implementation, but it seems to still struggle with it.

Reply • Share ›

Sabyasachi Dey • 4 days ago

I am an IT Consultant and this article is very helpful to understand NLP applications of neural networks...thank you so much!!!

Reply • Share ›

luvprkurs • 4 days ago
I am an IT Consultant and this article is very helpful to understand NLP applications of neural networks...thank you so much!!!
• Reply • Share ›

Greg • 7 days ago
this bears very procatively on issues of species and genus (Aristotelian, Abelardian). Thank you
• Reply • Share ›

AlexanderL • 2 months ago
Is the miraculous relationships between words (e.g. France-Paris; Italy-Rome) only found in Mikolov's word2vec model, or is it a common phenomena for all word embedding models?
• Reply • Share ›

郑瀚 • 5 months ago
i would say it's a great post, learning a lot
• Reply • Share ›

Hieu Nguyen • 8 months ago
please help! Can I make small dataset for myshelf? Because I want to enter a key and it return a answer near meaning in file dataset what I did?
• Reply • Share ›

satish jasti • 8 months ago
Thanks a lot Christopher
• Reply • Share ›

Amina Imam Abubakar • 8 months ago
A great article especially for the starters like me. My question is does word emedding on sentiment analysis work best with annotated text or unannotated text?
• Reply • Share ›

Phil Ritchie • a year ago
A great article Christopher, thanks. I see I am coming to the party late. One remaining gap/doubt I have in my understanding is around the features used for individual word vectors. Is it correct that they can be composed of co-occurrence frequencies? I've seen lots of literature about co-occurrence frequencies using a windowing algorithm but not certain how these combine to give final vector values.
• Reply • Share ›

chrisolah Mod → Phil Ritchie • a year ago
You can create word vectors with certain factorizations of co-occurence statistics. This is very similar to how you can construct word vectors by having a neural net predict which words will be near by.

That said, not all word vectors have the same properties. For example, if you train a word embedding to predict which words will occur near another word, opposite words will often have similar embeddings (because they occur in similar contexts). On the other hand, if you train word embeddings for translation, they will be very different (because they translate to very different sets of words in the target language). See https://arxiv.org/pdf/1410....

(Note that I don't follow this literature very carefully and could be pretty out of date in my understanding.)
1 • Reply • Share ›

Kevin Liu • 2 years ago
Does it feel natural to force difference vectors to be the same?
• Reply • Share ›

Lucas McLane • 2 years ago
Excellent article! While ramping up on ML/DL concepts I found this article extremely enlightening.
• Reply • Share ›

Katy Lee • 2 years ago
Thanks for the good summary the explanation. I love it.
• Reply • Share ›

Divya Sivasankaran • 2 years ago
This is an amazingly well written article! Keep them coming :)
• Reply • Share ›

mayday1991 • 2 years ago
Great!!
• Reply • Share ›

Roydell Clarke • 2 years ago
Are there any sample code?
I learn faster when I see some starter code to drive home the concepts.
• Reply • Share ›

chrisolah Mod ➚ Roydell Clarke • 2 years ago
There is sample code for this in a number of languages and frameworks. Here's some for TensorFlow:

https://www.tensorflow.org/...
• Reply • Share ›

Roydell Clarke ➚ chrisolah • 2 years ago
Thanks you so much for the link.

Roydell Clarke

Chief Creative Strategist
Phone: 613-899-4145
• Reply • Share ›

AR • 2 years ago
Hi, great article. I'm just curious -- what happens to words that have multiple meanings? Where do their vectors end up relative to each and all of the common meanings?
• Reply • Share ›

chrisolah Mod ➚ AR • 2 years ago
The vector should learn some compromise between the different meanings.

Ideally you'd like to disambiguate the different word senses and learn a different vector for each one. Some people work on this!
• Reply • Share ›

Sanjaya Wijeratne • 2 years ago
This is eye candy for anyone new to/interested in deep learning. Thank you very much for this and the other great articles you have written on this topic.
• Reply • Share ›

viral • 2 years ago
I am trying to implement CBOW model to come up with word representations. The idea is to further use the vectors for some other task.
my CBOW model written in python runs very slow. Its taking > 45 mins for single iteration over 3500 sentence corpus (3500 isnt that huge). I am using numpy and doing all calcs as matrix, vectoe multiplications. Any suggestions for me.
• Reply • Share ›

chrisolah Mod ➚ viral • 2 years ago
You should look into a modern neural network library like TensorFlow. It's really hard to make things efficient with NumPy because you can't optimize the computation as a whole and everything goes through python

python.

Reply • Share ›

Fogetti • 2 years ago

Great article! I like the clear explanation. I am new to machine learning and deep learning also. This will be my definitive introduction to the field. :)

Reply • Share ›

Narayanan Ellango • 2 years ago

was also here... nice.

Reply • Share ›

Rahul Sharma • 2 years ago

I am really delighted that this post has made me understand the concept of word embedding and how is it being used and making me to read the references with more interest. This was a really interesting work. I appreciate your dedication. I have chosen NLP as my graduation project and this article has saved me.

Reply • Share ›

Sam DeHority • 2 years ago

Linguistics student here. The problem here is that sentences carry meaning back and forth across the sentence in a far more complicated manner than just between adjacent words. For example, in the sentence "Which book did you read?" where there is a close association between "which book" and "read". This sentence would be expected to pass the correctness check for a 5-vector of words. On the other hand, "Which book did you sleep?" Seems to also pass with the same association module.

Reply • Share ›

Fred Mailhot ➚ Sam DeHority • 2 years ago

The general point you've made is definitely right, given the (in principle) unboundedness of some kinds of syntactic dependencies. Nonetheless, given a suitably large & diverse corpus, I don't think that the particular (type of) situation you describe above would obtain.

A 5-gram detection/"verification" network trained on a boatload of diverse data would learn (among other things) that (a) "read" is typically followed by nouns (or NPs), and perhaps even more specifically Ns close to "book", and (b) that "sleep" is typically not followed by those kinds of things. I.e. it would learn about argument selection, and possibly selectional restrictions. Finally, it would see lots of things like "Which [nounish thing] did you [verb-that-takes-a-complement-ish thing]" and few or no instances of "Which [noun-ish things] did you [verb-that-doesn't-take-a-complement-ish thing]", learning that the latter are likely bad examples of "grammatical" 5-grams.

Reply • Share ›

chrisolah Mod ➚ Fred Mailhot • 2 years ago

Good point, Fred!

It's also the case that people train word embeddings on more sophisticated tasks, like language modelling or translation. I only chose the n-gram example because it was simple!

Reply • Share ›

Vered Shwartz • 2 years ago

Great article. I read a lot about this topic but I think nothing helped my understanding as your article. Thanks!

Reply • Share ›

Udit Saini • 3 years ago

really great article. can you also point out some source where we can implement it practically.
Thanks

Reply • Share ›

Djellel Difallah • 3 years ago

Hi! I am looking for the code that does the example you are using, i.e., train on valid 5-grams and predict the validity of new ones. Do you know where can I find such implementation? (or any pointer that might be useful)
Thanks in advance!

Reply • Share ›

Djellel Difallah ➚ Djellel Difallah • 3 years ago

I did some homework, now I am using word2vec to get vectors representing the words and Random forest for the prediction task. My question is more specific now:
How would you specifically combine the vectors of an n-gram? computing the average isn't helpful and appending the vectors might dilute the meaning.. what do you think?
Thanks!

 • Reply • Share ›