



The Implementation of a Pointer Network Model for Traveling Salesman Problem on a Xilinx PYNQ Board

Shenshen Gu¹(✉), Tao Hao¹, and Shaofu Yang²

¹ School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200072, China
gushenshen@shu.edu.cn

² School of Computer Science and Engineering, Southeast University, Nanjing 211189, China

Abstract. In this paper, a pointer network model for traveling salesman problem (TSP) was implemented on a Xilinx PYNQ board which supports Python and Jupyter notebook and is equipped with ZYNQ SOC. We implement a pointer network model for solving TSP with Python and Theano firstly, then train the model on a GPU platform, and eventually deploy the model on a PYNQ board. Unlike traditional neural network implementation, hardware libraries on PYNQ (Overlays) are used to accelerate the pointer network model application. The experimental results show that the pointer network model for TSP can be deployed on the embedded system successfully and achieve good performance.

Keywords: Pointer networks · Traveling salesman problem · Theano PYNQ · FPGA

1 Introduction

In recent years, deep neural networks (DNNs) are widely used in many artificial intelligence applications, particularly tasks involving computer vision [1], speech recognition [2], and robotics [3]. Currently, DNN models usually require a Graphics Processing Unit (GPU) to accelerate computation, so they mostly have been developed and applied on large machines with powerful computation capacity. With increasingly need of DNN models deployed in mobile devices, there is a growing concern in deep learning area about how to deploy powerful and cost-effective DNN models in an embedded system. DNN on embedded system projects have been launched by some researchers [4]. A TensorFlow-on-Raspberry-pi Project was issued by Sam Abrahamms. In addition, a Binary Neural Network project that converts the floating-point parameters into binary values on an FPGA [5] has been published by a Xilinx research group. Deep learning models can be trained off-line and then implemented onto embedded system, so that the system only needs to focus on improving the throughput of forward propagation.

Meanwhile, DNNs have made remarkable achievements in solving combinatorial optimization problems. For instance, Vinyals solves the traveling salesman problem (TSP) with recurrent neural networks (RNNs) [6]. TSP is a classical example of combinatorial optimization arising in many areas of theoretical computer science. It can be described as follows. Given a set of city coordinates, one needs to search the space of permutations to find an optimal sequence of nodes with minimal total tour length. TSP plays an important role in microchip design, DNA sequencing, and robot path planning. In [6], a new architecture termed as Pointer network is proposed for solving large scale TSPs. In this network, attention mechanism is used as a pointer to select a position from the input sequence as an output symbol. The pointer network is shown as a simple and effective model for solving TSPs. Therefore, it is meaningful to deploy the pointer network in an embedded system for various wearable applications.

FPGA is very suitable for parallel computing, and have been widely used to accelerate the neural network and machine learning algorithm [7]. Nowadays, most embedded devices are composed of ARM-based processor and the hardware programmability of an FPGA, such as Xilinx Zedboard, ZYBO, and so on. Python has more advantages in scientific computation and data processing than other programming languages. Due to the fact that a lot of software libraries in Python exist, which make data sampling, analysis, and processing very convenient. Python has consistently been ranked the top of Lists of Programming languages for deep learning. Motivated by the release of PYNQ from Xilinx which aids in the interfacing with custom hardware in the FPGA fabric and providing many useful utilities, such as downloading bitstreams from within the application, we consider how to use Python in an FPGA development environment.

Currently, many popular open-source deep learning framework programming tools such as TensorFlow, Theano, Caffe all support Linux platform, and all support Python interface. Comparing these neural network framework tools, we found that Theano is the most suitable for PYNQ development board. Because Theano can be installed on PYNQ easily and runs well under a 32bit Linux Operation System (OS). What's more, the support of Theano for customized layer is very high.

The rest of this paper is organized as follows. In Sect. 2, recurrent neural network and architecture of pointer network are introduced briefly. Section 3 introduces how to implement the pointer network model for TSP based on Theano and FPGA accelerator Overlay design in detail. Next, in Sect. 4, the training process of the model and performance on PYNQ board are shown. Finally, a conclusion is drawn to summarize this work in Sect. 5.

2 Recurrent Neural Network and Pointer Network Model

In this section, we first introduce RNNs, especially Long Short Term Memory (LSTM) which is the basic cell of pointer networks. And then, the architecture of the pointer network model will be described.

2.1 Recurrent Neural Network

RNNs are becoming an increasingly popular way to process and predict sequences of data. RNNs have shown excellent performance in problems such as speech recognition, machine translation and scene analysis. RNNs are recurrent because they perform the same computations for all the elements of a sequence of inputs, and the output of each element depends on not only the current input but also all the previous computations.

