A Study of Munchausen Reinforcement Learning and Double Deep Q-Learning

Sameer Singh, Emmanuel Ojuba

1 Introduction

Temporal Difference (TD) learning is one of the most famous reinforcement learning (RL) method that uses bootstrapping to estimate the true value of a state from the current estimate of the value function. In this paper we study a state of the art TD learning algorithm called Munchausen Reinforcement Learning proposed by Vieillard et. al [13]. Munchausen Reinforcement Learning (M-RL) modifies the immediate reward by augmenting it by a scaled log-policy and optimizes it.

We study the M-RL concept and its theoretical basis and apply it to the context of Deep Q-Learning. Furthermore, we study the Double Deep Q learning method proposed by Hasselt et al [7]. We implement these algorithms on two discrete action environments, Cartpole and Atari breakout, to examine performance and we also conduct an ablation study on our M-RL implementation.

Even though the modifications of M-RL can be applied to any TD algorithm we will solely focus on Deep Q Networks, since they are one of the most commonly used RL algorithms. Before, introducing the M-RL we also briefly go over the Deep Q Network.

2 Munchausen Reinforcement Learning

An RL algorithm works under the assumption of a Markov Decision Processes (MDP) framework. An MDP is a simplified representation of an environment using a 4-tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma\}$, where $\mathcal{S}$ and $\mathcal{A}$ are state and action spaces, $\mathcal{P}$ is transition function, r is the reward function and γ is the discount factor. Q-learning is a TD algorithm that try to optimize for the q-function. The RL agent has a policy π associated with it that is mapping from a state to an action (deterministic policy) or a probability distribution over actions (stochastic policy). An RL algorithm also defines a quantify called state-value function, $q_\pi(s,a) = \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, t_0 = t]$, which describes how good a particular action is in a state for the policy π. The goal of the RL agent is to find the optimal policy $\pi_* \in \text{argmax}_\pi q_\pi$. The q_* value corresponding to π_* satisfies the Bellman equation $q_*(s,a) = r(s,a) + \gamma \mathbf{E}_{s'|s,a}[\max_{a'} q_*(s',a')]$. In q-learning, an agent observe transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ and uses the Bellman equation to update the successive q-function estimates as $q(s_t,a_t) \leftarrow q(s_t,a_t) + \eta(r_{t+1} + \gamma \max_{a'} q_*(s_{t+1},a') - q(s_t,a_t))$, where η is the learning rate.

In this paper the authors argue that the q-function is not the only value that can be used to bootstrap RL, the policy can also be used. If an optimal deterministic policy π_* is known, then log of policy (log-policy) for the optimal action is 0 and the log-policy for the sub-optimal actions is $-\infty$. We can add this quantity to the reward to facilitate the learning process, without altering the optimal control. However, since optimal policy π_* is not known, we would use the RL agent's

current estimate π instead. Also, since optimizing deterministic policies is mathematically unstable we assume that the policy is stochastic.

So, M-RL can be summarized as adding scaled log-policy $\alpha \log \pi(a_t|s_t)$ to the reward function r_{t+1} in a TD algorithm, for a stochastic policy which improves the bootstrapping process by aiding the RL agent agent's guess about what actions are good.

2.1 Deep Q Network

A Deep Q Network (DQN) in it's simplest form is a Q Learning method with non linear function approximation. DQN uses a Deep Neural Network as a function approximater which eliminates the need for hand crafted features for the function. However, this naive DQN is not very stable and is prone to overfitting. DQN overcome these shortcomings using 4 major techniques: 1. Experience Replay 2. Target Network 3. Clipping Rewards and 4. Skipping Frames.

2.2 Soft Deep Q Network

M-RL assumes stochastic policies while DQN computes deterministic policies. Therefore, before applying the M-RL modifications to DQN, we need to convert the resulting policies to be stochastic. This can be achieved by taking a maximum entropy RL approach, where we not just maximize the return but also the entropy of the returning policy.

