Learning to Prune Deep Neural Networks via Fast Reinforcement Learning

Anonymous Authors[1]

Abstract

This paper proposes PuRL - a deep reinforcement learning (RL) based algorithm for pruning neural networks. Unlike current RL based model compression approaches where feedback is given only at the end of each episode to the agent, PuRL provides rewards at every pruning step. This enables PuRL to achieve sparsity and accuracy comparable to current state-of-the-art methods, while having a much shorter training cycle. PuRL achieves more than 80% sparsity on the ResNet-50 model while retaining a Top-1 accuracy of 75.37% on the ImageNet dataset. Through our experiments we show that PuRL is also able to sparsify already efficient architectures like MobileNet-V2. In addition to performance characterisation experiments, we also provide a discussion and analysis of the various RL design choices that went into the tuning of the Markov Decision Process underlying PuRL. We also provide insights into how pruning affects the network and shifts weight distributions at the layer level. Lastly, we point out that PuRL is simple to use and can be easily adapted for various architectures.

1. Introduction

Neural network efficiency is important for specific applications, e.g., deployment on edge devices such as mobile phones, as well as more global climate considerations given the carbon footprint of large models (Strubell et al., 2019). To improve model efficiency, one successful approach is to *compress* models into smaller versions. Weight pruning has emerged as a viable solution methodology for model compression (Han et al., 2016), but pruning weights effectively remains a difficult task — the search space of pruning actions is large, and over-pruning weights (or pruning them the wrong way) leads to deficient models (Frankle et al., 2019; Deng et al., 2009). Current state-of-the-art methods rely on heuristics, for example, quantization, factorization and magnitude pruning (Gale et al., 2019) and while these hand-designed heuristics work well, they can be slow to obtain and not as good as learnt policies (He et al., 2018).

In this work, we approach the pruning problem from a decision-making perspective, and propose to automate the the weight pruning process via reinforcement learning (RL). RL provides a principled and structured framework for network pruning, yet has been under-explored. There appears to be only one existing RL-based pruning method, namely AutoML for Model Compression (AMC) (He et al., 2018). Here, we build upon AMC and contribute an improved framework: Pruning using Reinforcement Learning (PuRL).

Compared to AMC, PuRL rests on a different Markov Decision Process (MDP) for pruning. One key aspect of our model is the provision of "dense rewards" — rather than rely on the "sparse" rewards (given only at the end of each episode), we shape the reward function to provide reward feedback at each step of the pruning process. This crucial change results in a far shorter training cycle and decreases the number of training episodes required by as much as 85%. The remaining design changes are informed by ablation-style experiments; we discuss these changes in detail and elucidate the trade-offs of different MDP configurations.

Experiments on a variety of models showcase that PuRL can achieve comparable performance to state-of-the-art without having to tweak the underlying MDP or RL hyper-parameters. We sparsify a ResNet-50 model, trained on ImageNet, by 80% and achieve a final accuracy of 75.37%. We also benchmark PuRL on other datasets like CIFAR-100 and other already efficient architectures like EfficientNet-B2, and achieve out-performance with respect to baselines. We transfer the same settings as those used in ResNet-50 to prune MobileNet-V2, without modifying the MDP and RL hyper-parameters and achieve comparable performance to state-of-the-art. Thus, PuRL can be easily adapted to run on different architectures.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2. Related Work

Various techniques have been proposed for compressing neural networks (Cheng et al., 2017). These include approaches like Pruning, Quantisation, Factorisation, Attention, Knowledge Distillation and Architecture Search (Almahairi et al., 2016; Ashok et al., 2017; Iandola et al., 2016; Pham et al., 2018b).

Many pruning techniques have been developed over the years,which use first or second order derivatives (LeCun et al., 1990; Hassibi & Stork, 1993), objective function sensitivity (Molchanov et al., 2017), distance or similarity measures (Srinivas & Babu, 2015) and simple magnitude based pruning (Ström, 1997; Zhu & Gupta, 2017). (Han et al., 2015) discovered a key trick to iteratively prune and retrain the network, thereby preserving a lot of accuracy. (Gale et al., 2019) do simple magnitude pruning, but by doing gradual pruning using a very high computational budget, they achieve a state-of-the-art accuracy-sparsity trade-off.

Quantisation techniques which restrict the bitwidth of parameters (Rastegari et al., 2016; Courbariaux et al., 2016) and tensor factorisation and decomposition which aim to break large kernels into smaller components (Mathieu et al., 2013; Gong et al., 2014; Lebedev et al., 2014; Masana et al., 2017) are also popular methods, however, they need to be optimised for specific architectures. In another approach, Attention networks (Almahairi et al., 2016) have two separate networks to focus on only a small patch of the input image. Training smaller student networks in a process called knowledge distillation (Ashok et al., 2017) has also proved effective, although it can require a large training budget. Lastly, architecture search techniques like new kernel designs (Iandola et al., 2016) or whole architecture designs (Pham et al., 2018a; Tan et al., 2019) have also become popular. However, in this case the large search space size requires huge computational resources to do search.

