CSLT TECHNICAL REPORT-2015XXXX [Wednesday 6th January, 2016]

An implementation of Pointer-Networks with Extensions

Xiaoxi Wang[1] and Dong Wang[1,2*]

*Correspondence: wang-dong99@mails.tsinghua.edu.cn
[1]Center for Speech and Language Technology, Research Institute of Information Technology, Tsinghua University, ROOM 1-303, BLDG FIT, 100084 Beijing, China
Full list of author information is available at the end of the article

Abstract

This report presents the technique for implementing Pointer-Networks as well as extending the models in two ways: deploying an incremental strategy for refine results for large size problems and adding extra memory slots to enhance the attention mechanism in order to generate more accurate results. By doing such preliminary research on these two extensions, we show the potential advantage of the incrementally learning strategy and memory structures that can benefit our future works.

Keywords: deep learning; sequence-to-sequence model; attention mechanism; convex hull problems

1 Introduction

This report presents the technique for building an implementation of Pointer-Networks (Ptr-Nets) [1]. The Ptr-Nets are RNN Encoder-Decoder models which are designed based upon the sequence-to-sequence (Seq-to-Seq) models [2] with content-based attention mechanism[3]. Traditionally, the general Seq-to-Seq model can deal with input sequences with varies of length, while the dictionary of outputs must be fixed. However, in many problems such as sorting, the size of output dictionaries vary according to its inputs. This makes it impossible to train a Seq-to-Seq model on size n problems also fit on a size of $n' \neq n$ problems. In order to overcome this shortage, the Ptr-Nets deployed content-based input attention as a pointing mechanism and replace the traditional dictionary-looking-up outputs by using the pointers as the outputs. Comparing with the Seq-to-Seq model, Ptr-Nets produce the probabilities of selecting over the encoded units of the input sequence, so the size of outputs is equal to the length of input sequences and changes when the input changes. By doing so, the Ptr-Nets are capable of solving several combinatorial problems, such as Convex Hull Problems, Delaunay Triangle Problems and Traveling Salesman Problems (TSPs).

In this paper we also proposes two extensions: 1) an incremental decoding strategy and 2) an encoding method with parallel memory slots. In the first extension we applied an incremental algorithm into the decoding process of Ptr-Nets model. It is only applicable when outputs is in a subset of corresponding inputs and no element appears more than once in the outputs, e.g. the 2-D convex hull problems. More specifically, the inputs are divided into several batches with a fixed size. At the first turn, the first batch is sent to the model and the model generates a sub-solution, then for each following turn, the output from last turn is joined to the current input batch and both are sent to the encoder to generate next sub-solution, until all batches are accepted and the solution is finally generated. The motivation of this strategy is to improve the solution when the size of problems is large. From

our experiments on Convex Hull Problems, we found that the Ptr-Nets can generate hulls that are very close to the target hulls; however, the accuracy of whether the exact points are generated correctly is dropping down significantly when the size of problems growing large.This is due to the difficulty of neural network models in generating solution meticulously; therefore, the incremental strategy can produce a much better results by refining the solutions repeatedly.

In the second extension, we add memory slots in parallel to the encoded states. The memories are hidden units encoded in the same way as its original encoder but different parameters. Then the latest hidden unit of the original encoder is used as the initial state of the decoder, and the memories are used for the attention mechanism in its decoding process. As we know, in Seq-to-Seq model, the hidden units in encoders are not only representations of input points, but also saved partial context information that had been encoded so far; therefore, our hypothesis is that the encoded information of each input and the context information may interfere each other. By adding an extra memory, during the attention precess in decoding the model will point back to the memory instead of the LSTM encoded hidden unit, so the encoder can focus on producing the context information. Out experiment results shows that this method outperforms the baseline results.

