Olympiads in Informatics, 2020, Vol. 14, 61–76

DOI: 10.15388/ioi.2020.05

Recommending Tasks in Online Judges using Autoencoder Neural Networks

Paolo FANTOZZI[1,2], Luigi LAURA[1,3]
[1]*Italian Association for Informatics and Automatic Calculus (AICA), Italy*
[2]*Sapienza University of Rome, Italy*
[3]*International Telematic University Uninettuno, Italy*
e-mail: fantozzi@diag.uniroma1.it, luigi.laura@uninettunouniversity.net

Abstract. Programming contests such as International Olympiads in Informatics (IOI) and ACM International Collegiate Programming Contest (ICPC) are becoming increasingly popular in recent years. To train for these contests, there are several Online Judges available, in which users can test their skills against a usually large set of programming tasks.

In the literature, so far few papers have addressed the problem of recommending tasks in online judges. Most notably, as opposed with traditional Recommender Systems, since the learners improve their skills as they solve more problems, there is an intrinsic dynamic dimension that has to be considered: when recommending movies or books, it is likely that the preferences of the users are more or less stable, whilst in recommending tasks this does not hold true.

In order to help the learners, it is crucial to recommend them tasks that are challenging but not unsolvable compared with their current set of skills. In this paper we present a Recommender System (RS) for Online Judges based on an Autoencoder (Artificial) Neural Network (ANN).

We also discuss the results of an experimental evaluation of our approach in both the scenarios in which we consider, or not, the intrinsic dynamic dimension of the problem. The ANNs are trained with the dataset of all the submissions in the Italian National Online Judge, used to train students for the Italian Olympiads in Informatics.

Keywords: autoencoder neural networks, recommender systems, programming contests.

1. Introduction

Programming Contests (PCs) are competitions in which participants are faced a set of tasks that require writing computer programs. Recent literature have emphasized the importance and the effectiveness of PCs in the process of learning computer programming (Audrito *et al.*, 2012; Astrachan, 2004; Blumenstein *et al.*, 2008; Dagienė, 2010; Garcia-Mateos and Fernandez-Aleman, 2009; Wang *et al.*, 2011).

In order to train for PCs, learners use Online Judges (OJs), also known as Programming Online Judges, i.e., web based e-learning tools where a user can submit solutions

to a programming task. The user chooses a task from the many available; after reading its statement, that includes the required formatting of input and output or the use of a programming interface, the user writes a code to solve the task. The code is submitted to the OJ, that verifies both the correctness, usually by testing it against a certain number of test cases, and the efficiency, by checking that the running time and/or the memory usage is under some limit. In Fig. 2 is shown an example of a programming task.

However, choosing the right task is becoming a complex problem, and an example of an *information overloading scenario*, as observed in Yera Toledo *et al.* (2018): an unexperienced user has to choose from thousands programming tasks, many of which are probably beyond his current abilities. For example, University of Valladolid Online Judge has more than 200k users and 2k tasks, whilst SPOJ accounts approximately 600k users and 6k (public) tasks; in Fig. 1 we can see the list of available problems in the Peking University Online Judge `http://poj.org`.

With so many available tasks, it is important to help users selecting their next task by using a Recommender System (RS). Traditionally, RS are broadly divided into two categories: Content Based ones, in which the recommendations derive from features of the items to be suggested, and Collaborative Filtering approaches, in which the suggestion is based on the items chosen by users *similar to the current one*.

As observed in (Audrito *et al.*, 2019), there are some peculiarities of Online Judges that prevent the use of a general Recommender System:

- Users slowly improve their abilities, one task after the other, so the general concept of user *preferences* does not apply: recommending a movie or a novel differs significantly from recommending a task; a user will probably still like a novel after one year, whilst he might find a task too easy after the same amount of time.
- Users with *similar* skills, i.e. users to whom we might want to suggest the same set of tasks, might behave very differently in OJs, thus preventing us from considering them *similar*. For example, one might solve all the tasks involving a given

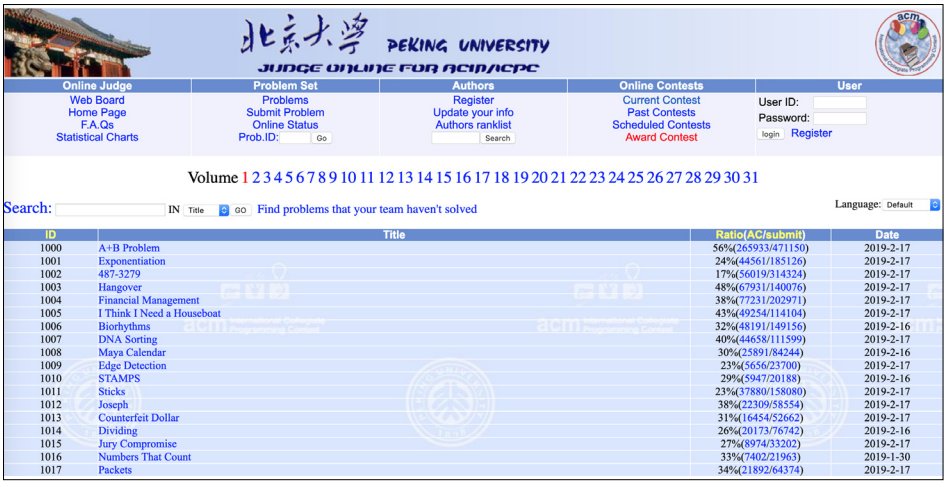
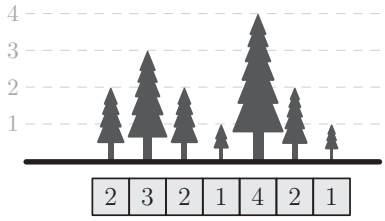


Fig. 1. The list of available problems in the Peking University OJ.

