Published in DeepBenBello · Follow

Oscar de Felice · Follow
Aug 28, 2020 · 10 min read ★

# A Deep Recommender Sy

How to build an hybrid recommender system based on triplet loss, embedding layers and making use of Keras.

The aim of this post is to describe how one can leverage a deep learning framework to create a hybrid recommender system *i.e.* a model exploiting both content and collaborative-filter data. The idea is to tackle issues in two different steps: first collaborative filtering and content based model separately, then a combination of the two, to get better results.

## Introduction

Before diving into recommender system, we have been recommended to introduce ourself.

Войдите в приложение "Medium" с аккаунтом Google

Михаил Соколов
sokolovmisha44@gmail.com

Продолжить как Михаил

Чтобы создать аккаунт, Google предоставит приложению "Medium" ваше имя, адрес электронной почты и фото профиля. См. Политику конфиденциальности и Условия использования сервиса "Medium".

We are a team of four good friends with different coding backgrounds, ranging from very limited to advanced coding experience. We all come from Physics studies (different specialisations, though), and we work in different areas. By creating **DeepBenBello.ai**, we made a little dream come true. We wanted to have something to work on together, to start engaging with tech applications and to create something beautiful. Hence this story is the first of a long series, we hope. Stay tuned!

Back to the recommender system, the goal is to design and develop an hybrid algorithm based on *Bayesian Personalised Ranking triplet loss* and implementing such a system in Keras.

### What Types of Recommender Systems Exist?

As a quick introduction, let’s try to briefly describe the various types of recommender system. We can split recommender systems in two classes:

- Collaborative filtering

**Collaborative filtering** assumes that users with
in the future. Netflix uses a variant of CF to reco
on similar users' tastes in similar movies. **Conte**
items in the future that share features — like br
the past. Amazon uses a variant of CBF to recom

## Encoding and Embeddings

Often data are not as simple as we would like th
can digest (that is *numbers*) we can make use of

Both encoding and embedding map categorical data in numerical vectors. The difference between the two is the fact that an encoder (like **one-hot encoder**) is a predetermined function associating a vector to each data row, while embedding vectors are low dimensional and learned. A neural network learns how to locate objects in an embedding space, placing similar entities close to each other.

To summarise, an **encoding** maps a categorical feature in a *m*-dimensional vector, where *m* is the number of categories of the feature. An **embedding** maps categorical features in an *n*-dimensional vector, where *n* is an hyper-parameter of the model.

A more detailed discussion about embedding can be found in the excellent post below.

**Light on Math ML: Intuitive Guide to Understanding GloVe Embeddings**
Understanding theory behind GloVe and Keras implementation!
towardsdatascience.com

The reason we prefer to use embeddings here is because it does not really make sense to treat each categorical value as being completely different from one another (this is **one-hot-encoding**). In our working case, we aim to build an hybrid recommender system to propose movies to users. Imagine you want to take into account user age range to get rid which movies they are more likely to appreciate. Is it correct to consider "equally different" a 29-years old user from a 31-years old and from a 62-years old? Embeddings will solve this issue. Indeed, we can make use of this technique to "learn" the relationships and inner connections between each possible value and our target variable.

compared to a single One-Hot-Encoding approa
that is the ***Embedding space*** — in where similar

