Spotlight: Optimizing Device Placement for Training Deep Neural Networks

Yuanxiang Gao [1 2] **Li Chen** [1] **Baochun Li** [1]

Abstract

Training deep neural networks (DNNs) requires an increasing amount of computation resources, and it becomes typical to use a mixture of GPU and CPU devices. Due to the heterogeneity of these devices, a recent challenge is how each operation in a neural network can be optimally placed on these devices, so that the training process can take the shortest amount of time possible. The current state-of-the-art solution uses reinforcement learning based on the policy gradient method, and it suffers from suboptimal training times. In this paper, we propose *Spotlight*, a new reinforcement learning algorithm based on proximal policy optimization, designed specifically for finding an optimal device placement for training DNNs. The design of our new algorithm relies upon a new model of the device placement problem: by modeling it as a Markov decision process with multiple stages, we are able to prove that Spotlight achieves a theoretical guarantee on performance improvements. We have implemented Spotlight in the CIFAR-10 benchmark and deployed it on the Google Cloud platform. Extensive experiments have demonstrated that the training time with placements recommended by Spotlight is 60.9% of that recommended by the policy gradient method.

1. Introduction

It takes an increasing amount of computation resources to train today's neural networks, and it is typical to employ a heterogeneous mixture of both CPU and GPU devices to meet such computation requirements (Sutskever et al., 2014; Bahdanau et al., 2015; He et al., 2016). With such a distributed environment of CPU and GPU devices, it is important to specify how each operation in a neural network should be matched to each of these CPU and GPU devices, referred to as the *device placement* problem. The objective is to find a match — or *placement* — of operations to devices, so that the time required to train a neural network can be minimized.

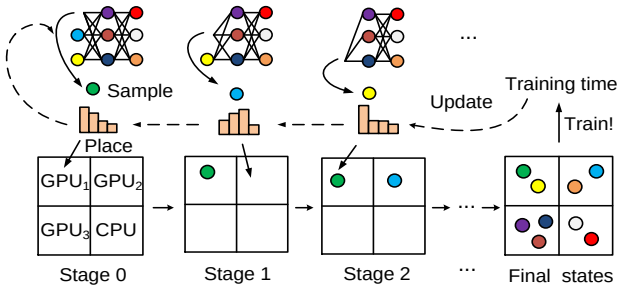


Figure 1. Using a multi-stage Markov decision process to model the device placement problem.

每个样本更新一次太慢

In recent literature, Mirhoseini *et al.* (Mirhoseini et al., 2017) proposed to solve the device placement problem using a reinforcement learning approach, based on the policy gradient method (Sutton et al., 2000). Unfortunately, the standard policy gradient method is known to be inefficient, as it performs one gradient update for each data sample (Shulman et al., 2017). With the vanilla policy gradient method, it took 27 hours over a cluster of 160 workers to find a placement that outperforms an existing heuristic (Mirhoseini et al., 2017). Such training costs are prohibitive and hardly acceptable by machine learning practitioners.

In broad strokes, we argue that the device placement problem can be solved more efficiently by using a more modern reinforcement learning approach in the literature, called *proximal policy optimization* (PPO) (Shulman et al., 2015; 2017; Heess et al., 2017). Proximal policy optimization was originally applied in the domain of continuous robot control, and was effective in teaching a simulated robot to perform complex tasks. While promising, applying proximal policy optimization to solve the device placement problem involves a non-trivial challenge that must be addressed: the device placement problem needs to be modeled as a Markov decision process (MDP).

[1]Department of Electrical and Computer Engineering, University of Toronto [2]School of Communication and Information Engineering, University of Electronic Science and Technology of China. Correspondence to: Yuanxiang Gao <yuanxiang@ece.utoronto.ca>, Li Chen <lchen@ece.utoronto.ca>, Baochun Li <bli@ece.toronto.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

In this paper, our first contribution is to model the problem as a multi-stage MDP, as illustrated in Fig. 1. At each stage, the system occupies a state about the placements of the previous operations on GPU and CPU devices. We then select the next operation and sample a probability distribution to obtain a placement recommendation on one of the available devices. After the operation is placed at a recommended device, the system transitions to the next stage with a new placement state where the previous operation has been placed. By repeating these transitions, the entire neural network is completely placed. The training time of the neural network with the final placement state is the reward of the MDP. With this reward, we update the set of probability distributions and repeat the placements again. Our objective is to guide the set of probability distributions towards desirable distributions that provide near-optimal training times.

Though proximal policy optimization provides a general framework to analyze MDPs, a highlight of this paper is a new and customized proximal policy optimization algorithm, called *Spotlight*, that is tailored for the device placement problem in particular. Since the device placement MDP has multiple stages, given a particular placement, the training time can be decomposed into the training time within each stage. We have found an approximation of such per-stage training times with a simple form, which leads to a lower bound for device placement. Based on these insights, Spotlight maximizes the lower bound of training times in each stage, and we are able to prove mathematically that it guarantees lower training times in future placements.

