ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 26-28 April 2017, i6doc.com publ., ISBN 978-287587039-1.
Available from http://www.i6doc.com/en/.

Collaborative Filtering with Neural Networks

Josef Feigl[1] and Martin Bogdan[2]

University of Leipzig - Department of Computer Engineering
Augustusplatz 10, 04109 Leipzig - Germany

Abstract. Collaborative filtering methods try to determine a user's preferences given their historical usage data. In this paper, a flexible neural network architecture to solve collaborative filtering problems is reviewed and further developed. It will be shown how modern adaptive learning rate methods can be modified to allow the network to be trained in about half the time without sacrificing any predictive performance. Additionally, the effects of Dropout on the performance of the model are evaluated. The results of this approach are demonstrated on the Netflix Prize dataset.

1 Introduction

Neural networks are usually not the first choice for collaborative filtering problems. A popular exception to this case are *Restricted Boltzmann Machines* (RBM). These generative stochastic artificial neural networks have proven to achieve strong performances on such tasks [1]. However, there are also strong connections between matrix factorization models and neural networks, which can be used to solve collaborative filtering problems: In [2], the authors describe a matrix factorization technique but also outline how it can be transferred into a neural network architecture.

Many improvements were made in the learning of neural networks in the last years. Such as Dropout [3], to improve regularization of deep neural networks, or RMSProp [4], to speed up the training time. In this paper, we will review this successful neural network architecture for solving typical collaborative filtering tasks, and adapt current state-of-the-art methods in neural network training for these kinds of problems.

This paper is structured as followed: In section 2 we start with a brief description of the general problem. Afterwards in section 3, we explain the neural net architecture and show how to improve it. The proposed model is evaluated in section 4 on the Netflix Prize dataset. We summarize our findings in section 5.

2 Preliminaries

Suppose we have a set of users $U = \{1, \dots, N\}$ and a set of items $I = \{1, \dots, M\}$ with $N, M \in \mathbb{N}$. Let $r_{ui} \in \mathbb{R}$ be the rating of user $u \in U$ given to item $i \in I$. Every user rates an item at most once. We have a set of known ratings $R = \{r_{ui} | u \in U, i \in I\}$ and denote $\hat{r}_{ui}$ as the predictions made by a model for a rating r_{ui} and $\hat{R}$ as the set of predictions made for R. Our goal is to minimize the root mean square error (RMSE) error between R and $\hat{R}$.

ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 26-28 April 2017, i6doc.com publ., ISBN 978-287587039-1.
Available from http://www.i6doc.com/en/.

This set R can be represented as a sparse matrix $\mathbf{R} \in \mathbb{R}^{N \times M}$, where each row represents an user and each column an item. The entries of this matrix are given by r_{ui}. Usually, most entries of this matrix are missing.

Common matrix factorization models try to solve this problem by decomposing this matrix into two smaller latent factor matrices $\mathbf{U} \in \mathbb{R}^{N \times K}$ and $\mathbf{I} \in \mathbb{R}^{K \times M}$ with $K \ll N, M$ such that

$$\mathbf{R} \approx \mathbf{UI}. \tag{1}$$

3 Model

Our basic model to find these two latent factor matrices will be a standard multilayer perceptron without biases. The network has three layers L: An input layer L^1 with N units, a hidden layer L^2 with K units and an output layer L^3 with M units. Thus, our network has as many units in the input layer as there are users and as many units in the output layer as there are items. The size of the hidden layer determines the size of the latent factors.

Let $\mathbf{W}^l_{ij}$ be the weight connecting the i^{th} unit of layer L^l with the j^{th} unit of layer L^{l-1}. Following this notation, we define $\mathbf{W}^l_{\cdot u}$ as the set of incoming weights from the u^{th} unit of layer L^{l-1} to all units of layer L^l. Let $\mathbf{W}^l$ furthermore be the set of all weights connecting to the l^{th} layer. All weights are initialized with uniformly distributed random numbers from the range $[-0.01, 0.01]$.

The weights $\mathbf{W}^2$ can be interpreted as the representations of the users U. Thus, $\mathbf{W}^2_{\cdot u} \in \mathbb{R}^K$ is the latent factor vector of user $u \in U$. The weights $\mathbf{W}^3$ can be interpreted in the same way as the representations of the items I. We will interpret the weights $\mathbf{W}^2$ and $\mathbf{W}^3$ as weight matrices with $\mathbf{W}^2 \in \mathbb{R}^{N \times K}$ and $\mathbf{W}^3 \in \mathbb{R}^{K \times M}$ for the rest of this paper.

The activation $\mathbf{a}$ of the layer l is then given by

$$\mathbf{a}^l = f(\mathbf{W}^l \mathbf{a}^{l-1}), \tag{2}$$

where $f : \mathbb{R} \to \mathbb{R}$ is the activation function. We achieved the best results by using identity activation functions.

