CS:605 Independent Study
Multi-armed bandit using Deep Q network

Keya Desai
Rutgers University
Email: keya.desai@rutgers.edu

Charles Cowan
Rutgers University
Email: cwcowan@rutgers.edu

Prakruti Joshi
Rutgers University
Email: phj15@scarletmail.rutgers.edu

***Abstract*—**

Multi-armed bandit problem in reinforcement learning refers to devising a technique to optimize maximum reward gained at each step where the bandits follow a particular distribution. As part of our independent study, we understand the bandit problems, and the traditional methods for solving it such as UCB, and ϵ - greedy. In this work, we apply the reinforcement algorithms of classical Q-learning model and its variants of Deep Q-learning, Double DQN [1], Actor - critic, and deep Q-learning with replay buffer to the problem of multi-armed bandits. We present our experiments with various set ups and report the robustness of the models across different distributions of bandits.

I. Introduction

The idea behind multi-armed bandit problems is of allocating resources among bandits/machines so as to maximise a reward. There are multiple bandits that give a reward based on an unknown underlying distribution. The player chooses one bandit at a time to maximise the sum of rewards. This is a classic reinforcement learning problem that exemplifies the exploration/exploitation trade-off dilemma.

Typical results are usually of the form - *if a policy for button pressing is going to perform well for all possible sets of buttons, then it can't do better than Regret(N) = O(ln(N))* where the constant in the big-O term depends on the parameters of the current set of buttons. Asymptotically, you typically see regret grow exponentially slowly with N. We aim to achieve this bound on regret with the Deep Q network in different set-up of bandits.

The existing methods for solving the multi-armed bandit problem requires the knowledge of underlying distribution of the bandits. Our goal is to develop a system which takes in a set of bandits as input without knowing anything about it distribution and outputs the best bandit over time, hence reducing the overload of the knowledge of the distribution of the bandits. This is the motivation for applying Deep Q-learning to the bandit problems.

Section II describes the Reinforcement learning algorithms which we apply to the bandit problem. The experimental set up and the results have been reported in Section III. We discuss the future scope of this work in IV and finally conclude the work in V. The code for this project can be found here.

II. Model

In this section, we describe the strategies implemented for solving the bandit problems. First, we start with the discussion of the evaluation metric for this problem.

The evaluation metric for how good a model is performing given bandit input is the *cumulative regret* over time. The intuition behind regret is estimating the payoff of selecting an action as compared to the optimal action in each state. Mathematically, regret is computed as -

$$\text{Regret(t)} = \text{Max Reward} - \text{Reward(t)}$$

The need for cumulative regret stems from the fact that independently computing regret discards the information captured over the time. Ideally, we want the cumulative regret to converge over time to a constant value as $t \to \infty$, implying that the regret at every time step has become 0 and the model always chooses the best bandit.

In this problem, max reward has been taken as the maximum mean out of all the bandits and to avoid the getting negative regret, the reward at time t is taken as the mean of the selected bandit rather than $N(\mu, \sigma)$. One of the variance reduction technique that can also be incorporated is the use of advantage instead of reward. Advantage calculation involves subtracting the reward from the average of means of the bandit distributions. This signifies the relative advantage of selecting a bandit from the set of bandits.

A. *Bandit problem strategies*

1) Naive strategy:

We started exploring the bandit problem with a naive strategy for selecting bandits. The strategy selects each bandit *n* times and records a sample mean estimate. Once all the bandits are explored for a fixed number of times we select the bandit with maximum sample mean for the rest of the episodes. The strategy is naive in the sense that it is requires $n * k$ trials, where k is the number of bandits. Also, this method fails to generalize to different set of bandits and may not perform well where the bandit distributions are quite similar.

2) Upper confidence Bound (UCB):

The main motivation for Upper confidence bound approach is to make the exploration more efficient. Ideally, we want to favor exploration of actions with a strong potential to have a optimal value. The UCB algorithm measures this potential by

an upper confidence bound of the reward value. Mathematical formulation of the above notion is as follows:

$$A(t) = \text{argmax}_a \left[Q_t(a) + \sigma(a) \sqrt{\frac{ln(t)}{N_t(a)}} \right]$$

Here, $N_t(a)$ denotes the number of times that action *a* has been selected prior to time *t*, $Q_t(a)$ denotes the values of the action *a* at time *t*. In our bandit problem, it refers to the sample mean of the bandit at time *t*.

Thus, as the number of trials increase, if a bandit has not yet been explored much, its sample count ($N_t(a)$) will be small, and hence overall value of A will be large, selecting that bandit. Hence, the formulation helps in balancing the current estimation Q and exploration rate.

3) ϵ - greedy:
A common method is reinforcement learning algorithms to balance exploration v/s exploitation is the ϵ - greedy method. In this approach, we explore a random action with ϵ probability and exploit the best known action with $1-\epsilon$ probability. This ensures that we explore new actions and do not get stuck in sub-optimal results. In our bandit problem, we maintain sample mean estimates for each bandit. We choose the bandit with highest sample mean at time-step *t* with $1-\epsilon$ probability and select a random bandit with ϵ. We experimented with different ϵ values and observed the cumulative regret across episodes. A simple variant is decreasing the ϵ with time (decaying ϵ). This is intuitive because as we gain experience with time, we are more confident about the optimal bandit and require less exploration.

B. *Policy gradient*

Policy Gradient methods learns the policy directly with a parameterized function with respect to the model parameters. It determines the optimal action to be taken in a given state. The *Policy Gradient Theorem* lays the theoretical foundation for various policy gradient algorithms. The reward function is defined as the expected return by following policy π. The model is trained to solve the optimization problem of maximizing the reward function. The gradient of this maximization reward is defined as follows:

$$\nabla J(\theta) = E_{\pi_\theta}[\nabla ln \pi(a|s;\theta) Q_\pi(s,a)]$$

We define a simple neural network model in TensorFlow 1 which learns the policy to pick actions by adjusting weight associated with each bandit. The weights are modified by applying gradient descent after receiving reward. The loss function for this policy gradient is defined as follows:

$$\text{Loss} = -log(\pi) * A$$

where π is weight associated with selected bandit and A is the advantage on selecting that bandit. This loss functions simplifies the above policy gradient function. It follows the intuition, that is the loss function allows the model to increase the weight for actions that yielded a positive reward, and decrease the weight for actions that yielded a negative reward. We ran the above model for a fixed time steps and checked its convergence to the optimal bandit. The implementation can be found over here - Bandits using Policy gradient. The comparison between the above methods can be seen in Figure 1.

C. *Iterative Q learning*

A value iteration approach of solving the problem was developed by estimating the value of switching to a particular bandit. The Bellman Equation was used to improve the estimate for the value of a Bandit. The dynamic programming approach yields a Q-table estimate for each state. Here each state refers to a particular bandit and the action refers to switching from a particular bandit to the next one. We observed the convergence of this iterative approach. The approach did converge to select the optimal bandit after certain steps (1000 steps). However, the shortcoming of this method is that it is valid only for an input set of fixed bandits. The learning of the approach cannot be extended to a new set of bandits. Hence, we resort to a more general and robust approach - Deep Q network. The implementation can be found over here - Iterative Q learning.

D. *Deep Q Network (DQN)*

The classical Q learning is an off-policy, value-iteration RL algorithm based on the well-known Bellman Equation:

$$Q^\pi(s,a) = E_{s`}[r + \lambda * Q^\pi(s`, a`)|s,a]$$

where the Q represents the cumulative value/utility of taking action a in state s by following policy π. It is composed of the immediate reward and a discounted reward which accounts for rewards achieved in the future time-steps. Solving a reinforcement learning problem is equivalent to finding the optimal Q value for each state and action.

