# Neural Geometric Combinatorial Optimization Problem Solver with Deep Reinforcement Learning

**Ziang Gao**[1] , **Mengyu Liu**[1] , **Lu Han**[1]
[1]Department of Electrical Engineering and Computer Science, Syracuse University
{zgao29, mliu71, lhan11}@syr.edu

## Abstract

We introduce a novel neural architecture to solve geometric combinatorial optimization problems. Based on the work of Bello *et al.*, we made some modification to graph embedding and output dictionary size, and then designed our core architecture. We applied the actor-critic reinforcement learning method, but used exponential moving expectation rather than expectation given by critic network. We implemented our model and tested it on three geometric combinatorial optimization problems, convex hull, Delaunary triangular and travelling salesman problem. In the results, we found that our architecture could achieve performance close to the optimal on Traveling Salesman Problem, but did not success on DT and CH problem. Finally, we concluded that our architecture need more improvement for TSP, and the modification of variable output dictionary is too complicated for both DT and CH problem.

## 1 Introduction

Pointer Networks (Ptr-Nets) proposed by Vinyals *et al.*[6] have been used for solving three challenging geometric combinatorial optimization problems - finding planar convex hulls, computing Delaunay triangulations, and the planar Travelling Salesman Problem (TSP) - using training examples alone. However, their architecture limited them to settings where the inputs are node coordinates and the model is trained with supervision. Recently, Bello *et al.*[1] introduced a new model that eliminated former constrain by using a linear transformation to map each node to an embedding, which is graph embedding[3], and a LSTM encoder to combine those embedded vectors to a latent vector for the decoder, and the later constrain by reinforcement learning (RL). Their improvement has made it possible to use neural networks solve TSP with up to 100 nodes nearly optimally.

Nonetheless, the work of Bello *et al.*, which we consider to be a general model to solve geometric combinatorial optimization problems, has not been tested on other problems yet. In addition, whether its architecture has room for improvement is still unknown. Those facts motivated us to conduct research on the combination of that model with two other problems and explore the way to improve its architecture. In our project, we proposed a small modification to the architecture and answered how to apply the model to convex hull and Delaunary triangular problems, which does not exist in any other existing works. We also analyzed the sensitivity of the model by testing it on 5 different hyper-parameter configurations and performing an ablation study for each problem. We provide our Pytorch implementation on Github , Code of CH Problem.

## 2 Methods

### 2.1 Architecture

**Core Network**

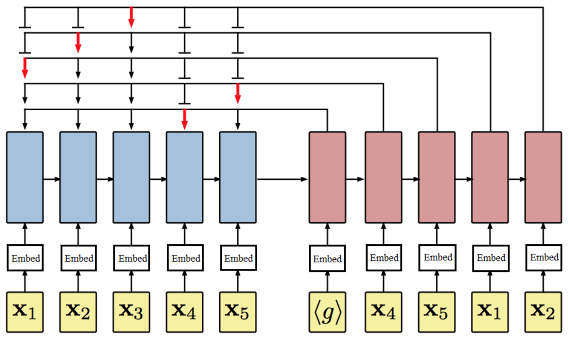


**Figure 1:** Our architecture

We basically followed the architecture of Bello *et al.*[1] and did some novel modification to it. Our architecture is shown in **Figure 1**

The followings are our modifications to the core network including our motivations to do them.

**Graph Embedding** In the architecture of Bello *et al.*, given a input graph represented by a tensor of node coordinates, it first uses a linear transformation as a graph embedding to map each node in the graph to a latent vector. We analyzed the rationality of introducing graph embedding. Although it increases the number of parameters in the network as well as the computing complexity, it clarifies the work of the LSTM encoder. Since the input sequence and output sequence are different, the two latent spaces that the two vectors belong to, which are both subsets of $\mathbb{R}^m$ where $m$ is the size of hidden units in the LSTM encoder, are different, which means we need another mapping from the graph space to the solution space. In Ptr-Net, the encoder has to do double work by learning a mapping from the input graph to the graph space and the second mapping, making it difficult to learning both mapping precisely. However, in the architecture of Bello *et al.*, the graph embedding does the job of learning the first mapping and thus frees the encoder from the double work. Therefore, the introduction of graph embedding is rational and necessary, but a linear transformation is too simple to represent the mapping, which is likely to be non-linear, precisely. One of state-of-art graph embedding algorithms using neural network is Structural Deep Network Embedding (SDNE)[7], which embeds huge graphs well using an autoencoder architecture of several dense layers. We believe multi-layer autoencoder can also achieve great performance in our architecture. So we use a three-layer autoencoder as our embedding, in which the number of neurons gradually decreases from the first layer to the third layer as Wang *et al.*[7] did.

