Google Cloud

# Embeddings

Lak Lakshmanan

Machine Learning on
Google Cloud Platform

The Art of ML

Hyperparameter Tuning

A Pinch of Science

The Science of Neural Networks

**Embeddings**

Custom Estimator

# Learn how to...

Use embeddings to:

- Manage sparse data

# Learn how to...

Use embeddings to:

- Manage sparse data
- Reduce dimensionality

# Learn how to...

Use embeddings to:

- Manage sparse data
- Reduce dimensionality
- Increase model generalization

# Learn how to...

Use embeddings to:

- Manage sparse data
- Reduce dimensionality
- Increase model generalization
- Cluster observations

# Learn how to...

Use embeddings to:

- Manage sparse data
- Reduce dimensionality
- Increase model generalization
- Cluster observations

Create reusable embeddings

# Learn how to...

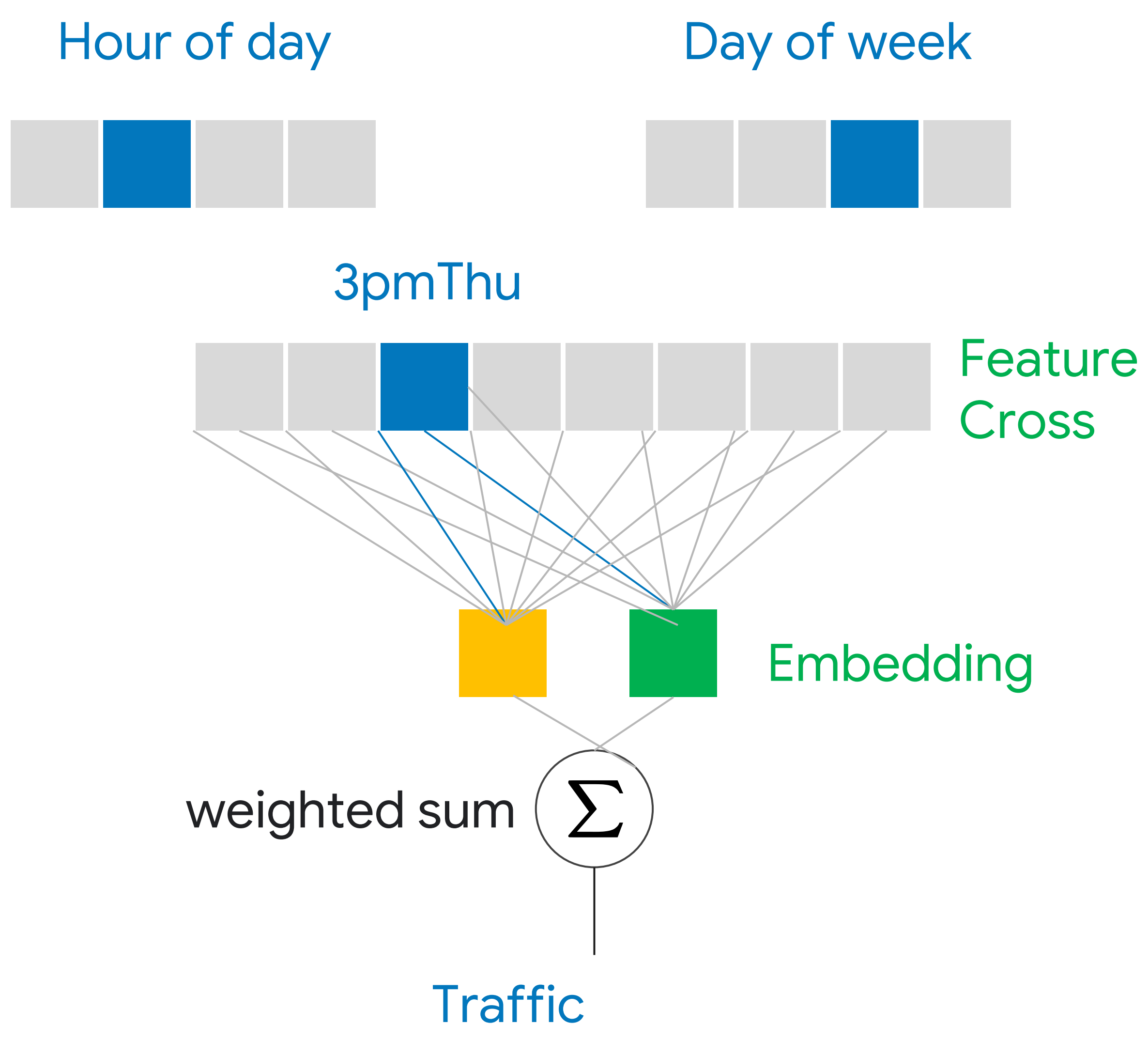
Use embeddings to:

- Manage sparse data
- Reduce dimensionality
- Increase model generalization
- Cluster observations

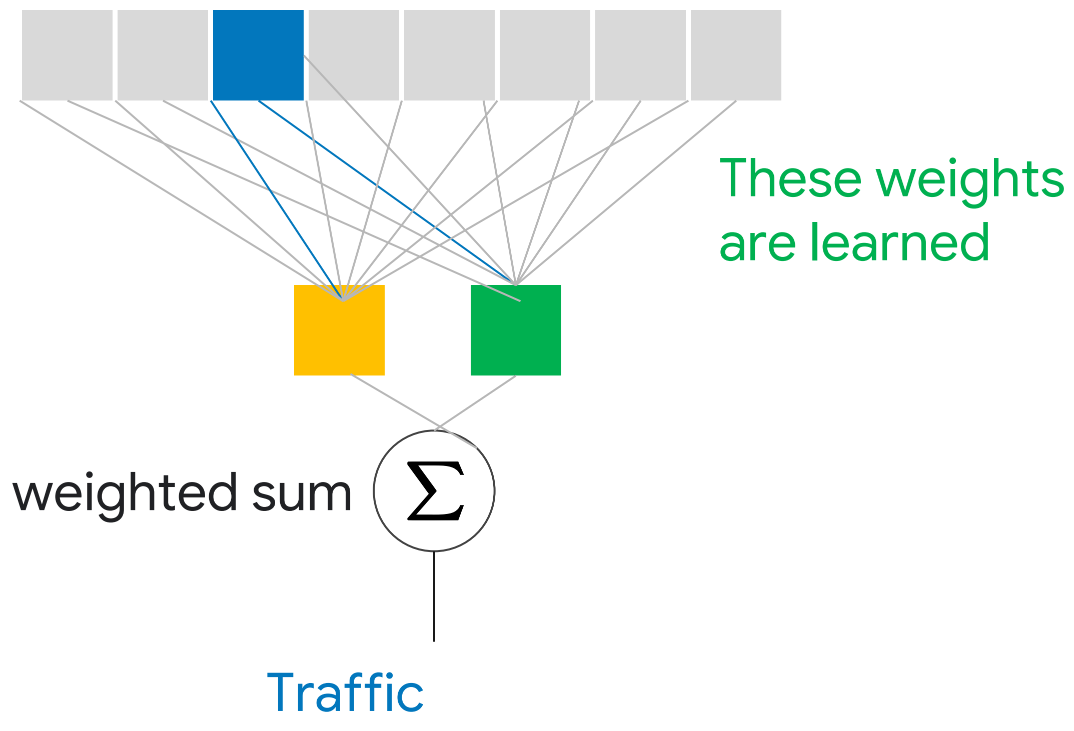
Create reusable embeddings

Explore embeddings in TensorBoard

# Creating an embedding column from a feature cross



# The weights in the embedding column are learned from data



# The model learns how to embed the feature cross in lower-dimensional space

| | | |
|---|---|---|
| 8am Tue | 0.8 | 0.7 |
| 9am Wed | 0.7 | 0.9 |
| 11 am Tue | 0.1 | 0.6 |
| 2 pm Wed | 0.1 | 0.7 |
| 2 am Tue | 0 | 0.1 |
| 2 am Wed | 0 | 0.1 |

similar

different

# Embedding a feature cross in TensorFlow

```
import tf.feature_column as fc

day_hr = fc.crossed_column(
   [dayofweek, hourofday],
   24x7 )

day_hr_em = fc.embedding_column(
   day_hr,
   2 )
```