Pruning comes out as a generalist approach not having restrictions in terms of the tasks it is applicable to. It has also shown strong results in terms of the overall compression achieved. Hence, we adopt pruning as our compression approach. However, pruning too has a large search space size and hence, traditionally, human expertise has been relied upon to do pruning. But with the advent of new search techniques like deep reinforcement learning, we can now automate the process of pruning. Lin et al. (2017) demonstrate one early approach for using RL to do pruning. They use RL to do a sub-network selection during inference. Thus, they actually don't really prune the network, but select a sub-network to do inference.

(He et al., 2018) demonstrate the first use of RL for pruning. They assign a reward to the RL agent based on approximate accuracy and overall compression rate of the model. At the end of training, they pick the best explored model and then fine-tune it to output the final model. However, they only reward the agent at the end of an episode (sparse rewards) and don't give it any reinforcement at each step in the episode. This slows down the learning process of the RL agent.

We improve upon this by creating a novel training procedure that rewards the agent at each step of the episode (dense rewards) and achieves faster convergence. Our approach is also general in nature and can be easily adapted for different architectures. We compare and report our performance with regards to AMC and other state-of-the-art pruning algorithms on the ImageNet dataset.

3. Pruning using Reinforcement Learning (PuRL)

This section details PuRL, our reinforcement learning method for network pruning. We begin with relevant background on Markov Decision Processes (MDPs) and then, formalize network pruning as an MDP; we specify the constituent elements, along with intuitions underlying their design.

3.1. Markov Decision Processes

A Markov decision process models a sequential decision-making task, and is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$. S represents the state space, and consists of states s in which the agent has the ability to choose actions $a \in \mathcal{A}$ (or some subset of available actions for each state). The transition function $\mathcal{T}(s, a, s') = p(s'|a, s)$, represents the probability of arriving at state s' having taken action a in state s and dictates the dynamics of the MDP. Each transition results in a reward, $\mathcal{R}(s, a, s')$, given to the agent.

The agent aims to solve the MDP by choosing a policy $\pi(s_t)$ that maximizes its expected reward:

$$\mathbb{E}\left[\sum_t \gamma^t \mathcal{R}(s_t, a_t = \pi(s_t), s_{t+1})\right] \tag{1}$$

where γ is a discount factor.

3.2. Pruning as a Markov Decision Problem

We model the task of pruning a neural network as a Markov Decision Problem (MDP). We formulate and construct each element of the MDP tuple i.e. $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$ to enable us to use RL for pruning. In the below subsections, we elaborate on each of the tuple elements.

3.2.1. State Representation

We represent the network state s through a tuple of features. We experiment with two different kind of representations.

The first is a simple representation scheme consisting of three features, $s = \langle l, a, p \rangle$, where l is the index of the layer being pruned, a is the current accuracy achieved on the test set (after retraining) and p corresponds to the proportion of weights pruned thus far. The attributes serve as indicators of the network state: they contain information about the proportion of network pruned till the current layer, and how the test accuracy evolves in the network as each layer is sparsified.

The second representation is a higher dimensional representation aimed at capturing more granular information on the state of the network compared to the first representation. It is formulated as, $s = \langle a_1, p_1, a_2, p_2, .., a_n, p_n \rangle$, where a_i is the test accuracy after pruning layer i and doing retraining and p_i is the sparsification percentage of layer i. s is a tuple of zeros at the start of every episode. Each tuple element is updated progressively as layer i is pruned. This helps represent information about each layer in a more detailed and persistent way. We report results from both state representations in the ablation experiments.

3.2.2. Action Space

The action space consists of actions where each action corresponds to an α value which decides the amount of pruning. We use a magnitude threshold derived from the standard deviation of the weights of a layer as our pruning criteria. We prune all weights smaller than this threshold in absolute magnitude and keep weights which are greater. The set of weights that get pruned when an $\text{Action}_i(\alpha)$ is taken for layer i are given by Equation 2.

$$\text{Weights Pruned}_i(\alpha) = \{w \mid |w| < \alpha\sigma(w_i)\} \tag{2}$$

A higher value of α increases the amount of weights pruned and vice-versa. Hence, increasing or decreasing α directly increases or decreases the amount of sparsity. Thus, we want our range to be flexible enough to handle any pruning targets. Hence, we set our range to be $\alpha \in \{0.0, 0.1, 0.2, \cdots, 2.2\}$.

