Dynamic Pointer Networks

Liang Xu
xul@fitme.ai

Abstract

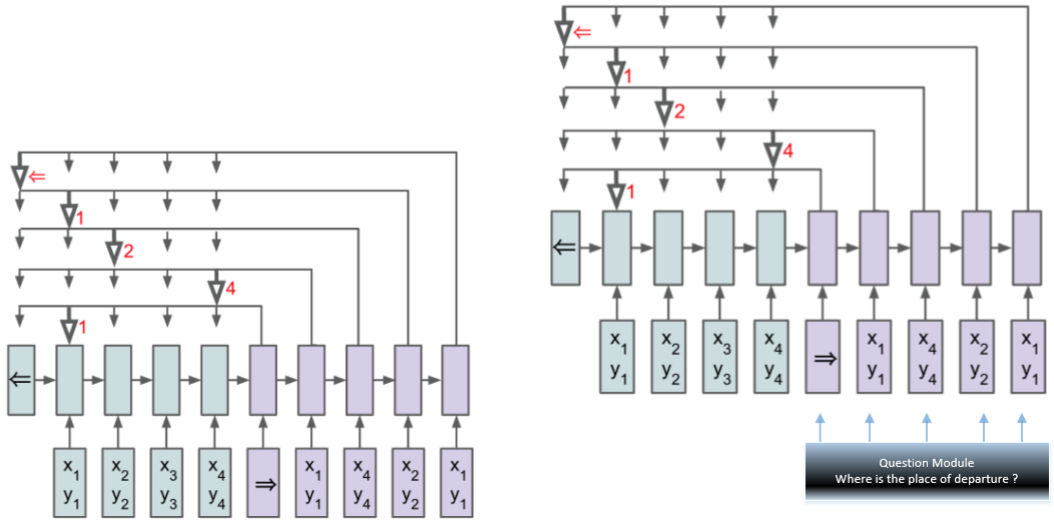
We introduce a new architecture able to learn multiple tasks simultaneous and various information can be retrieve from same input, while each element in target sequence point to a position in source input sequence. These problems cannot be handled well by existing models like sequence to sequence model [1], Sequence-to-sequence with Attention model [2], and Pointer Networks [3]. These models 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. It 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], machine reading comprehension [5], chatbot [6] 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 and goal. 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 [7] will be used to encode it, 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 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 model can do multiple tasks simultaneous and only train once.
- We show that a purely data driven approach can solve algorithmic problems precisely.

2 Models

In 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 gets 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 $(e_1, e_2, \dots, e_n)$ and $(d_1, d_2, \dots, d_m)$. We compute the attention vector at each time-stamp i as follows:

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

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 we care about. v^T, W_1, W_2 is trainable parameters, it 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:

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

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

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 $P = \{P_1, P_2 \ldots, P_n\}$ 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 describe 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 :

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

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 encode, we get $q = \{q_1, q_2 ..., q_m\}$.

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 which is a from a "pointer" indicate the position of input sequence at time-stamp i as follows:

$$d_i = [d_i; q_i]$$

$$u_j^i = v^T \tanh\left(W_1\ e_j + \ W_2\ d_i\right) \qquad j \in (1,2 \ldots, n) \qquad (4)$$

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

3 Motivation and Multiple Tasks

3.1 Motivation

Our motivation for Dynamic Pointer Network was come from issue when we were doing slot filling for our task oriented dialogue system. Some time we need to retrieve several information from same user utterance. For example, where user say 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*, or each ask relevant questions. Pointer Network can be rained to retrieve information, but it lacks of mechanism to handle retrieve multiple information from different perspective.

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 $C=\{C_1\ ,C_2\ \ldots,C_m\}$ is a sequence of number represent 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 $C=\{=>,1,5,3,6,0,4,2,7,<=\}$
(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 $C=\{=>,0,0,0,0,0,0,0,1<=\}$
(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 $C=\{=>,7,2,4,0,6,3,5,1<=\}$

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.

We assign each element in output sequence to 0 except for the largest natural number.

4 Experiment Results

4.1 Training

We use one stance of model and only train once, and then performance prediction. Our model use single layer of GRU with 100,200 or 300 hidden units, optimized by Adam with a learning rate of 0.001, 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.

	n	**Accu-Asc**	**Accu-Dec**	**Accu-Max**	**Steps at report**
	4	**100.0%**	**100.0%**	**100.0%**	**100**
	8	**100.0%**	**100.0%**	**100.0%**	**1,200**
	20	**32%**	**32%**	**23%**	**35,000**

Table 1: accuracy of three different tasks under one model. *n* represent length of target sequence. Asc, Dec, Max indicates ascend, descend, largest value.

From above table we can see that for short length of sequence, it can learn to performance multiple task quickly. For example, when sequence length is 8, the total possible combination is 40,320; when sequence length is 20, the total possible combination is 2.4e19, which is impossible to brute force, or remember all possible data by the model.
And as sequence become longer, it can take longer time and more steps to training, and accuracy is not so high. However, 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%)

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 input 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 may explore more 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.

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] Sepp Hochreiter and J¨urgen Schmidhuber. Long short-term memory. Neural

computation, 9(8):1735–1780, 1997.

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

[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