As a special RNN architecture, LSTM implements a learned memory controller for avoiding vanishing or exploding gradients [8]. LSTM is a network that consists of cells (LSTM blocks) linked to each other. Each LSTM block contains three types of gate: Input gate, Output gate and Forget gate, respectively, which perform the functions of writing, reading, and resetting on the cell memory. There are some variations on the LSTM architecture and all those variations have similar performance as shown in [9]. This is the vanilla LSTM [10], which can be formulated as follows:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \quad (1)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \quad (2)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad (3)$$

$$\tilde{c}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (4)$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \quad (5)$$

$$h_t = o_t * \tanh(c_t) \quad (6)$$

where i , f and o represent input, forget, and output gate respectively, x is the input vector of the layer, W is the model parameters, c is memory cell activation, \tilde{c} is the candidate memory cell gate, h is the layer output vector, σ is the logistic sigmoid function, and $*$ is element wise multiplication. And $t - 1$ represents results from the previous time step.

2.2 Architecture of Pointer Networks

The architecture of pointer networks in [6] is applied to solve TSP. Given an input sequence, this type of deep neural architecture (see Fig. 1) combines the popular sequence-to-sequence learning framework [11] with a modified Attention Mechanism [12] to learn the conditional probability of an output whose values correspond to positions in an input sequence.

To solve the problem that the encoder output dictionary size depends on the length of the input sequence, the pointer network adjusts the standard attention mechanism to create pointers to elements in the input sequence. The following modification to the attention model was proposed:

$$u_j^i = v^T \tanh(W_1 e_j + W_2 d_i) \quad j \in \{1, 2, \dots, n\} \quad (7)$$

$$p(C_i | C_1, \dots, C_{i-1}, P) = \text{softmax}(u^i) \quad (8)$$

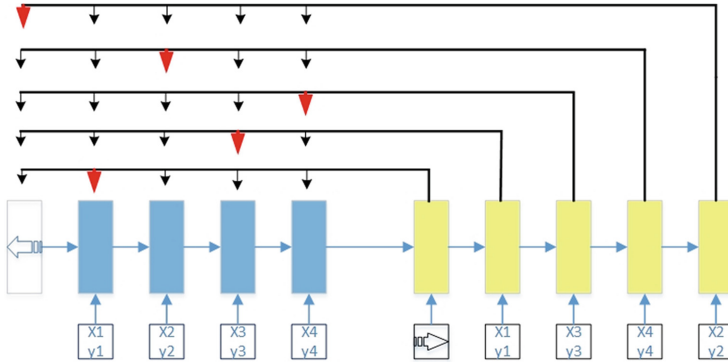


Fig. 1. Architecture of the pointer network (encoder in blue, decoder in yellow) (Color figure online)

where softmax normalizes the vector u^i (of length n) to be an output distribution over the dictionary of inputs, and v , W_1 , W_2 are learnable parameters. Note that unlike the standard attention mechanism, the pointer network model does not use the encoder states to propagate extra information to the decoder, but instead uses u_j^i as pointers to the input sequence elements [13].

Figure 1 illustrates the architecture of the pointer network, which mainly consists of encoder network and decoder network. An encoder converts the input sequence to a code (blue) that is fed to a decoder (yellow). The input/output pairs (P, C^P) for TSP are illustrated in detail. The input sequence $P = \{P_1, \dots, P_n\}$ is the cartesian coordinates representing the cities. $C^P = \{C_1, \dots, C_n\}$ is a permutation of integers from 1 to n representing the optimal path.

3 Implementation

In this section, the pointer network model for TSP design, training, and deployment methods and procedures will be concretely described. The training of the neural network will consume much computing resource and time, so we use a GPU to train the network, and then deploy the model on PYNQ board.

3.1 Implementation of Pointer Network Model for TSP Based on Theano

Figure 2 shows the procedure of implementing a pointer network model. The first step is to generate TSP training dataset, preprocess the data and then load it. We set the maximum and minimum number of nodes in a set of data, then generate the random number between maximum and minimum as the number of nodes, and then generate randomly plane coordinates in the $[0, 1] \times [0, 1]$ square. Without loss of generality, in the training dataset, we always start from the first

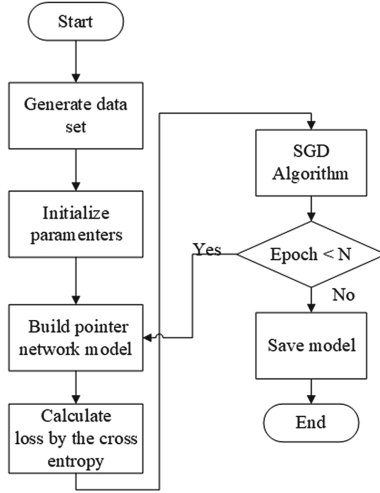


Fig. 2. Scheme of implementing a pointer network model

city in order to keep consistency. For small scale TSP, dynamic programming algorithm are implemented to obtain exact solutions. For large scale TSP, due to the importance of TSP, many good and efficient algorithms have provided reasonable approximate solutions, so we can use benchmark for TSP to create samples. The training data is saved as `tsp.pkl.gz`.