However, a common problem with the classical Q learning model is that it is not scalable i.e. finding optimal Q value for state-action pair is not possible. Specially, in our bandit problems, each new state is newly generated and states do not repeat. Since the Q table follows dynamic programming approach, it shall not be able to estimate unseen states well. Hence, the tabular approach of Q update does not apply here. Deep Q network solves the problem as they are function approximators (Neural networks) used to estimate the Q value for any given state-action pair.

Thus, we have the loss function defined as follows:

$$L_i(\theta_i) = E_{s,a\ p(.)}[(y_i - Q(s,a;\theta_i)^2]$$
$$\text{where} \quad y_i = E_{s`}[r + \lambda * max_{a`} Q^\pi(s`, a`; \theta_{i-1})|s,a]$$

State representation in bandit - One essential task was to define what consists a state in the context of our bandit problem. While modeling, we limited our scope to normal bandits with parameters mean and sigma. Thus, one possible

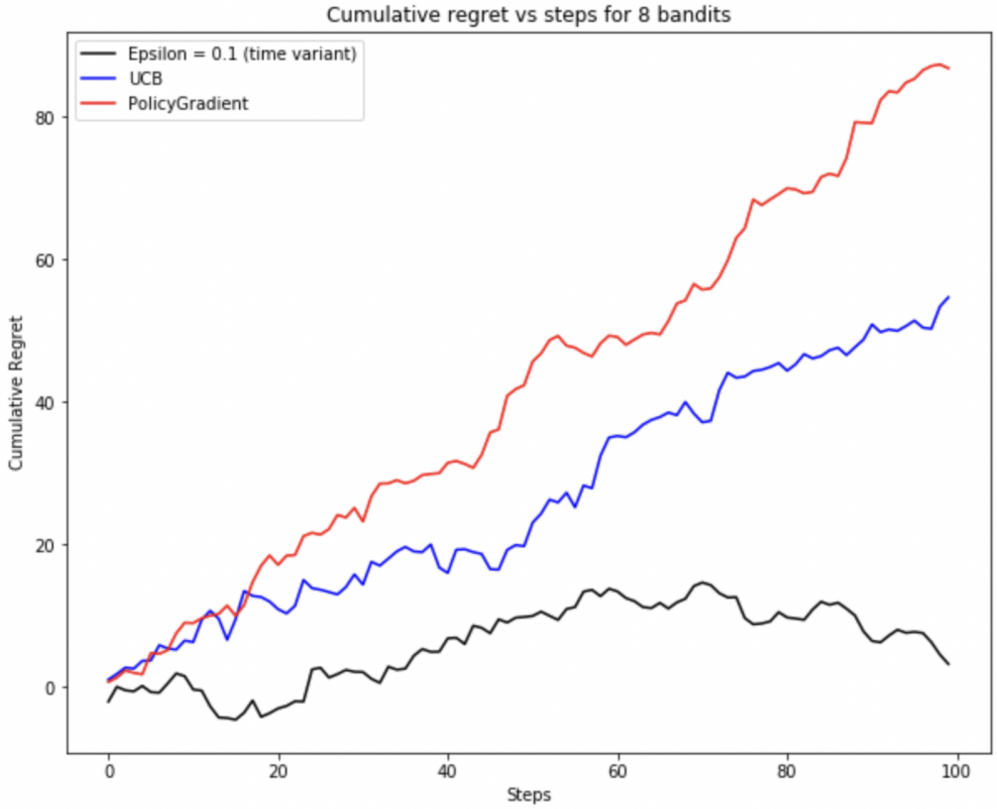


Fig. 1: Comparison between UCB, decaying ϵ - greedy and Policy gradient. The first two methods converge in terms of cumulative regret over time.

way to represent our knowledge of bandits is using the sample mean and the sample counts for each bandit. Thus, at time t and for m bandits, the input state for the model would be -

$$s_t = [[\hat{\mu_1}^t, \hat{\mu_2}^t, ..., \hat{\mu_m}^t], [\hat{k_1}^t, \hat{k_2}^t, ..., \hat{k_m}^t]]$$

This state is updated after every timestep as we received a reward from the selected bandit. The model should be able to generalize to different sets of bandits.

E. Double DQN

The main intuition behind Double DQN is that the regular DQN often overestimates the Q-values of the potential actions to take in a given state. This is the problem of maximization bias faced in DQN. Thus, double DQN aims to decouple target and current Q-value by introducing two DQN networks. They function in a symmetric way where by random probability, a particular network, say A is used to select the action which maximizes Q in given state and network B is updated. Thus, this structure reduces overall instability in the network and is more robust. However, computationally it is more expensive than regular DQN.

F. Actor - Critic

Actor-Critic models combines the benefits of both - value-iteration methods such as Q-learning and policy-iteration methods such as Policy Gradient. The actor runs for certain epsiodes/timesteps and collecting experience. After a certain point, the critic evaluates the model and the actor copies the updated network of the critic model. Here, the actor and critic models can be either DQN or Double DQN. In our implementation, we have kept the model as DQN (as explained in II-D). A subtle difference between using DQN model and using Double DQN is that in the DQN model, actor/critic have asymmetric roles whereas in Double DQN both networks are interchangeable.

G. Replay Buffer

One of the major challenge in reinforcement learning problem is that the network weights can oscillate or diverge due to the high correlation between actions and states. However, the sequence of states that are generated using the bandit distribution are highly correlated. The technique of replay buffer or experience buffer is used to overcome this problem. We collect the (state, action, reward, new state) information across episodes and hence across multiple set of bandit distributions. We then sample from this large buffer of data to fit out model and update weights. This reduces correlation bias and mimics the human behavior of forgetting past experience once new information dominates.

III. Experiments

For all the experiments, we train the model on 2 or 3 bandits and test the model on a fixed set of test bandits of size equal to the number of bandits used for training. The training has been done on 100 time steps with $\epsilon = 0.2$, $\beta = 0.9$ unless mentioned otherwise. In each round of the training, a fixed number of episodes run and in each episode, the model is trained on a new randomly generate bandit for 100 time steps. At the end of all episodes, the model is tested on the test bandit and the next round of training is repeated.

Metric of average cumulative regret has been used for testing the performance of the model. For computing the average regret, a sample regret trajectory is generated as follows - the current state is reset for all the bandits. Using the current state and the trained model, the bandit with the highest Q value is selected. Regret is computed as the difference of maximum mean out of all the bandits and the mean of the selected bandit. The current state (sample means and counts) is updated based on the selected bandit. This process is repeated for 100 time steps. At the end of the time steps, the cumulative sum of the regret is taken. Summing up, the model plays the bandit game for 100 time steps on the test bandits, based on which the cumulative regret is computed at each time step. The regret is averaged over by making the model play the game 10 more times on the same test bandits and averaging the cumulative regret at each time step.

A. *Deep Q-learning*

The deep q-learning model was initially implemented using TensorFlow 1 and run as a session graph . Later, we moved to TensorFlow 2 and all the models have been implemented using TensorFlow 2. The Q-network has been modeled as three dense layers with number of hidden nodes as (100, 50, Number of Bandits). The activation function used is ReLU.

