Consistent Dropout for Policy Gradient Reinforcement Learning

Matthew Hausknecht 1 Nolan Wagener 2

Abstract
Dropout has long been a staple of supervised learning, but is rarely used in reinforcement learning. We analyze why naïve application of dropout is problematic for policy-gradient learning algorithms and introduce consistent dropout, a simple technique to address this instability. We demonstrate consistent dropout enables stable training with A2C and PPO in both continuous and discrete action environments across a wide range of dropout probabilities. Finally, we show that consistent dropout enables the online training of complex architectures such as GPT without needing to disable the model’s native dropout.

1. Introduction
Dropout, the practice of randomly zeroing a fraction of unit activations at select neural network layers, is an established technique for regularizing neural network learning and reducing overfitting (Srivastava et al., 2014). However, in the field of reinforcement learning, dropout is rarely applied. Consider the large body of work without dropout, including high-profile projects like AlphaGo (Silver et al., 2016), AlphaStar (Vinyals et al., 2019), and OpenAI Five (Berner et al., 2019) – none of which mention dropout. Even models that are commonly supervised-trained with dropout, such as Transformers (Vaswani et al., 2017), disable dropout entirely when repurposed for reinforcement learning. This trend is reinforced by our open source tools: Surveying six popular open source RL frameworks 1, we found none that supported dropout.

We contend that the naïve application of dropout in reinforcement learning can be detrimental to learning performance. To test this hypothesis, we added dropout layers to the policy networks of A2C and PPO agents in three RL frameworks: RLlib (Liang et al., 2017), Stable-Baselines3 (Raffin et al., 2019), and Tianshou (Weng et al., 2020). Agents were then trained on the MuJoCo Half-Cheetah-v3 continuous control environment using the hyperparameters recommended by their respective codebases. To summarize results in Appendix B, Tianshou’s A2C experienced large negative returns (e.g., $-10^7$) in five of sixteen runs, and all sixteen PPO runs prematurely terminated due to NaN values. Similarly, RLlib experienced instability in five of six A2C runs and two of six PPO runs. Stable-Baselines3, on the other hand, completed all runs without error, though we found that A2C experienced large policy loss magnitudes on seven of twelve runs and PPO declined in performance by 21% on average with dropout. Thus, we conclude that instability stemming from naïve application of dropout is not an artifact of a single implementation or codebase but instead affects to various degrees all of the tested frameworks. 2

To better understand why dropout is problematic, Section 4 shows the instability stems from different dropout masks being applied to the same observations during rollouts and updates. This mismatch leads to policy collapse for A2C and reduced performance for PPO. Furthermore, we show that more complex networks such as GPT (Radford et al., 2019) suffer more acutely than MLPs from the instabilities introduced by naïve application of dropout. To correct this instability, Section 5 introduces consistent dropout, a simple and mathematically sound technique that consists of saving the random dropout masks generated during rollouts and reapplying the same masks during updates. Experimental results in Section 6 show this technique stabilizes learning for both A2C and PPO on discrete and continuous environments across a spectrum of dropout probabilities. We also show this technique allows for GPT to be trained online without needing to disable dropout.

The contributions of this paper are: 1) Demonstrate that naïve application of dropout creates instability for policy gradient methods across a variety of open-source RL frameworks, algorithms, and environments. 2) Analyze the source of instability and introduce a simple method of saving and

---

1Microsoft Research 2Institute for Robotics and Intelligent Machines, Georgia Institute of Technology. Correspondence to: Matthew Hausknecht <matthew.hausknecht@gmail.com>, Nolan Wagener <nolan.wagener@gatech.edu>.

1Surveyed frameworks were Dopamine (Castro et al., 2018), RLlib (Liang et al., 2017), OpenAI Baselines (Dhariwal et al., 2017), Tianshou (Weng et al., 2020), Stable-Baselines3 (Raffin et al., 2019), and rlpyt (Stooke & Abbeel, 2019).

2Notably, the use of Tanh nonlinearities in policy networks reduced instability compared to ReLU. Although we did not perform a detailed investigation, we suspect this relates to Tanh bounding the network outputs.
2. Related Work

Farebrother et al. (2018) investigated dropout as a regularization technique for the DQN learning agent (Mnih et al., 2015) on Atari environments. Their findings indicate that moderate levels of dropout \((p = 0.1)\), paired with \(\ell_2\) regularization improved zero-shot generalization on variants of the training environment. However, the authors note that dropout slowed the training performance and typically required more training iterations to compete with the unregularized baseline.

