MIlestone 1 Proposal and Data Selection

Neural Collaborative Filtering with Non-Negative Matrix Factorization

Muduo Wang

Fall 2021

https://github.com/Mwang413/Mwang413

Which Domain?

The domain is the same as the first project, with one exception. I'd imagine a data scientist is more likely to work with the first project, which is implementing a pre-trained model and perhaps training a model with given architecture.

However, for this second project, it is more likely that a machine learning engineer will experience this work, since it involves a lot more research and personally developing and improving an algorithm, rather than simply using an existing one and working more with the data.

Original Research Article

- https://arxiv.org/abs/1708.05031

Original code

- https://github.com/hexiangnan/neural_collaborative_filtering

A different implementation of the NCF architecture
Tensorflow implementation (Model Garden)

- https://github.com/tensorflow/models

Matrix Factorization choices to select for improvement on the current architecture
Matrix Factorization

- https://en.m.wikipedia.org/wiki/Matrix_factorization_(recommender_systems)

Non-Negative Matrix Factorization

- Introduction:
  - https://en.wikipedia.org/wiki/Non-negative_matrix_factorization
- Advancement Research:
  - https://papers.nips.cc/paper/2000/file/f9d1152547c0bde01830b7e8bd60024c-Paper.pdf
  - https://blog.acolyer.org/2019/02/18/the-why-and-how-of-nonnegative-matrix-factorization/
  - https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html

Neural Network and modeling options to select for improvement on the current architecture

- Advancement Research:
    - ANN vs CNN vs RNN: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/
    - XGBoost https://xgboost.readthedocs.io/en/latest/
    - Random Forest vs XGBoost vs Deep Neural Network https://www.kaggle.com/arathee2/random-forest-vs-xgboost-vs-deep-neural-network

Which Data?

(See project 1)

I was also given a dataset that my workplace received that eventually became the training and testing data. This contains around 10 million interactions between users and items. A lot was done also on this dataset, including shrinking, partitioning, and creating negative instances.

Research Questions? Benefits? Why analyze these data?

(See project 1.)

- What can be improved about the NCF?
- How well does the NCF perform on different kinds of data?

Improvements to the NCF will be beneficial to recommending new interactions between existing users and items. The most immediate use case of this development is to reduce the cost of unuseful solicitation and find better ways of creating revenue through products. Like a movie that you haven't seen on Netflix, or an Amazon item that you might likely purchase. However, this is certainly not the only thing it can contribute to, recommendation models also create new opportunities for adventures between all aspects of life and the people in it: Traveling, books, educational courses, interests, even Tinder matches.

What Method?

The method for this project is largely a continual research and implementation cycle, examining each layer and its mathematical properties. The project will implement each layer of the NCF, including General Matrix Factorization, Multi-Layer Perceptron, and Neural Matrix Factorization, while inserting the Non-Negative Matrix Factorization in

place of the GMF. And finally, the model will be evaluated based on its hit rates on the test data and labels, which is a technique often used for recommendation models.

Potential Issues?

This project's main challenge is using customized layers within a Keras model training process.

A second challenge is the size of data in training the model. The training process often returns Out-Of-Memory errors for larger datasets. Datasets are partitioned in training and in testing.

A third challenge was creating an evaluation metric that fit the data properly. This took a chunk of time to make sure the model was evaluated correctly on the given data.

Concluding Remarks

In summary, this project aims at improving the original NCF infracture, and insert a NNMF to improve the NCF's performance on different types of data to give better recommendations.

Milestone 2 **Check Point**

Any surprises from your domain from these data?

There are no surprises in the data. However, a lot of surprises came from the model building. The work that I am doing is actually quite new, even the original publication is as recent as 2017. Because of this, there aren't a lot of answers to a lot of my questions. I had the true experience of a research scientist through this project, doing something I have never seen before. Not that it's never been done before, since I borrowed a lot of the codes that I used in my research. The process included lots of combining of previously existing architectures, into new interactions between them.

Is the dataset what you thought it was?
The dataset is very basic, just IDs of users and items, denoting positive interactions between users and items. I did not have too much expectation of the model (which is like the "data" of my project). However, it didn't take too long to understand each part of the model's infrastructure, but some of the original NCF's code syntax was very difficult to work with and make improvements to. The surprising and frustrating part of the model was that I could not see the results and transformation to the data at the end of each layer. I eventually worked through this, but this felt very unintuitive.