**Output Dictionary Size** Because for TSP the output dictionary size is always equal to the number of nodes, Bello *et al.* fixed the size in their work. In our project, however, we have two problems having output dictionaries of variable sizes. To be specific, the output dictionary size of convex hull problem for neural network is equal to the number of nodes.But for the final result, the clip of the numbers of nodes is based on the amount of maximum area in subset of input nodes. While that of Delaunary triangulation is likely to be greater than the number of nodes. Therefore, we made the output dictionary size of our model variable.

For DT problem, because we assigning a probability for each pick, if we pick a zero, which means the stop sign, then we stop, we will have a variable length sequence output.We modified the loop in the forward part of the Ptr-Net, because the length of the solution should be variable, so we need to change it from a for loop to a while loop. However, then this will make each prediction have different size and the shape of the batch could not be formed as an matrix. So we add zeros to the end to make the batch shape as a matrix.

For CH problem, the size of output of the neural network is set to be same as the size of input.We find the whole sequence of the points.But there is tricky for giving a reasonable result of the CH problem. By using iteration, the areas of polygon of subset are calculated. The maximum area combination is selected to be the the final sequence of the solution given by the system. The maximum area is also the reward of reinforcement learning system.

**Actor-Critic Learning**
Actor-critic learning is one of the most state-of-art reinforcement learning method. Bello *et al.* used the core network as an actor, chose Euclidean tour length as the reward for TSP, and used a neural network as the critic to estimate the expected tour length from the input sequence of solution.Instead of simply minimizing the total tour length, they tried to minimize the difference between the actor's total tour length and the critic's expected tour length, which is also called the advantage. Their objective function is

$$J(\theta|s) = \mathbb{E}_{\pi\sim p_\theta(\cdot|s)} L(\pi|s) \tag{1}$$

where $\theta$ represents the parameters in the actor, $s$ is a graph sampled from the graph dataset with a distribution $S$, $\pi$ means a tour, $L(\pi|s)$ is the tour length of a tour $\pi$ as the solution to $s$, and $p_\theta(\pi|s)$ is a stochastic policy given by the actor with parameters $\theta$. The expectation $b(s)$ is given by the critic. If the advantage is defined as $L(\pi|s) - b(s)$, then the goal should be minimize this advantage since the tour length is expected to be as short as possible. Based on the REINFORCE[8] algorithm, the gradient of the advantage is formulated by

$$\nabla_\theta J(\theta|s) = \mathbb{E}_{\pi\sim p_\theta(\cdot|s)}[(L(\pi|s) - b(s))\nabla_\theta \log p_\theta(\pi|s)] \tag{2}$$

For mini-batch training, by randomly sampling graphs $s_i$ for $i \in [1, ...n]$ from a distribution $S$ into a batch of size $B$, and sampling a single tour $\pi_i \sim p_\theta(\cdot|s_i)$ per graph, the gradient in (2) can be approximated with Monte Carlo sampling as follows:

$$\nabla_\theta J(\theta|s) \approx \frac{1}{B}\sum_{i=1}^{B}[(L(\pi_i|s_i) - b(s))\nabla_\theta \log p_\theta(\pi_i|s_i)] \tag{3}$$

We can then train the model with mini-batches and use an optimizer, such as Adam optimizer, to minimize the objective function based on the gradient in (3).

In our project, based on the reinforcement learning of TSP in the work of Bello *et al.*, we designed our own reinforcement learning methods for other two problems. For DT problem, we use a different reward function since we did not care about the length anymore. We use the ratio of 1 - percentage of intersection over percentage of union, where $S_p$ is the predicted solution and $S_opt$ is the optimal solution generated by scipy library. The main goal of this format is trying to make the reward converge to 0, in other words, the more accurate predicted solution, the lower reward.

$$Reward_{DT} = \frac{1 - S_p \wedge S_{opt}}{S_p \vee S_{opt}} \tag{4}$$

For the CH problem, the maximum subset areas are taken to be the reward as early mentioned before.

$$Reward_{CH} = max(s_1, s_2, ..s_n) \tag{5}$$

In the above equation $s_i$ = area the subset of output sequence. $Reward_{CH}$ is the reward of the reinforcement learning system for the CH problem.

