Introduction

In this Technology Review I will cover the basics of TensorFlow and its applications in recommendation engine. Topics covered will be the What Why and How of TensorFlow, its reuse in recommendation systems. Finally, its drawbacks and competitors.

What ?

TensorFlow is an open source high performance library for numerical computation. It's not restricted for machine learning but for any numerical computation. Its numeric programming library allows you to write your computation code in a high-level language like python and have it executed in a very fast way at runtime. It is Google's attempt to put the power of Deep Learning into the hands of developers around the world.

Why?

Google open sourced TensorFlow because it can empower many other companies and machine learning engineers to develop models that can be run on production systems at scale.

How?

The way TensorFlow works is that you create a directed graph or a DAG to represent the computation that you want to perform. The nodes of the graph represent mathematical operations, which could be as simple as addition subtraction or could be complex math functions like matrix multiplication. Nodes are connected by edges which represent the data flowing in and out of the nodes. The data could be as a scaler, a one-dimensional array i.e. a vector, a two dimensional array i.e. matrix. The data flows through the graph, hence the name Tensor flow[1]

Current State of Recommendation Systems

Techniques such as memory-based collaborative filtering, which uses similarity based measures to perform recommendation, do not perform well once user and item data becomes sparse, as is the case with most content and product applications. Commonly used rule-learning techniques such as alternating least squares and support vector machines have been state-of-the-art in the prior decade. Among the many recent advances in recommender systems, there have been two key concepts that help solve the challenges faced in large-scale systems: Wide & Deep Learning for Recommender Systems and deep matrix factorization[2] . In this paper, I will give a small introduction of matrix factorization.

TensorFlow for Recommendation Systems

One of the key components of any successful company is customer retention and the best way to retain customers is to recommend them relevant company products. Recommendation engines have been built for variety of use cases such as predicting the right start time for commute to work, movie recommendations on Netflix or book recommendations on Amazon. TensorFlow is one of the popular technologies used in recommendation engines because of its speed to do matrix factorization, perform multi-task learning, provide better user representations and fairness objectives. Developing comprehensive recommendation systems involves several steps starting

[1] https://app.pluralsight.com/library/courses/introduction-tensorflow-update-1/table-of-contents

[2] https://www.oreilly.com/content/when-two-trends-fuse-pytorch-and-recommender-systems/

with obtaining a training dataset, embedding the vectors, and, most importantly, the code to build the engine. To reduce the complexity TensorFlow has launched an open-source package called TensorFlow Recommenders that makes building, evaluating, and serving sophisticated recommender models easy. TensorFlow allows for building recommendation engines with or without GPU[3]

Matrix Factorization

Matrix factorization is a way to generate “latent” features when multiplying two different kinds of entities. Collaborative filtering is the application of matrix factorization to identify the relationship between items’ and users’ entities. For example, perhaps a user has rated books like “Harry Potter” and “Lord of the Rings” highly but the biography of Alexander Hamilton poorly. A latent variable that would explain this observation is that the highly rated books are part of the fantasy category and the user values fantasy books. In this case we may have a latent fantasy variable

Recommendation Systems using Matrix Factorization in TensorFlow

Author Dr. Emmanuel Tsukerman in this git[4] repo shows how to use TensorFlow to do matrix factorization for collaborative filtering. For readability purpose, I am attaching the code below also. Here the author is trying to “latent”, or hidden, variables that are responsible for users’ ratings. Here is what is happening in the code. Matrix is the user’s matrix. and matrix Q is Books(Products) matrix are created and set the loss to be the squared error between the product of the P and Q matrices and the ratings matrix. Gradient descent, which is provided out of the box from TensorFlow is selected as the optimizer. As you can see, the product of P and Q is a very close approximation of the original ratings. By increasing K, one can improve the approximation but that increases the likelihood of overfitting. Suppose that a new user has been observed with the following ratings. Step 7 and 8 show how we can predict a new user’s ratings using the history of prior ratings. The implementation is straightforward in TensorFlow. All you need is to implement an Ordinary Least Squares fit for a new row of P, which through multiplication with Q will result in the matrix R having a new row corresponding to that of the new user. TensorFlow also provides prediction functionality and with the objective being to minimize the loss[5].

Sample code of using Matrix Factorization in TensorFlow.