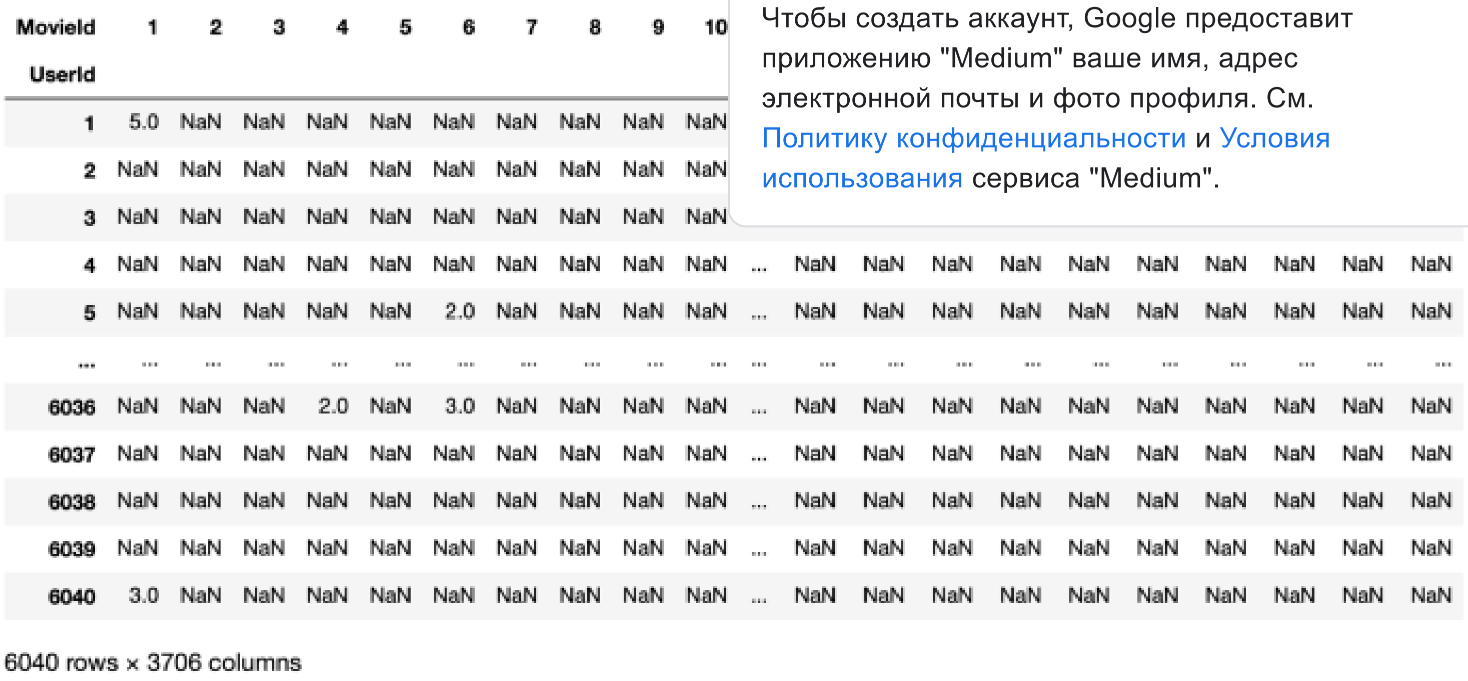
Finally, let's have a look at our data.

## The dataset

For this application we use the notorious *moviel*
recommender system, taking advantage from bo
user/movie ratings. These data are stored in thr
`ratings.dat` .

The *readme* of the dataset contains a quite exhaustive explanation of the files. User information contains gender, occupation, age range and zip-code (note: we drop the latter as it is plenty of zip-codes referring to just one user), while movie dataset contains (further than movie id) title and genre. Ratings are what will drive the supervised training.

Ratings dataframe can be reshaped in a (sparse) matrix as follows



| MovieId | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| UserId | | | | | | | | | | | | | | | | | | | | | |
| 1 | 5.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | | | | | | | | | | | |
| 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | | | | | | | | | | | |
| 3 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | | | | | | | | | | | |
| 4 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 5 | NaN | NaN | NaN | NaN | NaN | 2.0 | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 6036 | NaN | NaN | NaN | 2.0 | NaN | 3.0 | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 6037 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 6038 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 6039 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 6040 | 3.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

6040 rows × 3706 columns

As one can see the matrix is sparse, as the majority of couples user-movie is not rated.

Here how we are going to use the information stored in this matrix. Our training set will be made of **triplets** *[user, liked movie, not liked movie]*. This takes as inspiration a neural network architecture for image recognition, called *Siamese Architecture*. According to Wikipedia, Siamese Neural Network is defined as

> **Siamese neural network** is an artificial neural network that use the same weights while working in tandem on two different input vectors to compute comparable output vectors. Often one of the output vectors is precomputed, thus forming a baseline against which the other output vector is compared. This is similar to comparing fingerprints or more technical as a distance function for Locality-sensitive hashing.

This is often used in conjunction with *Triplet loss*, again according to Wikipedia:

> **Triplet loss** is a loss function for artificial neural networks where a baseline (anchor) input is compared to a positive (truthy) input and a negative (falsy) input. The distance from the baseline (anchor) input to the positive (truthy) input is minimized, and the distance from the baseline (anchor) input to the negative (falsy) input is maximized.

