DOI: 10.1609/aaai.12013

RECOMMENDER SYSTEMS

Deep learning for recommender systems: A Netflix case study

Harald Steck | **Linas Baltrunas** | **Ehtsham Elahi** | **Dawen Liang** | **Yves Raimond** | **Justin Basilico**

Correspondence
Harald Steck, Netflix.
Email: hsteck@netflix.com

Abstract

Deep learning has profoundly impacted many areas of machine learning. However, it took a while for its impact to be felt in the field of recommender systems. In this article, we outline some of the challenges encountered and lessons learned in using deep learning for recommender systems at Netflix. We first provide an overview of the various recommendation tasks on the Netflix service. We found that different model architectures excel at different tasks. Even though many deep-learning models can be understood as extensions of existing (simple) recommendation algorithms, we initially did not observe significant improvements in performance over well-tuned non-deep-learning approaches. Only when we added numerous features of heterogeneous types to the input data, deep-learning models did start to shine in our setting. We also observed that deep-learning methods can exacerbate the problem of offline–online metric (mis-)alignment. After addressing these challenges, deep learning has ultimately resulted in large improvements to our recommendations as measured by both offline and online metrics. On the practical side, integrating deep-learning toolboxes in our system has made it faster and easier to implement and experiment with both deep-learning and non-deep-learning approaches for various recommendation tasks. We conclude this article by summarizing our take-aways that may generalize to other applications beyond Netflix.

INTRODUCTION

In the early 2010s, deep learning was taking off in the machine-learning community fueled by impressive results on a variety of tasks in different domains including computer vision, speech recognition, and natural language processing (NLP). At that time there was a stir in the air within the recommender-systems research community: Will the wave of deep learning also wash over recommenders to deliver tremendous improvements? As with many others, we at Netflix were intrigued by this question and the potential of deep learning to improve our recommendations. While the answer is now quite clear that deep learning is useful for recommender systems, the path to understand where deep learning is beneficial over existing recommendation approaches was an arduous one. This is evidenced by how many years it took for such methods to get traction in the research community. But it was a rewarding path as evidenced by a subsequent bloom of work on the subject. Our own investigations into deep

learning at Netflix took a similar path: initial exuberance faced the stark reality of well-tuned baselines. This led to a clearer understanding of the relationship between deep learning and other recommender models. The obstacles gave way when we combined the lessons from these approaches with insights about what kinds of problems different deep-learning approaches excel as well as their limitations. Along the way, it also yielded practical lessons on how to get deep learning to work in real-world recommendation settings.

In this article, we first set the scene for our journey with an overview of various personalization tasks on the Netflix service and a focus on the key task of personalized ranking of movies and TV shows on the Netflix service. We then discuss several specific properties of the data available in recommendation settings like at Netflix, and the practical challenges they impose for creating recommendation algorithms.

Looking at deep learning through a recommendation lens allowed us to investigate a variety of models for different recommendation tasks at Netflix. From this, we will share insights into a couple of important types of models. First, we discuss bag-of-items models that make use of the set of videos watched by a member. While powerful, these models ignore temporal information, so we subsequently discuss sequential models that address this deficit. We found both approaches to be valuable for different tasks.

After initial struggles with deep-learning approaches, our experiments indicated that deep-learning algorithms especially started to shine in recommendation problems when we provided them with additional heterogeneous features and information sources. In contrast, other models remained competitive in the classic recommendation setting where only user-item interaction-data are used (as is often the case in the literature, see also Ferrari Dacrema, Cremonesi, and Jannach (2019)), and when these methods were properly tuned. However, this flexibility of deep-learning models allowed us to find scenarios where we could build models that obtained large improvements in offline metrics evaluated on historical data.

After finding improvements in offline metrics, we subsequently found that these gains, even when very large, did not always translate to online performance in an A/B test with real members. To address this, we needed new offline metrics that were better proxies of the online metrics. Apart from that, we cover additional practical aspects required to use deep learning in a recommendation system supporting hundreds of millions of members. The available deep-learning toolboxes provide a flexible framework that makes it very easy to develop and modify model architectures for recommender systems in practice.

Finally, while we focus on key learnings from using deep learning for recommendation systems at Netflix, we will also outline take-aways that may generalize to other applications as well.

RECOMMENDATIONS AT NETFLIX

The main task of our recommender system at Netflix is to help our members discover content that they will watch and enjoy to maximize their long-term satisfaction. This is a challenging problem for many reasons, including that every person is unique, has a multitude of interests that can vary in different contexts, and need a recommender system most when they are not sure what they want to watch. Doing this well means that each member gets a unique experience that allows them to get the most out of Netflix. As a monthly subscription service, member satisfaction is tightly coupled to a person's likelihood to retain with our service, which directly impacts our revenue. Therefore, the value of a recommender system can be measured by the increase in member retention. Over years of the development of personalization and recommendation technologies, we have been able to repeatedly create meaningful improvements in retention (Gomez-Uribe and Hunt 2015).

The specific recommendation task as well as the available data play a crucial role when it comes to the question of which algorithm works the best, as we found in numerous controlled online experiments. At a video-streaming service, the primary decision of the recommender system is to choose which videos to show to each member on their Netflix homepage after they log into their profile on any device. This personalization task is itself divided into different sub-tasks using recommender systems designed to cover different member needs, each of which can be powered by a different algorithm.

At Netflix there is no single model that drives all recommendations but rather a set of techniques that are all aligned on their goal to increase member satisfaction. Through experimentation with various kinds of recommendation algorithms, we found that there is no "silver bullet" the best-performing method (whether deep learning or other) depends on the specific recommendation task to be solved as well as on the available data. For this reason, different kinds of machine learning models are used to generate personalized recommendations for the different parts (e.g., rows) of the Netflix homepage. Various aspects of the Netflix recommender system, prior to using deep learning, are outlined in Gomez-Uribe and Hunt (2015). Dividing the recommendation problem in multiple sub-tasks allows us to combine a diversity of different approaches and also makes the research

FIGURE 1 Netflix homepage where we highlighted various recommendation tasks

and development of new or improved recommendation algorithms more modular and scalable.