3.1 Training

3.1.1 Forward-Propagation

Each rating $r_{ui} \in R$ is used as a single training example. A binarized version $\mathbf{a}^1 = 1_u \in \{0, 1\}^N$ of u serves as the input for the network. It is defined as the indicator vector $1_u := (x_0, x_1, \cdots, x_N)$ with $x_j = 1$ if $j = u$ and $x_j = 0$ otherwise. Using 1_u as input also means that only the weights $\mathbf{W}^2_{\cdot u}$ contribute anything to the output of the hidden layer L^2. Thus, the output $\mathbf{a}^2$ is given by

$$\begin{aligned} \mathbf{a}^2 &= f(\mathbf{W}^2 1_u) \\ &= f(\mathbf{W}^2_{\cdot u}). \end{aligned} \tag{3}$$

ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 26-28 April 2017, i6doc.com publ., ISBN 978-287587039-1.
Available from http://www.i6doc.com/en/.

In practice, this can be achieved by simply selecting the u^{th} row of the weight matrix $\mathbf{W}^2$ and applying the given activation function to it.

For every rating $r_{ui} \in R$, we are only computing the output for the unit of item i. This means, that only the weights $\mathbf{W}^3_{i\cdot}$ are used to compute the output of the network $\hat{r}_{ui}$:

$$\hat{r}_{ui} = \mathbf{a}^3 = f(\mathbf{W}^3_{i\cdot}\mathbf{a}^2). \tag{4}$$

In a network with no biases and only identity activation functions, the prediction for a rating r_{ui} is given by multiplying the u^{th} row of the user weight matrix $\mathbf{W}^2$ and the i^{th} column of the item weight matrix $\mathbf{W}^3$. Therefore, we achieve the decomposition of matrix $\mathbf{R}$ by:

$$\mathbf{R} \approx \mathbf{W}^2\mathbf{W}^3. \tag{5}$$

To shorten the notations, we say that for each rating r_{ui} only the units u and i are *active*. That means, that only the weights $\mathbf{W}^2_{\cdot u}$ and $\mathbf{W}^3_{i\cdot}$ are contributing anything to the output of the network.

3.1.2 Backpropagation

We are trying to solve a typical regression problem and therefore using the mean squared error as the cost function C_0 for the training of the network:

$$C_0 = \frac{1}{2|R|} \sum_{r_{ui} \in R} (r_{ui} - \hat{r}_{ui})^2 \tag{6}$$

We backpropagate the error $r_{ui} - \hat{r}_{ui}$ only through the active units of the model. This way, only the weights of the active units are updated accordingly to the common backpropagation algorithm [5]. All other weights remain unchanged.

A training epoch is done, when all ratings $r_{ui} \in R$ were processed.

3.2 Parameter Updates

Collaborative filtering tasks are often based on highly skewed datasets. Since we are always only updating the weights of active units, users with many ratings or very popular items get updated much more often, as they are more present in the training dataset. This can lead to overfitting of popular users or items and underfitting of unpopular ones. The popular momentum method for parameter updates does not account to this problem as a single learning rate parameter is used globally and equally for all weight updates [6]. To avoid these problems and improve convergence, it is helpful to use per-dimension learning rates, e.g. one learning rate for each weight. One adaptive learning rate method is RMSProp, which was applied with success in practice [7]. However, apart from the increased convergence we also noticed a decreased predictive performance (see 4.2).

ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 26-28 April 2017, i6doc.com publ., ISBN 978-287587039-1.
Available from http://www.i6doc.com/en/.

3.2.1 Consecutive Approach

To combine the speed of RMSProp and the predictive performance of the momentum method, we make use of a consecutive approach and apply RMSProp only in the first k training iteration and then switch to the momentum method.

We use $\Delta\mathbf{W}(t)$ as a short notation for the update of all active weights of the model at epoch $t > 0$. Our proposed update method is then given by:

$$\Delta\mathbf{W}(t) = \begin{cases} -\frac{\eta}{\sqrt{\alpha \cdot r(t-1) + (1-\alpha) \cdot g(t)^2 + \gamma}} \cdot g(t), & \text{if } t \leq k \\ p\Delta\mathbf{W}(t-1) - \eta g(t), & \text{otherwise} \end{cases} \tag{7}$$

We denote p as the momentum parameter, η as the learning rate and $g(t)$ as the gradient of the weights at iteration t computed during backpropagation. The parameter $0 < \alpha < 1$ denotes the decay rate. The damping factor γ is used to stabilize the denominator [4].

3.3 Bias Layer

We extend the neural network by integrating multiple biases b to further improve the final accuracy of the model and to speed up training. This is done by introducing a bias layer as the last layer of the network. This layer will modify the output of the network by

$$\hat{r}_{ui} = \mathbf{a}^3 + b_u + b_i + b_g, \tag{8}$$

where b_g represents the global bias, b_u the user bias and b_i the item bias. All biases are initialized with zeros and will be updated only when they are active, whereas the global bias is active for all samples.

