Dynamic Pointer Networks

Liang Xu
xul@fitme.ai

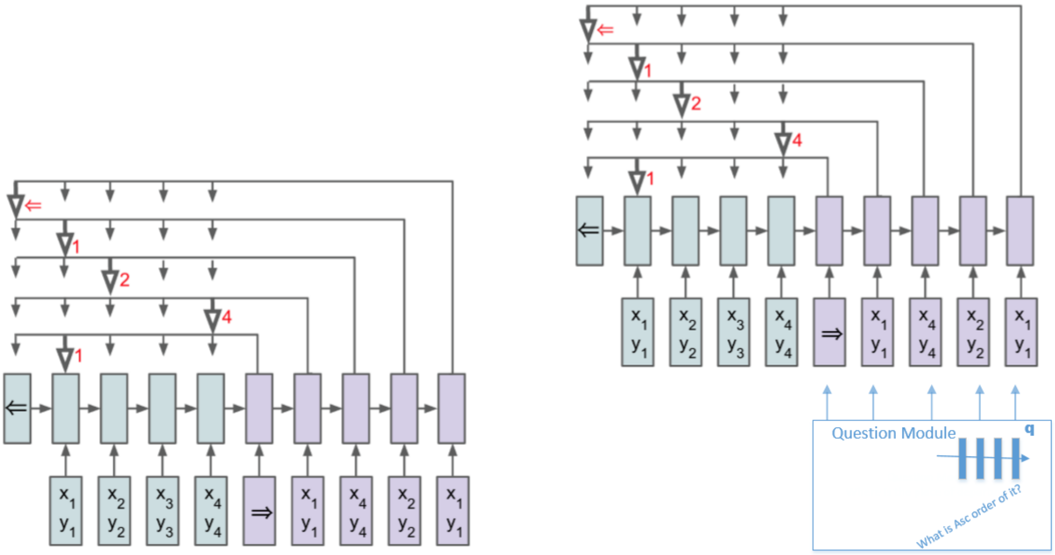
Abstract

We introduce a new architecture able to learn multiple tasks simultaneous and it can retrieve various information from same input, while each element in target sequence point to a position in source input sequence. These problems cannot be handled well by existing models like sequence to sequence model [1], Sequence-to-sequence with Attention model [2], and Pointer Networks [3]. These models could lack pointer mechanism to limit output space to input sequence length or not good at learn multiple mappings. Our model solves the problem by incorporating a new module to provide model with information to indicate mapping relationships. We call this architecture a Dynamic Pointer Net. We show it can learn to performance three distinct algorithmic problems effectively using only data-driven style by training once under one instance of model.

1 Introduction

Sequence-to-sequence model [1] use one RNN to map input sequence to an embedding and use another RNN to map the embedding to an output sequence. Sequence-to-sequence with Attention model [2] use this encoder-decoder structure but add an attention mechanism which allow the model to automatically search for parts of a source sentence that are relevant to predicting a target word. Pointer Network [3] introduce a "pointer" mechanism that learn the conditional probability of an output sequence that corresponding to positions in an input sequence. This mechanism is particular useful for solving problems that output space limit to input space. These developments and recently enhancements made it possible to apply neural network to various domains and achieving new state-of-art results in natural language processing, including machine translation [4], open domain dialogue system [5] and so on.

However, these above models, Pointer Network in particular, can only learn one mapping relationship between input sequence and output sequence at one once, since there is no explicit information to indicate which kinds of relationship it should learn. In this paper, we address this limitation by incorporating a new module representing mapping relationship. Thus, it learns to generate target sequence with intent. We demonstrate that our new architecture, Dynamic Pointer Networks, can be trained to handle three different algorithmic problems under one instance of model – sorting natural number as ascending, sorting natural number as descending, finding largest natural number. The models produce solutions to these problems in purely data driven fashion. Our proposed approach is show in Figure 1.



(a) Pointer Network. (b) Dynamic Pointer Network

Figure 1:

(a) Pointer Network - An encoding RNN convert the input sequence to vector, a decoding RNN generate output sequence using the embedding vector. Attention is used to compute score between encode hidden states and decode hidden states. The output of the attention mechanism is a softmax distribution with dictionary size equal to the length of the input.