Similarly, Cobbe et al. (2018) examined the effects of dropout and \(\ell_2\) regularization on the Procgen benchmark (Cobbe et al., 2019). Using the PPO algorithm, they found both \(\ell_1\) dropout and \(\ell_2\) dropout to be the state distribution induced by running \(\pi_\theta\) in \(\mathcal{M}\) for \(t\) time steps from the initial state distribution \(d_0\). Defining the state visitation \(d^\pi_t(s) = (1 - \gamma) \sum_{t=0}^\infty \gamma^t d^\pi_{t+1}(s)\), it can be shown (Sutton et al., 2000) that the policy gradient has the form

\[
\nabla J(\theta) = \mathbb{E}_{D_{s,a}} [A^\pi(s,a) \nabla_\theta \log \pi_\theta(a|s)]
\]

Practical implementations of policy gradient algorithms roll out \(\pi_\theta\) and collect state-action-reward triplets \((s, a, r)\) into a buffer \(D\). After learning an approximate advantage function \(\hat{A}(s, a)\) (e.g., through generalized advantage estimation (Schulman et al., 2016)), they form the loss function

\[
L_{PG}(\theta) = -\mathbb{E}_{(s,a) \sim D} [\hat{A}(s,a) \log \pi_\theta(a|s)]
\]

and subsequently perform a gradient descent step.

4. Why Dropout Disrupts Policy Gradient

We now assume that \(\pi_\theta\) is a multi-layer neural network where there is a subset of neurons wherein each activation is zeroed out with some probability, as shown in Fig. 1. Letting \(m\) be the resulting dropout mask, our stochastic policy has the form \(\pi_\theta(a|s) = \sum_m p(m) \pi_\theta(a|s, m)\), where \(p(m)\) is the probability function of the mask. In practice, dropout is treated as an unobserved internal process of the network, so that we observe a noisy estimator of \(\pi_\theta(a|s)\).

Due to this practice, at rollout time, we sample some dropout mask \(\bar{m}\) at a given state \(s\) and then sample an action from \(\pi_\theta(a|s, \bar{m})\) to apply in the MDP. At update time, we sample

\[
\nabla J(\theta) = \mathbb{E}_{D_{s,a}} [A^\pi(s,a) \nabla_\theta \log \pi_\theta(a|s)]
\]
Table 1. Forward passes over the same state with different dropout masks change output actions and reduce the log probabilities:

| NET       | DROPOUT | \(d(a^0,a^1)\) | \(\log \pi_\theta(a^0|s,m^1)\) |
|-----------|---------|----------------|----------------------------------|
| 3-LAYER   | 0       | 0 ± 0          | −0.83 ± 0.00                     |
| MLP w/    | 0.1     | 0.04 ± 0.03    | −0.84 ± 0.01                     |
| 4 CONTINUOUS ACTIONS | 0.25 | 0.06 ± 0.05 | −0.86 ± 0.02 |
| ACTIONS   | 0.9     | 0.63 ± 0.25    | −1.13 ± 0.38                     |
|           | 0.0     | 0 ± 0          | −0.83 ± 0.00                     |
| GPT w/    | 0.0     | 0 ± 0          | −0.83 ± 0.00                     |
| 4 CONTINUOUS ACTIONS | 0.25 | 0.06 ± 0.05 | −0.86 ± 0.02 |
| ACTIONS   | 0.75    | 0.25 ± 0.03    | −1.13 ± 0.38                     |
|           | 0.9     | 0 ± 0          | −0.83 ± 0.00                     |
| 3-LAYER   | 0.75    | 0.25 ± 0.03    | −1.13 ± 0.38                     |
| MLP w/    | 0.1     | 0.18 ± 0.39    | −1.35 ± 0.04                     |
| 4 DISCRETE ACTIONS | 0.25 | 0.51 ± 0.50 | −1.34 ± 0.05 |
| ACTIONS   | 0.5     | 0.60 ± 0.48    | −1.34 ± 0.10                     |
|           | 0.75    | 0.71 ± 0.47    | −1.39 ± 0.21                     |
|           | 0.9     | 0.75 ± 0.43    | −1.51 ± 0.58                     |

A new dropout mask when querying the policy for its log-probability. This results in a modified loss function for policy optimization:

\[
L_{PG}^{\theta}(\theta) = -E_{(s,a) \sim D}E_{p(m)}[\tilde{A}(s,a) \log \pi_\theta(a|s,m)].
\]

The use of different dropout masks for the same state \(s\) at rollout time versus update time can lead to vastly different action probabilities. For A2C, this manifests in \(\pi_\theta(a|s)\) being close to zero and \(\log \pi_\theta(a|s)\) approaching negative infinity. At this point, updates become unstable as policy loss and gradient magnitudes explode. In practice, gradient norm clipping can keep training numerically stable, but learning performance still suffers due to the gradient estimator having high variance. Table 1 demonstrates how the application of different dropout masks can change the outputs and lower the log probabilities of actions in an untrained, newly initialized network. These effects only become more pronounced over the course of training.

4.1. Dropout and PPO

PPO (Schulman et al., 2017) optimizes a clipped objective that ensures the deviation of the current policy \(\pi_\theta\) from the rollout policy \(\pi_{\theta_{old}}\) is small:

\[
L_{PPO}(\theta) = -E_D[\min(r(s,a,\theta)\tilde{A}(s,a),
\text{clip}(r(s,a,\theta), 1 - \epsilon, 1 + \epsilon)\tilde{A}(s,a))],
\]

where \(r(s,a,\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}\) is the ratio of the action probabilities under the new and old policies, respectively, and \(\epsilon\) is a small constant which determines how far the policy is allowed to diverge from the old policy. In addition to this clipped objective, PPO also has the option to stop taking gradient steps if the KL divergence between the new and old policies grows past a threshold. This early stopping mechanism can help ensure monotonic policy improvement by preventing gradient updates from extrapolating too far in any direction.