The second step is to make parameter initialization module. The initial value for the weights and biases of the neural network need to be configured. The next step is to build the pointer network model. We first define the LSTM cell function module, and then according to the architecture of pointer network in Sect. 2.2, make an encoder network and a decoder network with LSTM cell. The softmax function, used in the output layer, is a kind of normalization function that can convert a vector into a probability distribution form and give every element a probability, whose definition is described as follows.

$$\sigma(z)_j = \frac{\exp(z_j)}{\sum_{k=1}^K \exp(z_k)} \quad j \in 1, \dots, K \quad (9)$$

The cross-entropy is more suitable for training RNN, which is used as loss function to calculate the gap between the prediction and the actual values. The definition of the cross-entropy function is described as follows.

$$H(p, q) = - \sum_{i=1}^n p(x) \log(q(x)) \quad (10)$$

Stochastic Gradient Descent (SGD) algorithm is used to update the weights of the neural network according to loss function. Above functions can be implemented conveniently with Python and Theano. The final step is to train the

model and save the trained pointer network model. The trained model will be saved in a npz file, which saves several arrays into a single file in uncompressed format. Then the trained model can be used to predict results.

3.2 FPGA Accelerator Overlay Design

PYNQ provides an Overlay framework to support interfacing with the board’s IO. However, any custom logic must be created and integrated by the developer. A Vivado project for a Zynq design consists of the PL design and the PS configuration settings. For PS configuration, it covers settings for system clocks and the clocks used in the PL. And, the PS settings in their Vivado project should be ensured to match the PYNQ image settings. The PL clock configuration are set as Table 1.

Table 1. PL clock configuration

PL Clock	FCL_CLK0	FCL_CLK1	FCL_CLK2	FCL_CLK3
Frequency	100.00 MHz	142.86 MHz	200.00 MHz	166.67 MHz

The schedule of creating a PYNQ overlay is described as follows. Hardware accelerators should be first designed and implemented to an IP through Vivado HLS, a high-level synthesis tool. Based on a base design which includes most of the peripheral interface overlay provided by the PYNQ project, a Vivado project is created and IPs are added to a block design and the design is synthesized, implemented, and a bitstream is generated, a tcl file of the block design is exported.

The communication between the PL and PS depends on an AXI interface. An IP with AXI interface can be easily created in Vivado Design Suite. In addition, PYNQ includes the MMIO Python class to simplify communication between the Zynq PS and PL. Once the overlay has been created, and the memory map can be known through Address Editor in Vivado, the MMIO can be used to access memory mapped locations in the PL. Matrix multiplication operations are the most frequent operation in the neural network, so a matrix multiplication accelerator is designed in this paper. Figure 3 shows the connection between the matrix multiplication and the Zynq processor.

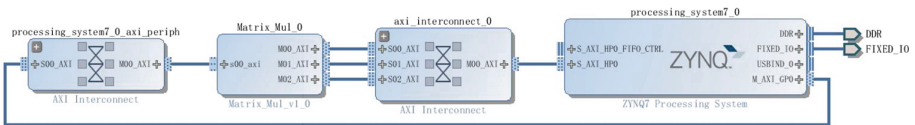


Fig. 3. The connection between the matrix multiplication and the Zynq processor

4 Experimental Results

To test the effectiveness of the implementation, a pointer network model for TSP was trained by supervised learning. The weights of neural networks were updated by SGD algorithm with a learning rate of 0.01. All the parameters were initialized obeying random distribution in $[-0.08, 0.08]$.

We obtained the loss values of 1000 epochs in the training process. As is shown in Fig. 4, the loss values are gradually reducing with the increase of training epochs, which means the training algorithms works well.

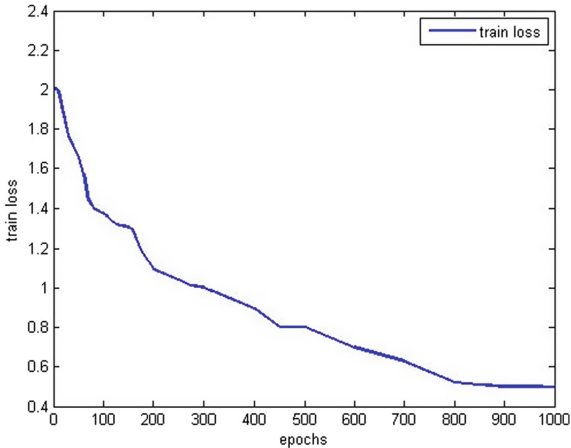


Fig. 4. The loss values of 1000 epochs

The PYNQ board should be configured as required. After booting up the linux system in the PYNQ board, we could view and run the notebooks interactively through our browser. The final size of the trained model is about 5 MB, which needed to be copied to the board file system, and loaded in Jupyter notebook. Then the model on PYNQ board can be used to predict the result of the TSP.