(b) Dynamic Pointer Network – same encode and decode with attention mechanism is used. In addition, it adds a question module to represent mapping relationship the model need to learn. To get representation of question, GRU [6] will be used to encode it, as it is compute more efficient than LSTM [7]. Then question will be concatenate to input of decode to form task-oriented information.

The main contributions of our work as follows:

- We propose a new architecture, that we call Dynamic Pointer Net, which is an enhanced version of Pointer Net. It is able to tackle key problem of retrieving a variety of information from same input by incorporating a new component to represent mapping information we care about, while still use a softmax probability distribution as "pointer".
- We apply the Dynamic Pointer Net model to three distinct algorithmic problems. We demonstrate that one instance of model can do multiple tasks simultaneous and only train once.
- We show that a purely data driven approach can solve algorithmic problems.

2 Models

In this section we will describe sequence-to-sequence with attention models [2], and pointer networks [3] in Sections 2.1 and 2.2. We then describe our model Dynamic Pointer Networks in Section 2.3.

2.1 Sequence-to-Sequence Model with Attention

Vanilla sequence-to-sequence model [1] generate output space solely from embedding vector which is encode from input sequence. This setting has a problem of handling long sequence. Sequence-to-sequence with attention model augmented the decoder by propagating extra contextual information from the input.

It first gets attention score from encode hidden states and current decode hidden state using alignment model; Then it computes possibility distribution for input sequence using softmax; finally, it computes weighted sum for input sequence according this possibility distribution as relevant information for current timestamp. This information together with current hidden state of decoder is used to generate a target token.

Concretely, let use define the encoder and decoder hidden state as $(e_1, e_2, \ldots, e_n)$ and $(d_1, d_2, \ldots, d_m)$. We compute the attention vector at each time-stamp i as follows:

$$\begin{aligned} u_j^i &= v^T \tan\left(W_1 \; e_j + \; W_2 \; d_i\right) & j \in (1,2 \ldots, n) \\ a_j^i &= softmax\left(u_j^i\right) & j \in (1,2 \ldots, n) \\ c_i &= \; \textstyle\sum_{j=1}^{n} a_j^i \; e_j \end{aligned} \tag{1}$$

u_j^i is the attention score; a_j^i is possibility distribution, it length is same as input sequence; c_i is the weighted sum representing information we need for current time-stamp. v^T, W_1, W_2 is trainable parameters, they will be learned during training process.

2.2 Pointer Networks

Sequence-to-sequence with attention model can learn to align between input sequence and output sequence dynamically. It is widely used in various tasks. However, it requires output vocabulary size to be fixed. Pointer Network [3] was invented to solve problems where output spaces of target sequences are limited to input sequences. In other words, it is good at solve combinational problems where the size of the output dictionary depends on the length of the input sequence.

Concretely, it use a simpler version of attention mechanism to compute "pointer" at time-stamp i:

$$u_j^i = v^T \tan\left(W_1 \; e_j + \; W_2 \; d_i\right) \qquad j \in (1,2 \ldots, n)$$

$$p(C_i \mid C_1, C_2 \ldots, C_{i-1}, P) = soft\max(u^i) \tag{2}$$

It uses same way as sequence to sequence with attention model to compute attention score. Instead of compute weighted sum based on this attention score, it direct use this attention score to find the "pointer", which represent the target position at this time-stamp. Here C_i represents decoder output, and

$P = \{P_1, P_2 ..., P_n\}$ is a sequence of n vectors representing encode input. Pointer Networks will map the target token to a position from input at each time-stamp.

2.3 Dynamic Pointer Networks

We will start to delineate Dynamic Pointer Network that allows us to retrieve various information from same source sequence under different perspectives. It can be trained once under one instance, and performance multiple tasks.

Question module

In order to let the model to generate target sequence with question in mind, we use a new component called *question module*. The purpose of it is to get representation of question. Question is usually a sequence of tokens. We encode the question via a recurrent neural network.

Give a question of T_Q words, hidden states for the question encoder at time-stamp i is given by：

$$q_i = GRU\left(L\left[w_i^Q, q_{i-1}\right]\right) \tag{3}$$

L represents word embedding matrix and w_i^Q represents the word index of the i word in the question.

We use embedding matrix map each token in question to vector. This embedding matrix will be learned during training. After encoding, we get $q = \{q_1, q_2 ..., q_m\}$. And we use the hidden state of last time-stamp to represent the question.