We have implemented Spotlight in the TensorFlow CIFAR-10 image classification benchmark, and deployed it on 10 CPU and GPU nodes in the Google Cloud platform. Spotlight takes only 9 hours on five worker machines to find even better placements than (Mirhoseini et al., 2017), and the training time using the placement found by Spotlight is 60.9% of that obtained from (Mirhoseini et al., 2017). Our detailed performance profiles have clearly shown that Spotlight discovered a non-trivial placement "trick" to improve its performance: it learns to utilize the partial connectivity of a convolutional neural network to balance the computation load on the GPUs, without incurring any communication overhead.

2. Device Placement MDP

To show a better intuition of the device placement problem in the framework of Markov Decision Processes (MDP), we present an example of placing a neural network on four devices (GPU_1, GPU_2, GPU_3 and CPU), as illustrated in Fig. 2. The operations are placed one by one, and can be characterized by the state transition graph, which is a full 4-ary tree in the MDP. Each state corresponds to a particular placement that is represented by a box with four squares (devices), each with different circles (operations).

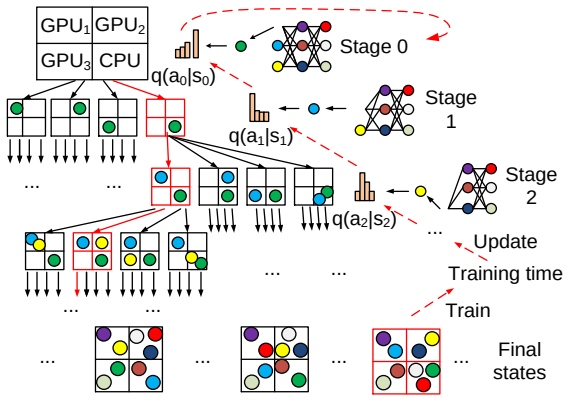


Figure 2. The state tree of a device placement MDP.

Starting from the initial state at the root of the tree, we have four options to place the first operation. Selecting a particular placement transitions the current state in Stage 0 to its next state in Stage 1. The decision on the placement is made based on the probability distribution $q(a_0|s_0)$, which indicates the preference of choosing each of the possible assignments at the current stage s_0. As shown in the figure, the fourth option of placement, *i.e.*, placing the operation on the CPU device, has the highest probability. Hence, this placement is the most probable to be selected and state s_0 transitions to s_1 in Stage 1. In a similar vein, the placement of the second operation will be selected based on $q(a_1|s_1)$, which implies a high preference for the placement on GPU_1, leading to the transition to the next state s_2 in Stage 2.

Such a state transition continues until all the operations are assigned at the final state. The training time of the neural network given by the complete placement along the state transition trajectory can be evaluated, which is used as a reward signal that is propagated backwards to adjust the set of assignment probability distributions. With the adjustment, our goal is to reduce the expected training time resulted from the placement given by the new set of probability distributions. This way, the set of assignment distributions would be gradually guided towards the distributions that lead to near-optimal performance, as evaluated by the time required for the training to complete.

3. Guaranteed Performance Improvements in Device Placement

We are now ready to elaborate how the device placement problem is solved by *Spotlight* in the framework of Markov decision processes, with a focus on analyzing the performance of our solution.

3.1. Expected Performance

In our device placement MDP, starting from the root state, operations in a network are placed one by one, each corresponding to a state transition in the MDP. The transition of a state is determined by an assignment probability, which characterizes the probabilities of selecting each possible placement. We call the entire set of transition probabilities a placement policy, defined as $\pi = \{q(a_0|s_0), ..., q(a_i|s_i), ..., q(a_{N-1}|s_{N-1})\}$, where a_i is a set of all the possible assignments, s_i is the current state, and N is the total number of stages.

The performance of a complete placement is evaluated by the resulted training time of the neural network, denoted by a random variable R. Mapped into the MDP framework, the training time performance determines the reward obtained in the final state. We use $r(s_n)$ to denote the reward function for state s_n in an MDP and define $r(s_n)$ as follows,

$$r(s_n) = \begin{cases} 0, & n < N \\ \overline{R} - R, & n = N, \end{cases} \tag{1}$$

where $\overline{R}$ denotes the average training time of all the previous trials. It serves as a baseline for evaluating the quality of a complete placement. Intuitively, a complete placement with lower training time is associated with a higher reward.

Given a placement policy π, each final state has a particular probability to be reached. Hence, the performance of the policy, denoted as $\eta(\pi)$, is best represented as the expected value of its final state reward:

$$\eta(\pi) = E_{\{a_0, a_1, ..., a_{N-1}\} \sim \pi}[r(s_N)]. \tag{2}$$

The expectation is defined as,

$$\eta(\pi) = \sum_{a_i} [\prod_{i=0}^{N-1} q(a_i|s_i) r(s_N)], \tag{3}$$

where $\prod_{i=0}^{N-1} q(a_i|s_i)$ represents the probability of traversing a particular trajectory $\{a_0, a_1, ..., a_{N-1}\}$ along the MDP state tree to the final stage N. The sum in Eq. (3) is taken over all possible trajectories.