Code taken from git repository[6]

Step1

```
import numpy as np
```

[3] https://blog.tensorflow.org/2020/09/introducing-tensorflow-recommenders.html

[4] https://github.com/emmanueltsukerman/build-a-recommendation-engine-with-tensorflow/blob/master/Building%20a%20Recommendation%20Engine%20with%20TensorFlow.ipynb
[5] https://app.pluralsight.com/guides/building-a-recommendation-engine-with-tensorflow
[6] https://github.com/emmanueltsukerman/build-a-recommendation-engine-with-tensorflow/blob/master/Building%20a%20Recommendation%20Engine%20with%20TensorFlow.ipynb

```
npratings = np.array(    [        [3.0, 3.0, 2.0, 3.0, 3.0, 3.0],        [4.0, 1.0, 1.0, 4.0, 5.0, 3.0],        [1.0, 2.0,
2.0, 1.0, 1.0, 2.0],        [3.0, 2.0, 1.0, 3.0, 4.0, 3.0],        [1.0, 5.0, 3.0, 1.0, 1.0, 3.0],        [2.0, 5.0, 3.0,
2.0, 3.0, 4.0],        [2.0, 2.0, 1.0, 2.0, 2.0, 2.0],        [1.0, 4.0, 3.0, 1.0, 2.0, 3.0],    ])
```

Step2

```
import tensorflow.compat.v1 as tf

tf.disable_v2_behavior()

K = 2

R = ratings

N = len(ratings)

M = len(ratings[0])

P = np.random.rand(N, K)

Q = np.random.rand(M, K)

ratings = tf.placeholder(tf.float32, name="ratings")

P_matrix = tf.Variable(P, dtype=tf.float32)

Q_matrix = tf.Variable(Q, dtype=tf.float32)

P_times_Q = tf.matmul(P_matrix, Q_matrix, transpose_b=True)

squared_error = tf.square(P_times_Q - ratings)

loss = tf.reduce_sum(squared_error)
```

Step3

```
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
```

Step4

```
sess = tf.Session()
init = tf.global_variables_initializer()

sess.run(init)

for i in range(5000):

        sess.run(train, {ratings: R})

final_P_times_Q = np.around(sess.run(P_times_Q), 3)

print(final_P_times_Q)

print()

final_P_matrix = np.around(sess.run(P_matrix), 3)

print(final_P_matrix)

print()

final_Q_matrix = np.transpose(np.around(sess.run(Q_matrix), 3))

print(final_Q_matrix)

print()
```

```
[[2.721 2.93  1.932 2.721 3.375 3.185] [4.056 1.135 0.787 4.056 4.902 3.014] [0.93  2.376 1.55  0.93
1.208 1.826] [3.136 1.898 1.269 3.136 3.83  2.877] [0.846 4.844 3.146 0.846 1.206 3.099] [2.157
4.917 3.211 2.157 2.778 3.916] [1.826 1.825 1.205 1.826 2.258 2.061] [1.226 4.198 2.733 1.226
1.635 2.979]][[1.422 0.955] [2.364 0.085] [0.382 0.895] [1.75  0.488] [0.146 1.924] [0.931 1.83 ]
[0.965 0.582] [0.424 1.621]][[1.705 0.39  0.275 1.705 2.057 1.22 ] [0.311 2.488 1.615 0.311 0.471
1.519]]
```

Step4

```
new_user_indices = [1, 2, 4, 5]

new_user_ratings = [2, 2, 1, 2]

new_user_P_row_initial = np.random.rand(1, K)
```

```
new_user_P_row = tf.Variable(new_user_P_row_initial, dtype=tf.float32)

new_user_P_row_times_Q = tf.matmul(new_user_P_row, final_Q_matrix)

res = tf.gather(new_user_P_row_times_Q, new_user_indices, axis=1)

squared_error = tf.square(new_user_ratings - res)

loss = tf.reduce_sum(squared_error)

optimizer = tf.train.GradientDescentOptimizer(0.01)

predict = optimizer.minimize(loss)
```

Step5

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
for i in range(50000):
	sess.run(predict)
```

Step6

```
final_new_user_P_row_times_Q = np.around(sess.run(new_user_P_row_times_Q),
3)print(np.round(final_new_user_P_row_times_Q))
[[1. 2. 2. 1. 1. 2.]]
```

Drawbacks

We just saw building recommendation systems using TensorFlow is relatively easy and also TensorFlow is great for large scale systems but it does have some drawback. It's slower than its competitors and primarily designed for Linux systems and provides less support for Windows users. However, Windows users can overcome this limitation by using the Anaconda prompt or the pip package. Recently TensorFlow 2.0 was launched which introduced a lot of breaking changes making the transition from 1.0 to 2.0 difficult for users.[7]

Competitors

The current biggest competitor to TensorFlow is PyTorch which is an open-source machine learning library developed by Facebook. Amongst its advantages is it is built on Python which is the

[7] https://careerfoundry.com/en/blog/data-analytics/pytorch-vs-tensorflow/#pytorc

preferred language for data scientist and machine learning engineers. It is easy to learn and use and supports a hybrid UI[8]

[8] https://careerfoundry.com/en/blog/data-analytics/pytorch-vs-tensorflow/#pytorc