# Transfer Learning of embeddings from similar ML models

```
import tf.feature_column as fc

day_hr = fc.crossed_column(
   [dayofweek, hourofday],
   24x7 )

day_hr_em = fc.embedding_column(
   day_hr,
   2,
   ckpt_to_load_from='london/*ckpt-1000*',
   tensor_name_in_ckpt='dayhr_embed',
   trainable=False
 )
```

# Transfer Learning of embeddings from similar ML models

First layer:  the feature cross

Second layer:  a mystery box labeled latent factor

Third layer: the embedding

Fourth layer: one side: image of traffic

Second side: image of people watching TV

# Representing feature columns as sparse vectors

These are all different ways to create a categorical column

If you know the keys beforehand:

```
tf.feature_column.categorical_column_with_vocabulary_list('employeeId',
    vocabulary_list = ['8345', '72345', '87654', '98723', '23451']),
```

If your data is already indexed; i.e., has integers in [0-N):

```
tf.feature_column.categorical_column_with_identity('employeeId',
    num_buckets = 5)
```

If you don't have a vocabulary of all possible values:

```
tf.feature_column.categorical_column_with_hash_bucket('employeeId',
    hash_bucket_size = 500)
```

# How do you recommend movies to customers?

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ☺ | 2 | | | | | | | 4 |
| ☺ | | | | 5 | | | 2 | |
| ☺ | | | | | | 3 | | |
| ☺ | | | 4 | | | | | |
| ☺ | | | | | 4 | | 5 | |
| ☺ | | 5 | | | | | | |

1 million customers


500,000 movies

# One approach is to organize movies by similarity (1D)

Average age of viewers

Shrek

Incredibles


The Triplets of Belleville

Harry Potter


Star Wars

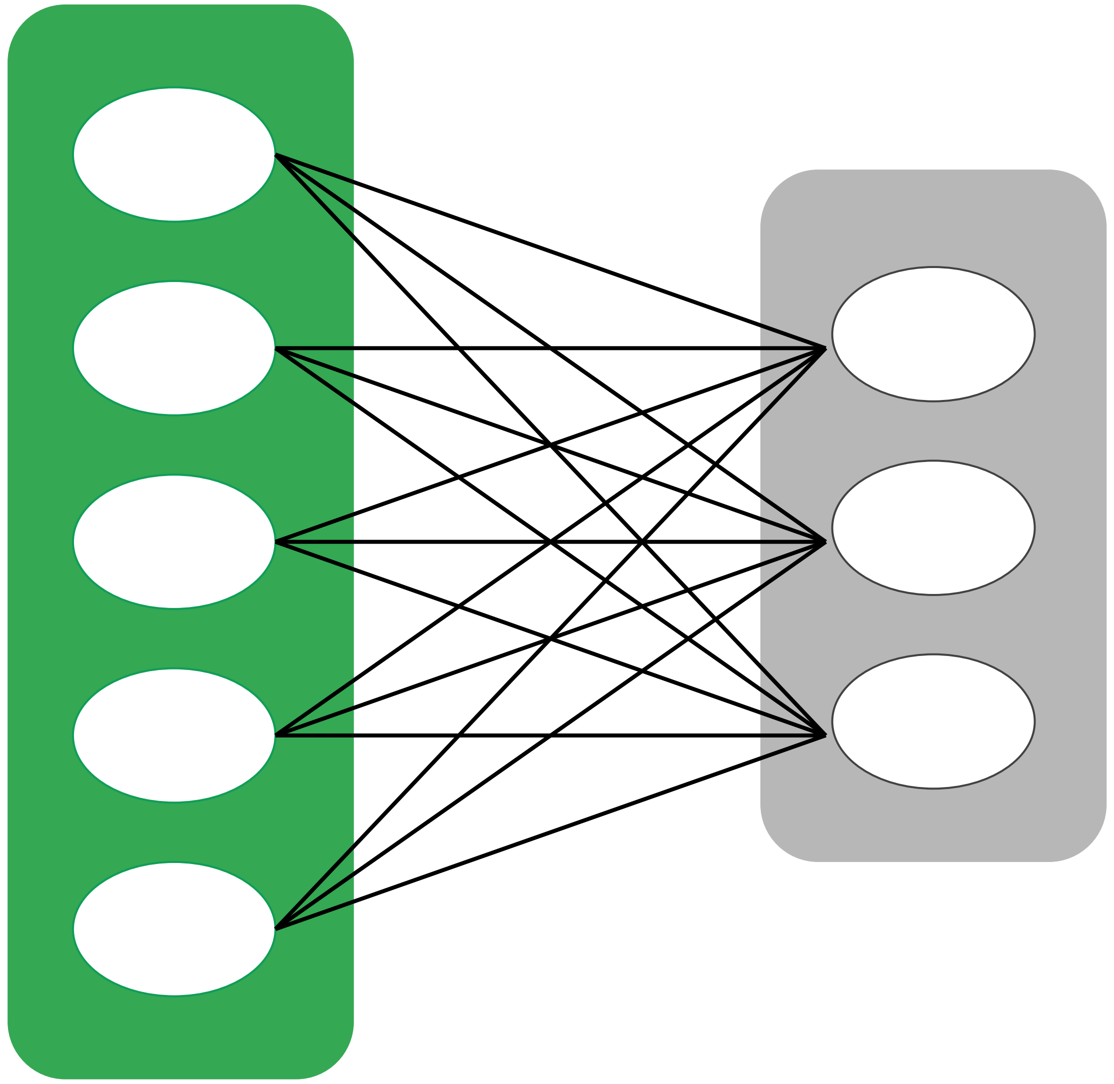
Bleu


The Dark Knight Rises

Memento

# Using a second dimension gives us more freedom in organizing movies by similarity

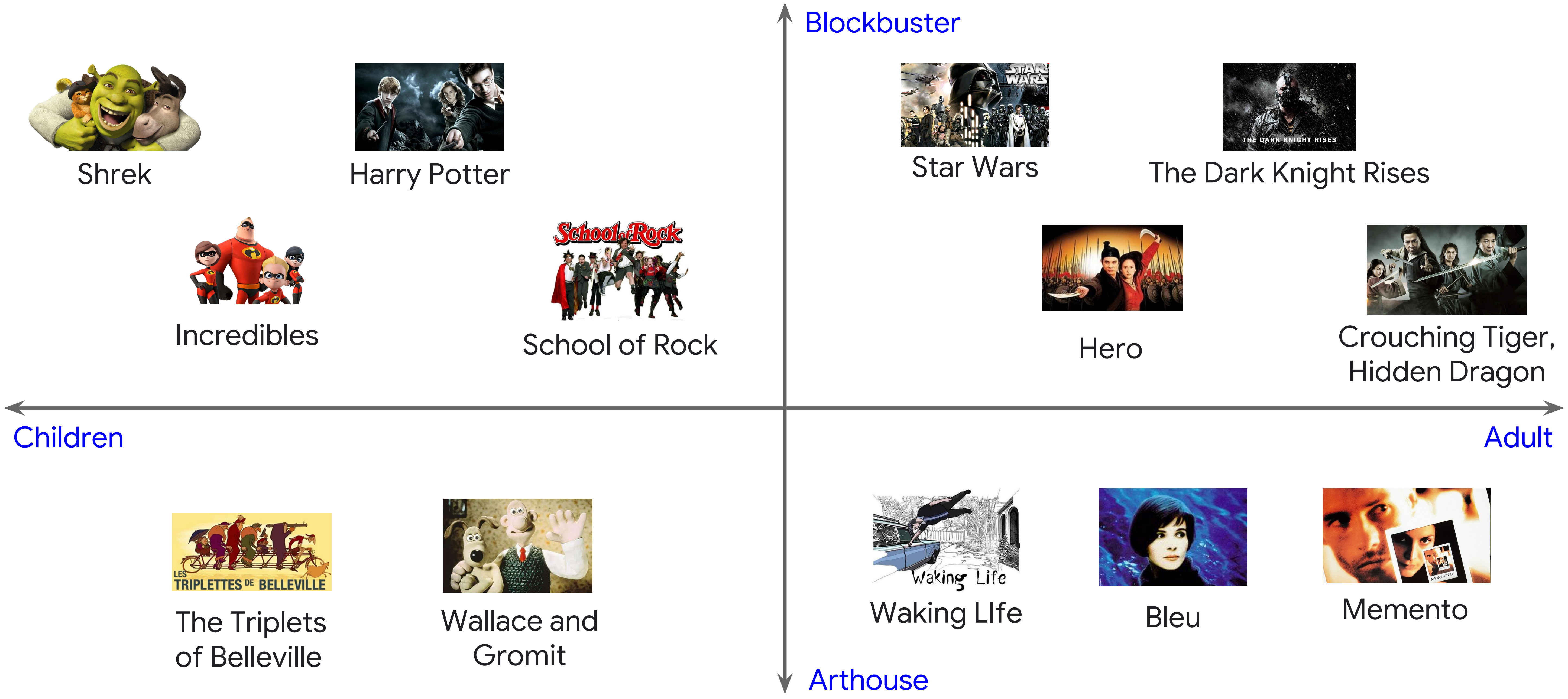
# A d-dimensional embedding assumes that user interest in movies can be approximated by d aspects