3.2. Equivalent Expressions of Performance

For performance analysis, we introduce the state-action value function of a placement policy π as follows:

$$Q_\pi(s_n, a_n) = E_{\{a_{n+1}, ..., a_{N-1}\} \sim \pi}[r(s_N)], \tag{4}$$

which represents the expected reward value obtained at the final state s_N, starting from a particular state s_n and assignment a_n, following the set of assignments $\{a_{n+1}, ..., a_{N-1}\}$ generated from π. Similarly, the state-action value function is defined as,

$$Q_\pi(s_n, a_n) = \sum_{a_i} [\prod_{i=n+1}^{N-1} q(a_i|s_i) r(s_N)]. \tag{5}$$

The performance of π can be naturally represented with such state-action values, which we refer to as Q-values for convenience. The expression of performance in Eq. (3) can be written as,

$$\eta(\pi) = \sum_{a_j} \prod_{j=0}^{n} q(a_j|s_j) \sum_{a_i} [\prod_{i=n+1}^{N-1} q(a_i|s_i) r(s_N)]. \tag{6}$$

Plugging Eq. (5) into Eq. (6), we find that the performance and the Q-values are connected by the following equation,

$$\eta(\pi) = \sum_{a_j} [\prod_{j=0}^{n} q(a_j|s_j) Q_\pi(s_n, a_n)]. \tag{7}$$

When a reward is obtained, the current policy π can be updated, resulting in a new policy π', which is a set of new distributions $\{q'(a_0|s_0), q'(a_1|s_1), ..., q'(a_{N-1}|s_{N-1})\}$. Following Eq. (7), the expected performance of the new policy can be expressed as:

$$\eta(\pi') = E_{\{a_0, ..., a_{n-1}\} \sim \pi'} [\sum_{a_n} q'(a_n|s_n) Q_{\pi'}(s_n, a_n)]. \tag{8}$$

Compared with Eq. (7), the summation of the joint probabilities are expressed as an expectation in Eq. (8).

According to this relation, the expected performance can be expressed with the expected Q-value at any stage n, which allows us to analyze the performance, to be elaborated in the next subsection.

3.3. Approximations of Performance

In device placement, our objective is to minimize the training time of the neural network to be placed, which corresponds to maximizing the expected performance (Eq. (8)) in the MDP. However, it is challenging to maximize Eq. (8) directly, since it is expressed with the new policy π', which is the decision to be made for an update.

To address this challenge, the proximal policy optimization method (Shulman et al., 2015; Shulman, 2016) is designed to optimize the performance by maximizing its lower bound expressed by its approximation, with a desirable performance improvement guarantee. Following this pattern, we derive such an approximation for device placement. We replace $Q_{\pi'}(s_n, a_n)$ in Eq. (8) with $Q_\pi(s_n, a_n)$ and replace the new policy π' in the expectation with the old policy, which yields

$$F_\pi(\pi') = E_{\{a_0, ..., a_{n-1}\} \sim \pi} [\sum_{a_n} q'(a_n|s_n) Q_\pi(s_n, a_n)]. \tag{9}$$

Such an approximation of performance in Eq. (9) can be directly optimized. A natural question is: what is the "distance" between the approximated performance in Eq. (9) and the true performance in Eq. (8)? We will answer this question in the next subsection by deriving the performance lower bound.

3.4. Performance Lower Bounds

We first introduce the notations that are necessary or convenient for analyzing the performance bound. Let $a_{[0:n-1]}$ denote a partial trajectory $\{a_0, ..., a_{n-1}\}$. $Q^{\pi,\pi'}(n)$ is short for the term $\sum_{a_n} q'(a_n|s_n)Q_\pi(s_n, a_n)$ in Eq. (9). Similarly, $Q^{\pi',\pi'}(n)$ is short for $\sum_{a_n} q'(a_n|s_n)Q_{\pi'}(s_n, a_n)$. We use $Q_{[\pi'-\pi]}(s_n, a_n)$ to represent $Q^{\pi',\pi'}(n) - Q^{\pi,\pi'}(n)$. Let $\epsilon_1 = \max_{s_n,a_n} |Q_{[\pi'-\pi]}(s_n, a_n)|$ and $\epsilon_2 = \max_{s_n} |Q^{\pi,\pi'}(n)|$. We further denote $D_{KL}^{\max}(\pi||\pi')$ as the maximal divergence between the old policy and the new policy over all the states, which is expressed as:

$$D_{KL}^{\max}(\pi||\pi') = \max_{s_n} D_{KL}(q(a_n|s_n)||q'(a_n|s_n)). \quad (10)$$

In Eq. (10), $D_{KL}(q(a_n|s_n)||q'(a_n|s_n))$ is the Kullback-Leibler (KL) divergence (Bishop, 2011) between the two probability distributions. Intuitively, $D_{KL}^{\max}(\pi||\pi')$ represents the distance between the two policies.

Given the notations above, we have the following theorem on a performance lower bound, the proof of which is presented in Appendix A.