Figure 2 shows plots for average cumulative regret for 100 time steps using the deep q-learning model. The same trained model has been tested on two different set of test bandits at the end of each training round. From the figure 2b, it can be observed that the minimum regret is achieved at the end of the first round itself and then the regret shoots up for the subsequent rounds. It looks like the the model is over-trained. But it is interesting to observe in the 2a that the minimum regret is achieved at the round of 4 and it goes up at the end of round 5. Hence, it can *not* be conclusively said that model over trains but that it is generalising well to different set of bandits.

1) *Q-network with permutations (Data Augmentation):*

Data Augmentation is a technique in machine learning where we manipulate/alter the data such that we have more data without changing the essence of original data. In our bandit problem, shuffling the bandits or permuting them does not change the distribution of the bandits or the most optimal bandit that the model is supposed to identify. This way we are avoiding introducing bias for any particular bandit as the order is irrelevant to us. The Q model should perform the same and should be invariant of this change. We experimented on a set of test bandit trained on the same deep q-learning network but tested with and without data augmentation. Dashed line in the plot represents regret on test bandit augmented - Figure 3.

2) *Q-network with decaying epsilon:*

The ϵ parameter in training is used to control the effect of exploration v/s exploitation. With ϵ probability, a random bandit will be selected and with $1 - \epsilon$ probability, bandit will be selected based on the current knowledge of the model. In exploitation the model uses its knowledge to select the best bandit at a time step. Whereas exploration allows the model to select a random bandit and update its knowledge from the reward obtained accordingly. At the beginning of the training time, we need to allow for more exploration so that the model is not stuck with the knowledge it has. But as the model learns more and more, the model becomes better at selecting the best bandit and exploring at this point, intuitively, will result in choosing the wrong bandit. To address this problem, we can use a decaying $\epsilon = \epsilon_0/t$ whose value decreases with time along with the probability of choosing a bandit at random.

Figure 4 compares the regret of a model with constant $\epsilon_0 = 0.2$ and one with a decaying $\epsilon = \epsilon_0/t = 0.2/t$. At t = 100, the value of $\epsilon = 0.02$. The models have been tested on the same set of test bandits at each round. It is interesting to observe that for the first 2 rounds, the model with *constant* ϵ performs better on the test bandits but as both the models learns more about the bandits, the model with *decaying* ϵ overtakes and gives the least overall regret. This supports the intuition that at the start of training (first 2 rounds), exploration is needed in order for the model to learn effectively but later exploitation is more fruitful.

3) *Q-network with replay buffer:*

For the implementation of replay buffer, we do not update the weights at every time step but instead store the following information - (state, action, reward, new state). Every 5th episode, we have (5 x 100) data points - 100 data points (time steps) per episode. Out of the 500 data points, 100 (20%) points are sampled randomly. The model weights are updated using these samples.

Figure 5a compares the average regret of a Q-model with and without replay buffer, tested on the same set of test bandits after every round. The model without replay buffer gives lower regret in the early round but the model with replay buffer starts giving lower regret in the later rounds. A parallel argument to the performance of *decaying* ϵ can be drawn over here. The model needs more data points to learn at the beginning of training but later sampling the points out of the larger pool of data points reduces the correlation between action and states and the model performs better.

To see the effect of the the number of samples on the performance of the replay buffer model, the sample size has been increased from 20% to 50%. Figure 5b shows the comparison of replay buffer model with sample size 20%(100 samples out of 500 data points) v/s sample size 50% (250

samples out of 500 data points). In the first 2 rounds, around time step 70, the regret for 50% samples is lower but then it increases and regret of model with 20% samples reduces. Similar behaviour can be observed for rounds 4 and 5. In round 3, model with 50% sample size has the least regret over the time steps. But no conclusion can be drawn as to which model has a better performance overall.

B. *Double Deep Q-learning and Actor-Critic*

In order to compare the performance of the models of Deep Q-learning, Deep Q-learning with replay buffer, Double Deep Q-learning, and Actor-Critic, only the least regret (average of regret over 100 time steps) on test bandits out of all the 5 rounds has been plotted. The legend shows the round for which the minimum regret has been obtained for each of the methods. For the replay buffer, 50% of the data points have been sampled. Figure 6 shows the comparison plot. It can be observed that Actor critic model gives the least regret after round 4, followed by Double Q after round 2. After approximately 65th time step, dqn with replay buffer gives lower regret than dqn model without replay buffer.

C. *Non-normal bandits*

To test the generality of the model, we test the DQN model on non-normal bandits when the model is trained on various sets of normal bandits. We chose the following three distributions and observed the behavior.

1) *Exponential bandits*:

Exponential distribution is a type of asymmetric function characterised by parameter λ. For regret, the expected value of exponential function is the rate of decay - λ itself. A comparison of regret on normal test bandits v/s exponential test bandits is shown in Figure 7. Training is done on deep q-network using normal bandits. We can observe that model performs better for exponential bandits and converges faster in figure 7a. However, in figure 7b, the model performs comparable for both exponential and normal bandits. Hence, the results are quite inconclusive in terms of exact convergence. However, we can say that the model is able to generalize and adapt well for new exponential distributions that it gets as input.

2) *Mixed Gaussian*:

Using the mixed gaussians centered at the same mean but with different variance to test the DQN model, the comparison of regret on normal test bandit v/s mixed Gaussian test bandit is shown in Figure 8. Training is done on DQN using normal bandits. Similar to exponential test bandits, the model is able to adapt to the family of two-mode distributions.

D. *Contextual Bandits*

We test the DQN model on contextual bandits of the type - *sum of the mean of the bandits is zero.* In this case one bandit gives some information about the rest of the bandits. This information is not encoded in the input vector and is left for the model to learn on its own. To answer the question - *If we train a network on bandits only on instances where the sum of the means is 0, is the network able to get additional performance (reduced regret) over the generalized trained network?*, we train a model with normal bandits with sum 0 and test the model also on a set of normal bandits with sum 0. This has been compared against the performance of a model trained on normal bandits with *no* constraint on sums of means and also tested on a set of normal bandits with *no* constraints on sum of means. The result is shown in figure 9. Normal bandits with sum of means 0 is a subset or a special case of the normal bandits. Hence, it is expected that the model is able to learn the best bandit faster.

IV. Future scope

Based on our discussions with Professor Charles Cowan some of the possible extensions and experiments are as follows:

A. *Data augmentation - (Mean shifting by keeping same variances)*

We can observe the behavior of the model when it is trained on a set of bandits with shifting the mean of all the bandits simultaneously by a value. Since, the variance is the same, the net effect of distribution should remain the same i.e. the best bandit remains the same.

B. *Finite episode information*

Since we are running each episode for a particular finite time steps, we discussed the possibility of introducing a *Bonus reward* at the end of each episode. The bonus reward scheme essentially suggests that the reward on selecting the best bandit at the final time step is huge and is zero if any other bandit is selected.

In this strategy, at the final time step, if the model selected the bandit with the largest mean, it is given a big reward else it is penalised and given a reward of 0. This should encourage the model to take a final decision about the bandit, prevent it from wandering and encode the information that the time steps are finite. Figure 10 show one such experiment where reward is kept to be 10.

Other possible strategies to encode the finite time step information are -

- Taking the weighted loss where in loss associated with higher time steps are penalized heavily.
- Experimenting with higher beta values including $\beta > 1$.
- Representing time step t in the input state vector.

C. *Policy gradient version*

In this work, we explored value iteration and modeled our network to find the optimal Q value for given state. Conversely, a policy gradient strategy can also be devised to determine the optimal bandit in each state.