To further reduce the search complexity, we also experiment with increasing the step size of our actions from 0.1 to 0.2, to sample less actions but still achieve same target prune rates. We report results in the ablation experiments. We use the same action space for all layers in the network. This is in contrast to current approaches like AMC and State of Sparsity which set a different pruning range for initial layers, in order to prune them by a lesser amount. Our approach is hence more general in this aspect.

3.2.3. Reward Function

Since, the RL agent learns the optimal sparsification policy based on the objective of maximization of total reward per episode, reward shaping helps in faster convergence (Ng et al., 1999). We formulate the total reward to be an accumulation of sub-rewards depending upon test accuracy and sparsification achieved. Since, we are motivated to jointly optimize for a given user specified target sparsity and target test accuracy, we give equal weighting to these two criteria. We normalize the achieved accuracy and sparsity with the target values. The reward function corresponding to a state s is given in Equation 3. Here, $A(s)$ and P denote the test accuracy and sparsity at state s and T_A, T_P and β correspond to the desired accuracy target, sparsity target and a fixed scaling factor set by the user.

$$\text{R(s)} = -\beta(\max(1-A(s)/T_A, 0)+\max(1-P(s)/T_P, 0)) \tag{3}$$

The first part of the equation is essentially a penalty term which penalises the agent if it achieves an accuracy lower than the target accuracy. The second part is also a penalty term and it penalises the agent if the sparsity is lower than the target. Thus, the reward design ensures that the agent jointly optimises for the desired sparsity and accuracy.

3.3. The PuRL Algorithm

We design the PuRL algorithm, to be fast and efficient, when solving the above-mentioned MDP. The first aspect of this is the choice of a good RL agent. The second and more important consideration is the rewards scheme i.e. sparse vs. dense rewards. We elaborate on each of these in the below subsections.

3.3.1. Choice of RL Agent

To solve the MDP we choose amongst various available RL algorithms. Our primary focus is on sample efficiency and accuracy, so that the network can learn the desired pruning rates in the shortest time possible. In particular we look at policy gradient algorithms and Q-learning. Policy gradient methods provide assurances of convergence, however, their sample complexity is high i.e. they require a large number of training episodes. Early experiments with Proximal Policy Optimization (PPO) (Schulman et al., 2017), a popular policy gradient method did not give meaningful results i.e. it failed to converge even after training for many steps. On the other hand, Deep Q-Network (DQN) (Mnih et al., 2013), a form of Q-learning, does a very fast exploration, however, is not very stable. It is very sample efficient due to its experience replay buffer. Each sample can be used more than once and hence, enabling faster learuning. And through careful design of our reward structure, we make DQN stable and hence, utilise it for doing pruning.

3.3.2. Making RL Fast: Dense Rewards

The pruning procedure consists of pruning the weights in a layer based on their magnitude first. The remaining weights are then retrained to get back the accuracy as close to the original accuracy. Retraining is an important aspect of this process as without it the accuracy diminishes very fast.

By setting a different α for each layer, we try to prune away the maximum redundancy specific to each layer. As mentioned in section 2, one way this has been done is to 1) assign alphas to each layer 2) prune each layer and 3) retrain the pruned network in the end. While this method works, as shown by (He et al., 2018), it may not be the fastest since it does not directly ascribe accuracy to the α of each layer. In other words, since retraining is only conducted after pruning all the layers and not after each layer (sparse rewards), the network cannot directly infer how accuracy is linked to α of each layer. This might elongate the training period since more samples are required to deduce effect of α of each layer to the network's final accuracy.

We try to mitigate this by giving rewards after pruning each layer as opposed to giving them at the end of the episode (dense rewards). We retrain the network after pruning each layer to get a test accuracy value. We do retraining by using only a small training set of 1000 images in the case of ImageNet experiments, so as not to add training overhead. We measure accuracy after each layer is pruned and pass it to the agent through the state embedding. As mentioned in section 4, this method of giving dense rewards helps achieve convergence much faster compared to giving sparse rewards.

The below steps summarize how PuRL is trained-

1. Pass the state representation of the current layer to PuRL
2. PuRL decides and outputs an α value to prune that layer
3. The layer is pruned, retraining is done and the corresponding reward is passed back to PuRL
4. Repeat the above until all layers have been pruned

A high level view of the PuRL algorithm is presented in Figure 1.

3.3.3. Full Training Pipeline

The full training pipeline consists of two stages. In the first stage, we train a 2-layer DQN that helps PuRL determine the optimal α values to apply to each layer. The second stage of the algorithm fine-tunes the pruned model for a few epochs on the full training set. Algorithm 1 gives a detailed outline of the PuRL algorithm.

