Appendices For DeepTOP: Deep Threshold-Optimal Policy for MDPs and RMABs

A Threshold Optimal Policy Gradient Theorem Proof for RMABs

Proof. Let $\bar{\rho}_{\lambda_t}(s_i)$ be the distribution that the state at time t is s_i when the initial state is chosen uniformly at random. We have $\bar{\rho}_{\lambda_t}(s_i) = \sum_{t=1}^{\infty} \gamma^{t-1} \bar{\rho}_{t,\lambda}(s_i)$. Given ϕ_i, we number all states in $\mathcal{S}_i$ such that $\mu_i^{\phi_i}(s_i^1) > \mu_i^{\phi_i}(s_i^2) > \dots$. Let $\mathbb{M}^0 = +M$, $\mathbb{M}^n = \mu_i^{\phi_i}(s_i^n)$, for all $1 \le n \le |\mathcal{S}_i|$, and $\mathbb{M}^{|\mathcal{S}_i|+1} = -M$. Also, let $\mathbb{S}_i^n$ be the subset of states $\{s_i | \mu_i^{\phi_i}(s_i) > \mathbb{M}^n\} = \{s_i^1, s_i^2, \dots, s_i^{n-1}\}$. Now, consider the interval $(\mathbb{M}^{n+1}, \mathbb{M}^n)$ for some n. For all $\lambda \in (\mathbb{M}^{n+1}, \mathbb{M}^n)$, $\mathbb{1}(\mu_i^{\phi_i}(s_i) > \lambda) = 1$ if and only if $s_i \in \mathbb{S}_i^{n+1}$. In other words, the threshold policy takes the same action under all $\lambda \in (\mathbb{M}^{n+1}, \mathbb{M}^n)$, and we use $\pi^{n+1}(s_i)$ to denote this policy. We then have

$$\nabla_{\phi_i} K_i(\mu_i^{\phi_i}) = \nabla_{\phi_i} \int_{\lambda=-M}^{\lambda=+M} \sum_{s_i \in \mathcal{S}_i} Q_{i,\lambda}(s_i, \mathbb{1}(\mu_i^{\phi_i}(s_i) > \lambda)) d\lambda = \sum_{s_i \in \mathcal{S}_i} \nabla_{\phi_i} \int_{\lambda=-M}^{\lambda=+M} Q_{i,\lambda}(s_i, \mathbb{1}(\mu_i^{\phi_i}(s_i) > \lambda)) d\lambda$$

$$= \sum_{s_i \in \mathcal{S}_i} \sum_{n=0}^{|\mathcal{S}_i|} \nabla_{\phi_i} \int_{\lambda=\mathbb{M}^{n+1}}^{\lambda=\mathbb{M}^n} Q_{i,\lambda}(s_i, \pi^{n+1}(s_i)) d\lambda$$

$$= \sum_{s_i \in \mathcal{S}_i} \sum_{n=0}^{|\mathcal{S}_i|} \Big(Q_{i,\mathbb{M}^n}(s_i, \pi^{n+1}(s_i)) \nabla_{\phi_i} \mathbb{M}^n - Q_{i,\mathbb{M}^{n+1}}(s_i, \pi^{n+1}(s_i)) \nabla_{\phi_i} \mathbb{M}^{n+1} + \int_{\lambda=\mathbb{M}^{n+1}}^{\lambda=\mathbb{M}^n} \nabla_{\phi_i} Q_{i,\lambda}(s_i, \pi^{n+1}(s_i)) d\lambda \Big), \tag{15}$$

where the summation-integration swap in the first equation follows the Fubini-Tonelli theorem and the last step follows the Leibniz integral rule. We simplify the first two terms in the last step by