We call the DQN with a stochastic policy Soft-DQN. Let τ be the temperature parameter scaling the entropy, it just amounts to replace the original regression target by:

$$q_{s-dqn}(r_t, s_{t+1}) = r_{t+1} + \gamma \sum_{a' \in A} \pi_{\bar{\theta}}(a'|s_{t+1})(q_{\bar{\theta}}(s_{t+1}, a') - \tau \ln \pi_{\bar{\theta}}(a'|s_{t+1}))$$

Notice also that in the limit $\tau \to 0$ we retrieve DQN.

2.3 Munchausen Deep Q Network

Now, that we have a stochatic DQN, we can add scaled log-policy to the reward. Let $\alpha \in [0, 1]$ be a scaling factor, the regression target of M-DQN is thus

$$q_{m-dqn}(r_t, s_{t+1}) = r_{t+1} + \alpha\tau \ln \pi_{\bar{\theta}}(a_t|s_t) + \gamma \sum_{a' \in A} \pi_{\bar{\theta}}(a'|s_{t+1})(q_{\bar{\theta}}(s_{t+1}, a') - \tau \ln \pi_{\bar{\theta}}(a'|s_{t+1}))$$

2.4 Why M-RL is better?

M-RL provides two theoretical guarantees. 1. Kullback–Leibler (KL) Regularization: Between any two consecutive policies, the M-RL performs KL divergence regularization, which averages the approximation error over the policies. 2. Increased action-gap: It increases the action-gap which also helps reduce the approximation error.

KL divergence is defined as the difference between the two probability distributions [10]. For two successive stochastic policies it can be defined as $KL(\pi_k||\pi_{k+1}) = \mathbf{E}[\ln \frac{\pi_k}{\pi_{k+1}}]$. KL regularization reduces the change between two successive policy update. This helps the RL agent avoid accumulation of errors over successive policies. By reframing the M-DQN in an abstract Approximate Dynamic Programming (ADP) framework it can be easily shown that M-RL implicitly performs KL regularization [13].

Action-gap is defined for a given state the difference between the state-action value of the most optimal action and the state-action value of other actions. Let $\delta_k^{\alpha,\tau}(s) = \max_a q_k(s, a) - q_k(s, \cdot)$,

be the action-gap for any state $s \in \mathcal{S}$ and for k^{th} iteration. It can be shown that for any $0 \leq \alpha \leq 1$ and for any $\tau > 0$,

$$\lim_{k \to \infty} \delta_k^{\alpha,\tau}(s) = \frac{1+\alpha}{1-\alpha} \delta_*^{(1-\alpha),\tau}(s)$$

where $\infty \cdot 0 = 0$ for $\alpha = 1$. So, with M-DQN the action-gap of DQN is multiplied by $\frac{1+\alpha}{1-\alpha}$.

3 Double Deep Q Network

We also study and implement the Double Deep Q Network proposed by Hasselt et al [7] [6], which is also another improvement on the DQN.

In the tabular setting Q-learning update is as thus:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a_t) - Q_t(s_t, a_t))$$

In the Double-Q learning paper [6], Hasselt et al show that the Q-value of the next state, $\alpha(r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a_t)$, could be subject to significant overestimations. Averaged over many experiments, the tabular $\max_a Q_t(s_{t+1}, a_t)$ is an explicit estimate for $\mathbb{E}\{\max_a Q_t(s_{t+1}, a_t)\}$, rather than the desired $\max_a \mathbb{E}\{Q_t(s_{t+1}, a_t)\}$.

To tackle this issue, Double-Q learning uses two estimators, Q^A and Q^B which learn from separate sets of experience samples. To perform the Q-update in Q^A, the maximal valued action a^* is estimated from Q^A, but the Q-value of the next state is selected from the other estimator, $Q^B(s', a^*)$. Performing the update in this manner solves the overestimation problem, although it is not the full solution to finding the desired $\max_a \mathbb{E}\{Q_t(s_{t+1}, a_t)\}$ and underestimations can occur.

The Double DQN paper [7] extends the Double Q learning concept to the context of Deep Q Network with some adjustments. The Q network serves as the first estimator while the target network serves as the second estimator, although the authors acknowledge these two networks are not fully decoupled as postulated in Double Q learning.

The bootstrap update in DDQN is as thus:

$$Y_t^{DoubleDQN} = R_{t+1} + \gamma * Q(S_{t+1}, argmax_a Q(S_{t+1}, a; \theta_t); \theta_t')$$

4 Environments

We utilize two environments to test our implementations.