2 Pointer Networks Review

The Pointer Network is a generation model that can be seen as a combination of sequence-to-sequence model and attention mechanism. The vanilla sequence-to-sequence model is composed by two RNNs, one encoder and one decoder. The Long Short Term Memory (LSTM) is used here for the RNNs. It use a input sequence $\mathcal{P}$ to produce the conditional probability of output sequence $\mathcal{C}^{\mathcal{P}}$, which can be represented as:

$$p(\mathcal{C}^{\mathcal{P}}|\mathcal{P};\theta) = \prod_{i=1}^{m(\mathcal{P})} p_\theta(\mathcal{C}_i|\mathcal{C}_1,...,\mathcal{C}_{i-1},\mathcal{P};\theta) \tag{1}$$

where θ denotes the parameter of the model and $(\mathcal{P},\mathcal{C}^{\mathcal{P}})$ denotes the pair of an input sequence and an output sequence. For the Ptr-Nets, $\mathcal{P} = \{P_1,...,P_n\}$ is a sequence of n vectors and $\mathcal{C}^{\mathcal{P}} = \{C_1,...,C_{m(\mathcal{P})}\}$ is a sequence of $m(\mathcal{P})$ indices, each between 1 and n. In addition, two special symbols $\Rightarrow$ and $\Leftarrow$ are used in the Ptr-Net model, where the $\Leftarrow$ works as a termination symbol of the output sequence and the $\Rightarrow$ is fed to the model just after the input sequence to tell the model switch to generation mode. More specifically, let $(e_1,...,e_n)$ and $(d_1,...,d_m(\mathcal{P}))$ denote the hidden states of encoder and decoder respectively. At time step j, the current hiddene_j is encoded from P_j as well as the last state e_{j-1}. For the first state e_1, an state that indicate the termination symbol $\Leftarrow$ is used as the previous state e_0. Once the encoding process is finished, the last state e_n has been encoded with the necessary information of the overall problem, so it is then used as the initial state for the decoder. Once the $\Rightarrow$ symbol is fed to the model, the model start generating decoding states $(d_1,...,d_m(\mathcal{P}))$ until the termination symbol $\Rightarrow$ is generated.

The most distinct advantage in Ptr-Nets is using outputs as pointers that can selecting from the encoding states. This is based upon the content-based attention mechanism[3]. The attention mechanism works as a soft search through the hidden states of the encoder. Once a state d_i is generated from the decoder at time step i, it is used to evaluate a focusing

weight a_j^i with each hidden state e_j in the encoder. This weight can be seen as a measurement of the relation between the state d_i and each state e_j. In many other papers[4, 5, 6, 7, 8] about the content-based attention mechanism, a_j^i is used to compute an "attention vector" d_i' on each time i, and using d_i' as additional information for decoding. All these process can be represented as follows:

$$\begin{aligned} u_j^i &= v^T \tanh(W_1 e_j + W_2 d_j) \quad j \in (1, ..., n) \\ a_j^i &= \text{softmax}(u_j^i) \quad\quad\quad\quad\quad\quad\quad\quad j \in (1, ..., n) \\ d_i' &= \sum_{j=1}^{n} a_j^i e_j \end{aligned} \tag{2}$$

where u_j^i can be seen as a focus weight on state e_j on time i and a_j^i is a softmax normalised version of u_j^i, so the attention vector d_i' is a weighted sum of all encoder hidden units.

However, since the pointer network using the attention mechanism as a pointer, instead of obtaining an "attention vector" over a soft search, it choose an a from all $a_j^i, j \in (1, ..., n)$ as a pointer, which is located according to:

$$\begin{aligned} u_j^i &= v^T \tanh(W_1 e_j + W_2 d_i) \quad j \in (1, ..., n) \\ p(\mathcal{C}_i | \mathcal{C}_1, ..., \mathcal{C}_{i-1}, \mathcal{P}) &= \text{softmax}(u^i) \end{aligned} \tag{3}$$

therefore applying a hard search over all hidden states. The input element correspond to the chosen state is now be pointed as the output for current d_i. As a result, the Ptr-Net select outputs from its input elements rather than using a fixed vocabulary.

