Parallel Recurrent Neural Network Architectures for Feature-rich Session-based Recommendations

Balázs Hidasi
Gravity R&D
Budapest, Hungary
balazs.hidasi@gravityrd.com

Massimo Quadrana *
Politecnico di Milano
Milan, Italy
massimo.quadrana@polimi.it

Alexandros Karatzoglou
Telefonica Research
Barcelona, Spain
alexk@tid.es

Domonkos Tikk
Gravity R&D
Budapest, Hungary
domonkos.tikk@gravityrd.com

ABSTRACT

Real-life recommender systems often face the daunting task of providing recommendations based only on the clicks of a user session. Methods that rely on user profiles – such as matrix factorization – perform very poorly in this setting, thus item-to-item recommendations are used most of the time. However the items typically have rich feature representations such as pictures and text descriptions that can be used to model the sessions. Here we investigate how these features can be exploited in Recurrent Neural Network based session models using deep learning. We show that obvious approaches do not leverage these data sources. We thus introduce a number of parallel RNN (p-RNN) architectures to model sessions based on the clicks and the features (images and text) of the clicked items. We also propose alternative training strategies for p-RNNs that suit them better than standard training. We show that p-RNN architectures with proper training have significant performance improvements over feature-less session models while all session-based models outperform the item-to-item type baseline.

Keywords

deep learning; recurrent neural networks; gated recurrent units; recommender systems; training strategies

1. INTRODUCTION

In traditional recommender systems algorithms it is often assumed that user history logs (e.g. clicks, purchases or views) are available. We show that this assumption does not hold in many real-world recommendation use cases: (1) many e-commerce sites do not require user authentication even for purchase; (2) in video streaming services users also rarely log in; (3) many sites have a small percentage of returning users; (4) or the user intent can change between different sessions, typical for e.g. classified sites. User tracking can partly solve the user identification problem across sessions (e.g. through fingerprinting technology, cookies), but this is often unreliable and also raises privacy concerns. So, in such cases the typical solution is to resort to item-to-item recommendations. Here we investigate how we can exploit session data to improve recommendations.

Given the absence of user profiles, it is vital to draw as much information as possible from the session clicks. Besides the click-stream data of the session (clicked item-IDs), one can also take into account the characteristics of the items that have been clicked. Items often come with rich feature representation such as detailed text description or (thumbnail) image. One can expect that e.g. users shopping for a particular type of item will click on items that have similar text descriptions and similar visual features. Images, text or potentially even richer features such as animated gifs are eventually what the user sees and what will determine the appeal of an item to a user. Features become particularly important in a session modeling setting where historical user specific data is missing or has no importance. Such features should be utilized to aid the session modeling process. Item features are also a very good way to deal with item cold start i.e. when a new item enters the pool of selectable items. The joint modeling of the sequence of clicked item-IDs and item features poses some interesting problems.

The algorithms in this work utilize deep learning techniques both to extract high quality features from visual information and to model the sessions. (We also extract features from text via bag-of-words.) Individual user sessions can be seen as sequence of clicks. We use Recurrent Neural Networks (RNNs) to model the session data. RNNs have been shown to perform excellent in modeling sequence data [15]. We introduce a number of parallel[1] RNN (p-RNN) ar-

*The author worked at Telefonica Research as an intern during the time of this research.

RecSys '16, September 15-19, 2016, Boston , MA, USA

DOI: http://dx.doi.org/10.1145/2959100.2959167

[1]Note that we use the term 'parallel' to indicate that for each aspect/feature of the clicked item (e.g. the item-ID, text description, image features) there is a separate RNN processing the input. 'Parallel' refers to the fact that we have multiple RNNs rather than distributed processing of the data or the algorithm computations. While distributed

chitectures to model the session clicks concurrently with the features (text or images) of the clicked items.

Instead of using a single RNN where all the data is used at the input in a concatenated form, we apply parallel architectures because of the very different nature of the data: image features tend to be much denser than the one-hot representation of the item-ID or the bag-of-words representation of text. Parallel training also allows us to fine-tune the individual networks with respect to their hyper-parameters while also maintaining a connection between the networks through shared parameters and optimization. We introduce 3 different architectures of p-RNNs that combine the ID-click data with the features of the clicked items. The architectures vary with regards to the shared model parameters and interactions of the hidden states (Section 3).

We also point out that training p-RNNs is not trivial. Standard simultaneous training can waste the capacity of the network, because different parts of the same network may learn the same relations from the data. Therefore we devise 3 alternative optimization procedures to train p-RNNs. We thoroughly evaluate the proposed p-RNN architectures on (1) video recommendations using the thumbnail and (2) product recommendations using text description on a classified site. We compare against the industry standard session-based solution, that is today the item-kNN.

2. RELATED WORK

Deep neural networks have been employed with tremendous success in image recognition, segmentation and retrieval tasks. Deep learning techniques have been used in tasks such as image and speech recognition [5, 10, 20] where unstructured data is processed through several convolutional and standard layers of (usually rectified linear) units.

