CS230

Deep Bayesian Recommendation Systems

Dmitri M. Saberi
Department of Mathematics
Stanford University
dsaberi@stanford.edu

Sal R. Spina
Department of Computer Science
Stanford University
salspina@stanford.edu

Abstract

When users interact with a website, typically their (retail product, movie, song, or social media account) recommendations will come from a model of rewards based on user behavior, as well as uncertainty in those predictions. The presence of deep neural networks can lead to better recommendations, but with the caveat of more difficult and costly uncertainty estimates. Importantly, modeling the risk in a recommendation helps to better navigate the *explore/exploit* tradeoff, i.e., to decide whether to recommend similar or new, novel content based on available context. In this paper, we explore using approximate Bayesian Neural Networks to yield better approximations of reward uncertainties, and implement our method with both Thompson Sampling and Upper Confidence Bound policies. We compare results across a variety of baseline methods from the literature.

1 Introduction

Recommendation systems (RS) are one of the most widespread and successful applications of modern machine learning. Widely impactful in social media [7], multimedia streaming services [13], retail [3], and many other sectors, RS engines often drive the way we interact with products in the era of big data. The immediate applicability (and profitability) of such systems has driven much research into incorporating all sorts of cutting-edge ML techniques. The generic RS setting rests well in the formalism of reinforcement learning (RL), and can be stated as follows: a user accesses a site at times $t \in \mathcal{T}$. At time t, there is a set of actions $a \in \mathcal{A}$ (e.g., recommended products, articles, or movies) that an engine could propose to the user. For each potential action $a \in \mathcal{A}$, we have access to some features $\mathbf{x}_{t,a}$ (e.g., user click data), and there is some (stochastic) reward associated with enacting a, namely $r_{t,a}$. We then posit a model f_{θ_a} of the expected rewards, and can pose the supervised learning problem:

$$\text{find } \theta_t^\star \text{ such that } f_{\theta_t^\star}(\mathbf{x}_{t,a}) \approx \mathbb{E}[r_{t,a}|\mathbf{x}_{t,a}]. \tag{1}$$

This is known as a *multi-armed contextual bandit* problem [10]. There are (speaking roughly) two main design choices here: (i) model architecture/training for f_θ, and (ii) deciding how to choose an optimal action $a \in \mathcal{A}$ based on the predicted rewards $f_{\theta^\star}(\mathbf{x}_{t,a})$. We will discuss several popular design choices for (i) that have been proposed (and used successfully) in the next section, so let us focus on (ii) for now. The metric associated to our choice of action is *cumulative regret:*

$$R(T) = \mathbb{E}\left[\sum_{t=1}^{T}(r_{t,a_t^\star} - r_{t,a_t})\right]$$

which we aim to minimize. Here, $r_{t,a_t^\star}^\star$ is the largest true reward among all $\{r_{t,a}\}_{a\in\mathcal{A}}$ at time t. An equivalent objective is to maximize total expected reward.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

Once we have produced a stochastic estimate of the rewards $\widehat{r}_{t,a} := f_{\theta^\star}(\mathbf{x}_{t,a})$, the naïve greedy approach is to pick action $a_t := \text{argmax}_{a \in \mathcal{A}} \widehat{r}_{t,a}$. But this poses a problem, namely that such recommendation engines would typically recommend only products that are *very similar* to products that the user has recently interacted with. In addition to being able to *exploit* our knowledge about user preferences, it is also crucial to *explore* the action space and give novel suggestions (this will, in turn, give us more information about user preferences). In order to do this, we need to quantify the uncertainty in our predictions, i.e., estimate properties of the posterior distribution $p(\theta_t|\mathcal{D}_t)$, where $\mathcal{D}_t$ is the available data at time t. One approach is called *Thompson Sampling* (TS): we sample $\theta_t^{\text{post}} \sim p(\theta_t|\mathcal{D}_t)$, and predict rewards as $\widehat{r}_{t,a} := f_{\theta_t^{\text{post}}}(\mathbf{x}_{t,a})$. Then we pick the action $a \in \mathcal{A}$ maximizing predicted rewards. Another (deterministic) approach is called the *upper confidence bound* (UCB) algorithm. In this case, an approximate confidence interval is constructed, so that $r_{t,a} \in (\widehat{r}_{t,a} - \widehat{\sigma}_{t,a}, \widehat{r}_{t,a} + \widehat{\sigma}_{t,a})$ with high probability. Then, the action is chosen to maximize the a weighted sum of risk and return:

$$a_t := \text{argmax}_{a \in \mathcal{A}} \widehat{U}_{t,a} := \widehat{r}_{t,a} + \gamma_t \widehat{\sigma}_{t,a}. \tag{2}$$