Olimpiadi Italiane di Informatica 2014

Fisciano, 18 – 20 settembre 2014 `taglialegna • IT`

Abbatti S.p.A. (which is the Italian brand of *tearDown INC*) is a big enterprise that works in the field of tree felling. In particular, it's been a few years since it started improving in tearing down *barky trees*, a peculiar kind of trees which is very tall and thick. This particular species grow in a very tidy way: the woods made of these trees are actually a long horizontal line of trunks, placed at one decameter (32, 8 feet) one another. Each one of the trees has a particular height, which is expressed by a positive number (decameters).



Tearing down one of these trees is a very hard thing to do and, although tearDown INC employes the most andnced technologies on the market, it is a very time consuming activity, since barky trees' bark is incredibly thick. The workers have the opportunity to chose in which direction (left or right) the tree should fall after the cut.

Each time a barky tree falls down it hits the trees that haven't been torn down which are placed on its falling trajectory; in other words, it tears down each tree which is closer than its height in the direction of the fall. Since the number of barky trees in this woods is huge this dynamic of the fall creates a domino effect.

In order to be the best enterprise in barky tree felling tearDown INC developed a system which is able to scan the whole wood, choosing which trees should be cut by workers and in which directions they should fall with the aim of cutting all the trees. It's important to recall that it's in the best interest of the enterprise to minimize the number of trees that need to be torn down by workers directly. Your role in this situation is to implement the system for tearDown INC.

Below we can see an example of a solution of the instance shown in the picture above: it is enough to cut two trees:

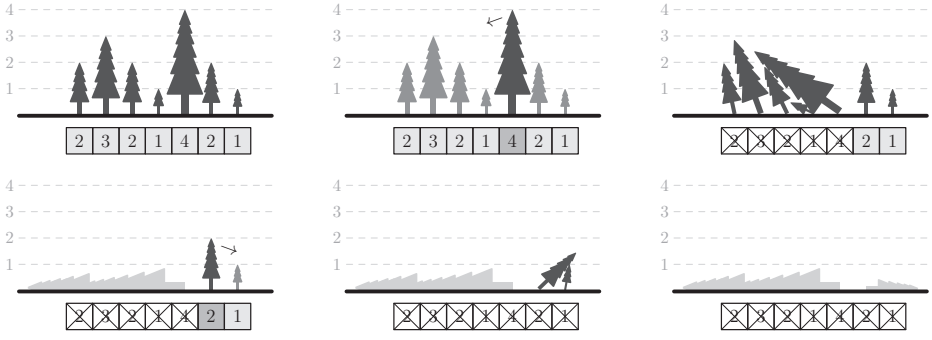


Fig. 2. An example of a problem from a programming contest; this task is taken from the final contest of the 2014 edition of the Italian Olympiads in Informatics (OII).

skill, while the other might just solve one task, related to that skill, and then move on to tasks involving different skills.

Most notably, as opposed with traditional Recommender Systems, since the learners improve their skills as they solve more problems, there is an intrinsic dynamic dimension that has to be considered: when recommending movies or books, it is likely that the preferences of the users are more or less stable, whilst in recommending tasks this does not hold true. In this paper we propose a task recommender system based on an Autoencoder Neural Network (ANN); in particular, we address both the *static* case, in which the user is represented by the task he solved, and the *dynamic* case, where we try to represent the growth of a user by the sequence of the problems he solved. For both cases we present a Recommender System (RS) for Online Judges based on an Autoencoder (Artificial) Neural Network (ANN). We trained and tested the ANN using data from the Online Judge used in the Italian Olympiads in Informatics (Olimpiadi Italiane di Informatica – OII) (Di Luigi *et al.*, 2016), targeted at secondary school students training. We compared our approaches against state of the art more classical recommender systems built using the Simple Python Recommendation System Engine (SurPRISE – `http://surpriselib.com`). The experimental results confirm the effectiveness of our approach.

Preliminary versions of this paper appeared in the Proceedings of the 17th International Conference on Distributed Computing and Artificial Intelligence (DCAI 2020) (Fantozzi and Laura, 2020a) (the static case), and in the Proceedings 13th International Workshop on Social and Personal Computing for Web-Supported Learning Communities (SPeL 2020) (Fantozzi and Laura, 2020b) (the dynamic case); the comparison against state of the art more classical recommender systems has not appeared before.

This paper is organized as follows: the next section provides the necessary background related to programming contests, online judges, and recommender systems, whilst our approach is detailed in Section 3. In Section 4 we detail our experimental findings and concluding remarks are addressed in Section 5.

2. Related Works and Background

In this section we discuss related work and the necessary background concerning programming contests, online judges, and recommender systems.

2.1. *Programming Contests and Online Judges*

A programming contest is a competition in which contestants are faced with a set of programming tasks, also called problems, to be solved in a limited amount of time and/or with a limited amount of memory usage.

A single task can be broken into different subtasks of increasing complexity: basic techniques might be enough to solve, within the given time and/or space limits, some of

the subtasks whilst the most difficult ones might require very specific algorithmic techniques and data structures. Popular programming contests are:

- The International Olympiads in Informatics (IOI), that are an annual programming competition for secondary school students patronized by UNESCO. `http://www.ioinformatics.org/`
- The ACM International Collegiate Programming Contest (ICPC) is a multitier, team-based, programming competition operating under the auspices of ACM. `https://icpc.baylor.edu/`
- The very recent International Olympiads in Informatics in Team (IOIT), that started in 2017, that are a team competition, like ACM ICPC, differently from IOI (individual competition). Currently there are only four nations involved: Italy, Romania, Russia, and Sweden. `https://ioi.team/`
- Google Code Jam, that is based on multiple online rounds that concludes in the World Finals. `https://code.google.com/codejam/`
- Facebook Hacker Cup, that is (citing from their site) "*an annual worldwide programming competition where hackers compete against each other for fame, fortune, glory and a shot at the coveted Hacker Cup*". `https://www.facebook.com/hackercup/`