Neural models for RecSys. Most of the work on deep models and recommendations focus on the classical collaborative filtering (CF) user–item setting. Restricted Boltzmann Machines (RBM) were one of the first neural networks to be used for classical CF and recommender systems [22]. More recently denoising autoencoders have been used to perform CF in a similar manner [30]. Deep networks have also been used in cross-domain recommendation whereby items are mapped to a joint latent space using deep neural networks [4].

Recurrent Neural Models. RNNs are the deep models of choice when dealing with sequential data (see [15] for a comprehensive review). RNNs have been used in image and video captioning, time series prediction, natural language processing, conversational models, text and music generation, and much more. Long Short-Term Memory (LSTM) [11] networks are a type of RNNs that have been shown to work particularly well, it includes additional gates that regulate when and how much to take the input into account and when to reset the hidden state. This helps with the vanishing gradient problem that often plagues the standard RNN models. Slightly simplified version of LSTM – that still maintains all their properties – are Gated Recurrent Units (GRUs) [2] which we use in this work.

Session-based recommendations. Classical CF methods (e.g. matrix factorization) break down in the session-based setting when no user profile can be constructed from past user behavior. A natural solution to this problem is the item-to-item recommendation approach [24, 14]. In this setting an item-to-item similarity matrix is precomputed from the available session data, items that are often clicked together in sessions are deemed to be similar. These similarities are then used to create recommendations. While simple, this method has been proven to be effective and is widely employed. Though, these methods only take into account the last click of the user, in effect ignoring the information of the previous clicks.

Markov Decision Processes (MDP) [25] are the only other approach that aims to provide recommendations in a session-based manner while taking into account information beyond the last click. MDPs are models of sequential stochastic decision problems. An MDP is defined as a four-tuple $\langle S, A, Rwd, tr \rangle$ where S is the set of states, A is a set of actions Rwd is a reward function and tr is the state-transition function. In recommender systems the set of actions are essentially the recommendations that can be shown. The simplest MPDs boil down to first order Markov chains where the next recommendation can be simply computed through the transition probabilities between items. A major issue with applying Markov chains in the session-based recommendation setting is that the state space quickly becomes unmanageable when trying to include all possible sequences of potential user selections over all items.

The first application of neural models in session-based recommendation used RNNs to model the session data [7]. The recurrent neural network is trained with a ranking loss on a one-hot representation of the session (clicked) item-IDs. The RNN is then used to provide recommendations on new user sessions. This work only focused on the clicked item-IDs while here we aim at modeling a much richer representation of the clicked items.

Feature-rich recommendations. Deep models have been used to extract features from unstructured content such as music or images [3, 26]. In recommender systems these features are then typically used together with more conventional collaborative filtering models. Convolutional deep network have been used to extract features from music files that are then used in a factor model [28]. More recently [29] introduced a more generic approach whereby a deep network is used to extract generic content-features from items, these features are then incorporated in a standard CF model to enhance the recommendation performance. This approach seems to be particularly useful in settings where there is not sufficient user–item interaction information. Image features that have been extracted using convolutional networks have been used in classical matrix factorization-type CF in [6, 16] to enhance the quality of recommendations.

3. P-RNN ARCHITECTURES

In this section we describe the proposed parallel RNN (p-RNN) architectures that utilize item representations (features) for session modeling. A p-RNN consists of multiple RNNs, one for each representation/aspect of the item (e.g. one for ID, one for image and one for text). The hidden states of these networks are merged to produce the score for all items. We also introduce baseline architectures, naive approaches for using the different item representations.

As a basis, we take the best RNN setting from [7]: a single GRU layer without feedforward layers and the TOP1 pairwise loss function along with session-parallel mini-batching. The input of the networks is the item ID of a transaction.

versions of RNNs exist [21] this is not the focus of this work.

The input then translates to either (a) a one-hot ID vector[2]; or (b) a precomputed dense image feature vector; or (c) a sparse unigram + bigram text feature vector. See details on feature extraction in Section 4. The proposed architectures use a subset of the above 3 item representations. The output is a score for every item indicating the likelihood of being the next item in the session. During training scores are compared to a one-hot vector created from the item ID of the next event in the session to compute the loss. In order to reduce computational costs, only the score of the target item and that of a small subset of "negative" items – sampled in each step (see [7]) – are computed during training.

