Single-Shot Pruning for Offline Reinforcement Learning

Samin Yeasar Arnob
Mila, McGill University
samin.arnob@mail.mcgill.ca

Riyasat Ohib
Georgia Institute of Technology
Atlanta, GA
riyasat.ohib@gatech.edu

Sergey Plis
Georgia State University
Atlanta, GA
splis@gmail.com

Doina Precup
Mila, McGill University,
DeepMind

Abstract

Deep Reinforcement Learning (RL) is a powerful framework for solving complex real-world problems. Large neural networks employed in the framework are traditionally associated with better generalization capabilities, but their increased size entails the drawbacks of extensive training duration, substantial hardware resources, and longer inference times. One way to tackle this problem is to prune neural networks leaving only the necessary parameters. State-of-the-art concurrent pruning techniques for imposing sparsity perform demonstrably well in applications where data-distributions are fixed. However, they have not yet been substantially explored in the context of RL. We close the gap between RL and single-shot pruning techniques and present a general pruning approach to the Offline RL. We leverage a fixed dataset to prune neural networks before the start of RL training. We then run experiments varying the network sparsity level and evaluating the validity of *pruning at initialization* techniques in continuous control tasks. Our results show that with 95% of the network weights pruned, Offline-RL algorithms can still retain performance in the majority of our experiments. To the best of our knowledge no prior work utilizing pruning in RL retained performance at such high levels of sparsity. Moreover, *pruning at initialization* techniques can be easily integrated into any existing Offline-RL algorithms without changing the learning objective.

1 Introduction

Function approximation with deep neural networks has been extremely successful in the last decade for a range of complex tasks. However, this performance comes at a significant computational cost and excessive memory requirements due to their good optimization and generalization performance in the highly overparamaterized region [32, 24, 1, 33]. It is an attractive proposition to prune such large networks with negligible loss in performance for real-time applications specially on edge devices with resource constraints. Pruning of networks [16, 12, 4, 11] has demonstrated reduction in inference-time resource requirements with minimal performance loss. The standard approach is to prune network after training and then perform a costly retraining step, thus requiring an extended training regime to generate sparse networks. Moreover, it is also difficult to train sparse networks from scratch which maintains similar performance to their dense counterparts

Offline Reinforcement Learning Workshop at Neural Information Processing Systems, 2021.

[10, 18]. Although, pruning before training is difficult, there are significant benefits in time and resource efficiency if we can prune networks before training.

In recent times, the Lottery Ticket Hypothesis [5] was proposed that details the presence of sub-architectures within a larger network, which when trained are capable of reaching the baseline accuracy of the dense networks or even surpass them in some cases. The sparsity in these sub-architectures can be exploited with suitable hardwares for computational efficiency gains, such as in [3], where authors managed to demonstrate a 5x efficiency gain while training networks with pre-specified sparsity.

There are a range of techniques in literature that provide methods to prune Deep Neural Networks at various stages of their training and testing. The most common of these techniques is to prune the network after training using some sort of predefined criterion that captures the significance of the parameters of the network to the objective function. A range of classical works on pruning used the second derivative information of the loss function [16, 12]. Perhaps the most intuitive of these approaches is magnitude pruning, where following training a subset of the parameters below some threshold is pruned and the rest of the parameters are retrained [10, 11] and regularization based methods [30, 23, 22, 31] which induces sparsity in the network during the optimization process.

