Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 00 (2015) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

24th International Meshing Roundtable (IMR24)

Recurrent Neural Networks for Geometric Problems

Meire Fortunato[a,*], Oriol Vinyals[b,*], Navdeep Jaitly[b]

[a]*University of California, Berkeley, Berkeley, CA, USA*
[b]*Google Inc., Mountain View, CA, USA*

Abstract

We introduce a new architecture using Recurrent Neural Networks (RRNs) to learn approximate solutions to three geometric problems – finding planar convex hulls, computing Delaunay triangulations, and the planar Travelling Salesman Problem – using training examples alone. We hope our results on these tasks will encourage a broader exploration of neural learning for discrete geometric problems, including mesh generation.

Peer-review under responsibility of organizing committee of the 24th International Meshing Roundtable (IMR24).

Keywords: Recurrent Neural Networks, Geometric Problems, Machine Learning

1. Introduction

RNNs have achieved state-of-the-art results in core problems in natural language processing such as translation [1,2] and parsing [3], image and video captioning [4,5]. However, the methods used on these cases still require the size of the output dictionary to be fixed *a priori*. We address this limitation by repurposing the attention mechanism of [2] to create pointers to input elements. We show that the resulting architecture, which we name Pointer Networks (Ptr-Nets), can be trained to output satisfactory solutions to three combinatorial optimization problems – computing planar convex hulls, Delaunay triangulations and the symmetric planar Travelling Salesman Problem (TSP). The resulting models produce approximate solutions to these problems in a purely data driven fashion (i.e., when we only have examples of inputs and desired outputs). The proposed approach is depicted in Figure 1.

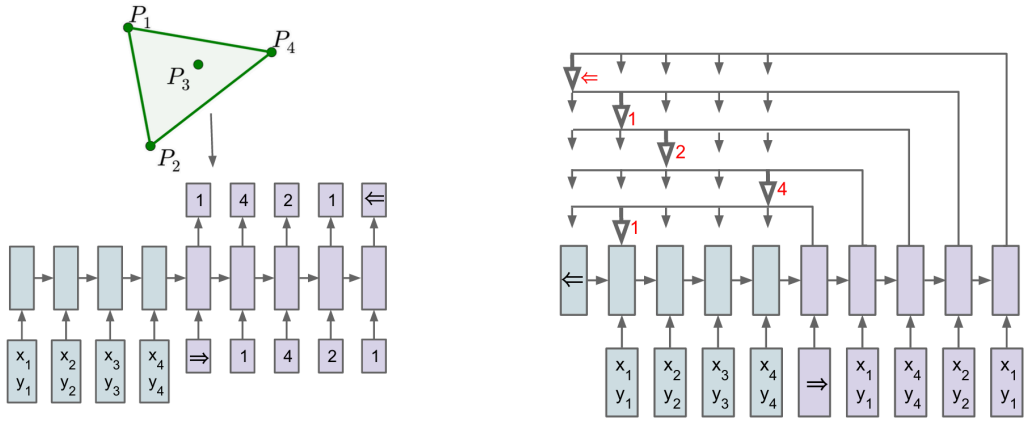
The main contributions of our work are as follows:

- We propose a new architecture, that we call Pointer Net, which is simple and effective.
- We apply the Pointer Net model to three distinct geometric problems (including finding Delaunay triangulations) and we obtain promising results.
- For the first time, a purely data driven approach can learn approximate solutions to geometric problems.

* Equal contribution.

E-mail addresses: meirefortunato@berkeley.edu (Meire Fortunato)., vinyals@google.com (Oriol Vinyals)., ndjaitly@google.com (Navdeep Jaitly).

Peer-review under responsibility of organizing committee of the 24th International Meshing Roundtable (IMR24).