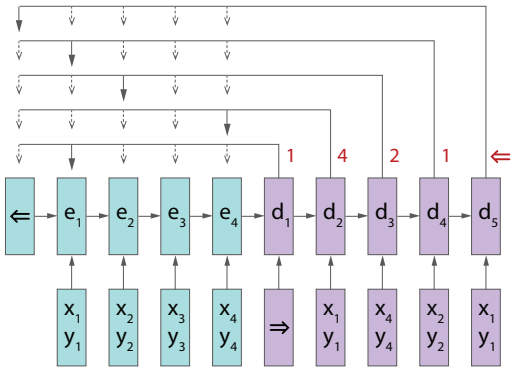

Figure 1: Ptr-Nets.

3 Incrementally Learning

The incrementally learning is a decoding strategy, which can benefit the performance on large size problems. Given a problem $P = \{p_1, p_2, ..., p_n\}$, where each p_j represents the coordinate of point j, we divide P into several disjoint subsets $P_1, P_2, ...$, where each subset contains w_{inc} elements. Initially, the model is fed with P_1 and generates an sub-solution $\mathcal{C}_1$, then the points correspond to $\mathcal{C}_1$ is united to P_2 and feed to the model to generate sub-solution $\mathcal{C}_2$. By repeating this, each step we add w_{inc} more points to the existing sub-solution to complete the solution. The experiment result is in section 5.2.

4 Adding Extra Memories

For adding memory slots in parallel to the encoded units, we encode another series of hidden states as the same way as the encoder but with different parameters. Let $(e_1, ..., e_n)$ denote the hidden units of the encoder and $(e'_1, ..., e'_n)$ denote the encoded memory slots. Then the equation 3 can be changed as follows:

$$\begin{aligned} u^i_j &= v^T \tanh(W_1 e'_j + W_2 d_i) \quad j \in (1, ..., n) \\ p(\mathcal{C}_i|\mathcal{C}_1, ..., \mathcal{C}_{i-1}, \mathcal{P}) &= \text{softmax}(u^i) \end{aligned} \tag{4}$$

And the original e_n is only worked as the initial state of decoder for decoding the d_i.

The changed model is depicted as Fig.2:

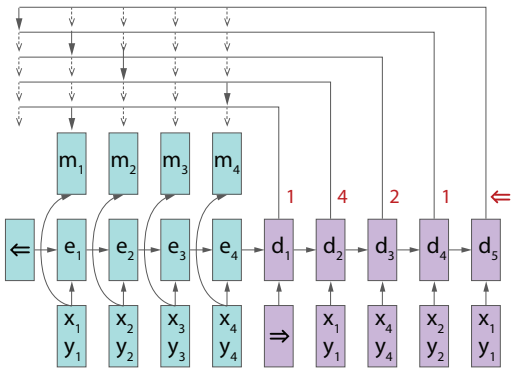

Figure 2: Ptr-Nets with Memories.

5 Experiments

The task we used to evaluated the model is the 2-D Convex Hull Problems. The training data contains $1M$ examples. Each example has the 2D coordinates of a point set with n elements and the corresponding convex hull to this point set. Here n is a random number within a range of $[5, 50]$ and selected according to the uniform distribution. For the test set, there are nine set of problems in size of $\{5, 10, 25, 50, 100, 200, 300, 400, 500\}$ in total. Each set contains 1K examples.

Our model is a typical RNN LSTM model with a dimension of 512. For training we use *adadelta* to reach a relatively good local minimum and then switch to *rmsprop* to refine the model and achieve much better results.