Other more elaborate techniques to find the lottery tickets include solving a separate optimization problem to find out the subset of weights of the lottery-ticket the sub-architecture [34, 9, 19]. However, pruning of randomly initialized network before training still seemed like a difficult task, as the connections of a randomly initialized network exhibits little information about their significance to the training process. This, however, was changed with the proposal of *Single-shot Network Pruning* (SNIP) [17], which managed to prune weights before training with great success by finding out sparse trainable sub-architectures. With SNIP it has been demonstrated that it is indeed possible to prune neural networks in one-shot at initialization. A recent work [29] challenges SNIP's pruning criterion of *connection sensitivity* and argues that this is sub-optimal as the gradient of each weight is susceptible to change after pruning due to complex interactions among weights. Therefore, with this technique there is a possibility of pruning weights that are vital for the flow of information through the network. Instead, the authors of [29] propose an alternative method, *Gradient Signal Preservation* (GraSP), that preserves the gradient flow of the network. These techniques where the network weights are pruned before training can be termed as *pruning at initialization.*

There are a few recent works [27, 21] that leverages different pruning techniques in Deep-RL algorithms but they prune the neural-network in-between Online-RL training. Since the RL agent gets updated in every iteration and collects data through environment interactions, there are significant shifts in the data-distribution. Thus, it makes harder for the pruning techniques to find the proper sub-networks that can perform the same. To the best of our knowledge, current state-of-the-art pruning in Online-RL methods can sparsify the networks up to 50% without sacrificing performance [27]. But in this work, we show we can do better.

Similar to supervised training, Offline-RL is trained with fixed dataset. Therefore, pruning techniques that are suitable for fixed dataset can be used in offline-RL as well. In this work, we explore single-shot pruning techniques in Offline-RL algorithms. This allow us to prune the networks even before we start training an RL agent. Up until very recently batch-dataset RL in the setting of continuous control was presumed to be a hard problem. This is due to not having access to environment interactions and RL agents needing to learn from a fixed dataset. But, we can instead leverage this fixed-dataset nature of offline-RL to apply one-shot pruning techniques that are not suitable for online-RL algorithms.

Through this work we want to excite the community more about offline-RL research and pruning techniques in RL algorithms. Our contributions in this work are as follows:

- In this work, we show experimental results of pruning methods in Offline-RL algorithms where we use the following one-shot pruning methods: *SNIP* [17] and *GraSP* [29]. We explain how these single-shot pruning methods can be integrated with Offline-RL algorithms.
- We demonstrate it is possible to prune 95% of the network parameters without losing performance in continuous control tasks.

- We also show that it is possible to reduce the memory required to store these pruned networks by 4x without any elaborate compression mechanism.

2 Preliminaries

We consider learning in a Markov decision process (MDP) described by the tuple (S, A, P, R). The MDP tuple consists of states $s \in S$, actions $a \in A$, transition dynamics $P(s'|s, a)$, and reward function $r = R(s, a)$. We use s_t, a_t and $r_t = R(s_t, a_t)$ to denote the state, action and reward at timestep t, respectively. A trajectory is made up of sequence of states, action and rewards $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_T, a_T, r_T)$. For continuous control task we consider an infinite horizon, where $T = \infty$ and the goal in reinforcement learning is to learn a policy which maximizes the discounted expected return $\mathbb{E}[\sum_{t=t'}^{T} \gamma^t r_t]$ in an MDP. In offline reinforcement learning, instead of obtaining data through environment interactions, we only have access to some fixed limited dataset consisting of trajectory rollouts of arbitrary policies. This setting is harder for agent as it can not further explore the environment and collect additional feedback. Thus can fail due to overestimation of values induced by the distributional shift between the dataset and the learned policy. Offline algorithms [8, 7, 14, 13] overcome the problem through either constraining policy or the value function estimation.

3 Methods of pruning at initialization

In this section, we discuss two methods of pruning at initialization namely, SNIP and GraSP and briefly discuss their criterions of pruning. A more elaborate discussion is available at [17, 29].