Theorem 1. *The expected performance of a new policy is bounded from below as follows:*

$$\eta(\pi') \geq F_\pi(\pi') - \epsilon_1 - 2\epsilon_2 n D_{KL}^{\max}(\pi||\pi'). \quad (11)$$

3.5. Theorem on Performance Improvement Guarantees

As aforementioned, directly maximizing the performance $\eta(\pi')$ when updating a policy is challenging. With Theorem 1, we can address this issue by maximizing the lower bound of the performance instead, which results in the following optimization problem:

$$\max_{\pi'} F_\pi(\pi') - \epsilon_1 - 2\epsilon_2 n D_{KL}^{\max}(\pi||\pi'). \quad (12)$$

In Eq. (12), π and π' represent the policies before and after the update. For convenience, we use $G_\pi(\pi')$ to denote the objective function in Eq. (12). If we iteratively maximize $G_\pi(\pi')$ in each policy update, the performance achieved with the updated policy is guaranteed to improve. Such a guarantee on performance improvements is rigorously presented in the following theorem, with its proof presented in Appendix B.

Theorem 2. *For the sequence of policies $\pi_0, \pi_1, \ldots, \pi_i, \pi_{i+1}$ generated by iteratively maximizing the performance lower bound $G_{\pi_j}(\pi_{j+1})$, $0 \leq j \leq i$, the following performance improvement guarantee holds:*

$$\eta(\pi_0) \leq \eta(\pi_1) \leq \eta(\pi_2) \leq ... \leq \eta(\pi_i) \leq \eta(\pi_{i+1}).$$

The general idea of its proof follows two steps. First, we prove that the performance lower bound $G_{\pi_j}(\pi_{j+1})$ is a minorization function (Hunter & Lange, 2004) of the true performance $\eta(\pi_{j+1})$. Then, we show that according to the Minorization-Maximization (MM) theory, iteratively maximizing the performance lower bound results in non-decreasing improvements of the true performance.

4. *Spotlight*: Iterative Performance Improvement

Guided by the theorems we have derived so far, we now present the main step in *Spotlight*: maximizing the performance lower bound in each update on the placement policy. To be specific, given the current policy π, we select the new policy π' that maximizes the objective function $G_\pi(\pi')$.

To obtain $F_\pi(\pi')$ in the objective, we conduct a series of equivalent transformations on its expression as follows:

$$\begin{aligned} F_\pi(\pi') &= E_{a_{[0:n-1]}\sim\pi}[\sum_{a_n} q(a_n|s_n)\{\frac{q'(a_n|s_n)}{q(a_n|s_n)}Q_\pi(s_n, a_n)\}] \\ &= E_{a_{[0:n]}\sim\pi}[\frac{q'(a_n|s_n)}{q(a_n|s_n)}Q_\pi(s_n, a_n)]. \end{aligned} \quad (13)$$

As Eq. (13) consists of an expectation, maximizing $G_\pi(\pi')$ becomes an unconstrained stochastic optimization problem (Spall, 2003), which is generally solved by finding an estimate of the expectation. Accordingly, we can solve problem (12) by maximizing the following estimated performance objective:

$$\max_{\pi'} \frac{1}{N} \sum_{\substack{n=0 \\ a_n\sim\pi}}^{N-1} [\frac{q'(a_n|s_n)}{q(a_n|s_n)}(\overline{R} - R) - \epsilon_1 - 2\epsilon_2 n D_{KL}^{\max}(\pi||\pi')], \quad (14)$$

where the term $\overline{R} - R$ is an estimate of $Q_\pi(s_n, a_n)$ in Eq. (13) and an average is taken over all stages.

The constants ϵ_1 and ϵ_2 in this objective are impossible to obtain in practice because they take the maximum of Q-values over the whole state space. Fortunately, as the objective penalizes the KL-divergence between π and π', they are close to each other. Accordingly, ϵ_1 is close to zero following its definition, and thus we can omit ϵ_1 in Eq. (14). The constant $2\epsilon_2 n$ in Eq. (14) is treated as a hyperparameter

β of the algorithm, with its value typically selected from 1 to 10 (Shulman et al., 2017). Finally, we derive the following practical objective,

$$\max_{\pi'} \frac{1}{N} \sum_{\substack{n=0 \\ a_n \sim \pi}}^{N-1} [\frac{q'(a_n|s_n)}{q(a_n|s_n)}(\overline{R} - R) - \beta D_{KL}(q||q')], \quad (15)$$

where the term of maximal divergence in Eq. (14) can be replaced by the divergence between the two distributions $D_{KL}(q(a_n|s_n)||q'(a_n|s_n))$ in Eq. (15) without impacting the final performance (Shulman et al., 2015).

We are now ready to present *Spotlight* in Algorithm 1, which iteratively maximizes the performance lower bound to improve performance. The policy π is initialized with uniformly random distributions, and the hyperparameter β is set as the typical value of 1 (Shulman et al., 2017). In each iteration, Spotlight first performs a downward pass from stage 0 to stage $N-1$. At each stage n, Spotlight samples the device assignment distribution $q(a_n|s_n)$ and obtains a device assignment for operation n. It collects the device assignment obtained at each stage into a vector a until a complete trajectory of device assignments is obtained. Then, Spotlight places the DNN with the newly obtained device assignments and trains the DNN to obtain its training time. If the training time is smaller than the current minimal training time of all the placements tried before, the current best placement a^* is updated as the newly obtained placement, with the current minimal training time refreshed.