5.1 The Baseline

We train our models before we know the exact setting of Vinyals, et.al's [1] baseline[1], so our baseline is a bit difference from Vinyals, et.al's. For Vinyals, et.al's baseline, the data settings contains 920K examples, 20K examples for each n from 5 to 50. The data also follow two order: (1) The first element of the outputs is always the smallest one in the sequence; (2) the points are sorted in counter-clockwise order. However, in our experiments we found that it is hard to reach the baseline results by following this order, but by starting from the elements that closed to the down-left corner, the model can reach a even better

[1] https://github.com/meirefortunato/Pointer_Networks/blob/master/Convex_Hull_README.txt

# points		5			10		
		accuracy	validation	area	accuracy	validation	area
Vinyals, et.al's baseline		92.0%	≥99%	99.6%	87.0%	≥99%	99.8%
Baseline		93.31%	99.45%	99.82%	90.92%	99.61%	99.97%
Incrementally learning (w_{inc})	1	-	-	-	-	-	-
	5	-	-	-	-	-	-
	10	-	-	-	-	-	-
	25	-	-	-	-	-	-
	50	-	-	-	-	-	-
Memory		93.69%	99.29%	99.84%	93.38%	99.56%	99.98%
# points		25			50		
		accuracy	validation	area	accuracy	validation	area
Vinyals, et.al's baseline		-	-	-	69.6%	≥99%	99.9%
Baseline		82.28%	98.08%	99.99%	67.93%	96.10%	99.98%
Incrementally learning (w_{inc})	1	-	-	-	60.45%	99.36%	99.69%
	5	-	-	-	69.08%	99.08%	99.98%
	10	-	-	-	69.03%	98.51%	99.98%
	25	-	-	-	67.64%	96.74%	99.98%
	50	-	-	-	67.93%	96.10%	99.98%
Memory		87.81%	99.56%	99.93%	78.16%	99.51%	99.88%
# points		100			200		
		accuracy	validation	area	accuracy	validation	area
Vinyals, et.al's baseline		50.3%	≥99%	99.9%	22.1%	≥99%	99.9%
Baseline		48.69%	99.52%	99.97%	12.63%	99.97%	99.86%
Incrementally learning (w_{inc})	1	43.93%	99.61%	99.88%	21.82%	99.88%	99.91%
	5	49.23%	99.55%	99.97%	24.41%	99.73%	99.94%
	10	49.10%	99.09%	99.97%	24.96%	99.42%	99.94%
	25	47.79%	96.55%	99.97%	25.96%	98.26%	99.95%
	50	47.29%	96.49%	99.97%	24.81%	98.26%	99.95%
Memory		58.95%	100.00%	99.92%	14.84%	99.95%	99.76%
# points		300			400		
		accuracy	validation	area	accuracy	validation	area
Vinyals, et.al's baseline		-	-	-	-	-	-
Baseline		3.00%	99.98%	99.74%	0.83%	100.00%	99.66%
Incrementally learning (w_{inc})	1	10.33%	99.87%	99.89%	5.16%	99.89%	99.87%
	5	11.82%	99.83%	99.91%	6.04%	99.91%	99.89%
	10	12.01%	99.62%	99.92%	6.36%	99.86%	99.89%
	25	12.77%	98.81%	99.92%	6.92%	99.11%	99.90%
	50	11.86%	98.45%	99.92%	6.19%	99.02%	99.90%
Memory		3.35%	99.86%	99.56%	0.85%	99.91%	99.45%
# points		500			-		
		accuracy	validation	area	accuracy	validation	area
Vinyals, et.al's baseline		1.3%	≥99%	99.2%	-	-	-
Baseline		0.19%	99.99%	99.59%	-	-	-
Incrementally learning (w_{inc})	1	2.44%	99.85%	99.84%	-	-	-
	5	2.86%	99.86%	99.86%	-	-	-
	10	3.09%	99.82%	99.87%	-	-	-
	25	3.54%	99.11%	99.88%	-	-	-
	50	3.28%	99.30%	99.87%	-	-	-
Memory		0.24%	99.96%	99.38%	-	-	-

Table 1: Experimental results

results. [2] In addition, in Vinyals, et.al's results, if the validation is below 99.0%, it will be reported as FAIL, but in our result we present the exact validation for reference and we observe that the validation is less than 99.0% in some case (e.g. our baseline on 50 points task.). One possible reason is due to our dataset issue (see footnote[2]). The other possible reason is that Vinyals, et.al may use beam search for decoding, which may improve the performance on large size problems, but no specific settings are declared in their paper. Meanwhile, they may only use standard stochastic gradient descent (SGD), while we are using *rmsprop*, so our baseline shows a slight better results on small size problems.