3.1 Single-shot Network Pruning at initialization

The first work to tackle pruning at initialization was SNIP [17] which exploits the idea of *connection sensitivity* to prune insignificant weights. They formalize this idea in terms of removing a single weight θ_q and the effect it has on the loss as:

$$S(\theta_q) = \lim_{\epsilon \to 0} \left| \frac{\mathcal{L}(\theta_0) - \mathcal{L}(\theta_0 + \epsilon\delta_q)}{\epsilon} \right| = \left| \theta_q \frac{\partial \mathcal{L}}{\partial \theta_q} \right| \tag{1}$$

where θ_q corresponds to the q^{th} element of θ_0, and δ_q is a one-hot vector whose q_{th} element equals to θ_q. The goal of SNIP is to essentially preserve the loss of the randomly initialized network before training. Although the idea to preserve the loss value was behind some classic works in pruning [16, 12], its importance is less obvious for pruning at initialization before the training begins. The authors of GraSP [29] instead argue that it is more important to preserve the training dynamics during pruning before training rather than the loss itself, because with the first technique there is a chance to make some layers too sparse that creates a bottleneck in the neural network for signal propagation. Therefore, they argue that a pruning technique i.e. Gradient Signal Preservation, that takes into account how the presence of a connection affects the training of the whole network would be preferable.

3.2 Gradient Signal Preservation

The idea of utilizing *Gradient Signal Preservation* (GraSP) to improve upon the work of SNIP was presented in the work [29] with the algorithm the authors termed as GraSP. Pruning a network results in fewer parameters and reduced connectivity which might lead to a decrease in the flow of gradients through the network thus slowing down the optimization process. More formally, a larger norm of the gradient points to each gradient update contributing towards a greater loss reduction to the first order, as indicated by the directional derivative:

$$\Delta\mathcal{L}(\theta) = \lim_{\epsilon \to 0} \frac{\mathcal{L}(\theta + \epsilon\nabla\mathcal{L}(\theta)) - \mathcal{L}(\theta)}{\epsilon} = \nabla\mathcal{L}(\theta)^T \nabla\mathcal{L}(\theta) \tag{2}$$

The goal of GraSP is to preserve (even increase if possible) the gradient flow after pruning the network. Similar to the classic work [16] the authors cast the pruning operation as adding a perturbation δ to the initial weights. A Taylor approximation is then used to characterize the effect of removing one weight to the gradient flow through the network.

$$\begin{aligned}\mathbf{S}(\delta) =&\Delta\mathcal{L}(\theta_0+\delta)-\Delta\mathcal{L}(\theta_0)\\ =&2\delta^T\nabla^2\mathcal{L}(\theta_0)\nabla\mathcal{L}(\theta_0)+\mathcal{O}(\|\delta\|_2^2)\\ =&2\delta^T\mathbf{Hg}+\mathcal{O}(\|\delta\|_2^2)\end{aligned} \tag{3}$$

where $\mathbf{S}(\delta)$ is an approximate measure of the change to equation 2. The dependencies among the parameters of the network is captured by the Hessian matrix, which acts as a predictor of the effects of removing multiple weights.

GraSP essentially uses equation 3 to calculate the score of each weight corresponding to its effect on the reduction of gradient flow after pruning. More precisely, a negative $S(\delta)$ will correspond to a reduction of gradient flow if the associated weight is pruned, while a positive value will result in an increase of gradient flow if said weights are pruned. Therefore the larger the scores associated with the weights, the lower their importance and those weights are removed first. Therefore the vectorized scores are calculated as:

$$\mathbf{S}(-\theta) = -\theta\odot\mathbf{Hg} \tag{4}$$

GraSP then removes the *top k* fraction of the weights for a given pruning ration of k to generate a pruning mask by computing the scores associated with each weight. Thus, GraSP takes the gradient flow into account while pruning the network.

