Binary Latent Representations for Efficient Ranking

Empirical Assessment

Maciej Kula
maciej.kula@gmail.com

ABSTRACT

Large-scale recommender systems often face severe latency and storage constraints at prediction time. These are particularly acute when the number of items that could be recommended is large, and calculating predictions for the full set is computationally intensive. In an attempt to relax these constraints, we train recommendation models that use binary rather than real-valued user and item representations, and show that while they are substantially faster to evaluate, the gains in speed come at a large cost in accuracy. In our Movielens 1M experiments, we show that reducing the latent dimensionality of traditional models offers a more attractive accuracy/speed trade-off than using binary representations.

KEYWORDS

Recommender Systems, Matrix Factorization, Binary Representations

1 INTRODUCTION

Industry ranking and recommendation systems need to scale to tens or hundreds of millions of items and users. As the number of items available to be recommended grows large, efficiently ranking the item catalogue to produce top-ranked recommendations becomes increasingly challenging. With large catalogues, ranking latency (in on-line settings) and compute and storage costs (in pre-compute settings) place significant constraints on the design of the entire recommender system.

In an on-line setting, where recommendation latency directly impacts the user experience, system designers have to work within a strict latency budget. Respecting that budget often means trading off model accuracy for speed, either via smaller and less expressive models (with lower-dimensionality representations) or through aggressive use of heuristics to exclude large classes of candidate items from scoring. For example, Pinterest uses a combination of heuristics and candidate-generation models to select approximately 1000 pins, from billions of available pins, as input to its related pins ranking system [10]. Similarly, approximate nearest neighbour search techniques are used at YouTube [3] to reduce the number of candidate videos that need to be scored for each user.

The challenge of low-latency scoring leads many systems to rely on a pre-compute system, where recommendations are calculated in advance and stored for serving at a later time. The primary disadvantage of such a system is its inability to react dynamically to user actions (such as new interactions that allow the recommendations to be refined) and recommendation context (location, time of day or year). This leads to often complex architectures that mix fairly simple on-line models, computationally expensive offline models, and intermediate complexity near-line algorithms where model updating is important [1].

Additionally, while a pre-compute solution removes model prediction from the critical path, it still requires substantial investment in computational and storage resources to calculate and cache the predictions of the model. Often the amount of computation required makes timely updating of the cached results impossible. At Pinterest, pre-calculation of related pins results was so demanding that ranking of different segments of the catalogue had to be staggered in time, leading to stale results and reduced ease of development [10, Section 5.5]. Koenigstein et al. [9] estimate that computing recommendations for the Yahoo! Music dataset [4] with a 50-dimensional latent factor model woud take over 135 hours.

To alleviate these constraints in both online and offline settings, we evaluate the use of binary item and user representations in learning-to-rank matrix factorization models. Following the approach of Rastegari et al. [13], we estimate binary user and item representations that are several times faster to score and requiring a fraction of the memory (for the same latent dimensionality). Unfortunately, they are substantially less accurate than standard learning-to-rank approaches. The resulting speed-accuracy trade-offs favour simply reducing the latent dimensionality of traditional models when evaluation speed is desired.

We speculate that binary representations are more useful in case of more complex models (such as deep convolutional models) where a single prediction requires many more floating operations than relatively compact bilinear recommender models.

2 BINARY LATENT REPRESENTATIONS

Rastegari et al. [13] introduce XNOR-Networks, an approach that combines 1-bit quantization of fully-connected and convolutional layers with representing dot products through scaled XNOR and bitcounting operations.

We adapt their approach to the recommendation task by learning binary versions of user and item vectors within a learning-to-rank factorization model. This allows us to use binary instead of floating point operations for computing recommendation rankings, which has significant speed and memory use advantages. In a real-valued n-dimensional factorization model, item and user representations take $4n$ bytes of memory, and computing a single prediction takes $2n$ floating point operations. In an equivalent binary model, $\frac{3}{32}n$ binary operations are required for each prediction (XOR, negation, and bitcounting), and the representations take $\frac{1}{8}n$ bytes of memory. Naive calculations suggest that, for the same latent dimensionality, a binary model would be over 20 times faster to score, and take less then 5% of memory to store its representations.

