Energy-Efficient Hardware for Language Guided Reinforcement Learning

Aidin Shiri
University of Maryland
Baltimore County

Arnab Neelim Mazumder
University of Maryland,
Baltimore County

Bharat Prakash
University of Maryland,
Baltimore County

Nitheesh Kumar Manjunath
University of Maryland,
Baltimore County

Houman Homayoun
University of California,
Davis

Avesta Sasan
George Mason University,
Fairfax

Nicholas R. Waytowich
US Army Research
Laboratory

Tinoosh Mohsenin
University of Maryland,
Baltimore County

ABSTRACT

Reinforcement learning (RL) has shown great performance in solving sequential decision-making problems. While a lot of works have done on processing state information such as images, there has been some effort towards integrating natural language instructions into RL. In this paper, we propose an energy-efficient architecture which is designed to receive both images and text inputs as a step towards designing RL agents that can understand human language and act in real-world environments. Different configurations are proposed to illustrate the trade off between the number of parameters and the model accuracy, and a custom low power hardware is designed and implemented on FPGA based on the best configuration. The hardware designed to be configurable with different parameters such as number of processing elements, so that it can easily balance power and performance. The high throughput configuration achieves 217 frames per second throughput with 1.2 mJ energy consumption per classification on Xilinx Artix-7 FPGA, while the low power configuration consumes less than 139 mW for 30 frames per second classification. Compared to the similar works using FPGA for hardware implementation, our design is more energy efficient and need less energy for generating each output.

KEYWORDS

Reinforcement Learning; Structured Language Instructions; FPGA

ACM Reference Format:
Aidin Shiri, Arnab Neelim Mazumder, Bharat Prakash, Nitheesh Kumar Manjunath, Houman Homayoun, Avesta Sasan, Nicholas R. Waytowich, and Tinoosh Mohsenin. 2020. Energy-Efficient Hardware for Language Guided Reinforcement Learning. In *Proceedings of the Great Lakes Symposium on VLSI 2020(GLSVLSI '20), September 7–9, 2020, Virtual Event, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3386263.3407652

GLSVLSI '20, September 7–9, 2020, Virtual Event, China

ACM ISBN 978-1-4503-7944-1/20/09…$15.00
https://doi.org/10.1145/3386263.3407652

1 INTRODUCTION

Reinforcement Learning (RL) is a goal-oriented paradigm of machine learning in which the agent tries to learn a policy to achieve complex tasks by trial and error. Reinforcement Learning is used for problems that involve sequential decision making where the agent needs to take actions in an environment to maximize cumulative future rewards. In reinforcement learning goals are specified using a reward function [1]. Human feedback can also be used to specify goals as shown in [2] and [3]. It is important to train the agent in a way that it easily abides the human instructions. One scalable way to do this is by using language instructions. Lately, using language to guide reinforcement learning agents has gained a great interest among AI researchers [4], [5], [6],[7]. These methods have shown improvement in making agents able to understand human language. Instructions are processed using a language processing techniques to generate embeddings to feed the agent along with the states. Besides making it easy to specify goals and rewards, language can also be used to convey constraints and improve the safety of reinforcement learning agents [8]. While hardware implementation of deep neural networks has gained a lot of attention in recent years [9–13], most of the previous works in Reinforcement Learning focused on the algorithm and theoretical aspects, and very a few works have considered hardware implementation for RL [14]. In this paper, we propose a hardware friendly architecture for Reinforcement Learning which can interpret language instructions to act in real-world scenarios. This paper makes the following contributions:

- Propose a hardware friendly architecture for Reinforcement Learning which can interpret structured language instructions.
- Optimize the complexity of the proposed architecture to find the best trade-off between accuracy (reward function), model size and computation.
- Propose a scalable and parameterized hardware in Verilog HDL that can receive image inputs from environment and language constraints for deploying on the embedded devices.
- Implement the proposed work with different configurations in terms of processing elements and parallelism on Artix7 FPGA and compare them in terms of power consumption, energy efficiency, latency, and utilized resources.