Algorithm 1 Single-shot Pruned Offline-RL Training

```
**Initialize Networks**: critic $Q_{\theta_1}$, $Q_{\theta_2}$, Actor $\pi_\phi$, VAE $V_\omega = \{V_{\omega_E}, V_{\omega_D}\}$
**Choose single shot pruning technique**: SNIP or GraSP
**Find the Pruning Weight Maps**:
$M_{\theta_1}$, $M_\phi$, $M_{\omega_E}$, $M_{\omega_D}$ = single-shot pruning $\Big(\mathcal{L}(\theta_1),\mathcal{L}(\phi),\mathcal{L}(\omega_E),\mathcal{L}(\omega_D)\Big)$
**Prune the networks:**
$\theta_1 \leftarrow \theta_1\odot M_{\theta_1}$,
$\theta_2 \leftarrow \text{copy}(\theta_1)$,
$\phi \leftarrow \phi\odot M_\phi$,
$\omega_E \leftarrow \omega_E\odot M_{\omega_E}$,
$\omega_D \leftarrow \omega_D\odot M_{\omega_D}$
**for** $t = 1$ **to** $T$ **do**
  Train Offline-RL algorithm
**end for**
```

4 Experiments

We perform our experiments on OpenAI Gym MuJoCo continuous control tasks [28, 2] on two different Offline-RL algorithms: *Batch-Constrained deep Q-learning* (BCQ) [8] and *Behavior Cloning* (BC) (implemented in [15]). Without changing anything within the RL objective, we integrate following two pruning approaches: *SNIP* [17] and *GraSP* [29]. These one-shot pruning methods find the important neural-network weights before initializing RL training loop and set the rest of the weights to zero which remains zero through-out the RL training. For example, to train a 95% sparse network, we will set the 95% weights to zero using one of these pruning techniques and will train the RL agent with the remaining 5% of the weights which the pruning methods finds to be more relevant to the RL learning objective.

We run our experiment varying different sparsity levels to understand at what extend we can reduce the model without sacrificing the performance. Since the one-shot pruning methods are independent of the Offline-RL objective, we expect these pruning methods to perform the same for other Offline-RL algorithms as well.

BCQ and BC share the same architecture, where they use separate Actor, Critic and a VAE neural network. Before starting the network training, we sample single batch of training samples (100 random (s, a, s', r) for SNIP and 200 random (s, a, s', r) samples for GraSP) to generate a pruning mask. We then prune the neural network with these mask and train the remaining weights. The complete process is detailed in Algorithm-1. Actor, Critic and VAE network have different objective functions. We individually compute these maximization/minimization objectives to find out the weights that are most relevant in optimizing these objectives.

We vary the sparsity of these networks from 10% to 95% and compare the performance of the Offline-RL algorithms. We perform our experiments on *Half-Cheetah-v2*, *Hopper-v2*, *Walker2d-v2* environments and train the offline-RL algorithms with *D4RL* [6] expert dataset. We plot the mean performance for seeds $0-4$ over 1 million gradient updates with 100% confidence interval. In figures 1 and 2 we observe that, for all the experiments, except for BCQ in Hopper task, the performance in consistent even with 95% sparse networks. This means with a fraction number of weights we still attain the performance of a large neural network. The pruning techniques can find the "sub-networks" that gives similar performance using only 5% parameters of the larger network.

This is important to note that *Dynamic Sparse TD3* (DS−TD3) [27], a very recent research work that uses dynamic sparse [20] in RL in online setting, can attain the performance up to 50% sparse network. DS−TD3 also adds additional computation within RL training to find the sparse network. On the other hand our proposed approach can leverage the fixed batch dataset offline training method and find a 95% pruned sub-architecture. Our proposed method not only shrinks the size of the network weights, with proper hardware and software optimization this will allow faster training and inference [25, 3]. This reduces the computation cost and allows RL algorithms to use in low-resource, large- data driven real-time applications.

In our initial experiments we tried varying (i) batch size and (ii) number of pruning iterations, but that does not provide any improvement over a single batch pruning loop. Since both methods perform similar to the non-pruned network, we do not conduct further experiments avoid unnecessary compute.