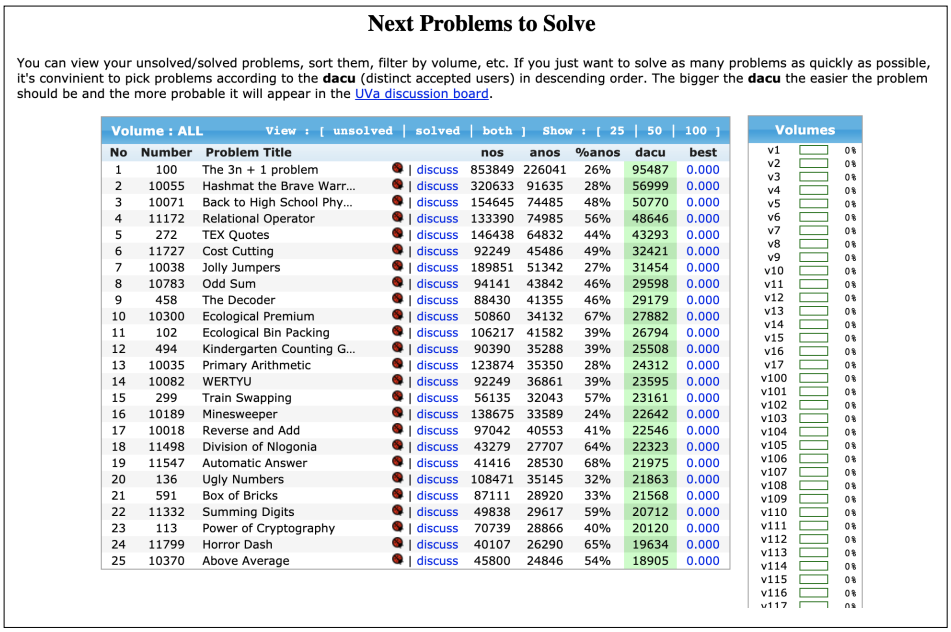
The Online Judges are, usually, web based platforms that provide a large number of programming tasks to be solved. There are several popular OJ platform, we cite the already mentioned University of Valladolid Online Judge `https://uva.onlinejudge.org`, Sphere Online Judge (SPOJ) `https://www.spoj.com/`, CodeChef `https://www.codechef.com/`, and Peking University Online Judge `http://poj.org`.

Yera and Toledo (Yera Toledo *et al.*, 2018) present a brief survey on OJs, whilst more information on tools and techniques for automatic evaluation of solutions submitted to OJs can be found in (Ala-Mutka, 2005; Caiza and Del Alamo, 2013).

2.2. *Recommender Systems in OJs*

As already observed in the introduction, despite the large amount of literature devoted to RS, the peculiarities of recommendation in OJs, where the relation user-item is way more complex than the typical RS cases, prevent from using standard techniques and forces the development of ad-hoc methods. This aspect is detailed in the paper of Audrito *et al.* (2012), where the authors propose a first approach on building a RS by tackling the problem of ranking tasks in Online Judges.

Indeed, so far few research focused in the recommendation of tasks in OJs: we mention the traditional collaborative filtering method with a new similarity measure adapted to the case (Toledo and Mota, 2014), and an approach based on fuzzy logic (Yera Toledo *et al.*, 2018). Caro and Jimenez considered user-based and similarity-based approaches In (Caro-Martinez and Jimenez-Diaz, 2017). Di Mascio *et al.* proposed a framework that can allow recommendations and that can foster motivation in students by means of a lightweight, badge-based, gamified approach (Di Mascio *et al.*, 2018).



Next Problems to Solve

You can view your unsolved/solved problems, sort them, filter by volume, etc. If you just want to solve as many problems as quickly as possible, it's convinient to pick problems according to the **dacu** (distinct accepted users) in descending order. The bigger the **dacu** the easier the problem should be and the more probable it will appear in the UVa discussion board.

Volume : ALL | View : [unsolved | solved | both] | Show : [25 | 50 | 100]

No	Number	Problem Title		nos	anos	%anos	dacu	best
1	100	The 3n + 1 problem	discuss	853849	226041	26%	95487	0.000
2	10055	Hashmat the Brave Warr...	discuss	320633	91635	28%	56999	0.000
3	10071	Back to High School Phy...	discuss	154645	74485	48%	50770	0.000
4	11172	Relational Operator	discuss	133390	74985	56%	48646	0.000
5	272	TEX Quotes	discuss	146438	64832	44%	43293	0.000
6	11727	Cost Cutting	discuss	92249	45486	49%	32421	0.000
7	10038	Jolly Jumpers	discuss	189851	51342	27%	31454	0.000
8	10783	Odd Sum	discuss	94141	43842	46%	29598	0.000
9	458	The Decoder	discuss	88430	41355	46%	29179	0.000
10	10300	Ecological Premium	discuss	50860	34132	67%	27882	0.000
11	102	Ecological Bin Packing	discuss	106217	41582	39%	26794	0.000
12	494	Kindergarten Counting G...	discuss	90390	35288	39%	25508	0.000
13	10035	Primary Arithmetic	discuss	123874	35350	28%	24312	0.000
14	10082	WERTYU	discuss	92249	36861	39%	23595	0.000
15	299	Train Swapping	discuss	56135	32043	57%	23161	0.000
16	10189	Minesweeper	discuss	138675	33589	24%	22642	0.000
17	10018	Reverse and Add	discuss	97042	40553	41%	22546	0.000
18	11498	Division of Nlogonia	discuss	43279	27707	64%	22323	0.000
19	11547	Automatic Answer	discuss	41416	28530	68%	21975	0.000
20	136	Ugly Numbers	discuss	108471	35145	32%	21863	0.000
21	591	Box of Bricks	discuss	87111	28920	33%	21568	0.000
22	11332	Summing Digits	discuss	49838	29617	59%	20712	0.000
23	113	Power of Cryptography	discuss	70739	28866	40%	20120	0.000
24	11799	Horror Dash	discuss	40107	26290	65%	19634	0.000
25	10370	Above Average	discuss	45800	24846	54%	18905	0.000

Volumes

v1 0% v2 0% v3 0% v4 0% v5 0% v6 0% v7 0% v8 0% v9 0% v10 0% v11 0% v12 0% v13 0% v14 0% v15 0% v16 0% v17 0% v100 0% v101 0% v102 0% v103 0% v104 0% v105 0% v106 0% v107 0% v108 0% v109 0% v110 0% v111 0% v112 0% v113 0% v114 0% v115 0% v116 0%

Fig. 3. The *Next Problem to Solve* section in the uHunt .