Algorithm 1 describes a normal DQN procedure which learns to select the threshold α_i for each layer. At the start of each episode, the model is initialized with pre-trained weights. The agent then takes a sequence of actions,

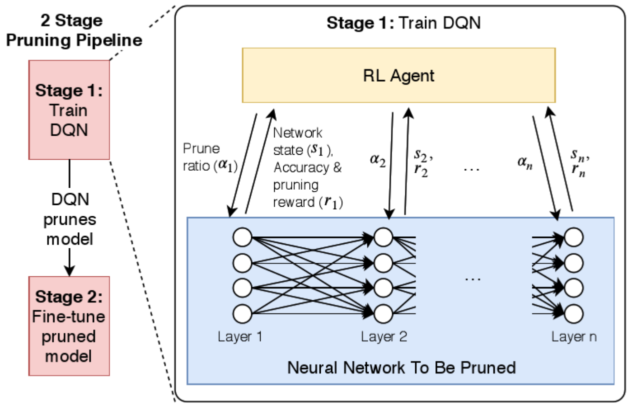

Figure 1: A high level view of the PuRL algorithm. PuRL assigns a unique compression ratio α to each layer. It then gets feedback on the test accuracy and sparsity achieved, after pruning that layer. This is in contrast to current approaches which only give feedback at the end of pruning the whole network. As a result, PuRL learns the optimal sparsity policy much faster than current approaches

while retraining the model for one epoch on training data as well as calculating the validation accuracy after each action, to effect the state transition. The reward is then calculated using Equation 3. After the model is trained for MAX_EPISODES episodes, the pre-trained model is put through actions $\text{PRUNE}_1(\alpha_1), \cdots \text{PRUNE}_T(\alpha_T)$, where α_i is the average value of α_i chosen by the agent in 5 episodes of the trained policy, and T is the total number of layers . The model is then fine-tuned with FINE_TUNE_EPOCHS epochs of training over the whole dataset to give the final pruned model.

We also search and set many RL hyper-parameters like buffer size, exploration fraction, number of episodes, etc. Settings prescribed by DQN authors don't work well for our case since they work on horizons of millions of steps while we work on a few thousand. Hence, we search and establish our own values. We set episodes to 55, exploration fraction to 0.7, buffer size to 400, discounting factor to 0.982 and target network update frequency to 15. We adapt the DQN agent provided by OpenAI Baselines (Dhariwal et al.) and train our custom gym training environment.

4. Experiments & Analysis

In this section, we describe computational experiments comparing PuRL to ablated variants, as well as baseline and state-of-the-art methods. Our primary goal was to clarify the effect of different design choices (described in section 4.1) to the pruning performance. Secondly, we demonstrate that PuRL achieves comparable results to state-of-the-art while using a 85% shorter RL training cycle by testing it

Algorithm 1 The PuRL Algorithm

```
Initialize DQN randomly
episodes ← 0, steps ← 0
while episodes ≤ MAX_EPISODES do
    model ← load_pre_trained_model()
    $t \leftarrow 0$
    Initialize $s_0$ with a 0 vector
    $a_0 \leftarrow \epsilon_t$-greedy$(Q(s_0, \cdot))$
    while $s_t$ is not terminal do
        Take $a_t$; observe $r_t$ and $s_{t+1}$.
        Add $(s_t, a_t, r_t, s_{t+1})$ to the experience replay buffer
        $a_{t+1} \leftarrow \epsilon_t$-greedy$(Q(s_{t+1}, \cdot))$
        if steps (mod train_frequency) = 0 then
            Train the Main DQN with a batch taken from the experience replay buffer
        if steps (mod update_frequency) = 0 then
            Update Target DQN parameters
        steps ← steps + 1, $t \leftarrow t + 1$
    episodes ← episodes + 1
model ← load_pre_trained_model()
Execute the action sequence $a_0, a_1, \cdots a_{T-1}$, where $a_i = (\sum_{\text{Trained Policy Episode}=1}^{\text{Trained Policy Episode}=5} a_t)/5$
for $i \leftarrow 0; i \leq$ FINE_TUNE_EPOCHS; $i \leftarrow i + 1$ do
    Fine-tune model with an epoch of ImageNet
return model
```

on CIFAR-100 and ImageNet datasets and different architectures like ResNet-50, MobileNet-V2 and WideResNet-28-10 (refer to section 4.2 and 4.3). Lastly, we showcase the generalization ability of PuRL by using the exact same settings to prune all architectures on ImageNet. We use the exact same MDP and the exact same RL hyper-parameters to achieve the results mentioned above.

4.1. Understanding the RL Design Space

We first conduct a series of ablation experiments to understand what components of the RL design space help make a good RL agent. The key elements of our design space are the state space, action space and the reward function. Please refer to section 3 for details about each. We start from a baseline set of choices and then change one choice at a time to see its effect with respect to the baseline. The choices that give superior result over the baseline are then eventually used. We experiment on the ResNet-50 architecture trained on the CIFAR-10 dataset. We set a target sparsity of 60% and target accuracy of 95% for our agent (via the reward function), in all experiments.