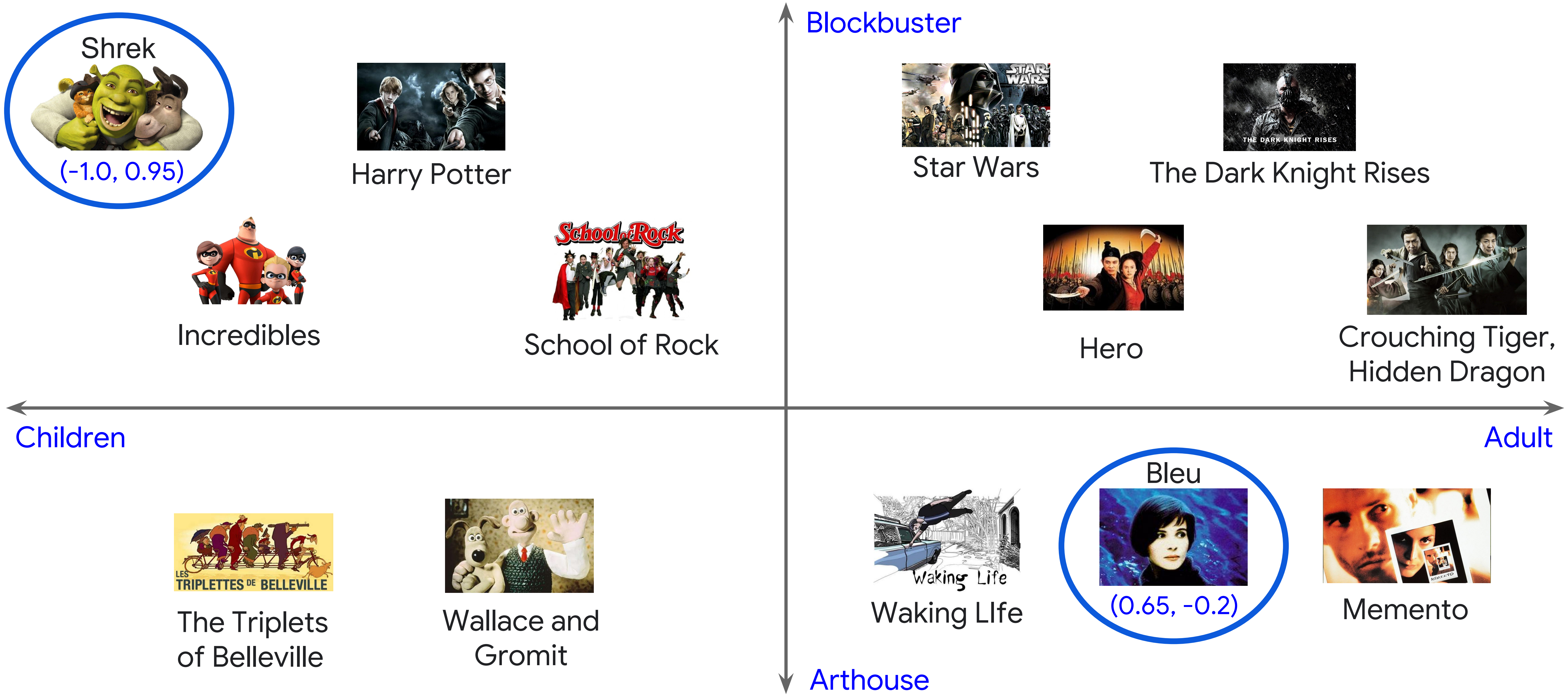
Input has N dimensions

Each input is reduced to a d-dimensional point

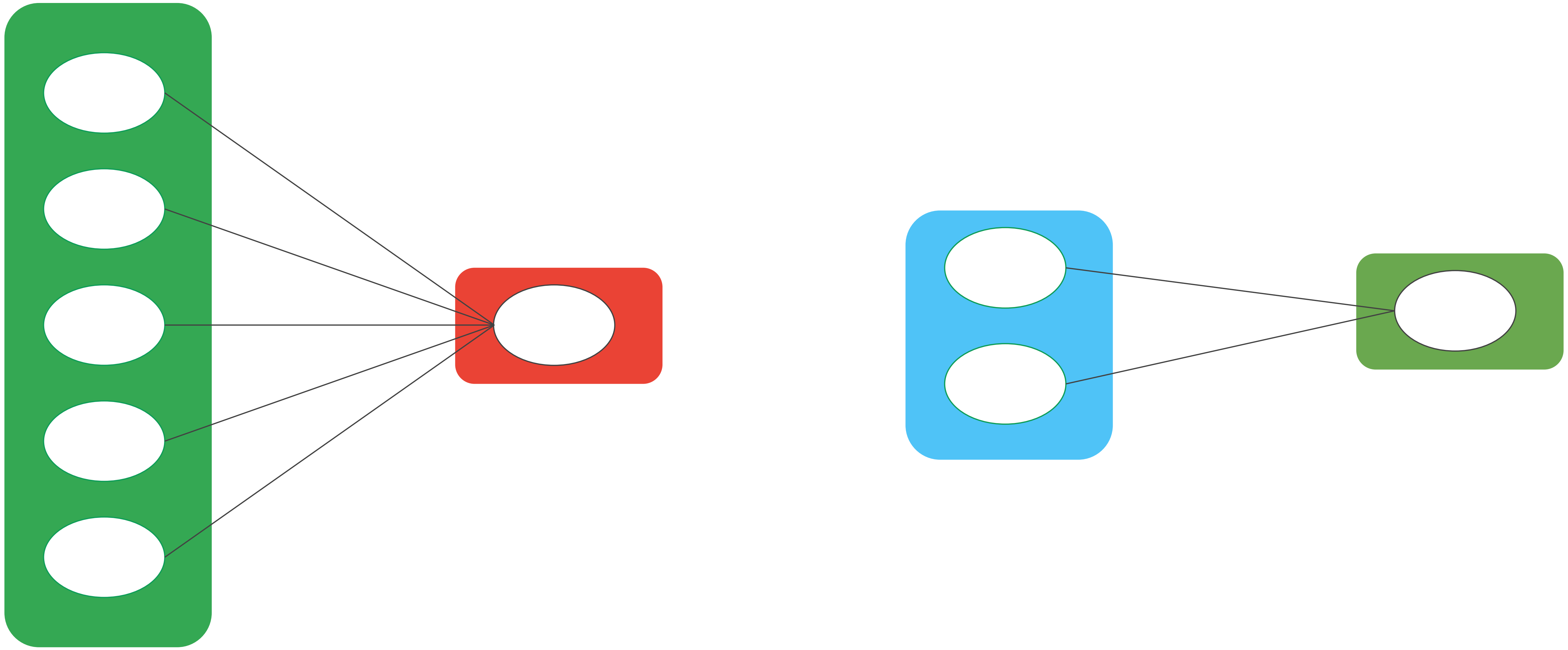
# We could give the axes names, but it is not essential ...