After installing the essential package, the trained model was deployed successfully on the PYNQ board. In our experiment, the PYNQ's CPU was clocked at frequency of 665 MHz. For different scale of TSP, the inference time of running pointer network models on the PC (equipped with Intel i5-2450M CPU @ 2GHz and 4 GB RAM), PYNQ board without Overlay and PYNQ board with Overlay respectively is shown in Table 2. It is found that the performance of the pointer network model deployed on PYNQ board is nearly comparable to that on conventional PC. In addition, Overlay can promote the performance of PYNQ board to some degree.

Table 2. Performance of the pointer network model

Number of nodes	10	20	30	40	50
Inference time on PC (sec.)	0.21	0.35	0.61	0.72	0.82
Inference time on PYNQ without Overlay (sec.)	0.29	0.44	0.71	0.83	0.96
Inference time on PYNQ with Overlay (sec.)	0.24	0.38	0.65	0.76	0.85

The core chip of PYNQ is a ZYNQ SOC XC7Z020-1CLG400C, which has plentiful programmable logic resource. The FPGA resource utilization is shown in Table 3. In our experiment, BRAM and LUT are mainly used. BRAM resource is used to restore the weights. LUT resource is consumed to register and routing.

Table 3. FPGA resource utilization

Resource	Utilization	Available	Utilization%
LUT	32414	53200	60.93
LUTRAM	2619	17400	15.05
FF	39252	106400	36.89
BRAM	74.50	140	53.21
DSP	11	220	5.0
IO	96	125	76.80
BUFG	6	32	18.75
MMCM	2	4	50.00

5 Conclusions

In this paper, a pointer network model for TSP is implemented with python and Theano and trained on a GPU platform. Furthermore, the trained model is deployed on PYNQ board through Jupyter notebook. As deep neural networks have delivered state-of-the-art performances in the fields of speech recognition, machine translation and machine vision, the deployment of well-trained neural networks to embedded equipment is a trend of future development. FPGA is a very promising accelerator for deep neural network due to its strong parallel-process capability, reconfigurability and low power consumption. Experimental results show that the implementation of pointer network for TSP is successful and the performance is promising which suggests that it can be applied to various models and fields.

Acknowledgments. The work described in the paper was supported by the National Science Foundation of China under Grant 61503233.

References

1. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**(7553), 436–444 (2015). Nature Publishing Group
2. Li, S., Wu, C., Li, H., Li, B., Wang, Y., Qiu, Q.: FPGA acceleration of recurrent neural network based language model. In: *Proceedings of the IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 111–118. IEEE Press (2015)
3. Lenz, I., Lee, H., Saxena, A.: Deep learning for detecting robotic grasps. *Int. J. Robot. Res.* **34**(4–5), 705–724 (2015). SAGE Publications Sage UK, London
4. Hao, Y., Quigley, S.: The implementation of a deep recurrent neural network language model on a Xilinx FPGA. *arXiv preprint [arXiv:1710.10296](https://arxiv.org/abs/1710.10296)* (2017)
5. Umuroglu, Y., Fraser, N.J., Gambardella, G., Blott, M., Leong, P., Jahre, M., Vissers, K.: Finn: a framework for fast, scalable binarized neural network inference. In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 65–74 (2017)
6. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. In: *Advances in Neural Information Processing Systems*, pp. 2692–2700 (2015)
7. Guan, Y., Yuan, Z., Sun, G., Cong, J.: FPGA-based accelerator for long short-term memory recurrent neural networks. In: *Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 629–634. IEEE Press (2017)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**(8), 1735–1780 (1997)
9. Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J.: LSTM: a search space odyssey. *IEEE Trans. Neural Netw. Learn. Syst.* **28**(10), 2222–2232 (2017)
10. Chang, A.X.M., Martini, B., Culurciello, E.: Recurrent neural networks hardware implementation on FPGA. *arXiv preprint [arXiv:1511.05552](https://arxiv.org/abs/1511.05552)* (2015)
11. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: *Advances in Neural Information Processing Systems*, pp. 3104–3112 (2014)
12. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. *arXiv preprint [arXiv:1409.0473](https://arxiv.org/abs/1409.0473)* (2014)
13. Milan, A., Rezafighi, S.H., Garg, R., Dick, A.R., Reid, I.D.: Data-driven approximations to NP-hard problems. In: *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI 2017)*, pp. 1453–1459 (2017)