There is an online tool, developed by Stephen and Felix Halim, authors of the book Competitive Programming (Halim and Halim, 2013), called uHunt, that helps its users to choose the next problem to be solved, as shown in Fig. 3: their very practical (and effective approach) is to rank the problems according to their **dacu**, i.e. the distinct accepted users. Indeed, as they state, "*The bigger the dacu the easier the problem should be and the more probable it will appear in the UVa discussion board*".

The "classical approaches" that use counting to estimate the grade of difficulty have a well known drawback: the items suggested to the users will be always the popular ones that will become even more popular, and so even more recommended. This means that a new item will never be suggested.

2.3. *Recommender Systems and Artificial Neural Networks*

The use of deep learning techniques for recommender systems is divided in two categories that we call *classical* and *hybrid*.

The classical approach uses the standard architectures of neural networks, applying them to this task. So, in this case, the most important part consists in the formulation of the problem, since that, if the input data are not suitable to be the input of that specific deep learning technique, then the result will be totally inaccurate.

The other approach is the hybrid one, that consists in using more than one type of architecture at the same time. This kind of approach is useful when the input data are

not easily representable as a standard structure, like a user-item matrix. In (Zhang *et al.*, 2016) the authors use a Convolutional Neural Network to extract features from images and then an Autoencoder to build the recommender system on the features.

In (Zhang *et al.*, 2017a), in order to build a recommender system for hashtag in tweets, the authors use at the same time some CNNs and some Recurrent Neural Networks (RNNs). In this work they use the CNNs to extract features from the image and then the RNNs to extract information from the text, combining them using different weights based on co-attention.

Since that a recommendation task is similar to a dimensionality reduction task, many of the state-of-the-art techniques use some kind of Autoencoder to map the input in a smaller space, that will be the representation of the correlations in the recommender system. In particular, Sedhain *et al.* (2015) introduce the using of a vanilla Autoencoder to build a recommender system. They use a partial masked input (the same techniques we use in this work) and try to reconstruct it in output, the elements added in the output will be the recommended elements. Strub and Mary (2015) extend the work of Sedain *et al.* (2015): they use a denoising Autoencoder instead of the vanilla Autoencoder to build a more robust system.

Chen and de Rijke (Chen and de Rijke, 2018) follow a similar approach, but they use a Variational Autoencoder to perform top-N recommendation. They encode both the user ratings and some side information in the compact space in the Autoencoder. Zhang *et al.* (Zhang *et al.*, 2017b) generalize the Contractive Autoencoder paradigm into matrix factorization framework. Li *et al.* (Li *et al.*, 2015) combine a probabilistic matrix factorization with Marginalized Denoising Stacked Autoencoders to perform collaborative filtering. This work can be considered as a general framework to use these kinds of techniques; in this context, several works, including (Van den Oord *et al.*, 2013; Wang *et al.*, 2015; Wang and Wang, 2014), can be viewed as special cases of this framework.

3. Recommending Tasks Using Autoencoder Neural Network

Our goal is to provide recommendations to the users regarding the next task to deal with among all the tasks in a system. To build this kind of system we assume that:

- If a user obtains a score for a task it means that it is the max score possible for that user in that task.
- A user should solve the problems sorted by their grade of difficulty for the user; thus, a user should never try to solve a problem that is much harder than the last one he solved.
- It is possible to deduce the score for a new problem based on the scores the user obtained in other problems.

Note that there is one more assumption that is valid for the static case but not for the dynamic: given a snapshot of the scores for the users it is not important the order followed by the user to solve the tasks to foresee the score for another task.

Based on the above assumptions, we decided to build a model that takes as input the current scores of a user and provides probable scores for the same user for other tasks. Then we can choose between the forecasted scores and suggest the task with the highest score between them.

If we consider the score as a judgment of the user for the item (i.e, the programming task), then we can just exploit the already known techniques for recommending items. We chose to use an Autoencoder to build the model. Since that we want to use just the scores of the users, without any information from other sources, we take as input the scores for all the tasks and we differentiate between static and dynamic:

- Static: we mask a fraction of the scores in input as a non-solved task and we perform back- propagation from the complete scores.
- Dynamic: we use the scores of a fixed moment in time as input and we perform backpropagation from the next moment.

In this way, in the bottleneck layer, there should be a compact representation of the similarities of the tasks.

4. Experimental Evaluation

In this section we describe the results of our experimental evaluation. We distinguish the two approaches, i.e. static and dynamic, in the next sections, and then compare the results against the ones obtained using a state of the art more classical recommender system built using the Simple Python RecommendatIon System Engine (SurPRISE – `http://surpriselib.com`).

4.1. *Autoencoder Neural Networks: Static Case*

To test the method we have designed, we have taken the submission to the OII Training platform (Di Luigi *et al.*, 2016) in a defined time range. The submissions were in the form:

```
< user_id, task_id, datetime, score >
```

where each submission corresponds to a possible solution to a task from a user that performs a certain score, based on many test cases. We filtered out all the scores equal to zero because we can't know if they were just users testing the behaviour of the platform. Then we considered only the best score for each task, for each user, to ignore all the attempts to solve the problem before the user found the solution.