If we have an approximate posterior $p(\theta_t|X_t, y_t)$ over the parameters, then estimating the standard deviation $\widehat{\sigma}$ would be one possible approach (used, e.g. in [9]). Other common approaches use a Chernoff-based bound. Generally, it becomes more difficult to obtain more accurate confidence intervals as the model f_θ gets more complex. For example, it is extremely difficult (or in some cases, intractible) to sample from the posterior distribution of a Bayesian Neural Network (BNN)[1]. But there has been recent work [12, 5] on more accurate and scalable (approximate) BNNs, and more generally, on uncertainty quantification in deep learning [1]. We focus on improving deep FCNN uncertainty quantification, towards the end of better navigating the explore/exploit tradeoff. In particular, we apply the Laplace approximation to achieve more accurate knowledge of the posterior, hence (hopefully) improving TS and UCB performance for FCNN reward predictions. We view this as especially crucial in the large-data regime, i.e. with many high-dimensional features, as FCNNs will learn more intricate relationships between different arms.

2 Related work

We discuss a few categories of approaches that have been applied to the problem (1). To our knowledge, the state-of-the-art for our setting is given by Murphy et. al, as described below in the Hybrid-neural-linear section.

Linear bandits. First introduced in [9], the linear approach simply uses the model $f_\theta(\mathbf{x}_{t,a}) = \theta_t^\top \mathbf{x}_{t,a}$. There are several clear benefits: for reward estimation, this approach permits an explicit solution for $\theta_{t,a}^\star$ via ridge regression. There is also a closed form posterior distribution for ridge regression with Gaussian noise, which allows for exact uncertainty quantification for UCB, and better posterior samples for TS. Additionally, due to its simplicity, this model is also extremely efficient. Of course, the tradeoff is that it has very limited expressibility and can't model complicated reward functions well (e.g., assumes that the arms are independent, while in practice they are highly correlated).

Neural bandits. The most prominant example of using fully connected neural networks (FCNNs) to model returns in the current setting is given by Zhou et. al [14] in their NeuralUCB algorithm. Roughly, the only main difference between their approach and ours is that their upper confidence bound is adaptively estimated using gradients of the network based on an analytic bound for their approximate interval. They achieve near-optimal regret bounds, and strongly outperformed LinUCB empirically (on the MNIST dataset, as well as covertype, magic, and statlog from the UCI ML Repository). We did not have time to benchmark against their method, but plan to for future work.

Hybrid neural-linear bandits. This category subsumes a variety of approaches. First, and truest to the name, there is the NeuralLinear algorithm introduced in [11], which uses a neural network as a feature extractor with a linear layer on top. There is also a variant of NeuralLinear that approximates the old covariance of context that is outside of the memory buffer, to avoid "catastrophic forgetting". We refer to this as NL − Lim in our benchmarks. Finally, the best performing methods to our

[1] Here, by a Bayesian neural network we simply mean a neural network where the parameters are assumed to be stochastic, and are given a prior distribution.

knowledge are the EKF-based methods proposed in [4], which roughly uses a Kalman filter to optimize a low-dimensional linear projection of the neural network parameters, and propogates a normally distributed belief state over time. That is, they sample $z \sim \mathsf{N}(\mu_t, \Sigma_t)$ at time t, compute $\theta_t = Az_t + \theta_*$ to get the neural network parameters, and use the neural network to infer the expected rewards as a function of z. They then use the extended Kalman filter update to get $(\mu_{t+1}, \Sigma_{t+1})$ from the context. This method worked extremely well on MovieLens (in comparison to the other baselines listed here, sans NeuralUCB), in addition to MNIST and a variety of tabular datasets.