4.1 Performance of Offline RL algorithm with pruned network

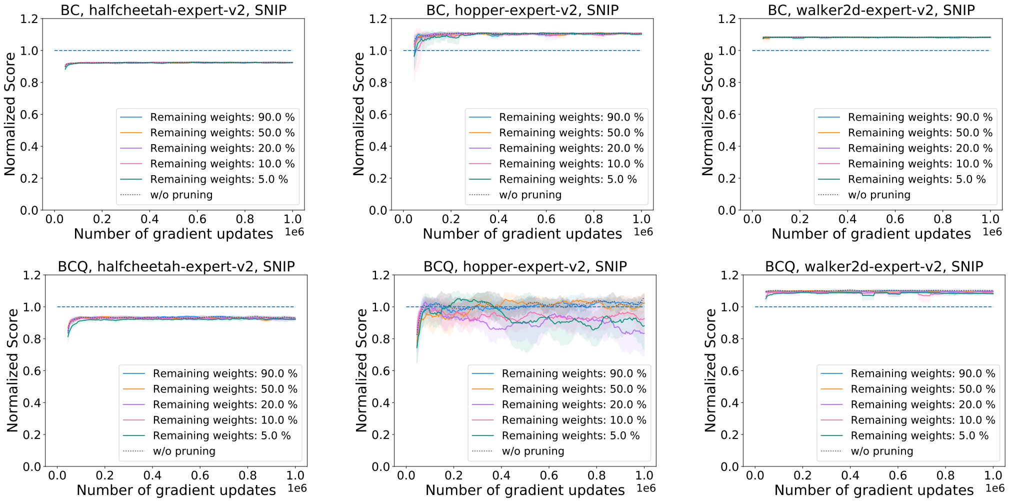


Figure 1: Performance plot of Offline-RL algorithms (BCQ, BC) varying sparsity using SNIP

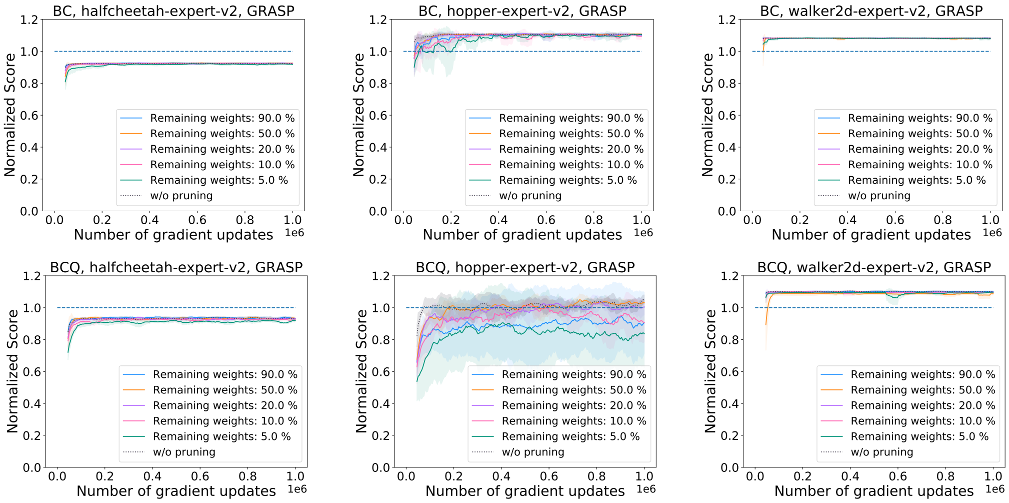
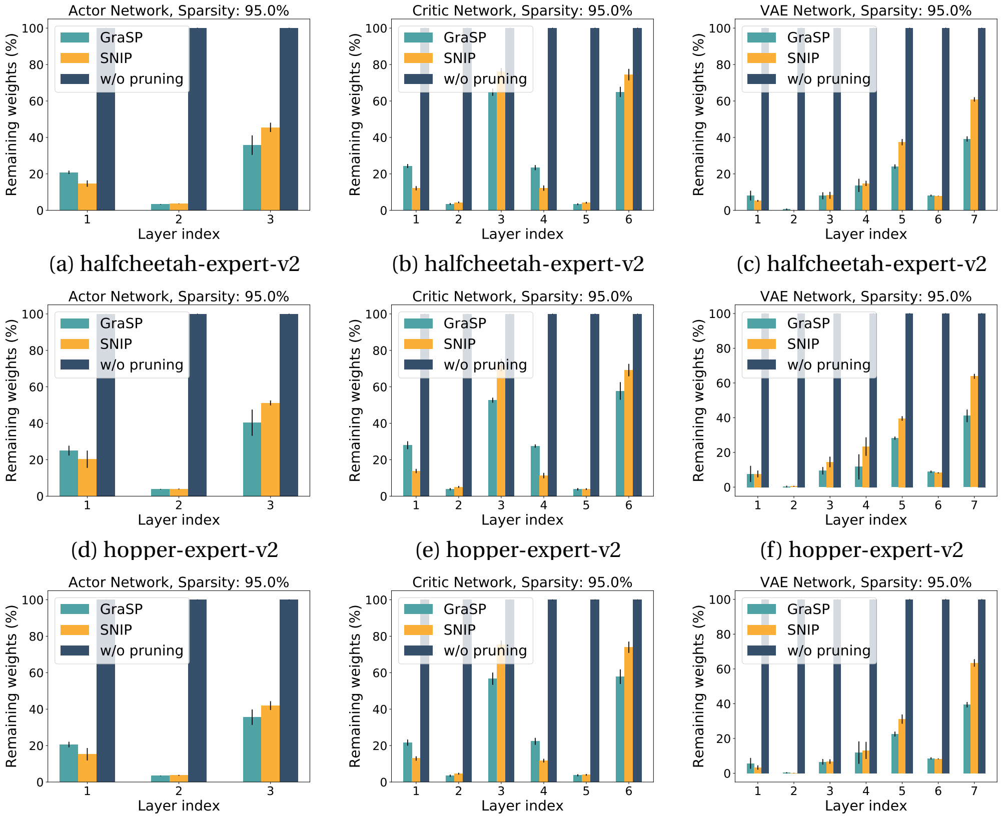