We performed a preliminary set of experiments with the original data: we built a user x task matrix where each cell contains the best score of the user for the task. The result is a 3148 x 409 matrix with 43051 non-empty cells. The matrix has an average of 105 submissions for each task and 13 for each user. The max number of users which have

submitted to the same task is 1070 and the max number of tasks with submissions from the same user is 336. We consider the zero valued cells as a problem with no submission from the user.

To use this matrix as a training set for this model, we duplicated the matrix and then we have randomly masked some positive scores with zero. The masked matrix will be the input to the model and the original matrix will be the output to reconstruct. The number of tasks masked for each user is a random number between 3 and 7, with the constraint that it should be anyway equal at most to the half of the submission for the user.

After the preliminary experiments, it was clear that data was too small, thus we performed an operation of data augmentation. in particular, we have repeated many times the same rows of the matrix with the result of a matrix with 8 times the rows of the original. Then each row has been randomly masked independently, so we unlikely had duplicated rows in the matrix. We have load all the data on a Google Colab instance with an available GPU. Then we have splitted the data on train and test set with a ratio of 0.8/0.2. We used Tensorow to build several models; the smallest was a Sequential model with 5 layers: the input layer, a dense 64 neurons layer, a dense 16 neurons layer, a dense 64 neurons layer, and an output layer with dimension equal to the input layer. All the activations for the layers are ReLU with a constraint of a max value of 1.0 (the max value of the score). We used an Adam optimizer with a learning rate of 0.001 and a mean squared error loss function.

We trained the model for 500 epochs with a batch size of 128, and we have imposed a validation split of 0.2. The resulting learning curve is the shown in Fig. 4, whilst the accuracy curve measured is depicted in Fig. 5.

The standard accuracy might be not fully representative of the error of the model (since that we have a sparse matrix), thus we computed a sum of the squared errors on each samples in the test set. The resulting values follow the distribution shown in Fig. 6; in Table 1 we report some stats of the SSE distribution.

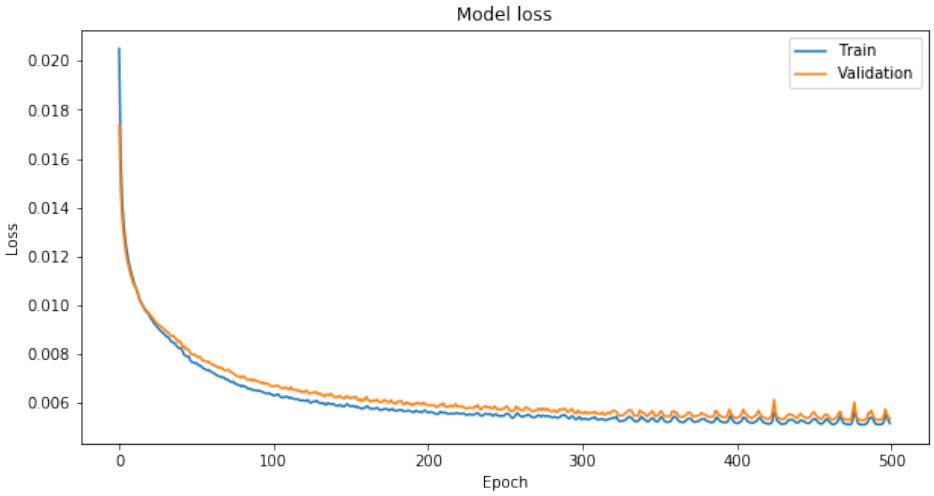


Fig. 4. Model loss – static case.

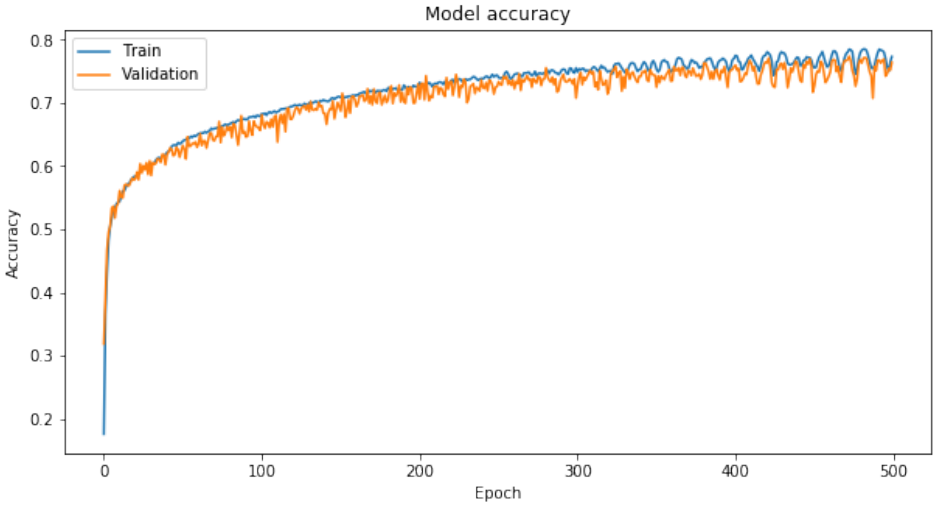


Fig. 5. Model accuracy – static case.