Figure 1 displays a Netflix homepage with red circles enumerating different recommendation tasks, each of which is powered by a different algorithm. For example, there is a dedicated algorithm (1) for choosing the first video to display prominently at the top of the homepage, another one for ranking already-watched videos that the user may want to continue watching (7), as well as others intended to help our members discover new videos (5). The output of each of these algorithms can be surfaced as different rows of recommended videos on the homepage. There are several more personalization tasks on the Netflix service that are beyond the scope of this article. For instance, there is an algorithm (4, 6) that selects which rows to present in a personalized way to create the structure of the homepage (Alvino and Basilico 2015). Apart from that, messages and notifications (3) sent to our members are personalized as well. We also use recommendation techniques as part of our search (2) engine (Lamkhede and Das 2019). In this article, we focus primarily on the task of recommending videos from the Netflix catalog to each member, as it is the most common across our recommendation settings. However, many of the insights we share extend to these other tasks as well.

Besides the recommendation task, the available data and its properties have a crucial impact on which recommendation algorithm works well. The first and most important distinction is whether the data contain only the user-item interactions (as is often the case in publicly available data sets used in the literature), or additional information like user attributes, item attributes or contextual information regarding the user-item interaction. Through experimentation we found that more complex models, including deep-learning algorithms, shine when we enriched data with such additional heterogeneous features.

Even though several common properties of the data have been discussed in the literature of recommender systems, it is worth reviewing them briefly before we outline additional challenges in the data for building real-world recommender-systems. The key differences to the data-sets used in other domains are as follows: first

and foremost, the observed/collected data are missing not at random (Hernández-Lobato, Houlsby, and Ghahramani 2014; Liang et al. 2016; Marlin, Zemel, and Roweis 2005; Marlin et al. 2007; Marlin and Zemel 2009; Steck 2010), that is, the entries with observed positives (e.g., played videos, clicked items, given ratings or thumbs, etc.) are not randomly distributed in the user-item interaction-matrix. This is a crucial difference to fields like compressive sensing or matrix completion, where the entries in the matrix are typically assumed to be missing at random. Second, the unobserved entries in a user-item interaction matrix may either be (true) negatives (i.e., the user is truly not interested in this item), or positives that were not observed (yet). Third, the observed data are typically extremely sparse, and the observed positives are very noisy as they originate from a stochastic process. Fourth, there is a large popularity-skew present in the data, that is, the popularities of the various items follow approximately a power-law distribution, resulting in many orders of magnitude in differences in the popularities of the various items. Regarding the users, there is a similar (approximate) power-law distribution, with a small number of very active users and a large number of less active users. This power-law distribution can cause modeling challenges due to distribution mismatch. It also poses a challenge in making fair and accurate recommendations regarding unpopular items or for users with low activity.

In a real-world recommender system, the various biases in the user-item interaction-data, like presentation or position biases, can possibly be amplified due to a *feedback loop*, where the recommender system is trained on the observed user-actions from a previous time-step, which may be biased due to the recommendations shown to the users at that time (Chaney, Stewart, and Engelhardt 2018). This is due to presentation bias, where users are more likely to interact with items shown more prominently by the system. Breaking (or at least dampening) the feedback loop is a key challenge in real-world recommender-systems. This poses not only a challenge for training recommender systems on the data that have been collected, but also results in a notable mismatch between offline and online metrics, as outlined later in this article.

MODELING APPROACHES

In this section, we outline our learnings when experimenting with various deep-learning models for recommendations, from simple baselines to more complex approaches. We divide these models into two groups in the following: bag-of-items approaches and sequential models. Properties that are common to both types of models are discussed as well. Following that, we describe our key insight, namely that deep-learning algorithms for recommendations excel at combining many heterogeneous features. In contrast, we found that simpler models that are well-tuned were competitive when only the user-item interactions were used (as is often the case in the literature, see also Ferrari Dacrema, Cremonesi, and Jannach (2019)).

Bag-of-items approaches

Analogous to bag-of-words approaches in NLP, many recommender systems make the bag-of-items assumption, which is a bag-of-videos in our case. In this approach, the model ignores the temporal order in which a user played videos and instead treats them as a set. The most common way to represent the training data in a bag-of-items setting is to turn the data into a sparse user-by-item matrix where each non-empty entry represents the user's feedback on an item, either implicit (like plays or clicks) or explicit (like ratings such as thumbs up or down). Compared to words, the amount of sequential information in video engagements is lower, with some exceptions like movie sequels or episodes of TV shows. Though there are also some temporal dynamics due to viewing trends, the bag-of-videos assumption can provide a useful approximation to the real world when we model a user's long-term interests, which tend to be relatively stable. Because of this and the sparsity of feedback, we have found it important to understand these long-term interests in our recommendations.

In the literature on recommender systems, especially during the Netflix Prize Competition (Bennet and Lanning 2007), several bag-of-items approaches were developed, including Restricted Boltzmann Machines (RBMs) (Salakhutdinov, Mnih, and Hinton 2007) as well as several variants of matrix-factorization models such as SVD++ (Koren 2008) or asymmetric matrix factorization (Paterek 2007; Jahrer, Toscher, and Legenstein 2010). Interestingly, both techniques have connections with deep learning: around the same time as the Netflix Prize, RBMs were also used to pre-train deep-learning models for computer-vision tasks in their early days (Hinton, Osindero, and Teh 2006; Hinton and Salakhutdinov 2006). However, the instability in training RBMs made it difficult to adapt them to other recommendation tasks beyond rating prediction. However, a generalized form in autoencoders did show more promise. The connection between matrix factorization and deep learning appears less obvious. But as we will illustrate, they can actually be linked through autoencoders.

Autoencoders have been used in various application areas, including machine translation (e.g., Tu et al. 2017) and computer vision (e.g., Rifai et al. 2011), as well as recommender systems (e.g., Liang et al. 2018; Steck 2019; Wu

et al. 2016). An autoencoder is comprised of two parts: an encoder and a decoder, both of which may consist of several hidden layers with nonlinear activation functions. The encoder converts the input-vector into a low-dimensional embedding, while the decoder converts the embedding back to the output-vector. The goal of autoencoders is to produce an output-vector that is close to the input-vector by minimizing the reconstruction error. In the recommendation task, the input (and output) vector is typically a multi-hot encoding of a user's play-history, that is, each element of the vector refers to a video, and its value is 1 if the user has played the video, and 0 otherwise. Besides binary vectors, continuous-valued ones may also be used, for example, to capture the time-duration a user watched a video.