$$\begin{aligned} &\sum_{s_i \in \mathcal{S}_i} \sum_{n=0}^{|\mathcal{S}_i|} \Big(Q_{i,\mathbb{M}^n}(s_i, \pi^{n+1}(s_i)) \nabla_{\phi_i} \mathbb{M}^n - Q_{i,\mathbb{M}^{n+1}}(s_i, \pi^{n+1}(s_i)) \nabla_{\phi_i} \mathbb{M}^{n+1} \Big) \\ &= \sum_{s_i \in \mathcal{S}_i} \sum_{n=1}^{|\mathcal{S}_i|} \Big(Q_{i,\mu_i^{\phi_i}(s_i)}(s_i, \mathbb{1}(s_i \in \mathbb{S}_i^{n+1})) - Q_{i,\mu_i^{\phi_i}(s_i)}(s_i, s_i \in \mathbb{S}_i^n) \Big) \nabla_{\phi_i} \mu_i^{\phi_i}(s_i) \\ &= |\mathcal{S}_i| \sum_{s_i \in \mathcal{S}_i} \bar{\rho}_{1,\mu_i^{\phi_i}(s_i)}(s_i) \Big(Q_{i,\mu_i^{\phi_i}(s_i)}(s_i, 1) - Q_{i,\mu_i^{\phi_i}(s_i)}(s_i, 0) \Big) \nabla_{\phi_i} \mu_i^{\phi_i}(s_i). \end{aligned} \tag{16}$$

Next, we expand the last term in (15). Note that $Q_{i,\lambda}(s_i, a_i) = \bar{r}(s_i, a_i) + \gamma \int_{\lambda'=-M}^{\lambda'=+M} \sum_{s_i'} p(s_i'|s_i, a_i) Q_{i,\lambda'}(s_i', \mathbb{1}(\mu_i^{\phi_i}(s_i') > \lambda')) d\lambda'$, where $p(\cdot|\cdot)$ is the transition probability. Hence, $\nabla_{\phi_i} Q_{i,\lambda}(s_i, \mathbb{1}(\mu_i^{\phi_i}(s_i) > \lambda)) = \nabla_{\phi_i} \gamma \int_{\lambda'=-M}^{\lambda'=+M} \sum_{s_i'} p(s_i'|s_i, a_i) Q_{i,\lambda'}(s_i', \mathbb{1}(\mu_i^{\phi_i}(s_i') > \lambda')) d\lambda'$. Using the same techniques in (15) and (16), we have

$$\sum_{s_i \in \mathcal{S}_i} \sum_{n=0}^{|\mathcal{S}_i|} \int_{\lambda=\mathbb{M}^{n+1}}^{\lambda=\mathbb{M}^n} \nabla_{\phi_i} Q_{i,\lambda}(s_i, \pi^{n+1}(s_i)) d\lambda = \sum_{s_i \in \mathcal{S}_i} \int_{\lambda=-M}^{\lambda=+M} \nabla_{\phi_i} Q_{i,\lambda}(s_i, \mathbb{1}(\mu_i^{\phi_i}(s_i) > \lambda)) d\lambda$$

$$= \gamma \sum_{s_i \in \mathcal{S}_i} \int_{\lambda=-M}^{\lambda=+M} \Big(\nabla_{\phi_i} \int_{\lambda'=-M}^{\lambda'=+M} \sum_{s_i' \in \mathcal{S}_i} p(s_i'|s_i, \mathbb{1}(\mu_i^{\phi_i}(s_i) > \lambda)) Q_{i,\lambda'}(s_i', \mathbb{1}(\mu_i^{\phi_i}(s_i') > \lambda')) d\lambda' \Big) d\lambda$$

$$= |\mathcal{S}_i| \sum_{s_i \in \mathcal{S}_i} \gamma \bar{\rho}_{2,\mu_i^{\phi_i}(s_i)}(s_i) \Big(Q_{i,\mu_i^{\phi_i}(s_i)}(s_i, 1) - Q_{i,\mu_i^{\phi_i}(s_i)}(s_i, 0) \Big) \nabla_{\phi_i} \mu_i^{\phi_i}(s_i)$$

$$+ \gamma \sum_{s_i \in \mathcal{S}_i} \int_{\lambda=-M}^{\lambda=+M} \Big(\sum_{s_i' \in \mathcal{S}_i} \int_{\lambda'=-M}^{\lambda'=+M} \nabla_{\phi_i} (p(s_i'|s_i, \mathbb{1}(\mu_i^{\phi_i}(s_i') > \lambda')) Q_{i,\lambda'}(s_i', \mathbb{1}(\mu_i^{\phi_i}(s_i') > \lambda'))) d\lambda' \Big) d\lambda.$$