4.1.1. Are Dense Rewards better than Sparse Rewards?

We compare sparse rewards i.e. rewards given to the agent only at the end of the episode and dense rewards i.e. rewards at each step of the episode, and try to answer which is better. Referring to Table 1, we compare sparse rewards (row 1) to dense rewards (row 2). Our dense rewards approach outperforms the sparse rewards by a huge margin, 4% on sparsity and 24% on accuracy. Dense rewards help the agent learn much faster by guiding the agent at each step instead of only at the end of the episode. We then use dense rewards as our baseline to conduct all further ablations.

4.1.2. Is Absolute Magnitude pruning better than Standard Deviation based pruning?

Referring to the Magnitude Target based ablation (row 3), we compare absolute magnitude based pruning to standard deviation based pruning (Section 3). For absolute magnitude, we set a sparsity target for a layer and then remove all small weights until we hit the desired sparsity level. As we observe, both the sparsity and accuracy results are lower in this case as compared to our baseline experiment (dense rewards). Thus, standard deviation based pruning is better.

4.1.3. Does Reward Shaping help?

In the experiments using Reward 2 and Reward 3, we investigate if reward shaping can help the agent achieve higher accuracy. For Reward 2, we allow the agent to receive positive rewards if it surpasses the given accuracy target (Equation 4). This is in contrast to the baseline reward function in which there is a cap on the maximum reward that the agent can achieve i.e. zero.

$$\mathrm{R}_2 = -S((A/T_A - 1) + \max(1 - PP/T_{PP}, 0)) \quad (4)$$

For Reward 3, we build on Reward 2 and give a cubic reward to the agent. The agent now sees cubic growth in positive reinforcement as it approaches and surpasses the accuracy target (Equation 5). Hence, by taking the same step size towards accuracy improvement as compared to Reward 2, the agent now gets rewarded more for it.

$$\mathrm{R}_3 = -S(((A/T_A)^3 - 1) + \max(1 - PP/T_{PP}, 0)) \quad (5)$$

The performance of both these functions is close to the baseline (dense rewards), but the baseline still outperforms them. The added complexity of these functions might require the agent to sample more steps to learn them well.

Experiment	State Space	Action Size	Reward Space: Prune Penalty	Reward Space: Accuracy Penalty	Reward Space: Accuracy Upside	Reward Space: Exponential Upside	Accuracy	Sparsity
Sparse Rewards (Benchmark)	3	0.1	✓	✓			68.0% ± 13.9%	66.1% ± 12.7%
Dense Rewards (Baseline)	3	0.1	✓	✓			91.8% ± 2.0%	70.1% ± 2.3%
Baseline + Magnitude Target	3	0.1	✓	✓			86.6% ± 3.9%	60.3% ± 0.8%
Baseline + Reward 2	3	0.1	✓	✓	✓		91.4% ± 0.7%	68.3% ± 2.2%
Baseline + Reward 3	3	0.1	✓	✓	✓	✓	90.7% ± 0.2%	69.4% ± 2.9%
Baseline + Action 2	3	0.2	✓	✓			92.2% ± 0.5%	70.6% ± 0.6%
Baseline + State 2	108	0.1	✓	✓			78.9% ± 8.6%	72.2% ± 4.5%

Table 1: Ablation results on perturbing State, Action and Reward spaces for the PuRL algorithm on the CIFAR-10 dataset. Error denotes standard error as measured on 3 trials. Dense rewards outperform sparse rewards by a huge margin on accuracy (rows 1 & 2). Stepping the action space by 0.2 (row 6) leads to a Pareto dominant solution over the baseline (row 2)

Experiment	State Space	Action Size	Reward Space: Prune Penalty	Reward Space: Accuracy Penalty	Reward Space: Accuracy Upside	Reward Space: Exponential Upside	Accuracy	Sparsity
Low Dimensional State	3	0.2	✓	✓			47.6 ± 2.3	87.4 ± 1.3
Higher Dimensional State	108	0.2	✓	✓			51.0 ± 0.3	82.0 ± 2.0

Table 2: Follow-up experiment on perturbing the State space on the ImageNet dataset. Error denotes standard error as measured on 3 trials. The higher dimensional state space (108 dimensions) performs better than the simple low dimensional state space (3 dimensions)

Hence, given a tight training budget, the baseline reward function performs well.

4.1.4. Are fewer Actions better?

In the experiment using Action 2 (row 6), we modify the action space to cover the same breadth of actions but have lesser number of actions. So the range remains the same but the step size between the actions increases. So instead of the actions being (0.0, 0.1, .. , 2.2), we now have (0.0, 0.2, .. , 2.2). We see that this experiment Pareto dominates the baseline i.e. it exceeds the baseline in both sparsity and accuracy. This is likely because with less number of actions to try, the agent is able to sample each action more and gain better knowledge of each action vis-a-vis the resultant performance metrics. Hence, it picks out better actions i.e. learns a better pruning policy given a particular layer in the network. Since, this experiment Pareto dominates the baseline, we use this feature in the final configuration of our RL agent.

