Using Deep Value Iteration for contextual Bayesian Multi-Armed Bandits

October 27, 2025

Abstract

We present a deep learning approach for solving contextual Bayesian multi-armed bandit (MAB) problems via value iteration. We model the contextual MAB problem as a Bayesian Markov Decision Process (MDP), and show that value iteration can then be used to find the optimal policy. Since this gives a very large state space, we approximate the value function using deep learning. That way we can find near-optimal policies for arbitrary problem sizes. We test this approach on standard UCI machine learning library problem instances and show that the algorithm performs well on most contexts compared to benchmarks, and has advantages on computation speed. We also show how this approach can be used in a practical scenario to optimize airline seat pricing. Overall, our work provides a flexible foundation for solving the contextual MAB problem.

1 Introduction

The Multi-Armed Bandit (MAB) problem is a fundamental challenge in sequential decision-making. First introduced by [20], it involves an agent that must choose among several arms, each yielding an unknown payoff, to maximize cumulative rewards. Classical approaches — such as epsilon-greedy ([3] provides more insight in the performance of this approach), Upper Confidence Bound (UCB, starting from [13], then described more specifically in [2]), and Thompson Sampling (TS, see [22]) — use heuristic rules to balance exploration and exploitation. Although these methods are computationally efficient, they typically do not guarantee optimal performance.

Optimal policies can be obtained using methods like the Gittins index (see for instance [6]), which can be computationally fast, but only works for regular, i.e. without context, MAB problems. With contextual MAB problems, we mean MAB problem instances where a context exists that influences the pay-off distribution. Recently, many approaches have been proposed using deep learning, often basing the underlying principle on the classical approaches of either UCB or TS. In this paper, we use value iteration instead as an underlying principle, which in contrast to the Gittins index is easily extendable to contextual MAB problems. We show that this approach works well and has advantages on computational power needed.

Vast literature exists about the MAB problem. We split our review over first more theoretical multi-armed bandit approaches, not necessarily taking context into account, and then the contextual case.

1.1 Multi-armed bandit background

[11] use a value iteration approach to solve the multi-armed bandit problem using the Gittins index (see [6], [8] and [7] for more details) to compute the optimal strategy. Another interesting case is in [18], which optimally solves the Bernoulli bandit problem through value iteration in a Bayesian manner, making use of a Beta-distribution as prior for the pay-off per arm, which is similar to our approach. In both these approaches, limits on number of arms and time are practically needed, and context cannot be incorporated, to make the solution computationally tractable, which is not the case in our approach.

1.2 Contextual Bandit Background

The literature on contextual MAB spans a diverse range of approaches with many contributions in recent years. A good starting point is [19], which provides both an extensive overview of approaches to solve the contextual MAB problem, and a thorough benchmark of deep learning bandit algorithms. Many methods apply some form of either UCB or TS to address the contextual MAB problem as well, and then focus on other interesting elements. For example, [21] employ the full context when computing regret - the difference between the optimal choice and the choice made by the algorithm, whereas our focus is solely on the current context available at decision time. In recent work, [25] and [12] derive theoretical bounds for contextual TS by adapting the sampling procedure to the context and incorporating an additional optimistic bonus to enhance performance, a sort of combination of UCB and TS. Other studies, such as [15], consider adversarial bandits under linear assumptions, so a problem where the reward is a linear function of the context-action pair, while [9] reduce the stochastic contextual linear bandit problem to a regular linear bandit to achieve competitive regret bounds. A linear bandit problem is a problem where the reward is a linear function of the action, and a contextual linear function one where the reward is a linear function of the context and action. Meanwhile, [27] propose an approach that bypasses complex posterior calculations by using frequentist sample mean and variance (so sample average uncertainty measures) in conjunction with deep learning to incorporate context. [5] combine Kalman filtering with a low-dimensional parameter subspace. Alternatively, [17] use tree ensembles instead of neural networks to incorporate context, also using UCB and Thompson-Sampling approaches to balance exploration and exploitation.

1.3 Our contribution

In our forthcoming paper X we provide a theoretical approach that shows how value iteration can be used as a principle to solve contextual MAB problems, however this calculation suffers from the curse of dimensionality. So to also practically make use of that approach, an approximation method is needed. In this paper, we briefly introduce the concept of value iteration to solve the contextual MAB problem as a background. Here we mostly focus on how this can practically be implemented using deep learning. We construct a network that closely follows the theoretical approach to solve the contextual MAB problem with value iteration. In that way, arbitrary problem sizes can be solved. This means that our Deep Value Iteration (DVI) algorithm can solve the generic contextual MAB problem of arbitrary problem size to near optimality. In our experiments, we compare our Deep Value Iteration (DVI) algorithm against various standard benchmark algorithms, and show that the approach performs competitively on standard benchmarks. Besides, we show practical results in an airline seat pricing setting, demonstrating the algorithms practical value.

2 Value Iteration in the contextual Bayesian MAB Setting

In this section we provide a brief overview of the value iteration approach we use conceptually, which will define our algorithm. *In our forthcoming paper X*, we theoretically develop the approach, have shown that this approach works, and can practically be implemented on small problem instances. So for complete detail, we refer the reader there.

2.1 Problem Setup and Definitions

We consider a contextual MAB-problem, which we model in a Bayesian manner. We define the current state of the problem as the current parameter estimates for the pay-off distribution. Using such a definition, we can model the state after choosing an action as the posterior distribution for the pay-off distribution based on the observed reward. With that, we have a fully defined Markov Decision Process (MDP) and we can apply value iteration to solve the MDP. More specifically, we define

- $\mathcal{A}$ a finite set of arms, or actions.
- $\mathcal{S}$ the state space, where each state $s \in \mathcal{S}$ encodes the parameters of the current parameter estimates for the pay-off distribution.
- $\mathcal{C}$ the context space
- $\mathcal{X}$ the augmented state space, combining the state estimates for the arms and the context, so $\mathcal{X} = \mathcal{S} \times \mathcal{C}$, with a specific state represented as (s, c).
- $R : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ a bounded reward function.
- $P(s', c' \mid (s, c), a)$ the transition probability from state (s, c) to (s', c') when action a is taken, as determined by the Bayesian update.
- $\gamma \in (0, 1)$ the discount factor.