The closest relatives of our algorithm are NeuralUCB and NeuralLinear. Our key distinguishing factor is our focus on more accurately modeling the *posterior distribution*, and in particular obtaining better confidence bounds for UCB (moreover, NeuralLinear doesn't train the FCNN for inference, just for feature extraction). LinUCB enjoys exact Bayesian inference, and thus obtains accurate confidence intervals in this manner. However, this is not numerically tractible for deep neural networks.

3 Dataset and Features

We use the MovieLens dataset, adapted from Murphy et. al's implementation[2] [4]. MovieLens is a standard dataset used for prototyping movie recommendation engines based on user behavior. The dataset consists of $\sim$100K ratings on a $1-5$ scale from 943 users on 1682 movies. We followed the data augmentation procedure of Murphy et. al [4], which is as follows: we compute a dense rank r SVD approximation $U_r \Sigma_r V_r^\top \approx X$, then X_{ij} (the rating user i gives to movie j) is the reward associated with context u_i. Note that the 943×1682 data matrix is sparse (i.e., there are quite a few movies that each user has not seen in the data matrix), so a low-rank approximation is appropriate. We construct a rank 50 approximate SVD for our purposes (so our feature size is 50).

Since we are in RL setting, all agents re-train based on available context. With this in mind, we split our data $80-20$ between training and validation sets. We used the training set to make our design choices (e.g., tune hyperparameters, change model architectures, iterate on the algorithm, etc.), and used the testing set to evaluate reward and train in an online setting on previously unseen data. Thus, models still train on the validation set (to incorporate new available contexts) but we do not adapt the model or optimization configuration after exposure to the validation data. As a note, Murphy et. al [4] do not incorporate a train/test splits for MovieLens in their paper (it seems that they opted to use several datasets for their method instead).

4 Methods

We first model rewards by a fully connected neural network, the same approach as [14]. Then, after training the network to yield parameters θ_t, we use a Laplace approximation to model the posterior distribution of the neural network in lieu of exact posterior inference:

$$p(\theta|X_t, y_t) = \mathsf{N}(\theta; \theta_t, \nabla^2 \mathcal{L}(\theta_t)^{-1}).$$

However, even this appoximation is intractible for deep FCNNs, since even small FCNNs will have (at least) thousands of parameters, and sampling from Gaussians (with dense covariance matrices) in that regime can be very costly. A typical approximation that is made for numerical effiency is just adopting a "last-layer" Laplace approximation [8], where the posterior distribution is only over the last-layer parameters of the FCNN (and all others are frozen at inference time). This is described in Algorithm 1. Approximate knowledge of the posterior is sufficient for TS, which is one of our subroutines; to do UCB, we first partition the parameters as $\theta_t = (\theta_{<L}, \theta_L^{\text{MAP}})$. We view $\theta_{<L}$ as "fixed" frozen parameters, and θ_L^{MAP} as the maximum a posteriori (MAP) estimate for the last layer as obtained by the training. so we can compute $f_{\theta<L}(\mathbf{x}_{t,a})$ for each action. Then, we have that our estimated rewards will be (approximately) distributed as:

$$\widehat{r}_{t,a} = f_{\theta_t}(\mathbf{x}_{t,a}) = \theta_L^\top f_{\theta_{<L}} \sim \mathsf{N}(f_{\theta_{<L}}(\mathbf{x}_{t,a})^\top \theta_{<L}^{\text{MAP}}, f_{\theta_{\leq L}}(\mathbf{x}_{t,a})^\top \Sigma_a^L f_{\theta_{\leq L}}(\mathbf{x}_{t,a})). \tag{3}$$

We note that there are several potential extensions and modifications to this algorithm. Although we opted for last-layer Laplace as a first concept for this type of approach, one could do a full Laplace approximation, or an approximate Kronecker product Laplace approximation as described in [5] as

