(https://google.com/racialequity?authuser=0)

Embeddings: Obtaining Embeddings

ıted Time: 10 minutes

There are a number of ways to get an embedding, including a state-of-the-art algorithm created at Google.

Standard Dimensionality Reduction Techniques

There are many existing mathematical techniques for capturing the important structure of a high-dimensional space in a low dimensional space. In theory, any of these techniques could be used to create an embedding for a machine learning system.

For example, principal component analysis (https://wikipedia.org/wiki/Principal_component_analysis) (PCA) has been used to create word embeddings. Given a set of instances like bag of words vectors, PCA tries to find highly correlated dimensions that can be collapsed into a single dimension.

Word2vec

Word2vec is an algorithm invented at Google for training word embeddings. Word2vec relies on the **distributional hypothesis** to map semantically similar words to geometrically close embedding vectors.

The distributional hypothesis states that words which often have the same neighboring words tend to be semantically similar. Both "dog" and "cat" frequently appear close to the word "vet", and this fact reflects their semantic similarity. As the linguist John Firth put it in 1957, "You shall know a word by the company it keeps".

Word2Vec exploits contextual information like this by training a neural net to distinguish actually co-occurring groups of words from randomly grouped words. The input layer takes a sparse representation of a target word together with one or more context words. This input connects to a single, smaller hidden layer.

In one version of the algorithm, the system makes a negative example by substituting a random noise word for the target word. Given the positive example "the plane flies", the system might swap in "jogging" to create the contrasting negative example "the jogging flies".

The other version of the algorithm creates negative examples by pairing the true target word with randomly chosen context words. So it might take the positive examples (the, plane), (flies, plane) and the negative examples (compiled, plane), (who, plane) and learn to identify which pairs actually appeared together in text.

The classifier is not the real goal for either version of the system, however. After the model has been trained, you have an embedding. You can use the weights connecting the input layer with the hidden layer to map sparse representations of words to smaller vectors. This embedding can be reused in other classifiers.

For more information about word2vec, see the tutorial on tensorflow.org (https://www.tensorflow.org/tutorials/word2vec/index.html?authuser=0)

Training an Embedding as Part of a Larger Model

You can also learn an embedding as part of the neural network for your target task. This approach gets you an embedding well customized for your particular system, but may take longer than training the embedding separately.

In general, when you have sparse data (or dense data that you'd like to embed), you can create an embedding unit that is just a special type of hidden unit of size *d*. This embedding layer can be combined with any other features and hidden layers. As in any DNN, the final layer will be the loss that is being optimized. For example, let's say we're performing collaborative filtering, where the goal is to predict a user's interests from the interests of other users. We can model this as a supervised learning problem by randomly setting aside (or holding out) a small number of the movies that the user has watched as the positive labels, and then optimize a softmax loss.

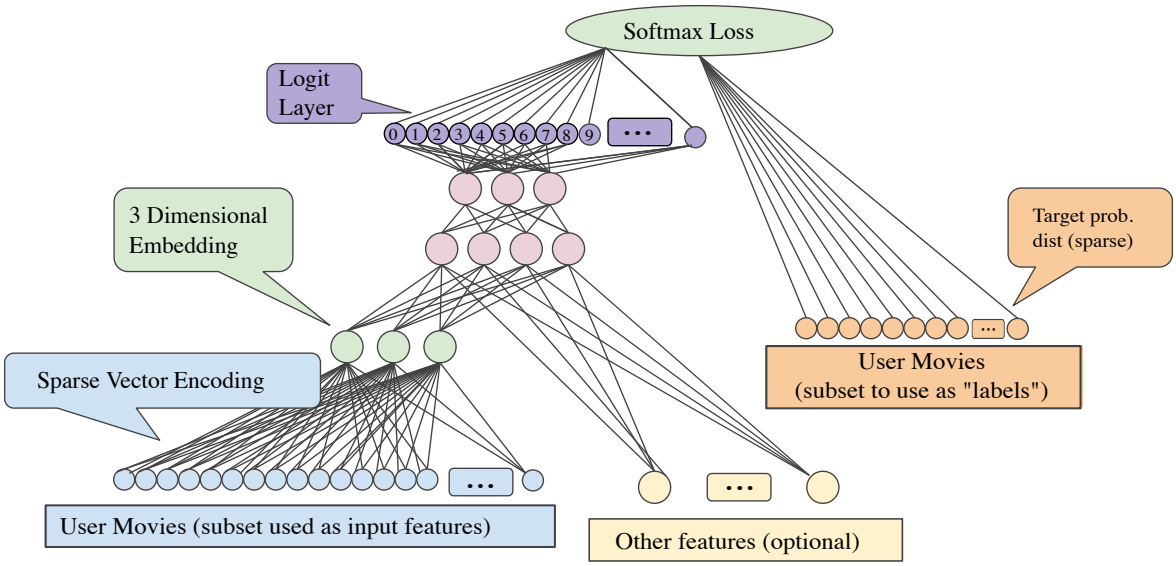

Figure 5. A sample DNN architecture for learning movie embeddings from collaborative filtering data.

As another example if you want to create an embedding layer for the words in a real-estate ad as part of a DNN to predict housing prices then you'd optimize an L_2 Loss using the known sale price of homes in your training data as the label.

When learning a *d*-dimensional embedding each item is mapped to a point in a *d*-dimensional space so that the similar items are nearby in this space. Figure 6 helps to illustrate the relationship between the weights learned in the embedding layer and the geometric view. The edge weights between an input node and the nodes in the *d*-dimensional embedding layer correspond to the coordinate values for each of the *d* axes.

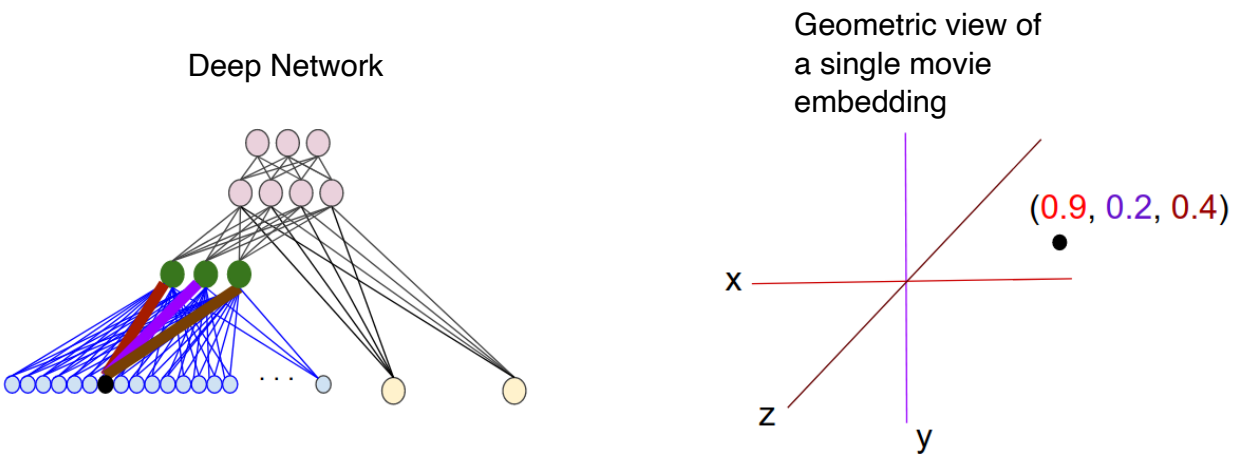

Figure 6. A geometric view of the embedding layer weights.