D. *Replay Buffer*

Implement replay buffer in actor-critic and Double Q network as well and conduct more experiments on the optimum sample size.

E. *Contextual Bandits*

Currently, we have experimented with one particular type of contextual bandits. Other types of bandits with varying context can be included to test the model. Better approaches to directly encode the context in the input state to the Q model can be devised.

V. Takeaways

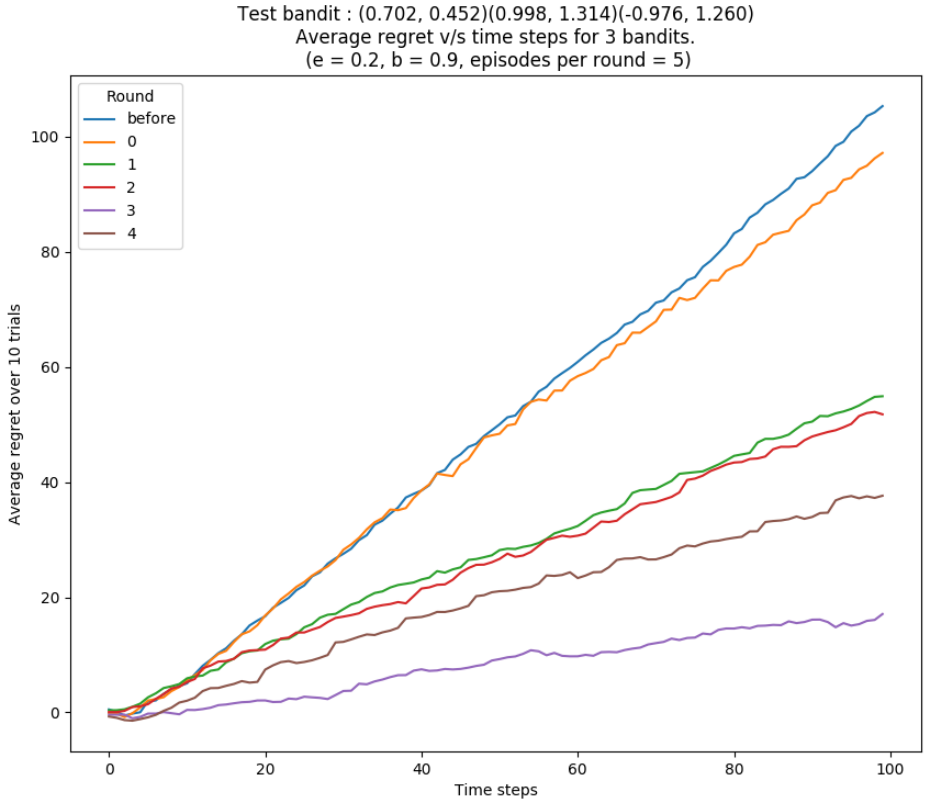
As part of this semester long independent study, we applied the RL algorithms of policy gradient, iterative q-learning, deep q-learning, double deep q-learning, and actor critic to the problem of multi-armed bandits. We studied the existing work and algorithms on bandits, experimented with the concept of exploration-exploitation in ϵ greedy, UCB, and decaying ϵ; reduced correlation among state and reward using replay buffer; augmented data using permutations. Our experiments of applying the deep q-model to test non-normal bandits (exponential, mixed gaussian, and contextual bandits) proves the generality of our model. We summarized our findings from the regret plots and thoroughly explained the future potential extensions to existing work. Finally, we deliver the TensorFlow implementation of Deep Q learning model (and its variants) for Bandit problem.

VI. Acknowledgement

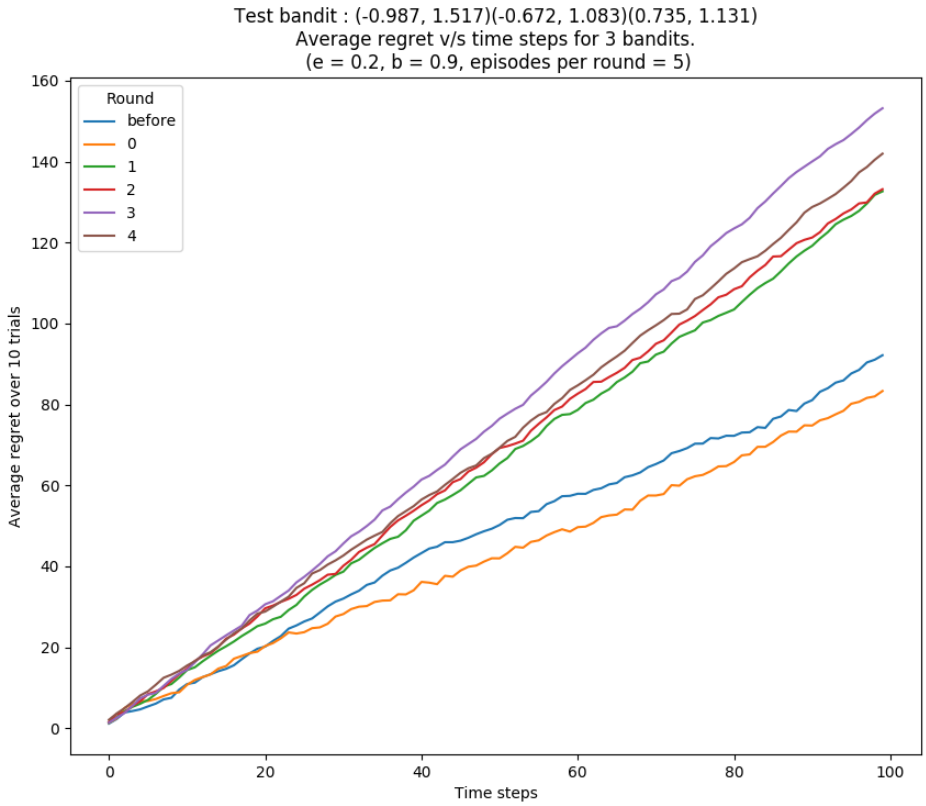
We would like to thank Professor Charles Cowan for his constant support and guidance as well as for providing necessary resources for the project. We thoroughly enjoyed the discussions and knowledge transfer sessions. His mathematical intuitions and proofs on the concepts were extremely interesting and valuable in our learning.

References

[1] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015.



(a) Test bandit set 1 : means and variances i) (0.702,0.452), ii) (0.998, 1.314), iii) (-0.976, 1.260)



(b) Test bandit set 2 : means and variances i) (-0.987, 1.517), ii) (-0.672, 1.083), iii) (0.735, 1.131)

Fig. 2: Deep Q-learning. Result of a Deep Q-learning network tested on 2 set of normal bandits.