[2] https://github.com/probml/bandits

Algorithm 1 BNN-UCB-LL and BNN-TS-LL

Require: $\gamma_t > 0$, learning rate $\eta > 0$, policy π, number of training epochs N, regularization $\lambda > 0$, neural network f_θ with $L > 0$ layers, number of arms A

for $t = 1, \ldots, T$ **do**

  partition $\theta_t = (\theta_{<L}, \theta_L)$ ▷ θ_L are last-layer parameters

  Estimate $\Sigma^L := \nabla^2_{\theta_L} \mathcal{L}(\theta_{<L}, \theta_L)^{-1}$, holding $\theta_{<L}$ fixed

  if π is TS **then**

    Sample $\theta_t^{\text{pred}} \sim \mathcal{N}(\theta_t, \Sigma^L)$

    Pick $a_t := \text{argmax}_{a \in [A]} f_{\theta_t^{\text{pred}}}(\mathbf{x}_{t,a})$

  else

    if π is UCB **then**

      Predict $\widehat{r}_{t,a} = f_{\theta_t}(\mathbf{x}_{t,a})$

      set $U_{t,a} := \widehat{r}_{t,a} + \gamma_t f_{\theta_{\leq L}}(\mathbf{x}_{t,a})^\top \Sigma_a^L f_{\theta_{\leq L}}(\mathbf{x}_{t,a})$

      pick $a_t := \text{argmax}_{a \in [A]} \widehat{U}_{t,a}$

    end if

  end if

  observe r_{t,a_t}

  update $X_t := [X_{t-1}^\top, \mathbf{x}_{t,a_t})]^\top$, $y_t := [y_{t-1}^\top, r_t]^\top$

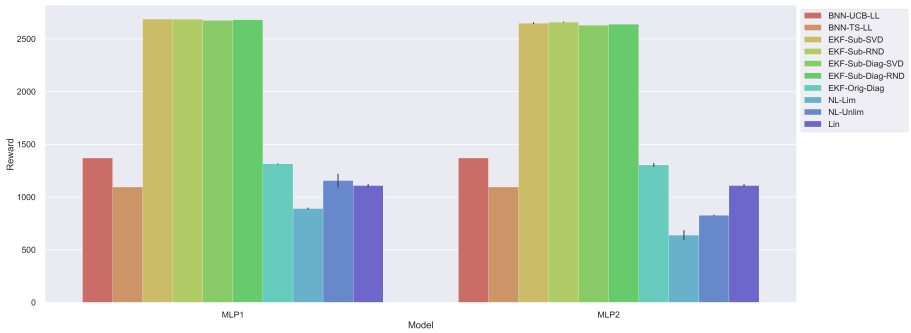
  $\theta_{t+1} = \mathsf{AdamOpt}(f_\theta, X_t, y_t, \eta, N)$