# The coordinates are called the 2D embedding for the movie



# It's easier to train a model with d inputs than a model with N inputs



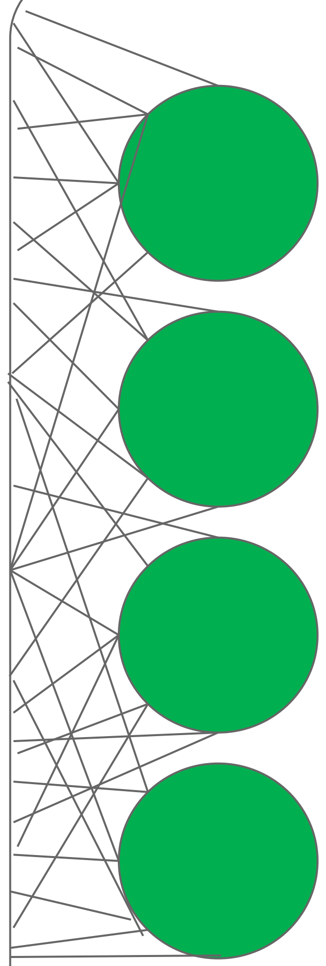
# Embeddings can be learned from data

| Example | movieId |
|---|---|
| 0 |  |
| 1 |  |
| 2 |   |
| 3 |  |
| 4 |  |

| # | Shrek | Incredible | Triplets ... | Harry Potter | Star Wars |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 |

Represent input as sparse vector

4-dimensional embedding implies we represent movies with 4 latent factors

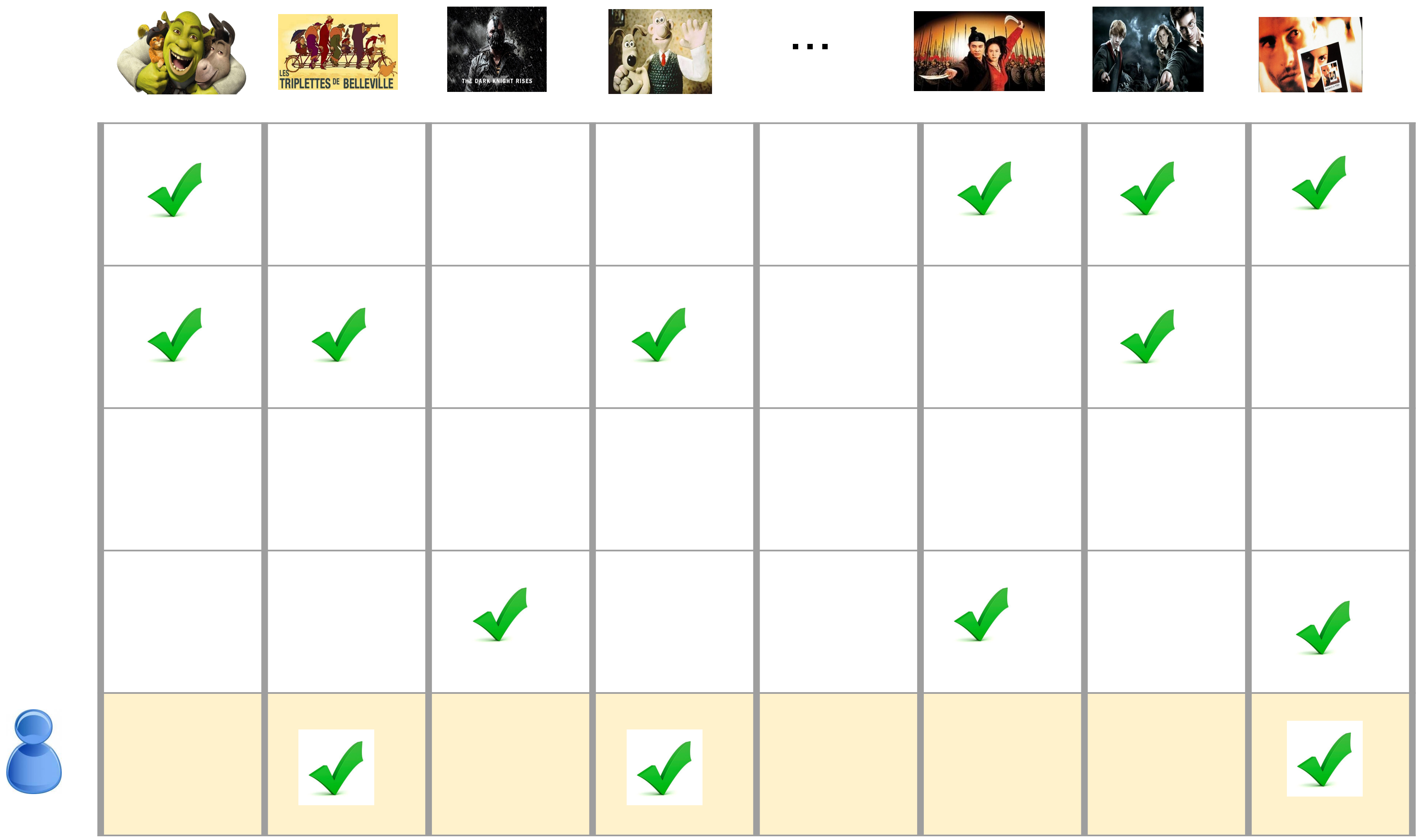


Embedded Layer

# Dense representations are inefficient in space and compute

Dense representation

(0, 1, 0, 1, 0, ..., 0, 1)