In the above equation, expanding the last term in time establishes (11). □

B DeepTOP-RMAB Algorithm Pseudocode

Algorithm 2 Deep Threshold Optimal Policy Training for RMABs (DeepTOP-RMAB)

for arm $i = 1, 2, \ldots, N$ **do**
  Randomly select initial parameters for the actor network ϕ_i and critic network θ_i.
  Set target critic network $\theta'_i \leftarrow \theta_i$, and initialize replay memory $\mathcal{M}_i$.
end for
for timestep $t = 1, 2, 3, \ldots$ **do**
  for arm $i = 1, 2, \ldots, N$ **do**
    Receive state $s_{i,t}$ from arm environment $\mathcal{E}_i$, and calculate the state value $\mu_i^{\phi_i}(s_{i,t})$.
  end for
  With probability $1 - \epsilon_t$, activate the V largest-valued arms and keep the remaining arms passive.
  Otherwise, randomly activate V arms with the remaining arms left passive.
  for arm $i = 1, 2, \ldots, N$ **do**
    Observe reward $r_{i,t}$ and next state $s_{i,t+1}$.
    Store transition $\{s_{i,t}, a_{i,t}, r_{i,t}, s_{i,t+1}\}$ in memory $\mathcal{M}_i$.
    Sample a minibatch of B transitions $\{s_{i,t_k}, a_{i,t_k}, r_{i,t_k}, s_{i,t_k+1}\}$, for $1 \leq k \leq B$ from memory $\mathcal{M}_i$.
    Randomly select B values $[\lambda_{i,1}, \lambda_{i,2}, \ldots, \lambda_{i,B}]$, for $1 \leq k \leq B$ from the range $[-M, +M]$.
    Update arm i's critic network using the estimated gradient in Equation (13).
    Update arm i's actor network using the estimated gradient in Equation (14).
    Soft update target critic θ'_i network parameters: $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$.
  end for
end for

C MDP Problems' Description

EV charging. The vector state $v_t = (C_t, D_t)$ consists of the charging requirement C_t, and the time remaining until the vehicle departs the station D_t at time t. In the simulations, we upper-bound the state elements with $C \leq 8$ and $D \leq 12$. The scalar state λ_t is sampled from an Ornstein-Uhlenbeck process with noise parameter 0.15, noise mean 0.0, and noise standard deviation 0.2.

If the agent chooses to charge the vehicle by selecting action $a_t = 1$, the agent then obtains a reward of $1 - \lambda$, and the MDP transitions to the next state $v_{t+1} = (C_t - 1, D_t - 1)$. Otherwise for $a_t = 0$, the reward is zero and the MDP transitions to next state $v_{t+1} = (C_t, D_t - 1)$.

If the charging spot is empty at the next timestep (i.e. $v_{t+1} = (0, 0)$), the environment randomly picks the charge requirement C_{t+1} and time until deadline D_{t+1} of the next EV vehicle. For the vehicle occupying the charging station, if it's charge requirement is not met by the deadline, the agent incurs a penalty of $F(C_t) = 0.2(C_t)^2$ that is subtracted from reward r_t. The net reward is then $r_t - F(C_t)$.

Inventory management. The warehouse can store a maximum of 1000 items, and is able to purchase new stock in bulks of 500 items. The selling price of a single item is set to 20. The vector state v_t is the current market shopping season at time t. The scalar state λ_t is the current warehouse inventory count at time t. We set 10 different shopping seasons indexed by b that model the customers' current demand rate. The corresponding demand rates for the seasons are 10 different Poisson distributions with parameters $\sin(b\pi/10) \cdot 300$ for $0 \leq b \leq 9$.

If the agent orders items (i.e. $a_t = 1$), it receives a reward equal to the total items' selling price minus the minimum of remaining inventory count and current demand rate. The next state v_{t+1} is then the next market season index $v_{t+1} = b + 1 \mod 10$. Otherwise for $a_t = 0$, the agent holds off on buying new items, and incurs a holding cost from the remaining unsold items. The next state is $v_{t+1} = b + 1 \mod 10$.