The autoencoder subsumes several "traditional" recommendation approaches as special cases. First, asymmetric matrix factorization (MF) (Jahrer, Toscher, and Legenstein 2010; Paterek 2007) can be viewed as a linear autoencoder with a single hidden layer. Note that asymmetric matrix factorization is different from (standard) matrix factorization in the following way (in fact, SVD++ (Koren 2008) combines both asymmetric MF and standard MF): when the training-data are given in the form of a user-video interaction-matrix, matrix factorization learns a matrix of user-embeddings as well as a matrix of video-embeddings. In contrast, asymmetric matrix factorization learns two (different) matrices of video-embeddings (corresponding to the encoder and decoder of the autoencoder) and constrains the user-embeddings to be the average of the video-embeddings (in the encoder) regarding the videos played by the user. In standard matrix-factorization, making recommendations for a new user or updating for a user with new data would involve an optimization procedure (a process which is typically referred to as "fold-in") to obtain the corresponding user-embedding. This typically involves running one step of an optimization algorithm to solve for the user-specific parameters (i.e., user-embedding) while keeping the item-embeddings fixed. On the other hand, this "fold-in" procedure is not needed when using asymmetric matrix factorization (or more generally, nonlinear autoencoder models), where recommendations can be computed by passing the input-vector through the encoder and decoder. This "amortized inference" proves widely applicable across many domains (Gershman and Goodman 2014; Kingma and Welling 2013; Rezende, S, and D 2014).

Second, neighborhood-based approaches may also be understood as a special case of autoencoders, given that both approaches are based on an item-to-item function. If we increase the dimension of the hidden layer in a linear autoencoder to be identical to the number of items in the input (and output) vector, we obtain a full-rank model (instead of a low-rank model as typically used for autoencoders) that is equivalent to a full-rank model having a single item-item matrix of model-parameters. This matrix corresponds to the item-item similarity matrix used in neighborhood-based approaches (see, e.g., Aiolli 2013; Verstrepen and Goethals 2014; Volkovs and Yu 2015). While this item-item matrix is often constructed using various heuristics (like cosine-similarity or various re-scaled/normalized versions of empirical item-item co-occurrence matrices), the view of neighborhood-based approaches as autoencoders provides a principled way of learning/optimizing such an item-item matrix, for example, see the EASE model (Steck 2019). Also the SLIM model (Ning and Karypis 2011) can be viewed from this perspective by including additional constraints on the learned parameters in the item-item matrix.

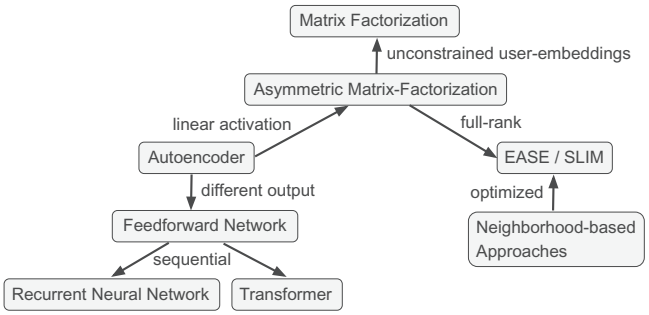

FIGURE 2 (Simplified) relationships among several models discussed in this article

A deep feedforward model may be viewed as a generalization of the autoencoder: now the input vector does not need to be the same as the output vector of the network. For instance, given a multi-hot encoding of the user's play-history in the input vector (as a bag of videos), a reasonable target of the feedforward model is to predict the most recent video that the user played (instead of predicting all the videos in the user's play-history, like in an autoencoder). While such a restriction of the target-vector may reduce the amount of available training-data, the benefit is that the model can learn the current interests of the users (beyond just their long-term interests), which also allows the model to account for some popularity trends.

Through the lens of autoencoders, we can see that several seemingly unrelated models commonly used in recommender systems are in fact closely connected. This can help us better design recommender systems for specific needs by flexibly combining different models. It also provides insights on how to further improve different models. This includes using different techniques for regularization, loss functions, activation functions, or optimization procedures. See Figure 2 for a visual reference

explaining relationships between various models, where we also included the sequential models discussed in the next section.

Sequential models

While the bag-of-items view is convenient, there is useful information in the sequential nature of a user's play-history that gets removed in a bag-of-items model. To accommodate this, many of the deep sequential models originally developed for NLP tasks can be adapted to the recommendation task, for example Hidasi et al. (2015). Instead of predicting the next word in a sentence, the recommendation model aims to predict the next item the user will interact with. Sequential methods are very effective in e-commerce applications for session-based recommendations: there, the lack of a clear user-identifier means that the only known information is the sequence of items visited so far within the user's session. Several models were found to show good performance (Hidasi et al. 2015; Quadrana et al. 2017; Zhang et al. 2019) on such tasks.

While we found a bag-of-items approach to be sufficient for some of the recommendation tasks on the Netflix homepage, certain tasks clearly benefited from using sequential models. We have experimented with various sequential models in recent years, including simple n-gram models, recurrent neural networks like LSTM (long short term memory) (Hochreiter and Schmidhuber 1997) or GRU (Cho et al. 2014), as well as transformer architectures like BERT (Devlin et al. 2018). While transformers are not (strictly speaking) sequential models, their attention mechanism has similar effects. Besides improving recommendation accuracy, the attention mechanism also provides a new and interesting way of determining an explanation for each recommended video.

Adding heterogeneous features

So far we have mostly described "traditional" recommendation-tasks where only user-item interaction-data are used. An interesting empirical finding in the literature and also in our production system is that, if they are well tuned, rather shallow model-architectures (typically with one to three hidden layers) achieve the best prediction accuracy in these "traditional" recommendation tasks. This is a key difference to other deep-learning application-areas, like image classification, where major improvements have been achieved by using much deeper architectures. While the full range of low-level to high-level features has to be learnt in image classification problems (e.g., from pixels to objects), in a traditional recommendation setup, the item-identifier are already provided in the data: in our case, video-IDs that a user has watched. This makes the need for learning various low-level features obsolete. Because this recommendation setup can be seen as learning the pairwise interaction of two categorical features (users and items), it becomes a pure representation-learning task, where the embeddings for both the user and item are learned. While it is possible for a deep network to learn from data how to interact those two embeddings with each other, it is much more efficient to multiply them directly, instead of requiring various layers in the network to learn basic operations like a dot product of vectors, which requires a much deeper architecture.