Adding dropout to the policy network increases the variance in the ratio of action probabilities \(r\). The PPO objective clips this variance and prevents policy loss \(L_{PPO}(\theta)\) from exploding. When using an early stopping threshold, we find that increased variance in \(r\) will often trigger immediate early stopping, preventing any updates from taking place. To preface our results, without early stopping we observe stable learning but notably reduced performance.

5. Policy Gradient with Dropout

We now present two methods to correctly compute the policy gradient in Eq. (1) in the presence of dropout.

5.1. Dropout-Marginalized Gradient

This method gives the correct policy gradient by marginalizing over possible dropout masks that are randomly sampled at update time: For some \(s \in S\) and \(a \in A\), the policy’s score function can be computed as:

\[
\nabla_\theta \log \pi_\theta(a|s) = \nabla_\theta \log \left( \sum_m p(m) \pi_\theta(a|s,m) \right)
= \sum_m p(m) \pi_\theta(a|s,m) \nabla_\theta \log \pi_\theta(a|s,m)
= \sum_m p(m|s,a,\theta) \nabla_\theta \log \pi_\theta(a|s,m).
\]

Thus, the score function is the expected score function of the dropout-conditioned policy under the posterior distribution of the dropout mask.

In this work, we use Eq. (3) to compute the score function. Since it is intractable to enumerate every dropout mask, we instead rely on sampling. For some \(s \in S\) and \(a \in A\), we sample \(N\) dropout masks \(m_1, \ldots, m_N\) i.i.d. from \(p(m)\).
Consider the modified MDP (Fig. 2) in which the policy outputs both the action and the dropout mask \(a, m\), so that the stochastic policy is now \(\pi_\theta(a, m|s) = p(m|s)\pi_\theta(a|s, m)\). This MDP is equivalent to the original since neither the dynamics nor the reward are modified, i.e., \(p(s'|s, a, m) = p(s'|s, a)\) and \(R(s, a, m) = R(s, a)\). As a consequence, the advantage function \(A^\pi\) remains a function only of the state \(s\) and original action \(a\).

Now, the score function of the policy is:

\[
\nabla_\theta \log \pi_\theta(a, m|s) = \nabla_\theta \left( \log p(m) + \log \pi_\theta(a|s, m) \right)
\]

so that the resultant policy gradient is

\[
\nabla J(\theta) = \mathbb{E}_{q_\pi(\theta)} \mathbb{E}_{\pi_\theta(a, m|s)} \left[ A^\pi(s, a) \nabla_\theta \log \pi_\theta(a, m|s) \right]
\]

\[
= \mathbb{E}_{q_\pi(\theta)} \mathbb{E}_{\pi_\theta(a, m|s)} [A^\pi(s, a) \nabla_\theta \log \pi_\theta(a|s, m)].
\]

Thus, we conclude that conditioning the policy on the same mask \(m\) that is sampled during rollouts results in an unbiased policy gradient at update time. Finally, we note that while the derivations in the preceding sections are specialized to dropout, they can be easily extended to other networks that internally use stochasticity, such as Bayesian neural networks and hidden layers that rely on sampling.

5.3. Implementation

From an implementation perspective, due to the computational challenges of using the dropout-marginalized gradient (explored in Section 6.1), we focus on the dropout-conditioned gradient for the remainder of this work and refer to it as consistent dropout. At rollout time, for some state \(s\), we sample a mask \(m\) and action \(a\) from \(\pi_\theta(a, m|s)\), observe a reward \(r\), and store the tuple \((s, a, r, m)\) into a buffer \(D\). During subsequent updates, the saved masks can be reapplied with the corresponding states and actions. The resulting policy loss we optimize has the form:

\[
L_{PG}(\theta) = -\mathbb{E}_{(s,a,r,m) \sim D} \left[ \hat{A}(s, a) \log \pi_\theta(a|s, m) \right].
\]

This reuse of dropout masks ensures that any change in the policy network’s output is due to a change in the policy network’s weights, rather than a different dropout mask.

To facilitate easy storage and retrieval of masks, we design a custom dropout layer shown in Figure 3 that retains the most recent masks from the forward pass, where they can be easily stored and reapplied at update time. From a computational perspective, this approach is essentially just as fast as the original, but does require additional storage of dropout masks whose size and number vary based on the architecture of the policy net. However, by storing masks as binary arrays, we find they often take less space than states or actions. Finally, additional source code and examples showing the use of this consistent dropout layer may be found in the supplementary materials.

6. Experiments

In order to compare consistent and inconsistent dropout, we evaluate A2C and PPO on several popular continuous and discrete environments. In particular, the aim of our experiments is to empirically understand the effects of inconsistent dropout on policy gradient algorithms and to understand if consistent dropout stabilizes policy gradient learning.