Next, Spotlight performs an upward propagation along the same trajectory as in the downward pass. At each stage n, Spotlight uses the collected information about the training time and the average training time to build the performance objective in Eq. (15), denoted by G_n. Traversing upwards, it sums up the performance objective at each stage, until the root state is reached when the performance objectives in all N stages are averaged to construct the performance objective, G/N. To maximize the performance objective, Spotlight performs ten stochastic gradient ascent (SGA) steps on this objective, with π' as optimization variables. The resulted new policy π' is used to update π for the next iteration. After K iterations of such performance maximization, the best placement a^* during the learning process will be our final solution to place the DNN.

Algorithm 1 Spotlight algorithm

Input: The set of available devices: $\{d_1, d_2, ..., d_M\}$
Output: A near-optimal device placement: a^*
Initialize π as uniform distributions; $\beta = 1$; min = ∞
for iteration = 1, 2, . . ., K **do**
  a = []; G = 0
  for n = 0, 1, . . ., $N-1$ **do**
    Sample $q(a_n|s_n)$ to get $a_n \in \{d_1, d_2, ..., d_M\}$
    a.append(a_n)
  end for
  Reconfigure the device placement of the DNN in TensorFlow as $a = [a_0, a_1, ..., a_{N-1}]$
  Train the DNN for ten steps
  Record the training time R
  if $R <$ min **then**
    $a^* = a$
    min = R
  end if
  for $n = N-1, N-2, \ldots, 0$ **do**
    $G_n = \frac{q'(a_n|s_n)}{q(a_n|s_n)}(\overline{R} - R) - \beta D_{KL}(q||q')$
    $G = G + G_n$
  end for
  Maximize $\frac{G}{N}$ w.r.t. π' with SGA for ten steps
  $\pi = \pi'$
end for

5. Implementation and Evaluation

5.1. Implementation and Setup

Devices. We have conducted our experiments with 10 machines on the Google Cloud platform. The machines are equipped with one Intel Broadwell 8-core CPU and either two or four NVIDIA Tesla K80 GPUs each.

Benchmark. We have implemented Spotlight in the CIFAR-10 image classification benchmark (CNN). The architecture of CIFAR-10 is a convolutional neural network (CNN) with three blocks, including two convolutional blocks and one fully connected block. Each convolutional block consists of a convolutional layer, a pool layer and a norm layer. The fully connected block consists of two fully connected layers.

Architecture. The policy π in Spotlight is represented by a two-layer sequence-to-sequence recurrent neural network (RNN) (Mirhoseini et al., 2017) with long short term memory (LSTM) cells (Hochreiter & Schmidhuber, 1997) and a content-based attentional mechanism (Chan et al., 2015). As shown in Fig. 3, the set of operations in a deep neural network (e.g., a convolutional neural network) is first compiled into a vocabulary, which consists of the names of operations. The set of names are fed into the sequence-to-sequence RNN, which generates a set of probability distributions, representing the policy $\pi = \{q(a_0|s_0), ..., q(a_N|s_N)\}$. Starting with uniformly random distributions, the RNN is trained by Spotlight to generate distributions towards better placements that are theoretically guaranteed.

Co-location group. With a total of 926 operations in the CNN of CIFAR-10, it is difficult to use an RNN to read such a large amount of operations due to vanishing and exploding gradient issues (Mirhoseini et al., 2017). Current practice (Mirhoseini et al., 2017) relies on the default co-location group in TensorFlow to collocate operations, which easily re-

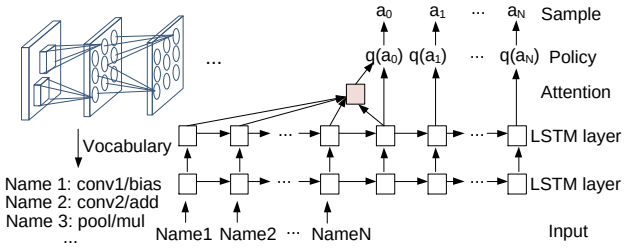


Figure 3. Using a sequence-to-sequence recurrent neural network to represent the placement policy.

sults in unrelated operations placed together, negatively impacting the balance of computation loads. We choose to perform a better co-location based on the observation that the name associated with each operation in TensorFlow, such as "`conv1/biases/add`," "`conv1/biases/mul`," or "`conv2/weight/exp`," describes the detailed function of the operation. As such, we group all the operations that share the same two-level prefix into a super operation, so that these operations are reasonably collocated, due to their close locations and similar functions. In our implementation, 926 operations in the CNN are grouped into 86 super operations.

Distributed training. Spotlight is trained on five GPU-CPU machines, with a controller holding the RNN and other workers (including the controller) holding the DNN. In each iteration of training, the RNN generates five sampled placements, each to be evaluated on a worker by training the DNN for ten steps with the placement. The training times obtained at all the workers are then sent to the controller, to be averaged to get a more accurate estimate of the training time resulting from the placement policy. Then, Spotlight updates the RNN and starts the next iteration.