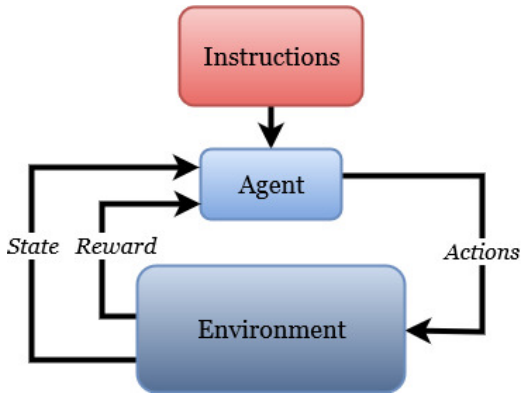


Figure 1: Proposed system with language instructions. In the conventional Reinforcement Learning model, agents interact with the environment through a set of actions and try to learn a certain policy by gaining the maximum possible reward.Meanwhile, adding the language instructions module force the agent to learn to accomplish the dictated goal.

2 BACKGROUND

In Reinforcement Learning, the agent is trained to perform a certain task which usually requires performing a sequence of decisions without a supervisor. The agent only gets a reward based on the completion of the tasks to learn the desired policy without specifying how to accomplish the task. Generally, Reinforcement Learning is modeled through Markov Decision Processes (MDP). MDP is a list of elements (S, A, P, R, γ), which denotes state space, action space, transition function, reward function, and discount factor respectively. The agent tries to interact with the environment through a set of possible moves called actions and the environment take the agent's action and current state and returns a reward and next state. The reward is a feedback from the environment by evaluating the agents performance for completing the task. The policy is the strategy with which the agent maps its state to the action that guarantees the optimum future reward. The Trajectory is a sequence of state actions. In this paper we propose an architecture similar to the conventional Reinforcement Learning structure described above; however,by adding a language module to the design we make the agent capable of performing the instructed task. In other words, the agent learns the policy of performing the desired task instructed by the language module.

3 PROPOSED SYSTEM ARCHITECTURE

A detailed overview of the proposed design and the system architecture will be discussed in this section. Also, we will explain how the RL Agent is designed and trained to accomplish the instructed goals with a structured language module.

As explained in the previous section, in the conventional reinforcement learning model, agents interact with the environment through a set of actions, and try to learn a certain policy by gaining maximum possible reward. In this work, we also added a Language Module that gives certain structured instructions to the agent to accomplish the desired tasks. The proposed system architecture consists of two main parts: the traditional Reinforcement Learning

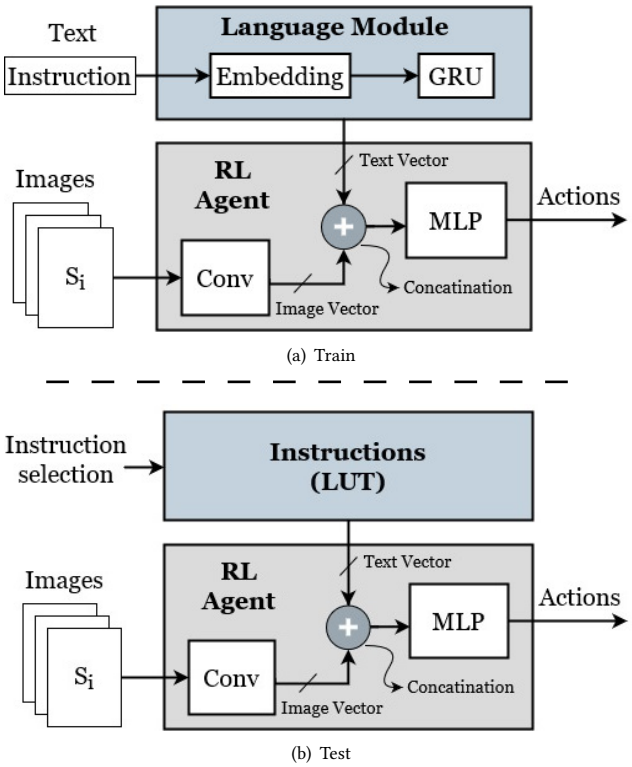


Figure 2: Proposed architecture for training and test. During training, RL Agent learns the policy with the instructions from integrated natural language processor. During test, the hardware-intensive language module is replaced by a lookup table to reduce the power and area and increase the efficiency of design.