After we had struggled to obtain significant improvements in recommendation accuracy by employing deep learning compared to well-tuned simpler models on "traditional" recommendation tasks, we took a step back to think about what deep learning could potentially enable us to do, and what was difficult with traditional models. We hypothesized that deep models could be particularly effective in making use of extra information beyond a user's implicit or explicit feedback. Indeed, when we carefully added additional heterogeneous information, we obtained large gains in recommendation accuracy.

In the following, we use the example of leveraging *time* as an additional feature to illustrate the power of deep learning and representation learning. First, note that time is not easy to represent within linear models as it carries several levels of cyclic and seasonal information. For instance, the seasonal effects in the movie-domain include measurable changes in taste-patterns as a function of the time of the day (e.g., more videos for children in the afternoon), as well as the day of the week (e.g., TV shows vs. movies), and holidays (e.g., horror movies leading up to Halloween).

In the recommender-systems literature, it has been shown that incorporating time into models increases model performance (Koren 2009). A common practice in Matrix Factorization, and also more recently in deep learning is to represent time by discretizing it. The discretization is usually done across different dimensions such as day or week (Hansen et al. 2020). While discretizing can help pick up the cyclical elements of time, the embedding-oriented approaches have the down-side that, while time-embeddings for the past can be learned, it is hard to do so for the future where the model is actually to be used (unless you have a flux capacitor). To address this, in one of our sequential models we observed large improvements when we used raw, continuous timestamps indicating the time when the user played a video in the past, along with the current time when making a prediction (see also (Covington, Adams, and Sargin 2016) for a similar

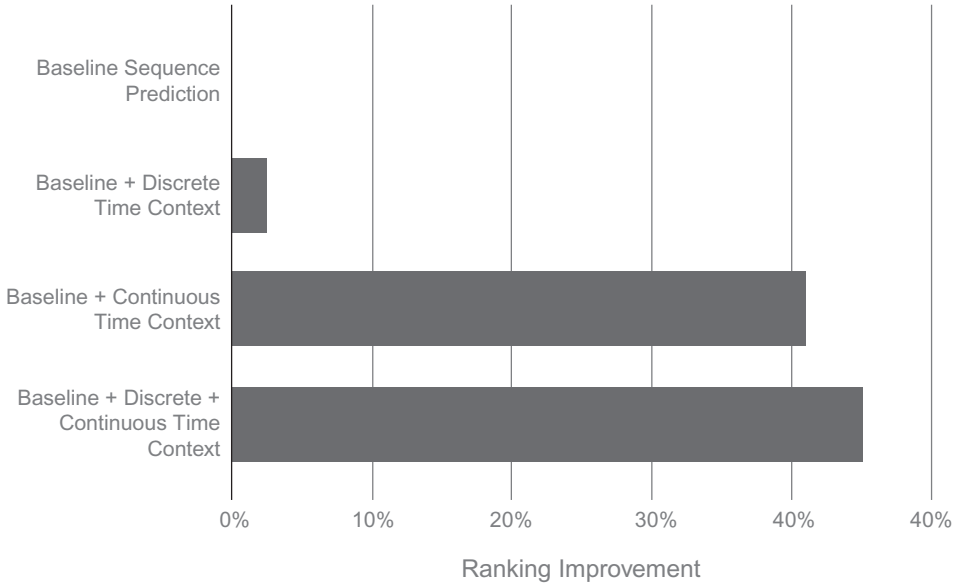

FIGURE 3 Ranking-improvements when adding time as context

approach developed at about the same time). Figure 3 shows an improvement by more than 30 percentage points in offline metrics when using continuous-time features versus discretized time. This experiment exemplifies deep-learning's power for representation learning: first, there is no need to define an arbitrary discretization-schema, which can be challenging when dealing with behaviors from people in the various time-zones around the world, as well as with different interaction-frequencies. Second, a simple representation of the data also allows the model to learn and extrapolate trends.

However, as usual with deep learning, explainability of the model becomes more challenging. Moreover, when using raw timestamps, the production algorithm will always use the current timestamp, which was never observed in the training set. It is an open question on how far ahead in time these methods can extrapolate and how fast the performance of the production system degrades without being retrained. For this reason, we ensure that such models are regularly retrained to prevent the need to extrapolate very far beyond the training set.

Apart from incorporating additional context or metadata features, personalization can benefit from using computer vision and other content understanding techniques (audio, video, text) to learn latent representations directly from the item. Such approaches are motivated by considerable improvements when used on still images (Chen et al. 2020a; 2020b; He et al. 2020a; Misra and van der Maaten 2020). Computer vision has also been used to improve recommendations in other application areas, for example, in fashion (Hsiao and Grauman 2020), while using the audio signal of songs can improve music-recommendations (van den Oord, Dieleman, and Schrauwen 2013).

In simple models, we found it difficult to integrate these types of complex features. For instance, Matchbox (Stern, Herbrich, and Graepel 2009) or Factorization Machines (Rendle 2010) are bi-linear models that may be viewed as extensions of matrix factorization to allow for additional features (besides item-IDs and user-IDs). Given the bi-linear nature of these models, they are naturally unable to learn higher-order interactions among several of the heterogeneous input features. Even though higher-order interactions can be directly provided as input, this requires substantial feature engineering and careful feature selection. In contrast, deep nonlinear models are able to learn the relevant higher-order interactions in an 'end-to-end' fashion to produce considerable gains in recommendation-accuracy.

PRACTICAL CHALLENGES

Using deep learning in industry brings forward many challenges and learnings that makes it different from an academic setting. Below, we discuss a few of those.

