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

Embeddings: Categorical Input Data

ıted Time: 10 minutes

Categorical data refers to input features that represent one or more discrete items from a finite set of choices. For example, it can be the set of movies a user has watched, the set of words in a document, or the occupation of a person.

Categorical data is most efficiently represented via **sparse tensors**, which are tensors with very few non-zero elements. For example, if we're building a movie recommendation model, we can assign a unique ID to each possible movie, and then represent each user by a sparse tensor of the movies they have watched, as shown in Figure 3.

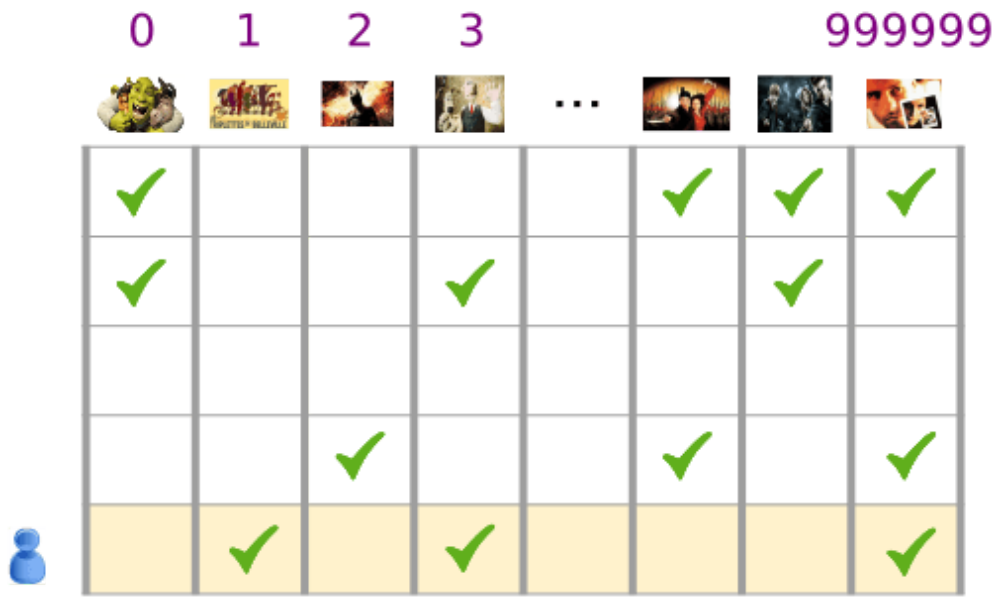

Figure 3. Data for our movie recommendation problem.

Each row of the matrix in Figure 3 is an example capturing a user's movie-viewing history, and is represented as a sparse tensor because each user only watches a small fraction of all possible movies. The last row corresponds to the sparse tensor [1, 3, 999999], using the vocabulary indices shown above the movie icons.

Likewise one can represent words, sentences, and documents as sparse vectors where each word in the vocabulary plays a role similar to the movies in our recommendation

example.

In order to use such representations (https://developers.google.com/machine-learning/crash-course/representation/video-lecture?authuser=0) within a machine learning system, we need a way to represent each sparse vector as a vector of numbers so that semantically similar items (movies or words) have similar distances in the vector space. But how do you represent a word as a vector of numbers?

The simplest way is to define a giant input layer with a node for every word in your vocabulary, or at least a node for every word that appears in your data. If 500,000 unique words appear in your data, you could represent a word with a length 500,000 vector and assign each word to a slot in the vector.

If you assign "horse" to index 1247, then to feed "horse" into your network you might copy a 1 into the 1247th input node and 0s into all the rest. This sort of representation is called a **one-hot encoding**, because only one index has a non-zero value.

More typically your vector might contain counts of the words in a larger chunk of text. This is known as a "bag of words" representation. In a bag-of-words vector, several of the 500,000 nodes would have non-zero value.

But however you determine the non-zero values, one-node-per-word gives you very *sparse* input vectors—very large vectors with relatively few non-zero values. Sparse representations have a couple of problems that can make it hard for a model to learn effectively.

Size of Network

Huge input vectors mean a super-huge number of weights for a neural network. If there are M words in your vocabulary and N nodes in the first layer of the network above the input, you have MxN weights to train for that layer. A large number of weights causes further problems:

- **Amount of data**. The more weights in your model, the more data you need to train effectively.
- **Amount of computation**. The more weights, the more computation required to train and use the model. It's easy to exceed the capabilities of your hardware.

Lack of Meaningful Relations Between Vectors

If you feed the pixel values of RGB channels into an image classifier, it makes sense to talk about "close" values. Reddish blue is close to pure blue, both semantically and in terms of the geometric distance between vectors. But a vector with a 1 at index 1247 for "horse" is not any closer to a vector with a 1 at index 50,430 for "antelope" than it is to a vector with a 1 at index 238 for "television".

The Solution: Embeddings

The solution to these problems is to use **embeddings**, which translate large sparse vectors into a lower-dimensional space that preserves semantic relationships. We'll explore embeddings intuitively, conceptually, and programmatically in the following sections of this module.

erms

ɔut layer
://developers.google.com/machine-
ıg/glossary?authuser=0#input_layer)

- one-hot encoding
(https://developers.google.com/machine-learning/gloss
authuser=0#one-hot_encoding)

arse features (https://developers.google.com/machine-learning/glossary?authuser=0#sparse_features)

Help Center (https://support.google.com/machinelearningeducation?authuser=0)

Previous
← Motivation from Collaborative Filtering
(https://developers.google.com/machine-learning/crash-course/embeddings/motivation-from-collaborative-filtering?authuser=0)

Next
Translating to a Lower-Dimensional Space →
(https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space?authuser=0)