Scalable Evolution Strategies Pipeline for Solving the Vehicle Routing Problem

Anonymous Author(s)
Affiliation
Address
email

1 Introduction

As a general framework for applying deep learning methods to solve a problem, Deep Reinforcement Learning (RL) has many applications. In this paper we study Deep RL as it applies to the Vehicle Routing Problem (VRP). Specifically, we focus on the capacitated variant of the VRP (CVRP), in which vehicles have a maximum carrying capacity and customers have varied demands. Currently in the literature, there are quite a few papers in which researchers have applied Deep RL to the CVRP [Bengio et al., 2018, Kool et al., 2019, Lu et al., 2020]. While the methods developed are able to produce solutions to problems fairly quickly, so far, they all use GPUs to train the models, which reduces scalability. Recently, OpenAI released a study on comparing Evolution Strategies (ES) with classic Deep RL training methods, such as Policy Gradient (PG), and found that ES uses less resources and performs similarly to state-of-the-art Deep RL training methods [Salimans et al., 2017]. The main benefit of this is that ES can be trained on CPUs in parallel, which costs less than training on a GPU. In light of this, we are motivated to replace traditional RL training methods in the research with ES for comparison.

Recent works To begin, there have been attempts to build an end-to-end model that takes the Traveling Salesman Problem (TSP) or VRP as an input and outputs a solution [Bello et al., 2016, Kool et al., 2019]. The seminal paper written by Vinyals et al. [2015] on Pointer Network largely influenced these designs. Pointer networks can be used to tackle combinatorial optimization problems using a sequence-to-sequence model in a supervised fashion, but this limits its applicability to problems outside of the training data [Bello et al., 2016]. To overcome these limitations, Bello et al. [2016] proposed training a Pointer Network model with actor-critic and produced good results on the TSP. Encouraged by this, Kool et al. [2019] proposed using an Attention based encoder-decoder model that was trained using PG with a baseline based on a deterministic rollout, rather than a value function, to reduce the variance. This Attention model was able to produce quality solutions on large VRP in a small amount of time [Kool et al., 2019]. Taking advantage of the speed of the model, Kool et al. [2019] was also able to further improve the performance of the model by performing a simple n-sized search, in which the best of n was recorded as the solution. Along with this type of sampling, Bello et al. [2016] also proposed a "Active Search" sampling method that took advantage of fast end-to-end models to search the solution space at inference time and improve the policy using the solutions found during search. Instead of using end-to-end models to produce solutions, Lu et al. [2020] propose a learning-based iterative method that starts from a random solution and trains an RL agent to use a rich set of improvement operators to improve the solution and when the agent is "stuck" a rule-based perturbation operator would be applied to restart the improvement process with minimal loss in solution quality (route length).

Our contributions We directly compare two RL models in the literature that have taken different approaches to the VRP. We then combine these models to create a pipeline that is more efficient and guarantees better solutions. We then consider the performance of each model when ES is used instead of PG and the benefits of a pipeline of these ES trained models.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

2 ES pipeline

Since the so-called "Learn 2 Improve" (L2I) local-search method developed by Lu et al. [2020] uses an initial solution, it may greatly benefit from having an initial solution generated by an end-to-end model rather than starting with a random solution. This is not just true of L2I, but of all local-search methods. Thus, we propose a pipeline that begins with a generated solution from an end-to-end model and then uses this solution to initialize a local-search method, such as L2I. This is a general pipeline framework, allowing for a variety of combinations to be compared to find optimal results. However, our main interest is in the application of Deep RL to the VRP, so we focus on a pipeline that uses the Attention model developed by Kool et al. [2019] as the end-to-end model and L2I as the local-search method. While the models used in both papers are trained using PG with a baseline, the way that models re-frame the CVRP in an RL context are somewhat different.

Attention Kool et al. [2019] uses an Attention-based encode-decoder model that outputs a solution given a VRP. The solution is incrementally built internally by the decoder which is fed embedded positional information, problem-specific context, and a mask which denotes visited nodes. Since this is done internally, the policy selects a solution given a problem instance. In this way, the action space for the RL environment is the solution space and the Attention model frames the VRP as a one-step episode.