(a) Sequence-to-Sequence (b) Ptr-Net

Fig. 1: **(a)** *Sequence-to-Sequence* - An RNN (blue) processes the input sequence to create a code vector that is used to generate the output sequence (purple) using the probability chain rule and another RNN. The output dimensionality is fixed by the dimensionality of the problem and it is the same during training and inference [1]. **(b)** *Ptr-Net* - An encoding RNN converts the input sequence to a code (blue) that is fed to the generating network (purple). At each step, the generating network produces a vector that modulates a content-based attention mechanism over inputs. The output of the attention mechanism is a softmax distribution with dictionary size equal to the length of the input.

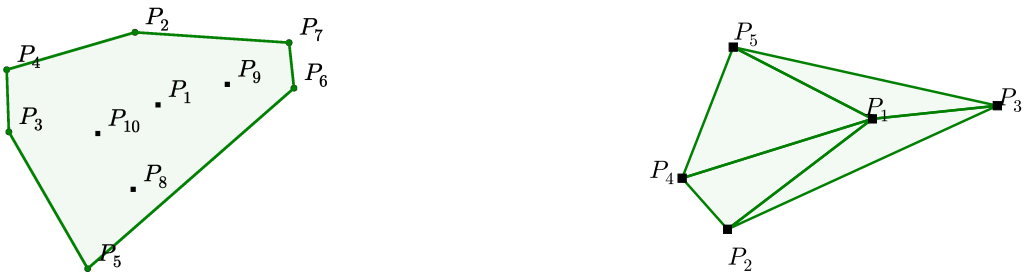
2. Models

A review of models [1] and [2], and the full description of our new Ptr-net model can be found in the more complete version of this work [6].

3. Motivation and Datasets Structure

In the following sections, we review the three problems we considered, as well as our data generation protocol.

In the training data, the inputs are planar point sets $\mathcal{P} = \{P_1, \ldots, P_n\}$ with n elements each, where $P_j = (x_j, y_j)$ are the cartesian coordinates of the points. In all cases, we sample from a uniform distribution in $[0, 1] \times [0, 1]$. The outputs $\mathcal{C}^{\mathcal{P}} = \{C_1, \ldots, C_{m(\mathcal{P})}\}$ are sequences representing the solution associated to the point set $\mathcal{P}$. In Figure 2, we find an illustration of an input/output pair $(\mathcal{P}, \mathcal{C}^{\mathcal{P}})$ for the convex hull and the Delaunay problems.



(a) Input sequence $\mathcal{P} = \{P_1, \ldots, P_{10}\}$, and the output sequence $\mathcal{C}^{\mathcal{P}} = \{\Rightarrow, 2, 4, 3, 5, 6, 7, 2, \Leftarrow\}$ representing its convex hull.

(b) Input $\mathcal{P} = \{P_1, \ldots, P_5\}$, and the output $\mathcal{C}^{\mathcal{P}} = \{\Rightarrow, (1, 2, 4), (1, 4, 5), (1, 3, 5), (1, 2, 3), \Leftarrow\}$ representing its Delaunay Triangulation.

Fig. 2: Input/output for (**a**) convex hull and (**b**) Delaunay triangulation. Here $\Rightarrow$ and $\Leftarrow$ represent beginning and end of sequence, respectively.

3.1. Convex Hull

We used this example as a baseline to develop our models and to understand the difficulty of solving combinatorial problems with data driven approaches. Finding the convex hull of a finite number of points is a well understood task in computational geometry, and there are several exact solutions available. In general, these solutions have $O(n \log n)$ time complexity, where n is the number of points considered.

Here, the elements of the output sequence C_i are indices between 1 and n corresponding to positions in the sequence $\mathcal{P}$, or special tokens representing beginning or end of sequence. See Figure 2 (a) for an illustration.

3.2. Delaunay Triangulation

A Delaunay triangulation for a set $\mathcal{P}$ of points in a plane is a triangulation such that each circumcircle of every triangle is empty, that is, there is no point from $\mathcal{P}$ in its interior. Exact $O(n \log n)$ solutions are available, where n is the number of points in $\mathcal{P}$.