Войдите в приложение "Medium" с аккаунтом Google

Михаил Соколов
sokolovmisha44@gmail.com

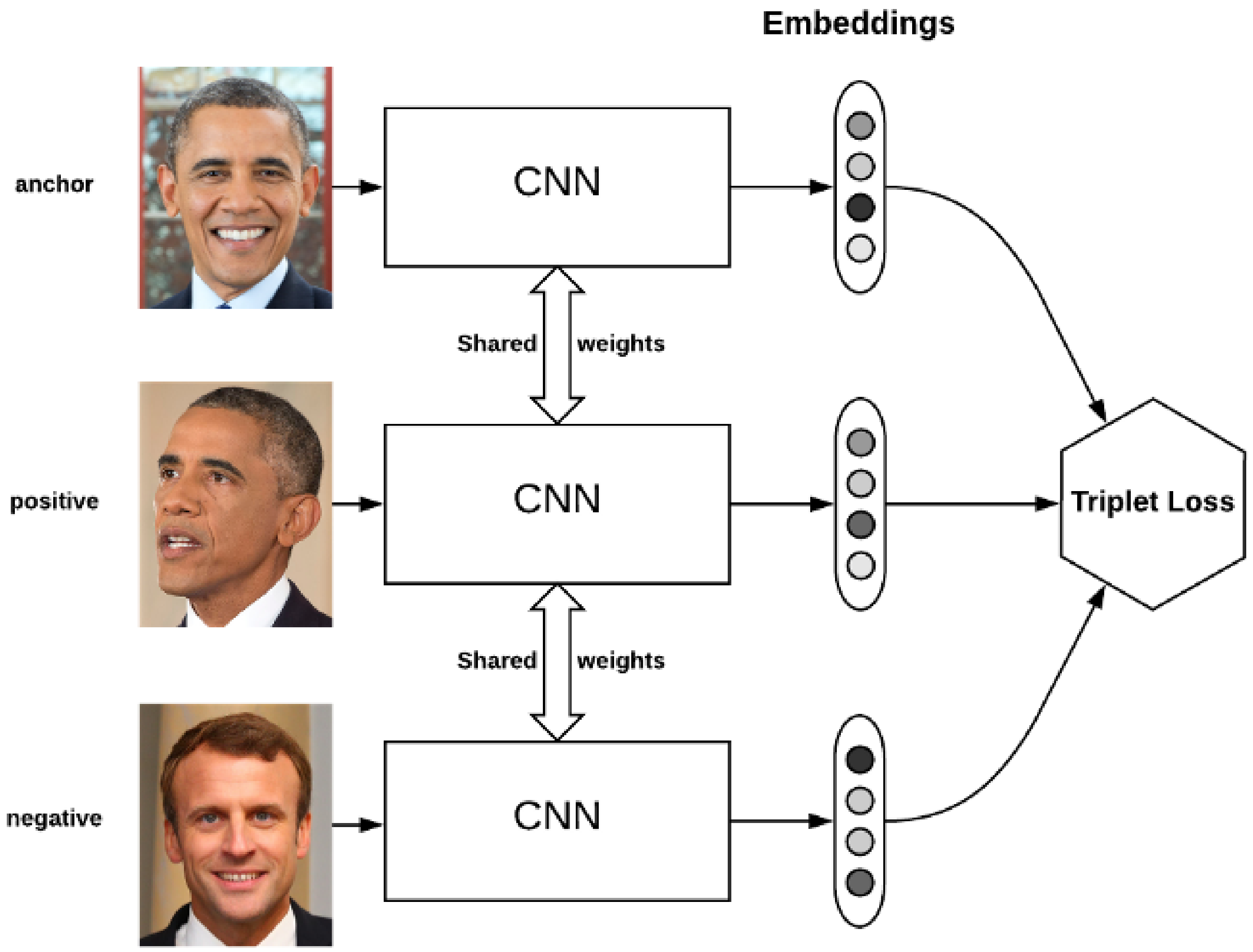
Продолжить как Михаил

Чтобы создать аккаунт, Google предоставит приложению "Medium" ваше имя, адрес электронной почты и фото профиля. См. Политику конфиденциальности и Условия использования сервиса "Medium".

In triplet loss basically, we have a triplet as inpu
any image of the person, positive is some other i
image of different person. The loss function can

$$\mathcal{L} = \max(\delta(a, p)$$

where $\delta(a,p)$ is the distance between anchor im
distance between anchor and negative image.



In the training phase we try to minimise such a loss.