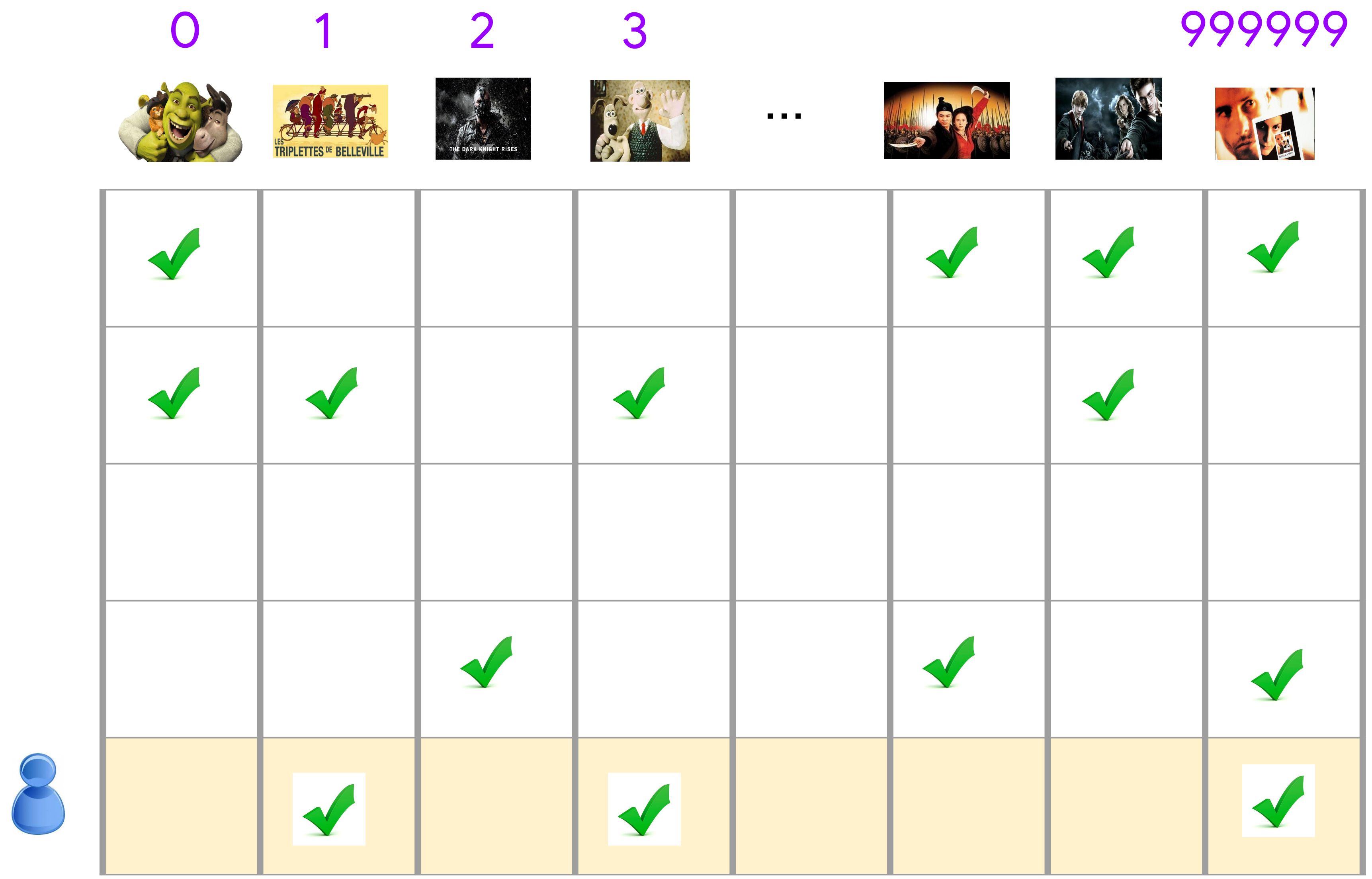


# So, use a sparse representation to hold the example

Build a dictionary mapping each feature to an integer from 0, ..., # movies - 1

Efficiently represent the sparse vector as just the movies the user watched:

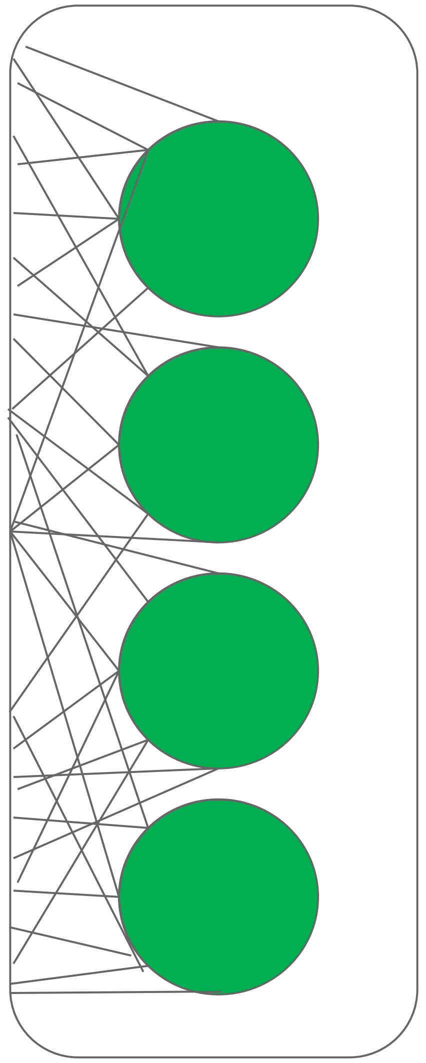
Represented as: (1, 3, 999999)



# Code to create an embedded feature column in TensorFlow

| Example | movieId |
|---|---|
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |

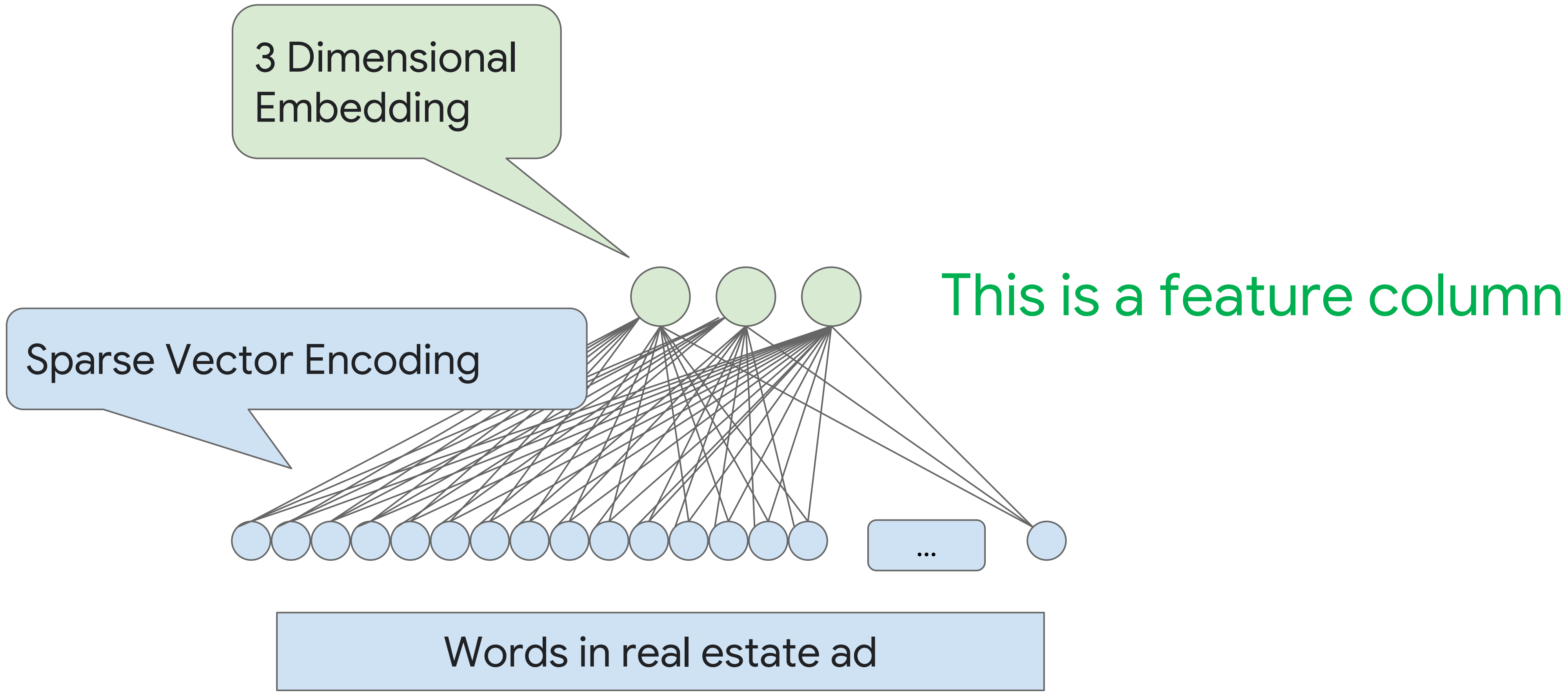
| # | Shrek | Incredible | Triplets ... | Harry Potter | Star Wars |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 |