The value function is then defined by

$$V(s, c) = \max_{a \in \mathcal{A}} \left\{ R(s, c, a) + \gamma \, \mathbb{E} \left[V(s', c') \mid s, c, a \right] \right\},$$

2.2 Value Iteration to solve the MAB problem

Using this definition, we can use value iteration to find the value function, and then find the optimal policy for contextual MAB problems.

To show that the value iteration approach works, there are two things we need. First, it is necessary that the conditions that make value iteration lead to the optimal policy also apply to this problem definition. Second, in case we do not know the actual distribution parameters but need to learn them, it is necessary that the estimates for the distribution parameters converge to their true values so that the first condition applies.

We define a contextual MAB problem as a Bayesian MDP with a state $x = (s, c)$, consisting of the belief s over the arms' pay-off distribution parameters, and the context c. If we have a finite action set, bounded rewards and a discount factor $\gamma < 1$, the Bellman operator

$$(TV)(s,c) = \max_{a \in A} \{R(s,c,a) + \gamma \mathbb{E}\left[V(x')\right]\}$$

is then a $\gamma-$contraction in the supremum norm. This is the same property as used in regular value iteration, and it guarantees the existence and uniqueness of the value function V^*, which follows from the conditions established in [23]. So the policy derived from V^* is optimal for the contextual MAB, just as in the standard context-free setting.

For the second condition, we need to make sure that the online estimates for the transition probabilities that we use in our algorithm approach their true values. That first means that the transition probabilities $P(s', c' \mid (s,c), a)$ need to be stationary. Furthermore, to have correct estimates for all states $x \in X$ it means that all states need to be visited 'often enough' to have reliable estimates for the transition and pay-off probabilities. Next to that, the priors selected need to assign positive mass to the true parameters.

With this, the approach works, and can be used to derive an optimal policy. However, the computational load quickly grows as the number of dimensions grow, and for most practical problem instances, the value function cannot be analytically computed.

3 Deep Learning approach for practical implementation

As it can be computationally expensive for large state or context spaces to compute the full value function, we propose using a deep learning approach to approximate the value function $V(s, c)$.

3.1 Network architecture

We approximate the state value $V(s, c)$ with a dual–stream network that separates (i) information from arm uncertainty and (ii) context signals. The two streams produce compact summaries that are fused nonlinearly to yield the final value estimate. This decoupling lets us pretrain and freeze the base (uncertainty) computation offline, while adapting only the context-dependent part online.

3.1.1 Input and notation

A state s consists of per–arm uncertainty descriptors $\{u_i\}_{i=1}^K$ (distribution parameters for each arm) and a normalized context vector $\tilde{c} \in \mathbb{R}^{d_c}$. Let $E(\cdot)$ denote a linear embedding of arm parameters, $\mathrm{Agg}(\cdot)$ an order–invariant aggregator over arms (we use the mean), and $\mathrm{MLP}_\star(\cdot)$ small multilayer perceptrons.

3.1.2 Components

Base stream (arm uncertainty) We embed each arm and aggregate:

$$e_i = E(u_i) \in \mathbb{R}^{d_e}, \qquad \bar{e} = \mathrm{Agg}\big(\{e_i\}_{i=1}^K\big) = \tfrac{1}{K}\sum_{i=1}^K e_i.$$

An MLP maps $\bar{e}$ to a base value summary $v_{\mathrm{base}} = \mathrm{MLP}_{\mathrm{base}}(\bar{e})$.

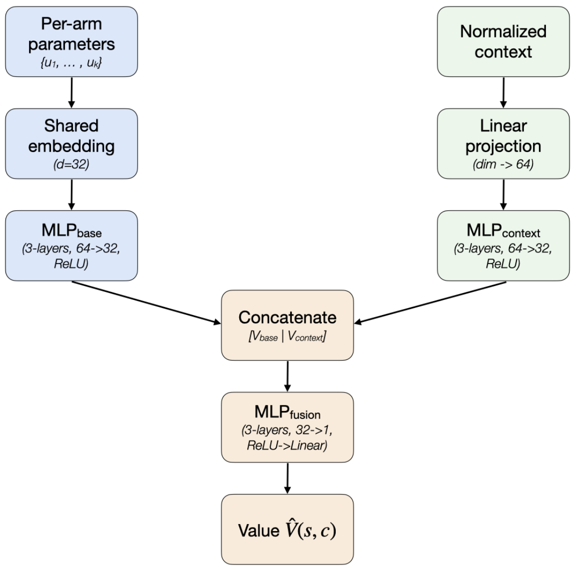

Figure 1: Dual–stream value network: (left) base stream on per–arm uncertainty, (right) context stream; outputs are fused to predict $V(s, c)$.

Context stream We project the context and pass it through its own MLP:

$$z = W_c\tilde{c}, \qquad v_{\text{ctx}} = \text{MLP}_{\text{ctx}}(z).$$

Fusion head We concatenate the two summaries and predict the value via a fusion MLP:

$$\hat{V}(s, c) \;=\; \text{MLP}_{\text{fusion}}\big(\,[v_{\text{base}} \,\|\, v_{\text{ctx}}]\,\big)\,.$$

A schematic is shown in Figure 1.

3.1.3 Training protocol (offline → online)

Offline pretraining We sample synthetic states with zero context to learn the base value surface and train the base stream + fusion head. After convergence, we freeze the base stream parameters to prevent context–induced drift in uncertainty processing.

Online adaptation We warm–start from the offline weights and continue training using actual transitions (s_t, a_t, r_t, s_{t+1}), updating only the context stream and fusion head. We employ experience replay with mini–batches for sample efficiency and stable bootstrapping. No explicit exploration heuristic is added; the greedy policy w.r.t. the learned value captures the value of information induced by the belief–MDP dynamics.

Stabilization and efficiency We use: (i) *experience replay* (FIFO buffer), (ii) *mini–batch* updates, (iii) input *normalization* to zero mean/unit variance, (iv) *gradient clipping* (global norm

≤ 1.0), and (v) Smooth L1 (Huber) loss for robust targets. To emphasize informative regions, we apply *stratified sampling*: 70% of sampled states for training are drawn from the lower 50% of the uncertainty parameter ranges (higher variance zones), the remainder uniformly.