In this example, the outputs $C^{\mathcal{P}} = \{C_1, \ldots, C_{m(\mathcal{P})}\}$ are the corresponding sequences representing the triangulation of the point set $\mathcal{P}$. Each C_i is a triple of integers from 1 to n corresponding to the position of triangle vertices in $\mathcal{P}$ or the beginning/end of sequence tokens. See Figure 2 (b).

3.3. Travelling Salesman Problem (TSP)

We focused on the planar symmetric TSP: given a list of cities, we wish to find the shortest possible route that visits each city exactly once and returns to the starting point. Additionally, we assume the distance between two cities is the same in each opposite direction.

Here the output sequence $C^{\mathcal{P}} = \{C_1, \ldots, C_n\}$ will be a permutation of integers from 1 to n representing the optimal path (or tour). For consistency, in the training dataset, we always start in the first city without loss of generality.

To generate exact data, we implemented the Held-Karp algorithm [7] which finds the optimal solution in $O(2^n n^2)$ (we used it up to $n = 20$). For larger n, producing exact solutions is extremely costly, therefore we also considered algorithms that produce approximated solutions: A1 [8] and A2 [9], which are both $O(n^2)$, and A3 [10] which implements the $O(n^3)$ Christofides algorithm.

4. Empirical Results

For details of the architecture or hyperparameter used for the Ptr-Nets results presented here, see the Architecture and Hyperparameters section on [6].

4.1. Convex Hull

We used the convex hull as the guiding task which allowed us to understand the deficiencies of standard models such as the sequence-to-sequence approach, and also setting up our expectations on what a purely data driven model would be able to achieve with respect to an exact solution.

We reported two metrics: accuracy, and area covered of the true convex hull (note that any simple polygon will have full intersection with the true convex hull). To compute the accuracy, we considered two output sequences C^1 and C^2 to be the same if they represent the same polygon. For simplicity, we only computed the area coverage for the test examples in which the output represents a simple polygon (i.e., without self-intersections). If an algorithm fails to produce a simple polygon in more than 1% of the cases, we simply reported FAIL.

The results are presented in Table 1 (a). We note that the area coverage achieved with the Ptr-Net is close to 100%. Looking at examples of mistakes, we see that most problems come from points that are aligned (see Figure 3 (d) for a mistake for $n = 500$) – this is a common source of errors in most algorithms to solve the convex hull.

The bottom half of Table 1 (a) shows that, when training our model on a variety of lengths ranging from 5 to 50, a single model is able to perform quite well on all lengths it has been trained on. More impressive is the fact that the model does extrapolate to lengths that it has never seen during training. Even for $n = 500$, our results are satisfactory and indirectly indicate that the model has learned more than a simple lookup.

