Recurrent Neural Network for Recommendation and User Representation

Ole Steinar Lillestøl Skrede

November 2016

Department of Computer and Information Science
Norwegian University of Science and Technology

Supervisor: Massimiliano Ruocco

Abstract

Contents

Chapter 1

Introduction

BLa bla bla

Chapter 2

Motivations

Matrix factorization has no sense of time, and is less efficient when we have less data (like in short sessions, without user history). RNNs utilizes memory of past events and can hopefully use the information of the last actions (and their sequence) to weigh up for the lack of more data on the user.

Matrix factorization approaches often resort to item-to-item recommendations (recommending similar items) in the case oh short sessions.

Progress in model architectures, training algorithms, and parallell computing has taken RNNs from being more of a curiosity to becoming a model that can compete with state-of-the-art models. The improvements have also made RNNs easier to use. Training them was really difficult earlier.

Many other models are tied by restrictions on the input and output data. RNNs avoids the assumption of independence between examples and are not limited to fixed-dimension input and output.

Chapter 3

State of the Art

A lot of this should be in the introduction instead
I have written too much about RNN, should write about the state of the art on session based recommender systems (?) Much of this information is sourced from [1] and should maybe be referenced more. However (what I have used so far) is pretty obvious (?) stuff for those who know a bit about RNNs, so might not be needed. This is a mess and needs to be cleaned up. *Seperate between state of the art on recommender systems and state of the art on RNN.

Neural networks achieve great results in many different machine learning tasks these days. One example is the use of deep neural networks in Google DeepMind's AlphaGo program, which became the first Computer Go program that to beat a professional human [2].

The growth in computing power has made neural networks more applicable. Also, they work well in some cases where other methods struggle.

Standard neural networks are not well suited to handle cases where the training and test data are not independent. And they are not suited for inputs and outputs of varying length either. Recurrent neural networks works better in these cases. If neural networks already have state of the art performance on a problem then it might be worth looking into using recurrent neural networks to further increase the performance. Standard neural networks are not the only model that lacks the ability to evaluate data points in sequence, for example support vector machines and logistic regression do not have a sense of time either.

Markov models can represent time dependencies, but becomes infeasible to use when the number of dependencies grow big. RNNs have the advantage that they can represent an exponential amount of hidden states in the hidden nodes.

This might not be worth mentioning.. Earlier it has been difficult to train RNNs due to vanishing and exploding gradients, but improved architectures (LSTM) and better gradient-following heuristics have made RNNs very usable [1].

For more detail on the state of the art on RNNs, we refer to a comprehensive paper by Lipton et al. called "A Critical Review of Recurrent Neural Networks for Sequence Learning" [1]

RNNs have become very popular in the last few years. Variants of Long Short Term Memory (LSTM) networks have improved RNNs a lot. Nowadays, researchers try to add attention to RNNs. This means that the RNN picks information from a larger collection of information, to look at during each step [3]. We are not exploring this in here.

LSTM and Bidirectional Recurrent Neural Networks (BRNN) are two of the most successful RNN architectures. LSTM overcomes the earlier problems with training. BRNN uses information from both the future and the past to output the current information. These two methods can be combined [1].

BRNN is not applicable in the online setting, since we don't have access to information from the future.

Recurrent neural networks have recently been used to outperform earlier neural network approaches and matching the best results of non-neural network approaches on natural language translation [4]. In 2014 RNN was used to do image captioning with then state-of-the-art results [5].

Some of the state-of-the-art models for recommender systems are matrix factorization models and restricted Bolzmann machines (RBM) [6].