Agent and the language module. The language module consists of an embedding and a Gated Recurrent Unit (GRU) which translates text instructions to a vector. The output of the language module is concatenated with the output of the agent and makes a one vector to pass through fully-connected layers. Figure1 illustrates the abstract model of Reinforcement Learning with the addition of an instruction module.

Figure 2 depicts the proposed architecture for the train and test. The RL agent receives an image and an instruction in the form of text and generates action at each step. The image is processed by a convolution neural network to output an image embedding. The language input is processed by an embedding layer followed by a Gated Recurrent Unit (GRU) layer to get a text embedding. These two embeddings are then concatenated and passed through an MLP to get the final action. The model is trained using Proximal Policy Optimization [15]. During the inference, the bulky language module is replaced with a simple lookup table that can feed the saved instruction vectors to the agent.

Figure 3 illustrates the detailed architecture of test . Our design consists of three convolutional layers, followed by two fully connected and an MLP layer. Images captured from the environment are passed into the convolutional layers and important features

Table 1: Network architecture specification used as the baseline (Config3) for the experiments

Layer	Input	Window	# Kernels	Activation
Conv 1	80*50*3	(5,5)	32	ReLU
Conv 2	38*23*32	(5,5)	32	ReLU
Conv 3	17*12*32	(4,4)	32	ReLU
Flatten	7*4*32	-	-	-
Dense 1	896	-	-	ReLU
Dense 2	128+128	-	-	ReLU
MLP	64	-	-	-

are extracted from images using a set of local filters. Once the convolution operation is performed, the image gets flattened into a single vector. Stride of 2x2 is used after each convolution layer instead of max-pooling operation, due to its simpler hardware implementation. During the test, the language module discussed in the previous section is replaced with a simple LUT which keeps the known structured instructions. The output vector of convolution layers is concatenated with the desired instruction vector from instructions LUT and fed into two fully connected layers followed by an MLP to generate the corresponding action. Table 1 shows the network architecture in detail that we use as the baseline design.

4 EXPERIMENTAL RESULTS

In this section, we present the environment setup, RL Agent configurations, and implementation results on FPGA for the corresponding environment. Also, we provide an analysis of implementation results for different metrics such as power consumption, throughput, and energy efficiency.

4.1 Environment Setup

Our experiments are performed on the MiniWorld environment [16]. It is a minimalistic 3D environment for Reinforcement Learning and robotics research. The agent can navigate rooms and manipulate objects using its first-person view as shown in Fig. 4-(a). In our experiments we set up a simple scenario where the agent is in a room and there are 2 boxes, one red and one blue box. The agent is spawned at random locations at each episode and it receives instructions in the form of text as shown in Table 2. Depending on the instruction, the agent has to reach one of the boxes. The agent gets a reward of 1 if it reaches the instructed box to pick, and 0 otherwise.

Table 2: Example instructions used in the environment to guide the RL agent

#	Instructions
1	Go to the red box
2	Reach the blue box
3	Go to the blue box
4	Reach red box

Table 3: Three network configurations with different kernel sizes. The baseline architecture is Config 3.

	Config 1	Config 2	Config 3
# Conv Kernels	8	16	32
# Parameters	49 K	86 K	176 K
# Computations	4 M	10.9 M	33.3 M
Memory (MB)	0.39	0.69	1.41

4.2 Optimization and Complexity Reduction

One of the main goals of this paper is to propose an efficient embedded hardware for Reinforcement Learning that can meet the throughput, area and power budget of embedded devices. Designing hardware for deep neural networks faces several challenges. These networks are computationally intensive and for achieving the best hardware performance i.e. small area, low power consumption, and high throughput, a careful trade off between design space hyperparameters should be taken into account. In this regard, we tested our architecture with three different configurations for the RL Agent to observe the performance of the model with respect to the number of computations and parameters. In deep neural networks, convolutional layers have the majority of the computation load of the architecture [17]. As a result, choosing the optimum number of filters can save a lot of timing and power budgets. In order to tackle the intensive computation problem, we reduce the number of filters and evaluate the design with three different filter sets for the convolution layer. The RL agent is trained using the proposed architecture and its performance is measured through recording the reward of completing the instructed task. First, second, and third convolutional layers are tested with the configuration of 8, 16, and 32 filter sets respectively. Table 3 summarizes the number of parameters, computations, and required memory space for each configuration, and Figure 4-(b) shows the reward function of each case with respect to the number of episodes. It can be observed that Config 2 achieves almost the same reward as the best configuration, Config 3 (the baseline), while requiring 2x less memory and 3x less computation. Therefore, we select the Config 2 for implementing on FPGA hardware.

