Pointer Networks for Deep Learning

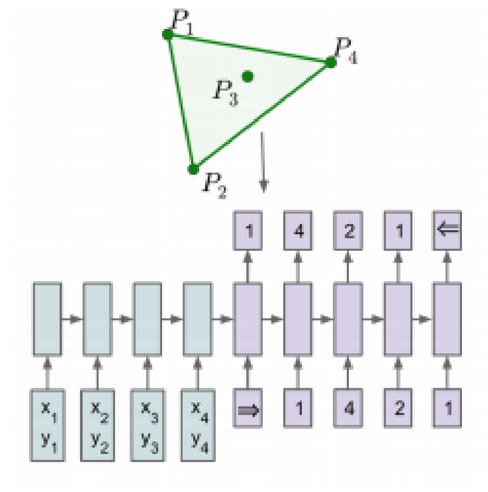
This post talks about **“Pointer Networks”** by Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. This work proposes a neural architecture to generate a variable-sized output sequence, which is a sequence of tokens/indexes of an input sequence. Since the length of the output sequence depends upon the size of the input sequence, it cannot be addressed by RNN based sequence-to-sequence models and Neural Turing machines. Pointer generator networks are applied to solve various combinatorial optimization and combinatorial search problems such as famous planar Travelling Salesman Problem (TSP), Delaunay Triangulation, Convex hull problem, and sorting variable lengths sequences. Pointer networks are also now being applied in text summarization problems to extract sentences from the documents, as mentioned in **“Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting”** by Yen-Chun Chen and Mohit Bansal. These networks generalize well for the sequence lengths beyond which the network was trained on.

Pointer networks can be said to be derived from the attention mechanism by **Bahdanau et al. 2015**. To understand Pointer nets, let us first understand sequence to sequence models, attention-based models seq-to-seq models, and then finally Pointer Networks.

RNN based sequence-to-sequence models:

In sequence-to-sequence models, we use two RNNs (LSTM/GRU) one encoder to encode the input sequence and other for the decoder to generate an output sequence. Consider an example of a convex hull with four points P1, P2, P3, and P4, as in the figure below.

In this figure, blue boxes with input sequence (*Xi, Yi)* represent encoder RNN, and purple boxes represent decoder RNN. Encoder's last hidden state output and " ➔" start token are fed into the decoder model for the first time step. Then for next time steps onwards, the output from the previous time step along with the hidden state of the last step time is fed to produce an output of the current time step. In this case output of time-step T0, which is ["1"], is supplied as input to the next time step to produce the output ["4"]. As we can see from the figure output sequence to complete convex hull will be ["1", "4", "2", "1"].



(a) Sequence-to-Sequence

{Figure1}: Encoder (Blue), Decoder (Purple) based sequence to sequence model for convex hull problem with four points

In the above equation Pi={P1, P2….Pn} are the sequence of "n" vectors and Ci={C1, C2…Cn} is the sequence of indices from 1 to n each. In the above figure1, "n" will be 4.

As in equation1, RNN (LSTM/GRU) can be used to model the conditional probability function. The RNN is fed Pi at each time step "i" until the end of the sequence is reached, which is marked by "←" end token.

$$p(\mathcal{C}^{\mathcal{P}}|\mathcal{P};\theta) = \prod_{i=1}^{m(\mathcal{P})} p_\theta(C_i|C_1,\ldots,C_{i-1},\mathcal{P};\theta).$$

{Equation 1}: Conditional probability equation of Parametric model (RNN)

In this type of sequential models, we need to train separate models for different values of "n" length of the sequence.

Attention Networks

Vanilla seq-to-seq model presented above generates the output sequence using a fixed representation of the input sequence by taking the hidden state of the last time-step. The fixed description limits the amount of information and computation that can flow through the generative decoder RNN model. Hence, to solve this problem, attention network was proposed by Bahdanau et al. 2015. In attention based sequential models, a context vector is formed at every time step of the decoder by giving weights to the input tokens. This context vector is calculated by multiplying attention weights to hidden state representations of each input tokens and summing them up. There are several ways to calculate attention weights, like training a neural network to compute these weights in parallel to sequence-to-sequence model, dot product, and scaled dot products.

Let us denote encoder hidden states as (e1, e2, e3 …, en) and decoder hidden states as (d1, d2, d3 …., dn). The context vector for time-step "i" is calculated as shown in below equation 2.

Here softmax function normalizes the vector "u" over the input sequence of length "n" to be the attention weights of the input sequence. Attention weights are multiplied by each encoder hidden states "e" and summed to form context vector.

$$\begin{aligned} u^i_j &= v^T \tanh(W_1 e_j + W_2 d_i) & j \in (1, \dots, n) \\ a^i_j &= \operatorname{softmax}(u^i_j) & j \in (1, \dots, n) \\ d'_i &= \sum_{j=1}^{n} a^i_j e_j & \end{aligned}$$

{Equation2}: Context vector calculation

More details about the attention model can be checked from this blog post.