Make-to-stock production. The environment models a queueing system with m servers serving at rate $1/\mu$ and a finite buffer with size S. There are W customer classes each with Poisson mean arrival rate β. The state at timestep t is (λ_t, v_t), with the scalar state $\lambda_t \in \{0, 1, \ldots, m + S\}$ and the vector state $v_t \in \{1, 2, \ldots, W\}$. If the agent picks the passive action $a_t = 0$, then the reward is equal to the holding

Table 1: Θ values for the recovering bandits' case.

Class	θ_0 Value	θ_1 Value
A	10	0.2
B	8.5	0.4
C	7	0.6
D	5.5	0.8

cost $h(\lambda_t) = -0.1(\lambda_t)^2$. For action $a_t = 1$, the agent receives total net reward of $R_v - h(\lambda_t)$ if the scalar state $\lambda_t = m + S$. Otherwise, the reward is the holding cost $-h(\lambda_t)$.

In the simulations, we set the number of servers $m = 50$, buffer size $S = 50$, number of customer classes $W = 50$, $\mu = 4$, and arrival rate $\beta = 1$ for all customer classes. The reward R_v is chosen to be evenly spaced between 200 and 10 depending on the number of customer classes W.

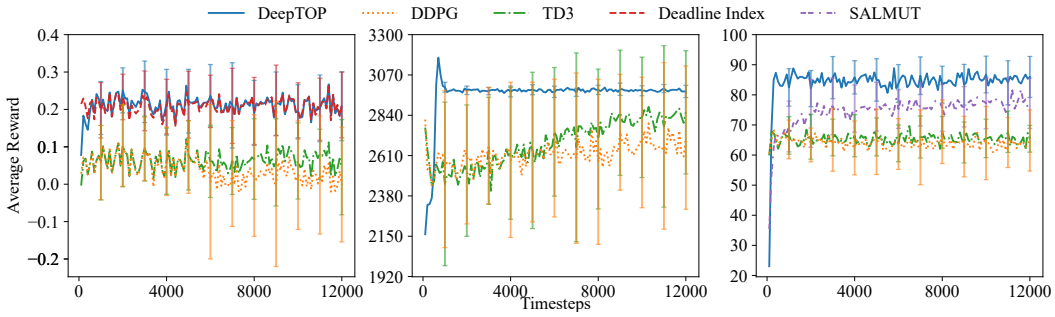
D Experiments' Details

For all Deep RL algorithms, we used PyTorch [22] to implement them, with Adam [18] as the optimizer. We used 10^{-4} as the learning rate for the actor networks, and a learning rate of 10^{-3} for the critic networks. We also set the initial learning rate of action-value function to 0.1 for tabular LPQL and tabular WIBQL. Tabular WIBQL initial learning rate for updating indices is 0.2. The warmup period has 1000 timesteps used for filling the memory $\mathcal{M}$ with transitions from random actions. We use a constant $\epsilon_t = 0.05$ through all timesteps. A discount factor of $\gamma = 0.99$ was selected. All neural network layers were initialized using PyTorch's default method. We update parameters using a minibatch size of 64 transitions, with policy updates made at every timestep after the warmup period ends. The neural networks used for results in Section 6 have two hidden layers with sizes [128, 128], with the input layer dimension depending on the state size. Output layer has a dimension of one.

The used code for TD3, tabular LPQL, and tabular WIBQL are licensed under the MIT license. The DDPG code we used is licensed under the Apache license 2.0. All Deep RL algorithms were trained using a computing cluster with 9632 computing cores distributed over 320 nodes. All algorithms were trained using CPU cores.