4.3 Hardware Analysis and FPGA Implementation Results

Figure 5 shows the detailed block diagram of the proposed hardware architecture. We designed our hardware using Verilog HDL and implemented it on low-power Xilinx Artix-7 FPGA, with Vivado IDE. Hardware is designed to be configurable with different hyperparameters to meet custom application requirements such as low power consumption or high throughput. Parameters include the number of processing elements (PE), filter shapes, number of filters in the convolutional layers, sizes of the dense layers are configurable to whether engage maximum parallel processing ability of FPGA or utilize the least possible hardware resources.

Our hardware consists of three main units: (1) The convolution block, which performs the convolution operation with ReLU activation function. (2) The fully connected block that implements the dense layer functionality with MLP as the output of the last

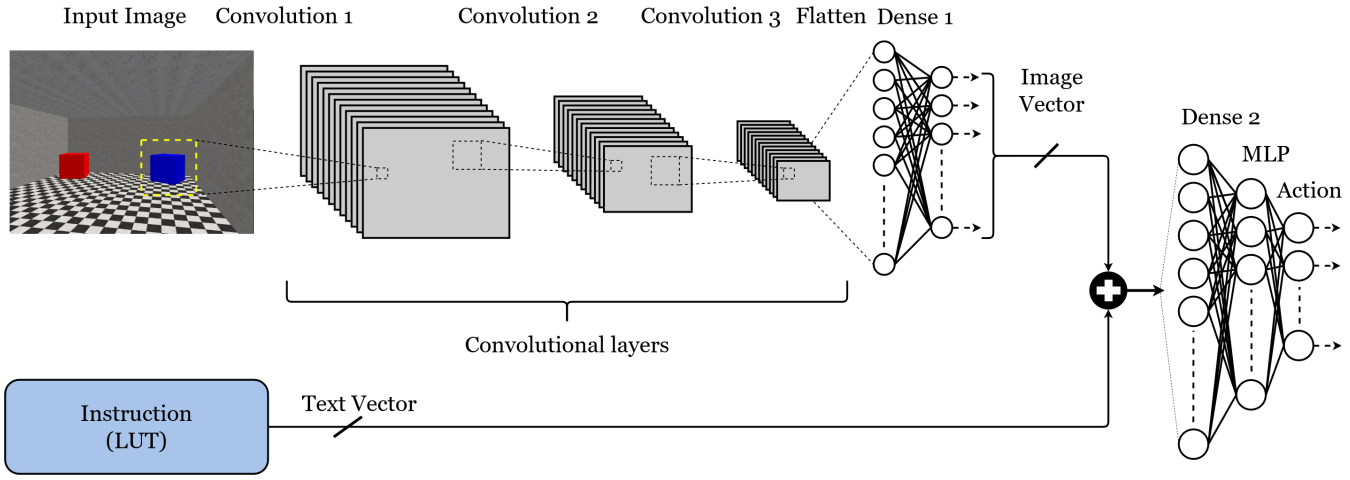
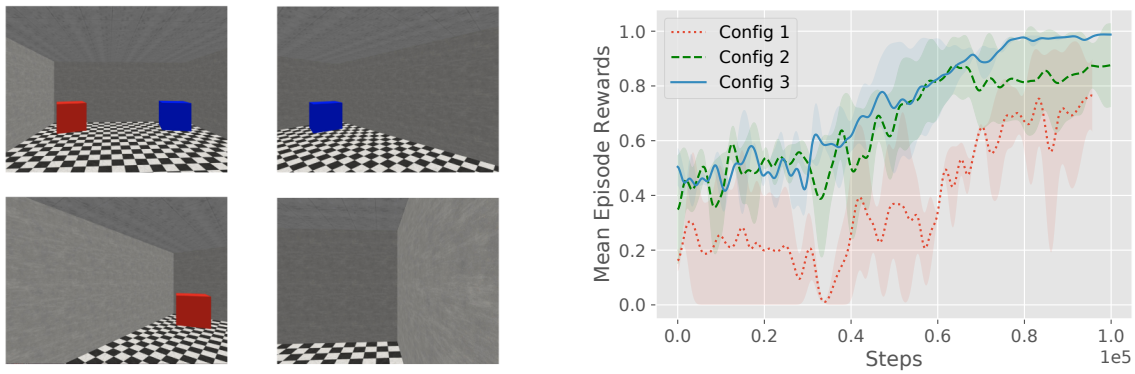