Have you had to adjust your approach or research questions?
Yes, I had to adjust my approach many times, and try and fail with different infrastructures, tools, and codesets. I had to use different NCF codes because some of them did not work on my computer or on cloud because of its Keras and Python version. I had to also find multiple versions of the NNMF to figure out how to insert it into the NCF.

Is your method working?
My method at the second week check point was not working yet. I had to figure out solutions to make things actually run, which was hard because Keras kept returning errors that were found in the package code, and I had researched hundreds of hours to find solutions to these errors. In the end, I adopted the third set of the NCF infrastructure. The code quickly ran, and I was able to start working on the NNMF change by the third and fourth week.

What challenges are you having?
This project's main challenge is using customized layers within a Keras model training process. I had a hard time using custom layers since I am in no way an expert with Keras, so doing something custom involved me spending hours and hours learning

how Keras works. A second challenge is the size of data in training the model. The training process often returns Out-Of-Memory errors for larger datasets. Datasets are partitioned in training and in testing. A third challenge was creating an evaluation metric that fit the data properly. This took a chunk of time to make sure the model was evaluated correctly on the given data. The final challenge was making sure the results were actually correct, checking for bugs in the process.

Foreword

This project is a continuation of the previous project on the implementation and the description of the mathematical properties and the infrastructure of the recommendation model: Neural Collaborative Filtering (He et al., 2017).

The second part, which is turned into the second project, will focus on resolving some of the issues that the NCF model faces, improving its algorithm with a specific technique - Non-Negative Matrix Factorization (NMF).

Objective

The end result of the project is to develop a better model that would help create better recommendations for interactions between items and users that have not yet happened but are statistically likely to happen. The most immediate use case of this development is to reduce the cost of unuseful solicitation and find better ways of creating revenue through products. However, this is certainly not the only thing it can contribute to, recommendation models also create new opportunities for adventures between all aspects of life and the people in it: Traveling, books, educational courses, interests, even Tinder matches.

The general matrix factorization is a barebones model within the NCF architecture. There are plenty of newer matrix based operations that are used today as standalone models, and this matrix factorization could be replaced with an enhanced and more modern version.

The proposal is to suggest different models to potentially perform well or better on this data, or different data. I intend to replace the General Matrix Factorization with a Non-negative Matrix Factorization, there has been research suggesting better

performances. I will also be replacing the basic MLP ANN model and attempt to reconstruct the model with other deep learning methods.

Issues with NCF

The NCF model can be hard to use, difficult to implement, and performs poorly on sparse data. As an extension of the previous project, the image below shows the result of the NCF model predicting top 8 interactions, and the percentage of times that test “user-item” interaction falls into one of the predictions.

```
we are at step: 100
the hit ratio at this step is 0.19
we are at step: 200
the hit ratio at this step is 0.18
we are at step: 300
the hit ratio at this step is 0.16
we are at step: 400
the hit ratio at this step is 0.16
we are at step: 500
the hit ratio at this step is 0.15
we are at step: 600
the hit ratio at this step is 0.15
we are at step: 700
the hit ratio at this step is 0.15
we are at step: 800
the hit ratio at this step is 0.16
we are at step: 900
the hit ratio at this step is 0.16
we are at step: 1000
the hit ratio at this step is 0.17
```

To review, this hit ratio represents the performance of the model in giving predictions based on test data that “hits” the test labels. The test data is a set of interactions, ordered by the keys belonging to users, where each user may interact with several different items. The activational layer of the model assigns a probability ratio to each of

its categories, which are the ID of the users. The users given highest probability values by the model are the top recommendations given by the model. The test label gives the user-item interaction that the model is trained to predict. The model aims to predict the interaction in its top given number of interactions. If the model's top predictions contain the test interaction, then the model has produced a "hit." The more hits that the model produces, the higher its usability and value.

The NCF model gives around 15-20% of the correct recommendations within the test data. Due to this low performance, there have been a lot of different recommendation models published since then. Papers With Code presents a nearly exhaustive collection of high performing recommendation models, including Global-K, VASP, and MG-GAT (Papers With Code, 2020).