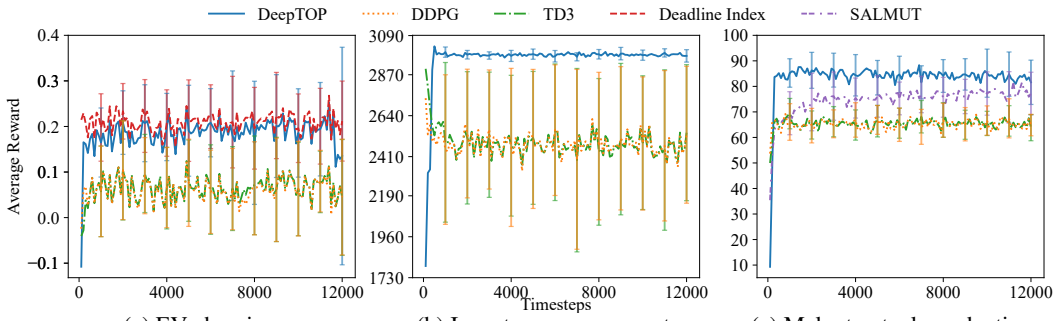
E Additional MDP Simulation Results Using Different Neural Network Architectures

We provide results here for the considered MDP problems for different neural network architectures. All other hyperparameters were kept the same as described in Appendix D.



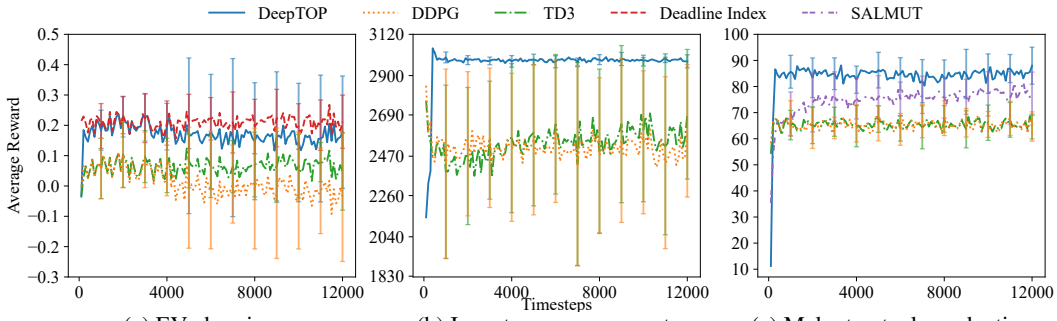
(a) EV charging. (b) Inventory management. (c) Make-to-stock production.

Figure 5: Hidden layers' size: [64, 128, 64]. Average reward results for the MDP problems.



(a) EV charging. (b) Inventory management. (c) Make-to-stock production.

Figure 6: Hidden layers' size: [32, 64, 64, 64, 64, 32]. Average reward results for the MDP problems.



(a) EV charging. (b) Inventory management. (c) Make-to-stock production.

Figure 7: Hidden layers' size: [64, 64, 64, 64, 64]. Average reward results for the MDP problems.

F Recovering Bandits' Description

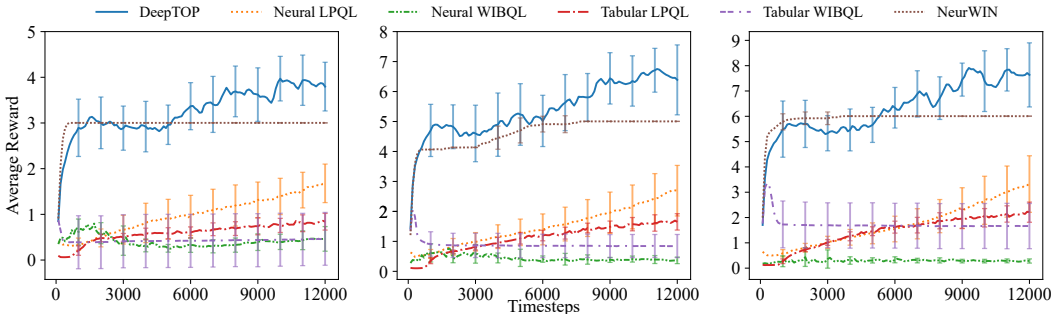
The state $s_{i,t}$ is the waiting time since the arm i was last activated, with the maximum waiting time z_{max} set to 100. If the agent chooses to activate the arm, the arm's state is reset to 1. The arm's reward is provided by the recovering function $f(s_{i,t})$, where if the arm is activated, the reward is the function value at $s_{i,t}$. Otherwise a reward of zero is given if the arm is left passive. The recovering reward function is generated from

$$f(s_{i,t}) = \theta_0(1 - e^{-\theta_1 \cdot s_{i,t}}). \tag{17}$$

We use the same $\Theta = [\theta_0, \theta_1]$ hyperparameters for setting the arms' reward classes as used in [20] and provide them in Table 1.