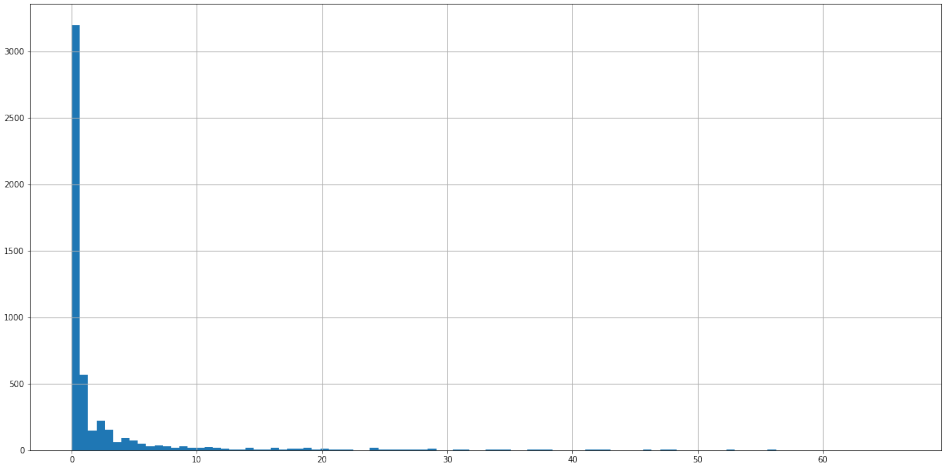


Fig. 6. Distribution of sum of squared errors (SSE) – static case.

Table 1

Statistics of the distribution of SSE (Fig. 9) – static case

mean	2.27
std deviation	6.07
min	0.00
25% (1st quartile)	0.00
50% (2nd quartile)	0.05
75% (3rd quartile)	1.38
max	66.19

4.2. *Autoencoder Neural Networks: Dynamic Case*

In the dynamic case, our interest was in the evolution of the users, so the dataset that we used was derived from the one, described in the previous section, for the static case. Indeed, from the baseline dataset, we built a user x task matrix where each cell contains the best score of the user for the task. The result is a 42155 x 409 matrix with 2035447 non-empty cells (we had to drop few submissions from the baseline dataset). As before, the dataset has an average of 105 submissions for each task and 13 for each user. The max number of users which have submitted to the same task is 1070 and the max number of tasks with submissions from the same user is 336. As before, we consider the zero cells as a problem with no submission from the user.

Since that the resulting matrix has many rows for each user (one row for each problem solved) that contains all the scores of the users until that moment, we use a row as input and we impose the next row for the same user as output. This means that we consider $n_i - 1$ samples for each user i, where ni is the number of problems solved by the user i.

As for the static case, we load all the data on a Google Colab instance with an available GPU and the data was split into train and test set with a ratio of 0.8/0.2. We used Tensorow to build several models; the smallest was a Sequential model with 11 layers: the input layer, two dense 128 neurons layer, two dense 64 neurons layers, a dense 32 neurons layer, two dense 64 neurons layer, two dense 128 neurons layers, and an output layer with dimension equal to the input layer. Also in this dynamic case, all the activations for the layers are ReLU with a constraint of a max value of 1.0, and we used an Adam optimizer with a learning rate of 0.001 and a mean squared error loss function. We trained this model for 100 epochs with a batch size of 128, and we have imposed a validation split of 0.2.

The resulting learning curve is the shown in Fig. 7, whilst the accuracy curve measured is depicted in Fig. 8.

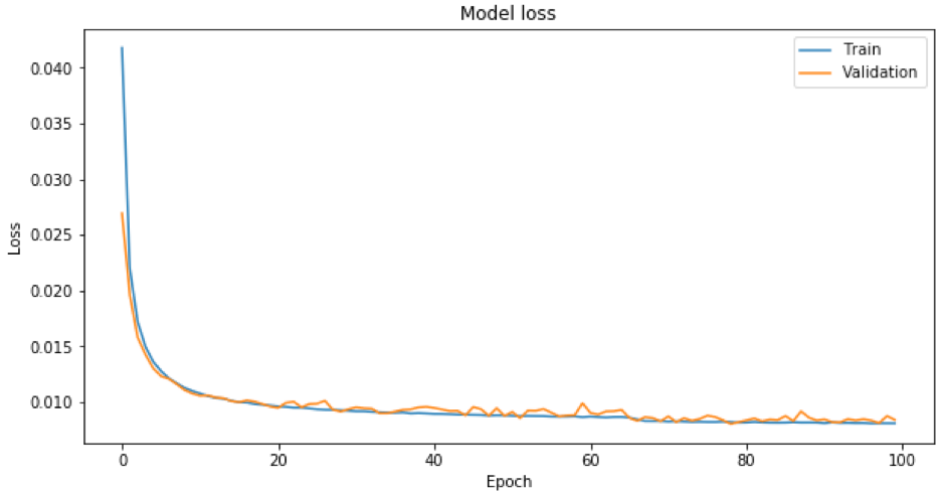


Fig. 7. Model loss – dynamic case.

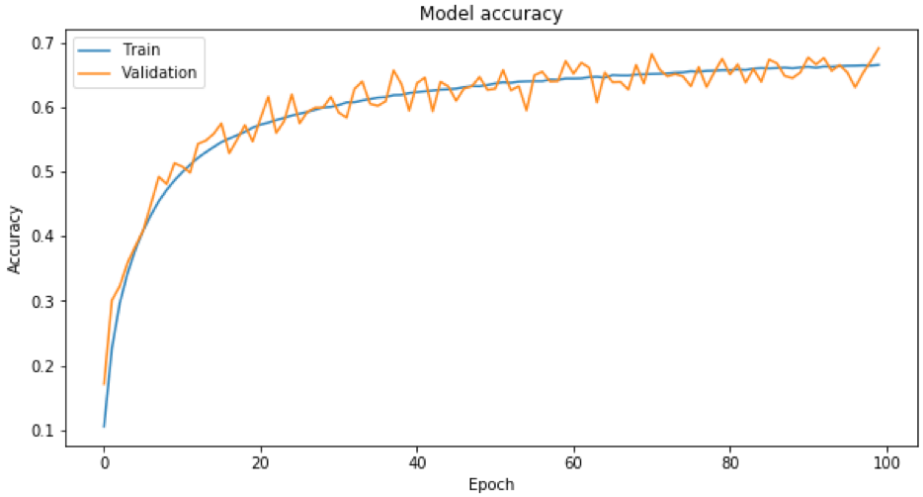


Fig. 8. Model accuracy – dynamic case.