6.1. Continuous-Action Environments

We conduct experiments on the Hopper, Walker2d, and Half-Cheetah MuJoCo tasks to compare the stability and performance of A2C and PPO when trained with consistent and inconsistent dropout probabilities of \((0/0.1/0.25/0.5)\).

Results on Half-Cheetah are shown in Figure 5 and similar results from the other MuJoCo tasks may be found in Appendix C. Across all MuJoCo environments, we find
import torch
from torch import nn

class ConsistentDropout(nn.Module):
    """
    Consistent Dropout Layer uses masks provided in source and stores masks in sink.
    """
    def __init__(self, source, sink, p=0.5):
        super().__init__()
        self.source = source
        self.sink = sink
        self.p = p
        self.scale_factor = 1 / (1 - self.p)
    def forward(self, x):
        if self.training:
            if self.source:
                mask = self.source.pop(0)
            else:
                mask = torch.bernoulli(x.data.new(x.data.size()).fill_(1 - self.p))
                self.sink.append(mask.bool())
            return x * mask * self.scale_factor
        else:
            return x

class ConsistentPolicyNet(nn.Module):
    def __init__(self, obs_size, action_size, hidden_size, dropout):
        super().__init__()
        self.source, self.sink = [], []
        self.policy_net = nn.Sequential(
            nn.Linear(obs_size, hidden_size),
            nn.ReLU(),
            ConsistentDropout(self.source, self.sink, p=dropout),
            nn.Linear(hidden_size, hidden_size),
            nn.ReLU(),
            ConsistentDropout(self.source, self.sink, p=dropout),
            nn.Linear(hidden_size, action_size)
        )
    def forward(self, obs, dropout_masks=None):
        if dropout_masks:
            self.source.extend(dropout_masks)
        action = self.policy_net(obs)
        masks = self.sink.copy()
        self.source.clear()
        self.sink.clear()
        return action, masks

Figure 3. Consistent Dropout Layer: PyTorch implementation of the consistent dropout layer and modifications needed to integrate into a policy network: The ConsistentDropout layer will retrieve and apply a dropout mask if provided in source and save the used dropout mask to sink. ConsistentPolicyNet demonstrates the integration of the ConsistentDropout layer into a policy network: Both source and sink are implemented using lists shared between all dropout layers in the network. In the forward pass, if dropout masks are provided they will be applied in correct order. Otherwise, random masks will be generated and returned. Returned masks can then be stored in the agent’s replay memory and reapplied at update time.
We tested three discrete-action Atari environments (Beamrider, Breakout, and Seaquest), and as shown in Table 2 A2C is highly unstable for any level of inconsistent dropout. PPO is more robust to inconsistent dropout and often remains stable and performant at dropout of 0.1. However, dropout probabilities of 0.25 and 0.5 degrade PPO’s performance significantly.

On the other hand, as shown in Table 2, consistent dropout effectively stabilizes A2C and PPO across all MuJoCo environments and dropout levels. All levels of consistent dropout lead to stable learning, although dropout of 0.5 notably reduces performance for both algorithms.

In terms of absolute performance, baseline models without dropout perform slightly better on average than equivalent models trained with dropout. This finding reflects the intuition that when training and testing on the same environment, less regularization is likely beneficial.

We also experimented with a version of PPO that uses an estimate of the marginalized gradient in Eq. (3). As shown in Fig. 4, the large variance of the gradient estimate dominates the learning, even with 100 samples used for Eq. (3).

6.2. Discrete-Action Environments

We tested three discrete-action Atari environments (Beamrider, Breakout, and Seaquest), and as shown in Table 2 A2C and Consistent PPO (PPO-C) achieve significantly more performance than their inconsistent counterparts on all MuJoCo environments. In Atari environments, PPO and PPO-C are much closer in relative performance with the trend favoring PPO-C.

Prior work on training transformer models with reinforcement learning disable dropout (Parisotto et al., 2019; Loynd et al., 2020). This is unsurprising as even small levels of dropout cause much larger changes to GPT’s outputs compared to similarly deep MLP networks (Table 1). As a result, PPO models trained with inconsistent dropout are unable to exceed random performance on Half-Cheetah. On the other hand, consistent dropout (Fig. 7) enables stable GPT training across all dropout probabilities. Extrapolating from this result, we believe consistent dropout is applicable to a variety of network architectures.

7. Evaluating with Dropout

Common practice for supervised learning is to disable dropout at evaluation time. However, it is unclear whether this practice would be beneficial for sequential decision making. To answer this question, we examined the effects of enabling or disabling dropout at evaluation time, and found a strong preference towards disabling dropout at evaluation time (Fig. 8). Specifically we observe a 2% improvement on average in final evaluation score when disabling dropout at evaluation time for the $p = 0.1$ models, a 17% improvement for the $p = 0.25$ models, and a 46% improvement in
Figure 5. MuJoCo-HalfCheetah-v3: Clipped score of A2C and PPO with and without consistent dropout probabilities 0/0.1/0.25/0.5. Curves are averages of three independent training runs.