Figure 3: The architecture of the proposed system in detail. The network consists of 3 Convolutional, 2 fully connected, and 1 MLP layers and takes an image with the size of 80x50 as an input and produces 6 actions as an output.



(a) Miniworld Environment (b) Average Reward

Figure 4: (a) The Miniworld environment in which the agent tries to learn the policy of picking up an instructed object. The environment backend is changed so that the agent could be instructed. (b) Reward achieved by the RL agent in the Miniworld environment with respect to the number of episodes for different configurations. It is observed that Config 2 and 3 perform almost the same, while Config 1 fails to reach the same reward after the certain number of steps.

layer. (3) Instruction memory which stores structured language instructions vector. The convolution block operates with a multiplier embedded in its design. Along with that, there are separate memory blocks for saving weights and feature maps. The input data is fed to the convolution block which calculates the valid convolution of the input using a multiplication unit (MU) and an adder with ReLu activation logic. Strides of two in each direction replaced the time consuming max-pooling operation. Following the convolutional layers, the first fully connected layer reads the data from the feature map memory and performs the its operation using one multiplier, adder, and a few registers. The address flow and control unit generates memory addresses depending on the layer functionality of either convolution or dense. At this moment, the instruction vector stored in the lookup table (LUT) is concatenated with the output of the first fully connected layer, and form the input of the next fully connected layer. As discussed in section 3, a lookup table with pre-trained instructions vector which replaces the bulky language processing module during the test. This is followed up by another fully connected layer that performs the same operation and finally, the last layer of the architecture which is a MLP layer. The output of this layer is the action generated by the agent to navigate toward the instructed task within the environment. The control unit which is a Finite State Machine controls data for flowing through different parts of the hardware. 8-bit fixed-point number format is used for representing the data inside the hardware which guaranties the optimum balance between accuracy and power consumption.

We implemented our design with 1,2,4 and 8 number of PEs to measure the latency, throughput, power consumption and energy efficiency of each case. Table 4 shows the hardware implementation results for each case at three different frequencies. As it is shown,

Table 4: FPGA implementation result of the best configuration with different frequencies and different number of processing elements (PE).

Frequency	# PE	Utilization				Power(mW)	Latency(ms)	FPS	Perf(GOPS)	Energy(mJ)	Energy Eff(GOPS/W)
		LUT	FF	DSP	BRAM						
25 MHz	1 PE	1154	446	7	80	116	128	8	0.05	14.8	0.43
	2 PE	1564	443	7	80	124	65	15	0.1	8.1	0.81
	4 PE	1694	462	7	80	**139**	33	**30**	0.2	4.6	1.44
	8 PE	1926	525	7	80	141	16.8	60	0.4	2.4	2.84
50 MHz	1 PE	1156	447	7	80	137	64	16	0.1	8.7	0.73
	2 PE	1576	449	7	80	**153**	32	**31**	0.2	4.9	1.31
	4 PE	1686	460	7	80	182	16	63	0.4	2.9	2.2
	8 PE	1925	516	7	80	188	8.4	119	0.8	1.6	4.26
90 MHz	1 PE	1158	452	7	80	**173**	35	**30**	0.18	6.1	1.04
	2 PE	1573	443	7	80	202	18	56	0.36	3.6	1.78
	4 PE	1693	459	7	80	256	9	111	0.72	2.3	2.81
	8 PE	1931	518	7	80	264	4.6	217	1.44	**1.2**	**5.45**