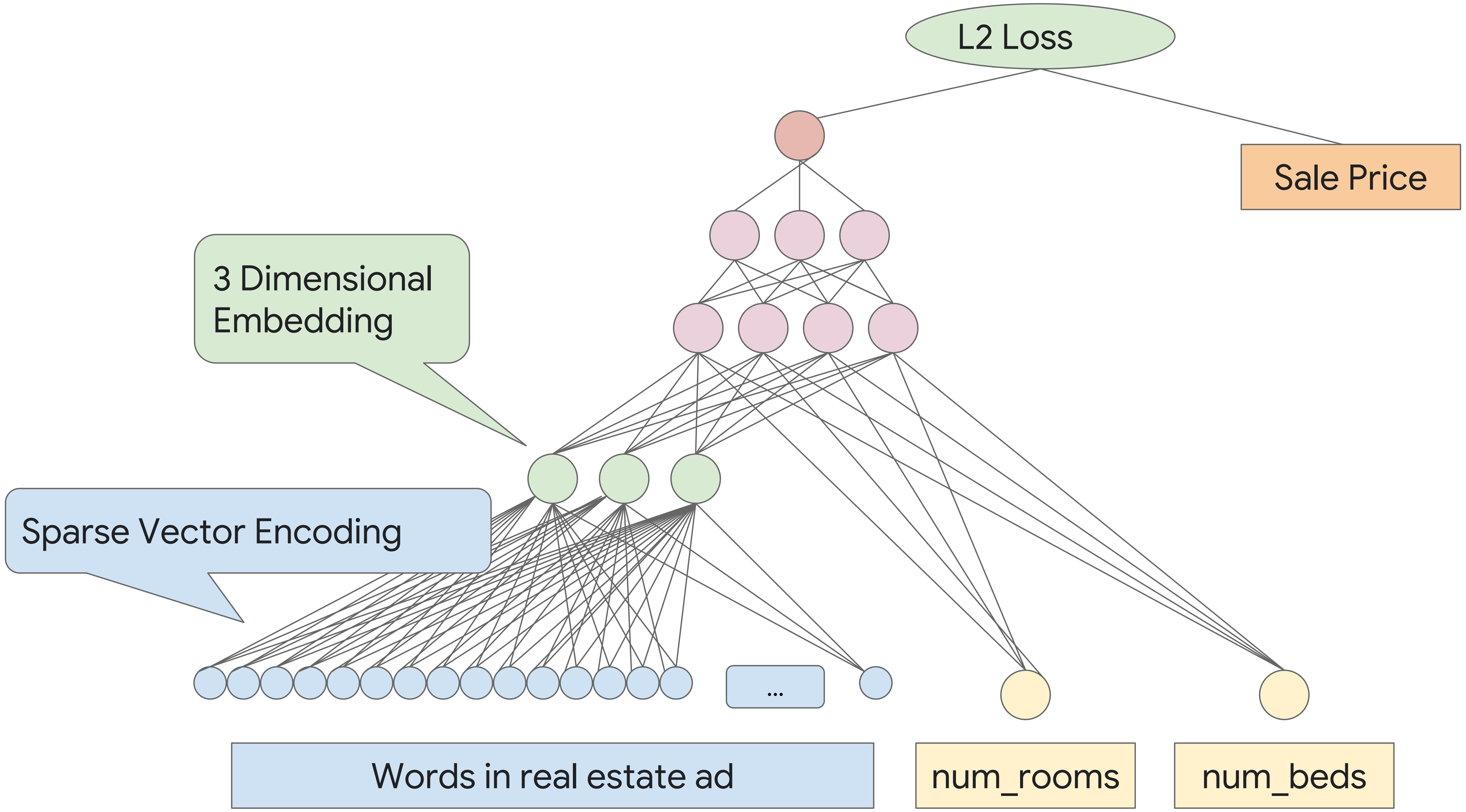
```
sparse_movie = layers.sparse_column_with_keys('movieId',
                                               keys=[...])
embedded_movie = layers.embedding_column(sparse_movie, 100)
```

# Embeddings are feature columns that function like layers

```
sparse_word = fc.categorical_column_with_vocabulary_list(
                    'word', vocabulary_list=englishWords)
embedded_word = fc.embedding_column(sparse_word, 3)
```