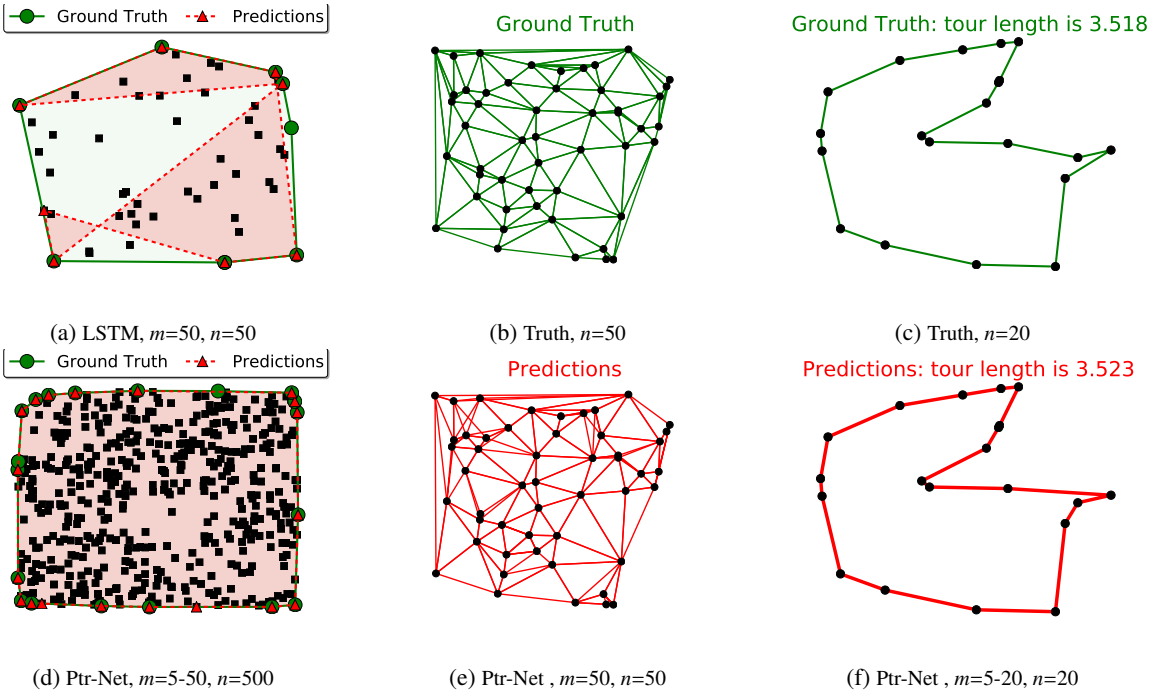
Table 1: In **(a)** , we have a comparison between LSTM, LSTM with attention, and our Ptr-Net model on the convex hull problem. Note that the baselines must be trained on the same n that they are tested on. In **(b)** , we have the tour length of the Ptr-Net and a collection of algorithms on a small scale TSP problem.

(a) Convex Hull problem

Method	trained n	n	Accuracy	Area
LSTM [1]	50	50	1.9%	FAIL
+attention [2]	50	50	38.9%	99.7%
Ptr-Net	50	50	72.6%	99.9%
LSTM [1]	5	5	87.7%	99.6%
Ptr-Net	5-50	5	92.0%	99.6%
LSTM [1]	10	10	29.9%	FAIL
Ptr-Net	5-50	10	87.0%	99.8%
Ptr-Net	5-50	50	69.6%	99.9%
Ptr-Net	5-50	100	50.3%	99.9%
Ptr-Net	5-50	200	22.1%	99.9%
Ptr-Net	5-50	500	1.3%	99.2%

(b) TSP problem.

n	Optimal	A1	A2	A3	Ptr-Net
5	2.12	2.18	2.12	2.12	2.12
10	2.87	3.07	2.87	2.87	2.88
50 (A1 trained)	N/A	6.46	5.84	5.79	6.42
50 (A3 trained)	N/A	6.46	5.84	5.79	6.09
5 (5-20 trained)	2.12	2.18	2.12	2.12	2.12
10 (5-20 trained)	2.87	3.07	2.87	2.87	2.87
20 (5-20 trained)	3.83	4.24	3.86	3.85	3.88
25 (5-20 trained)	N/A	4.71	4.27	4.24	4.30
30 (5-20 trained)	N/A	5.11	4.63	4.60	4.72
40 (5-20 trained)	N/A	5.82	5.27	5.23	5.91
50 (5-20 trained)	N/A	6.46	5.84	5.79	7.66



(a) LSTM, m=50, n=50 (b) Truth, n=50 (c) Truth, n=20

(d) Ptr-Net, m=5-50, n=500 (e) Ptr-Net , m=50, n=50 (f) Ptr-Net , m=5-20, n=20

Fig. 3: Examples of our model on Convex hulls (left), Delaunay (center) and TSP (right), trained on m points, and tested on n points. A failure of the LSTM sequence-to-sequence model for Convex hulls is shown in (a). Note that the baselines cannot be applied to a different length from training.

4.2. Delaunay Triangulation

The Delaunay Triangulation test case is connected to our first problem of finding the convex hull. In fact, the Delaunay Triangulation for a given set of points triangulates the convex hull of these points.