Mismatch in offline and online settings

A common problem when developing recommender systems is that offline performance (when evaluated on held-out historical data) is not reflective of online performance (when evaluated in an A/B-test where the recommendations are presented to users). We observed this acutely when trying deep-learning models. In some cases we saw significant offline gains over other methods. When tested online, however, these gains would sometimes disappear

or, in rare cases, result in worse performance. This is generally a manifestation of correlational performance not being indicative of causal behavior. But why would this problem be exacerbated in the case of deep-learning models? We would like to highlight a few observations in the following.

Sensitivity to objective functions and proxy metrics: The problem formulation in recommender systems tends to be in the form of some (short-term) proxy metrics (e.g., clicks or plays), whereas the actual quantity we want to optimize is subjective and hard to measure, like long-term user satisfaction. The connection between these proxy metrics and the actual metric of interest may be quite complicated and may even break in certain ranges (or while extrapolating). Therefore, if a machine-learning model overfits to these short-term metrics, it may possibly deviate far away from the actual metric of interest. Even when simpler models do not show such deviations, more powerful models, such as deep-learning models can break this connection. Simply speaking, if a deep-learning model is given the wrong problem to solve, it will solve it more accurately than less powerful models would. We hence need ways of training on short-term behavior (e.g., clicks or plays) to optimize long-term behaviors. An additional challenge is that short-term actions can be quite noisy in the sense that subtle changes in the definition of the (short-term) training-objective can lead to big changes in the produced recommendations.

Distribution mismatch: This is in general true whenever machine learning models are deployed in the real world, that is, the data which are used to train machine-learning models are not reflective of the population for which the model will be used. Covariate shift is a concrete example of distribution mismatch in which the distribution of input features is different between the training data and the real world. Traditional techniques to fix distribution mismatch like importance sampling have been shown to be less effective with powerful deep-learning models (Byrd and Lipton 2019). These problems are well known in the recommender-system literature and are active areas of research, for example (Wang et al. 2020).

Fairness and explainability: This is another very important aspect, and it is gaining a lot of attention recently (Caton and Haas 2020). When a deep-learning model (or any machine-learning model) is deployed, we need to be careful of how it may treat real-world entities (in the case of Netflix, members and videos for example), and whether there are any unintentional biases that cause the model to treat some entities in an unfair way. It is again related to the issue of offline–online mismatch as it may not be possible to easily evaluate a model from a fairness perspective as we may not have the appropriate offline evaluation-data. A simple example is a model doing well for the majority of the data and poorly on a minority. Increasingly, explainability techniques are being employed to evaluate the fairness of the model (Tan et al. 2018). Deep-learning models are complex and hard to explain. We found techniques like LIME (Ribeiro, Singh, and Guestrin 2016), SHAP (Lundberg and Lee 2017) and Integrated Gradients (Sundararajan, Taly, and Yan 2017) to be particularly helpful in explaining deep-learning models.

Overcoming all of these problems in the Netflix setting meant iterative tweaking and refinement of both the metrics and the deep-learning models through a series of online tests and offline analyses. In the process, we not only improved our models, but also our metrics. The latter also enabled us to better evaluate new recommendation methods in the future. Examples of these refinements include off-policy evaluation (Bibaut et al. 2019; Su et al. 2020; Vlassis et al. 2019; Vlassis, Gil, and Chandrashekar 2021), and the use of bandits (Li et al. 2010) or reinforcement-learning (He et al. 2020b) techniques. Contextual bandit techniques in particular are able to break the feedback loop and remove various biases (e.g., Wang et al. 2020) in the data by introducing some amount of randomness into the recommendations. With bandit algorithms, we can continuously gather cleaner training-data by keeping track of the propensities for the shown recommendations. Even though the user-experience may be occasionally slightly degraded in the short-term due to this randomization, it helps improve the quality of recommendations in the long-term. We found these approaches very effective in our online tests where the careful design of the exploration approach meant the initial impact of some randomness can be within the noise-floor of the algorithm.

A complementary approach to exploration is to use the fact that there are different ways of discovering videos on the Netflix service. For instance, if a video gets recommended to a member, there is no need for the member to search for it. In contrast, if a video or category of videos is not recommended to a member, it may trigger the member to search for it. Hence, the feedback loop can be partially broken by training the recommender system not only on the videos that were discovered from pages of recommendations, but also on the videos found via search (and analogously for a search algorithm). The advantage of this approach is that it does not require any randomization of the displayed videos, and hence does not result in any short-term degradation of the recommendations shown to the user. The disadvantage of this approach obviously is that it is difficult to quantify to what degree the feedback loop was broken, and the importance of the different data sources has to be carefully tuned in the training data. Nevertheless, we found this approach to be an effective component for (partially) breaking the feedback loop, as it comes at no cost/degradation of the user-experience. Of course,

this approach is only applicable in recommendation tasks where there are several ways for a user to discover items.

Aspects of the industrial setting

There are multiple considerations to balance when developing and deploying machine learning algorithms at a scale of serving hundreds of millions of users. A machine-learning model is analogous to a kernel in an operating system in the sense that a lot of work is put into developing the surrounding ecosystem beyond the model itself, so that the model works well. Therefore, it is important to consider practical aspects of the system when considering deep learning.

Deep-learning software and hardware infrastructure: In the last couple of decades, neural networks have been winning the hardware and software lottery (Hooker 2020). This means that in recent years we have seen plenty of advances in hardware and software specifically suited for training and deploying deep neural networks. An industrial-strength machine-learning development and deployment ecosystem is important for successful continuous innovation. Well-supported ecosystems save time for the researchers and engineers and provide a de-facto standard for model interoperability. In fact, from a practical perspective, one of the great benefits of using deep-learning approaches for recommendation is being able to use deep-learning frameworks such as TensorFlow (Abadi and others 2015), Keras (Chollet and others 2015), or PyTorch (Paszke and others 2019) for recommendation problems. In particular, the automatic differentiation and tensor-oriented computation of such tools work well even for simpler models like Matrix Factorization. These frameworks enable simple scaling of training and inference by using specialized hardware such as GPU or TPU. It also becomes easy to train models on large amounts of data that would not fit in the main memory of a machine. Built-in monitoring-infrastructure enables faster debugging. At Netflix we had a lot of machine-learning libraries that were developed internally. Usually, each custom model had to implement its own monitoring and interfaces to work with the larger ecosystem. Such customization has its costs in support, speed of development, and deployment. The above mentioned libraries yield unparalleled standardization and efficiency, even if used to train a Matrix Factorization model. Such de-facto standardization also enables faster on-boarding of new team members.