Naturally, using binary instead of real-valued representations entails a loss of representational power, and the resulting models can be expected to be less accurate. Nevertheless, binary models may still be very useful in a production system, for two reasons. Firstly, they may offer the system designer a more favourable trade-off between speed and accuracy than simply changing the number of latent dimensions in a real-valued model. If latency is the binding constraint, switching to a binary model may offer better or comparable speed with better ranking performance. If accuracy is the priority, a binary model may offer equivalent ranking performance while being faster to score. Secondly, many production systems resort to heavy use of heuristics when selecting candidates to be scored by their ranking systems (as scoring all candidates would be infeasible). To the extent that these heuristics are suboptimal, replacing them with a binary representation model would improve the overall accuracy of the system.

To quantify the trade-offs involved, we construct and fit both a real-valued and a binary version of simple implicit learning-to-rank factorization model. We introduce the notation and the models below.

Let U be the set of users, and I be the set of items. Each user interacts with a number of items S^+; the set of all remaining items is denoted by S^-.

In a traditional real-valued latent factor model, the prediction r_{ui} for any user-item interaction pair $(u, i) \in U \times I$ is given by

$$r_{ui} = \mathbf{u}_u \cdot \mathbf{i}_i + b_u + b_i \tag{1}$$

where $\mathbf{u}_u$ denotes the user and $\mathbf{i}_i$ the item n-dimensional latent vectors, and b_u and b_i the user and item biases.

Rastegari et al. [13] show that the dot product of two real-valued vectors can be approximated in the binary domain by (1) binarizing the input vectors using the sign function (so that the results lie in $\{1, -1\}$), (2) taking their dot product, and (3) scaling the result by the average magnitude of the vectors' elements. Letting

$$\alpha = \frac{1}{n}\|\mathbf{i}_i\|_{\ell 1} \tag{2}$$

and

$$\beta = \frac{1}{n}\|\mathbf{u}_u\|_{\ell 1} \tag{3}$$

represent the scaling factors, the approximation is given by

$$\mathbf{u}_u \cdot \mathbf{i}_i \approx \alpha\beta \left(\operatorname{sign}(\mathbf{u}_u) \cdot \operatorname{sign}(\mathbf{i}_i)\right) \tag{4}$$

Applying this to the latent-factor model, the prediction for a user-item pair is given by

$$r_{ui} = \alpha\beta \left(\operatorname{sign}(\mathbf{u}_u) \cdot \operatorname{sign}(\mathbf{i}_i)\right) + b_u + b_i \tag{5}$$

Note that the scaling factors α and β need to be stored (in addition to the latent representations and biases) to compute predictions in the binary model. Their use also implies a constant cost of two floating-point multiplications per dot product when using binary representations.

The model is trained using backpropagation, and so we use a smooth approximation to the derivative of the sign function to facilitate training. Following Courbariaux et al. [2], we define

$$\frac{d\,\operatorname{sign}}{dw} = \begin{cases} 1 & \text{if } |w| \leq 1 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

We use two loss functions to train the model:

- Bayesian personalised ranking (BPR, Rendle et al. [14]), and
- adaptive sampling maximum margin loss, following Weston et al. [17].

For both loss functions, for any known positive user-item interaction pair (u, i), we uniformly sample an implicit negative item $j \in S^-$. For BPR, the loss for any such triplet is given by

$$1 - \sigma\left(r_{ui} - r_{uj}\right), \tag{7}$$

where σ denotes the sigmoid function. The adaptive sampling loss is given by

$$\left|1 - r_{ui} + r_{uj}\right|_+ . \tag{8}$$

For any (u, i) pair, if the sampled negative item j results in a zero loss (that is, the desired pairwise ordering is not violated), a new negative item is sampled, up to a total of k attempts. This leads the model to perform more gradient updates in areas where its ranking performance is poorest.

For the purposes of this paper, we focus on a simple bilinear collaborative filtering model, and do not use any external metadata information or model the sequential nature of the data. However, the binary dot product approach generalizes beyond simple factorization models, and can be applied as a component of any model whose final scoring step involves a dot product between user and item representations. In particular, models using recurrent [6] or convolutional [11] item or user representations can easily be augmented to use binary dot products in the final ranking stages.