The TOP1 loss is the regularized approximation of the relative rank of the relevant item. The relative rank of the relevant item is given by $\frac{1}{N_S} \cdot \sum_{j=1}^{N_S} I\{\hat{r}_{s,j} > \hat{r}_{s,i}\}$ where N_S is the sample size, $\hat{r}_{s,i}$ is the predicted score of the target item, $\hat{r}_{s,j}$ is the score of negative (other) items in the sample and $I\{\cdot\}$ is approximated with a sigmoid. Optimizing for this loss modifies the parameters so that the score for i is high. This loss though is unstable as some positive items also act as negative examples and so scores tend to become increasingly higher. To avoid this, the scores of the negative examples are pushed towards zero through a regularization term. The loss function thus takes the following form: $L_s = \frac{1}{N_S} \cdot \sum_{j=1}^{N_S} \sigma\left(\hat{r}_{s,j} - \hat{r}_{s,i}\right) + \sigma\left(\hat{r}_{s,j}^2\right)$. Given that a typical item set in recommender systems is in the 100,000s, evaluating this loss over all items is computationally prohibitive. We thus use a sampling procedure whereby the loss is evaluated on a sample of the items. For a given session we use the items in the other sessions of the mini-batch as negative samples. This is computationally cheap and also allows us to get samples from the real distribution of the data.

We devised the following architectures (see Figure 1). Due to limited space, we only present architectures with ID and image features; for text features one can proceed analogously as with image ones. The parallel architectures can also deal with ID, image and text features simultaneously. The architectures can be separated into two groups:

1. Baseline architectures:

ID only: This architecture only uses the one-hot ID vectors and is identical to the one used in [7]. It serves as a baseline in our experiments.

Feature only: The input of this variant is one of the content feature vectors (image or text). Otherwise it works similarly to the previous network.

Concatenated input: The easiest way to combine different item representations is to concatenate them. This network uses the concatenated representations as its input.

2. p-RNN architectures:

Parallel: The first parallel architecture trains one GRU network for each of the representations. Outputs are computed from the concatenation of the hidden layers of the subnets.[3] Training can be done in different ways (see training strategies below).

Parallel shared-W: This architecture differs from the previous one by having a shared hidden to output weight matrix. Scores are not computed for each subnetwork separately. Instead, the weighted sum of the hidden states is multiplied by a single weight matrix to produce the output. Having a shared weight matrix greatly reduces the number of parameters and thus decrease training times and overfitting. This model is also analogous to the pairwise model from context-aware factorization research.[4]

Parallel interaction: In this architecture, the hidden state of the item feature subnet(s) is multiplied by the hidden state of the ID subnet in an element-wise manner before computing the score of the subnet(s). Mixing different aspects of the session to compute item scores is analogous to context-aware preference modeling. For that task [9] found that the *interaction model*, i.e. the sum of the user–item and user–context–item interaction to perform the best. This architecture mimics that model with the ID subnet being promoted to the primary representation of the session. The main difference to the context-aware task is that all of our representations are session representations and not (mostly) independent dimensions. Also note that contrary to the original interaction model, the output weight matrix (item feature matrix) is not shared in our model.

3.1 Training p-RNNs

Training parallel architectures is not trivial. Standard backpropagation across the whole architecture can produce suboptimal results due to different components of the architecture learning the same relations from the data. This can be avoided by pretraining some parts of the network and training the rest afterwards. This cycle can be done several times, motivated by the success of alternating methods like ALS for matrix factorization. Note that while the parameters of fixed networks remain unchanged they still participate in the forward pass and they are only excluded from the backpropagation. We developed the following training strategies for p-RNNs:

Simultaneous: Every parameter of every subnet is trained simultaneously. Serves as the baseline.

Alternating: Subnets are trained in an alternating fashion per epoch. For example, the ID subnet is trained in the first epoch, while the others are fixed; then we fix the ID subnet and train the image subnet for one epoch; and so on. The cycle restarts after each subnet was trained.

Residual: Subnets are trained one after the other, on the residual error of the ensemble of the previously trained subnets. The cycle does not start over, but the individual training of a subnet is longer compared to the alternating method. For example, the ID subnet is trained for 10 epochs, then the image subnet is trained on the residual error of the ID subnet and so on.

Interleaving: Alternating training per mini-batch. For each mini-batch of training examples, the first subnet is trained, the second subnet is trained on the residual error for the current mini-batch and so on. The more frequent alternation allows for a more balanced training across subnets without the drawbacks of the simultaneous training.

4. FEATURE EXTRACTION

In this Section we describe the feature extraction process

[2]Also referred to as 1-of-N encoding. The length of the vector equals to the number of items and only the coordinate corresponding to the ID is 1, the others are 0.

[3]Equivalent to computing output scores separately, weighting them and applying through the final activation function.

[4]The hidden to output matrix is the item feature matrix. The hidden states are different representations of the session, i.e. different context dimensions.

