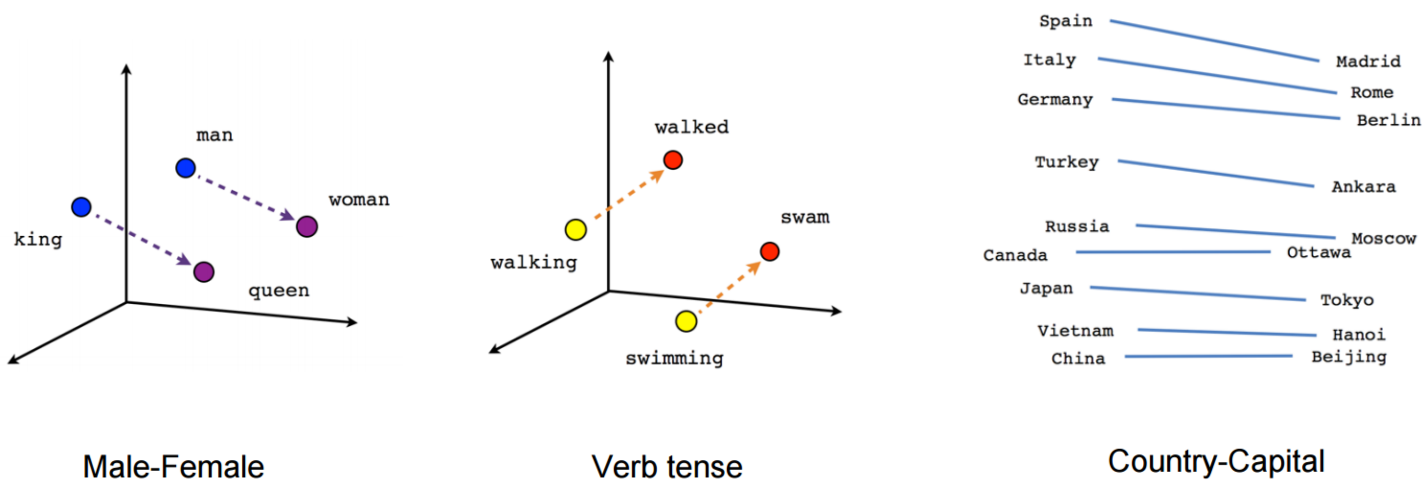
Deep Learning #4: Why You Need to Start Using Embedding Layers

And how there's more to it than word embeddings.

Rutger Ruizendaal
Jul 17, 2017 · 7 min read



This post is part of a series on deep learning. Checkout the other parts here:

1. Setting up AWS & Image Recognition
2. Convolutional Neural Networks
3. More on CNNs & Handling Overfitting

. . .

Welcome to part 4 of this series on deep learning. As you might have noticed there has been a slight delay between the first three entries and this post. The initial goal of this series was to write along with the fast.ai course on deep learning. However, the concepts of the later lectures are often overlapping so I decided to finish the course first. This way I get to provide a more detailed overview of these topics. In this blog I want to cover a concept that spans multiple lectures of the course (4–6) and that has proven very useful to me in practice: Embedding Layers.

Upon introduction the concept of the embedding layer can be quite foreign. For example, the Keras documentation provides no explanation other than "Turns positive integers (indexes) into dense vectors of fixed size". A quick Google search might not get you much further either since these type of documentations are the first things to pop-up. However, in a sense Keras' documentation describes all that happens. So why should you use an embedding layer? Here are the two main reasons:

1. One-hot encoded vectors are high-dimensional and sparse. Let's assume that we are doing Natural Language Processing (NLP) and have a dictionary of 2000 words. This means that, when using one-hot encoding, each word will be represented by a vector containing 2000 integers. And 1999 of these integers are zeros. In a big dataset this approach is not computationally efficient.

2. The vectors of each embedding get updated while training the neural network. If you have seen the image at the top of this post you can see how similarities between words can be found in a multi-dimensional space. This allows us to visualize relationships between words, but also between everything that can be turned into a vector through an embedding layer.