Extendability of deep-learning models to new attributes and objectives: Another advantage of using neural-network models in recommender systems is their flexibility. As discussed earlier, it is easy to extend the model to include richer sources of data. A lot of custom-work was done in the past to extend Matrix Factorization to include time as a feature (Koren 2009), or to add contextual information (Karatzoglou et al. 2010). Neural networks make these extensions as straightforward as adding new features to supervised models. It also allows one to implement many other simple extensions of Matrix Factorization such as specialized loss functions (e.g., Shi et al. 2012) or various training/optimization techniques (e.g., Kingma and Ba 2015). Moreover, deployment of Matrix Factorization methods rewritten as a neural network can be much simpler. While a Matrix Factorization model would need to do fold-in as described earlier, a neural network performs analogous fold-in as a standard step of inference.

Ease of integration of deep-learning models in the existing ecosystem: One of the benefits of deep learning software is its adaptability with popular software stacks, for example the Java Virtual Machine. Before tools like Tensorflow, a researcher would either implement their own deep-learning model in a software stack that may not be compatible with the production stack. Because of this mismatch, it took considerable effort to port our early deep-learning models to run in our production stack, which made iterating on them cumbersome. Adaptability of modern deep-learning software with many popular software stacks offered a big advantage and the iteration speed between offline research and online usage accelerated significantly.

Scalability of model training and prediction/inference: Scaling deep-learning model-training not only depends on a specialized software/hardware stack but also on the characteristics of the datasets. For example, consider multiclass classification, which is a very important task in recommendation systems (Elahi et al. 2019; Liang et al. 2018). Multiclass classification is usually done by using a softmax activation in the output-layer of deep-learning models. In language modeling or other domains with a large number of items/labels, the softmax computation is the main bottleneck in scaling such systems. This is not the case for recommender systems at Netflix. The dataset used in Netflix recommender-systems typically deals with a medium-sized item set and hundreds of millions of members. Such settings perfectly fit many deep-learning architectures that use the softmax output-layer to make final predictions. Moreover, given the relatively sparse nature of the user-item interactions, parallel stochastic gradient descent (Recht et al. 2011) works very well for large-scale training of deep-learning models. These aspects enable us to use complex architectures successfully on Netflix datasets. In fact, even with complex architectures and efficient data formats, the training is often IO bound and not CPU/GPU bound, in contrast to typical problems with a similar dataset size in image or language learning. At Netflix, we were able to train

neural-network models (due to the available toolboxes) on much more data than other machine learning models. While using increased amounts of data with specialized hardware can potentially increase training costs, we found in our online experiments that the observed benefits easily outweighed the costs.

CONCLUSIONS

Making deep-learning models successful and widely adopted within the Netflix recommender system provided valuable learnings in machine-learning research and engineering. In particular, solving existing problems (e.g., traditional recommender systems) with a new approach (deep learning) sometimes only has limited benefits. In fact, well-tuned traditional methods are very strong baselines when only user-item interaction data are used. Instead, deep learning can effectively solve new problems that were considered very challenging for traditional methods, such as finding good representations for the time domain, or extending the range and modalities of inputs considered such as images, text, and videos. Applying these techniques beyond traditional framings of the recommendation problem opens the door to considerable improvements.

On the other hand, the use of powerful deep-learning models can also amplify weaknesses in recommendation systems, for example overfitting on short-term proxy objectives that may be misaligned with longer-term objectives, such as user satisfaction. Finding ways to better encode these long-term objectives, as well as controlled experiments to measure long-term user satisfaction, are key.

Another positive side-effect of using deep learning is the superior machine-learning software stack. It enables faster model training, implementation, deployment, debugging and better support of existing infrastructure.

Many deep-learning models originally developed in other areas of machine learning, like NLP, have been successfully adapted to the domain of recommender systems. While such cross-pollination will likely continue, we also expect the development of new approaches that are even more specific to the properties of the available data and to the various recommendation tasks.

ACKNOWLEDGEMENTS

All this work on deep learning to improve the Netflix recommender system would have not been possible without Ko-Jen Hsiao, Jingu Kim, Anoop Deoras, Maryam Esmaeili, Alex Chen, David Seetapun, Aish Fenton, and numerous other (current and former) colleagues at Netflix.

REFERENCES

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, et al. 2015. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems." http://tensorflow.org/.

Aiolli, F. 2013. "Efficient Top-N Recommendation for Very Large Scale Binary Rated Datasets." In *ACM Conference on Recommender Systems (RecSys)*.

Alvino, C., and J. Basilico. 2015. "Learning a Personalized Homepage." In *Netflix Tech Blog*. http://techblog.netflix.com/2015/04/learning-personalized-homepage.html.

Bennet, J., and S. Lanning. 2007. "The Netflix Prize." In *Workshop at SIGKDD-07, ACM Conference on Knowledge Discovery and Data Mining*.

Bibaut, A., I. Malenica, N. Vlassis, and M. van der Laan. 2019. "More Efficient Off-Policy Evaluation through Regularized Targeted Learning." In *International Conference on Machine Learning (ICML)*.

Byrd, J., and Z. Lipton. 2019. "What is the Effect of Importance Weighting in Deep Learning?." In *International Conference on Machine Learning (ICML)*.

Caton, S., and C. Haas. 2020. "Fairness in Machine Learning: A Survey."

Chaney, A. J. B., B. M. Stewart, and B. E. Engelhardt. 2018. "How Algorithmic Confounding in Recommendation Systems Increases Homogeneity and Decreases Utility." In *ACM Conference on Recommender Systems (RecSys)*.

Chen, T., S. Kornblith, M. Norouzi, and G. Hinton. 2020a. "A Simple Framework for Contrastive Learning of Visual Representations." arXiv:2002.05709.

Chen, T., S. Kornblith, K. Swersky, M. Norouzi, and G. Hinton. 2020b. "Big Self-Supervised Models are Strong Semi-Supervised Learners." arXiv:2006.10029.

Cho, K., B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. 2014. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation." In *Empirical Methods in Natural Language Processing (EMNLP)*.