3.1.4 Hyperparameters

Arm-parameter embedding size	32
Hidden size per linear layer	64
Activation	ReLU
Final hidden (fusion head)	32
Optimizer	Adam (lr = 10^{-3}, weight decay = 10^{-5})
Loss	Smooth L1 (Huber, δ=1)
Replay buffer size	1000
Online mini–batch size	32

3.2 Full algorithm

A full version of the algorithm can be seen in algorithm 1.

Algorithm 1 Deep Learning Approximate Value Iteration (Generic)

Require:
- A set of possible actions $A = \{a_1, \ldots, a_m\}$
- A function NextStateDistribution(s, a) returning possible next states and associated probabilities
- A neural network V_θ initialized with random parameters
- Number of training iterations N, batch size B, discount factor γ, scaling factor c
- Procedure GenerateSampleStates(M) that returns a set of M states

```
1: function TrainValueFunction
2:     sampleStates ← GenerateSampleStates($M$)
3:     Compute sample $\mu$ and $\sigma$ from sampleStates          ▷ mean, stdev for normalization
4:     for iter ← 1 to $N$ do
5:         batchIdx ← random subset of size $B$ from $\{1, \ldots, M\}$
6:         batchStates ← sampleStates[batchIdx]
7:         batchTargets ← []
8:         for each state $s$ in batchStates do
9:             bestValue ← $-\infty$
10:            for each action $a \in A$ do
11:                $r$ ← ImmediateReward$(s, a)$          ▷ Domain-specific reward
12:                $v_{\text{future}} \leftarrow 0$
13:                for each $(s', p_{s'})$ in NextStateDistribution$(s, a)$ do
14:                    $v_{\text{future}} \mathrel{+}= p_{s'} \times V_\theta(\text{Normalize}(s', \mu, \sigma))$
15:                end for
16:                actionValue ← $r + \gamma \cdot v_{\text{future}}$
17:                if actionValue > bestValue then
18:                    bestValue ← actionValue
19:                end if
20:            end for
21:            batchTargets ← batchTargets ∪ {bestValue/$c$}          ▷ Scale down for stability
22:        end for
23:        $\hat{X}$ ← Normalize(batchStates, $\mu, \sigma$)
24:        $\hat{y}$ ← valueNet($\hat{X}$)          ▷ Forward pass
25:        $\ell$ ← Loss($\hat{y}$, batchTargets)          ▷ e.g. Smooth L1 or MSE
26:        BackpropAndUpdate($\ell, V_\theta$)          ▷ Optimizer step with gradient clipping
27:    end for
28:    return $V_\theta$          ▷ Trained approximate value function
29: end function
```

3.3 Sensitivity to input and design

Next to overall results, an analysis was done on algorithm design and prior specification.

3.3.1 Synthetic Contextual Bandit Environment (for sensitivity tests)

For this, we run a synthetic MAB benchmark, instead of one of the problems from Section A. This is to test the results on stochastic outcomes combined with context, instead of the transformed classification problems where the outcomes are deterministic. At each round t:

- A context $x_t \in \mathbb{R}^d$ is drawn i.i.d. from a standard normal, $x_t \sim \mathcal{N}(0, I_d)$.
- The learner chooses an arm $a_t \in \{1, \dots, K\}$.
- The (noise–free) mean reward for arm a in context x is

  $$\mu(a, x) \;=\; b(a) \;+\; w_a^\top x,$$

  where $b(a)$ is a deterministic base payoff and $w_a \in \mathbb{R}^d$ is an arm–specific context weight.
- The observed reward is real–valued,

  $$r_t \;=\; \mu(a_t, x_t) + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, \sigma^2),$$

  with $\sigma > 0$ controlling the noise level.

We consider three base-payoff families to vary global curvature across arms:

$$b_{\text{linear}}(a) = a, \qquad b_{\text{quadratic}}(a) = a^2, \qquad b_{\text{exponential}}(a) = \exp(a/K),$$

where $a \in \{0, \dots, K-1\}$ and the exponential form is scaled to avoid numerical blowup. Arm weights are sampled once at initialization as $w_a \sim \mathcal{N}(0, I_d)$ and kept fixed throughout a run. The tuple $(K, d, \sigma, \text{structure})$ controls difficulty: more arms or higher noise increase exploration demands; larger d increases contextual estimation burden; the `structure`$\in$ {linear, quadratic, exponential} toggles nonlinearity in $b(\cdot)$.

Given a context x, the expected rewards for all arms are

$$\mu(\cdot, x) \;=\; \big(b(0), \dots, b(K-1)\big)^\top + Wx,$$

with $W = [w_0^\top; \dots; w_{K-1}^\top] \in \mathbb{R}^{K \times d}$. The contextual oracle chooses $a^\star(x) = \arg\max_a \mu(a, x)$. We report cumulative regret $\sum_{t=1}^T \big(\mu(a^\star(x_t), x_t) - \mu(a_t, x_t)\big)$. A wider set of parameters was tested, but results displayed here are for:

Number of arms: 8

Number of context dimensions: 4

Structure: Quadratic

Noise standard deviation: 0.5

First, a comparison of the impact of network design is shown in figure 2. For this, the network design differences are:

Deep Value Iteration The network described in 3.1.