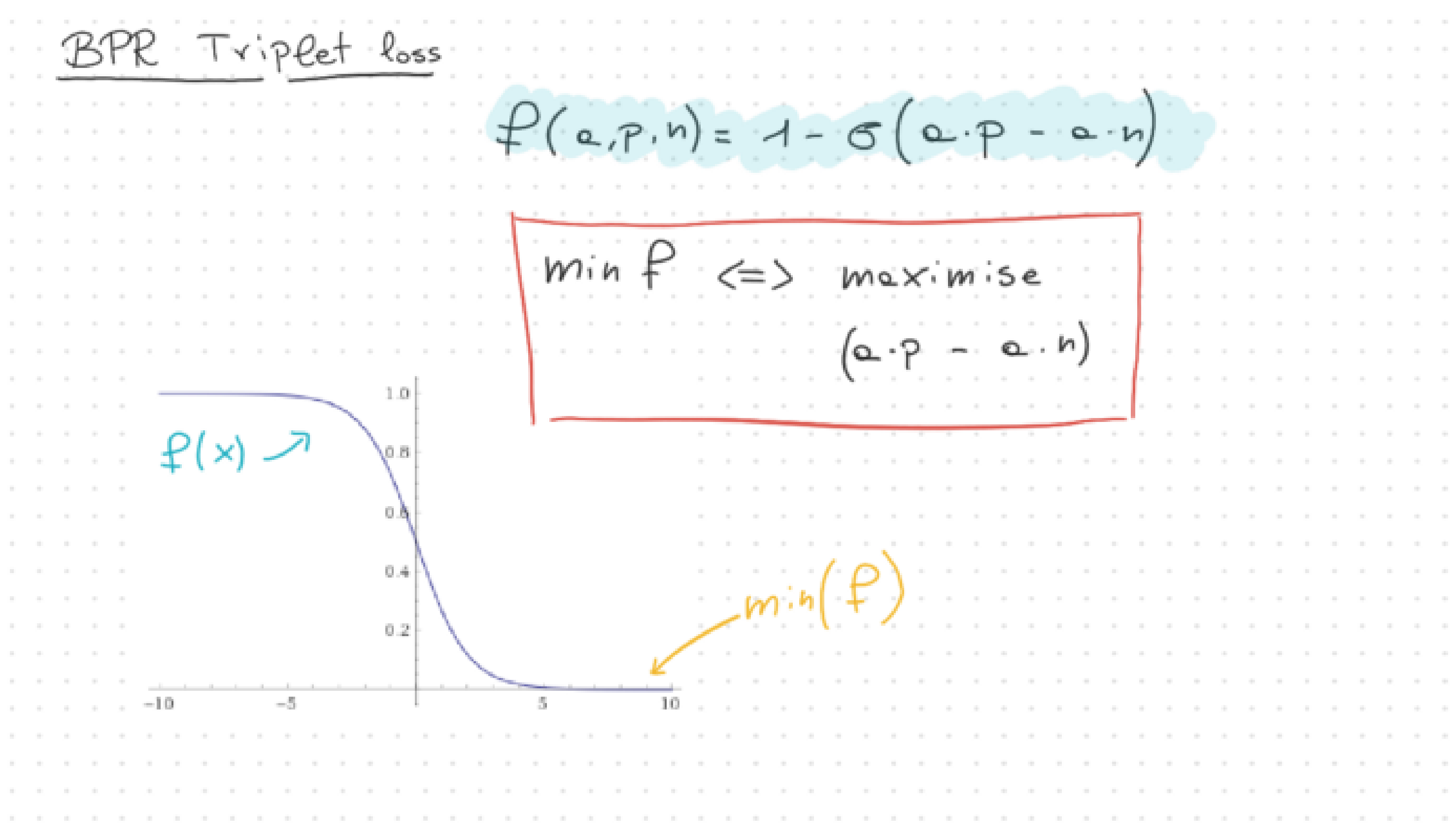
A recommender system can be seen in the same way. We have an *anchor user,* and two recommended items, *positive* and *negative*, meaning the recommendation should associate the user to the positive item and penalise the negative one.

recognition task is the fact that user and items a
optimise, as the triplet loss one, such that in the
recommending the positive samples and push d
adjusting model parameters.

Bayesian Personalized Ranking criterion comes

$$\mathcal{L} = 1 - \sigma(u_{\text{emb}} \cdot p$$

where $\sigma$ denotes the *sigmoid function* while user, positive item and negative item are represented by their embedding vectors.


BPR Triplet loss
$f(a,p,n) = 1 - \sigma(a \cdot p - a \cdot n)$
min f <=> maximise (a·p − a·n)
f(x) ↗
min(f)
1.0
0.8
0.6
0.4
0.2
−10
−5
5
10

A poorly drawn slide, showing how minimising BPR Triplet loss function is equivalent to maximising the distance between (Anchor, Positive item) and (Anchor, Negative item).