Figure 2: Performance plot of Offline-RL algorithms (BCQ, BC) varying sparsity using GraSP

4.2 Visualization of the Network After Pruning

We compare the layer wise sparsity of the pruned network to that of the regular dense network and observe the effect of layer-wise pruning. In figure 3, we compare the layer-wise remaining weights using different techniques after pruning them with 95% sparsity. We plot the mean number of layer-wise weights and it's standard-deviation for seeds 0-4. For both SNIP and GraSP we find similar pattern of pruning, where it does not prune all the layers uniformly. Both methods preserve more weights for the last layer to preserve gradient flow. But since GraSP's objective focuses on preserving the gradient flow [29] they preserve weights at the last layer than SNIP.



(a) halfcheetah-expert-v2 (b) halfcheetah-expert-v2 (c) halfcheetah-expert-v2

(d) hopper-expert-v2 (e) hopper-expert-v2 (f) hopper-expert-v2

(g) walker2d-expert-v2 (h) walker2d-expert-v2 (i) walker2d-expert-v2

Figure 3: Visualization of the remaining weights per-layer of the neural networks

4.3 Network Weights Reduction

We use "*torch.to_sparse()*" function from PyTorch [26] library to get the sparse matrix, which stores the weights and corresponding index vectors. Since sparse indexing (green line in figure 4) requires additional index vectors, it takes more memory to save regular dense network weights (blue line in figure 4). For 95% sparsity, we are able to reduce the memory size to 4*x* compared to the regular dense networks. With more sophisticated compression mechanisms to save sparse matrices, it will be possible to achieve further reduction in memory requirements. In Table 1 we compare the memory (in Megabytes) it takes to save these networks.