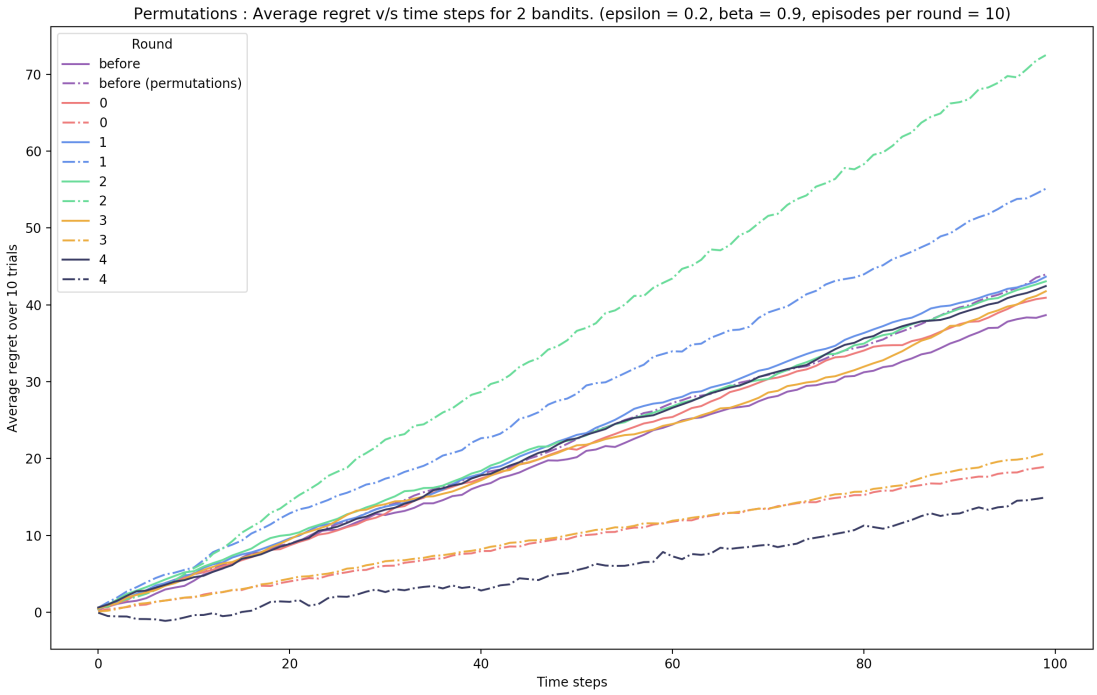


Fig. 3: Data augmentation. Result of a set of test bandit trained on the same deep q-learning network but tested with and without data augmentation. Dashed line in the plot represents regret on test bandit augmented.

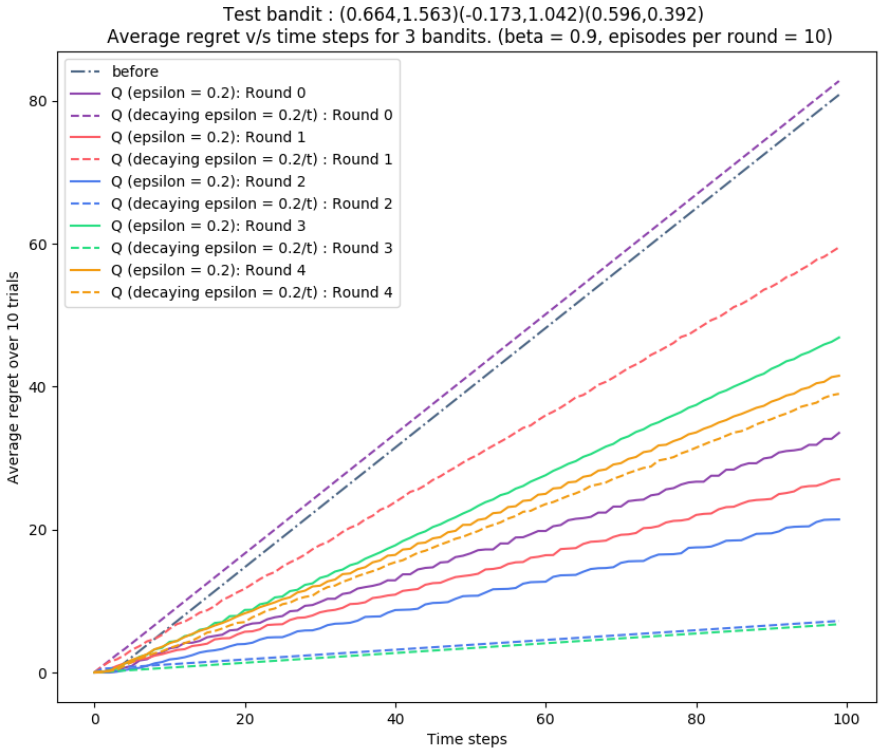
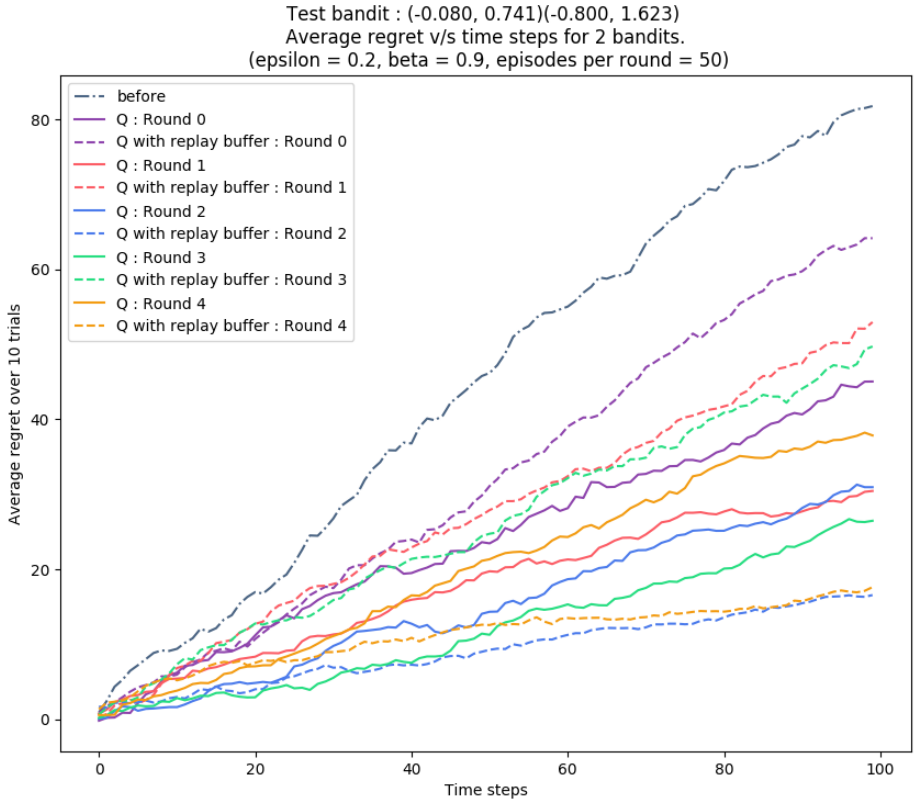
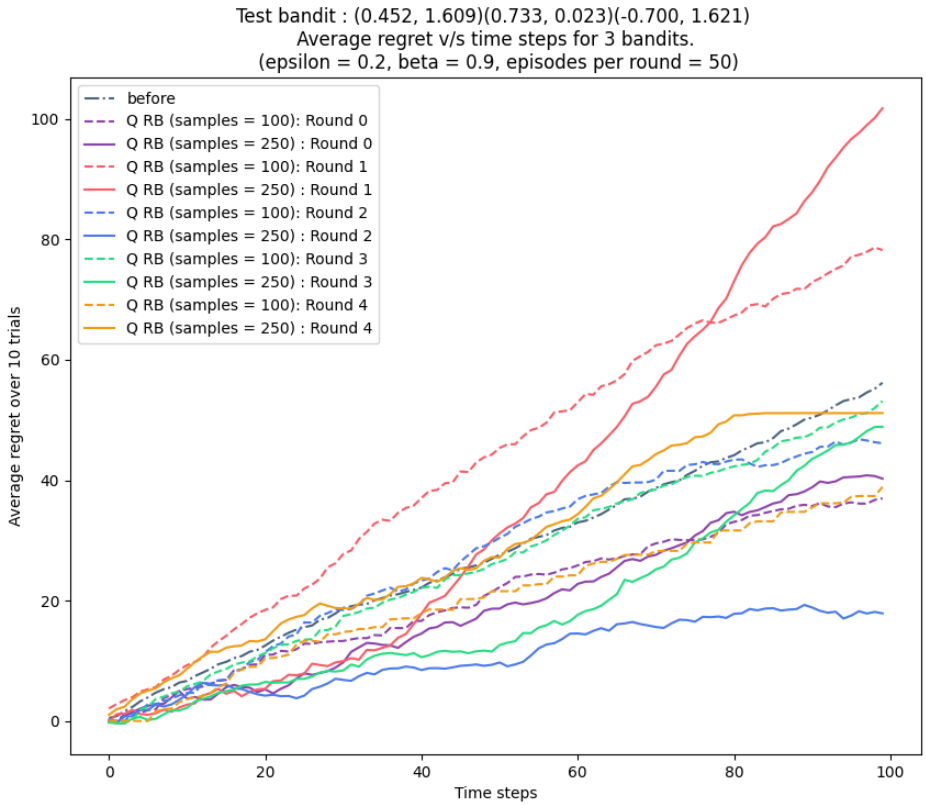


