Hard Pointer Networks

Lawrence Chan chanlaw@wharton.upenn.edu

João Sedoc joao@seas.upenn.edu

Fall 2016

Abstract

Previous research showed that

1 Introduction

The main change between a hard pointer network and a pointer network [1] is the fact that we use an argmax instead of a softmax for attention. So far for sorting this seems to be better.

2 Model

2.1 Sequence-to-sequence Models

In a sequence-to-sequence model [2], two RNNs (generally LSTMs) are used to model the conditional probability of an output sequence $\mathcal{C}^{\mathcal{P}} = \{C_1, ..., C_k\}$ given an input sequence $\mathcal{P} = \{P_1, ..., P_n\}$ using the probability chain rule:

$$\mathbb{P}(\mathcal{C}^{\mathcal{P}}|\mathcal{P}) = \prod_{i=1}^{k} P(C_i|C_1, ...C_{i-1}, \mathcal{P})$$

One RNN (known as the encoder) encodes the input sequence $\mathcal{P}$, while another RNN (known as the decoder or generator) uses the encoding to generate the output sequence $\mathcal{C}$. That is, the encoder learns to create a fixed-dimensional representation v of a sequence $\mathcal{P}$, which the decoder uses to generate the conditional probabilities:

$$v = \text{Encoder}(\mathcal{P})$$
$$\mathbb{P}(C_i|C_1, ..., C_{i-1}, \mathcal{P}) = \text{Decoder}(C_1, ..., C_{i-1}, v)$$

2.2 Content Based Input Attention

In the simple sequence-to-sequence model, the information that flows between the encoder and decoder networks is limited to the size hidden state of the

encoding RNN. In contrast, the attention mechanism of [3] allows the network to access any the hidden states $e_1, ...e_n$ of the encoding network. Let d_i be the hidden state of the decoding network at time i. We compute the attention vector at time i, a_i as follows:

$$u_i^j = v \cdot \tan(W_1 e_j + W_2 d_i)$$
$$a_i = \text{softmax}(u_i)$$

where u_i^j is the jth entry of the u_i vector. Here, v, W_1, W_2 are learned parameters. We then treat this as an "attention mask" over the encoded states:

$$d_i' = \sum_j a_i^j e_j$$

Finally, we concatenate d_i' to d_i and use this concatenated vector as the hidden state from which to make predictions and to pass to the next time step.

2.3 Pointer Network

The pointer network of [1] is a simplification of the attention-augmented network. Instead of using attention to blend hidden states of the encoding network and concatenating this result to the hidden state of the decoding network, we treat the attention vector as the desired conditional probabilities for the output sequence. The attention vector is then used to blend elements of the input sequence and this result is treated as input of the decoder in the following timestep. That is:

$$\begin{aligned} u_i^j &= v \cdot \tan(W_1 e_j + W_2 d_i) \\ a_i &= \text{softmax}(u_i) \\ \mathbb{P}(C_i | C_1, ..., C_{i-1}, \mathcal{P}) &= a_i \end{aligned}$$

The input to the decoder at the next timestep is

$$\text{Input}_{i+1} = \sum_j a_i^j P_j.$$

As [1] notes, this means that at inference time this solution does not respect the contraint present at training time that the inputs to the decoder correspond to "pure" elements of the input sequence.

2.4 Hard Pointer Network

We now describe a further extension (or rather further simplification) to the pointer network framework. Instead of feeding the attention-weighted input sequence to the decoding network, we take the hardmax over attention (replacing

