DEEP DETERMINISTIC POLICY GRADIENT BOOSTED DECISION TREES

Clayton W. Thorrez
claytonthorrez@gmail.com

ABSTRACT

Recently in the field of machine learning research, two of the strategies leading to successful papers are: 1. Combining two existing works and 2. Having a catchy acronym. In this work we combine two unrelated machine learning topics, Deep Deterministic Policy Gradients, (DDPG) (Lillicrap et al., 2016), and Gradient Boosted Decision Trees (GBDT) (Breiman, 1997; Friedman, 2001) and introduce DDPGBDT, a novel state-of-the-art machine learning acronym which solves one continuous control task on one seed with heavy hyperparameter tuning.

1 MOTIVATION

This work was not motivated by a theoretical idea or empirical discovery leading to further experimentation. The true reason this exists is that there was a unique pairing of machine learning algorithms which complimented each other in a unique way which we could not overlook. The acronym for Deep Deterministic Policy Gradients ends with a G, and the acronym for Gradient Boosted Decision Trees starts with a G. Additionally, the G's in both names stand for the same word allowing for a natural portmanteau. Furthermore, all letters in both names have the so called "long e" sound adding a comical ring to the pronunciation of the final acronym. (Think bibbidi-bobbidi-boo.)

After completing the hard work of coming up with a novel name and premise, all that remained was to find a way to mash these two ideas together and cherry pick results to make it look like it was a good idea in the first place.

2 BACKGROUND

Before we introduce the novel technical details of DDPGBDT, we will give some background information on the individual components and related work.

2.1 DDPG

Deep Deterministic Policy Gradients has been a popular method in reinforcement learning since it was introduced in 2015. DDPG is an actor-critic architecture where both components are neural networks and the state and action are both continuous vectors. The actor network takes as input the state of the environment and deterministically produces an action to take. The critic network Q takes in both a state and the action and produces $Q(s_t, a_t)$. This represents how much discounted reward the critic thinks the agent will get starting in state s_t, taking action a_t, and following the policy parameterized by the actor $\mu(\cdot)$ until the end of the episode.

$$Q(s, a) = \mathbb{E}_{\mu}\Big[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\Big]$$

The critic is trained to minimize the squared temporal difference error.

$$L_Q = \frac{1}{N}\sum_{i=0}^{N}(r_i + \gamma Q(s_{i+1}, \mu(s_{i+1})) - Q(s_i, a_i))^2$$

We train the critic to minimize this loss using gradient descent or a variant like Adam (Kingma & Ba, 2014) on N (s_t, a_t, r_t, s_{t+1}) tuples sampled from the experience replay. Here s, a, r are state, action, and reward. The discount factor γ is a number in $[0, 10]$ which describes how much value the agent should give to reward at time $t + 1$ compared to at time t.

The actor is updated to maximize the predicted Q value output by the critic using the gradient of the predicted Q value with respect to the actor parameters θ_μ. This gradient is obtained by back-propagating through the critic network and using the chain rule.

$$\frac{\partial Q(s,a)}{\partial \theta_\mu} = \frac{\partial Q(s,a)}{\partial a} \frac{\partial a}{\partial \theta_\mu}$$

Another way to think of this is to set the loss function for the actor to be $-1 * \sum_{i=0}^{N} Q(s_i, a_i)$ and allow an autograd engine like PyTorch (Paszke et al., 2019) to perform the optimization.

In a nutshell you train a critic network to accurately tell you how good taking a certain action is in a certain state, and you train an actor network to produce actions which the critic thinks are good.

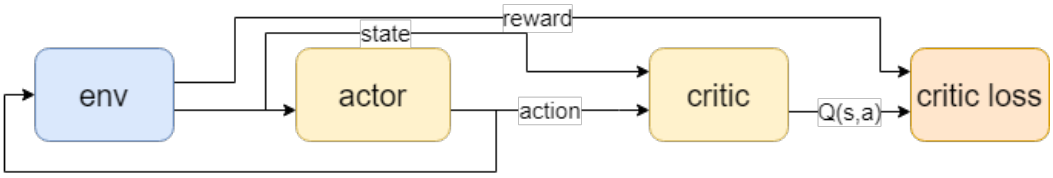

Figure 1: Control flow diagram for DDPG. The environment takes in actions and produces the next state and a reward. The actor maps a state to an action. The critic takes a state and an action and produces a Q value, and the critic loss can be calculated from rewards and Qs.

There are several very good properties which DDPG has. It is sample efficient in that it can learn from the same data multiple times through experience replay as opposed to many other popular actor-critic methods which require on-policy learning. It is also deterministic in that a single state will always map to the same action. In many other methods the actor network learns parameters of an action distribution and to actually act it requires a sampling step.