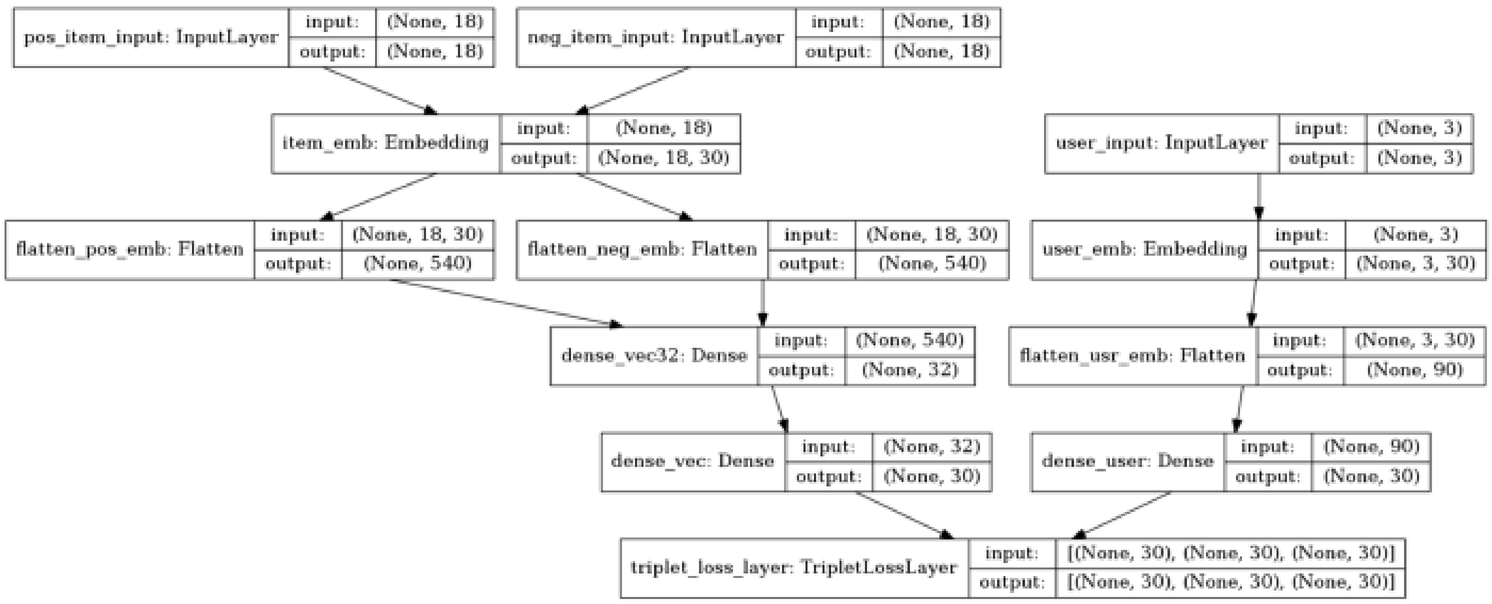
It is noteworthy that we are now able — in a quite simple way — to leverage user and item features but also to take into account previously expressed feedbacks. This is precisely an hybrid recommender system. I think this may express clearly the power of the embedding tool.

## The model

Let's put all of this in a neural network model. We are going to use Keras functional API

Hence we can define the layers adapted to receive user and movie vectors. To do so we need to define an Embedding layer for users and one for items. After that we are ready to define the model. This is done by a function

This gives the model represented in the figure below


pos_item_input: InputLayer | input: (None, 18) | output: (None, 18)
neg_item_input: InputLayer | input: (None, 18) | output: (None, 18)
item_emb: Embedding | input: (None, 18) | output: (None, 18, 30)
user_input: InputLayer | input: (None, 3) | output: (None, 3)
flatten_pos_emb: Flatten | input: (None, 18, 30) | output: (None, 540)
flatten_neg_emb: Flatten | input: (None, 18, 30) | output: (None, 540)
user_emb: Embedding | input: (None, 3) | output: (None, 3, 30)
dense_vec32: Dense | input: (None, 540) | output: (None, 32)
flatten_usr_emb: Flatten | input: (None, 3, 30) | output: (None, 90)
dense_vec: Dense | input: (None, 32) | output: (None, 30)
dense_user: Dense | input: (None, 90) | output: (None, 30)
triplet_loss_layer: TripletLossLayer | input: [(None, 30), (None, 30), (None, 30)] | output: [(None, 30), (None, 30), (None, 30)]

Note how item input layers share the same weights.

## Some comments

items.