Attention Networks

They have been recent developments in the field of NLP, Machine Translation and most of the State Of The Art (SOTA)…

towardsdatascience.com

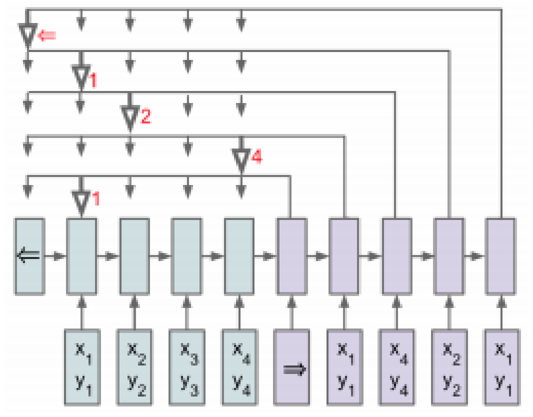
This model performs significantly better than vanilla sequence-to-sequence models on most of the sequential models. But it does not apply to problems where the output dictionary size depends on the input.

Pointer Network

The Pointer network can be thought of as a simple extension (instead of a reduction) of the attention model.

In each decoder time-step, the generating network produces a vector that modulates content-based attention weights over inputs. These weights are calculated by taking softmax operation with dictionary size equal to that of the length of the input sequence.

In the pointer network, these attention weights/masks are not further used to calculate the context vector for the next time step. These weights are considered as pointers to the input sequence. Input time-step having the highest weight is considered as output for that decoder time-step.



(b) Ptr-Net

{Figure2}: Pointer network solution for convex hull problem in Figure1.

$$\begin{aligned} u^i_j &= v^T \tanh(W_1 e_j + W_2 d_i) \quad j \in (1, \ldots, n) \\ p(C_i | C_1, \ldots, C_{i-1}, \mathcal{P}) &= \text{softmax}(u^i) \end{aligned}$$

{Equation3}: Pointer calculation

From Equation3, it can be seen that softmax operation over “u” is not further used to calculate context vector to feed as information to the current decoder step. The output of softmax operation points to the input token having the maximum value.

Consider the output of the first step of the decoder step is “1” as in Figure2. Then for the next time-step corresponding input token representation of input [*X1, Y1*] along with the decoder hidden state representation of the previous time-step is fed to the network to calculate hidden state representation of current time step. The output of the current step is “4”, hence [*X4, Y4*] goes to the input for the next step.

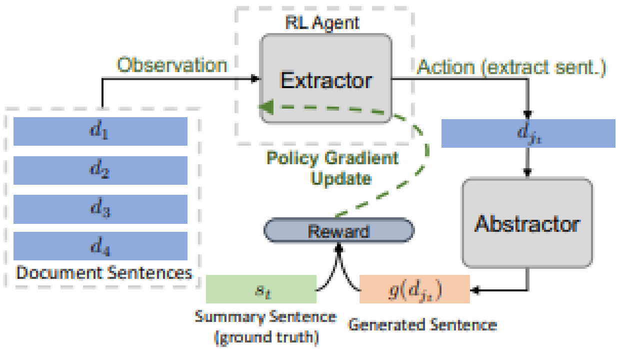
It should be understood that a simple RNN sequence-to-sequence model could have solved this by training to point at the input targets indexes directly. However, at inference, this solution does not respect the constraint that the outputs map back to the input indices exactly. Without the restrictions, the predictions are bound to become blurry over more extended sequences.

Applications of Pointer Network

1) Text Summarization

It has been seen from experiments, and research papers that are combining both attractive and extractive methods give better summary results than trained using only one method.

Figure3 shows the architecture diagram of the text summarization algorithm by combining the extraction agent and an abstraction. The Pointer network is used by the extractor to extract a sequence of unique sentences from a complete set of sentences in the document.

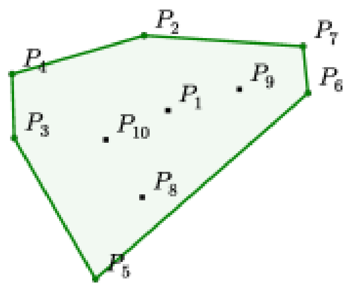


[Figure3]: Text summarization using both extraction and abstraction. (Paper cited in the end.)

2) Convex hull Problem

To find the convex hull of a finite number of points is a well-understood task in computational geometry, and there are several exact solutions available. To solve this problem using an entirely data-driven approach, the pointer network is experimentally found to give better results than vanilla RNN models.

In Figure 4, the sequence of points P[2, 4, 3, 5, 6, 7, 2] represents the boundary of the convex hull. The pointer network is trained by taking input as P[1,2,3....10] and output as a sequence Cp[2,4,3,5,6,7,2]. Since outputs point back to the indices of the input sequence, the pointer network model provides better results than other neural network models.