Chollet, F., E. Bursztein, Q.S. Zhu, H. Jin, and A. Eldeeb. 2015. "Keras." https://keras.io.

Covington, P., J. Adams, and E. Sargin. 2016. "Deep Neural Networks for YouTube Recommendations." In *ACM Conference on Recommender Systems (RecSys)*.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova. 2018. "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding."*arXiv preprint arXiv:1810.04805*.

Elahi, E., W. Wang, D. Ray, A. Fenton, and T. Jebara. 2019. "Variational Low Rank Multinomials for Collaborative Filtering with Side-Information." In *ACM Conference on Recommender Systems (RecSys)*.

Ferrari Dacrema, M., P. Cremonesi, and D. Jannach. 2019. "Are We Really Making much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches." In *ACM Conference on Recommender Systems (RecSys)*.

Gershman, S., and N. Goodman. 2014. "Amortized Inference in Probabilistic Reasoning." In *Annual Conference of the Cognitive Science Society*.

Gomez-Uribe, C. A., and N. Hunt. 2015. "The Netflix Recommender System: Algorithms, Business Value, and Innovation."

ACM Transactions on Management Information Systems 6(4): Article 13.

Hansen, C., C. Hansen, L. Maystre, R. Mehrotra, B. Brost, F. Tomasi, and M. Lalmas. 2020. "Contextual and Sequential User Embeddings for Large-Scale Music Recommendation." In *ACM Conference on Recommender Systems (RecSys)*.

He, X., B. An, Y. Li, H. Chen, R. Wang, X. Wang, R. Yu, X. Li, and Z. Wang. 2020b. "Learning to Collaborate in Multi-Module Recommendation via Multi-Agent Reinforcement Learning without Communication." In *ACM Conference on Recommender Systems (RecSys)*.

He, K., H. Fan, Y. Wu, S. Xie, and R. Girshick. 2020a. "Momentum Contrast for Unsupervised Visual Representation Learning." In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hernández-Lobato, J. M., N. Houlsby, and Z. Ghahramani. 2014. "Probabilistic Matrix Factorization with Non-Random Missing Data." In *International Conference on Machine Learning (ICML)*.

Hidasi, B., A. Karatzoglou, L. Baltrunas, and D. Tikk. 2015. "Session-Based Recommendations with Recurrent Neural Networks." arXiv:1511.06939.

Hinton, G. E., S. Osindero, and Y.-W. Teh. 2006. "A Fast Learning Algorithm for Deep Belief Nets." *Neural Computation* 18.

Hinton, G. E., and R. R. Salakhutdinov. 2006. "Reducing the Dimensionality of Data with Neural Networks." *Science* 313.

Hochreiter, S., and J. Schmidhuber. 1997. "Long Short-Term Memory." *Neural Computation* 9(8): 17351780.

Hooker, S. 2020. "The Hardware Lottery." arXiv:2009.06489.

Hsiao, W.-L., and K. Grauman. 2020. "ViBE: Dressing for Diverse Body Shapes." In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jahrer, M., A. Toscher, and R. Legenstein. 2010. "Combining Predictions for Accurate Recommender Systems." In *ACM Conference on Knowledge Discovery and Data Mining*.

Karatzoglou, A., X. Amatriain, L. Baltrunas, and N. Oliver. 2010. "Multiverse Recommendation: N-Dimensional Tensor Factorization for Context-Aware Collaborative Filtering." In *ACM Conference on Recommender Systems (RecSys)*.

Kingma, D. P., and J. Ba. 2015. "Adam: A Method for Stochastic Optimization." In *Int. Conference on Learning Representations (ICLR)*.

Kingma, D. P., and M. Welling. 2013. "Auto-Encoding Variational Bayes." In *Int. Conference on Learning Representations (ICLR)*.

Koren, Y. 2008. "Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model." In *ACM Conference on Knowledge Discovery and Data Mining*, 426–34.

Koren, Y. 2009. "Collaborative Filtering with Temporal Dynamics." In *ACM Conference on Knowledge Discovery and Data Mining*.

Lamkhede, S., and S. Das. 2019. "Challenges in Search on Streaming Services: Netflix Case Study." In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*.

Li, L., W. Chu, J. C. Langford, and R. E. Schapire. 2010. "A Contextual-Bandit Approach to Personalized News Article Recommendation." In *International World Wide Web Conference (WWW)*.

Liang, D., L. Charlin, J. McInerney, and D. Blei. 2016. "Modeling User Exposure in Recommendation." In *International World Wide Web Conference (WebConf)*.

Liang, D., R. G. Krishnan, M. D. Hoffman, and T. Jebara. 2018. "Variational Autoencoders for Collaborative Filtering." In *International World Wide Web Conference (WebConf)*. https://github.com/dawenl/vae_cf.

Lundberg, S. M., and S.-I. Lee. 2017. "A Unified Approach to Interpreting Model Predictions." In *Advances in Neural Information Processing Systems (NeurIPS)*.

Marlin, B., and R. Zemel. 2009. "Collaborative Prediction and Ranking with Non-Random Missing Data." In *ACM Conference on Recommender Systems (RecSys)*.

Marlin, B., R. Zemel, and S. Roweis. 2005. "Unsupervised Learning with Non-Ignorable Missing Data." In *Workshop on Artificial Intelligence and Statistics (AISTATS)*.

Marlin, B., R. Zemel, S. Roweis, and M. Slaney. 2007. "Collaborative Filtering and the Missing At Random Assumption." In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Misra, I., and L. van der Maaten. 2020. "Self-Supervised Learning of Pretext-Invariant Representations." In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ning, X., and G. Karypis. 2011. "SLIM: Sparse Linear Methods for Top-N Recommender Systems." In *IEEE International Conference on Data Mining (ICDM)*, 497–506.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, et al. 2019. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In *Advances in Neural Information Processing Systems (NeurIPS)*.

Paterek, A. 2007. "Improving Regularized Singular Value Decomposition for Collaborative Filtering." In *KDDCup*.

Quadrana, M., A. Karatzoglou, B. Hidasi, and P. Cremonesi. 2017. "Personalizing Session-Based Recommendations with Hierarchical Recurrent Neural Networks." In *ACM Conference on Recommender Systems (RecSys)*.