In table 1, we show both Vinyals, et.al's baseline and our baseline together for comparison in detail.

[2] When writing this report we notice that it could be some difference on the settings between our model and Vinyals, et.al's, we will do more investigation on this in our future works.

5.2 Incrementally Learning

For incrementally learning, the training method is same as the baseline. The only difference is how we deploy the model on the process of testing. Given a test example, we first divided the input sequence into groups, where each group contains w_{inc} elements (the last group may contains less than w_{inc} elements). The w_{inc} is denoted as incremental window. We test different $w_{inc} \in \{1, 5, 10, 25, 50\}$ on different size of problems (number of points).

For the first turn, the first group of w_{inc} input elements are fed to the model. Then a number of w'_1 outputs will be generated. In following turns, each time we append next group of input elements to the outputs in previous turn, so there are $w'_i + w_{inc}$ elements in the ith in total as the input elements.

The model are evaluated in two ways as same as in [1], accuracy and area covered of the target convex hull. The accuracy is computed as if the output sequence represents the same polygon as the sequence produced by the target sequence.

The results are presented in table 1. We note that the accuracy on number points ≥ 200 outperforms other experiment groups. However, this method costs much more time than the original method.

5.3 Adding Memory Slots

Our experiment results shows that this method outperforms the baseline results in table 1. However, as noted in section5.1, we did not apply the beam search in the decoding, so we should do it in future work to see whether this can also outperform the incrementally learning results on large size problems.

6 Conclusions

In sum, Ptr-Nets can show some unique capabilities for approximately solving combinatorial problems. As the discussion in [9], Ptr-Nets provide an mechanism for selecting discrete targets. However, in [8] Ptr-Nets are viewed as technique for geometry reasoning instead of combinatorial problem deducing, since all three tasks presented in [1] are only in planar spaces. It is worth to argue whether it can solve combinatorial problems in general.Due to the structure limitation of current neural networks, some general combinatorial problems may not be able to represented as some special data structures (e.g, vectors with fixed dimension number) that fit for the current neural networks. From the experimental result, Ptr-Nets can produce good solutions for small size problems but become less meticulous when the size grow large. By deploying incremental strategy, we can obtain relatively well-fined solution. With additional memory slots, the overall performance can be improved but still need refine on large size solutions.

Author details
[1]Center for Speech and Language Technology, Research Institute of Information Technology, Tsinghua University, ROOM 1-303, BLDG FIT, 100084 Beijing, China. [2]Center for Speech and Language Technologies, Division of Technical Innovation and Development, Tsinghua National Laboratory for Information Science and Technology, ROOM 1-303, BLDG FIT, 100084 Beijing, China.

References

1. Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly, "Pointer networks," *arXiv preprint arXiv:1506.03134*, 2015.
2. Ilya Sutskever, Oriol Vinyals, and Quoc VV Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.
3. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.
4. Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan, "Show and tell: A neural image caption generator," *arXiv preprint arXiv:1411.4555*, 2014.
5. Andrej Karpathy, Justin Johnson, and Fei-Fei Li, "Visualizing and understanding recurrent networks," *arXiv preprint arXiv:1506.02078*, 2015.
6. Minh-Thang Luong, Hieu Pham, and Christopher D Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.
7. Alexander M Rush, Sumit Chopra, and Jason Weston, "A neural attention model for abstractive sentence summarization," *arXiv preprint arXiv:1509.00685*, 2015.
8. Tim Rocktaschel, Edward Grefenstette, Karl Moritz Hermann, Tomas Kocisky, and Phil Blunsom, "Reasoning about entailment with neural attention," *arXiv preprint arXiv:1509.06664*, 2015.
9. Kyunghyun Cho, Aaron Courville, and Yoshua Bengio, "Describing multimedia content using attention-based encoder-decoder networks," *Multimedia, IEEE Transactions on*, vol. 17, no. 11, pp. 1875–1886, 2015.