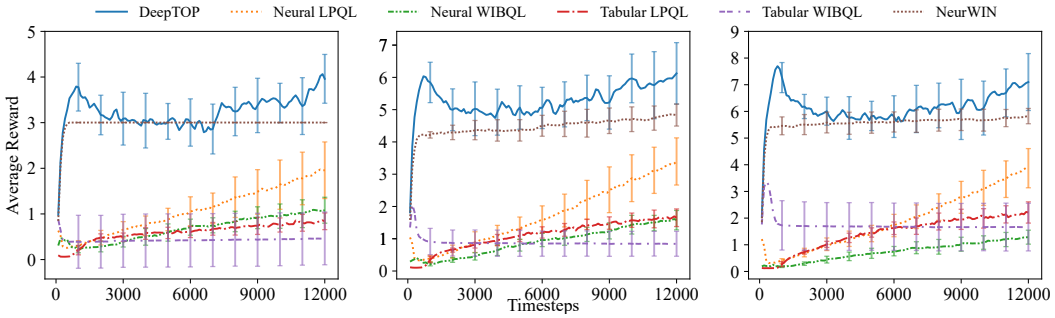
G Additional One-Dimensional Bandits' Simulation Results Using Different Neural Network Architectures

We provide results here for the considered RMAB problem for different neural network architectures. All other hyperparameters were kept the same as described in Appendix D.



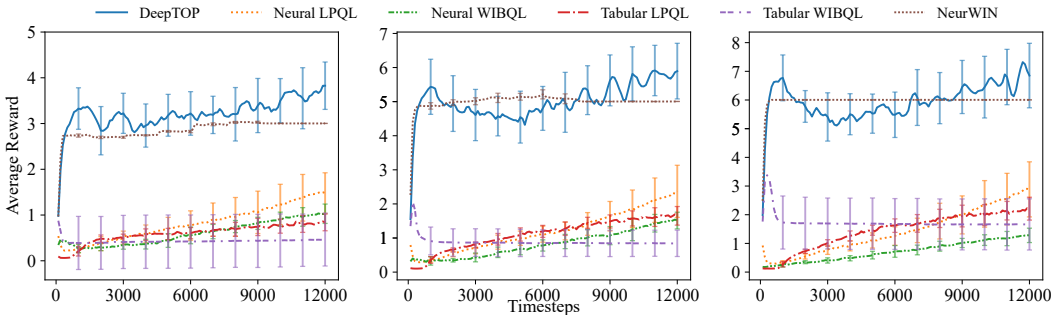
(a) $N = 10$. $V = 3$. (b) $N = 20$. $V = 5$. (c) $N = 30$. $V = 6$.

Figure 8: Hidden layers' size per arm: $[64, 128, 64]$. Average reward results for the one-dimensional bandits.



(a) $N = 10$. $V = 3$. (b) $N = 20$. $V = 5$. (c) $N = 30$. $V = 6$.