Incorporate Question with Pointer Mechanism

Question will be concatenate with original decode input to form a new decode input. This decode input is maxed with information from question, so that it will represent input for decode in a more target-oriented way. We get target token based on a "pointer" that indicate the position of input sequence at time-stamp i as follows:

$$d_i = [d_i; q]$$

$$u_j^i = v^T \tan\left(W_1 \ e_j + \ W_2 \ d_i\right) \quad j \in (1,2 \ldots, n) \tag{4}$$

$$p(C_i \mid C_1, C_2 ..., C_{i-1}, P) = soft\max(u^i)$$

As we can see from above formulation, q is the same at each time-stamp.

3 Motivation and Multiple Tasks

3.1 Motivation

Our motivation for Dynamic Pointer Network was come from issue when we exploring ways to do slot filling for our task oriented dialogue system. Some time we need to retrieve several information from same user utterance. For example, where a user says something like this:

"I want to reserve a taxi to *west lake* at *4p.m.* today depart from *xixi national wetland park*".

After detect that user want to *reserve taxi*, we need to retrieve necessary information, such as *destination, start time, place of departure*. Pointer Network can be rained to retrieve information, but it cannot be used directly to retrieve multiple information from different perspective based on same input.

3.2 Multiple tasks and its settings

In the training data, the inputs are natural numbers sets $P=\{P_1,P_2...,P_n\}$ with n elements each. Input sequence is generated by select n numbers from a set of natural number, and shuffle randomly. Output sequence $\mathrm{C}=\{\mathrm{C}_1\ ,\mathrm{C}_2\ \ldots,\mathrm{C}_\mathrm{m}\}$ is a sequence of number representing solution for a specific problem, each number is "pointer" to a position to input sequence. We show an input/output pair (P,C) for sorting natural number in ascending , sorting natural number in descending and find largest natural number problem in Figure 2.

(a) Sorting natural number in ascending:
question is: "what is the ascending order for this sequence"?
Input $P=\{5,1,7,3,6,2,4,8\}$, and output sequence $\mathrm{C}=\{=>,1,5,3,6,0,4,2,7,<=\}$

(b) Sorting natural number in descending:
question is "what is the biggest value in this sequence"?
Input $P=\{5,1,7,3,6,2,4,8\}$, and output sequence $\mathrm{C}=\{=>,0,1,2,3,4,5,6,0,<=\}$

(c) Find largest natural number:
question is: "what is the ascending order for this sequence"?
Input $P=\{5,1,7,3,6,2,4,8\}$, and output sequence $\mathrm{C}=\{=>,7,2,4,0,6,3,5,1,<=\}$

Figure 2: Input/output representation for three different tasks. => and <= represent beginning and end of sequence. Notice that in our example, input is all the same, but output sequence is different.

For find largest natural number task, instead of assign of only output the index for the largest natural number, we assign from $\{0,1,2,3..\}$ and reassign 0 to the position with the largest value. We use this style for our task to facilitate our experiment.

4 Experiment Results

We will first detail our configuration for training the model, its accuracy in tasks and then look in details of some examples of model's prediction.

4.1 Training and Accuracy

We train one instance of model, and then performance prediction for all three different tasks. Our model use single layer of GRU with 100,200 or 300 hidden units, optimized by Adam with a learning rate of 0.0001, weight random initialized from

-0.1 to 0.1, gradient clipping of 5.0, l2 regularization of 0.0001, batch size is set to 64. No hyper-parameter fine tuning is used through the experiment.

Train in alternate mode:

n	Accu-asc	Accu-des	Accu-max	Accu-avg	Steps
4	100.0%	100.0%	100.0%	100.0%	100
8	100.0%	100.0%	100.0%	100.0%	350
20	40.7%	42.0%	95.4%	59.4%	1,000
20	86.0%	86.8%	95.5%	89.4%	4,000
20	98.4%	98.9%	97.7%	98.2%	9,000

Table 1: Accuracy of three different tasks under one model. Train tasks in alternate model, in other words, in one batch we train first task, in another batch we train the second task, then train the third task. *n* represent length of target sequence. Asc, Dec, Max, Avg indicates ascend, descend, largest value, average accuracy. Steps stand for how many steps be trained when corresponding accuracy was reported.

Train in mix mode:

n	Accu	Steps
4	100.0%	100
8	100.0%	350
20	64.4%	1,000
20	93.1%	4,000
20	98.6%	9,000

Table 2: Accuracy of tasks under one model. We train three tasks together, means that we put three different tasks together in mini batch.