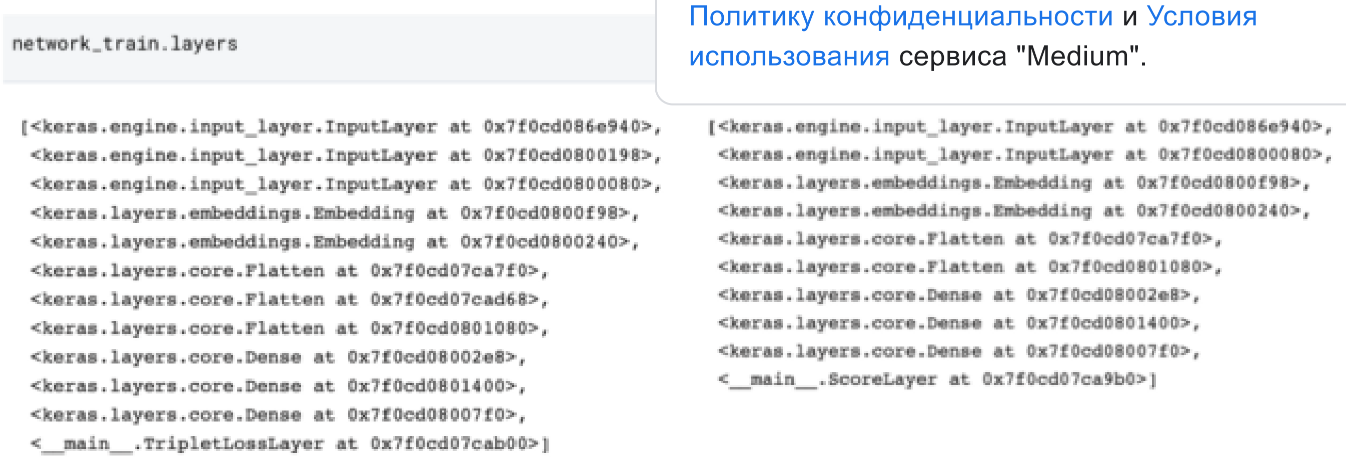
A further noteworthy aspect of the function abo
train, the other to predict. This because, as men
already implemented in Keras. To be sure the tw
for the last layer, we can print the ` layers` attrib

Войдите в приложение "Medium" с аккаунтом Google

Михаил Соколов
sokolovmisha44@gmail.com

Продолжить как Михаил

Чтобы создать аккаунт, Google предоставит приложению "Medium" ваше имя, адрес электронной почты и фото профиля. См. Политику конфиденциальности и Условия использования сервиса "Medium".



```
network_train.layers
```

```
[<keras.engine.input_layer.InputLayer at 0x7f0cd086e940>,
 <keras.engine.input_layer.InputLayer at 0x7f0cd0800198>,
 <keras.engine.input_layer.InputLayer at 0x7f0cd0800080>,
 <keras.layers.embeddings.Embedding at 0x7f0cd0800f98>,
 <keras.layers.embeddings.Embedding at 0x7f0cd0800240>,
 <keras.layers.core.Flatten at 0x7f0cd07ca7f0>,
 <keras.layers.core.Flatten at 0x7f0cd07cad68>,
 <keras.layers.core.Flatten at 0x7f0cd0801080>,
 <keras.layers.core.Dense at 0x7f0cd08002e8>,
 <keras.layers.core.Dense at 0x7f0cd0801400>,
 <keras.layers.core.Dense at 0x7f0cd08007f0>,
 <__main__.TripletLossLayer at 0x7f0cd07cab00>]
```

```
[<keras.engine.input_layer.InputLayer at 0x7f0cd086e940>,
 <keras.engine.input_layer.InputLayer at 0x7f0cd0800080>,
 <keras.layers.embeddings.Embedding at 0x7f0cd0800f98>,
 <keras.layers.embeddings.Embedding at 0x7f0cd0800240>,
 <keras.layers.core.Flatten at 0x7f0cd07ca7f0>,
 <keras.layers.core.Flatten at 0x7f0cd0801080>,
 <keras.layers.core.Dense at 0x7f0cd08002e8>,
 <keras.layers.core.Dense at 0x7f0cd0801400>,
 <keras.layers.core.Dense at 0x7f0cd08007f0>,
 <__main__.ScoreLayer at 0x7f0cd07ca9b0>]
```

As one may notice, the shared layers have actually the same memory address.

This is important because we need to train our network for *one-shot learning* meaning, it has to correctly predict preferences even for a never-seen user or an unknown item.

## Training the model

We are now ready to pass to the train part of our recommender system.

### Construction of training triplets

Before going and train this model we need to build the batches. Recall our training set is made by triplets (*user, positive movie, negative movie*).

As one can see from the picture above, the model expects a list of three arrays as input. Corresponding to the feature vectors of users and items.

Hence, the batch composer function will take information from the users dataset, from movies dataset and from the ratings matrix to compose triplets.