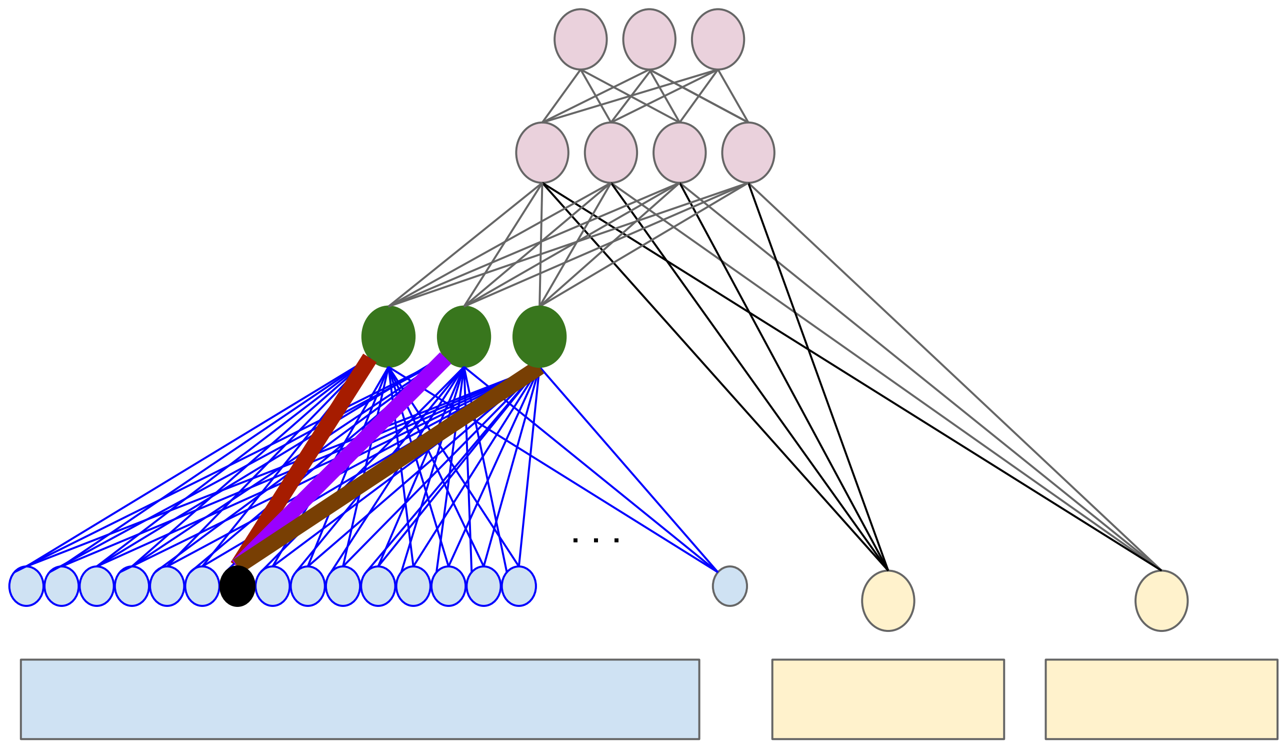


# The weights in the embedding layer are learned through backprop just as with other weights

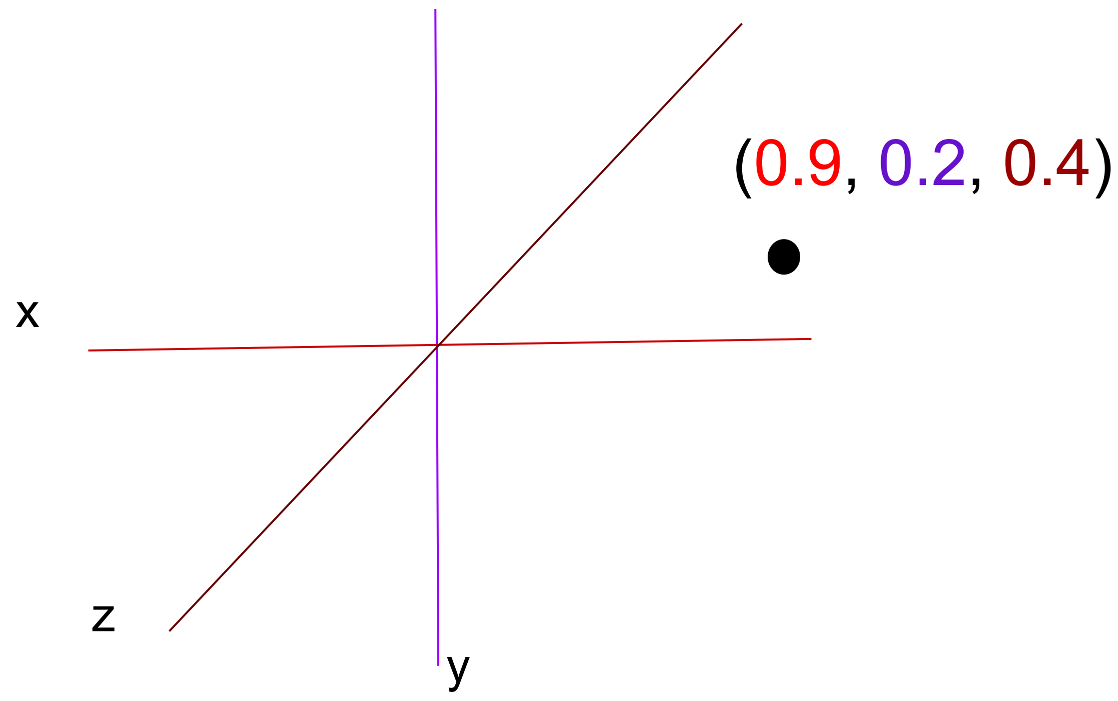


# Embeddings can be thought of as latent features

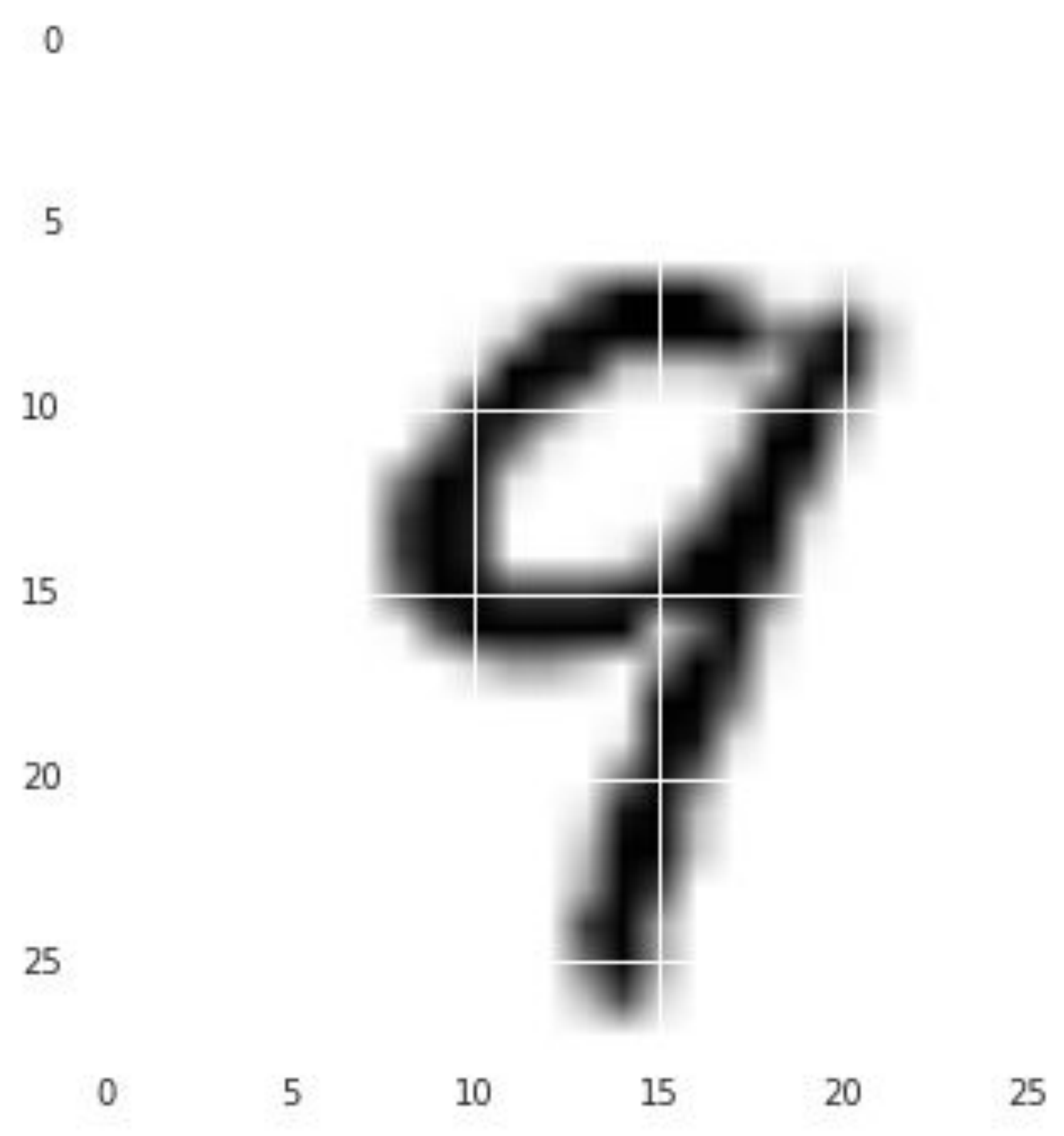
Deep Network



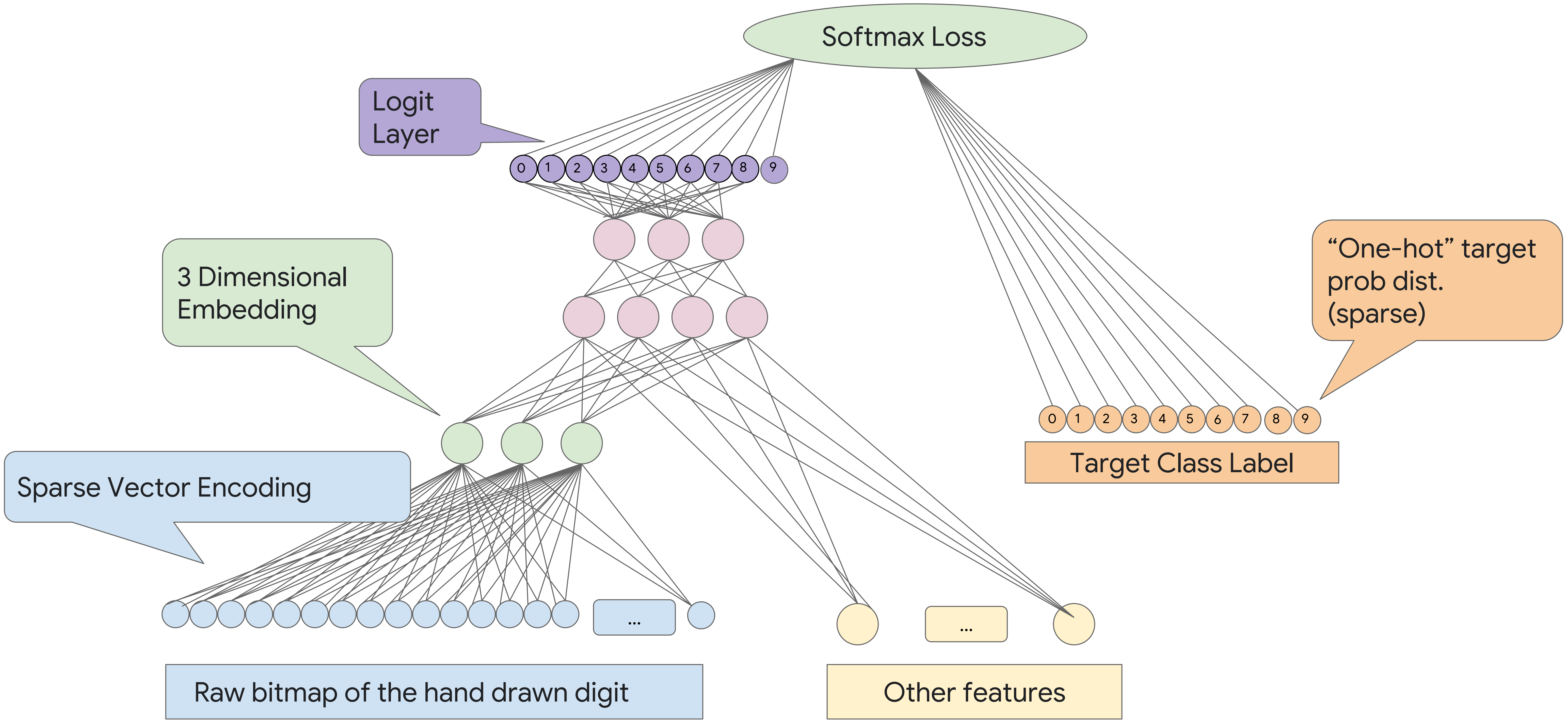

Geometric view of a single movie embedding



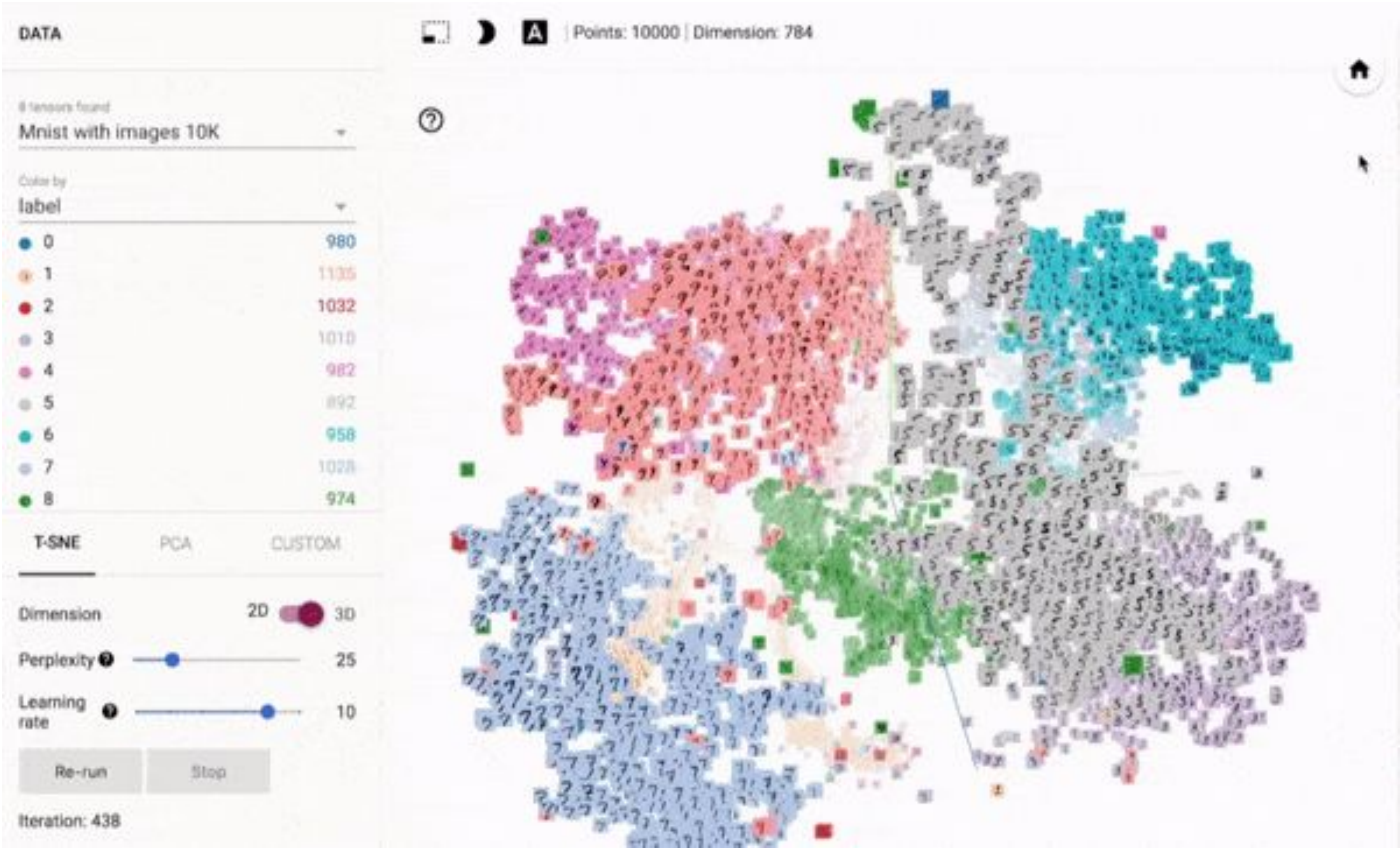
# Embeddings provide dimensionality reduction



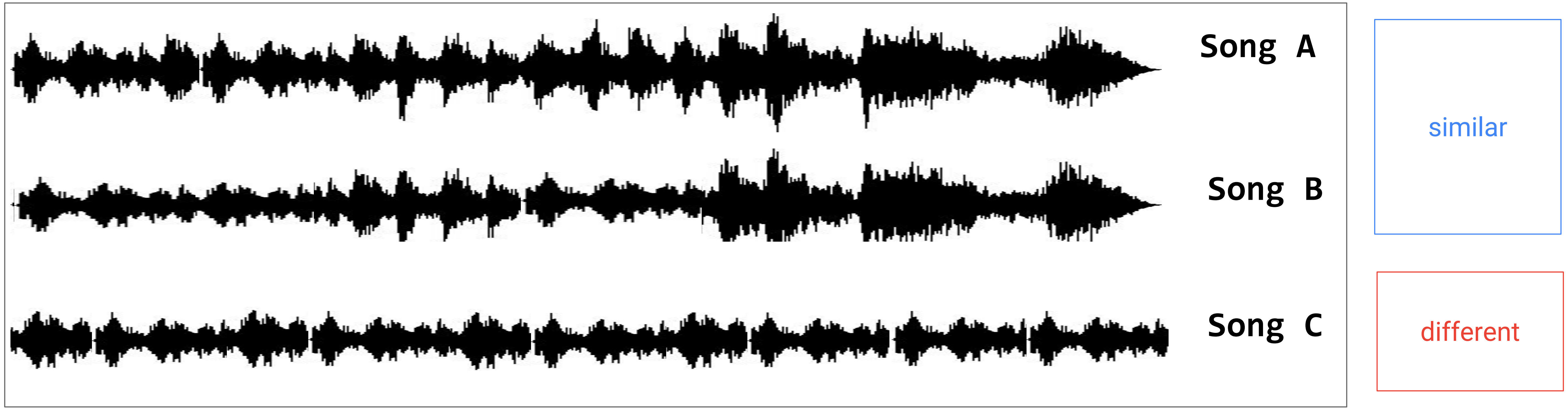
# Embeddings provide dimensionality reduction



# The result of embedding is such that similar items are close to each other



# You can take advantage of this similarity property of embeddings




# A good starting point for number of embedding dimensions

Higher dimensions -> more accuracy

Higher dimensions -> overfitting, slow training

$$dimensions \approx \sqrt[4]{possible\ values}$$

Empirical tradeoff

cloud.google.com