Since that the standard accuracy doesn't represent well the error of the model (i.e., we have a sparse matrix) we computed a sum of the squared errors on each samples in the test set.

The resulting values follow the distribution shown in Fig. 9; in Table 2 we report some stats of the SSE distribution.

Overall, from the results shown above, it seems that the static approach, described in the previous section, seems to perform better than the dynamic one; this might be due to the way the dataset has been built, and we plan to compare the two approaches against other different datasets.

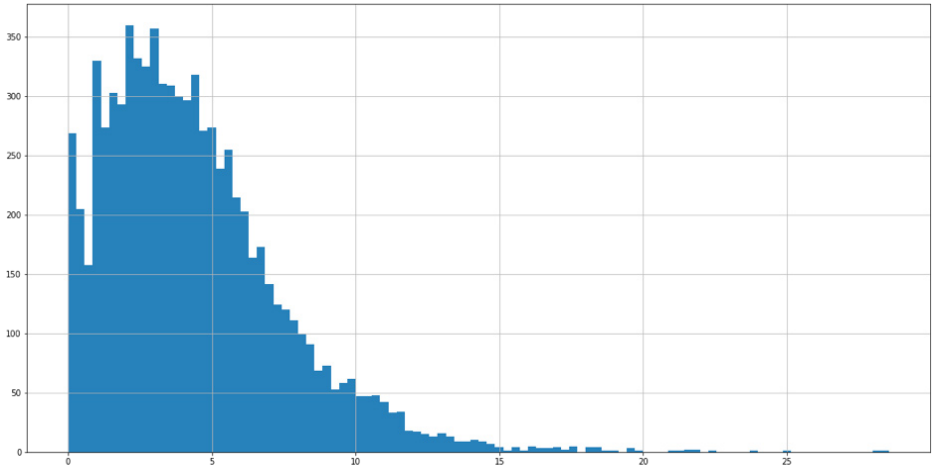


Fig. 9. Distribution of sum of squared errors (SSE) – dynamic case.

Table 2
Statistics of the distribution of SSE (Fig. 9) – dynamic case

mean	4.40
std deviation	3.10
min	0.00
25% (1st quartile)	2.12
50% (2nd quartile)	3.86
75% (3rd quartile)	5.98
max	28.56

4.3. *Comparison Against Classical Recommender Systems*

In this section we compare our approaches against state of the art Recommender Systems built using the python SurPRISE library: we used 11 models from this library. We trained these models using the static dataset, i.e. the original dataset; the dynamic dataset derives from this one and the dynamic ANNs, as seen in the previous section, performed not as well as the ones trained on the static dataset.

Indeed, we experimented with 32 different ANNs for the static case and, after evaluating the best performers, we experimented with 8 different ANNs for the dynamic case. In Table 3 we can see the mean square error (MSE) for all 11 SurPRISE models,

Table 3
The Mean Square Error of all the tested model, sorted from the biggest (worst) to the smallest (best). Here we compare all the models from the SurPRISE library against the three best performers of the several Autoencoder Neural Network we tested

Model	MSE
NormalPredictor	0.1004863793
CoClustering	0.0903564508
SlopeOne	0.0599415086
NMF	0.0545891728
KNNBasic	0.0544956720
KNNWithZScore	0.0528847039
KNNWithMeans	0.0526918100
SVD	0.0512103173
BaselineOnly	0.0484940726
SVDpp	0.0479013953
KNNBaseline	0.0478215959
(dynamic) autoencoder-plus-time-512-128-Dropout(0.1)-512-lr0.001	0.0031555248
(dynamic) autoencoder-plus-time-2048-512-Dropout(0.1)-2048-lr0.001	0.0025685430
(dynamic) autoencoder-plus-time-1024-256-Dropout(0.1)-1024-lr0.001	0.0025619145
(static) autoencoder-2048-512-Dropout(0.1)-2048-lr0.0005	0.0000349358
(static) autoencoder-1024-256-Dropout(0.1)-1024-lr0.0001	0.0000281454
(static) autoencoder-2048-512-Dropout(0.1)-2048-lr0.0001	0.0000191324

and for the three best ANNs for both the static and the dynamic case. The results shown in the table are sorted from the biggest (worst) to the smallest (best). It seems that, at least for this dataset, the ANNs outperform each model from the SurPRISE library. In the table, the three best results belong to ANNs trained for the static case, but in all our experiments, i.e. 40 different ANNs (32 for the static case and 8 for the dynamic case), we did not observe such a clear separation between the ANNs trained with the two different datasets.

5. Conclusions

In this paper we proposed the design of a recommender system for tasks suggestions in Online Judges, based on a Autoencoder Neural Network. We trained the ANN with the data from the OJ used by the secondary school students training for the Italian Olympiads in Informatics (Olimpiadi Italiane di Informatica – OII) (Di Luigi *et al.*, 2016; Di Luigi *et al.*, 2018).

We tested two different approaches: a *static* one, that is more typical of a recommender system, and a *dynamic* one, in which the dataset has been modified in order to explicitly represent the evolution of a user. We also compared our approaches against more traditional Recommender Systems model built using the python SurPRISE library.

We definitely think that Online Judges deserve their specific recommender systems, and we hope that our one is a first step to the development of such systems. We plan to implement our approach inside the italian OJ, and we are available to collaborate with the developers of other Online Judge systems, by either implementing RS into those systems or by testing our models against other datasets.

References

Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2), 83–102.

Astrachan, O. (2004). Non-competitive programming contest problems as the basis for just-in-time teaching. In: *Frontiers in Education, 2004. FIE 2004. 34th Annual*, pages T3H/20–T3H/24 Vol. 1.