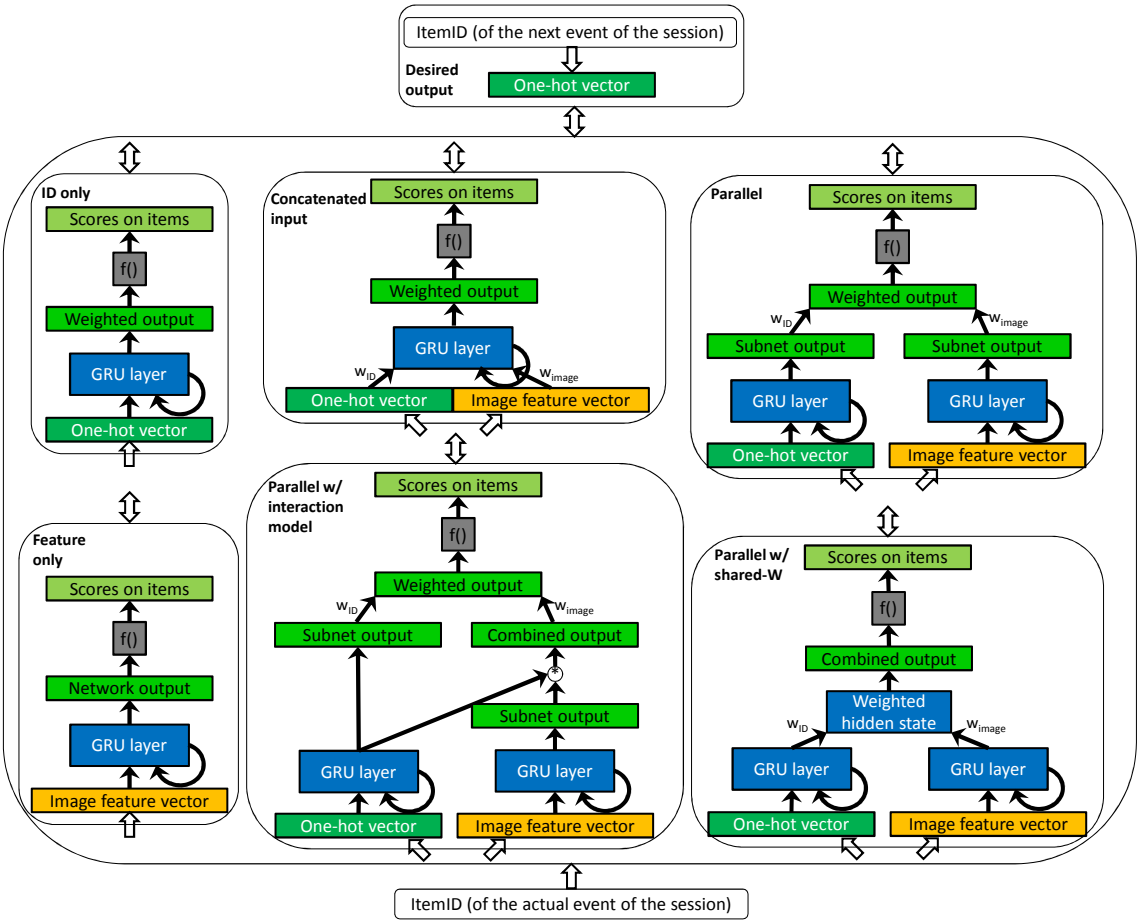

Figure 1: RNN architectures for feature inclusion. Architectures are presented with ID and image features only, but text features can be used in place of the image features as well. The parallel architectures can also be extended to have networks for ID, image and text features simultaneously. f() is the nonlinearity applied to the network output (tanh in our case). *Top row (left to right):* ID only; Concatenated input; Parallel. *Bottom row (left to right):* Feature only; Parallel with interaction model; Parallel with shared weight matrix.

used to create item representations from images and text. Image features were extracted from video thumbnails, while text features are based on product descriptions.

Encoding images. Recently, deep learning research on convolutional neural networks (CNNs) achieved breakthroughs in a variety of different image-related tasks, like object recognition, image segmentation, video classification, etc. [12, 13] even surpassing human performance on the task of object recognition [5]. Unlike other approaches, CNNs don't require prior feature extraction, since they are capable of working on the raw image data. CNNs trained on millions of images produce image features that can then be used as input in other algorithms e.g. clustering [3, 26]. These models generalize well and also perform well on images that the CNN has never encountered during training and can thus be used as generic feature extractors. This makes CNNs ideal for extracting high quality image features.

We used the GoogLeNet [27] implementation of the Caffe deep learning framework [12] to extract features from the thumbnails of the videos. The network was pre-trained as an image classifier on the ImageNet ILSVRC 2014 dataset [19] that contains 1.2M images organized into 1000 categories. The video thumbnails first had to be scaled down and cropped in order to fit the input of the network. Features were extracted from the last average pooling layer.[5] The feature vectors were normalized to an $l2$ norm of 1. The image feature representation we end up with is a real-valued vector of length 1024.