Next, we designed our training procedure based on the policy gradient algorithm above, and used Adam optimizer to minimize each objective functions. The size of our training dataset is 1,280,000. We did not add noise to the gradient, or use dropout, batch normalization or any other tricks in our training. To examine the quality of training results, we used a validation dataset of small size (12,800) to test the trained model regularly during each training. No final test of any pre-trained model is done and the results from the validation tests are used to evaluate the performance of the pre-trained models.

### Training Procedure and Update Rule

For TSP, the training procedure and update rule, which are similar to those of Bello *et al.*, are shown in **Algorithm 3** in **Appendix A**. In the algorithm, $L^t(\pi_i^t|s_i^t)$ is the reward of an input graph and a corresponding solution at training step $t$, $E^t$ is the critic's expectation at time $t$, $g_\theta^t$ is the policy gradient of model parameter $\theta$ at time $t$, $\theta^t$ is the model parameter at time $t$. Since we do not use a network as a critic but make the critic's expectation move exponentially, our training procedure does not include anything about the critic network, and, obviously, the update rule of the expectation is also different. The rest of those of Bello *et al.* remains the same in the algorithm.

For DT, the training procedure and update rule, which are similar to those of Bello *et al.*, are shown in **Algorithm 2** in **Appendix A**. In this algorithm, other terms and expression are the same as those in TSP, the difference is the while loop break condition and the reward part.

For CH, the training procedure and update rule, which are similar to those of Bello *et al.*, are shown in **Algorithm 1** in **Appendix A**. In this algorithm,

## 2.2 Experimental Tasks

Since all of us are dealing with planar geometric problems, our datasets are all sets of two-dimensional matrices storing a series of coordinates. To be specific, a matrix in a dataset of TSP is

$$\begin{bmatrix} x_1 & \dots & x_i & \dots & x_n \\ y_1 & \dots & y_i & \dots & y_n \end{bmatrix} \tag{6}$$

where $n$ is the number of nodes in a problem instance. We chose $n = 20$ in all problems. For Delaunay triangulation problem, a pair of $-1$ is appended before the first entry of the matrix to represent the stop sign. The dataset is generated by choosing $x_i$ and $y_i$ randomly from an uniform distribution on $(0, 1)$. As mentioned before, our training dataset consists of 1,280,000 problem instances and our validation dataset contains 12,800 problem instances.

### Convex Hull Problem

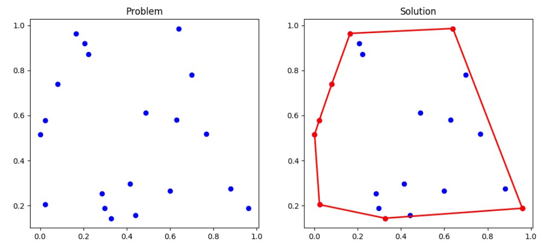


**Figure 2:** An example of Convex Hull and its solution

Lu Han works on planar convex hull problem. In mathematics, the convex hull or convex envelope or convex closure of a set X of points in the Euclidean plane or in a Euclidean space (or, more generally, in an affine space over the reals) is the smallest convex set that contains X. For instance, when X is a bounded subset of the plane, the convex hull may be visualized as the shape enclosed by a rubber band stretched around X[2]. The time complexities of output-sensitive algorithms to planar convex hull problem are $O(n \log n)$.