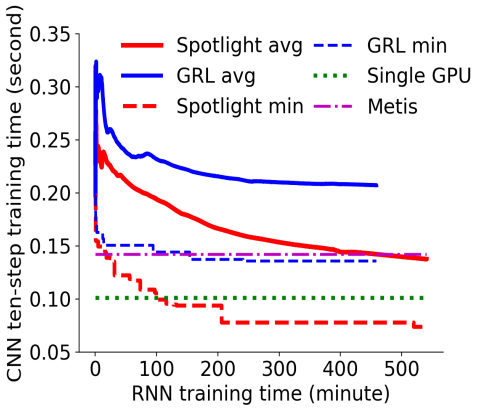
5.2. Baselines

We compare Spotlight with the following baselines:

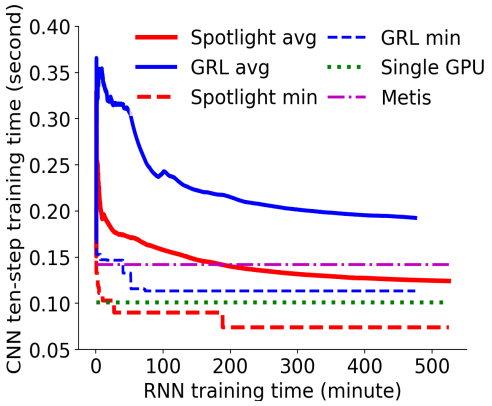
Single GPU. As the default placement of CIFAR-10 benchmark (CNN), this placement executes the whole CNN on a single GPU, except for the input operations, which are executed on a CPU.

Metis. To obtain this placement baseline, we generate a cost model of operations in the CNN, which records the input/output size, the duration and the mutual dependencies of operations. Then, we feed the cost model into the Metis graph partitioner (Karypis & Kumar, 1998) to partition the graph into three parts (2 GPU + CPU case) or five parts (4 GPU + CPU case), with each part assigned to one device.

Synchronous/Asynchronous towers. Synchronous towers (Mirhoseini et al., 2017) place an individual CNN model replica on each GPU, which independently performs a forward pass and a backward propagation to compute the gradients of its CNN replica. All the gradients computed by each GPU are transferred to the CPU to be averaged. Then, the CPU updates the parameters of the CNN and transfers them to each GPU for the next iteration. In asynchronous towers, each GPU does not need to wait for the gradients computed by other GPUs before performing an update.



(a) 4 GPUs and 1 CPU



(b) 2 GPUs and 1 CPU

Figure 4. Performance in Spotlight training.

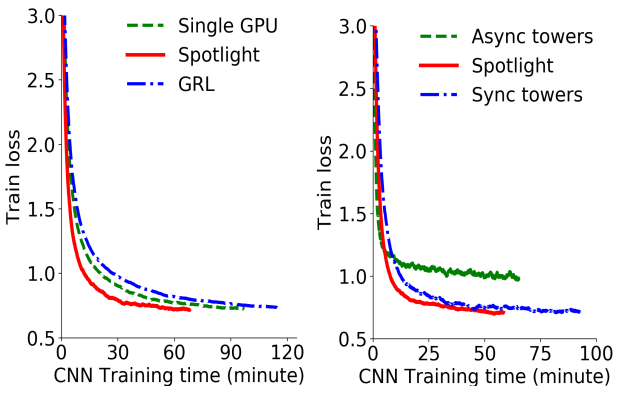
GRL. The reinforcement learning (RL) approach proposed by Google (Mirhoseini et al., 2017) is referred to as GRL for short. With the policy gradient algorithm, Google also trains a sequence-to-sequence RNN to find the device placements. The policy gradient algorithm in GRL uses the following update rule:

$$\theta' = \theta + \frac{1}{N} \sum_{a_n \sim \pi} \bigtriangledown_\theta \log q(a_n|s_n) \cdot (\overline{R} - R),$$

where θ is the parameter of the RNN.

5.3. Performance of Spotlight Training

Fig. 4(a) shows the ten-step training time of the CNN as a function of the training time used to find device placements either with Spotlight or GRL. Over the course of



(a) 2 GPUs and 1 CPU (b) 4 GPUs and 1 CPU

Figure 5. Performance in CIFAR-10 training.

reinforcement learning, 20000 sampled placements were evaluated over five distributed machines, within a period of 500 minutes. As shown in Fig. 4(a), the average ten-step training time of Spotlight decreases monotonously and rapidly throughout the training. The placement skills of Spotlight improve significantly faster than GRL. According to the learning curve, GRL suffers from occasional oscillations and slow progress. In contrast, Spotlight keeps improving the performance, as guaranteed by our theorem.

We also track the minimum training time among placements sampled by Spotlight and GRL, respectively, during the training. As obviously observed, Spotlight outperforms the baseline of single GPU placement after only 100 minutes. In contrast, GRL does not find any placement that outperforms the single GPU baseline throughout the training. Fig. 4(b) shows the learning curves of Spotlight and GRL for the environment with 2 GPUs and 1 CPU. In a similar vein, Spotlight smoothly decreases the average training time, outperforming GRL by a large margin. The highlight is that Spotlight outperforms the single GPU baseline after 20 minutes.