From above table we can see that for short length of sequence, it can learn to performance multiple task quickly. When sequence length is 8, it only takes 350 steps to reach 100% accuracy, and the total possible combination is 40,320; when sequence length is 20, the total possible combination is 2.4e19, which is impossible to do brute force, or remember all possible data by the model.

And as sequence become longer, it longer time to reach than short sequence, but both mode can receive high accuracy in a short period of time. The accuracy is significant higher than accuracy of random guess, which is less than 1% (for sorting problem accuracy will be much less than 0.01%).

Compare to train three tasks in alternate mode, train in mix mode reach higher performance in same training steps, but as model approach to coverage, performance difference become smaller and smaller.

4.2 Case Study

Below we show some specific prediction from our model.

n=8, train in mix mode

Task	Input	Output	Prediction

Asc sort	[5, 3, 4, 6, 2, 8, 1, 7]	[6,4,1,2,0,3,7,5]	[6,4,1,2,0,3,7,5]
Des sort	[5, 3, 4, 6, 2, 8, 1, 7]	[5,7,3,0,2,1,4,6]	[5,7,3,0,2,1,4,6]
Max	[5, 3, 4, 6, 2, 8, 1, 7]	[0,1,2,3,4,0,6,7]	[0,1,2,3,4,0,6,7]

n=20, train in mix mode

Asc sort	[7,14,11,19,18,8,12,9,13,3, 5, 1,6,17,16,15,20,4,10,2]	[11,19,9,17,10,12,0,5,7, 18,2,6,8,1,15,14,13,4,3,16]	[11,19,9,17,10,12,0,5,7, 18,2,6,8,1,15,14,13,4,3,16]
Des sort	[7,14,11,19,18,8,12,9,13,3, 5, 1,6,17,16,15,20,4,10,2]	[16,3,4,13,14,15,1,8,6,2, 18,7,5, 0,12,10,17,9,19, 11]	[16,3,4,13,14,15,1,8,6,2, 18,7,5,0,12,10,17,9,19,18]
Max	[7,14,11,19,18,8,12,9,13,3, 5, 1,6,17,16,15,20,4,10,2]	[0,1,2,3,4,5,6,7,8,9,10, 11,12,13,14,15,0,17,18,19]]	[0,1,2,3,4,5,6,7,8,9,10, 11, 12,13,14,15,0,17,18,19]

Table 3: Model’s prediction of different tasks using same input at different target sequence length.

when n (sequence length) is 8, model can make correct prediction for all three distinct tasks; when n is 20, from the example we can see that two tasks including sort in asc order and find max value, it make correct prediction. And for our example, it makes a mistake in last token of output sequence in des sort task. But as model approach to coverage closer and closer, finally the model will able to make correct prediction completely.

5 Conclusions

In this paper we described Dynamic Pointer Network, a new architecture that allows us to learn multiple tasks simultaneous that retrieve multiple information from same input, while each element in target sequence point to a position in source sequence. We demonstrate that Dynamic Pointer Network can be used to learn three algorithmic problems effectively in short sequence though data-driven style. We want to apply this model to other applications, including slot filling which motivate to create it. We also want to explore variant and complex architecture based on the idea proposal in this paper, and try to tackle more difficult problems. An implementation of this model is available at Github [8].

References

[1] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems, pages 3104–3112, 2014.

[2 Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In ICLR 2015, arXiv preprint arXiv:1409.0473, 2014.

[3] Oriol Vinyals, Meire Fortunato, Navdeep Jaitly. Pointer Networks. arXiv preprint ArXiv:1506.03134,2015

[4] Yonghui Wu, Mike Schuster, Zhifeng Chen el at. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

[5] Iulian V Serban, Alessandro Sordoni,Yoshua Bengio, Aaron Courville, and Joelle Pineau. 2015. Building end-to-end dialogue systems using generative hierarchical neural network models. In Proc.of AAAI.

[6] *Chung, Junyoung; Gulcehre, Caglar; Cho, KyungHyun; Bengio, Yoshua (2014). "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". arXiv:1412.3555*

[7] Sepp Hochreiter and J¨urgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[8] Tensorflow implementation of sorting natural number as ascending, sorting natural number as descending, finding largest natural number problem. Available at https://github.com/brightmart/dynamic_pointer_network