3.4 Regularization

Overfitting the training data is a common problem for neural networks, especially for networks with many parameters. This gets even more severe when ratings have to be estimated for rarely active users or items.

A common method to prevent overfitting is L_2 regularization or *weight decay*, which attempts to limit the growth of the weights by shrinking them proportionally towards zero.

3.4.1 Dropout

Dropout is a simple and effective form of regularization, which is rather different compared to L_2 regularization. By ignoring (dropping out) a certain percentage of all neurons of a layer, we are sampling different sub-nets from the full neural net at every iteration. One natural explanation for its use as a regularization technique is that every neuron has to learn more robust features, since it cannot rely on the incoming neurons [3].

We will make use of both methods to regularize our model.

ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 26-28 April 2017, i6doc.com publ., ISBN 978-287587039-1.
Available from http://www.i6doc.com/en/.

4 Experiments and Results

4.1 Setting

The Netflix Prize dataset contains 100 480 507 ratings coming from 480 189 users for 17 770 movies. All ratings are integers in the range 1 to 5. About 99% of all possible user-movie-combinations are unrated. A sample of this training set, the probe dataset, which consists of 1 408 395 ratings, will be used as validation data throughout this paper. The RMSE is used to evaluate the accuracy of all models. Netflix's own benchmark, *Cinematch*, achieved a RMSE of 0.9514 [8].

4.2 Impact of adaptive updates

We found that the use of RMSProp leads to a much faster convergence compared to the momentum method (see Figure 1, left panel). The net with RMSProp reached its best result after only 4 epochs, compared to the 16 epochs of the momentum method. However, the net trained with the momentum method converged to a significantly lower final error.

Using our proposed consecutive approach $RMSProp(k)$ with $k = 1$, we achieved about the same final error as the momentum method in about half the time. This is a state-of-the-art result for this dataset (see Table 1).

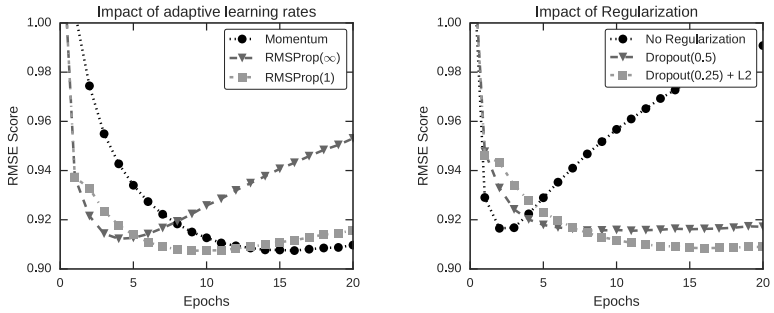

Fig. 1: $RMSProp(1)$ reaches the same minimum as the momentum method in about half the time (left). Dropout strongly regularizes the neural net but also decreases the predictive performance compared to L_2 regularization (right). We used $K = 100$ for all plotted models.

4.3 Impact of Dropout

We applied Dropout only in the hidden layer of the net, but have not noticed any improvements in terms of the final accuracy of the model for the Netflix Prize dataset (see Figure 1, right panel). While it is still useful as a form of regularization, models using L_2 regularization achieved a stronger predictive performance.

ESANN 2017 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 26-28 April 2017, i6doc.com publ., ISBN 978-287587039-1. Available from http://www.i6doc.com/en/.

Table 1: Comparison with other authors

Model	**RMSE**
Cinematch	0.9514
Singular Value Decomposition [8]	0.9167
Conditional RBM [1]	0.9070 [1]
Our Model ($K = 500$)	0.9048

[1] estimated from Figure 3.

The combination of Dropout and L_2 regularization did not lead to any noticeable improvements, too. However, whereas models using the L_2 regularization overfitted rather quickly after reaching their minimum, models with Dropout remained almost stable at their minimum for the remaining training epochs.

5 Summary

In this paper, we have reviewed and updated a neural net architecture to solve collaborative filtering problems. Our experiments demonstrated that Dropout, while still strongly regularizing the net, is not able to significantly improve the results. We have shown how to combine the classical momentum method with modern adaptive learning rate methods to reduce training time by almost 50% without losing predictive performance. This approach is able to achieve a state-of-the-art performance on the Netflix Prize dataset.

References

[1] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 791–798, New York, NY, USA, 2007. ACM.

[2] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res.*, 10:623–656, June 2009.

[3] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[4] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[5] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[6] B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[7] Tom Schaul, Ioannis Antonoglou, and David Silver. Unit tests for stochastic optimization. In *International Conference on Learning Representations*, Banff, Canada, 2014.

[8] Andrey Feuerverger, Yu He, and Shashi Khatri. Statistical significance of the netflix challenge. *Statist. Sci.*, 27(2):202–231, 05 2012.