Fig. 4: Decaying epsilon. Two deep q-learning networks, one with $\epsilon = 0.2$ and other with a decaying $\epsilon = 0.2/t$, tested on the same set of bandits.



(a) Comparison of Q-learning with and without replay buffer



(b) Comparison of Q-learning with replay buffer for 20% samples and 50% samples

Fig. 5: Deep Q-learning with Replay buffer. Result of a Deep Q-learning with replay buffer **samples**

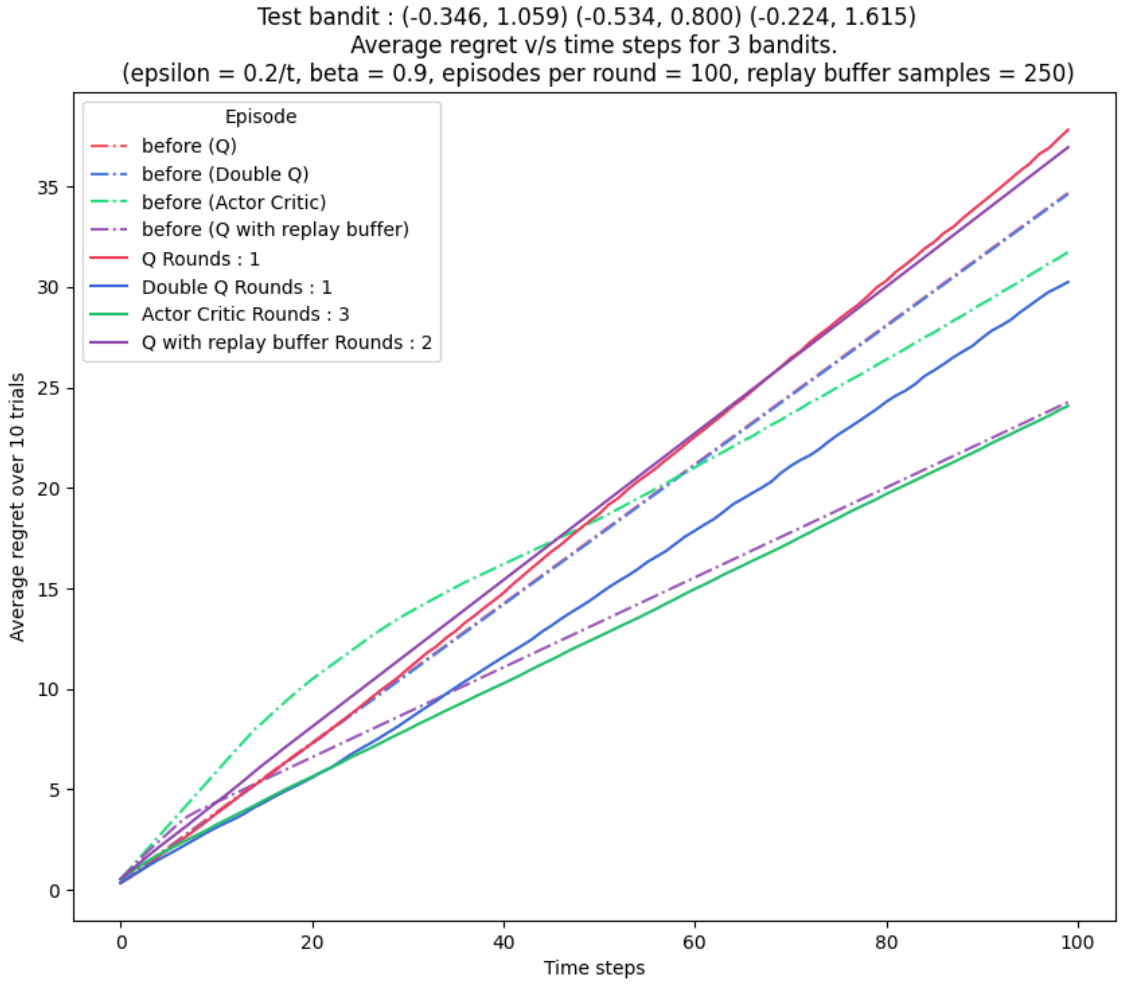
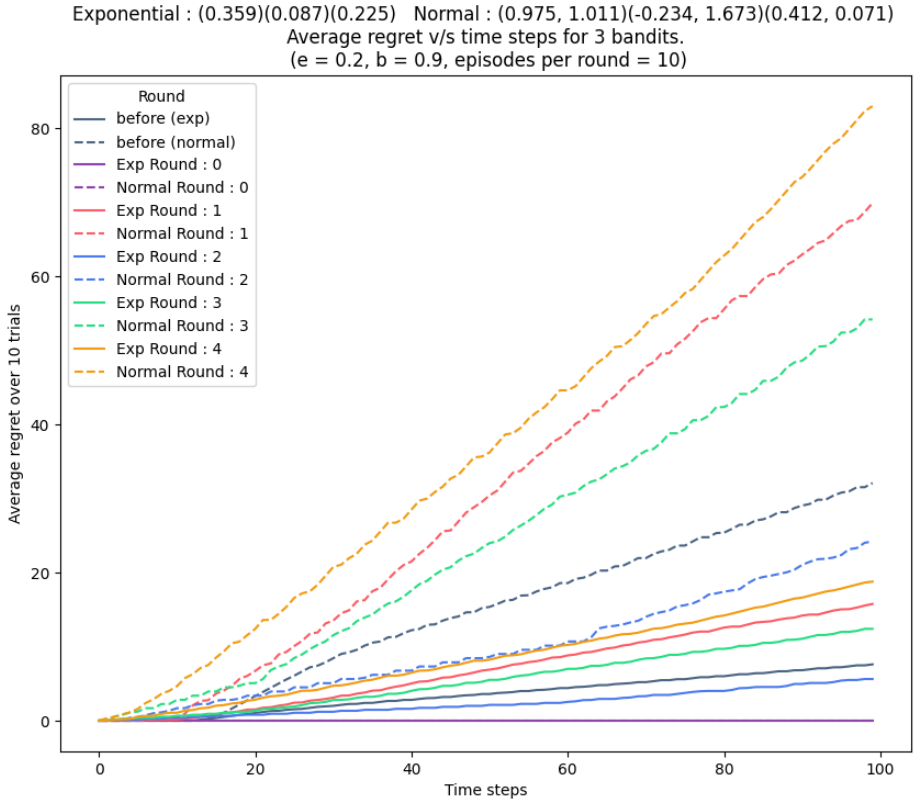
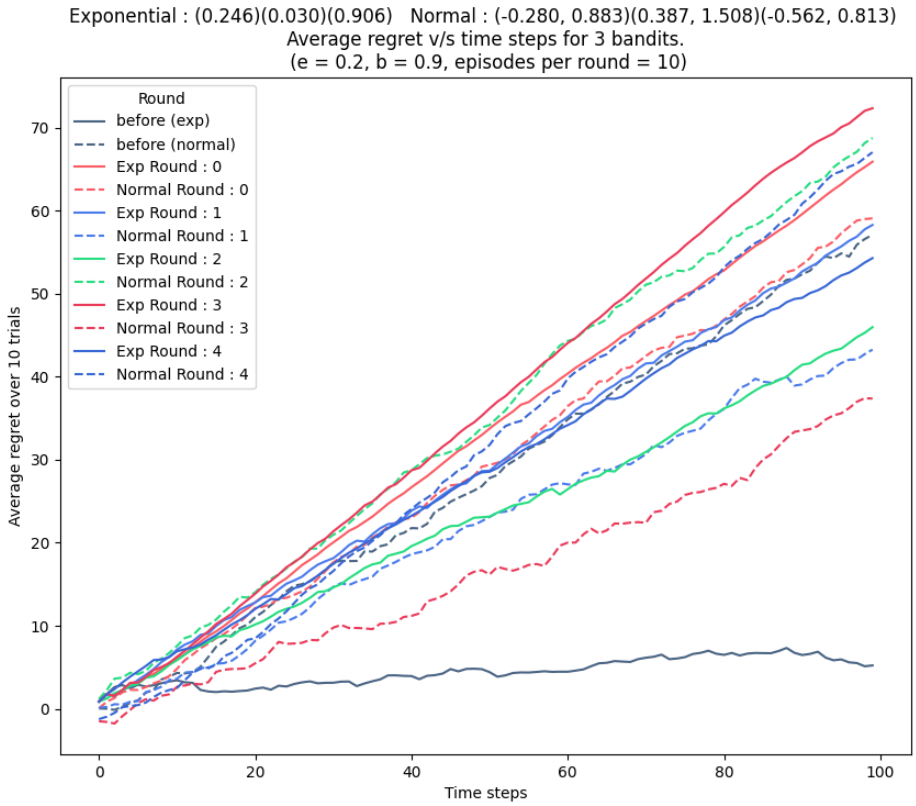