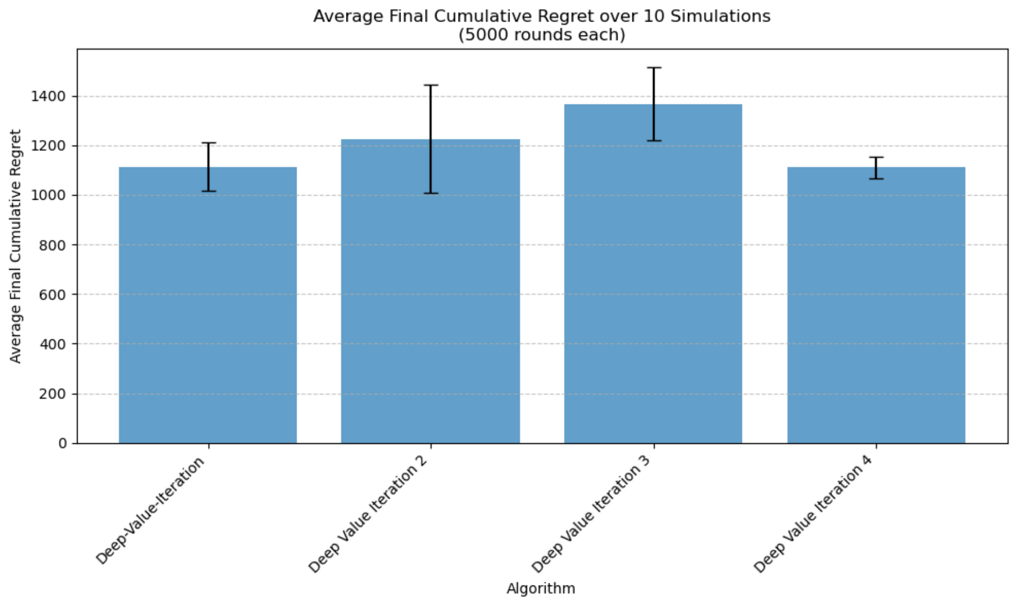

Figure 2: Results overview of comparison of various network designs.

Deep Value Iteration 2 The same network, but then with a larger number of nodes in the network, 256 per layer instead of 64.

Deep Value Iteration 3 The same network, but next to increased network size (to 256), the value network parameters are not frozen during the online phase, but are also re-trained.

Deep Value Iteration 4 The same network architecture as described in this paper, but instead of using a neural network to learn the context corrections, we use Gradient Boosting.

As can be seen in figure 2, a larger network does not lead to better results. However, gradient boosting for the context corrections performs on-par with the standard network, but has a slightly smaller standard deviation in results.

Next to that, a comparison is made to see the impact of prior misspecification. To show the effect, we compare against Q-Learning as a very simple baseline comparison. We compare a Beta prior-Bernoulli posterior with a Gamma prior-Negative Binomial posterior, for the same synthetic MAB problem. Figure 3 shows the effect.

From figure 3 it is directly obvious that the prior choice is extremely important. An incorrect prior can completely break down the performance.

3.4 Results on a Logistic Contextual Bandit simulation

We first evaluate Deep Value Iteration (DVI) on a synthetic but standardizable contextual bandit in which the dominant statistical challenge is learning arm payoffs under uncertainty, rather than pure context classification. For a more standard benchmark used in more papers, we refer to A.

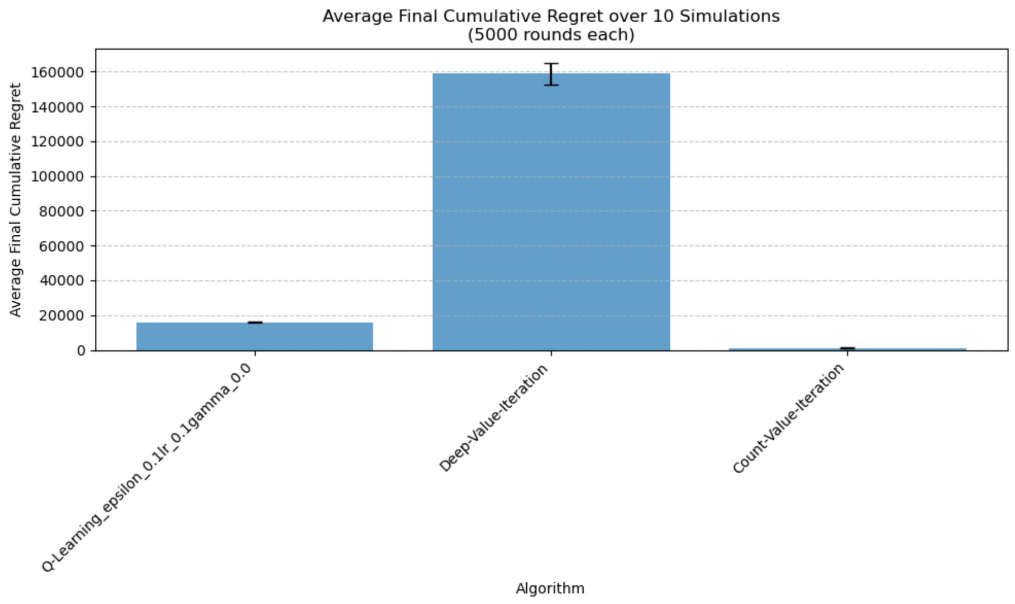

Figure 3: Results overview of an incorrect prior.

However, we focus mainly on this simulation and a practical pricing example in 3.5, because the standard benchmarks used in A focus more on context classification rather than arm uncertainty.

Here, at each round t, a d–dimensional context $x_t \sim \mathcal{N}(0, I_d)$ is observed and an arm $a_t \in \{1, \dots, K\}$ is chosen. The Bernoulli reward obeys a logistic GLM with a shared slope and per–arm intercepts:

$$\Pr(r_t{=}1 \mid x_t, a_t{=}a) \;=\; \sigma\big(\alpha_a + x_t^\top \beta\big), \qquad \sigma(z) = \tfrac{1}{1+e^{-z}}.$$

Initialization follows a fixed generative recipe (same across methods): arm effects $\alpha_a \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma_\alpha^2)$ with $\sigma_\alpha = 1.0$, and a shared slope β drawn from $\mathcal{N}(0, I_d)$ and then rescaled to $\|\beta\|_2 = 0.4\,\sigma_\alpha$. Because $x \sim \mathcal{N}(0, I_d)$, the contextual term $x^\top \beta$ is $\mathcal{N}(0, \|\beta\|_2^2)$ with standard deviation 0.4, i.e., context contributes non-trivially, but is smaller than the intercept variability. This makes the problem arm–centric (learning the unknown $\{\alpha_a\}$ is crucial), while still allowing the optimal arm to depend on x.