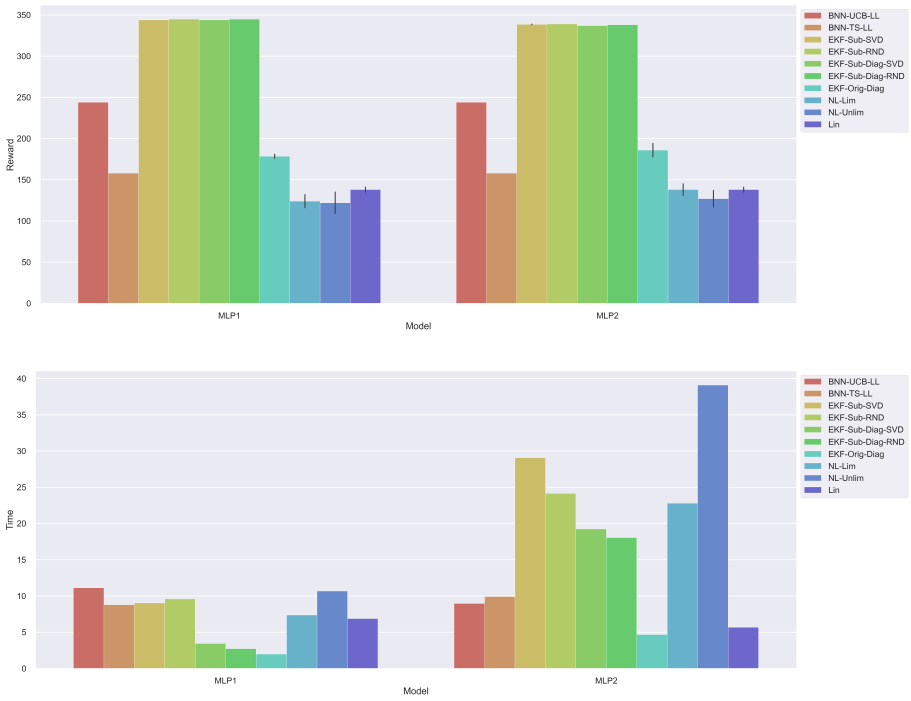
end for

an alternative. We expect methods of this nature to perform even better, especially on larger datasets with access to more compute. In these cases, one would replace the estimated reward distribution (3) with an approximation of the posterior predictive distribution.

5 Experiments/Results/Discussion

We used FLAX (a JAX-based neural network library [6]) and OPTAX (a JAX-based optimization library [2]) for our implementation. For our model architecture f_θ, we used a fully-connected neural network with two layers, a ReLU activation, and hidden layer size of 50. We trained using the Adam optimizer with an ℓ_2-loss, a learning rate of 0.1, and an ℓ_2-regularization of 0.7 for 10 epochs. Following Murphy [4], we perform all *baseline* experiments (except LinUCB) with two different feature extractors, a one-layer (MLP1) and two-layer MLP (MLP2) that are trained during warmup. We do not use these feature extractions for our method, and relative performance is not impacted too much. We found that since we were learning in the online setting with a fairly small neural network, increasing the learning rate was helpful. Additionally, since overfitting in the adaptive online setting would be typical of a deeper model, we found that higher regularization was helpful in achieving better results. For the same reason, the number of epochs could not be too high (when it was, performance degraded significantly). Interestingly, SGD did not work nearly as well as ADAM for our experiments; we expected the added stochasticity to be helpful, but it did not end up being the case. We experimented with larger FCNN architectures (e.g., hidden layes of size $64, 32, 32$), but this did not increase performance, likely due to the dataset being relatively small.





(*Top Figure:*) A comparison of the cumulative rewards of our methods (BNN-UCB-LL and BNN-TS-LL) vs. the baselines described in Murphy et. al, described in Section 2. This is on the **train set.** (*Middle figure:*) A comparison of the cumulative rewards of our methods (BNN-UCB-LL and BNN-TS-LL) vs. the baselines described in Murphy et. al, described in Section 2. This is on the **test set.** (*Bottom figure:*) A comparison of the CPU time for 5 (cross-validation) iterations on the *test set.* Note that for the deeper feature extractor MLP2, compute times are generally much higher, as expected.

We saw that Murphy's EKF methods still performed the best out of all considered methods, and outperformed ours by a wide margin. However, we outperformed all other baselines (both hybrid neural-linear methods and LinUCB), and were generally on par with the computational efficiency of other methods. Notably, our UCB variant outperformed our TS variant across the board (this was consistent during the experimentation process). Due to the low-dimensional approximation employed by EKF, its computational efficiency was also on par with the simpler, cruder methods. Interestingly, the feature extractors MLP1 and MLP2 did not impact performance too much for any baseline methods. This helps explain why neural-linear struggles, as FCNN-based feature extraction does not seem particularly useful for this task.

6 Conclusion/Future Work

We view our results as very promising for methods of this nature for recommendation systems. There are several next steps we plan on taking in our continued work on this project. First, we plan to train larger model architectures on the MovieLens1M and the MovieLens25M datasets with more computational power. The relative performance of neural networks over hybrid methods should increase the large-data regime; to this end, the computational times we observed in Figure 3 seem to demonstrate that these methods would scale reasonably (but would still be much slower than the baselines as model complexity increases). We also wish to implement the following more powerful techniques for approximating posteriors of FCNNs: (i) full Laplace approximations, (ii) Kronecker-factored Laplace approximations [12], and (iii) MCMC for full posterior inference on shallow networks. The third point is primarily to understand the fundamental limits of performance for these methods. A final addition that would be interesting would be to derive an online update (or use a Markov chain-like method) for the distribution of our parameters. All EKF subroutines used something of this nature, and adaptively changing the posterior directly at each step seemed to work much better than re-estimating the posterior based on updated parameters.