4.1.5. Is more information better for the agent?

In the last experiment with State 2, we vary the state space and make it 108 dimensional instead of 3 dimensional. The idea here is to give the agent more information on the state representation i.e. more details on the state of the network that is being pruned. We assign 2 dimensions per layer and since there are 54 layers in ResNet-50, we have 108 dimensions overall. One of these two dimensions is the percentage pruning done in a layer and the other is the post accuracy received after pruning that layer and doing retraining. Initially, all the dimensions are set to zero. Once, a particular layer is pruned, the dimensions corresponding to that layer are updated (See Section 3 for details).

We see that in this experiment, the agent achieves less accuracy than the baseline however, prunes more than it. Hence, none of the experiments Pareto dominate each other and its inconclusive to determine which one is better. To get more evidence on this, we carry out a further ablation on the ImageNet dataset as well. Referring to Table 2, we see that the 108 dimensional state outperforms the 3 dimensional state. Hence, more information is indeed better and we use this feature in the final configuration as well.

4.2. Scaling PuRL to CIFAR100

We first experiment with PuRL on the WideResNet-28-10 architecture (Zagoruyko & Komodakis, 2016) on the CIFAR-100 (Krizhevsky et al.) dataset. We compare it to a uniform pruning baseline where every layer is pruned by the same amount to achieve a target sparsity of 93.5%. PuRL outperforms the baseline in Table 3 on both the sparsity and final accuracy.

Method	WideResNet-28-10		
	Sparsity	Top-1 Acc. Pre-Pruning	Top-1 Acc. Post-Pruning
Baseline	93.50%	82.63%	72.42%
PuRL	93.90%	82.63%	80.63%

Table 3: Comparison of the PuRL algorithm to a uniform pruning baseline on the WideResNet-28-10 architecture on CIFAR-100 dataset. PuRL beats the baseline by a huge margin

Method	ResNet-50			
	Sparsity	Top-1 Acc. Pre Pruning	Top-1 Acc. Post Pruning	Epochs
AMC	80%	76.13%	76.11%	120
State of Sparsity	80%	76.69%	76.52%	153
PuRL	80.27%	76.13%	75.26%	90
PuRL	80.27%	76.13%	75.37%	120

Table 4: Performance of PuRL vs. state-of-the-art pruning algorithms on ImageNet. PuRL achieves comparable results to state-of-the-art

4.3. Generalization across Architectures on ImageNet

To evaluate the performance of our agent on large scale tasks we experiment with the ImageNet dataset. We prune a pretrained ResNet-50 model using an iterative pruning scheme as mentioned in (Han et al., 2015) to preserve accuracy by providing gradual pruning targets for the network. We prune in three steps achieving 36.55%, 75.73% and 80.27% sparsity, respectively. We do fine-tuning for 30 epochs after each step, bringing the total fine-tuning epochs to 90. We also report results on another variant, where we fine-tune for another 30 epochs, bringing the total epochs to 120, to see whether accuracy is still increasing or not. We compare our performance with two state-of-the-art pruning algorithms AMC: AutoML for Model Compression (He et al., 2018) and State of Sparsity (Gale et al., 2019). We prune more than 80% and achieve comparable accuracy to state-of-the-art methods (see Table 4 for full results).

Furthermore, PuRL finishes each RL training cycle in just 55 episodes, compared to 400 episodes required by AMC, due to the dense reward training procedure. We also conduct experiments on other state-of-the-art efficient architectures on ImageNet to see whether our pruning algorithm can make these architectures even more sparse. We experiment on MobileNet-V2 (Sandler et al., 2018) and EfficientNet-B2 (Tan & Le, 2019). Referring to Table 5, PuRL achieves more than 1.5x sparsity compared to AMC without much loss in accuracy.

Method	MobileNet-V2			
	Sparsity	Flops reduction	Top-1 Acc. Pre Pruning	Top-1 Acc. Post Pruning
AMC	Not reported	30%	71.8%	70.8%
PuRL	43.3%	47.9%	71.9%	69.8%

Table 5: Comparison of PuRL to AMC for the MobileNet-V2 architecture.