Audrito, G., Demo, G. B., and Giovannetti, E. (2012). The role of contests in changing informatics education: A local view. *Olympiads in Informatics*, 6.

Audrito, G., Mascio, T. D., Fantozzi, P., Laura, L., Martini, G., Nanni, U., and Temperini, M. (2019). Recommending tasks in online judges. In: *Methodologies and Intelligent Systems for Technology Enhanced Learning, 9th International Conference, MIS4TEL 2019, Avila, Spain, 26–28 June, 2019*, volume 1007 of *Advances in Intelligent Systems and Computing*, pages 129–136. Springer.

Blumenstein, M., Green, S., Fogelman, S., Nguyen, A., and Muthukkumarasamy, V. (2008). Performance analysis of game: a generic automated marking environment. *Computers and Education*, 50, 1203–1216.

Caiza, J. and Del Alamo, J. (2013). Programming assignments automatic grading: Review of tools and implementations. In: *INTED2013 Proceedings*, *7th International Technology, Education and Development Conference*, pages 5691–5700. IATED.

Caro-Martinez, M. and Jimenez-Diaz, G. (2017). Similar Users or Similar Items? Comparing Similarity-Based Approaches for Recommender Systems in Online Judges. In: Aha, D. W. and Lieber, J., editors, *Case-Based Reasoning Research and Development*, volume 10339, pages 92–107. Springer International Publishing, Cham.

Chen, Y. and de Rijke, M. (2018). A collective variational autoencoder for top-n recommendation with side information. In: *Proceedings of the 3rd Workshop on Deep Learning for Recommender Systems*, pages 3–9.

Dagienė, V. (2010). Sustaining informatics education by contests. In: *International Conference on Informatics in Secondary Schools-Evolution and Perspectives*, pages 1–12. Springer.

Di Luigi, W., Fantozzi, P., Laura, L., Martini, G., Morassutto, E., Ostuni, D., Piccardo, G., and Versari, L. (2018). Learning analytics in competitive programming training systems. In: *2018 22nd International Conference Information Visualisation (IV)*, pages 321–325.

Di Luigi, W., Farina, G., Laura, L., Nanni, U., Temperini, M., and Versari, L. (2016). oii-web: an interactive online programming contest training system. *Olympiads in Informatics*, 10, 195–205.

Di Mascio, T., Laura, L., and Temperini, M. (2018). A framework for personalized competitive programming training. In: *2018 17th International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–8.

Fantozzi, P. and Laura, L. (2020a). Collaborative recommendations in online judges using autoencoder neural networks. In: *Proceedings of the 17th International Conference on Distributed Computing and Artificial Intelligence (DCAI 2020)*.

Fantozzi, P. and Laura, L. (2020b). A dynamic recommender system for online judges based on autoencoder neural networks. In: *13th International Workshop on Social and Personal Computing for Web-Supported Learning Communities (SPeL 2020)*.

Garcia-Mateos, G. and Fernandez-Aleman, J. L. (2009). Make learning fun with programming contests. In: *Transactions on Edutainment II*, pages 246–257. Springer.

Halim, S. and Halim, F. (2013). *Competitive Programming, Third Edition*. Lulu. com.

Li, S., Kawale, J., and Fu, Y. (2015). Deep collaborative filtering via marginalized denoising auto-encoder. In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 811–820.

Sedhain, S., Menon, A. K., Sanner, S., and Xie, L. (2015). Autorec: Autoencoders meet collaborative filtering. In: *Proceedings of the 24th International Conference on World Wide Web*, pages 111–112.

Strub, F. and Mary, J. (2015). Collaborative filtering with stacked denoising autoencoders and sparse inputs.

Toledo, R. Y. and Mota, Y. C. (2014). An e-learning collaborative filtering approach to suggest problems to solve in programming online judges. *Int. J. Distance Educ. Technol.*, 12(2), 51–65.

Van den Oord, A., Dieleman, S., and Schrauwen, B. (2013). Deep content-based music recommendation. In: *Advances in Neural Information Processing Systems*, pages 2643–2651.

Wang, H., Wang, N., and Yeung, D.-Y. (2015). Collaborative deep learning for recommender systems. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1235–1244.

Wang, T., Su, X.and Ma, P., Wang, Y., and Wang, K. (2011). Ability-training-oriented automated assessment in introductory programming course. *Computers and Education*, 56, 220–226.

Wang, X. and Wang, Y. (2014). Improving content-based and hybrid music recommendation using deep learning. In: *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 627–636.

Yera Toledo, R., Caballero Mota, Y., and Martínez, L. (2018). A Recommender System for Programming Online Judges Using Fuzzy Information Modeling. *Informatics*, 5(2), 17.

Zhang, F., Yuan, N. J., Lian, D., Xie, X., and Ma, W.-Y. (2016). Collaborative knowledge base embedding for recommender systems. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 353–362.

Zhang, Q., Wang, J., Huang, H., Huang, X., and Gong, Y. (2017a). Hashtag recommendation for multimodal microblog using co-attention network. In: *IJCAI*, pages 3420–3426.

Zhang, S., Yao, L., and Xu, X. (2017b). Autosvd++: An efficient hybrid collaborative filtering model via contractive auto-encoders. In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 957–960, New York, NY, USA. Association for Computing Machinery.

P. Fantozzi is involved in the training of the Italian team for the IOI since 2018. He is a Ph.D. student in Engineering in Computer Science at "Sapienza" University of Rome. He is lecturer at LUISS University for the courses: Lab of computer skills, Customer intelligence and big data analysis logics, Introduction to network science.

L. Laura is Associate Professor at Uninettuno university; he is involved in the training of the Italian team for the IOI since 2007, and since 2012 is in the organizing committee of the Italian Olympiads in Informatics. He got a Ph.D. in Computer Science in the "Sapienza" University of Rome.