Figure 6. Atari-Beamrider: Clipped score of A2C and PPO with inconsistent and consistent dropout probabilities 0/0.1/0.25/0.5. Curves reflect averages of three independent training runs.

Figure 7. GPT Training: PPO training of GPT models on HalfCheetah-v3. With any level of inconsistent dropout all PPO models are unable to learn. Consistent dropout stabilizes training for all dropout probabilities.

score for the $p = 0.5$ models. Finally, we find that models trained with inconsistent dropout are largely unaffected by enabling or disabling dropout at evaluation time. Further evaluation results may be found in Appendix C.

8. Limitations

This study pertains only to on-policy policy gradient methods. Other types of reinforcement learning algorithms such as off-policy and model-based reinforcement learning are affected by dropout in different ways, which are beyond the scope of this paper.

Having stabilized dropout for policy gradient algorithms, our findings raise the larger question of whether dropout aids generalization to unseen tasks or novel variations of training environments. Related work (Cobbe et al., 2018; Farebrother et al., 2018) has found beneficial generalization effects from small amounts of dropout, and its common use in supervised settings suggests that dropout may aid generalization especially when training overparameterized networks. However, we expect that thoroughly exploring how dropout contributes to generalization in sequential decision settings is a substantial undertaking, one that lies beyond the scope of this work.
9. Conclusions

We demonstrate that naïve application of dropout to policy gradient reinforcement learning results in instability due to the mismatch between the dropout masks that are applied during rollouts and those used during updates. We address this instability through consistent dropout, a simple approach for saving and reapplying the same dropout masks during rollouts and updates. We demonstrate that this solution is both theoretically sound and empirically stable on a variety of continuous and discrete actions environments. Additionally, we show that this technique scales to enable stable online training of transformer models such as GPT. Given the complete lack of support for dropout in open-source reinforcement learning frameworks, we hope this work will pave the way for a re-evaluation of this time-tested technique.

References

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. Dota 2 with large scale deep reinforcement learning. CoRR, abs/1912.06680, 2019. URL http://arxiv.org/abs/1912.06680.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018. URL http://arxiv.org/abs/1812.06110.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. CoRR, abs/2106.01345, 2021. URL https://arxiv.org/abs/2106.01345.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying generalization in reinforcement learning. CoRR, abs/1812.02341, 2018. URL http://arxiv.org/abs/1812.02341.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. arXiv preprint arXiv:1912.01588, 2019.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. OpenAI Baselines. https://github.com/openai/baselines, 2017.

Farebrother, J., Machado, M. C., and Bowling, M. Generalization and regularization in DQN. CoRR, abs/1810.00123, 2018. URL http://arxiv.org/abs/1810.00123.

Janner, M., Li, Q., and Levine, S. Offline reinforcement learning as one big sequence modeling problem. Advances in Neural Information Processing Systems, 34, 2021.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Gonzalez, J., Goldberg, K., and Stoica, I. Ray RL-Lib: A Composable and Scalable Reinforcement Learning Library. CoRR, abs/1712.09381, 2017. URL http://arxiv.org/abs/1712.09381.

Loynd, R., Fernandez, R., Celikyilmaz, A., Swaminathan, A., and Hausknecht, M. Working memory graphs. In International Conference on Machine Learning, pp. 6404–6414. PMLR, 2020.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. Nature, 518 (7540):529–533, February 2015. ISSN 0028-0836. URL http://dx.doi.org/10.1038/nature14236.

Parisotto, E., Song, H. F., Rae, J. W., Pascanu, R., Gülçehre, Ç., Jayakumar, S. M., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., Botvinick, M. M., Heess, N., and Hadsell, R. Stabilizing transformers for reinforcement learning. CoRR, abs/1910.06764, 2019. URL http://arxiv.org/abs/1910.06764.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., and Dormann, N. Stable-Baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

Schulman, J., Heess, N., Weber, T., and Abbeel, P. Gradient estimation using stochastic computation graphs. In Advances in Neural Information Processing Systems, 2015.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In International Conference on Learning Representations, 2016.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. CoRR, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.
Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016. ISSN 0028-0836. doi: 10.1038/nature16961.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Stooke, A. and Abbeel, P. rlpyt: A Research Code Base for Deep Reinforcement Learning in PyTorch. *CoRR*, abs/1909.01500, 2019. URL http://arxiv.org/abs/1909.01500.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pp. 1057–1063, 2000.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D., and Silver, D. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

Weng, J., Zhang, M., Yan, D., Su, H., and Zhu, J. Tianshou. https://github.com/thu-ml/tianshou, 2020.
A. Hyperparameters

Table 3. **Hyperparameters**: Networks consisted of 3-layer MLPs with ReLU nonlinearities. Dropout was applied between each layer of the network. Note that PPO Consistent uses a target-KL early stopping condition to terminate updates when the KL divergence passes a threshold. We attempted to run standard PPO with the same parameter but found that it would trigger early termination due to the large KL divergences caused by inconsistent dropout masks. Thus, the best performance for standard PPO was with the target-KL disabled.