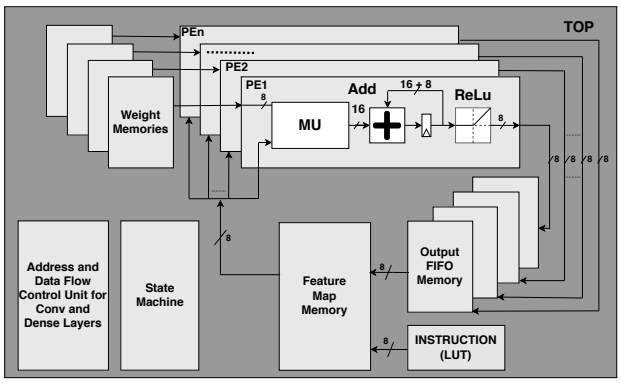


Figure 5: Block diagram of the proposed hardware architecture. The hardware architecture includes a top-level control unit which controls the convolution, fully connected, and MLP layers.

the best configuration in terms of energy efficiency is 8 PEs running at 90 MHz that resulted in an average 5.45 GOP/S/W and 1.2 mJ for each inference. This configuration consumes 264 mW to generates 217 actions per second which is much higher than typical 30 frames per second requirement of vision camera devices. Taking power constraint into the account, one can run this design with 4 PEs at 25 MHz to meet the real-time video constraint and consume less than 139 mW. Increasing the number of PEs and frequency increases the power and energy consumption, but because the increase of throughput and reduction of latency is higher than the increase of former two, it results in better power efficiency. Generally, it can be observed that while lower frequencies and number of PEs, yield lower power consumption, increasing them result in better energy efficiency.

4.4 Comparison with existing works

In recent years, many FPGA based accelerators for convolutional neural network and computer vision applications have been proposed. This paper proposes a hardware for reinforcement learning applications which takes images as its input. Therefore, we compare our design with previous works that targeted computer vision and image classification applications. Implementation results show that compared to [18], our hardware with an optimum selection of configuration parameters, number of PEs, and frequency, can consume 52.5X less energy per classification and is 13X more energy efficient. Likewise, it consumes 28X less energy per classification with 3.3X higher energy efficiency compared to [19]. Compared to [20], although our image dataset has larger dimension, the results show significantly higher throughput and less power consumption, and better energy efficiency. Also, our design shows a performance almost equivalent to [9] peak performance, despite using significantly larger input data. Comparison of this work with the existing FPGA based deep neural network is summarized in Table 5.

5 CONCLUSION

In this paper, we proposed an energy efficient hardware architecture for reinforcement learning that receives structured language as instructions for better guidance. We verified our design by testing its accuracy in Miniworld environment. Performance of the model is measured with three different configuration and the best configuration is selected to implement on hardware. The proposed hardware architecture is scalable with different number of PEs, and could be configured to achieve high throughput or low power consumption. It achieves 217 frames per second throughput with 1.2 mJ energy consumption per classification, when configured to run at maximum speed. At low power configuration, the hardware consumes less than 139 mW for 30 frames per second throughput. Comparing to the related works that use similar case studies on the similar platform, our hardware shows better performance with lower energy consumption and higher energy efficiency.

6 ACKNOWLEDGMENT

This project was sponsored by the U.S. Army Research Laboratory under Cooperative Agreement Number W911NF-10-2-0022. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official

Table 5: Comparison of the proposed hardware with related existing works

	[20]	[9]	[18]	[19]	**This Work**
Application	Image Classification (CIFAR)	Physical Activity Monitoring	Image Classification (CIFAR)	Image Classification (CIFAR)	**Reinforcement Learning**
Platform	FPGA Zynq	FPGA Artix7	FPGA Zynq	FPGA Zynq	**FPGA Artix7**
Input Dimension	32x32x3	64x40x1	32x32x3	32x32x3	**80x50x3**
Precision	16-bit fixed-point	16-bit fixed-point	16-bit fixed-point	32-bit floating-point	**8-bit fixed-point**
Freq (MHz)	180	100	100	100	**90**
Throughput (fps)	35.7	491	46.7	63.5	**217**
Performance (GOPS)	0.94	1.6	1.23	1.67	**1.44**
Power (mW)	>17587	175	2944	1015	**264**
Energy (mJ/frame)	>500	0.35	63	33.7	**1.2**
Energy Efficiency (GOPS/W)	<0.05	9.14	0.42	1.65	**5.45**