The primary disadvantage of DDPG is that is is very sensitive to differences in hyperparameters and results have variance. (Haarnoja et al., 2018; Duan et al., 2016; Henderson et al., 2018)

2.2 GBDT

Gradient Boosted Decision Trees are one of the most widely used and most accurate supervised machine learning algorithms. (Breiman, 1996) It is an iterative, ensemble method which trains new weak learners to improve the model from the previous iteration.

To begin the process, a decision tree is trained on the training data to minimize some objective function. Then in the iterative portion, the gradients of the objective with respect to the predictions are computed. So with model $f_i(\cdot)$, input x, label y, and objective $L(\cdot, \cdot)$, we need to compute $\frac{\partial L(f_i(x), y)}{\partial f(x)}$. The next round of training uses the same training data inputs, but uses the negative gradients as the label. In this way, f_{i+1} becomes an approximation of the gradient.

$$f_{i+1}(x) \approx \nabla_{f_i(x)} L$$

Thus $f_i(x) + \eta f_{i+1}(x)$ can be seen as taking an η-sized step in the direction which minimizes the loss. This process can be repeated and in the end you train many decision trees, each of which makes an incremental improvement over the last.

While this is a gradient descent method, it is fairly different from the gradient descent used for neural networks as that method directly updates parameter values using the gradient of the loss with respect to the parameters. This creates an additive model by iteratively adding the gradient of the loss with respect to the previous iteration's output.

3 DDPGBDT

In this work we combined elements of DDPG and GBDT to create DDPGBDT. At the heart, the model is functionally the same as what is shown for DDPG in Figure 1. The difference is that the actor and critic neural networks are replaced with GBDT regression models. In the case of the critic, very little modification was required. It still maps an input which is a concatenated state and action into a scalar and trains by minimizing the mean squared temporal difference error. During training, we sample (s_t, a_t, r_t, s_{t+1}) tuples from an experience replay buffer and compute critic labels.

$$y_i = r_i + \gamma Q(s_{i+1}, \mu(s_{i+1}))$$

The actor is much more difficult to train. The most brilliant part of DDPG is that we can get an approximate gradient of the Q values by back propagating through the critic network. However in this case, the critic is a GBDT. Due to the decision tree structure of the base learners, the critic is a step function. It has many points of discontinuity where the gradient does not exist, and at every other point the gradient is 0.

However, since it is a sum of functions, it does smooth out a little bit when many are added. We decided to try to take a finite difference approximation of the gradient. (Taylor, 1717) The finite differences gradient is built on the definition of a derivative as a limit.

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

We used the central difference where we add and subtract a small constant ϵ from a to be able to see how the function changes in both directions. Thus with critic function $Q(s, a)$, we estimate the gradient of $Q(s, a)$ with respect to the action a as follows.

$$\frac{\partial Q(s,a)}{\partial a} \approx \frac{Q(s, a+\epsilon) - Q(s, a-\epsilon)}{2\epsilon}$$

This gradient can be used to fit a gradient boosted decision tree which maps from state to action whose objective is to generate actions which maximize the output of a critic model for given states.

While on the surface, DDPG and GBDT seem to have very little in common, as one is a continuous, fully differentiable reinforcement learning algorithm, and the other is an additive, discrete classification algorithm, they do share one interesting property which distinguishes them from most modern machine learning techniques. They both deal with gradients of a function with respect to the outputs of another function. In the case of DDPG we back propagate all the way through the critic function to find the gradient with respect to the output of the actor function, the action. In most deep learning settings we only really care about the gradient of the loss with respect to the model parameters. In GBDT as well we compute the gradient of the loss function with respect to the predictions, which are the output of the main model function. In GBDT there are no trainable parameters of the tree-based model to take a gradient with respect to. We did not anticipate this when beginning the work but discovered it during implementation and believe it is perhaps the only notable piece of information in this manuscript.

4 Results

4.1 Extremely Cherry-picked Results

We were able to train a DDPGBDT model to completely solve a control task which keeps a pendulum on top of a cart from falling over by moving the cart side to side. At each timestep, the agent picks a scalar action in the range $[-1, 1]$ which determines how strongly to push the cart left or right in order to keep the pendulum from falling over. A video of the best DDPGBDT model playing several episodes perfectly is available: `https://www.youtube.com/watch?v=ivszueHQCLQ`.