| VARIANT | HYPERPARAMETER | MuJoCo | Atari |
|---------|----------------|--------|--------|
| A2C     | MAX STEPS      | 1E8    | 1E8    |
|         | WORKERS        | 16     | 16     |
| A2C CONSISTENT | STEPS PER EPOCH | 80    | 5      |
|         | LEARNING RATE  | 7E-4   | 1E-4   |
|         | CRITIC LR      | 7E-4   | -      |
|         | ENTROPY COEFFICIENT | 0.01 | 0.01 |
|         | DISCOUNT FACTOR | 0.99  | 0.99   |
|         | GAE-LAMBDA     | 0.95   | 0.95   |
|         | HIDDEN SIZE    | 64     | 512    |
|         | GRADIENT NORM CLIP | 0.5  | 0.5    |
|         | RMS PROP EPSILON | 3E-6  | 3E-6   |
|         | ADVANTAGE NORMALIZATION | 1    | 0      |
| PPO     | MAX STEPS      | 1E8    | 1E8    |
|         | WORKERS        | 16     | 16     |
|         | LEARNING RATE  | 3E-4   | 1E-4   |
|         | STEPS PER EPOCH | 4096  | 4096   |
|         | GRADIENT STEPS PER UPDATE | 16  | 16    |
|         | BATCH SIZE     | 64     | 64     |
|         | HIDDEN SIZE    | 64     | 512    |
|         | VALUE COEFFICIENT | 0.5  | 0.5    |
|         | ENTROPY COEFFICIENT | 0.01 | 0      |
|         | GRADIENT NORM CLIP | 0.5  | 0.5    |
|         | DISCOUNT FACTOR | 0.99  | 0.99   |
|         | GAE-LAMBDA     | 0.97   | 0.95   |
|         | CLIP RATIO     | 0.2    | 0.2    |
|         | TARGET KL      |        |        |
| PPO CONSISTENT | MAX STEPS      | 1E8    | 1E8    |
|         | WORKERS        | 16     | 16     |
|         | LEARNING RATE  | 3E-4   | 1E-4   |
|         | STEPS PER EPOCH | 4096  | 4096   |
|         | GRADIENT STEPS PER UPDATE | 16  | 16    |
|         | BATCH SIZE     | 64     | 64     |
|         | HIDDEN SIZE    | 64     | 512    |
|         | VALUE COEFFICIENT | 0.5  | 0.5    |
|         | ENTROPY COEFFICIENT | 0.01 | 0      |
|         | GRADIENT NORM CLIP | 0.5  | 0.5    |
|         | DISCOUNT FACTOR | 0.99  | 0.99   |
|         | GAE-LAMBDA     | 0.97   | 0.95   |
|         | CLIP RATIO     | 0.2    | 0.2    |
|         | TARGET KL      | 0.01   | 0.01   |
| GPT     | MAX STEPS      | 1E8    |        |
|         | WORKERS        | 16     |        |
| GPT CONSISTENT | CRITIC LEARNING RATE | 7E-4 |        |
|         | STEPS PER EPOCH | 1024  |        |
|         | GRADIENT STEPS PER UPDATE | 128 |        |
|         | BATCH SIZE     | 64     |        |
|         | HIDDEN SIZE    | 64     |        |
|         | ENTROPY COEFFICIENT | 0.01 |        |
|         | GRADIENT NORM CLIP | 0.5  |        |
|         | DISCOUNT FACTOR | 0.99  |        |
|         | GAE-LAMBDA     | 0.97   |        |
|         | CLIP RATIO     | 0.2    |        |
|         | TARGET KL      | 0.01   |        |
|         | BLOCK SIZE     | 8      |        |
|         | NUMBER OF LAYERS | 4     |        |
|         | NUMBER OF HEADS | 4      |        |
B. RL Framework Testing

The following tables show results from testing inconsistent dropout on three open-source reinforcement learning frameworks. This was accomplished by adding dropout layers to the policy networks and running the A2C and PPO on the HalfCheetah-v3 environment using the hyperparameters suggested by the respective codebase. In all the following tables, where ReLU was not used, Tanh was applied.

Table 4. Tianshou (version 0.4.2) on HalfCheetah-v3: For A2C, ReLU activation seems to be the biggest causative factor of instability, while Tanh is a stabilizer. ActionClipping can also help stabilize when dropout levels are moderate. Reward normalization may have a slightly helpful effect, but less than other factors. PPO even with moderate levels of dropout demonstrated broad instability: PPO† indicates early termination of the experiment due to instabilities resulting in NaN values.