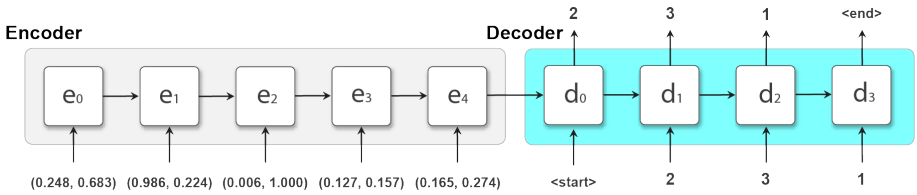

Figure 1: A sequence-to-sequence model [2] solving a simple convex hull problem

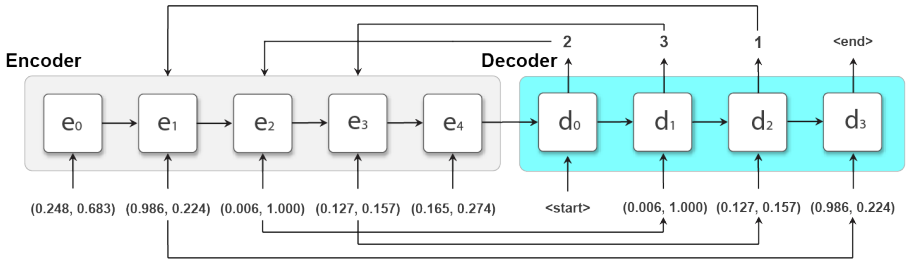

Figure 2: A pointer network [1] solving the same convex hull problem

the maximum element with a 1 and all the other elements with a 0) and use this to select an element of the input sequence to feed to the decoder. As before,

$$\begin{aligned} u_i^j &= v \cdot \tanh(W_1 e_j + W_2 d_i) \\ a_i &= \text{softmax}(u_i) \\ \mathbb{P}(C_i | C_1, ..., C_{i-1}, \mathcal{P}) &= a_i \end{aligned}$$

However, the input to the decoding network is simply the element of the input sequence corresponding to the maximum element of a_i:

$$\begin{aligned} k &= \text{argmax}_j \ a_i^j \\ \text{Input}_{i+1} &= P_k \end{aligned}$$

This means that the inputs to a hard pointer network are the same during inference as in training.

2.5 Multi-Pointer Network

Finally, we consider an alternative simplification to the pointer network framework. Instead of feeding the attention-weighted input sequence to the decoding network, we apply a threshold mask on the attention vector (replacing every element of the attention vector with value less than the threshold value with 0 and e very element with value greater than the threshold value with 1), then take the average of the elements indicated by the resulting vector. That is, we

compute the vector m_i^j:

$$m_i^j = \begin{cases} 1 & a_i^j \geq \text{threshold} \\ 0 & a_i^j < \text{threshold} \end{cases}$$

The input to the decoding network is:

$$\text{Input}_{i+1} = \frac{1}{||m_i||} \sum_j m_i^j P_j.$$

3 Experimental Results

We evaluated our networks on the sorting and convex hull problems. [1]

3.1 Sorting

In the sorting problem, the input is a sequence of real numbers between 0 and 1 and the desired output is the indices of the real numbers in sorted order. For example, an input sequence may be $\{0.7, 0.649, 0.921, 0.01, 0.52\}$, and the desired output for this sequence is $\{4, 5, 2, 1, 3, \langle\text{eos}\rangle\}$ (corresponding to the sorted sequence $\{0.01, 0.52, 0.649, 0.7, 0.921\}$). Note that in this problem, not counting the end of sequence character, the output sequence and input sequence is always of equal length.

Method	**LSTM Units**	**N**	**# of examples**	**Accuracy**
Ptr-Net	32	5	1000k	0.9133
Hard-Ptr-Net	32	5	1000k	**0.9383**
Multi-Ptr-Net	32	5	1000k	0.9277
Ptr-Net	32	10	1000k	0.3279
Hard-Ptr-Net	32	10	1000k	**0.5763**
Multi-Ptr-Net	32	10	1000k	0.4898
Ptr-Net	64	10	1000k	0.3537
Hard-Ptr-Net	64	10	1000k	0.6025
Multi-Ptr-Net	64	10	1000k	**0.6053**
Ptr-Net	256	10	1000k	0.6081
Hard-Ptr-Net	256	10	1000k	0.7336
Mutli-Ptr-Net	256	10	1000k	**0.7870**

Table 1: Comparison of different networks on sorting. N is the number of elements to sort.

We sampled our points from the uniform distirbution on $[0, 1]$. We trained each of our networks by using the Adam optimizer [4] on one pass of 1M data

[1] Our code is available at `github.com/chanlaw/pointer-networks`.