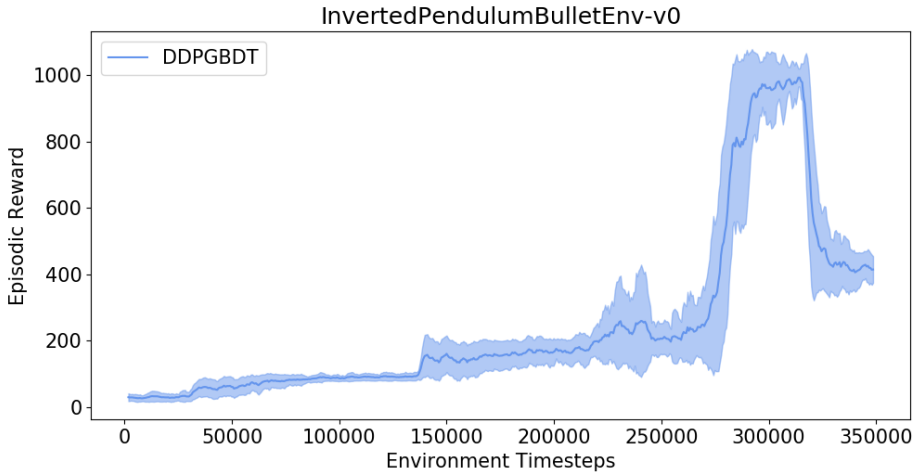

Figure 2: The learning curve for DDPGBDT on the inverted pendulum environment. The x-axis is total timesteps of interaction with the environment and the y-axis is the mean episodic reward $\pm$ standard deviation. The maximum score is 1000.

As shown in Figure 2, the agent trained using DDPGBDT learns very slowly before rapidly improving to completely solve the task for a brief time, and then quickly loses its progress. The model used in the video is one saved at the peak of the learning curve.

4.2 Disclaimers

While this curve may kind of look good, and the video is a real decision tree model which can solve a popularly used reinforcement learning benchmark environment, these results are not truly representative of DDPGBDT. It took several weeks of model tweaking, hyperparameter tuning, and environment hacking to get a single result which solved an environment. Before this result was obtained, we experimented with 6 other single-action continuous control tasks and had little to no success. Countless different combinations of hyperparameters were swept over before finding a combination which works. While the result is *technically* reproducible as it has a fixed seed, it does not work for other seeds. While standard DDPG is notorious for being brittle, DDPGBDT suffers from this problem to an even stronger degree. What this graph really shows is mostly blind luck.

In order to put the graph in perspective, we compared our result with an open source implementation of DDPG from Stable Baselines3 with default hyperparameters. (Raffin et al., 2019)

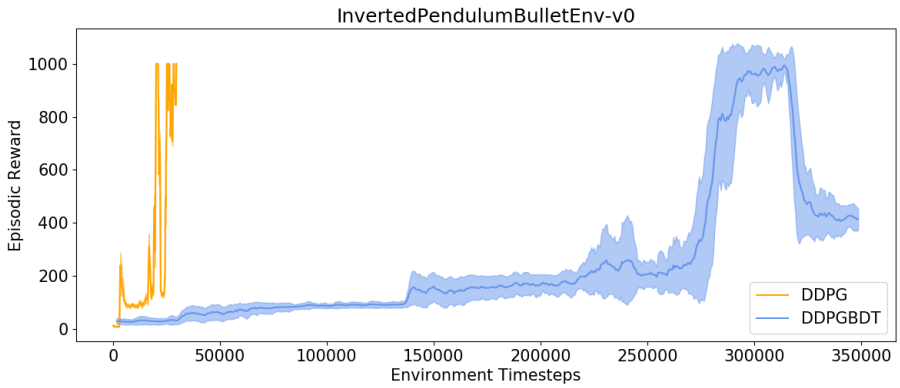

Figure 3: The learning curve for baseline DDPG vs. DDPGBDT

The non-tuned, out-of-the-box DDPG solved the environment in about one tenth of the amount of timesteps it took DDPGBDT. DDPG does still show high variance, as it falls to low performance after having solved the environment. However, DDPG is able to recover back to high scores in a way DDPGBDT does not.

5 Implementation Details

5.1 Basic Implementation Details

This project was implemented in Python and we used LightGBM for the Gradient Boosted Decision Trees. (Ke et al., 2017) We used the OpenAI Gym and PyBullet packages to to evaluate the algorithm on continuous control tasks. (Brockman et al., 2016; Coumans & Bai, 2016–2019) The code for this project is publicly available at `https://github.com/cthorrez/ddpgbdt`.

5.2 Details You Usually Can Only Figure Out From the Code

Mathematically speaking, the description in the previous sections is sufficient to define the models and an algorithm to train them, however there are many details in the implementation, without which the entire system fails.

5.2.1 Scaling and Training the Trees

One major weakness of using trees in this setting is that unlike neural networks, they grow when they are trained which means the next iteration is more computationally expensive. This is especially problematic in this setting as we are training two trees and training for a long time, such as hundreds of thousands of environment timesteps.

The mitigation we employed was to train infrequently and use very large batch sizes. In the end we only added new trees every 600 environment timesteps and when we did train, we trained using batch sizes of $50,000$ to be maximally data-efficient.