Method	EfficientNet-B2		
	Sparsity	Top-1 Acc. Pre Pruning	Top-1 Acc. Post Pruning
Baseline	59.0%	79.8%	68.9%
PuRL	59.5%	79.8%	74.5%

Table 6: Comparison of PuRL to uniform pruning baseline on the state-of-the-art EfficientNet-B2 architecture on the ImageNet dataset. PuRL outperforms the baseline on both the sparsity and accuracy

At the same time, PuRL achieves this performance on MobileNet-V2 without any changes in the underlying hyper-parameters compared to ResNet-50. Thus, PuRL can be easily used across architectures without the requirement of modifying the underlying MDP. For EfficientNet-B2, Table 6, we compare PuRL to a uniform pruning baseline. PuRL outperforms the baseline on both sparsity and final accuracy, achieving an accuracy improvement of more than 5%. In this case as well, we set the exact same hyper-parameters and MDP setting as that of ResNet-50 and MobileNet-V2. However, since Efficient-B2 is very deep, having 116 layers compared to 54 in ResNet-50, we do early-stopping of the RL episode, to make the training even faster. We stop the episode if the test accuracy drops to less than 0.1% and move on to the next episode.

4.4. Weights Analysis - To Remove or Not To Remove

We conduct an analysis to understand which weights are important for the network to retain accuracy and which are redundant and can be removed. We do this by assessing how pruning changes the weight distributions at the layer level. More efficient pruning schemes can be designed by keeping the below insights in mind.

In terms of the comparison, we compare weight distributions across three stages of the pruning pipeline. We consider the ResNet-50 model trained on the ImageNet dataset. As can be seen in Figure 2, the first stage is the original trained model without any pruning. The second stage is after pruning 74.9% of the connections but without doing any retraining. The third stage is the last stage where the pruned

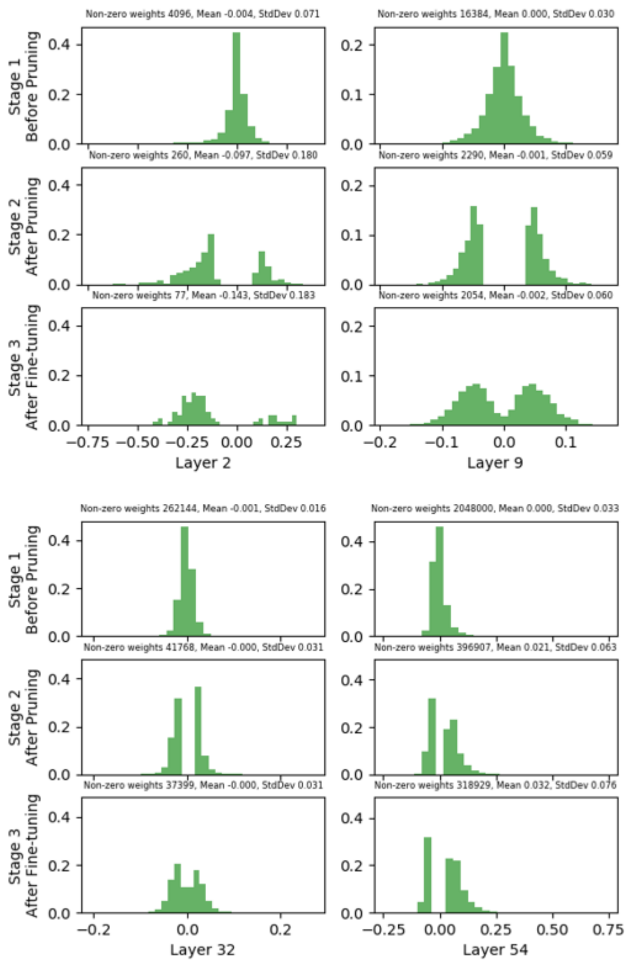

Figure 2: Analysis of weight distributions in the network before pruning, after pruning and after fine-tuning the network.

model has been retrained to gain back accuracy. Selected layers from the network are shown in the figures.

The first behaviour we observe is that of mean shifting across the weights. The starting mean value of weights gets shifted as the three stages progress. As can be seen in Layer 2, mean changes from -0.004 before pruning starts to -0.143 after fine-tuning finishes. The reason for this is that on elimination of zero-weights, the mean gets biased towards the sign which has more number of weights. So in Layer 2, since majority of the weights are negative, the mean becomes negatively biased after fine-tuning. This property is not a generic property and is only observed for certain layers. Secondly, a more generic property that is observed across many layers is that the distribution changes from unimodal to bimodal. In no layer, did the distribution change back to unimodal, even after fine-tuning. Layer 9 demonstrates this well. This means that it is safe to remove a bulk of the weights centred around zero. Thirdly, even though the distribution never becomes unimodal again, in many layers the weights are seen to move back closer to zero after fine-tuning. As can be seen from Layer 32, the gap created by pruning in the weight distribution in stage two is repopulated to some extent in stage three. This behaviour is observed across many layers albeit to different extents. This phenomenon points to the fact that while bulk of the weights near zero can be removed, a small number of values still have a role to play in improving accuracy.