Setup and metrics We use $K{=}20$ arms and $d{=}10$ context dimensions. Each algorithm is run for $T{=}5000$ rounds and averaged over 50 independent seeds. Performance is reported as cumulative regret against a contextual oracle that knows (α, β) and plays $a_t^\star(x_t) = \arg\max_a \sigma(\alpha_a + x_t^\top \beta)$ at every round. We also report end-to-end wall-clock time.

Compared methods We compare DVI to strong contextual bandit baselines: LinUCB, LinTS, Tree-UCB, and a bootstrap-based exploration method (BOE). For linear/GLM methods we supply the hybrid feature map $\phi(x, a) = [\,x;\, e_a\,]$, so that all baselines can represent per–arm fixed effects (intercepts) alongside the shared slope on x.

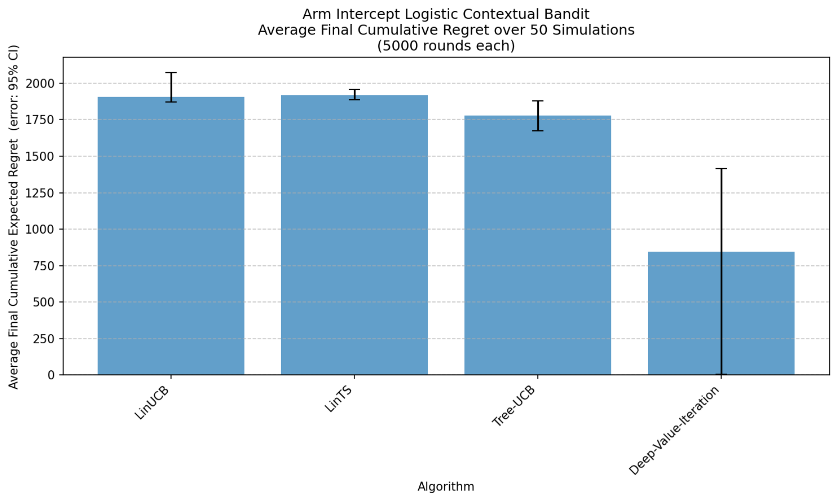

Figure 4: Contextual Bandit (K=20, d=10): average final cumulative regret over 50 runs (T=5000 rounds each). Error bars show $\pm$ one standard deviation.

Algorithm	Avg. Regret ↓	Std. (over 50 runs)	Wall-clock (s)
LinUCB	1901.06	59.53	19.47
LinTS	1913.94	37.22	27.88
Tree-UCB	1795.72	64.58	970.60
BOE	1914.59	79.71	20.28
Deep Value Iteration	**650.79**	651.92	1136.75

Table 1: Benchmark (K=20, d=10, T=5000). Deep Value Iteration (DVI) achieves the lowest average cumulative regret. Higher variance across seeds reflects nonconvex training; means are computed over 50 independent runs.

Results Figure 4 summarizes average final cumulative regret; Table 1 lists means, standard deviations, and wall-clock time. DVI attains substantially lower regret than the baselines (about 3$\times$ improvement on average), demonstrating that planning with a value function over belief states can efficiently trade off exploration and exploitation in this arm-uncertainty regime. DVI's across-seed variability is larger, which can follow from the nonconvex value-function training and approximate planning, but the mean performance advantage is clear. Runtime reflects the additional planning/learning cost (here measured only online).

3.5 Price optimization application

We can use this deep learning approach to optimize the pricing of paid seats in an airline context. We use

State s as the distribution for the number of seats that will be sold with action a.

Action a as the price that is chosen.

Reward $r(s, a)$ as the revenue obtained, which is defined as $s \cdot a$.

For a full description of the problem set-up, the simulation model and more background, where also full code can be found, see [4].

To model the state distribution, we use a Beta prior distribution and a negative binomial posterior, allowing updating the posterior analytically. The negative binomial prior fits well with the number of seats sold, since it's a non-negative discrete distribution. Because of those two properties, it's a practical choice, even though actual airline seat demand might deviate from negative binomial. We assume that in this case, we can also learn about other actions (so, prices) if we select one. We explicitly assume that if a certain number of seats sell at a given price level x, at every lower price $p < x$, at least the same number of seats would have also been sold. Conversely, if no seats are sold at price level x, we assume no seats would sell at any higher price $p > x$. This is also a practical assumption. On the one hand, we didn't want to impose too much structure to keep the experiment in a simulation model generic, and this seems at least reasonable. At the same time, not assuming anything about other prices, would leave information available unused. We could easily adjust the algorithm to have a stronger, or less strong, assumption on this and we simply consider this a practical choice.

To have a fair comparison, in this case we also use Thompson Sampling and Q-Learning to benchmark against. Q-Learning is a very straightforward and simple to understand algorithm as a simple baseline, where here we use $\epsilon = 0.05$, $\gamma = 0.95$ and learning rate $= 0.95$. Thompson Sampling is in practice often a good algorithm, with limited parameter tuning. To transform this into a bandit problem, we allow 64 price levels (or arms). We use a simulation model for seat pricing to test performance of the algorithms, see the appendix in [4] for exact detail of how the simulation model is set up. Results of this can be seen in figure 5.

3.6 Computation times

On regular problem instances like those from the UCI machine learning library, training the network offline to approximate the value function usually takes about 5-10 seconds on a regular 2021 Macbook Pro, while running the algorithms on problem instances for 5000 rounds usually takes about 30-60 seconds. Memory usage is no issue due to the limited network sizes used. With that, the algorithms are easily tractable on any modern hardware. In contrast, running Neural Thompson Sampling on the same device, computation times are magnitudes of order more as can be seen in figure 8. Furthermore, with Neural Thompson Sampling, computation times grow very quickly when problem sizes get larger as can be seen from the same figure. Leaving out Neural Thompson Sampling to have a better view on the remaining algorithm, run times can be seen in figure 6. Deep Value Iteration strikes a good balance between the best performance in terms of regret, and the best performance in terms of computation times. Furthermore, computation times are relatively stable in terms of problem size, while other algorithms increase computation times more with increasing problem sizes.