7 Contributions

Both team members contributed equally to this project.

8 Acknowledgements

We'd like to thank Faris Sbahi for helpful conversations and guidance.

References

[1] Moloud Abdar et al. "A review of uncertainty quantification in deep learning: Techniques, applications and challenges". In: *Information Fusion* 76 (2021), pp. 243–297. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2021.05.008. URL: https://www.sciencedirect.com/science/article/pii/S1566253521001081.

[2] Igor Babuschkin et al. *The DeepMind JAX Ecosystem*. 2020. URL: http://github.com/deepmind.

[3] M. Benjamin Dias et al. "The Value of Personalised Recommender Systems to E-Business: A Case Study". In: *Proceedings of the 2008 ACM Conference on Recommender Systems*. RecSys '08. Lausanne, Switzerland: Association for Computing Machinery, 2008, pp. 291–294. ISBN: 9781605580937. DOI: 10.1145/1454008.1454054. URL: https://doi.org/10.1145/1454008.1454054.

[4] Gerardo Duran-Martin, Aleyna Kara, and Kevin Murphy. "Efficient Online Bayesian Inference for Neural Bandits". In: *arXiv e-prints*, arXiv:2112.00195 (Nov. 2021), arXiv:2112.00195. arXiv: 2112.00195 [cs.LG].

[5] Michael W. Dusenberry et al. "Efficient and Scalable Bayesian Neural Nets with Rank-1 Factors". In: *arXiv e-prints*, arXiv:2005.07186 (May 2020), arXiv:2005.07186. arXiv: 2005.07186 [cs.LG].

[6] Jonathan Heek et al. *Flax: A neural network library and ecosystem for JAX*. Version 0.6.3. 2020. URL: http://github.com/google/flax.

[7] Liang Jiang et al. "A hybrid recommendation model in social media based on deep emotion analysis and multi-source view fusion". In: *Journal of Cloud Computing* 9.1 (Oct. 2020). DOI: 10.1186/s13677-020-00199-2. URL: https://doi.org/10.1186/s13677-020-00199-2.

[8] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. "Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks". In: *arXiv e-prints*, arXiv:2002.10118 (Feb. 2020), arXiv:2002.10118. arXiv: 2002.10118 [stat.ML].

[9] Lihong Li et al. "A Contextual-Bandit Approach to Personalized News Article Recommendation". In: *arXiv e-prints*, arXiv:1003.0146 (Feb. 2010), arXiv:1003.0146. arXiv: 1003.0146 [cs.LG].

[10] Tyler Lu, Dávid Pál, and Martin Pál. "Showing Relevant Ads via Lipschitz Context Multi-Armed Bandits". In: *Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010. URL: http://jmlr.csail.mit.edu/proceedings/papers/v9/lu10a/lu10a.pdf.

[11] Carlos Riquelme, George Tucker, and Jasper Snoek. "Deep Bayesian Bandits Showdown: An Empirical Comparison of Bayesian Deep Networks for Thompson Sampling". In: *arXiv e-prints*, arXiv:1802.09127 (Feb. 2018), arXiv:1802.09127. arXiv: 1802.09127 [stat.ML].

[12] Hippolyt Ritter, Aleksandar Botev, and David Barber. "A Scalable Laplace Approximation for Neural Networks". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=Skdvd2xAZ.

[13] Phonexay Vilakone et al. "An Efficient movie recommendation algorithm based on improved k-clique". In: *Human-centric Computing and Information Sciences* 8.1 (Dec. 2018). DOI: 10.1186/s13673-018-0161-6. URL: https://doi.org/10.1186/s13673-018-0161-6.

[14] Dongruo Zhou, Lihong Li, and Quanquan Gu. "Neural Contextual Bandits with UCB-based Exploration". In: *arXiv e-prints*, arXiv:1911.04462 (Nov. 2019), arXiv:1911.04462. arXiv: 1911.04462 [cs.LG].