3 EXPERIMENTS

To assess the accuracy-speed trade-offs enabled by the XNOR approach, we conduct an experiment on the Movielens 1M dataset [5]. The dataset contains 1 million ratings from 6000 users on 4000 movies. Because the computational speed improvements offered by the binary representations are linear in the size of the catalogue, the results on this dataset should be indicative of results that can be obtained on larger datasets. At the same time, the dataset's small size enables us to easily test hyperparameter configurations and estimate models with high latent dimensions.

Experimental setup

We randomly divide the dataset into a training, test, and validation set. In order to build a picture of the accuracy-speed trade-offs afforded by real-valued and binary latent models, we explicitly build models for between 32 and 1024 latent dimensions. For every latent dimensionality, we conduct a random search over the hyperparameter space, and pick optimal initial learning rate, loss function, L2 penalty, minibatch size and number of training epochs for each algorithm based on their performance on the test set. The final results are obtained from ranking interactions from the validation set. We use mean reciprocal rank (MRR) as a measure of ranking quality.

Benchmark results are measured on an Intel Xeon E5-2686v4 CPU by running a 500 repetitions of scoring 100,000 items and averaging the results.

Implementation

The model is implemented in PyTorch and trained using the nVidia K40 GPUs. The implementation is only marginally more complex than a standard learning-to-rank model, and is accomplished by simply replacing the user-item vector dot product operation with its binary counterpart. The change results in a small increase in training time.

During training, the embedding parameters are stored and updated as single precision floating point values. Similarly, the XNOR dot product is carried out using floats in $\{1, -1\}$. The initial learning rate, loss function, L2 penalty, minibatch size and number of training epochs are treated as model hyperparameters. All models are trained using the Adam training rate schedule [8].

The prediction code runs on the CPU and is implemented in C using Intel X86 AVX2 SIMD intrinsics. SIMD (Single Instruction Multiple Data) instructions allow the CPU to operate on multiple pieces of data in parallel, achieving significant speedups over the scalar version. We use explicit intrinsics rather than compiler autovectorization to ensure that neither the real-valued nor the binary prediction code is unfairly disadvantaged by the quality of compiler autovectorization.

The real-valued prediction code is implemented using 8-float wide fused multiply-add instructions (`_mm256_fmadd_ps`). In the binary version, the real-valued embedding parameters used in training are discarded, and the derived 1-bit weights are packed into 32-bit integer buffers. The XNOR dot product is implemented using 8-integer wide XOR operations (256 binary weights are processed at a time), followed by a popcount instruction to count the number of on bits in the result. We use the `libpopcnt` [12] library for the bit counting operations.

Both versions use 32-bit aligned input data to utilise aligned SIMD register load instructions. The code is compiled using GCC 4.8.4 for the AVX2-enabled Broadwell architecture.

The source code for both model training and prediction is available on Github at https://github.com/maciejkula/binge.

Results

Table 1 summarises the results of the Movielens 1M experiment. Each row presents the best MRR (mean reciprocal rank) results for both real-valued and binary models of a given dimensionality. MRR ratio denotes the fraction of the real-valued MRR the binary model achieves for that dimensionality; PPMS (predictions per millisecond) ratio denotes how many more predictions the binary model can compute per millisecond.

Results on scoring speed and memory use broadly confirm the naive calculations from section 2, converging to over 20 times faster scoring at around 3% of memory as latent dimensionality increases. Memory savings plateau as latent dimensionality increases and storing the binary scaling factors becomes less important.

As expected, for the same dimensionality a binary model achieves lower ranking accuracy. On average, the accuracy loss when moving from a continuous to a binary model of the same latent dimensionality is around 11%, varying between 6% and 23%.

Unfortunately, while continuous models retain good accuracy as latent dimensionality decreases, binary models' representational power sharply deteriorates. Moving from the 1024 to 32 dimensions in the continuous model implies

Table 1: Movielens 1M results

Dimension	MRR	Binary MRR	MRR ratio[1]	PPMS[2]	Binary PPMS	PPMS ratio[3]	Memory use ratio[4]
32	0.077	0.059	0.768	87,758	264,146	3.010	0.091
64	0.075	0.066	0.878	45,992	220,833	4.802	0.062
128	0.076	0.071	0.935	23,870	166,252	6.965	0.047
256	0.078	0.071	0.908	12,284	190,131	15.477	0.039
512	0.080	0.073	0.912	6074	122,637	20.190	0.035
1024	0.080	0.075	0.936	3056	61,301	20.056	0.033