### Delaunay Triangulation Problem

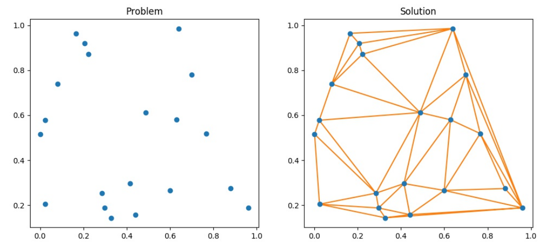


**Figure 3:** An example of Delaunay Triangulation and its solution

Mengyu Liu is responsible for planar Delaunary triangulation problem. In mathematics and computational geometry, a Delaunay triangulation (also known as a Delone triangulation) for a given set $P$ of discrete points in a plane is a triangulation $DT(P)$ such that no point in $P$ is inside the circumcircle of any triangle in $DT(P)$. Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation; they tend to avoid sliver triangles. The triangulation is named after Boris Delaunay for his work on this topic from 1934[4]. The Delaunary triangulation problem are also $O(n \log n)$.

Delaunay Triangulation is more complex than the other 2 problems, the input of three questions are the same, but the TSP return the same number of vertices, Convex Hull return less number of vertices, but Delaunay Triangulation return more number of vertices usually, because the result is a list of triples, each triple represents three vertices of a triangle, and no guarantee triples are pairwise-disjoint. For example, if we have a 20 vertices graph. The original graph as the left figure shows, and the DT result as shown in right figure. But in some sense, DT is easier, because we don't need the coordinates of vertices, we only need the number of vertices, which reduce 2 dimensions to 1 dimension. Also, the order of permutation will not change the triangulation result.

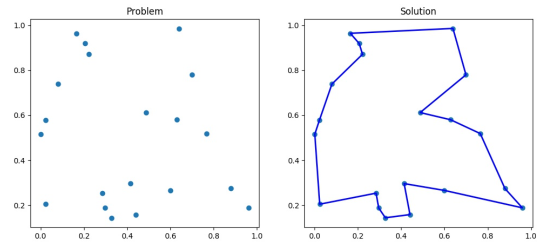


**Figure 4:** An example of TSP and its solution

**Travelling Salesman Problem**
Ziang Gao focus on Travelling Salesman Problem. The Travelling Salesman Problem (TSP), deals with creating the ideal path that a salesman would take while travelling between cities. The solution to any given TSP would be the Shortest way to visit a finite number of cities, visiting each city only once, and then returning to the starting point. TSP[5] is a kind of NP-hard combinatorial optimization problem. The best known exact dynamic programming algorithm for TSP has a complexity of $\Theta(2^n n^2)$, making it infeasible to scale up to large instances.

The problem of TSP is a 2D graph of several nodes and the solution to TSP is a sequence of either the coordinates or the serial numbers of the nodes in the problem. **Figure 4** shows an example of TSP. On the left side is a graph of several nodes as the input of a TSP, and the tour as the solution is drawn on the right side.

## 3 Results

### 3.1 Convex Hull Problem

The CH project is implemented as previously designed.The three groups of data-set are used to train and test. The outcome graph is shown below. There is no significantly improve as the time elapse.The result supervise learning of the convex hull are better than this version of reinforcement learning.The unsatisfied result may be caused by the complexity of the reward function. The reward function seems to be hard for the neural network to recognize the target to do the policy gradient. In the future, the simplicity of the reward function would be designed to make the training work successfully.

### 3.2 Delaunary Triangulation Problem

Unfortunately, due to shortage of time and the difficulty of the problem, the framework we used can only run one batch, after this batch, the program will exit, parameters become

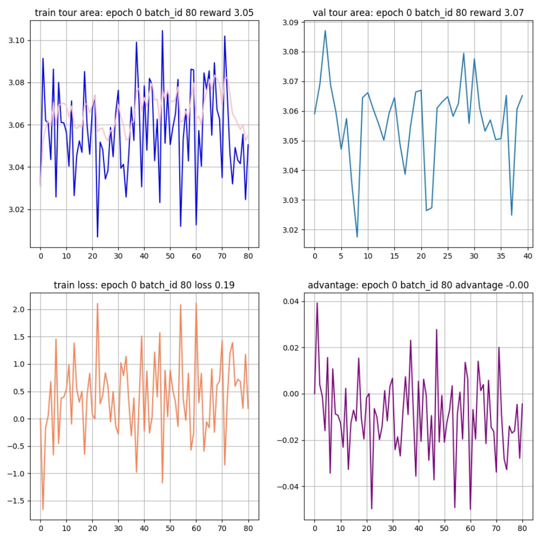


**Figure 5:** Results of training of convex hull

Nan, which will make the back propagation meaningless. I did not figure out where the bug is, may be the shape of layers are not matched. What we have after the batch are predicted sequence(i.e. partial solution) and the rewards result of the reinforcement(Could be viewed in debug mode).