Figure 2 demonstrates the feature quality by showing the 3 most similar images to two query images, where similarity is defined as the cosine similarity between the image feature vectors. Given the good quality, we do not plug the CNN directly into the RNN, as it would introduce unnecessary complexity to the training and is also unpractical, because (a) this network would converge much slower as it needs to learn the model on incomplete/changing item representations; (b) the network would not be suitable for datasets with lower number of items, as 10,000s of items are not enough to leverage the full potential of the CNN; (c) retraining would take much longer. Another possibility is to use the pretrained network and fine tune the item representations during the training of the RNN. This did not make much difference in our experiments, therefore we did not use fine tuning.

[5]Extracted as the value of the *pool5_7x7_s1* layer.

Figure 2: Top3 similar images to query images, based on cosine similarity of image feature vectors.

Encoding text. Given the strict limitation on the length of the descriptions imposed by online classified advertisement platforms, advertisers usually provide rather concise text for their items. The main goal of the description is to attract the attention of potentially interested users. Therefore, descriptions often contain only the main characteristics of the item and use syntactically incorrect sentences. Moreover, it is not rare to have descriptions written in multiple languages to capture a broader audience. The majority of descriptions of our dataset used a subset of 3 main languages with a handful of less frequent ones also present.

Given the inherent noise in user generated, unstructured text and multiple languages in our data, we adopted the classical bag-of-words representation to encode product descriptions. First we concatenated the title and the description of the items.[6] We filtered stopwords and extracted unigrams and bigrams from text, and discarded all the entries that appear only once. Finally, the resulting bag-of-words was weighted using TF-IDF [23]. The item representation is a sparse vector of length 1,099,425 with an average of 5.44 non-zero coordinates.

We tried other methods to extract features from unstructured text, e.g. distributed bag-of-words [18] and Language Modeling with RNNs [17]. However the classical bag-of-words with TF-IDF was found to work best with our data. We attribute this to the noisiness of user generated content. The lack of English text and the presence of multiple languages prevented us from using pre-trained word representations, e.g. from word2vec.[7]

Experiments with adding an embedding layer between the features and the network resulted in worse performance, therefore, the classical bag-of-words/TF-IDF features were used as item representations and were used directly as the input of the RNNs.

5. RESULTS

The evaluation was done on two proprietary datasets. The first dataset – coined VIDXL – was collected over a 2-month period from a Youtube-like video site, and contains video watching events having at least a predefined length. During the collection item-to-item recommendations were displayed next to the featured video, generated by a selection of different algorithms. The second dataset consists of product view events of an online classified site. We refer to this dataset as CLASS. The site also had recommendations displayed with different algorithms during the collection period.

During the preprocessing of the raw event streams we filtered out unrealistically long sessions as these are likely due to bot traffic. We removed sessions of one (single click) event because they are not useful for session-based recommendations and also removed items whose support is below five, as items with low support are not ideal for modeling. The sessions of the last day of each dataset are put into the test set. Each session is assigned to either the training or the test set, we do not split the data mid-session. We also filter items from the test dataset if they were not in the training set. This affects only a tiny fraction of the items. The datasets are summarized in Table 1.

Table 1: Properties of the datasets.

Data	Train set		Test set		Items
	Sessions	Events	Sessions	Events	
VIDXL	17,419,964	69,312,698	216,725	921,202	712,824
CLASS	1,173,094	9,011,321	35,741	254,857	339,055

We evaluate wrt. the (sequential) next-event prediction task, i.e. given an event of the session how well can the algorithms predict the next event of the session. The trained model is fed with the events of a session one after another and we check the rank of the selected item of the next event. The hidden state of the network is reset to zero after a session ends.

As recommender systems recommend only a few items at once, the relevant item should be amongst the first few items of the list. Therefore, our first evaluation metric is recall@20 that is the proportion of cases having the desired item amongst the top-20 items of all test cases. Recall does not consider the actual rank of the item as long as it is below 20. This is an accurate model for certain practical scenarios where no recommendation is highlighted and their absolute order does not matter. Recall also usually correlates well with important online KPIs, such as click-through rate (CTR)[8]. The second metric used in the experiments is MRR@20 (Mean Reciprocal Rank). MRR is the average of the reciprocal ranks of the desired items. The reciprocal rank is set to zero if the rank is above 20. MRR takes the rank of the item into account, which is important in cases where the order of recommendations matters (e.g. the lower ranked items are only visible after scrolling).

5.1 Thumbnail based video recommendation

We extracted image features from the thumbnails of the videos, see Section 4 for the details of the feature extraction. We experimented with different architectures and training strategies described in Section 3 to see how image data can contribute to recommendation accuracy.

Similarly to [7], the networks optimize for TOP1 loss using adagrad. The parameters – such as dropout, learning rate, momentum, batch size, etc. – of the ID only and the feature only networks were optimized on a hold out validation set. Then the networks were retrained on the full training set (validation set included) and the final results were measured on the test set. Due to the size of the VIDXL dataset, the more complex networks used the optimal parameters of the ID and image networks in their corresponding subnets. The weights of the subnets were set to be equal as we did not get significantly different results until either of the subnet weights was set to zero.