5.3 Reward Hacking

DDPGBDT was having trouble learning on most of the environments we tested it on such as `CartPoleContinuousBulletEnv-v0`, `InvertedPendulumSwingupBulletEnv-v0`, and `InvertedPendulumBulletEnv-v0`. We theorized that this was because the rewards at each timestep were always 1 so the trees never saw other values and the finite differences method was unable to produce non-zero gradients. So what we did was manually set the reward to -10 for timesteps in which the environment ended before the max time limit due to failure.

5.4 Rollbacks

Another novel trick we added was to rollback a training iteration if the performance of the model dropped significantly after an update. Something we noticed during development was steady improvement for a period of time and then one bad update would lose all progress and the learner never recovered. So we added logic that evaluates the model on 15 new episodes after each update and if the average sum of rewards has dropped by at least 25%, then we rollback both the actor and critic updates and multiply the learning rate by 0.75.

5.5 Hyperparameters

As the entire success of this algorithm requires the precise setting of all hyperparameters, we report them in Table 1.

Parameter Name	Symbol (if mentioned in paper)	Description	Value
gamma	γ	Discount factor for valuing rewards in the future compared to the present	0.99
learning_rate	η	The step size for new tree	0.05
min_child_samples		The minimum amount of values to be in a tree leaf	1
num_leaves		The maximum number of leaves in a tree	31
batch_size	N	The number of timestamps to train on during each training cycle	50,000
max_buffer_size		The number of recent tuples to store in the experience replay	60,000
rollback_thresh		The amount by which the reward must decrease to trigger a rollback of the update	25%
rollback_lr_decay		The amount by which to multiply the learning rate in the event of a rollback	0.75
train_every		The number of environment timesteps between training and eval iterations	600
num_timesteps		The total number of timesteps to train for	400,000
epsilon	ϵ	The small number used in finite differences calculations	0.01
eps		The exploration parameter. The initial probability of taking a random action during training.	0.75
eps_decay		The amount to multiply eps by each timestep	0.99
min_eps		The minimum chance of taking a random action during training	0.2
seed		The random seed for LightGBM, numpy random sampling, and OpenAI Gym	0

Table 1: The name, description and value for each hyperparameter used in DDPGBDT

6 Discussion

6.1 Strengths

Aside from the novel state-of-the-art acronym, the only semi-plausible advantage of DDPGBDT is that there is some degree of model interpretability. GBDT models have natural ways to calculate feature importance based on the frequency with which it is used as the splitting feature and the gains resulting from those splits. It is not inconceivable that feature importances could be used to gain insights as to why an agent behaves a certain way.

6.2 Weaknesses

There are a myriad of disadvantages to using this approach. Here are some of the most important ones.

- The size and complexity of additive decision tree models grows continuously during training, making this approach unsuitable for tasks which require long training.
- Decision tree predictors are non-continuous meaning their gradients do not exist. This forces us to rely on inefficient and high variance finite difference approximations.
- The original weaknesses of DDPG are amplified. DDPGBDT is even more brittle to slight changes in hyperparameters.
- DDPGBDT does not have a natural way to extend to environments with multi-dimensional action spaces. Multi-output GBDT is an area of active research but the implementations have not been incorporated into the popular GBDT packages. (Zhang & Jung, 2020)

- Perhaps the most damning criticism of all is that the acronym is not completely accurate. While it is undeniable that the model is still Deterministic, still uses a Policy Gradient, and still uses Gradient Boosted Decision Trees, when we swapped out the neural nets it could be said that we lost our Deep. :(

6.3 Conclusion

In this work we introduced DDPGBDT, a novel algorithm for reinforcement learning and demonstrated that with extreme hyperparameter tuning it is capable of solving one task when run with a specific random seed.

Despite this incredible success combined with the truly revolutionary nature of the name, this method does have drawbacks which can make it unsuitable for some use cases.

7 Acknowledgements

Thank you to Kegan Thorrez for informing me about SIGBOVIK and suggesting that my weird project might be suitable for submission here. Thank you to reddit user _ericrosen who suggested a pendulum environment when I asked for the easiest possible environment to test a bad reinforcement learning algorithm on.

References

Leo Breiman. Arcing classifiers. Technical report, 1996.

Leo Breiman. Arcing the edge. Technical report, 1997.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai Gym, 2016.

Erwin Coumans and Yunfei Bai. PyBullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1329–1338, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/duan16.html.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001. ISSN 00905364. URL http://www.jstor.org/stable/2699986.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

Peter Henderson, R. Islam, Philip Bachman, Joelle Pineau, Doina Precup, and D. Meger. Deep reinforcement learning that matters. In *AAAI*, 2018.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

Brook Taylor. *Methodus Incrementorum Directa et Inversa*. Impensis Gulielmi Innys, 1717.

Zhendong Zhang and Cheolkon Jung. Gbdt-mo: Gradient-boosted decision trees for multiple outputs. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2020. doi: 10.1109/TNNLS.2020.3009776.