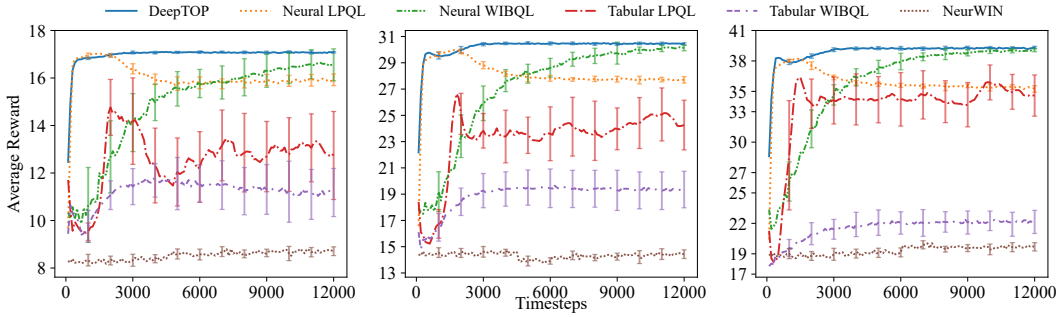
Figure 9: Hidden layers' size per arm: $[32, 64, 64, 64, 64, 32]$. Average reward results for the one-dimensional bandits.



(a) $N = 10$. $V = 3$. (b) $N = 20$. $V = 5$. (c) $N = 30$. $V = 6$.

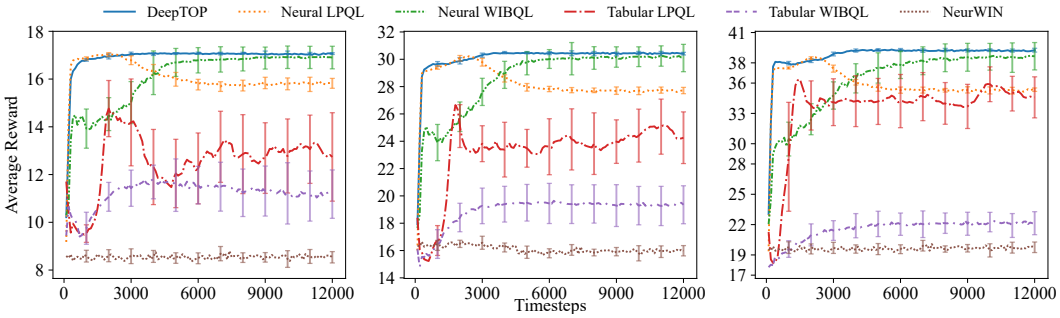
Figure 10: Hidden layers' size per arm: $[64, 64, 64, 64, 64]$. Average reward results for the one-dimensional bandits.

H Additional Recovering Bandits' Simulation Results Using Different Neural Network Architectures



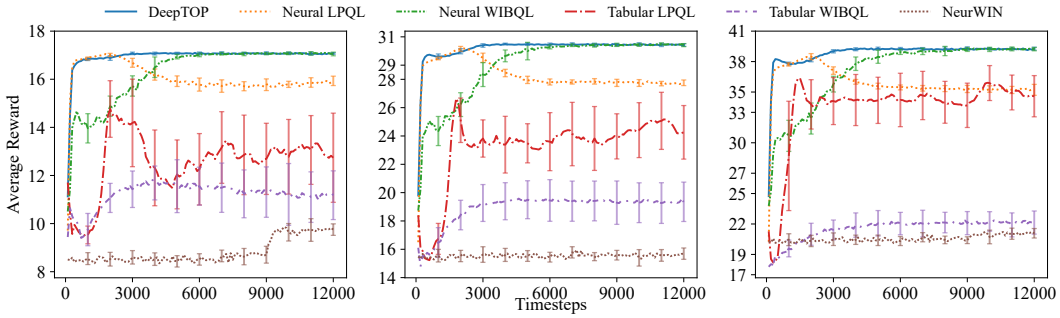
(a) $N = 10$. $V = 3$. (b) $N = 20$. $V = 5$. (c) $N = 30$. $V = 6$.

Figure 11: Hidden layers' size per arm: $[64, 128, 64]$. Average reward results for the recovering bandits.



(a) $N = 10$. $V = 3$. (b) $N = 20$. $V = 5$. (c) $N = 30$. $V = 6$.

Figure 12: Hidden layers' size per arm: $[32, 64, 64, 64, 64, 32]$. Average reward results for the recovering bandits.



(a) $N = 10$. $V = 3$. (b) $N = 20$. $V = 5$. (c) $N = 30$. $V = 6$.

Figure 13: Hidden layers' size per arm: $[64, 64, 64, 64, 64]$. Average reward results for the recovering bandits.