| ALG | DROPOUT | REW | ACT | RELU | BEST | FINAL |
|-----|---------|-----|-----|------|------|-------|
| A2C | 0       | ✔   | ✔   |  ✔   | 1556 | 1542  |
| A2C | 0       | ✔   | ✔   |  ✔   | 2949 | 2927  |
| A2C | 0.1     | ☑   | ✔   |  ✔   | 3192 | 2969  |
| A2C | 0.1     | ☑   | ✔   |  ✔   | 3085 | 2693  |
| A2C | 0.1     | ☑   | ✔   |  ✔   | 3593 | 3357  |
| A2C | 0.1     | ☑   | ✔   |  ✔   | 1628 | −3e7  |
| A2C | 0.1     | ✔   | ✔   |  ✔   | 1193 | 1152  |
| A2C | 0.1     | ✔   | ✔   |  ✔   | 537  | 573   |
| A2C | 0.1     | ✔   | ✔   |  ✔   | 3066 | 2900  |
| A2C | 0.1     | ✗   | ✔   |  ✔   | 3204 | −1.2e8|
| A2C | 0.25    | ✗   | ✔   |  ✔   | 1049 | 977   |
| A2C | 0.25    | ✗   | ✔   |  ✔   | 171  | −13   |
| A2C | 0.25    | ✗   | ✔   |  ✔   | 1065 | −3.4e7|
| A2C | 0.25    | ✗   | ✔   |  ✔   | −146 | −3.9e9|
| A2C | 0.25    | ✔   | ✔   |  ✔   | 1141 | 1139  |
| A2C | 0.25    | ✔   | ✔   |  ✔   | 1137 | 1048  |
| A2C | 0.25    | ✔   | ✔   |  ✔   | 3872 | 3313  |
| A2C | 0.25    | ✔   | ✔   |  ✔   | 260  | −2.1e9|
| PPO | 0       | ✔   | ✔   |  ✔   | 7472 | 5645  |
| PPO | 0       | ✔   | ✔   |  ✔   | 8743 | 7890  |
| PPO†| 0.1     | ✔   | ✔   |  ✔   | 1708 | −335  |
| PPO†| 0.1     | ✔   | ✔   |  ✔   | 1548 | 463   |
| PPO†| 0.1     | ✔   | ✔   |  ✔   | 2267 | 1162  |
| PPO†| 0.1     | ✔   | ✔   |  ✔   | 1372 | 1050  |
| PPO†| 0.1     | ✔   | ✔   |  ✔   | 2238 | 1813  |
| PPO†| 0.1     | ✔   | ✔   |  ✔   | 408  | 42    |
| PPO†| 0.1     | ✔   | ✔   |  ✔   | 2268 | 1870  |
| PPO†| 0.1     | ✔   | ✔   |  ✔   | 1007 | −55   |
| PPO†| 0.25    | ✔   | ✔   |  ✔   | 1001 | 881   |
| PPO†| 0.25    | ✔   | ✔   |  ✔   | 279  | 153   |
| PPO†| 0.25    | ✔   | ✔   |  ✔   | 1270 | 1006  |
| PPO†| 0.25    | ✔   | ✔   |  ✔   | 614  | 246   |
| PPO†| 0.25    | ✔   | ✔   |  ✔   | 1292 | 1167  |
| PPO†| 0.25    | ✔   | ✔   |  ✔   | 705  | −123  |
| PPO†| 0.25    | ✔   | ✔   |  ✔   | 348  | 1321  |
| PPO†| 0.25    | ✔   | ✔   |  ✔   | 348  | −160  |
Table 5. Stable-Baselines3 (version 1.2.0a0) on HalfCheetah-v3. Policy loss is shown in to provide an intuition on about the stability of the training process. Very high policy losses correlate to high-magnitude $\log \pi(a|s)$ values, a symptom of unstable dynamics induced by inconsistent dropout.

| ALG | DROPOT | ReLU? | Ortho Init? | Final | Policy Loss |
|-----|--------|-------|-------------|-------|-------------|
| A2C | 0      | ✗     | ✗           | 974   | 2.5         |
| A2C | 0      | ✗     | ✓           | 688   | −1.1        |
| A2C | 0      | ✓     | ✗           | 526   | 8           |
| A2C | 0.1    | ✗     | ✓           | 1954  | −5.4        |
| A2C | 0.1    | ✓     | ✗           | 1003  | −12.1       |
| A2C | 0.1    | ✓     | ✓           | 650   | −8.8        |
| A2C | 0.1    | ✓     | ✗           | 1639  | 8.8e3       |
| A2C | 0.1    | ✓     | ✓           | −40   | −242        |
| A2C | 0.25   | ✗     | ✗           | 1325  | 26          |
| A2C | 0.25   | ✓     | ✗           | 499   | 25          |
| A2C | 0.25   | ✓     | ✓           | 1497  | −2.9e5      |
| A2C | 0.25   | ✓     | ✓           | 950   | −2.7e5      |
| A2C | 0.5    | ✗     | ✗           | 116   | 1e3         |
| A2C | 0.5    | ✓     | ✗           | 639   | −7.2e3      |
| A2C | 0.5    | ✓     | ✗           | 1527  | 1.7e8       |
| A2C | 0.5    | ✓     | ✓           | 1083  | −7.8e8      |