If you run the $dt\_main.py$, this will load the modified data set and print out the first batch generated by Ptr-Net, then the program will crash. No other results will be produced.

### 3.3 Traveling Salesman Problem

**Default Configuration Training and Test**
The default configuration for TSP can be checked in Appendix B. It is chosen based on the configuration of the model with the best performance in the work of Bello *et al.*. **Figure 6** shows the results of the training and test with the default configuration. Here four kinds of training and testing logs are summarized, which are the training tour length, validation tour length, training advantage and training loss. The definition of the tour length and advantage can be checked in **Actor-Critic Learning**. The gradient of training loss is (3).

As can be seen in **Figure 6**, the tour length in both training and test decreased quickly and converged after only 3 epochs. The pink line in the graph of training tour length is the expectation of the critic. It shows the similar descent process as the tour length of the actor, which means the exponential moving could make the critic not only have patient to wait for the actor's progress when the actor get stuck, but also upgrade its standards timely after the actor makes a progress. It is this back-and-forth process that makes the advantage oscillate

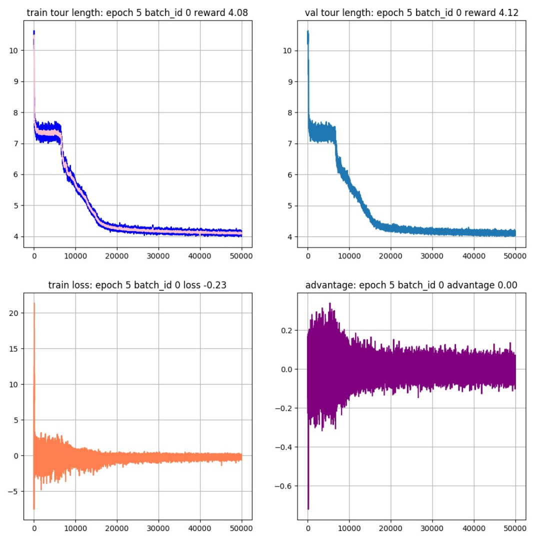


**Figure 6:** Results of training with the default configuration for TSP

up and down 0. It seems that the mean of the training loss is also 0, but the decrease of its amplitude is enough to make the model learn a lot. Finally, the tour lengths at the end of training and test are 4.08 and 4.12 respectively, but unfortunately it seemed to be pretty close to the optimal (3.82), neither of them surpassed the work of Bello *et al.* (3.82) or even the Ptr-Net with supervised learning (3.88).

**Different Configuration Training and Test**

Ziang Gao believes that the modifications of the following hyperparameters may have influences on the model's performance on TSP. First, the hidden units of both encoder and decoder are considered to determine the network's memory capacity. The greater the number of hidden units is, the larger memory capacity the network will have, and then the better the pre-trained model's performance is likely to be. So, in order to improve the performance, he did a experiment to increase the hidden units from 128 to 256. Moreover, the embedding size, the length of the embedding vector, is also believed to play a important role in the network. The embedding size determines the complexity of the embedding space. If it is too small, the embedding vector is probably not able to represent the problem graph well, making the solution worse or totally wrong. Thus, he decreased the embedding size from 128 to 64 in another experiment to see if the performance is worse. Also, the important coefficient $beta$ in the actor-critic algorithm was changed from 0.9 to 0.8 in the third experiment and the clip of the gradient was changed from 2.0 to 5.0 in the fourth experiment. In the last experiment, the number of glimpses was increased to 5.

The results of the experiments above were merged together into four 3D graphs (shown in **Figure 7**) for comparison, each of them is a result from the training tour length, the testing tour length, the training advantage and the training loss. All changes made to the model did not effect the convergence of the learning process. The differences between them was mainly the speed of convergence. It can be seen in the graphs that the decrease of $\beta$ and the increase of the maximum gradient can make the learning process converge faster.