To speed up evaluation, we computed the rank of the rel-

[6]We explicitly avoided to repeat the title if this was already written at the beginning of the text

[7]https://code.google.com/p/word2vec

evant item compared to the 50,000 most supported items. While this evaluation methodology somewhat overestimates the rank and thus the evaluated metrics are a little bit higher, the comparison of the algorithms remains fair [1].

Table 2 summarizes the results with different architectures and training methods. In this experiment the number of hidden units was set to 100 for the baseline architectures and 100 per subnetwork for p-RNNs. The networks are trained for 10 epochs as there is no significant change in the loss after that. The networks are compared to the item-kNN algorithm, the de-facto solution for item-to-item and item-to-session recommendation tasks in the industry. p-RNNs with 100+100 hidden units can easily outperform the ID only network with 100 units, due to the additional information source and the increase of the overall capacity of the network. Therefore we also measured the accuracy of the ID only network with 200 units. Note that this is a very strong baseline, because having 200 hidden units increases the capacity of the network ∼4 times, while having 100+100 only doubles it. Also, the doubled capacity of p-RNN is split between two information sources, therefore it is clearly in disadvantage to even an RNN with doubled capacity. Nevertheless we show that p-RNNs can often beat this strong baseline as well, while they are typically better than the ID only network with 100 units.

Table 2: Results on VIDXL, using image features extracted from thumbnails. The best results are typeset in bold. p-RNNs use 100+100 hidden units, others use 100 unless stated otherwise. Performance gain over item-kNN is shown in parentheses.

Method	Recall@20	MRR@20
Item-kNN	0.6263	0.3740
ID only	0.6831 (+9.07%)	0.3847 (+2.85%)
ID only (200)	0.6963 (+11.17%)	0.3881 (+3.77%)
Feature only	0.5367 (−14.30%)	0.3065 (−18.05%)
Concatenated	0.6766 (+8.03%)	0.3850 (+2.94%)
Parallel (sim)	0.6765 (+8.01%)	0.4014 (+7.34%)
Parallel (alt)	0.6874 (+9.76%)	0.4331 (+15.81%)
Parallel (res)	0.7028 (+12.21%)	**0.4440** (+18.72%)
Parallel (int)	**0.7040** (+12.41%)	0.4361 (+16.60%)
Shared-W (sim)	0.6681 (+6.66%)	0.4007 (+7.13%)
Shared-W (alt)	0.6804 (+8.63%)	0.4035 (+7.88%)
Shared-W (res)	0.6425 (+2.58%)	0.3541 (−5.31%)
Shared-W (int)	0.6658 (+6.31%)	0.3715 (−0.66%)
Int. model (sim)	0.6751 (+7.78%)	0.3998 (+6.90%)
Int. model (alt)	0.6847 (+9.32%)	0.4104 (+9.74%)
Int. model (res)	0.6749 (+7.76%)	0.4098 (+9.56%)
Int. model (int)	0.6843 (+9.25%)	0.4040 (+8.02%)

Similar to [7], the RNN outperforms the item-kNN baseline by a large margin. The recall for the RNN on this task is very high, therefore it is very hard for the more advanced architectures to significantly improve on this result. The network that is trained on the image features only is significantly worse than the ID only network and even worse than the item-kNN, demonstrating that the sequence of item features in and of itself is not enough to model the session well. Feeding the network with the concatenated input of IDs and image features, because the stronger input dominates during the training, thus the performance hardly differs from that of the ID only network. It is hard for a single GRU layer to handle two types of inputs at once, resulting in a performance very similar to that of the ID only network. Adding item features using the naive approach has no observable benefits. Therefore we propose to use p-RNNs instead.

Moving on to the proposed p-RNN architectures, one can see that several configurations perform significantly better than the strong ID only baseline. Due to the originally high recall of the network, these novel architectures mostly increase the MRR, i.e. they don't find more relevant items, but they rank them better. The best performing architecture is the classic parallel one. With the naive simultaneous training it is significantly better than the strong ID only baseline wrt. MRR, but slightly worse wrt. recall. With simultaneous training, different components of the p-RNN learn the same relations from the data, thus the full capacity of the network is not leveraged. Therefore we propose using alternative training strategies.

The best of the alternative training methods is residual training, closely followed by the interleaving one, but the alternating training is also not far behind. The p-RNN with residual training outperforms the strong ID only baseline by 14.07% in MRR, while achieving similar recall. The improvement is even greater over the industry de facto item-kNN solution: 12.21% in recall and 18.72% in MRR.

Table 3: Results on VIDXL, using image features extracted from thumbnails. The best results are typeset in bold. p-RNN architectures use 500+500 hidden units, others use 1000. Performance gain over item-kNN is shown in parentheses.