This concept might still be a bit vague. Let's have a look at what an embedding layer does with an example of words. Nevertheless, the origin of embeddings comes from word embeddings. You can look up word2vec if you are interested in reading more. Let's take this sentence as an example (do not take it to seriously):

> *"deep learning is very deep"*

The first step in using an embedding layer is to encode this sentence by indices. In this case we assign an index to each unique word. The sentence than looks like this:

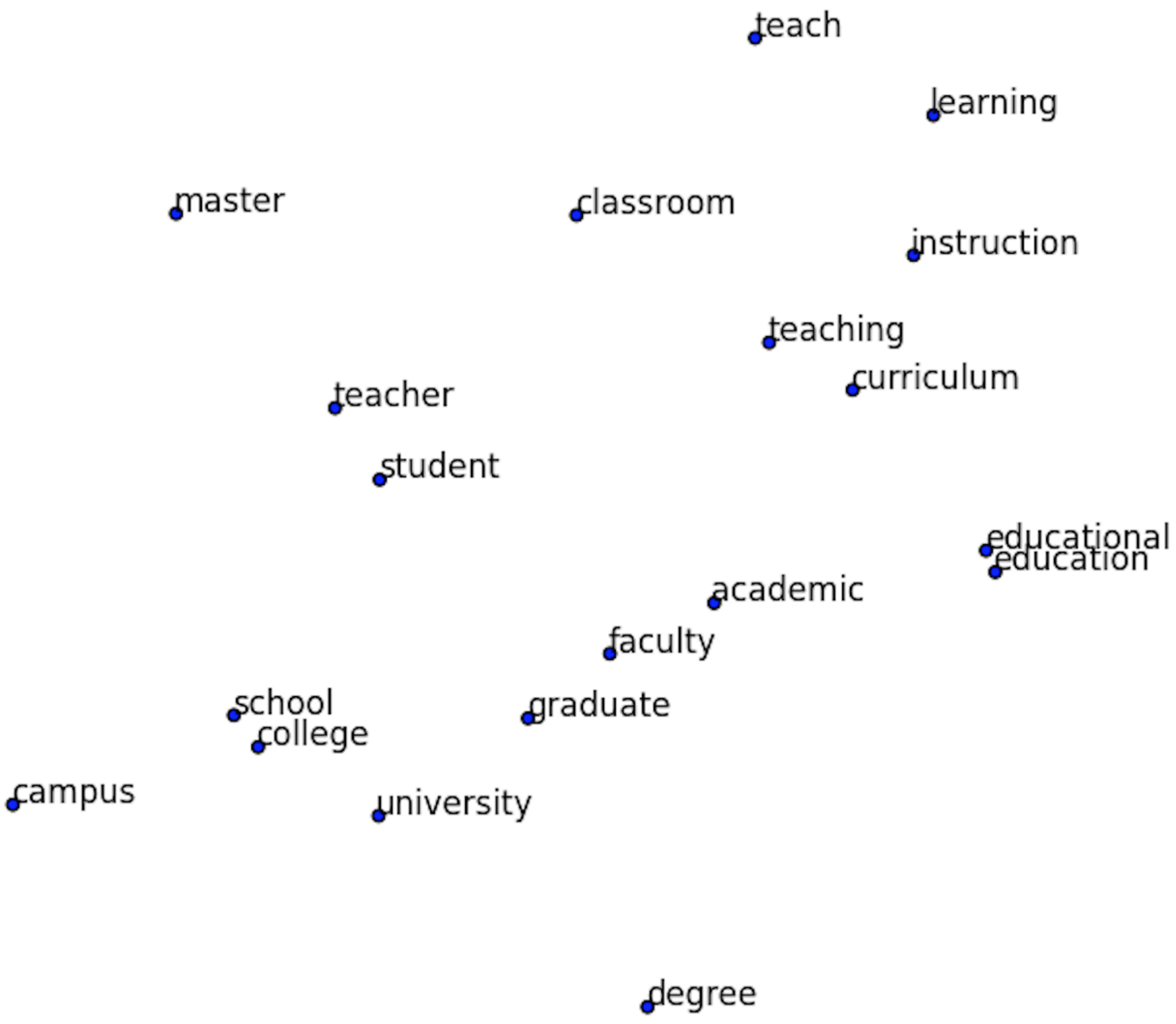
> *1 2 3 4 1*

The embedding matrix gets created next. We decide how many 'latent factors' are assigned to each index. Basically this means how long we want the vector to be. General use cases are lengths like 32 and 50. Let's assign 6 latent factors per index in this post to keep it readable. The embedding matrix than looks like this:

