TensorRec: A Recommendation Engine Framework in TensorFlow

James Kirk
Oct 24, 2017 · 5 min read

When building recommendation systems, I have been frustrated by how much effort I spend on data manipulation and API-building when real progress comes from developing algorithms that better understand my users and items.

That is why I built TensorRec, a framework intended to streamline the logistics of a TensorFlow-based recommendation engine and free you up to focus on the interesting stuff: developing your ideas for representation (also called embedding) functions, loss functions, and more robust learning.

Edit 1/22/19: There is now a Getting Started guide for TensorRec

TensorRec is a recommendation algorithm with an easy API for training and prediction that resembles common machine learning tools in Python. It also gives you the flexibility to experiment with your own representation and loss functions, letting you build a recommendation system that is tailored to understanding your particular users and items.

The TensorRec project is still young, but I invite any usage, participation, or criticism that you have to offer.

In building TensorRec, I had four goals:

1. Build a recommendation engine capable of learning from explicit positive and negative feedback.
2. Allow for arbitrary TensorFlow graphs to be used as representation functions and loss functions.
3. Provide reasonable defaults for representation functions and loss functions.
4. Pack as many Machine Learning buzzwords into a Medium post as possible.

While the first three goals remain a work in progress — your mileage may vary — I'm *very* satisfied with number four.

How It Works

TensorRec scores recommendations by consuming user and item features (ids, tags, or other metadata) and building two low-dimensional vectors, a "user representation" and an "item representation". The dot product of these two vectors is the score for the relationship between that user and that item — the highest scores are predicted to be the best recommendations.

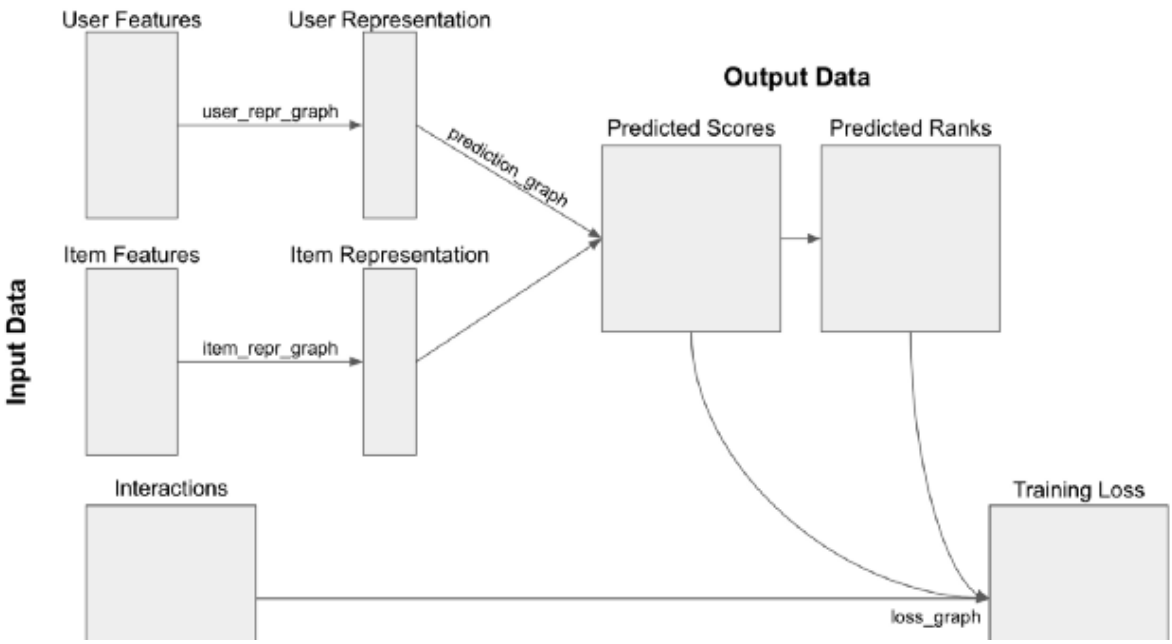
```
# Predict scores for all users and all items
predictions =
model.predict(user_features=user_features,

item_features=item_features)
```

The algorithm used to generate these representations, called the representation function, can be customized: anything from a straight-forward linear transform to a deep neural network can be applied, depending on the particulars of the problem you need TensorRec to solve and the feature data you have available. Also, the user and item representation functions can be customized independently. You can find an example of representation function customization here.