4.1 Cartpole

The CartPole-v0 environment from the OpenAI gym is a simulation of a pole attached by an un-actuated joint to a cart, which moves along a frictionless track. The pole starts upright and the goal is to keep the cart from falling over by applying force to left or right. The state of the environment is completely defined by the cart position, cart velocity, pole angular position and pole velocity. The system is controlled by applying a force of +1 or -1 to the cart. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center, or the timeout of 200 steps is exceeded.

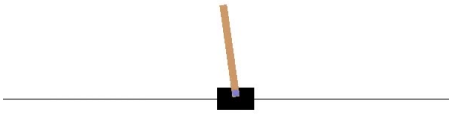

Figure 1: A snapshot of Cartpole environment

4.2 Atari 2600 Breakout

Breakout is a popular Atari game. In this project, we utilize BreakoutNoFrameskip-v4 environment provided in the OpenAI gym [3]. This version of the breakout environment has deterministic actions, as well no frame skipping during gameplay. Following the implementation by Minh et al [11], the state consists of a stack of 4 consecutive 84x84 grayscale images of the game screen. The goal in this environment is to maximize the game score.

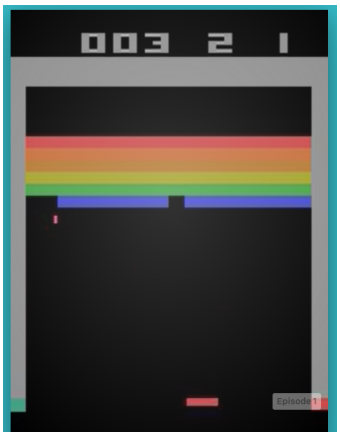

Figure 2: A snapshot of Atari 2600 Breakout environment

5 Experiments

5.1 Model Architectures and Parameters

For the Cartpole environment, we utilize the baseline multilayer perceptron model provided by OpenAI [5] consisting of three fully connected layers and a linear layer which outputs the Q-value for each action (detail in Fig 3). Following OpenAI's model, we utilize a discount factor, γ of 1, a replay buffer size of 50,000 and linearly decay ϵ from an initial value of 1.0 to a final value of 0.02 over the first 20,000 timesteps (10% of total timesteps).

For the Breakout environment we utilize the convolutional neural network architecture defined in the groundbreaking Deepmind paper [11] [4] (detail in Fig 4). Following the parameters

in a Keras tutorial [4], we utilize a γ of 0.99, a replay buffer size of 100,000 and linearly decay ϵ from an initial value of 1.0 to a final value of 0.1 over the first 1M frames.

We optimize in both environments using an Adam optimizer [9], which is different from the RMSprop optimizer [8] used in the original DQN. Notably, Viellard et al [13] observe that the Adam-optimized DQN outperforms the RMSProp-optimized DQN in the ablation study they performed.

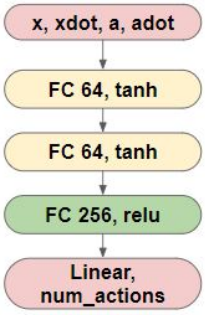

Figure 3: Cartpole Agent Model Architecture

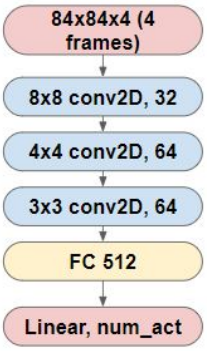

Figure 4: Breakout Agent Model Architecture

5.2 M-DQN

We modify the DQN framework to include the modifications proposed in the paper and use the same hyperparameters used, entropy temperature $\tau = 0.03$, Munchausen scaling factor $\alpha = 0.9$, and clipping value $l_0 = -1$. To avoid numerical issues, the log-policy term is clipped; therefore we replace $\tau \log \pi(a|s)$ by $[\tau \log \pi(a|s)]_{l_0}^{0}$.