We reported two metrics: accuracy and triangle coverage in percentage (the percentage of triangles the model predicted correctly). Note that, in this case, for an input point set $\mathcal{P}$, the output sequence $C(\mathcal{P})$ is, in fact, a set. As a consequence, any permutation of its elements will represent the same triangulation.

Using the Ptr-Net model for $n = 5$, we obtained an accuracy of 80.7% and triangle coverage of 93.0%. For $n = 10$, the accuracy was 22.6% and the triangle coverage 81.3%. For $n = 50$, we did not produce any precisely correct triangulation, but obtained 52.8% triangle coverage. See the middle column of Figure 3 for an example for $n = 50$.

4.3. Travelling Salesman Problem

We considered the planar symmetric travelling salesman problem (TSP), which is NP-hard as the third problem. Given that the Ptr-Net implements an $O(n^2)$ algorithm, it was unclear if it would have enough capacity to learn a useful algorithm solely from data.

Table 1 (b) shows all of our results on TSP. The number reported is the length of the proposed tour. Unlike the convex hull and Delaunay triangulation cases, where the decoder was unconstrained, in this example we set the beam search procedure to only consider valid tours. Otherwise, the Ptr-Net model would sometimes output an invalid tour – for instance, it would repeat two cities or decided to ignore a destination. This procedure was relevant for $n > 20$, where at least 10% of instances would not produce any valid tour.

The first group of rows in the table show the Ptr-Net trained on optimal data, except for $n = 50$, since that is not feasible computationally (we trained a separate model for each n). Interestingly, when using the worst algorithm (A1) data to train the Ptr-Net, our model outperforms the algorithm that is trying to imitate.

The second group of rows in the table show how the Ptr-Net trained on optimal data with 5 to 20 cities can generalize beyond that. The results are virtually perfect for $n = 25$, and good for $n = 30$, but it seems to break for 40 and beyond (still, the results are far better than chance). This contrasts with the convex hull case, where we were able to generalize by a factor of 10. However, the underlying algorithms are of far greater complexity than $O(n \log n)$, which could explain this phenomenon.

5. Conclusions

In this paper we described Ptr-Net, a new architecture for RNNs that allows us to naturally use our model to solve geometric problems – such as computing convex hulls and finding Delaunay triangulations.

Even though we have not adjusted the model to be problem dependent, the results using the general framework are quite good. An alternative to improve performance on a specific task, would be to add domain knowledge to the model. For example, in the Delaunay triangulation case, at decoding time, we could constrain the beam search to only output valid triangulations for the initial set of points. We could also start with any triangulation for the point set, and let the neural network decide if an edge should be flipped or not. We hope that this work will be explored further, and RNNs will start being used by the community in order to solve geometric problems.

References

[1] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in: Advances in Neural Information Processing Systems, 2014, pp. 3104–3112.
[2] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, In ICLR 2015, arXiv preprint arXiv:1409.0473 (2014).
[3] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, G. Hinton, Grammar as a foreign language, arXiv preprint arXiv:1412.7449 (2014).
[4] O. Vinyals, A. Toshev, S. Bengio, D. Erhan, Show and tell: A neural image caption generator, In CVPR 2015, arXiv preprint arXiv:1411.4555 (2014).
[5] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, T. Darrell, Long-term recurrent convolutional networks for visual recognition and description, In CVPR 2015, arXiv preprint arXiv:1411.4389 (2014).
[6] O. Vinyals, M. Fortunato, N. Jaitly, Pointer networks, arXiv preprint arXiv:1506.03134 (2015).
[7] R. Bellman, Dynamic programming treatment of the travelling salesman problem, Journal of the ACM (JACM) 9 (1962) 61–63.
[8] Suboptimal travelling salesman problem (tsp) solver, . Available at https://github.com/dmishin/tsp-solver.
[9] Traveling salesman problem c++ implementation, . Available at https://github.com/samlbest/traveling-salesman.
[10] C++ implementation of traveling salesman problem using christofides and 2-opt, . Available at https://github.com/beckysag/traveling-salesman.