3.7 Discussion

Experimental evaluation shows that our Deep Value Iteration (DVI) algorithm is capable of solving standard problem instances mostly on par with benchmark algorithms. NeuralTS and TreeUCB are in general outperforming the DVI algorithm, however, at much higher computational cost, while DVI shows to be relatively stable in computation times. Furthermore, our approach is very flexible

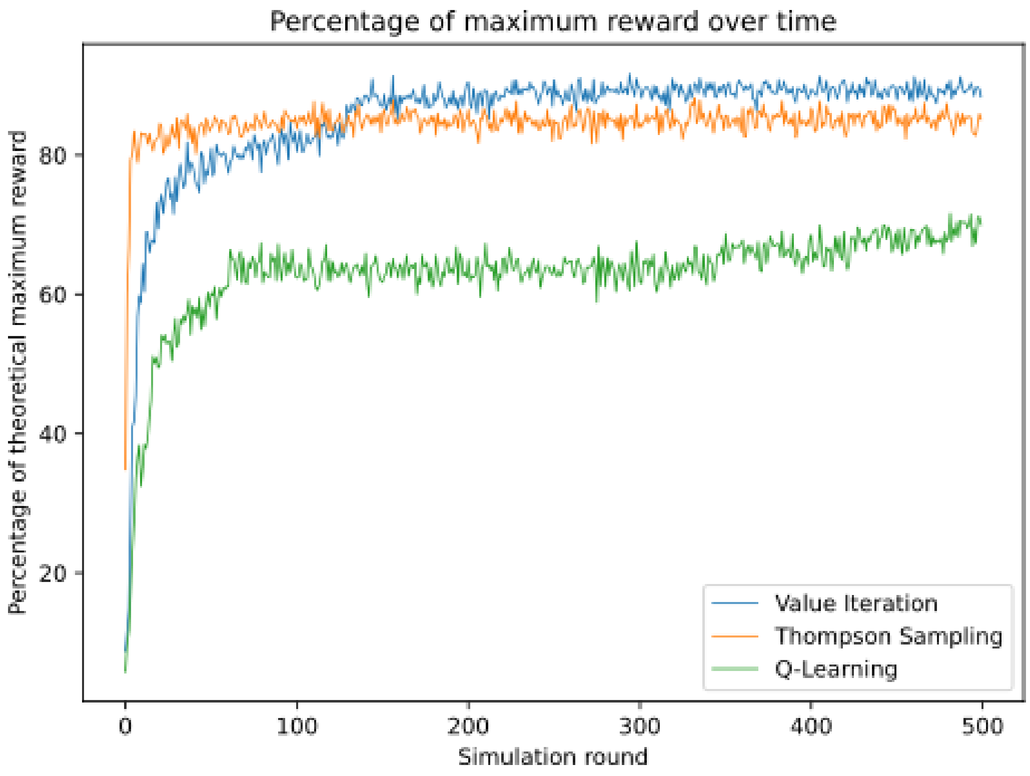

Figure 5: Average percentage of theoretical maximum award for the various algorithms in a seat pricing context

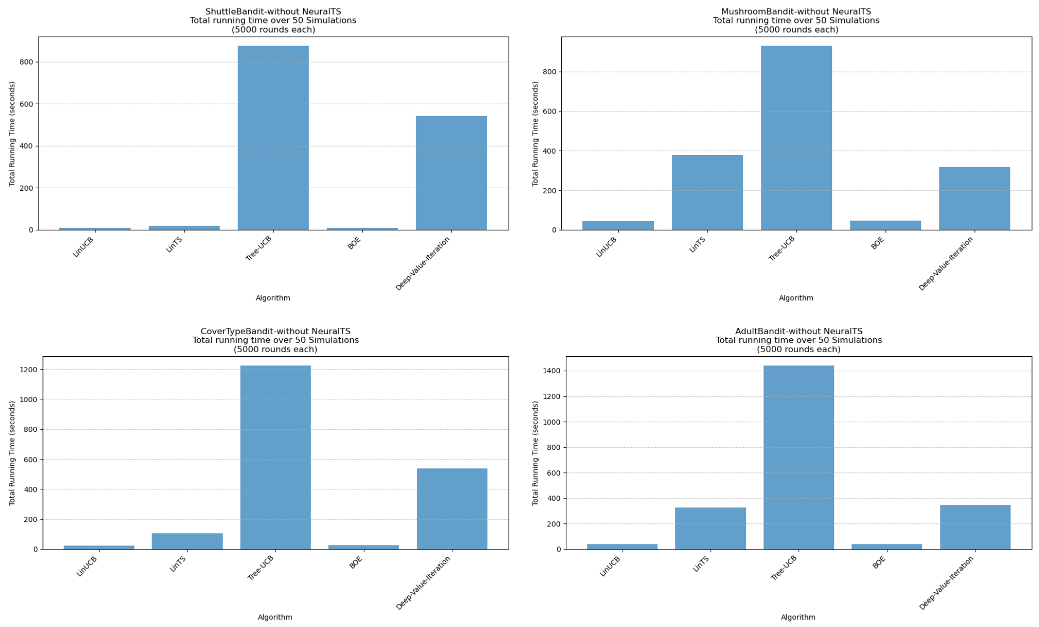

Figure 6: Total running time of all algorithms on the Shuttle, Mushroom, CoverType and Adult bandit problems without Neural Thompson Sampling.

and if more context knowledge is available, it is easy to incorporate this into the modeling of the context influence. In that way, we provide a good baseline that can be used to solve more specific problem instances, as we have shown in the price optimization example.

4 Conclusion

We have presented a theoretical framework for applying value iteration to Bayesian contextual multi-armed bandit problems, which provides a flexible basis to solve the contextual MAB problem. To address the downside of this approach, the curse of dimensionality, we use deep learning to approximate the exact analytical value function. We can then also use this approach to incorporate context in the same deep learning network. Our approach usually matches benchmark algorithms while reducing computational and memory demands. This efficiency makes the method suitable for extensions and practical applications in more complex scenarios. The context can of course be very dependent in a contextual MAB-problem, so how to model the value of the context in a completely generic setting is complicated. This approach gives a flexible approach to do so.