5.3 Performance of M-DQN and D-DQNM

We compare the performance of our M-DQN and D-DQN implementations to the baseline DQN method. For each method, we train the model and record the running average reward (over 100 timesteps). We also utilize five different seeds and we report the seed-averaged running average reward as can be seen in Figs 5 and 6. We also display the standard error in the plots. We can see

that MDQN outperforms DQN in both environments, which is consistent with the observations of the paper. It attains higher average reward and learns faster.

On the other hand, DDQN seems to be more stable than DQN in the Cartpole environment, although it's average reward is lower. The higher stability is in line with the observations made in the DDQN paper [6], although the lower reward is not expected.

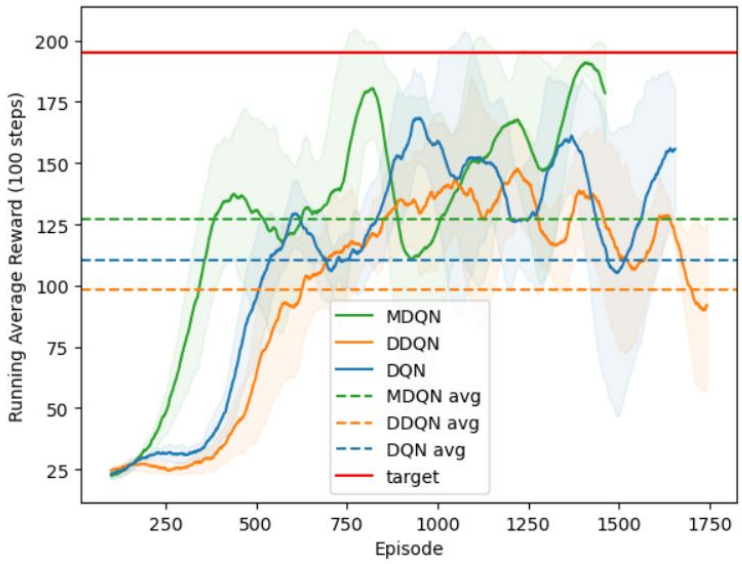

Figure 5: Episodic Rewards Cartpole Training for all 3 Models

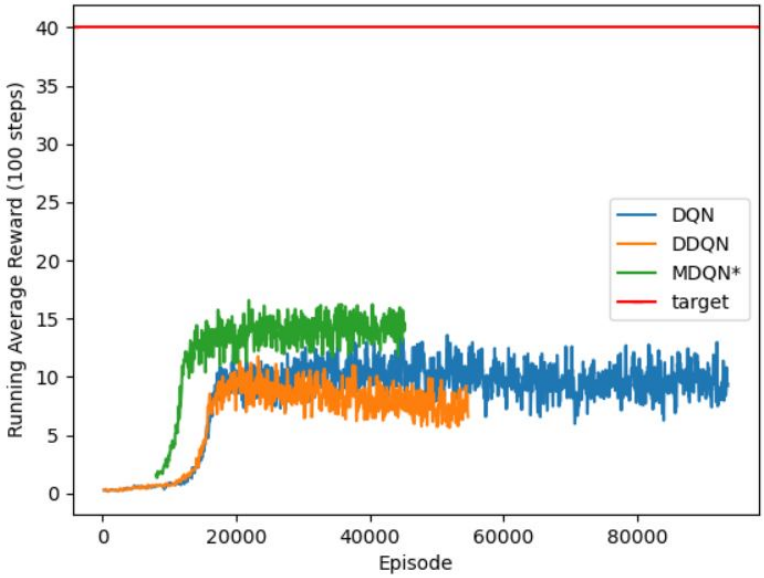

Figure 6: Episodic Rewards during Atari Breakout Training for all 3 Models.
*Some history was accidentally unrecorded in early training stages for MDQN; the MDQN plot was shifted to compensate based on the performance of SDQN in the ablation study (Section 5.4)

5.4 M-DQN Ablation Study

Like the authors of the M-DQN paper, we also perform an ablation study. While a natural comparison would be to remove the Munchausen term, the authors show that M-DQN executes entropy regularization with an implied temperature coefficient of $(1-\alpha)\tau$. So we present results for a Soft-DQN with both a τ and $(1-\alpha)\tau$ temperature.

In both environments, Soft-DQN(τ) and Soft-DQN$(1-\alpha)\tau$ perform slightly better than M-DQN, which is different from the observations in the original paper where Soft-DQN performed worse than both M-DQN and the Adam-optimized DQN.