As such, this is a consideration for future work, whereby more nuanced pruning and quantisation schemes can be conceptualised that have sensitivity to near zero values. This means that simple magnitude based pruning techniques might not be the best, since they remove all weights near zero. A ’softer’ pruning approach which does not remove all near zero weights might be better suited to preserving accuracy.

5. Conclusion

In this paper, we present PuRL - a fully autonomous RL algorithm for doing large scale compression of neural networks. By improving the rewards structure compared to current approaches, we shorten the training cycle of the RL agent from 400 to 55 episodes. We further do a detailed set of ablation experiments to determine the impact of each MDP component to the final sparsity and accuracy achieved by the agent and arrive on the MDP configuration that achieves the best pruning results. We achieve results comparable to current state-of-the-art pruning algorithms on the ImageNet dataset, sparsifying a ResNet-50 model by more than 80% and achieving a Top-1 accuracy of 75.37%. We also benchmark PuRL on other architectures like WideResNet-28-10 including already efficient architectures like MobileNet-V2 and EfficientNet-B2. Lastly, our algorithm is simple to adapt to different neural network architectures and can be used for pruning without a search for each MDP component.

References

Almahairi, A., Ballas, N., Cooijmans, T., Zheng, Y., Larochelle, H., and Courville, A. Dynamic capacity networks. In *International Conference on Machine Learning*, pp. 2549–2558, 2016.

Ashok, A., Rhinehart, N., Beainy, F., and Kitani, K. M. N2n learning: Network to network compression via policy gradient reinforcement learning, 2017.

Cheng, Y., Wang, D., Zhou, P., and Zhang, T. A survey of model compression and acceleration for deep neural networks, 2017.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1, 2016.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. Openai baselines (2017). *URL https://github.com/opfenai/baselines*.

Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. Stabilizing the lottery ticket hypothesis, 2019.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Gong, Y., Liu, L., Yang, M., and Bourdev, L. Compressing deep convolutional networks using vector quantization, 2014.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *International Conference on Learning Representations*, 2016.

Hassibi, B. and Stork, D. G. Second order derivatives for network pruning: Optimal brain surgeon. In Hanson, S. J., Cowan, J. D., and Giles, C. L. (eds.), *Advances in Neural Information Processing Systems 5*, pp. 164–171. Morgan-Kaufmann, 1993.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–800, 2018.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size, 2016.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-100 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., and Lempitsky, V. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *International Conference on Learning Representations*, 2014.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In Touretzky, D. S. (ed.), *Advances in Neural Information Processing Systems 2*, pp. 598–605. Morgan-Kaufmann, 1990. URL `http://papers.nips.cc/paper/250-optimal-brain-damage.pdf`.

Lin, J., Rao, Y., Lu, J., and Zhou, J. Runtime neural pruning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 2181–2191. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/6813-runtime-neural-pruning.pdf`.

Masana, M., Weijer, J. v. d., Herranz, L., Bagdanov, A. D., and Alvarez, J. M. Domain-adaptive deep network compression. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. doi: 10.1109/iccv.2017.460. URL `http://dx.doi.org/10.1109/ICCV.2017.460`.

Mathieu, M., Henaff, M., and LeCun, Y. Fast training of convolutional networks through ffts, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *NIPS Deep Learning Workshop*, 2013.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. *International Conference on Learning Representations*, 2017.

Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pp. 278–287, 1999.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameters sharing. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4095–4104, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018a. PMLR. URL `http://proceedings.mlr.press/v80/pham18a.html`.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing, 2018b.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. *Lecture Notes in Computer Science*, pp. 525–542, 2016. ISSN 1611-3349. doi: 10.1007/978-3-319-46493-0_32. URL `http://dx.doi.org/10.1007/978-3-319-46493-0_32`.

Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017.

Srinivas, S. and Babu, R. V. Data-free parameter pruning for deep neural networks. *Procedings of the British Machine Vision Conference 2015*, 2015. doi: 10.5244/c.29.31. URL http://dx.doi.org/10.5244/C.29.31.

Strubell, E., Ganesh, A., and McCallum, A. Energy and policy considerations for deep learning in nlp. *57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

Ström, N. Sparse connection and pruning in large dynamic artificial neural networks, 1997.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2019. doi: 10.1109/cvpr.2019.00293. URL http://dx.doi.org/10.1109/CVPR.2019.00293.

Zagoruyko, S. and Komodakis, N. Wide residual networks. In Richard C. Wilson, E. R. H. and Smith, W. A. P. (eds.), *Proceedings of the British Machine Vision Conference (BMVC)*, pp. 87.1–87.12. BMVA Press, September 2016. ISBN 1-901725-59-6. doi: 10.5244/C.30.87. URL https://dx.doi.org/10.5244/C.30.87.

Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression, 2017.