Indices	Latent Factors					
1	.32	.02	.48	.21	.56	.15
2	.65	.23	.41	.57	.03	.92
3	.45	.87	.89	.45	.12	.01
4	.65	.21	.25	.45	.78	.82

Embedding Matrix

So, instead of ending up with huge one-hot encoded vectors we can use an embedding matrix to keep the size of each vector much smaller. In short, all that happens is that the word “deep” gets represented by a vector [.32, .02, .48, .21, .56, .15]. However, not every word gets replaced by a vector. Instead, it gets replaced by index that is used to look-up the vector in the embedding matrix. Once again, this is computationally efficient when using very big datasets. Because the embedded vectors also get updated during the training process of the deep neural network, we can explore what words are similar to each other in a multi-dimensional space. By using dimensionality reduction techniques like t-SNE these similarities can be visualized.



t-SNE visualization of word embeddings

. . .

Not Just Word Embeddings

These previous examples showed that word embeddings are very important in the world of Natural Language Processing. They allow us to capture relationships in language that are very difficult to capture otherwise. However, embedding layers can be used to embed many more things than just words. In my current research project I am using embedding layers to embed online user behavior. In this case I am assigning indices to user behavior like 'page view on page type X on portal Y' or 'scrolled X pixels'. These indices are then used for constructing a sequence of user behavior.

In a comparison of 'traditional' machine learning models (SVM, Random Forest, Gradient Boosted Trees) with deep learning models (deep neural networks, recurrent neural networks) I found that this embedding approach worked very well for deep neural networks.

The 'traditional' machine learning models rely on a tabular input that is feature engineered. This means that we, as researchers, decide what gets turned into a feature. In these cases features could be: amount of homepages visited, amount of searches done, total amount of pixels scrolled. However, it is very difficult to capture the spatial (time) dimension when doing feature-engineering. By using deep learning and embedding layers we can efficiently capture this spatial dimension by supplying a sequence of user behavior (as indices) as input for the model.

In my research the Recurrent Neural Network with Gated Recurrent Unit/Long-Short Term Memory performed best. The results were very close. From the 'traditional' feature engineered models Gradient Boosted Trees performed best. I will write a blog post about this research in more detail in the future. I think my next blog post will explore Recurrent Neural Networks in more detail.

Other research has explored the use of embedding layers to encode student behavior in MOOCs (Piech et al., 2016) and users' path through an online fashion store (Tamhane et al., 2017).

. . .

Recommender Systems

Embedding layers can even be used to deal with the sparse matrix problem in recommender systems. Since the deep learning course (fast.ai) uses recommender systems to introduce embedding layers I want to explore them here as well.

Recommender systems are being used everywhere and you are probably being influenced by them every day. The most common examples are Amazon's product recommendation and Netflix's program recommendation systems. Netflix actually held a $1,000,000 challenge to find the best collaborative filtering algorithm for their recommender system. You can see a visualization of one of these models here.

There are two main types of recommender systems and it is important to distinguish between the two.

1. Content-based filtering. This type of filtering is based on data about the item/product. For example, we have our users fill out a survey on what movies they like. If they say that they like sci-fi movies we recommend them sci-fi movies. In this case al lot of meta-information has to be available for all items.

2. Collaborative filtering: Let's find other people like you, see what they liked and assume you like the same things. People like you = people who rated movies that you watched in a similar way. In a large dataset this has proven to work a lot better than the meta-data approach. Essentially asking people about their behavior is less good compared to looking at their actual behavior. Discussing this further is something for the psychologists among us.

In order to solve this problem we can create a huge matrix of the ratings of all users against all movies. However, in many cases this will create an extremely sparse matrix. Just think of your Netflix account. What percentage of their total supply of series and movies have you watched? It's probably a pretty small percentage. Then, through gradient descent we can train a neural network to predict how high each user would rate each movie. Let me know if you would like to know more about the use of deep learning in recommender systems and we can explore it further together. In conclusion, embedding layers are amazing and should not be overlooked.

If you liked this posts be sure to recommend it so others can see it. You can also follow this profile to keep up with my process in the Fast AI course. See you there!

References