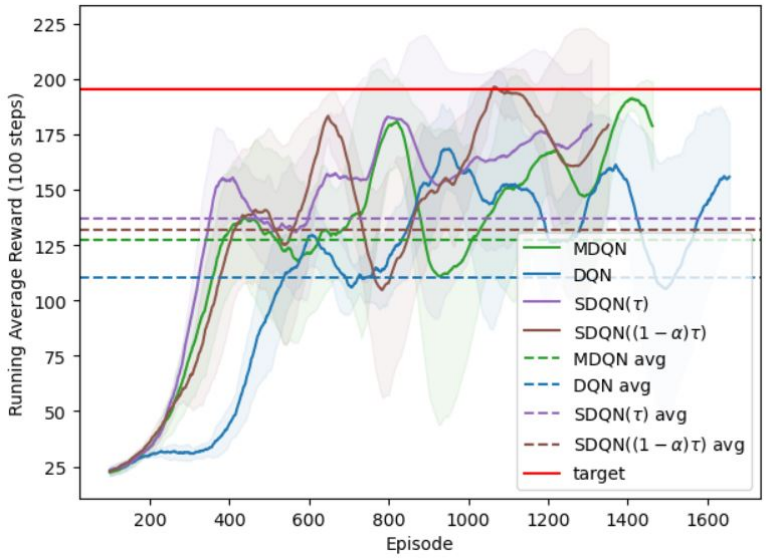

Figure 7: Episodic Rewards Cartpole Training for all 3 Models

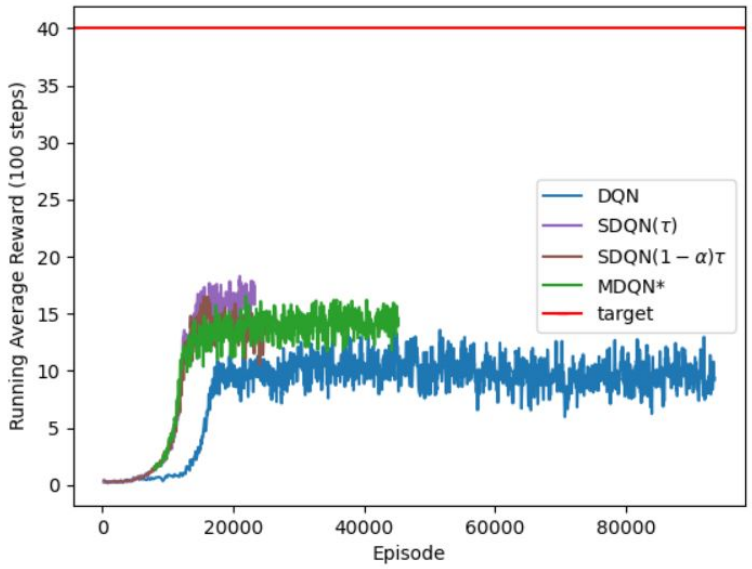

Figure 8: Episodic Rewards during Atari Breakout Training for all 3 Models.
*Some history was accidentally unrecorded in early training stages for MDQN; the MDQN plot was shifted to compensate based on the performance of SDQN in the ablation study (Section 5.4)

6 Code

We adapt the code base provided in a Tensorflow tutorial [4] [1] for training an Atari breakout game for both environments. The code can be found in this Github repository: https://github.com/Mogbo/DeepQ-Project

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. Deepmind lab, 2016.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[4] Jacob Chapman and Mathias Lechner. Deep q-learning for atari breakout tutorial. https://keras.io/examples/rl/deep_q_network_breakout/references, 2020.

[5] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017.

[6] Hado Hasselt. Double q-learning. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23, pages 2613–2621. Curran Associates, Inc., 2010.

[7] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press, 2016.

[8] Geoffrey Hinton. Neural networks for machine learning. http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[10] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, and et al. Human-level control through deep reinforcement learning, Feb 2015.

[12] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[13] Nino Vieillard, Olivier Pietquin, and Matthieu Geist. Munchausen reinforcement learning. *arXiv preprint arXiv:2007.14430*, 2020.