Neighborhood methods have been extensively used in session-based recommendations. Elaborate more on this

3.1 Matrix factorization models

Matrix factorization is a collaborative filtering approach that tries to estimate hidden latent features. These latent features determines how a user would rate an item. For example if the items are movies, then a user might prefer action movies, so we want to discover this latent feature and thus be able to predict the rating from a certain user on a certain item. Quickly explained we have a matrix $\mathbf{R}$ that contains all known ratings done by users on items. $\mathbf{R}$ has the size $|\mathbf{U}| \times |\mathbf{I}|$, where $\mathbf{U}$ is the set of users and $\mathbf{I}$ is the set of items. If we want to discover L latent features, then we try to find two matrices $\mathbf{P}$ (a $|\mathbf{U}| \times L$ matrix) and $\mathbf{Q}$ (a $|\mathbf{I}| \times L$ matrix), such that

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{R}}$$

Here $\hat{\mathbf{R}}$ is an approximation to $\mathbf{R}$. This approximation can be found by making initial guesses for $\mathbf{P}$ and $\mathbf{Q}$, and then use gradient decent to improve the approximation over multiple iterations. Then, to find an approximation of a user rating on an item, we calculate the dot product of the two vectors corresponding to the user and the item in $\mathbf{P}$ and $\mathbf{Q}$ respectively. A more detailed explanation of this can be found here [7]

3.2 Restricted Bolzmann machines

Restricted Bolzmann machines also try to find latent features that objects might have in common. RBM are a form of neural network that has a visible and a hidden layer. The hidden layer has nodes for each object (e.g. movies) and the visible nodes represent the features (e.g. being a sci-fi movie, or being an animated movie). Given input data, the BRM tries to fit the weights in the network to best represent the observed data. This input could as an example be binary vectors that represent which movies users like, where each vector represents one user. After the training is done, the BRM can be asked which movies someone who likes sci-fi probably would like (if we know that a node represents this feature), and the network would activate the nodes that represented the sci-fi movies. Note that this activation is done with probability, so a sci-fi movie would have a high chance of being activated and visa versa for non-sci-fi movies. We don't need to know which features the nodes represent, when we have a new user that we would like to recommend movies to, we use an input vector like the ones used in training and this will activate the relevant features for this user, and the RBM can then recommend movies. I don't like this explanation.. I don't think it is very good. Might need to read more about BRM to be able to give a better explanation. But the point of this paper is not to be a comprehensive guide to BRMs, so I should not use too much space on it. Maybe use a short and better explanation and refer to a more comprehensive explanation?

3.3 About the papers I have read

This stuff is sort of about the state of the art on RNN as recommender system.

3.3.1 Session-based Recommendations with Recurrent Neural Networks

Hidasi et al. [8] have recently proposed to use RNNs in a recommender system. In their experiments they achieve marked improvements over currently widely used approaches. They propose a new ranking loss function that is suited to training their model. In the network they use GRU layers, this is used to combat the vanishing gradient problem. As input they used 1-of-N encoding (one-hot encoding). They also tried using an input vector that also encoded the previous clicked items, with the hope to strengthen the memory effect. The latter input method is a normalized vector where the oldest clicks have the lowest value. Also, they tried using an embedding layer between the input and GRU layers, but the 1-of-N encoding always performed better. But connecting the input layer to the deeper GRU layers as well, improved results. They session parallell mini-batcehs. Ask Massimiliano how it is possible to use session parallel mini-batches here. My intuition is that one needs to run sessions in serial order in a batch, since the hidden states must be reset between each session. So the sessions can't interfere with each other... There might be hundreds of thousands of items, so it is not feasible to calculate a score for each item at each step, sampling is therefore used. The highest ranked items in the output are chosen as the positive examples, and the highest ranking items from other training examples are used as negative examples. These negative examples are probably popular items that the user ignored. Bayesian Personalized Ranking and a custom TOP1 ranking are used as loss functions. The result of the experiments is that the RNN performs substantially better than the item-KNN approach (which was the best of the baseline approaches used to compare results).

3.3.2 Modelling Contextual Information in Session-Aware Recommender Systems with Neural Networks

Build on the above paper, but only considers unseen items as valid recommendations.

Here a session is defined to end when a user has been inactive for more than a predefined number of minutes.

They use a richer description of the events than the last paper. The last paper used item clicks as events and only cared about what item that was. Here multiple types of events are used, and these have a number of properties but are all related to an item. And each item is described by a number of item attributes. Seems like no item id is available, only a description of each event by its attributes.