5.4. Performance of CIFAR-10 Training

After the RNN training, the best placement found by Spotlight or GRL is used to train the CIFAR-10 CNN for 100K steps until the model achieves an 87% accuracy. Fig. 5(a) shows the training curves under different placements for the environment with 2 GPUs and 1 CPU. As observed, the placement found by Spotlight reduces the training time compared with both of the baselines. Specifically, with the placement given by Spotlight, it takes 70 minutes for the CNN to converge, which is 70% of the time with the single GPU placement and 60.9% of the time with GRL.

For the environment with 4 GPUs and 1 CPU, the training of CNN placed by Spotlight is faster than that with data-parallel placements, as shown in the training curves in Fig. 5(b). To be specific, the training completes within 95 minutes with synchronous towers, while Spotlight results in a training time of 58 minutes, which is $\sim$ 40% better than synchronous towers. Asynchronous towers result in a shorter training completion time but a slower convergence than synchronous towers. In comparison, Spotlight leads to a shorter training time without compromising the speed of convergence.

5.5. Tricks Learned by Spotlight

To understand the rationale of the placements found by Spotlight, we compare the performance profiles of Spotlight with placements given by other heuristics. All these placements put the input pipeline on CPU, following the rule of thumb.

Fig. 6(a) shows the placements given by the single GPU baseline, where one of the GPUs is busy with executing all the operations while the other GPU is left idle. Fig. 6(b) presents the placement found by GRL, with the norm and pool layers assigned to GPU_1, reducing the computation load on GPU_0. However, the inter-connections between layers introduce communications between GPUs, which slows down the training compared with the single GPU placement. Different from GRL, Spotlight significantly reduces inter-CPU communications with the insight that the inter-connected layers in CNNs are partially connected so that they can be split into two parts, each on a GPU, without introducing communication overhead. The operations on the inter-connected layers should be split so that the inter-connected parts are placed on one GPU and those unconnected parts are distributed to two GPUs, as shown in Fig. 6(c). Learning from scratch, Spotlight discovers such an optimal way to place CNNs, which significantly outperforms other placements.

We further compare the placements given by synchronous towers and Spotlight in the environment with 4 GPUs and 1 CPU. Due to parameter updates in synchronous towers, frequent communications are incurred between CPU and GPUs, leading to a communication bottleneck on the CPU which significantly delays the training, as illustrated in Fig. 6(d). In contrast, Spotlight incurs minimal communication overheads, as shown in Fig. 6(e) (GPU_1 and GPU_3 are left idle as the deep CNN of CIFAR-10 benchmark only allows two-part splits.). Obviously, Spotlight manages to utilize the characteristic of partial connectivity of the CNN to balance computations on GPUs with negligible communication overheads.

5.6. Performance in Other Benchmarks

To demonstrate the generality of performance improvement achieved by Spotlight, we have evaluated it with two more datasets: the TensorFlow Neural Machine Translation (NMT) (Wu et al., 2016; NMT) and the TensorFlow

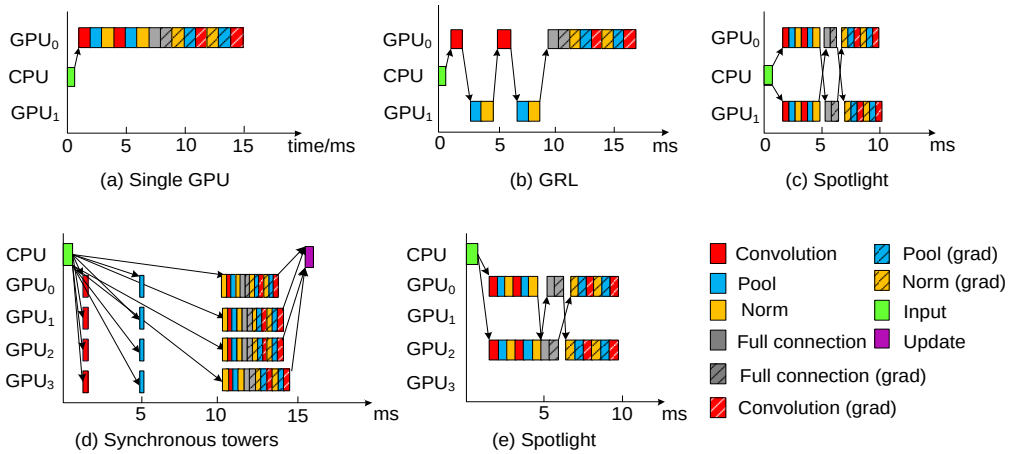


Figure 6. Single-step training time profiles: Spotlight discovers optimal placements by utilizing the partial connectivity of the CNN.

RNN language model (RNNLM) (Jozefowicz et al., 2016; RNN). NMT is an 8-layer sequence-to-sequence network, and RNNLM is a 4-layer LSTM network.