Network variant	Recall@20	MRR@20
ID only, 10 epochs	**0.7279** (+16.21%)	0.4350 (+16.32%)
ID only, 20 epochs	0.7207 (+15.07%)	0.4287 (+14.63%)
Feature only	0.6479 (+3.45%)	0.4089 (+9.34%)
Concatenated	0.7216 (+15.22%)	0.4291 (+14.72%)
Parallel (sim)	0.7084 (+13.10%)	0.4420 (+18.19%)
Parallel (alt)	0.7142 (+14.03%)	0.4456 (+19.15%)
Parallel (res)	0.7165 (+14.40%)	0.4513 (+20.67%)
Parallel (int)	**0.7262** (+15.59%)	**0.4587** (+22.64%)

By increasing the number of hidden units, the capacity of the RNN increases, thus this parameter has a large effect on performance. However this parameter also obeys the law of diminishing returns. We found that results do not improve significantly above ∼1000 hidden units on this problem. We ran experiments with 1000 units on non-parallel and 500+500 units on the best performing p-RNN architecture (i.e. classic parallel) to confirm that adding item features can also benefit session modeling when increasing the network capacity and/or the number of epochs has diminishing returns. Table 3 depicts the results.

With more hidden units, the performance increases and even the feature only network outperforms the item-kNN baseline as the capacity of the network is enough to leverage the information in the image features. But otherwise the relation between the results is similar to that of the previous experiments. This further underpins that p-RNNs with alternative training strategies are vital for efficiently incorporating item features into learning session models. Further increasing the number of hidden units and/or the number of epochs did not increase the performance of any network significantly, but p-RNN architectures significantly outperform

the ID only network with more than 2 times larger capacity in terms of MRR (~7% with interleaving training) and have similar recall. This means that the proposed architectures with the proposed training strategies can significantly increase performance, even when increasing the capacity of the network has diminishing returns. In other words, adding additional data sources (item features) can increase the accuracy of recommendations beyond the maximum achievable just from item IDs. However handling multiple sources requires special architectures and training: p-RNNs and alternative training strategies.

5.2 Using product descriptions

We repeated the last experiment – i.e. baseline RNNs had 1000 hidden units; the classic p-RNN had 500 per subnet – on the CLASS dataset with features extracted from product descriptions instead of images. See the detailed feature extraction process in Section 4. The experimental setup was the same as before, except that we opted for ranking all items during evaluation, because it is possible to do the full evaluation within reasonable time due to the significantly smaller size of this dataset (compared to VIDXL).

Table 4: Results on CLASS, using textual features extracted from product descriptions. The best results are typeset in bold. p-RNN architectures use 500+500 hidden units, others use 1000. Performance gain over item-kNN is shown in parentheses.

Training info	Recall@20	MRR@20
Item-kNN baseline	0.2387	0.0839
ID only, 10 epochs	0.2849 (+19.38%)	0.1062 (+26.56%)
ID only, 20 epochs	0.2783 (+16.60%)	0.1051 (+25.26%)
Feature only	0.2397 (+0.47%)	0.0937 (+11.70%)
Conc. input	0.2844 (+19.17%)	0.1029 (+22.64%)
Parallel (sim)	0.2741 (+14.85%)	0.1019 (+21.53%)
Parallel (alt)	0.2877 (+20.97%)	0.1096 (+30.62%)
Parallel (res)	**0.2946** (+23.44%)	**0.1119** (+33.36%)
Parallel (int)	0.2854 (+19.57%)	0.1058 (+26.17%)

The results (see Table 4) concur with that of the earlier experiments with image features. The text only network significantly outperforms the item-kNN baseline in terms of MRR. This confirms that textual features can be effectively exploited to generate better rankings. However it falls short when compared with the ID only network. This suggests that text features alone are not enough. With concatenated input, the network performs similarly to the ID only network analogously to previous experiments.

The proposed alternative training strategies are of crucial importance when training p-RNNs, the simultaneous training is clearly suboptimal wrt. recommendation accuracy. The classic p-RNN with alternative training strategies significantly outperformed both the ID only RNN and item-kNN in *both* recall and MRR. Residual training proved to be the best strategy in this experiment with ~6% improvement in recall and ~6.5% in MRR over the ID only network. Note that further increasing the number of hidden units or number of epochs for the baseline RNNs did not improve the results any further. Thus using text based item features in p-RNNs with proper training can also increase recommendation accuracy beyond what is achievable from IDs only.