| PPO | 0      | ✗     | ✓           | 2238  | −0.007      |
| PPO | 0      | ✓     | ✗           | 3211  | −0.001      |
| PPO | 0      | ✓     | ✗           | 1272  | −0.0006     |
| PPO | 0      | ✓     | ✓           | 1277  | −0.001      |
| PPO | 0.1    | ✗     | ✗           | 1963  | 0.04        |
| PPO | 0.1    | ✓     | ✗           | 1256  | 0.05        |
| PPO | 0.1    | ✓     | ✓           | 1830  | 0.04        |
| PPO | 0.1    | ✓     | ✓           | 2463  | 0.04        |
| PPO | 0.25   | ✗     | ✗           | 1468  | 0.04        |
| PPO | 0.25   | ✓     | ✗           | 1383  | 0.04        |
| PPO | 0.25   | ✓     | ✓           | 2217  | 0.05        |
| PPO | 0.25   | ✓     | ✓           | 1825  | 0.05        |
| PPO | 0.5    | ✗     | ✗           | 1331  | 0.06        |
| PPO | 0.5    | ✓     | ✓           | 1328  | 0.06        |
| PPO | 0.5    | ✓     | ✗           | 904   | 0.06        |
| PPO | 0.5    | ✓     | ✓           | 993   | 0.05        |

Table 6. RLlib (version 1.8.0) on HalfCheetah-v3. Dagger denotes training became numerically unstable and produced errors. For these runs, we report the final performance level before the error.

| ALG | DROPOT | ReLU? | Final |
|-----|--------|-------|-------|
| A2C | 0      | ✗     | 712   |
| A2C | 0      | ✓     | 747   |
| A2C | 0.1    | ✗     | −393  |
| A2C | 0.1    | ✓     | −439  |
| A2C | 0.25   | ✗     | −253  |
| A2C | 0.25   | ✓     | −594  |
| A2C | 0.5    | ✗     | −643  |
| A2C | 0.5    | ✓     | −596  |
| PPO | 0      | ✗     | 2044  |
| PPO | 0      | ✓     | 5390  |
| PPO | 0.1    | ✗     | 909   |
| PPO | 0.1    | ✓     | −109  |
| PPO | 0.25   | ✗     | 383   |
| PPO | 0.25   | ✓     | −238  |
| PPO | 0.5    | ✗     | 867   |
| PPO | 0.5    | ✓     | 1517  |
C. Full MuJoCo Results

Hyperparameters were tuned on Half-Cheetah-v3 then deployed without further modification on Hopper-v3 and Walker2d-v3.

Figure 9. **Half-Cheetah-v3**: Training performance of various models on MuJoCo Half-Cheetah-v3. Curves are averages of three independent training runs.

Figure 10. **Hopper-v3**: Training performance of various models on MuJoCo Hopper-v3. Curves are averages of three independent training runs.

Figure 11. **Walker2d-v3**: Training performance of various models on MuJoCo Walker2d-v3. Curves are averages of three independent training runs.

Figure 12. **Half-Cheetah-v3 Deterministic Evaluation with Dropout**: Curves are averages of three independent training runs.
Figure 13. **Hopper-v3 Deterministic Evaluation with Dropout**: Curves are averages of three independent training runs.

Figure 14. **Walker2d-v3 Deterministic Evaluation with Dropout**: Curves are averages of three independent training runs.

Figure 15. **Half-Cheetah-v3 Deterministic Evaluation No-Dropout**: Curves are averages of three independent training runs.

Figure 16. **Hopper-v3 Deterministic Evaluation No-Dropout**: Curves are averages of three independent training runs.

Figure 17. **Walker2d-v3 Deterministic Evaluation No-Dropout**: Curves are averages of three independent training runs.
D. Full Atari Results

Hyperparameters were tuned for Beamrider then deployed without further modification on Breakout and Seaquest. The lower performance from PPO without dropout versus PPO Consistent without dropout is due to the lack of target-KL early stopping threshold for PPO. If this threshold is not disabled, PPO, due to inconsistent dropout, fails to perform any policy updates and makes no learning progress.

Figure 18. **Beamrider**: Training performance of various models on Atari Beamrider. Curves are averages of three independent training runs.

Figure 19. **Breakout**: Training performance of various models on Atari Breakout. Curves are averages of three independent training runs.

Figure 20. **Seaquest**: Training performance of various models on Atari Seaquest. Curves are averages of three independent training runs. Poor performance from all algorithms indicates hyperparameters which were tuned for Beamrider are suboptimal for Seaquest.

Figure 21. **Beamrider Deterministic Evaluation with Dropout**: Curves are averages of three independent training runs.
Figure 22. Breakout Deterministic Evaluation with Dropout: Curves are averages of three independent training runs.

Figure 23. Seaquest Deterministic Evaluation with Dropout: Curves are averages of three independent training runs.

Figure 24. Beamrider Deterministic Evaluation with No-Dropout: Curves are averages of three independent training runs.

Figure 25. Breakout Deterministic Evaluation with No-Dropout: Curves are averages of three independent training runs.

Figure 26. Seaquest Deterministic Evaluation with No-Dropout: Curves are averages of three independent training runs.