**Ablation Study**

Since the change of the method for graph embedding is one of the main novelty of our architecture, it will be interesting to do an ablation study on this part. Adding noise in the validation was chosen to perturb the model after training. The added noises are a Gaussian noise with a mean of 0, a standard deviation of 1, and a Gaussian noise with a mean of 0.5 and a standard deviation of 1. This ablation study was only done with the default configuration and the results are shown in **Figure 8**. The tour lengths of both of them could still decrease and converge in training and test, but their performance became worse, both of them achieved a tour length of 4.16 in training and over 4.45 in test. Thus, adding noises to the embedding layer did have an impact on the performance. In addition, since the noise was only added in test, we can infer from the slight difference between the tour length of two kinds of noises that severer noises will do more harm. Therefore, the embedding layer is one of the most significant parts in our architecture to ensure a good performance for TSP.

## 4 Discussion

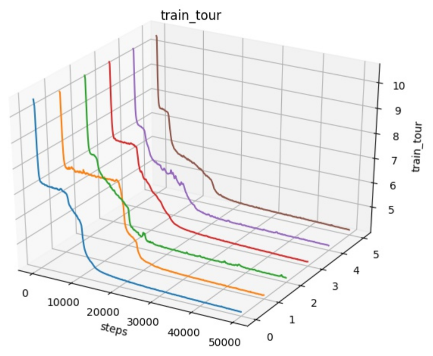
### 4.1 Convex Hull Problem

In the solution of CH problem, the area reward replace the target as shown in the Vinyal *et al.*[6] paper. In the rest of part, the project just optimize the structure of design in the Vinyal's paper.

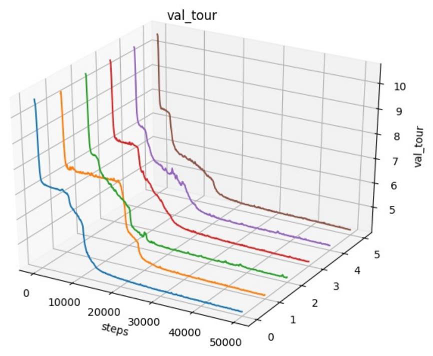
The most challenging part of CH problem project is the design of variable output. At first, the modification of the structure of neural network has been attempted. But lack of experience to deal with this variable length, the result is not as expected.Therefore, the second modification is based on the after-processing instead of network itself. Logically, this method guarantee the completeness of network and meet the requirement of the CH problem in the same time.

The performance of the CH problem reinforcement solution is not fully successful. The design of the critics is the initial part to be responsible for this result. The supervised learning is more suitable for this problem. The modification of the reward method may help to improve the performance.

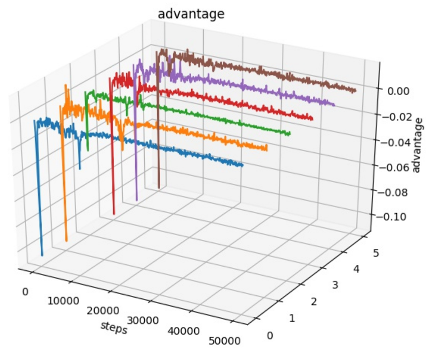
In the future work, the first priority is to analyze the exact reason to make the policy gradient fail. The concrete analysis help us to realize the problem and fix the related part.



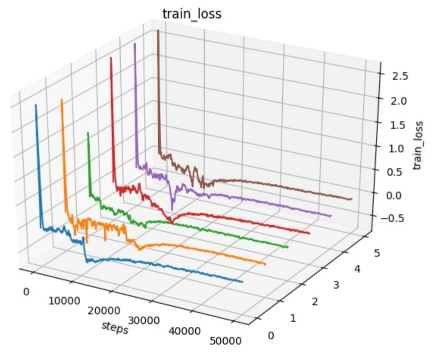
(a) training tour length



(b) testing tour length



(c) training advantage



(d) training loss

**Figure 7:** Results of experiments with different configurations for TSP
blue: embedding size = 64; orange: hidden size = 256; green: $\beta$ = 0.8;
red: default; purple: maximum gradient = 5.0; dark red: number of glimpses = 5