Recht, B., C. Re, S. Wright, and F. Niu. 2011. "Hogwild!: A Lock-free Approach to Parallelizing Stochastic Gradient Descent." In *Advances in Neural Information Processing Systems (NeurIPS)*.

Rendle, S. 2010. "Factorization Machines." In *IEEE International Conference on Data Mining (ICDM)*, 995–1000.

Rezende, D. J., M.S., and W.D. 2014. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models." In *International Conference on Machine Learning (ICML)*.

Ribeiro, M. T., S. Singh, and C. Guestrin. 2016. "Why should I Trust You? Explaining the Predictions of any Classifier." In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*.

Rifai, S., P. Vincent, X. Muller, X. Glorot, and Y. Bengio. 2011. "Contractive Auto-Encoders: Explicit Invariance during Feature Extraction." In *International Conference on Machine Learning (ICML)*.

Salakhutdinov, R., A. Mnih, and G. Hinton. 2007. "Restricted Boltzmann Machines for Collaborative Filtering." In *International Conference on Machine Learning (ICML)*.

Shi, Y., A. Karatzoglou, L. Baltrunas, M. Larson, N. Oliver, and A. Hanjalic. 2012. "CLiMF: Learning to Maximize Reciprocal Rank with Collaborative Less-is-More Filtering." In *ACM Conference on Recommender Systems (RecSys)*.

Steck, H. 2010. "Training and Testing of Recommender Systems on Data Missing Not At Random." In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 713–22.

Steck, H. 2019. "Embarrassingly Shallow Autoencoders for Sparse Data." In *International World Wide Web Conference (WWW)*.

Stern, D., R. Herbrich, and T. Graepel. 2009. "Matchbox: Large Scale Bayesian Recommendations." In *International World Wide Web Conference (WWW)*.

Su, Y., M. Dimakopoulou, A. Krishnamurthy, and M. Dudik. 2020. "Doubly Robust Off-Policy Evaluation with Shrinkage." In *International Conference on Machine Learning (ICML)*.

Sundararajan, M., A. Taly, and Q. Yan. 2017. "Axiomatic Attribution for Deep Networks." In *International Conference on Machine Learning (ICML)*.

Tan, S., R. Caruana, G. Hooker, and Y. Lou. 2018. "Distill-and-Compare: Auditing Black-Box Models using Transparent Model Distillation." In *AAAI/ACM Conference on AI, Ethics, and Society*.

Tu, Z., Y. Liu, L. Shang, X. Liu, and H. Li. 2017. "Neural Machine Translation with Reconstruction." *AAAI Conference on Artificial Intelligence*.

van den Oord, A., S. Dieleman, and B. Schrauwen. 2013. "Deep Content-Based Music Recommendation." In *Advances in Neural Information Processing Systems (NeurIPS)*.

Verstrepen, K., and B. Goethals. 2014. "Unifying Nearest Neighbors Collaborative Filtering." In *ACM Conference on Recommender Systems (RecSys)*.

Vlassis, N., A. Bibaut, M. Dimakopoulou, and T. Jebara. 2019. "On the Design of Estimators for Bandit Off-Policy Evaluation." In *International Conference on Machine Learning (ICML)*.

Vlassis, N., F. Gil, and A. Chandrashekar. 2021. "Off-Policy Evaluation of Slate Policies under Bayes Risk." arXiv:2101.02553.

Volkovs, M. N., and G. W. Yu. 2015. "Effective Latent Models for Binary Feedback in Recommender Systems." In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*.

Wang, Y., D. Liang, L. Charlin, and D. Blei. 2020. "Causal Inference for Recommender Systems." In *ACM Conference on Recommender Systems (RecSys)*.

Wu, Y., C. DuBois, A. X. Zheng, and M. Ester. 2016. "Collaborative Denoising Auto-Encoders for Top-N Recommender Systems." In *ACM Conference on Web Search and Data Mining (WSDM)*.

Zhang, S., L. Yao, A. Sun, and Y. Tay. 2019. "Deep Learning Based Recommender System: A Survey and New Perspectives." *ACM Computing Surveys* 52.

AUTHOR BIOGRAPHIES

Harald Steck is a research scientist at Netflix. He has over twenty years of experience in machine learning, and over the past decade has focused on collaborative-filtering approaches. He holds a Ph.D. in Computer Science from the Technical University of Munich.

Linas Baltrunas is a research scientist at Netflix, working on page construction algorithms. His research interests include algorithmic user experience adaptation, context aware recommender systems and session based recommendation algorithms. He received a Ph.D. in Computer Science from the University of Bolzano where he worked on context aware recommender systems.

Ehtsham Elahi is a machine learning researcher at Netflix. His areas of interest span probabilistic modeling, approximate Bayesian inference, reinforcement learning and model interpretability. Applications of his work include personalized ranking, slate modeling and optimization and recommender systems' transparency and explainability.

Dawen Liang is a research scientist at Netflix, working on core personalization algorithms. His research interests include probabilistic models and (approximate) inference, as well as causal inference, and their applications to recommender systems. He completed his Ph.D. in the Electrical Engineering Department at Columbia University, working on probabilistic latent variable models for analyzing music, speech, text, and user behavior data.

Yves Raimond is a Director of Machine Learning at Netflix, where he leads a mixed team of researchers and engineers building the next generation of Machine Learning algorithms used to drive the Netflix experience. Before that, he was a Lead Research Engineer in BBC R&D, working on information extraction from Multimedia content. He holds a PhD in Music Information Retrieval from Queen Mary, University of London.

Justin Basilico is a research/engineering director at Netflix. He leads an applied research team that creates the algorithms used to personalize the Netflix homepage through machine learning, recommender systems, and large-scale software engineering. Prior to Netflix, he worked in the Cognitive Systems group at Sandia National Laboratories. He has an MS in Computer Science from Brown University and a BA in Computer Science from Pomona College.

How to cite this article: Steck, H., L. Baltrunas, E. Elahi, D. Liang, Y. Raimond, and J. Basilico. 2021. Deep learning for recommender systems: A netflix case study. *AI Magazine* 42: 7–18. https://doi.org/10.1609/aaai.12013.