Fig. 6: Comparison of all models. Comparison of average regret of all the networks on the same test bandit.



(a)



(b)

Fig. 7: Exponential bandits Comparison of regret on normal test bandit v/s exponential test bandit. Training is done on DQN using normal bandits.

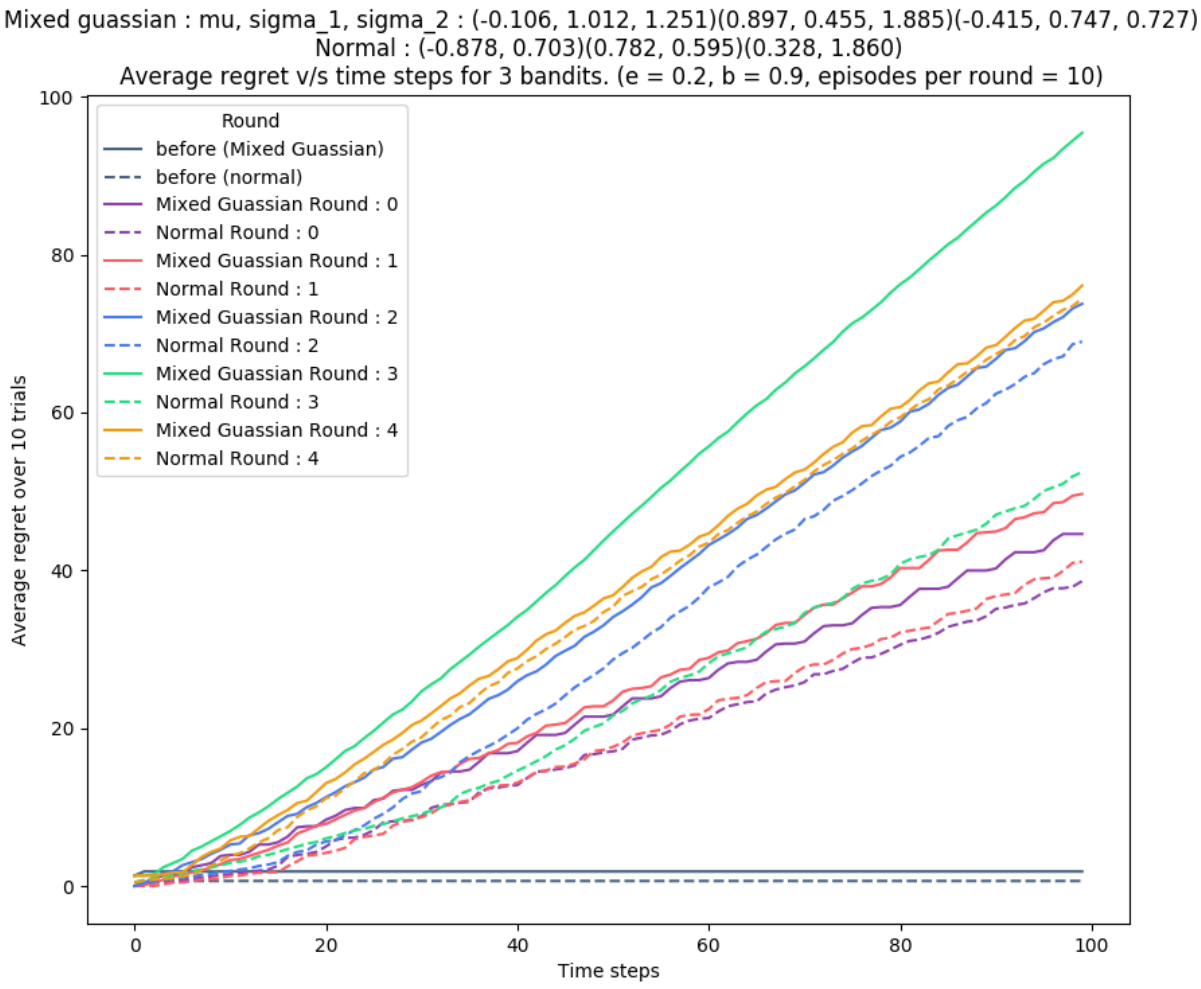


Fig. 8: Mixed Gaussians. Comparison of regret on normal test bandit v/s mixed gassian test bandit. Training is done on deep q-network using normal bandits.