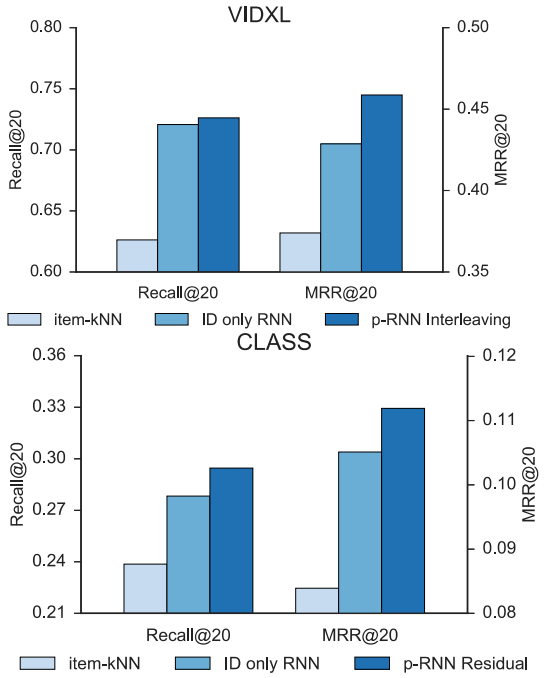

Figure 3: Comparing best performing p-RNN against the ID only RNN and item-kNN.

We demonstrate the power of the proposed solution (500 units per subnet) by comparing it to item-kNN and the ID only network with 1000 hidden units on Figure 3.

6. CONCLUSION

In this paper we examined the use of item features (image data and text) in RNN-based session modeling to improve session-based recommendations. We pointed out that using item features in and of themselves is not enough to properly model the session and combining multiple data sources efficiently (e.g. ID and image) is not trivial. We proposed p-RNN architectures that can leverage the added value of multiple item representations. We devised alternative training strategies (alternating, residual and interleaving training) that fit these architectures better than backpropagating the error through the whole network. The proposed architectures and training methods significantly outperformed both RNNs with ID only input and the industry de facto solution for this problem, item-kNN. Finally, we showed that by using p-RNNs with alternative training, recommendation accuracy can be increased beyond the maximum that is achievable by RNNs with ID only input by increasing their capacity and/or the number of training epochs.

Acknowledgments

The work leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under CrowdRec Grant Agreement n° 610594.

7. REFERENCES

[1] A. Bellogin, P. Castells, and I. Cantador. Precision-oriented evaluation of recommender systems: An algorithmic comparison. In *RecSys'11: 5th ACM Conf. on Recommender Systems*, pages 333–336, 2011.

[2] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *SSST-8: 8th Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, 2014.

[3] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In *ICML'14: 31st Int. Conf. on Machine Learning*, pages 647–655, 2014.

[4] A. M. Elkahky, Y. Song, and X. He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *WWW'15: 24th Int. Conf. on World Wide Web*, pages 278–288, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[6] R. He and J. McAuley. VBPR: Visual Bayesian Personalized Ranking from implicit feedback. *CoRR*, 1510.01784, 2015.

[7] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. *International Conference on Learning Representations*, 2016.

[8] B. Hidasi and D. Tikk. Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback. In *ECML-PKDD'12, Part II*, number 7524 in LNCS, pages 67–82. Springer, 2012.

[9] B. Hidasi and D. Tikk. General factorization framework for context-aware recommendations. *Data Mining and Knowledge Discovery*, 30(2):342–371, 2015.

[10] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[11] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *MM'14: 22nd ACM Int. Conf. on Multimedia*, pages 675–678, 2014.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NISP'12: 26th Annual Conf. on Neural Information Processing Systems 2012*, pages 1106–1114, 2012.

[14] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[15] Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *CoRR*, 1506.00019, 2015.

[16] J. McAuley, C. Targett, Q. Shi, and A. van den Hengel. Image-based recommendations on styles and substitutes. In *SIGIR'15: 38th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 43–52, 2015.

[17] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur. Recurrent neural network based language model. In *INTERSPEECH'10: 11th Ann. Conf. of the Int. Speech Communication Association*, pages 1045–1048, 2010.

[18] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, 1310.4546, 2013.

[19] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *Int. Journal of Computer Vision*, 115(3):211–252, 2015.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. ImageNet large scale visual recognition challenge. *CoRR*, 1409.0575, 2014.

[21] H. Sak, O. Vinyals, G. Heigold, A. Senior, E. McDermott, R. Monga, and M. Mao. Sequence discriminative distributed training of long short-term memory recurrent neural networks. *Entropy*, 15(16):17–18, 2014.

[22] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML'07: 24th Int. Conf. on Machine Learning*, pages 791–798, 2007.

[23] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, Aug. 1988.

[24] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW:01: 10th Int. Conf. on World Wide Web*, pages 285–295, 2001.

[25] G. Shani, R. I. Brafman, and D. Heckerman. An MDP-based recommender system. In *UAI'02: 18th Conf. on Uncertainty in Artificial Intelligence*, pages 453–460, 2002.

[26] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. In *CVPR'14: IEEE Conf. on Computer Vision and Pattern Recognition Workshops*, June 2014.

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, 1409.4842, 2014.

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

[29] H. Wang, N. Wang, and D.-Y. Yeung. Collaborative deep learning for recommender systems. In *KDD'15: 21th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 1235–1244, 2015.

[30] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester. Collaborative denoising auto-encoders for top-N recommender systems. In *WSDM'16: 9th ACM Int. Conf. on Web Search and Data Mining*, pages 153–162, 2016.