policies, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.
[2] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.
[3] Sunil Gandhi, Tim Oates, Tinoosh Mohsenin, and Nicholas R Waytowich. Learning behaviors from a single video demonstration using human feedback. 2019.
[4] Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. Learning to understand goal specifications by modelling reward. *arXiv preprint arXiv:1806.01946*, 2018.
[5] Prasoon Goyal, Scott Niekum, and Raymond J Mooney. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*, 2019.
[6] Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, Denis Teplyashin, et al. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*, 2017.
[7] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. *Def*, 2(6):4, 2006.
[8] Bharat Prakash, Nicholas Waytowich, Ashwinkumar Ganesan, Tim Oates, and Tinoosh Mohsenin. Guiding safe reinforcement learning policies using structured language constraints.
[9] Ali Jafari, Ashwinkumar Ganesan, Chetan Sai Kumar Thalisetty, Varun Sivasubramanian, Tim Oates, and Tinoosh Mohsenin. Sensornet: A scalable and low-power deep convolutional neural network for multimodal data classification. *IEEE Transactions on Circuits and Systems I: Regular Papers*, (99):1–14, 2018.
[10] Ali Jafari, Morteza Hosseini, Houman Homayoun, and Tinoosh Mohsenin. A scalable and low power dcnn for multimodal data classification. In *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–6. IEEE, 2018.
[11] Morteza Hosseini, Hirenkumar Paneliya, Utteja Kallakuri, Mohit Khatwani, and Tinoosh Mohsenin. Minimizing classification energy of binarized neural network inference for wearable devices. In *20th International Symposium on Quality Electronic Design (ISQED)*, pages 259–264. IEEE, 2019.
[12] Lahir Marni, Morteza Hosseini, Jennifer Hopp, Pedram Mohseni, and Tinoosh Mohsenin. A real-time wearable fpga-based seizure detection processor using mcmc. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2018.
[13] Katayoun Neshatpour, Arezou Koohi, Farnoud Farahmand, Rajiv Joshi, Setareh Rafatirad, Avesta Sasan, and Houman Homayoun. Big biomedical image processing hardware acceleration: A case study for k-means and image filtering. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1134–1137. IEEE, 2016.
[14] Bharat Prakash, Mark Horton, Nicholas Waytowich, William David Hairston, Tim Oates, and Tinoosh Mohsenin. On the use of deep autoencoders for efficient embedded reinforcement learning. In *ACM Proceedings of the 29th Edition of the Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 2019.
[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
[16] Maxime Chevalier-Boisvert. gym-miniworld environment for openai gym. https://github.com/maximecb/gym-miniworld, 2018.
[17] Morteza Hosseini, Mark Horton, Hiren Paneliya, Uttej Kallakuri, Houman Homayoun, and Tinoosh Mohsenin. On the complexity reduction of dense layers from o (n 2) to o (nlogn) with cyclic sparsely connected layers. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.
[18] Ying Wang, Jie Xu, Yinhe Han, Huawei Li, and Xiaowei Li. Deepburning: Automatic generation of fpga-based learning accelerators for the neural network family. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, New York, NY, USA, 2016. Association for Computing Machinery.
[19] Guanwen Zhong, Akshat Dubey, Cheng Tan, and Tulika Mitra. Synergy: An hw/sw framework for high throughput cnns on embedded heterogeneous soc. *ACM Trans. Embed. Comput. Syst.*, 18(2), March 2019.
[20] Gopalakrishna Hegde, Nachiappan Ramasamy, Nachiket Kapre, et al. Caffepresso: an optimized library for deep learning on embedded accelerator-based platforms. In *2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES)*, pages 1–10. IEEE, 2016.
[21] Katayoun Neshatpour, Hosein Mohammadi Makrani, Avesta Sasan, Hassan Ghasemzadeh, Setareh Rafatirad, and Houman Homayoun. Design space exploration for hardware acceleration of machine learning applications in mapreduce. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 221–221. IEEE, 2018.
[22] Katayoun Neshatpour, Hosein Mohammadi Mokrani, Avesta Sasan, Hassan Ghasemzadeh, Setareh Rafatirad, and Houman Homayoun. Architectural considerations for fpga acceleration of machine learning applications in mapreduce. In *Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 89–96, 2018.