Both Feed Forward Neural Networks (FFNN) and RNNs are used to produce the top-N recommendations in this model. RNN is used to capture data dependency between events. FFNN is as a ranking score estimator. So the RNN creates a representation of the current state of the session, which is sent to the FFNN along with the data on the new item. Both the event data (input to the RNN) and the item data are embedded before being sent further. The embedded item data and the RNN representation output of the session/event is gathered together and dropout is applied as regularization on the FFNN. Not clear how the dropout is applied, but seems to be on the FFNN. However the figure shows a merge/dropout layer before the FFNN, so not really sure..

They also use a Matrix Factorization model. Consider explaining this model that they use a bit more. Both models (MF and NN) outperform the chosen baseline approaches. RNN considered more attractive when no session modeling assumption is made.

3.3.3 Deep Neural Networks for YouTube Recommendations

Write a bit about this one as well?

Chapter 4

Challanges

In some cases it might be hard to evaluate the output, e.g. in a translation problem where there might be multiple correct outputs. In our case we sort of have the same problem in that there might be multiple items that the user would click on if presented to him.

So the recommender systems score on the test set might not be a very good indicator of how good a recommender system is. Can be hard to tell whether a system is truly strong or if it only overfits the performance metrics.

Much of the recent improvement and success of RNNs come from the exploration of new architectures. Therefore, much of the work in making a good recommender system with RNN might be to find a good architecture. Hopefully we can use lessons and knowledge from similar problems where RNNs have been used with promising results

Chapter 5

Ideas

- Encode more information in input to RNN? E.g. time.
- Use RNN to learn embedding space: SESSION_2_VEC
- Combine session output vectors to a user vector: USER_2_VEC
- Use word2vec to represent title of book. Give this as input to RNN (Or use vector representation of other types of input)
- Use LSTM (long term short term memory) or BRNN instead of GRU. (GRU is a special version of LTSM).
- When we have a rich user history (logged in users), we can make use of information such as: user generally give low ratings, an item generally get low ratings. We don't have less such information in shorter sessions, but there might still be some information we could extract and utilize. Two different types of information here (sort of): extra information that the (e.g.) e-commerce site can supply (e.g. geo location), and information that we can extract from the data we already have in a dataset. The second variant is sort of what we want the NN to discover on its own though.
- It is (probably) inefficient to use the one-hot encoding to encode the input items to our RNN, maybe it would be smart to use a word2vec approach and thus be able to capture some more meaning in the item representation. If sessions are viewed as sentences, a slightly modified word2vec algorithm should be able to encode the input items. But are items words or characters in such an analogy? Or maybe it does not matter.
- Since there might be multiple relevant items the recommender system could present to the user and that the user would want to click (not one clearly

correct output), it could be useful to let users give feedback to the system by e.g. rating the top N items suggested by the system. In practice, if the RNN recommender system is used online, this is partially solved since it can show more than one item to the user and get feedback on which item was clicked (if any). But when the user clicks on one item it does not tell us whether the others were totally irrelevant or just a bit less relevant than the item clicked.

If the user clicks on one of the other suggestions later in the session, this might be perceived as a positive feedback to that recommendation, but it is hard to make such assumptions and speculations in a good way.

Chapter 6

Experimentation

sdfg

Chapter 7

Analysis of Results

dsfd

Chapter 8

Conclusion

sdfgsdfg

Chapter 9

Further Work

sdfgsdfg

Appendix A

List of Acronyms

RNN Recurrent Neural Network

RBM Restricted Bolzmann machines

LSTM Long Short Term Memory

FFNN Feed Forward Neural Networks

Appendix B

appendix-b

Bibliography

[1] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[2] BBC. Google achieves ai 'breakthrough' by beating go champion. `http://www.bbc.com/news/technology-35420579`.

[3] Christopher Olah. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[4] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[5] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, and Alan L. Yuille. Deep captioning with multimodal recurrent neural networks (m-rnn). *CoRR*, abs/1412.6632, 2014.

[6] Will Kirwin. Implicit recommender systems: Biased matrix factorization. `http://activisiongamescience.github.io/2016/01/11/Implicit-Recommender-Systems-Biased-Matrix-Factorization/`.

[7] Albert Au Yeung. Matrix factorization: A simple tutorial and implementation in python. `http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/`.

[8] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *CoRR*, abs/1511.06939, 2015.