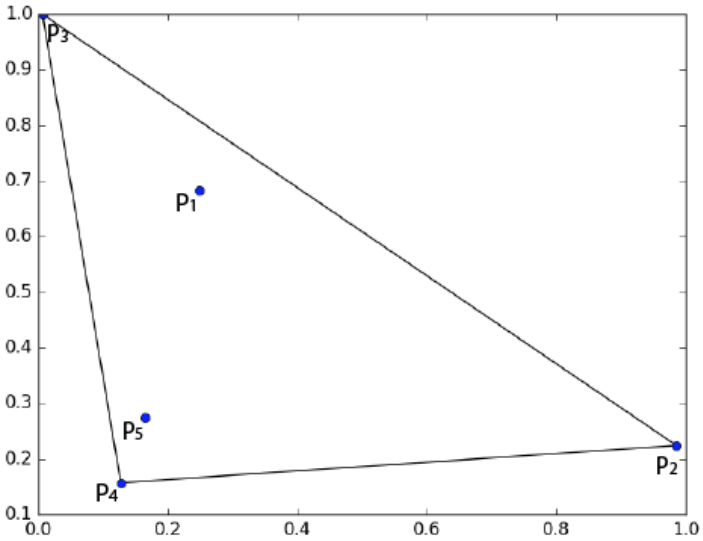

Figure 3: The convex hull of $\{(0.248, 0.683), (0.986, 0.224), (0.006, 1.000), (0.127, 0.157), (0.165, 0.274)\}$

points to minimize the batch log loss, with a mini-batch size of 128 and learning rate of 0.001. We report our results in Table 1.

3.2 Planar Convex Hull

In the convex hull problem, the input is a sequence of points on the coordinate plane, and the output is a sequence of indices corresponding to the vertices on the convex hull (the smallest closed convex set that contains the points). For example, an input sequence could be

$$\{(0.248, 0.683), (0.986, 0.224), (0.006, 1.000), (0.127, 0.157), (0.165, 0.274)\}$$

and a correct output sequence could be $\{2, 3, 1\}$, as shown in Figure 3.

We sampled our points uniformly from $[0, 1] \times [0, 1]$. As with the sorting problem, we used the Adam optimizer to minimize the batch log loss, with a mini-batch size of 128 and learning rate of 0.001. During training, our target sequences consist of points in counter clockwise order, starting from the point with the smallest x value. Interestingly, we found that the starting point of our sequence at matters greatly - for example, starting the sequence at the point on the convex hull with the smallest index led to an almost 10% decrease in accuracy after 100K points for each network, while starting at a random point lead to an almost 15% decrease for each network. We report our results in Table 2 and Figure 4.

Method	LSTM Units	N	Steps	Accuracy
Ptr-Net	512	50	10K	0.258
Hard-Ptr-Net	512	50	10K	**0.342**
Multi-Ptr-Net	512	50	10K	0.290
Ptr-Net	512	50	100K	0.546
Hard-Ptr-Net	512	50	100K	**0.604**
Multi-Ptr-Net	512	50	100K	0.553
Prt-Net	512	50	1000K	0.722
Hard-Ptr-Net	512	50	1000K	**0.743**
Multi-Ptr-Net	512	50	1000K	0.694

Table 2: Comparison of different networks on the convex hull task. Accuracies averaged over 12.8k test points.

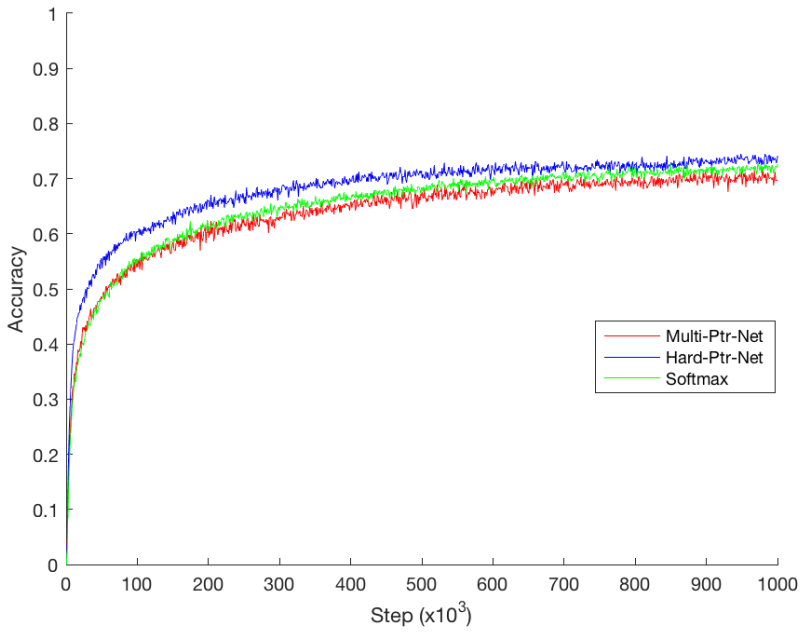

Figure 4: The test accuracy of the three networks as a function of number of steps trained on the convex hull task reported in Table 2.

4 Discussion

5 Conclusion

References

[1] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[2] 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.

[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] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.