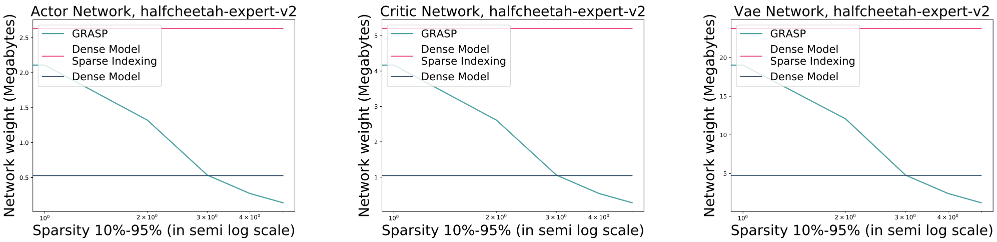


Figure 4: Comparison of memory requirement of sparse and dense models

Table 1: Memory Size of the Network Weights in Megabytes (mb)

Method	Actor	Critic	VAE
Dense Model	0.5287	1.04492	4.7621
Dense Model Sparse Indexing	2.6318	5.2035	23.7739
GraSP (95% sparse)	**0.14099**	**0.2768**	**1.2122**
SNIP (95% sparse)	**0.14297**	**0.2824**	**1.2303**

5 Future Work

We use D4RL [6] dataset for this experiment where expert data were collected from the same data distribution. In real-world application that will not be the case. And one-shot techniques does not guarantee performance under changes in the underlying distribution.

6 Conclusion

Network latency is one of the more crucial aspects of deploying a deep RL into real world application where it needs to process large dataset in real-time (i.e. self-driving car, deploying bot in games, financial data analysis etc.). This also hinders applying RL in low resource devices (i.e. embedded system, edge devices etc.). As a step towards this direction we conduct experiments on pruning techniques in offline RL algorithms. In this paper we show, how simple single-shot pruning plug-in prior to training can reduce the network parameters by 95% while maintaining performance. This sparse model saves 4x in memory without using any compression mechanism and with proper hardware integration [25, 3] it promises faster training and higher inference time.

7 Acknowledgement

Riyasat Ohib and Sergey Plis were in part supported by R01 DA040487 and RF1 MH121885 from NIH.

References

[1] Sanjeev Arora et al. “Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 322–332.

[2] Greg Brockman et al. “OpenAI Gym”. In: *CoRR* abs/1606.01540 (2016). arXiv: 1606.01540. URL: http://arxiv.org/abs/1606.01540.

[3] Sourya Dey et al. “Pre-defined sparse neural networks with hardware acceleration”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9.2 (2019), pp. 332–345.

[4] Xin Dong, Shangyu Chen, and Sinno Jialin Pan. “Learning to prune deep neural networks via layer-wise optimal brain surgeon”. In: *arXiv preprint arXiv:1705.07565* (2017).

[5] Jonathan Frankle and Michael Carbin. “The lottery ticket hypothesis: Finding sparse, trainable neural networks”. In: *arXiv preprint arXiv:1803.03635* (2018).

[6] Justin Fu et al. “D4RL: Datasets for Deep Data-Driven Reinforcement Learning”. In: *CoRR* abs/2004.07219 (2020). arXiv: 2004.07219. URL: https://arxiv.org/abs/2004.07219.

[7] Scott Fujimoto and Shixiang Shane Gu. “A Minimalist Approach to Offline Reinforcement Learning”. In: *CoRR* abs/2106.06860 (2021). arXiv: 2106.06860. URL: https://arxiv.org/abs/2106.06860.

[8] Scott Fujimoto, David Meger, and Doina Precup. “Off-Policy Deep Reinforcement Learning without Exploration”. In: *CoRR* abs/1812.02900 (2018). arXiv: 1812.02900. URL: http://arxiv.org/abs/1812.02900.

[9] Nicolas Gillis et al. “Grouped sparse projection”. In: *arXiv preprint arXiv:1912.03896* (2019).

[10] Song Han, Huizi Mao, and William J Dally. “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding”. In: *arXiv preprint arXiv:1510.00149* (2015).

[11] Song Han et al. “Learning both weights and connections for efficient neural networks”. In: *arXiv preprint arXiv:1506.02626* (2015).