The Google research team has published an article presenting their review and use of the NCF model. The article states that:

> In [our] work, we revisit the experiments of the NCF paper that popularized learned similarities using MLPs. First, we show that with a proper hyperparameter selection, a simple dot product substantially outperforms the proposed learned similarities. Second, while a MLP can in theory approximate any function, we show that it is non-trivial to learn a dot product with an MLP. Finally, we discuss practical issues that arise when applying MLP based similarities and show that MLPs are too costly to use for item recommendation in production environments while dot products allow to apply very efficient retrieval algorithms. We conclude that MLPs should be used with care as embedding combiner and that dot products might be a better default choice.

Replacing GMF

This research aims at resolving this issue by replacing the General Matrix Factorization layer within the NCF infrastructure. This is the API documentation describing the details of the Keras Embedding Layer, used by the original NCF model. Only a short description is provided for the function of this layer: "Turns positive integers

(indexes) into dense vectors of fixed size" (Keras, 2020). This is an unfortunately short description of the function of the process of embedding conversion. To put in another way, the Embedding Layer is rather like a one-hot encoded dictionary, without the one-hot encoding. Saxena gives a detailed description on the process of embedding a vector (2020). The function of the Embedding Layer is to represent input data, containing weights that connect the input data to a look-up table.[1]

The original NCF architecture transforms the input data, connecting these Embedding Layers after the Input Layer. The NCF then connects the two resulting embeddings with a dot-product. Finally, the dot-product layer is connected to an activational dense layer. The research replaces this process with a different approach, namely, Non-Negative Matrix Factorization.

NNMF

NNMF is based on the mathematical theorem that states "any polynomial with no constant or linear terms can be factored as a product of matrices with no constant terms" (Eisenbud, 1980). This is later developed into a factorization algorithm that minimizes the squared error of an original matrix and the resemblance returned by its two factors. The graphical illustration below is a simplified demonstration of the mathematical process of the NNMF operation (MH1200, 2014).

Let A be a matrix such that it is approximated by the dot product of matrices C * R.

[1] For full disclosure, this research lacks more detail around this subject (partially due to its author's lack of understanding). More work needs to be done differentiating the difference between an Embedding Layer and Non-Negative Matrix Factorization.

A11	A12	A13	A14
A21	.	.	.
A31	.	.	.
A41	.	.	.

=

<table>
<tr><td>C11</td><td>.</td><td>.</td><td>.</td><td rowspan="3">*
(element-wise dot-product)</td><td>R11</td><td>R21</td><td rowspan="3">(Add more columns for higher dimensions for factors to increase fitting on data.</td></tr>
<tr><td>C12</td><td>.</td><td>.</td><td>.</td><td>.</td><td>.</td></tr>
<tr><td colspan="4">(Add more rows for higher dimensions for factors to increase fitting on data.</td><td>.</td><td>.</td></tr>
</table>

Then:
A11 ≅ C(1,1) x R(1,1) + C(1,2) x R(2,1) + C(1,3) x R(3,1) + etc.
A12 ≅ C(1,2) x R(1,2) + C(1,2) x R(2,2) + C(1,3) x R(3,2) + etc.
A21 = C(2,1) x R(2,1) + C(2,2) x R(2,2) + C(2,3) x R(2,3) + etc.
etc.
.
.

That is, every item in the original matrix is approximated by the dot product of the item's corresponding column in matrix C and its corresponding column in matrix R.

In essence, the Matrix Factorization Layer performs a dot product operation to create a similarity matrix between the Embedding Layers derived from the “Users” Input Layer and “Items” Input Layer. Generally, gradient descent is used for NNMF to find the minimum of error. However, this sometimes cannot find the global minimum of error, instead settling for a local minimum. This research implements a multiplicative update process to find the global minimum. The article “Multiplicative Update” presents the research, stating:

> while gradient descent is a simple procedure, convergence can be slow, and the convergence can be sensitive to the step size. In an attempt to overcome this, Lee and Seung applied multiplicative update rules, which have proved particularly popular in NMF applications since then (Plumbley et al., 2010).