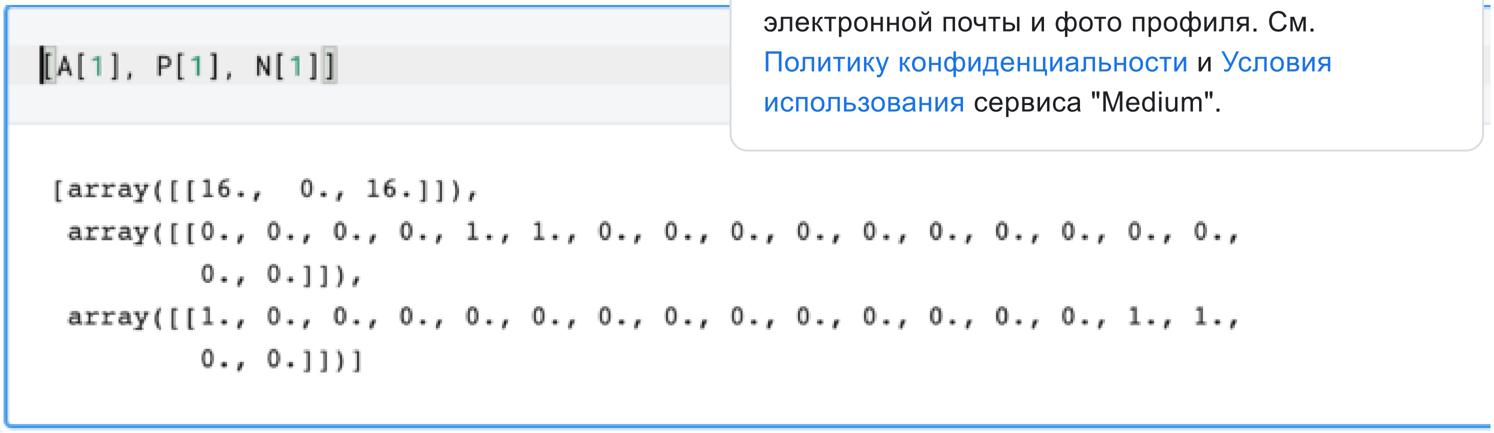
Войдите в приложение "Medium" с аккаунтом Google

Михаил Соколов
sokolovmisha44@gmail.com

Продолжить как Михаил

Чтобы создать аккаунт, Google предоставит приложению "Medium" ваше имя, адрес электронной почты и фото профиля. См. Политику конфиденциальности и Условия использования сервиса "Medium".

where *A* is an array of shape *(batch_size, n_user*
*(batch_size, n_item_features)*. Each row of the a
triplet referring to the first user (coupled to a m
ranked) is



```
[A[1], P[1], N[1]]
```

```
[array([[16.,  0., 16.]]),
 array([[0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0.]]),
 array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1.,
         0., 0.]])]
```

In order to get to this result, we defined a function.

**Translating ratings matrix into triplets**

The function randomly select a user and among its positive/negative ratings randomly picks two to compose the triplet.

**Training**

Having obtained the training set *i.e.* a bunch of triplets (user feature vec, liked movie feature vec, not liked movie feature vec), we can train our model on it.

Once triplets have been composed we can train our model. We have chosen a batch size of 32 and a 10000 iterations.

Войдите в приложение "Medium" с аккаунтом Google

Михаил Соколов
sokolovmisha44@gmail.com

Продолжить как Михаил

Чтобы создать аккаунт, Google предоставит приложению "Medium" ваше имя, адрес электронной почты и фото профиля. См. Политику конфиденциальности и Условия использования сервиса "Medium".