L2I Lu et al. [2020] uses a combination of improvement and perturbation steps that take an initial solution and gradually change it into a more optimal solution. This system has two parts: An improvement controller, which is done using RL, and a perturbation controller, which is rule-based. The RL environment within this system is essentially a multi-armed bandit problem in which the agent is given a solution and is asked to choose the best improvement operator to apply to the solution. If improvements begin to stagnate, that is no improvement has occurred for a predetermined number of steps, a perturbation step will take place and a new solution will be reconstructed from the stagnated solution. Then the improvement process will begin again. The amount of timesteps per "episode", $T = n - p$, where n is the number of rollout steps, or the total improvement/perturbation steps allowed per problem, and p is the amount of perturbations. Lu et al. [2020] uses $n = 40000$, however they also demonstrate that trivially increasing n will lead to depreciating performance gains. In training and experiments, we use $n = 20000$ because we found that it requires half the processing time per problem on average and performs essentially the same. Also, Lu et al. [2020] mentions using an ensemble model in which multiple policies are queried every improvement step and the best solution from the policies is chosen, however we found that similar performances were attainable using only one policy.

Pipeline To train a pipeline of these models, the Attention model must be trained first and then used to generate solutions for training the L2I model.This is due to the fact that the Attention model is trained on batches of problems and the L2I model is trained over multiple steps on a single problem. This splits training into two parts that cannot be done in parallel. While an L2I model that is trained using random initial solutions can be tested with a trained Attention model, we found that there are no performance improvements in doing this. In Figure 1, we have the learning curves of each of the models given 20 hours of training using an Nvidia Tesla V100 GPU. As seen in the figure, the pipeline model was able to match the average tour distance of the L2I in half the amount of time.

2.1 Evolution strategies

In addition to using the original Attention and L2I models in the pipeline, we also create ES versions of the models and build different pipelines with each. In ES, a random population of k perturbations, $\boldsymbol{\epsilon_i} \in N(0, 1)$; $1 \leq i \leq k$, are generated every timestep, t, of the episode. Then k actions, $a_{ti} = \pi(s|\boldsymbol{\theta_t} + \sigma\boldsymbol{\epsilon_i})$, are selected, where θ represents the parameters of the model and σ controls the exploration around θ in the parameter space. The parameters are then updated according to

$$\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} + \frac{1}{k\sigma} \sum_{i=1}^{k} F(a_{ti}|s)\boldsymbol{\epsilon_i}$$

where $F(a|s)$ is the reward function. Since the Attention and L2I models frame the RL environment differently, how ES is used to update the models is also slightly different.

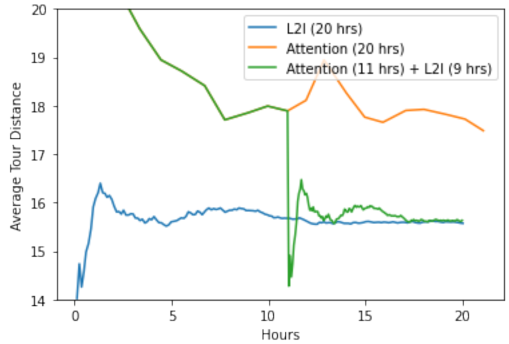

Figure 1: Learning curve of Attention, L2I, and Pipeline models.

Attention w/ ES The Attention model frames the CVRP as a one step episode which means that if updated according to ES, an update would be performed after each episode which would lead to stability issues. Instead, a modified ES update will be applied in which the perturbation population is created before each batch and batches are evaluated for each member of the population. Essentially, instead of updating every timestep, an update is applied with every batch. The learning curve of the Attention model with and without ES is viewable in Figure 2a. The Attention model is trained with an Nvidia Tesla V100 GPU and the ES variant is trained on a CPU using a perturbation population of $k = 5$. The Attention model with PG seems to be more training efficient than the ES variant. This is likely due to the fact that the Attention model frames the VRP as a one-step episode. In their paper, Salimans et al. [2017] mentioned that ES models would perform better when trained with long episodes with a large amount of timesteps. Given larger k and more CPUs, the ES variant would likely do better as it would be able to search a larger portion of the parameter space in less time.