Table 1. Per-step training time (in seconds) of placements given by the baselines for the environment with 4 GPUs. Experts place each LSTM layer on one GPU.

Models	Experts	Metis	GRL	*Spotlight*
RNNLM	3.86	6.12	3.15	2.27
NMT	5.54	10.50	4.74	3.62

We compare the best placements discovered by Spotlight and GRL after about 12 hours of training on the Google Cloud, by evaluating both their single-step and end-to-end training times. As shown in Table 1, the single-step training time reduction of Spotlight over GRL is 27.9% for RNNLM and 23.6% for NMT, respectively. We further evaluate the end-to-end training time by training the NMT for 12000 iterations given the placement found by either Spotlight or GRL. With Spotlight, it takes 12.5 hours to finish, achieving a reduction of 21.9% compared with 16 hours given by GRL. Due to the per-step training time reduction, the performance improvement of Spotlight over GRL persists over a longer period of training.

6. Related Work

The first work that tackles the device placement problem for training deep neural networks is the reinforcement learning approach based on the policy gradient algorithm (Mirhoseini et al., 2017). However, the inefficiency of the policy gradient algorithm incurs a prohibitively high cost to find a placement that outperforms the existing heuristics. Moreover, Mirhoseini *et al.* did not model the device placement problem as an MDP. In contrast, we derived the important performance improvement theorem and develop the Spotlight algorithm for the device placement MDP. With its theoretically guaranteed performance improvement, Spotlight is able to significantly improve the learning efficiency.

More recently, (Mirhoseini et al., 2018) proposed a hierarchical architecture that adds a feedforward neural network to automatically classify operations into groups. It improves the learning efficiency. However, it still relies on the policy gradient algorithm to train the architecture. Orthogonal to this work, Spotlight has improved the efficiency with the design of a more advanced algorithm.

Several existing works in the literature applied the policy gradient algorithm in other problems, such as cluster scheduling (Mao et al., 2016), video streaming (Mao et al., 2017), and neural architecture design (Pham et al., 2018). Spotlight, with a customized proximal policy optimization theory, provides a better choice for the settings of these problems.

7. Conclusions

In this paper, we have proposed Spotlight based on the development of a customized proximal policy optimization theory, designed to find optimal device placements for training deep neural networks (DNN). The device placement problem is modeled as a Markov decision process (MDP), which is the first MDP model for this problem. We have derived a new performance lower bound and prove a new performance improvement theorem for device placement. Based on this theorem, we have designed the Spotlight device placement algorithm, and implemented it across the CIFAR-10, RNNLM and NMT benchmarks. Extensive experiments on the Google cloud platform have demonstrated that Spotlight discovered optimal placement tricks that outperform the best heuristics by a large margin. With a significantly lower training time, the placement found by Spotlight outperforms the state-of-the-art device placement algorithm based on policy gradient method.

References

Convolutional neural network. In *https://www.tensorflow.org/tutorials/deep_cnn*.

Neural machine translation. In *https://github.com/tensorflow/nmt*.

Recurrent neural network. In *https://www.tensorflow.org/tutorials/recurrent*.

Bahdanau, D., Kyunghyun, C., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2011.

Chan, W., Jaitly, N., Le, Q., and Vinyals, O. Listen, attend and spell. In *arXiv:1508.01211*, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Heess, N., TB, D., Sriram, S., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, A., and Riedmiller, M. Emergence of locomotion behaviours in rich environments. In *arXiv:1707.02286*, 2017.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 1997.

Hunter, D. R. and Lange, K. A tutorial on mm algorithms. *The American Statistician*, 2004.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. Exploring the limits of language modeling. In *arXiv:1602.02410*, 2016.

Karypis, G. and Kumar, V. Metis: Software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. 1998.

Mao, H., Alizadeh, M., Menache, I., and Kandula, S. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 2016.

Mao, H., Netravali, R., and Alizadeh, M. Neural adaptive video streaming with pensieve. In *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2017.

Mirhoseini, A., Pham, H., Le, Q., Steiner, B., Larsen, R., Zhou, Y., Kumar, N., Norouzi, M., Bengio, S., and Dean, J. Device placement optimization with reinforcement learning. In *International Conference on Machine Learning*, 2017.

Mirhoseini, A., Goldie, A., Pham, H., Steiner, B., Le, Q. V., and Dean, J. A hierarchical model for device placement. In *International Conference on Learning Representations*, 2018.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search with parameter sharing. In *International Conference on Machine Learning*, 2018.

Shulman, J. *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*. PhD thesis, University of California, Berkeley, 2016.

Shulman, J., Levine, S., Moritz, P., Jordan, M., and Abbeel, P. Trust region policy optimization. In *International Conference on Machine Learning*, 2015.

Shulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. In *arXiv:1707.06347*, 2017.

Spall, J. C. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Wiley, 2003.

Sutskever, I., Vinyals, O., and Le, Q. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pp. 3104–3112, 2014.

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

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., and *et al.* Google's neural machine translation system: Bridging the gap between human and machine translation. In *arXiv:1609.08144*, 2016.