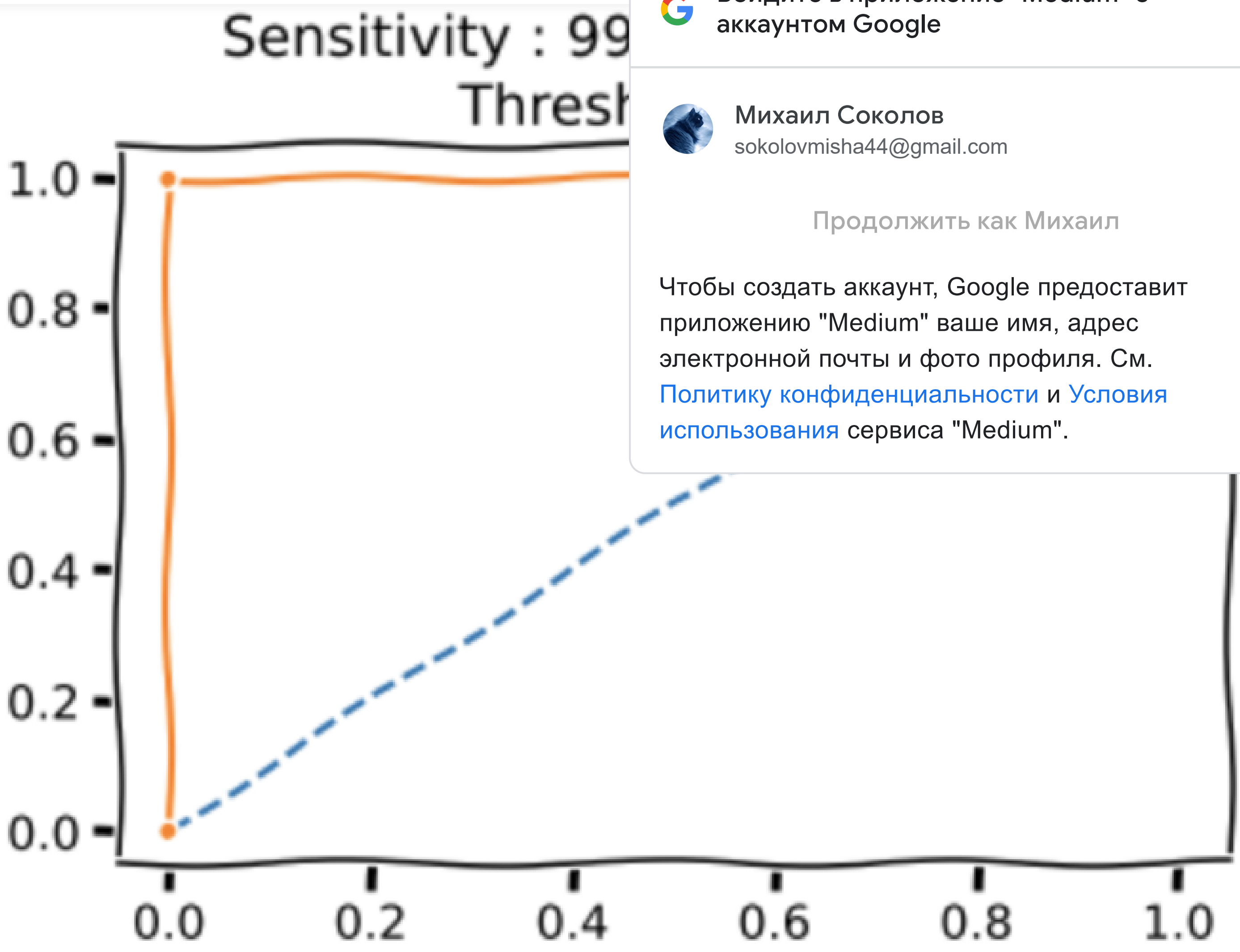
As one might expect, the triplet loss is close to 1
during the training to a value approximately equ

## Evaluation and metrics

Now the question everyone working in the field

In our situation, we created a model producing embeddings (*i.e.* vectors) that we can use to compute distances. In other words, if an item *i* is likely to be appreciated by an user *u*, their distance will be small, otherwise they will be separated by a great distance in the embedding space.

Thus, we need a **threshold:** if the found distance is under the threshold then we recommend the item to the user, if the distance is above the threshold then that item is not recommended. We have to choose this threshold carefully. This is a typical ROC curve problem. For this reason, one metric for evaluation could be Area Under the Curve (AUC).



The AUC after the training of the model.

## Predictions: Making actual recommendations

The aim of a recommender system is precisely to recommend (likely appreciated) items to users. Thus, given a user and a list of items, we can sort the list by “preference score”.

## Conclusion

This was just a very simple application. The model was not optimised at all, but I think one can build upon this to achieve interesting results. Especially a more careful hyperparameter tuning and a better choice of the Neural Network architecture would improve performances a lot. It could be interesting to see whether with a model to get tuned hyper-parameters we can obtain much better performance.

Another improvement one can do is to work on the input layers. Data cleaning and maybe enriching considering (for example) geographical data would be beneficial.

Still, I think this can be a good example of creating a not-so-trivial neural network model to be

The full code on which this post is based can be

## Acknowledgements

I would like to thank my dear friend and excepti
@alessandro.angioi, for code revision and fruitf

Войдите в приложение "Medium" с аккаунтом Google

Михаил Соколов
sokolovmisha44@gmail.com

Продолжить как Михаил

Чтобы создать аккаунт, Google предоставит приложению "Medium" ваше имя, адрес электронной почты и фото профиля. См. Политику конфиденциальности и Условия использования сервиса "Medium".