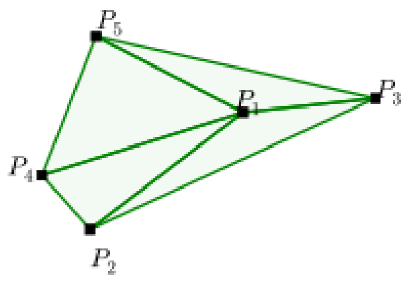
(a) Input $\mathcal{P} = \{P_1, \ldots, P_{10}\}$, and the output sequence $\mathcal{C}^{\mathcal{P}} = \{\Rightarrow, 2, 4, 3, 5, 6, 7, 2, \Leftarrow\}$ representing its convex hull.

[Figure4]: Convex Hull

3) Delaunay Triangulation

A Delaunay triangulation for a set 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 P in its interior.

In the above Figure5 each set of three points [(1,2,4),(1,4,5),(1,3,5),(1,2,3)] represent set of triangulation of the point set P1. Order of the sequence does not matter here; it is just written lexicographically. The pointer network can be trained here as we know the input and output sequence of indices.

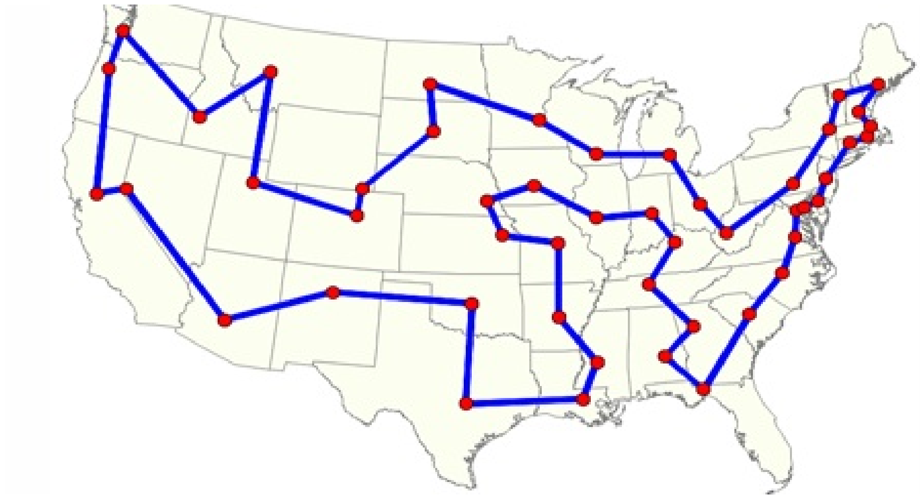


(b) Input $\mathcal{P} = \{P_1, \ldots, P_5\}$, and the output $\mathcal{C}^{\mathcal{P}} = \{\Rightarrow, (1,2,4), (1,4,5), (1,3,5), (1,2,3), \Leftarrow\}$ representing its Delaunay Triangulation.

[Figure5]: Delaunay Triangulation

4) Travelling Salesman Problem (TSP)

TSP arises in many areas of theoretical computer science and is a critical algorithm used for microchip design or DNA sequencing. The trivial TSP problem is to find the shortest possible route that visits each city exactly once and returns to the starting point. It is assumed that the distance between the two cities is the same in each opposite direction.



[Figure6]: Travelling salesman problem

The input-output pairs have a similar format as in the convex hull problem. There are “n” different cities or points in a plane, and we have to travel to each town in the minimum time. The input sequence will be jumbled “n” points without any order, and the output sequence will be an ordered sequence of the same points that represent traveling in minimum time. A pointer network can be trained by taking inputs and outputs of different values of “n.” It is also found that the pointer network generalized well for even those values of “n” it was not trained on.

Citations

Pointer Networks by Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly

Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting by Yen-Chun Chen, and Mohit Bansal

Useful Links

devsisters/pointer-network-tensorflow

TensorFlow implementation of Pointer Networks. Support multithreaded data pipelines to reduce I/O latency. To train a…

github.com

shirgur/PointerNet

Pytorch implementation of Pointer Network. Contribute to shirgur/PointerNet development by creating an account on…

github.com

ChenRocks/fast_abs_rl

This repository contains the code for our ACL 2018 paper: Fast Abstractive Summarization with Reinforce-Selected…

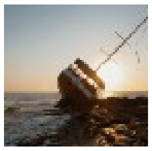
github.com

Public domain.

More From Medium

You Don’t Have to Use Docker Anymore

Martin Heinz in Towards Data Science

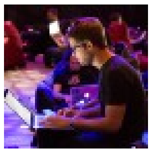


Python is Slowly Losing its Charm

Jason Dsouza in Towards Data Science

Go Programming Language for Artificial Intelligence and Data Science of the 20s

Dasaradh S K in Towards Data Science



How to spot a data charlatan

Cassie Kozyrkov in Towards Data Science

10 Awesome Python 3.9 Features

Farhad Malik in Towards Data Science

10 Underrated Python Skills

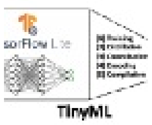
Nicole Janeway Bills in Towards Data Science

Python 3.9

James Calam in Towards Data Science

Tiny Machine Learning: The Next AI Revolution

Matthew Stewart, PhD Researcher in Towards Data Science



Learn more.

Medium is an open platform where 170 million readers come to find insightful and