## 4.2 Delaunary Triangulation Problem

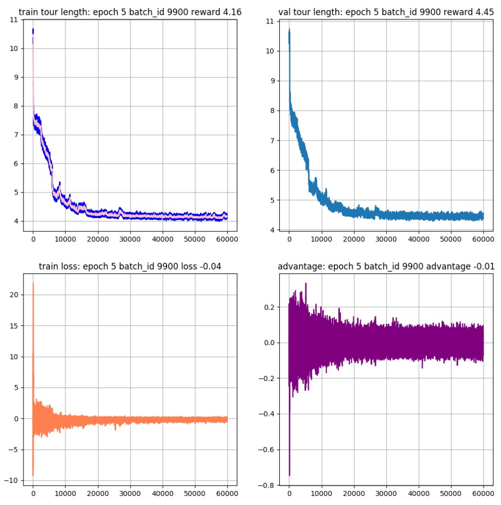
Vinyals did not include DT problem test results as a table in his paper, but he mentioned the accuracy for the 20 points problem is $22.8\%$ which is depressing[6]. And there is no single implementation online of solving DT problem using Ptr-Net. So I doubt whether this framework is suitable for this problem. I think the most important parts we did for this problem are the modified graph embedding, new reward function and the variable length sequence.

I think the most challenging part of this project is trying to understand what the each layer will output, and what do they mean. It cost us about 30 hours to figure out how to produce a variable length sequence. Because there is no single implementation available for DT problem, so it is impossible to compare the results.

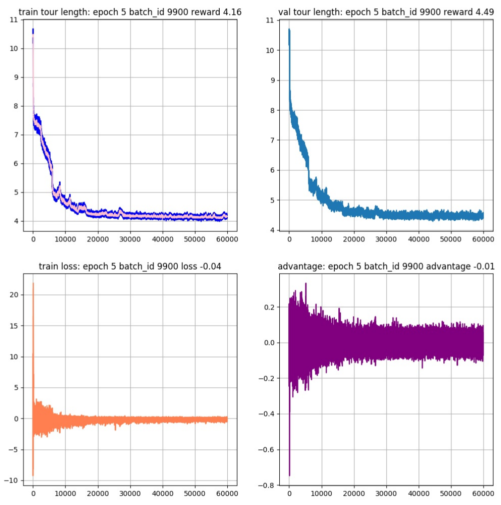
If I were to continue working on this project, the next highest priority task is developing an average sequence length controller. For different number of points task, we should control the number of triangles we create. For example, for 20 points problem, we could control the sequence length from 20 to 35 since a length 3 sequence does not possible to cover all triangles.This range should be learned from existing algorithms as prior knowledge for training. I think this will increase the accuracy and coverage rate to an acceptable level or the training time will be shorter since this problem is pretty special.

## 4.3 Traveling Salesman Problem

For TSP, our architecture was tested with the default configurations and other different configurations. The results showed that our architecture could be trained quickly and achieve the performance close to the optimal. It was expected to achieve the same performance as the work of



(a) mean: 0, standard: 1



(b) mean: 0.5, standard: 1

**Figure 8:** Results of ablation study for TSP

Bello *et al.* and surpass Ptr-Net, but unfortunately it did not. The most challenging parts was the understanding of the attention mechanism. The experiments with different configurations showed that the decrease of $\beta$ and the increase of the maximum gradient can make the learning process converge faster. And the ablation study showed the significance of the embedding layer in the model's learning for TSP.

In the future, the study of this project should be first focused on the problems with more nodes. Only problems with 20 nodes have been tested for convenience, but the increase of nodes will surely challenge the architecture and thus provide more possibility for us to improve it.

## References

[1] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016. https://openreview.net/pdf?id=Bk9mxlSFx.

[2] William F Eddy. A new convex hull algorithm for planar sets. *ACM Trans. Math. Softw.*, 3(4):398–403, 1977. https://arxiv.org/pdf/1212.6043.pdf.

[3] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018. https://arxiv.org/pdf/1705.02801.pdf.

[4] Der-Tsai Lee and Bruce J Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, 1980. https://link.springer.com/article/10.1007/BF02574375.

[5] Amanur Rahman Saiyed. The traveling salesman problem. *Indiana State University*, 2:1–15, 2012. https://s3.amazonaws.com/academia.edu.documents/33829999/aman.pdf.

[6] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015. http://papers.nips.cc/paper/5866-pointer-networks.pdf.

[7] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016. https://www.kdd.org/kdd2016/papers/files/rfp0191-wangAemb.pdf.

[8] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. https://link.springer.com/content/pdf/10.1007/BF00992696.pdf.