Overall, our work lays a theoretical and practical foundation for addressing contextual MAB problems. It would be interesting to further research even more generic scenarios like non-stationary reward distributions, or how to incorporate partial observability.

References

[1] Shipra Agrawal and Navin Goyal. "Thompson sampling for contextual bandits with linear payoffs". In: *International conference on machine learning*. PMLR. 2013, pp. 127–135.

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem". In: *Machine learning* 47 (2002), pp. 235–256.

[3] Nicolo Cesa-Bianchi and Paul Fischer. "Finite-time regret bounds for the multiarmed bandit problem." In: *ICML*. Vol. 98. Citeseer. 1998, pp. 100–108.

[4] Kevin Duijndam, Ger Koole, and Rob van der Mei. "Bayesian reinforcement learning to optimize paid ancillary revenue in the airline industry". In: *Journal of Revenue and Pricing Management* (2025), pp. 1–17.

[5] Gerardo Duran-Martin, Aleyna Kara, and Kevin Murphy. "Efficient online bayesian inference for neural bandits". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2022, pp. 6002–6021.

[6] John Gittins. "A dynamic allocation index for the sequential design of experiments". In: *Progress in statistics* (1974), pp. 241–266.

[7] John Gittins, Kevin Glazebrook, and Richard Weber. *Multi-armed bandit allocation indices*. John Wiley & Sons, 2011.

[8] John C Gittins. "Bandit processes and dynamic allocation indices". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 41.2 (1979), pp. 148–164.

[9] Osama A Hanna, Lin Yang, and Christina Fragouli. "Contexts can be cheap: Solving stochastic contextual bandits with linear bandit algorithms". In: *The Thirty Sixth Annual Conference on Learning Theory*. PMLR. 2023, pp. 1791–1821.

[10] Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. *UCI Machine Learning Repository*. 2017. URL: `http://archive.ics.uci.edu`.

[11] Vikram Krishnamurthy, Bo Wahlberg, and Frank Lingelbach. "A value iteration algorithm for partially observed markov decision process multi-armed bandits". In: *Math. of Oper. Res* (2005), pp. 133–152.

[12] Yuko Kuroki et al. "Best-of-Both-Worlds Algorithms for Linear Contextual Bandits". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2024, pp. 1216–1224.

[13] Tze Leung Lai and Herbert Robbins. "Asymptotically efficient adaptive allocation rules". In: *Advances in applied mathematics* 6.1 (1985), pp. 4–22.

[14] Lihong Li et al. "A contextual-bandit approach to personalized news article recommendation". In: *Proceedings of the 19th international conference on World wide web*. 2010, pp. 661–670.

[15] Haolin Liu, Chen-Yu Wei, and Julian Zimmert. "Bypassing the simulator: Near-optimal adversarial linear contextual bandits". In: *Advances in Neural Information Processing Systems* 36 (2024).

[16] Benedict C May et al. "Optimistic Bayesian sampling in contextual-bandit problems". In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 2069–2106.

[17] Hannes Nilsson et al. "Tree ensembles for contextual bandits". In: *arXiv preprint arXiv:2402.06963* (2024).

[18] Sebastian Pilarski, Slawomir Pilarski, and Dániel Varró. "Optimal policy for Bernoulli bandits: Computation and algorithm gauge". In: *IEEE Transactions on Artificial Intelligence* 2.1 (2021), pp. 2–17.

[19] Carlos Riquelme, George Tucker, and Jasper Snoek. "Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling". In: *arXiv preprint arXiv:1802.09127* (2018).

[20] Herbert Robbins. "Some aspects of the sequential design of experiments". In: (1952).

[21] Jon Schneider and Julian Zimmert. "Optimal cross-learning for contextual bandits with unknown context distributions". In: *Advances in Neural Information Processing Systems* 36 (2024).

[22] William R Thompson. "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples". In: *Biometrika* 25.3-4 (1933), pp. 285–294.

[23] JN Tsitsiklis and B Van Roy. "An analysis of temporal-difference learning with function approximation". In: *IEEE Transactions on Automatic Control* 42.5 (1997), pp. 674–690.

[24] Tianyu Wang et al. "Towards practical lipschitz bandits". In: *Proceedings of the 2020 ACM-IMS on Foundations of Data Science Conference*. 2020, pp. 129–138.

[25] Ruitu Xu, Yifei Min, and Tianhao Wang. "Noise-adaptive thompson sampling for linear contextual bandits". In: *Advances in Neural Information Processing Systems* 36 (2024).

[26] Weitong Zhang et al. "Neural thompson sampling". In: *arXiv preprint arXiv:2010.00827* (2020).

[27] Rong Zhu and Mattia Rigotti. "Deep bandits show-off: Simple and efficient exploration with deep networks". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 17592–17603.

A Standard MAB problem comparison

Coming from [19], and also used in many other papers, we use the same comparison against some of the UCI machine learning problems (at [10]), turned into contextual MAB instances. Like in other papers, the classification task is reinterpreted as a contextual bandit: at each round the algorithm is given a context (so, the feature vector), chooses one of the arms (so, the classification label) and receives a binary reward indicating whether the label was correct. We test against four problem instances. In the Shuttle bandit problem, from the NASA Statlog Shuttle dataset (from [10]), the context is a 9-dimensional vector of sensor readings, and there are 7 arms corresponding to shuttle fault classes. The Mushroom bandit problem (from [10]) is the task to decide whether a mushroom is edible or not. Based on a context of 112 binary features about a mushroom, the algorithm needs to decide whether to eat the mushroom or not. So there are in this case 2 arms. The CoverType bandit problem (from [10]) uses 54 environmental attributes (e.g. soil composition, elevation) as context, and the algorithm needs to decide on 7 possible forest cover types. The Adult bandit problem (from [10]) has 94 context variables describing features of a census, where the action is to decide for a high-income or low-income, so 2 potential outcomes. In short:

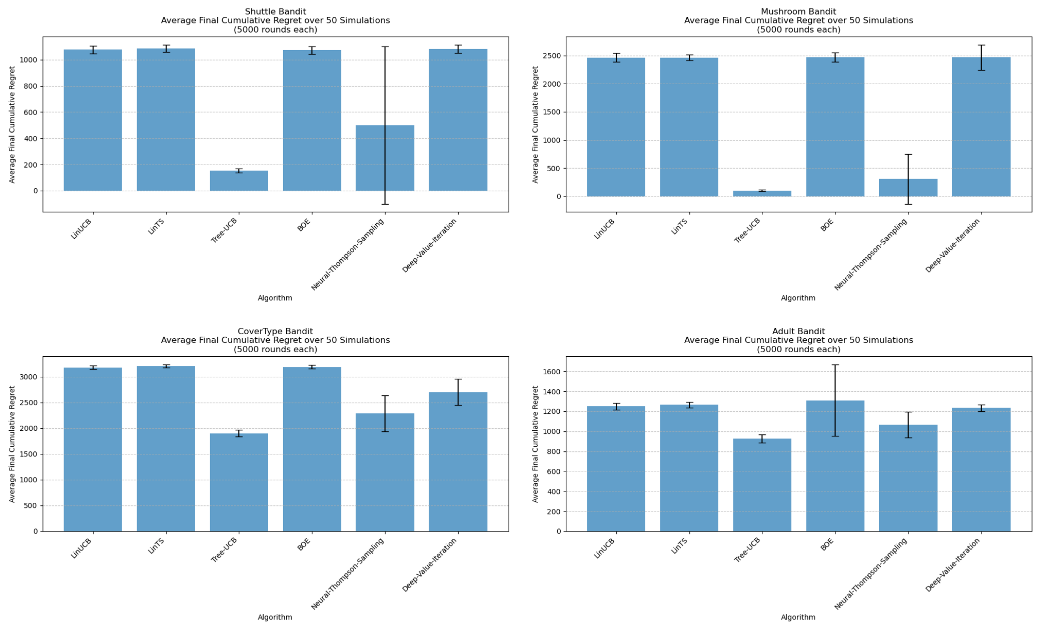

Figure 7: Average regret and standard deviation compared to benchmark algorithms on the Shuttle, Mushroom, CoverType and Adult bandit problems.

Environment	Context Dimensions	Number of arms	Reward
Shuttle Bandit	9	7	1 if correct, 0 otherwise
Mushroom Bandit	112	2	1 if correct, 0 otherwise
CoverType Bandit	54	7	1 if correct, 0 otherwise
Adult Bandit	94	2	1 if correct, 0 otherwise

Note that these problem instances are popular and therefore we also benchmark against them. However, they rely heavily on context and lack stochasticity in the reward pay-off. This is inherent in how the problems are created, as they are actually classification problems transformed into a bandit problem. Therefore, they are a good benchmark whether an algorithm is well able to learn mapping high-dimensional inputs to labels for the context-part. However, they are less relevant to benchmark the ability to handle stochasticity in pay-off and inherent structure in that, and with that balancing exploration versus exploitation.

Like for instance [5], we first provide the algorithms with 100 warm-up samples to learn the relation between context, action and reward, and then test the algorithms for 5000 rounds. To benchmark the algorithms, we then run this approach 50 times, and average the regret after the 5000 rounds. We then benchmark the algorithm against LinUCB (see [14] for more detail), Linear Thompson Sampling (see for instance [1]), TreeUCB (see [24] for more detail), Bayesian Optimistic Exploration (BOE, see for instance [16]) and Neural Thompson Sampling (see for instance [26]). Results are in figure 7.

From these results, we find that this approach performs on-par or better than Linear Thompson Sampling, LinearUCB and BOE, but is generally outperformed by TreeUCB and Neural Thomp-

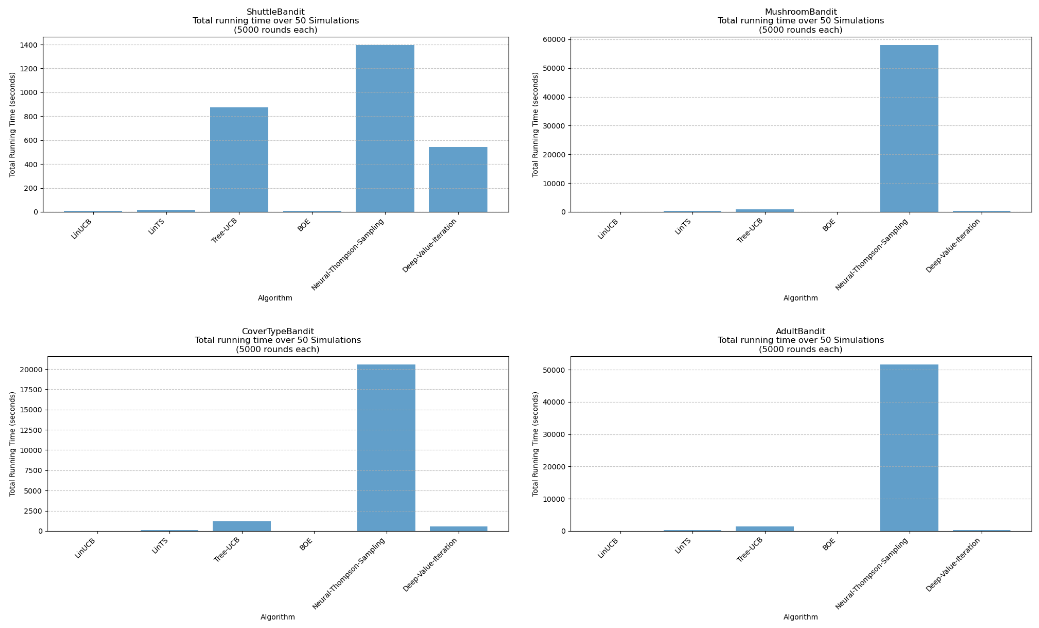

Figure 8: Total running time of all algorithms on the Shuttle, Mushroom, CoverType and Adult bandit problems.

son Sampling on these contexts. We think this comes from more the structure of TreeUCB and Neural Thompson Sampling which are build up more from the context, and have a final element in the algorithm to decide an arm. However, an advantage of the DVI algorithm is the the better computational needs and limited memory footprint compared to other algorithms, as can be seen in figure 8 (see also section 3.6 for more detail).