[1] Ratio of binary model MRR to real-valued model MRR
[2] Predictions Per Millisecond: how many items can be scored per millisecond
[3] Ratio of binary PPMS to real-valued PPMS
[4] Ratio of memory required to store binary vs. real-valued parameters

a 29 times increase in prediction speed at the expense of a modest 4% decrease in accuracy. This compares favourably to switching to binary representations: moving to a 1024-binary dimensional representation implies a sharper accuracy drop at 6% in exchange for a smaller 20 times increase in prediction speed. Moving to 32 dimensions yields a further speed gains at 86 times, but at the cost of a considerable loss of accuracy at 26%.

The results suggest that latent recommender models have good representational power even at low latent dimensionalities. This advantage is lost when using binary embeddings, which need to be high-dimensional to achieve comparable accuracy. At the same time, binary representations' speed advantage is only evident at high dimensions: at low dimensions, only floating-point operations can use enjoy the advantages of SIMD operations, and the fixed overhead of binary scaling factors α and β constitutes a larger proportion of the total computational cost. We conjecture that the attractiveness of binary representations is tightly coupled with high-dimensional models where a single prediction requires many floating point operations, such as convolutional neural networks. Relatively compact latent factor models do not fall into this category.

4 ALTERNATIVE APPROACHES

Given the limited success of binary embedding, a more promising approach is offered by existing work that focuses on reducing the number of dot products which need to be performed in order to retrieve top recommendations. Koenigstein et al. [9] introduces an exact branch-and-bound based on metric trees as well as an approximate algorithm that clusters users and serves recommendations computed for the cluster centers. Shrivastava and Li [16] extend well-known [7] locality-sensitive hashing techniques to maximum inner product search (MIPS) through asymmetric transformations of the query and candidate vectors.

The method that is most closely related to the model we present in this paper is Shen et al. [15], who expand on MIPS by learning asymmetric binary hash functions. The hash functions are trained to minimize the L2 norm between the inner product matrix of the original input vectors and the inner product matrix of their binary representations.

5 CONCLUSION

While prediction latency is a pressing concern for many recommender systems, we find that the already compact latent factor models do not stand to benefit from using binary latent representations. The sharp drop in representational power exhibited by binary embeddings makes other approaches (such as exact [9] and approximate [16] maximum inner product search) more promising avenues when optimizing large-scale ranking.

REFERENCES

[1] Xavier Amatriain. 2013. Big & personal: data and models behind netflix recommendations. In *Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*. ACM, 1–6.

[2] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[3] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 191–198.

[4] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. 2011. The yahoo! music dataset and kdd-cup'11. In *Proceedings of the 2011 International Conference on KDD Cup 2011-Volume 18*. JMLR. org, 3–18.

[5] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5, 4 (2016), 19.

[6] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[7] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 604–613.

[8] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[9] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. 2012. Efficient retrieval of recommendations in a matrix factorization framework. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 535–544.

[10] David C Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. 2017. Related Pins at Pinterest: The Evolution of a Real-World Recommender System. In *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee, 583–592.

[11] Corey Lynch, Kamelia Aryafar, and Josh Attenberg. 2015. Images Don't Lie: Transferring Deep Visual Semantic Features to Large-Scale Multimodal Learning to Rank. *arXiv preprint arXiv:1511.06746* (2015).

[12] Wojciech Muła, Nathan Kurz, and Daniel Lemire. 2016. Faster Population Counts using AVX2 Instructions. https://github.com/kimwalisch/libpopcnt. *arXiv preprint arXiv:1611.07612* (2016).

[13] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, 525–542.

[14] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press, 452–461.

[15] Fumin Shen, Wei Liu, Shaoting Zhang, Yang Yang, and Heng Tao Shen. 2015. Learning Binary Codes for Maximum Inner Product Search. In *The IEEE International Conference on Computer Vision (ICCV)*.

[16] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Advances in Neural Information Processing Systems*. 2321–2329.

[17] Jason Weston, Samy Bengio, and Nicolas Usunier. 2011. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, Vol. 11. 2764–2770.