[12] Babak Hassibi, David G Stork, and Gregory J Wolff. “Optimal brain surgeon and general network pruning”. In: *IEEE international conference on neural networks*. IEEE. 1993, pp. 293–299.

[13] Ilya Kostrikov et al. “Offline Reinforcement Learning with Fisher Divergence Critic Regularization”. In: *CoRR* abs/2103.08050 (2021). arXiv: 2103.08050. URL: https://arxiv.org/abs/2103.08050.

[14] Aviral Kumar et al. “Conservative Q-Learning for Offline Reinforcement Learning”. In: *CoRR* abs/2006.04779 (2020). arXiv: 2006.04779. URL: https://arxiv.org/abs/2006.04779.

[15] Aviral Kumar et al. “Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction”. In: *CoRR* abs/1906.00949 (2019). arXiv: 1906.00949. URL: http://arxiv.org/abs/1906.00949.

[16] Yann LeCun, John S Denker, and Sara A Solla. “Optimal brain damage”. In: *Advances in neural information processing systems*. 1990, pp. 598–605.

[17] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. “Snip: Single-shot network pruning based on connection sensitivity”. In: *arXiv preprint arXiv:1810.02340* (2018).

[18] Hao Li et al. “Pruning filters for efficient convnets”. In: *arXiv preprint arXiv:1608.08710* (2016).

[19] Tuanhui Li et al. “Compressing convolutional neural networks via factorized convolutional filters”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3977–3986.

[20] Junjie Liu et al. “Dynamic Sparse Training: Find Efficient Sparse Network From Scratch With Trainable Masked Layers”. In: *CoRR* abs/2005.06870 (2020). arXiv: 2005.06870. URL: https://arxiv.org/abs/2005.06870.

[21] Dor Livne and Kobi Cohen. “PoPS: Policy Pruning and Shrinking for Deep Reinforcement Learning”. In: *CoRR* abs/2001.05012 (2020). arXiv: 2001.05012. URL: https://arxiv.org/abs/2001.05012.

[22] Christos Louizos, Max Welling, and Diederik P Kingma. “Learning sparse neural networks through L_0 regularization”. In: *arXiv preprint arXiv:1712.01312* (2017).

[23] Rongrong Ma et al. "Transformed ℓ_1 regularization for learning sparse deep neural networks". In: *Neural Networks* 119 (2019), pp. 286–298.

[24] Behnam Neyshabur et al. "The role of over-parametrization in generalization of neural networks". In: *International Conference on Learning Representations*. 2018.

[25] Nvidia. "How Sparsity Adds Umph to AI Inference". In: (2020). URL: https://blogs.nvidia.com/blog/2020/05/14/sparsity-ai-inference/.

[26] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[27] Ghada Sokar et al. "Dynamic Sparse Training for Deep Reinforcement Learning". In: *CoRR* abs/2106.04217 (2021). arXiv: 2106.04217. URL: https://arxiv.org/abs/2106.04217.

[28] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.

[29] Chaoqi Wang, Guodong Zhang, and Roger Grosse. "Picking winning tickets before training by preserving gradient flow". In: *arXiv preprint arXiv:2002.07376* (2020).

[30] Huanrui Yang, Wei Wen, and Hai Li. "Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures". In: *arXiv preprint arXiv:1908.09979* (2019).

[31] Jihun Yun et al. "Trimming the ℓ_1 Regularizer: Statistical Analysis, Optimization, and Applications to Deep Learning". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7242–7251.

[32] Chiyuan Zhang et al. "Understanding deep learning (still) requires rethinking generalization". In: *Communications of the ACM* 64.3 (2021), pp. 107–115.

[33] Guodong Zhang, James Martens, and Roger Grosse. "Fast convergence of natural gradient descent for overparameterized neural networks". In: *arXiv preprint arXiv:1905.10961* (2019).

[34] Tianyun Zhang et al. "A systematic dnn weight pruning framework using alternating direction method of multipliers". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 184–199.