L2I w/ ES ES can be implemented exactly as described earlier to update the L2I model as the CVRP is framed as a traditional multi-step episode. We chose to use $k = 1$ here to minimize the time spent per problem in training. As with the Attention model test, the L2I model is trained with an Nvidia Tesla V100 GPU and the ES variant is trained on a CPU. In Figure 2b, we have the learning curve of the L2I model with and without ES. They are almost identical. The success of the ES variant could be due to the power of the improvement operators available within the L2I method. Nonetheless, further improvement should be possible given larger k and more CPUs.

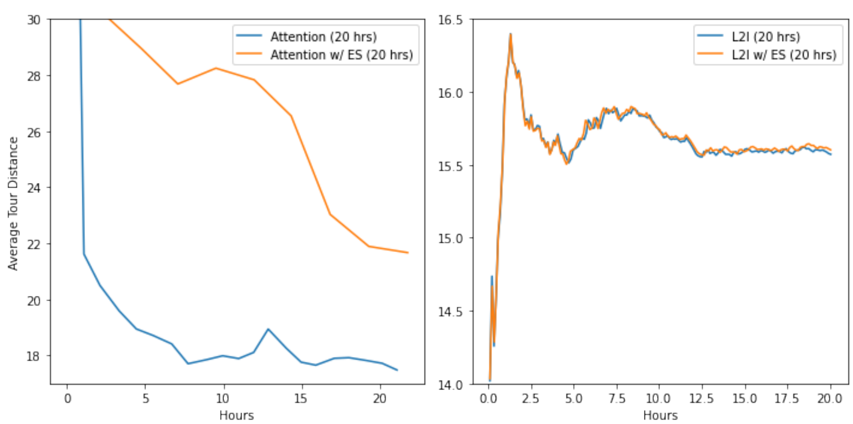

Figure 2: a) Learning curve of Attention w/ and w/o ES and b) L2i model w/ and w/o ES.

Pipelines w/ ES Finally we combined the models and their variants into pipelines and trained them. The ES variant portions of the pipelines were trained on a CPU, while the regular models were trained on an Nvidia Tesla V100 GPU. The training curves of each of these pipelines can be seen in Figure 3. While having different initial starts, the pipelines are essentially the same with minor differences that can be accounted for with the fact that the ES variant of L2I is slower than the original. This seems to show that the L2I method, irrespective of the update method and initial solution, will perform well.

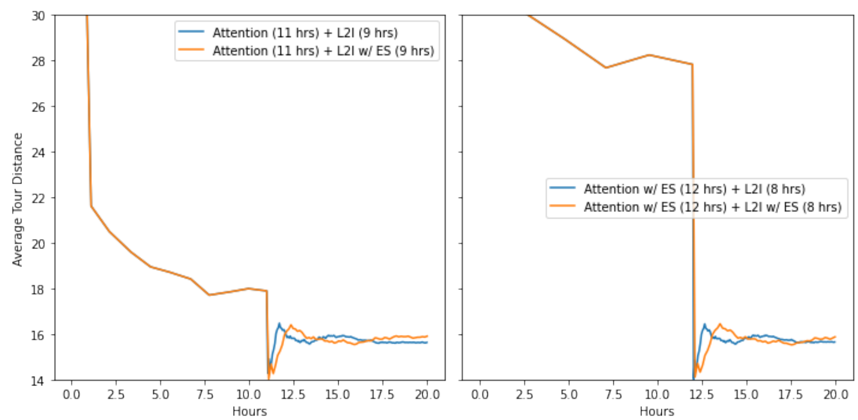

Figure 3: Learning curves of 4 different Pipelines

From the training curves, it is clear that L2I w/ ES can be trained just as efficiently as L2I. This means that, given limited resources, it would be better to have multiple CPUs in parallel to train ES variant than one GPU to train L2I. ES will not benefit every model as seen with Attention, but when it does, it should be applied to further scale the model at a low cost and improve training efficiency.

References

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *ArXiv*, 2018.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *ICLR*, 2019.

Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *ICLR*, 2020.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *ArXiv*, 2017.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *ArXiv*, 2016.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *NIPS*, 2015.