TensorRec learns by comparing the scores it generates to actual interactions (likes/dislikes) between users and items. The comparator is called the "loss function," and TensorRec allows you to customize your own loss functions as well. You can find an example of a custom loss function here.

The functions for fitting a TensorRec model are similar to fitting functions for other common machine learning tools:

```
# Fit the model for 5 epochs
model.fit(interactions, user_features, item_features,
          epochs=5, verbose=True)
```



Update 3/30/18 — System Diagram

Inspiration

I have used LightFM, developed by Maciej Kula and Lyst, extensively and I am impressed with its performance and usability. I wrote a blog post about using LightFM here.

LightFM generates user and item representations by functioning as a factorization machine and learning linear embeddings for each feature. By taking the dot product of these two representation vectors, you get a unitless score that, when ranked, tells you how good (or bad) a given user-item match would be.

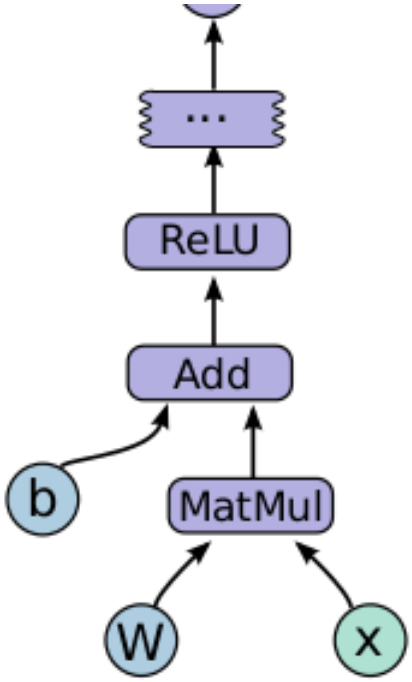
This linear factorization method is effective and computationally efficient, but I have run into issues using LightFM with imbalanced, redundant, inconsistent, or highly-correlated features — learning to

meaningfully embed these features would require a more complex representation function, such as a deep neural network. This representation function would be able to learn nuanced relationships in the user and item features and increase the overall information capacity of the system. This made me curious to explore options for representation functions that were non-linear, and I developed TensorRec as a framework to do just that.

TensorFlow for Recommendations

TensorFlow, originally developed by Google, is an open source tool that allows you to build, optimize, and distribute large, arbitrary machine learning systems.

In TensorFlow, a machine learning process is expressed as a 'graph' showing how data flows through the system. These graphs can be visualized using TensorBoard, giving a clearer explanation of the machine learning process at-hand.



A single ReLU layer of a neural network shown as a TensorFlow graph.

To build our recommendation system, we need TensorFlow graphs that accomplish four tasks:

1. Transform input data into feature tensors for easy embedding.
2. Transform user/item feature tensors into user/item representations (the representation function).
3. Transform a pair of representations into a prediction.
4. Transform predictions and truth values into a loss value (the loss function).

TensorRec solves 1 and 3 while providing modularity and reasonable defaults for 2 and 4, giving you the freedom to experiment with representation and loss functions. All you need to do is plug in a function that builds your custom representation function graph or loss function graph, like this example.

What's Missing

Many recent advances in information retrieval have come from sophisticated loss functions. Particularly interesting to me are pairwise loss functions, such as WARP, but these are challenging to represent as TensorFlow graphs. TensorRec's value to users would increase with implementation of these loss functions.

Valuable additions to TensorRec would be new features for dealing with large interaction data, managing model state, and handling implicit interactions.

If you'd like to contribute to, or even just try out, TensorRec I'd love to hear your feedback either on GitHub or in the comment section of this post.

Note: This is a personal project and, at time of writing, is not associated with Spotify

Thanks to Darien Maillet Valentine and Eric Wyler.

Machine Learning Data Science TensorFlow Recommendation System

Python