This is the source code of the NMF implementation in Keras and Python 3 (Tran, 2019). This layer is used as a custom layer within a Keras model, as opposed to “traditional layers” (Tensorflow, 2020).

```
class PredictedIJ(layers.Layer):

    def __init__(self, k, **kwargs):
        self.k = k
        super(PredictedIJ, self).__init__(**kwargs)

    def build(self, input_shape):
        self.lammbda = self.add_weight(
            name='lammbda',
            shape=(self.k, ),
            initializer=initializers.Constant(1 / self.k),
            constraint=constraints.NonNeg(),
            trainable=True
        )
          super(PredictedIJ, self).build(input_shape)  # Be sure to call this at the end

    def call(self, x):
        # x_i is n x k, x_j is n x n_pairs x k
```

```
        x_i, x_j = x
        lammbda = tf.linalg.tensor_diag(self.lammbda)
        x_i = x_i @ lammbda
                    return backend.squeeze(tf.matmul(x_i[:, None, :], x_j,
transpose_b=True), axis=1)[:, :, None]

    def compute_output_shape(self, input_shape):
        return (input_shape[0][0], input_shape[1][1], 1)
```

Results

The model performs surprisingly well. It has a hit ratio of 100% with top 8 interactions. To recall from earlier, the NCF model only hits the test interaction around 15-20% of the times within its top 8 predicted interactions.

```
we are at step: 100
the hit ratio at this step is 1.00
we are at step: 200
the hit ratio at this step is 1.00
we are at step: 300
the hit ratio at this step is 1.00
we are at step: 400
the hit ratio at this step is 1.00
we are at step: 500
the hit ratio at this step is 1.00
```

Even at top-4 items, the model still returns 100% hit rate. However, at top-3 items, the model returns 0% hit rate. This is a weird behavior, and more work needs to be done in this regard.

In conclusion, the model that this research presents is an improvement upon the original NCF infrastructure, which combines the MLP and GMF models and presents a 15-20% hit ratio for top-8 interactions. The improved model which uses NMF returns a 100% hit ratio for top-4 to top-8 interactions.

Challenges

This project's main challenge is using customized layers within a Keras model training process. There are a lot of difficulties implementing any kind of changes to a model that is not in the form of the "traditional layers." A process called "eager execution" in Keras which deals with viewing results of a model's layers at the end of each layer. This has an inconsistent behavior in some models and some processes, which increases difficulties when creating a customized layer of a model. Time was most spent in completing the task of fitting a custom layer with the compatible Keras layer into the original NCF. Multiple NCF architectures beyond the original publication was used for attempting at this task, and many of them besides one was finally able to be used.

A second challenge is the size of data in training the model. The training process often returns Out-Of-Memory errors for larger datasets. Datasets are partitioned in training and in testing.

A third challenge was creating an evaluation metric that fit the data properly. This took a chunk of time to make sure the model was evaluated correctly on the given data.

Questions

1. If you could redo the project again, what would you have done differently?
2. Are there ways to improve the neural-network? How?
3. Are there ways to improve the custom NNMF layer? How?
4. Are there better algorithms to use for the NNMF layer? Which?
5. Would you have done more training if given the time? Why?
6. Will the data show even better contrast of results to the NCF on data that is more sparse?
7. Which platform would you choose to perform your next training task?
8. What were some ways that helped with research done for model training and improvement?
9. What tools would you have liked to have more knowledge of with tasks like these?
10. Was this research project rewarding?
11. What areas of this project deserve more study and time?

References

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chura, T. (2017). *Neural Collaborative Filtering.* ArXiv. https://owl.purdue.edu/owl/research_and_citation/apa_style/apa_formatting_and_style_guide/reference_list_author_authors.html

Papers With Code (2020). *Recommendation Systems.* Paperswithcode.com https://paperswithcode.com/task/recommendation-systems

Plumbley, M., ... Bro, R. (2010). *Non-negative mixtures.* Multiplicative Update, Handbook of Blind Source Separation. https://www.sciencedirect.com/topics/computer-science/multiplicative-update

Saxena, S. (2020). *Understanding Embedding Layer in Keras.* Medium. https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce#:~:text=Embedding%20layer%20enables%20us%20to,way%20along%20with%20reduced%20 dimensions

Steffen, R., Krichene, W., Zhang, L., & Anderson, J. (2020). *Neural Collaborative Filtering vs. Matrix Factorization Revisited.* Google Research. https://research.google/pubs/pub50164/

Tensorflow (2020). *Custom Layers.* https://www.tensorflow.org/tutorials/customization/custom_layers

Tran, A. (2019). *model.py.* Github. https://github.com/allentran/keras-nmf/blob/master/keras_nmf/model.py