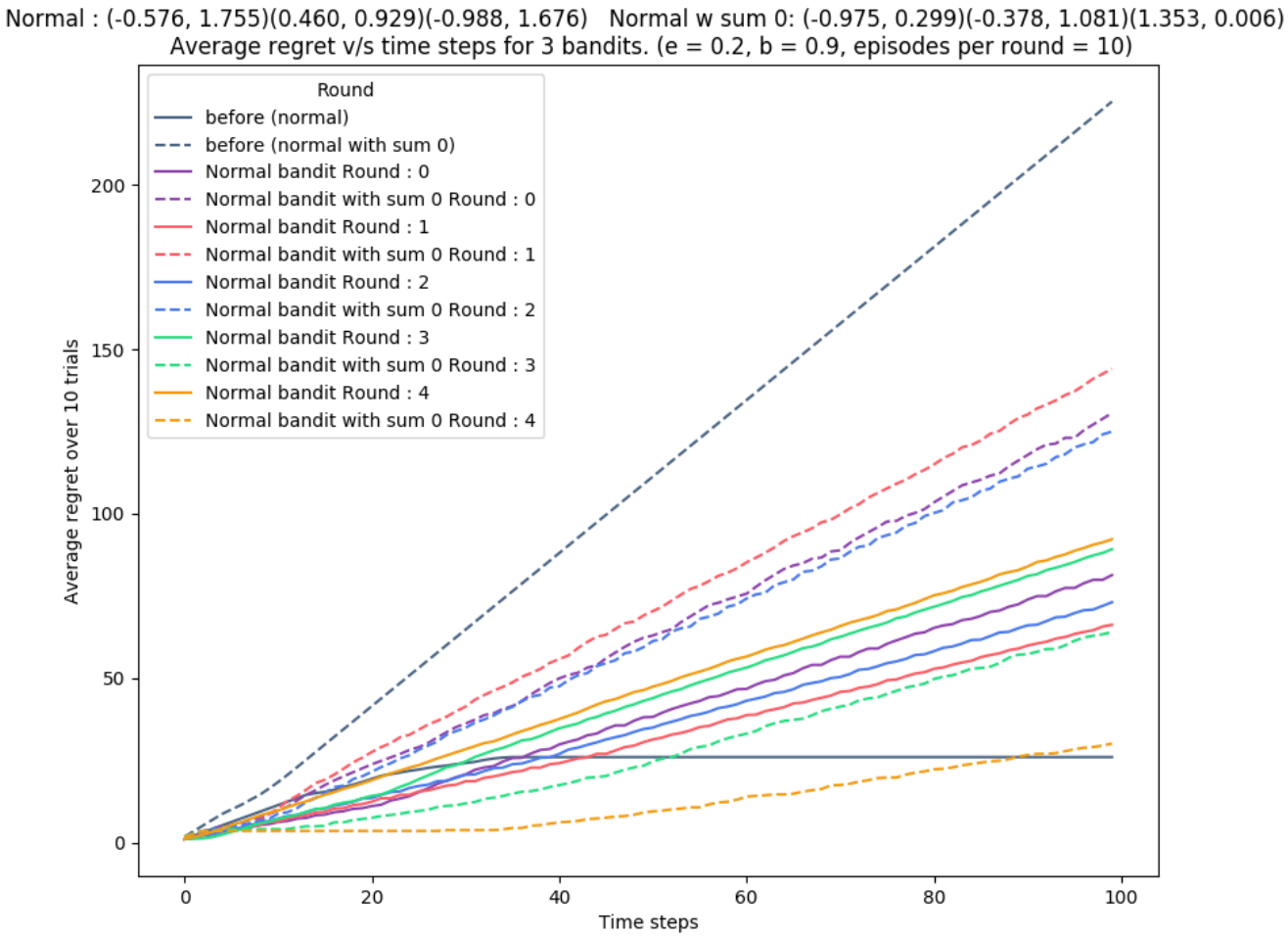


Fig. 9: Normal bandits with sum of mean 0. Comparison of regret on normal test bandit v/s normal bandit with sum of mean 0. Training is done on deep q-network on normal bandits and on normal bandit with sum of mean 0 respectively.

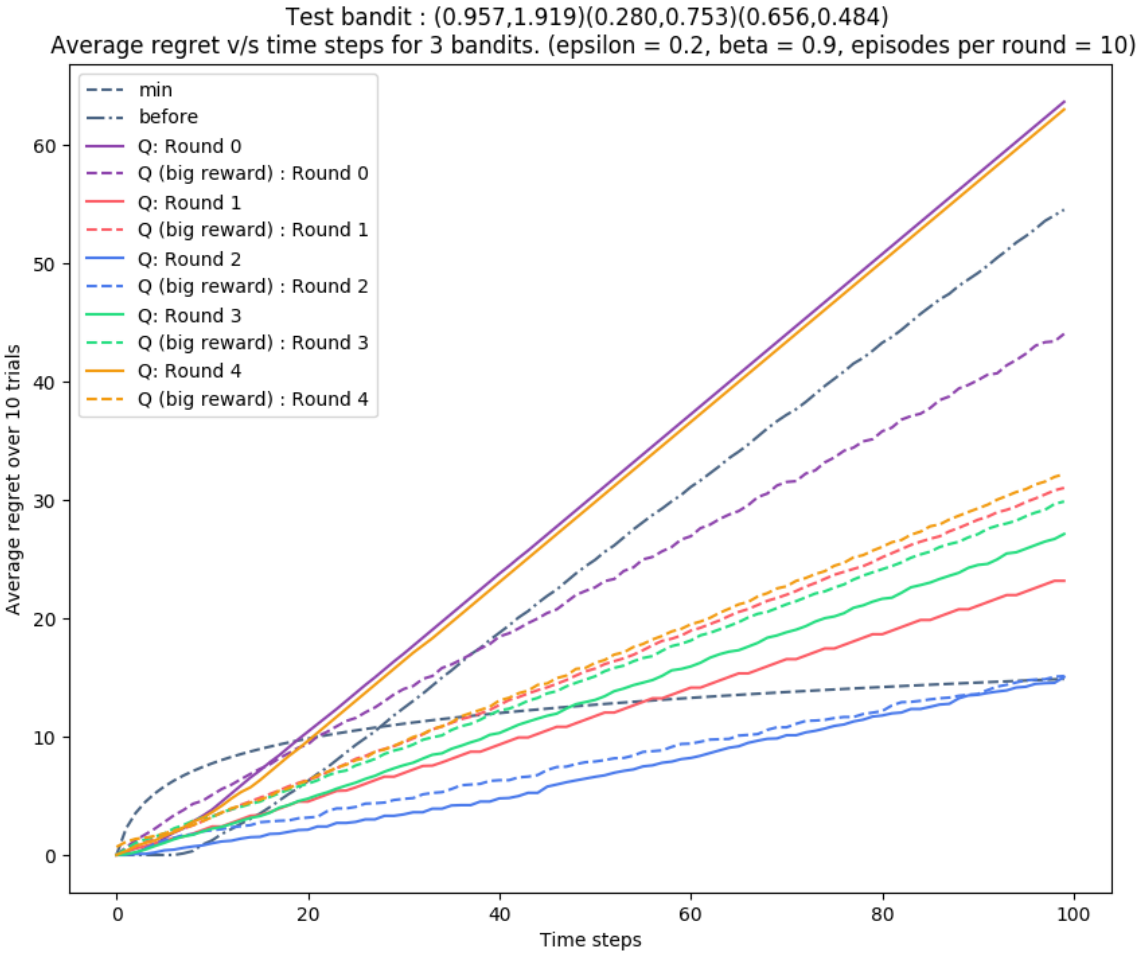


Fig. 10: Big reward. Comparison of DQN model with a DQN model in which at the final time step, if the model selected the bandit with the largest mean, it is given a reward of 10 else it is penalised and given a reward of 0. Regret is computed on the same set of test bandits