ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

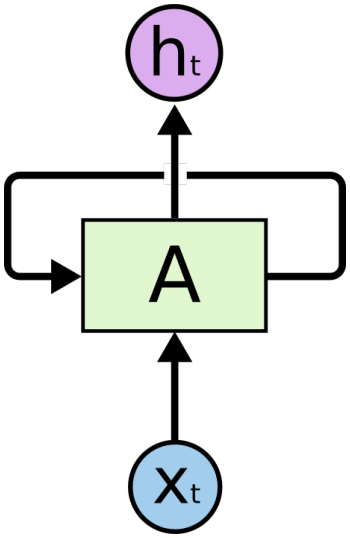
Institut für Integrierte Systeme
Integrated Systems Laboratory

Department of Information Technology and
Electrical Engineering

Autumn 2018

RNN Acceleration Extension for RISC-V

Project Report I



Renzo Andri
renzo.andri@iis.ee.ethz.ch

October 16, 2019

Supervisors: Prof. Dr. Luca Benini, lbenini@iis.ee.ethz.ch
Tomas Henriksson, Thomas.Henriksson@huawei.com

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Radio Resource Management (RRM) is getting more and more complex with new demands and developments like the fifth generation of mobile broadband system. Resource allocation has real-time constraints and is affected by several challenges. As machine learning has been evolved dramatically in recent years, more and more ML approaches are applied in RRM. But these new algorithm impose new challenges on real-time embedded systems, as they are compute intense and come with high memory requirements, but also provide a lot of potential of optimization due to their regular patterns. In this project, we will research extensions for sequential machine learning models for RISC-V cores in the RRM field and will be discussed in the following sections.

1.1 Machine Learning and Sequential Models

Even though Machine Learning has a long history, it has shown his real big breaktrhough in the last few decades driven by the availability of high compute capabilities and accessibility to large datasets which made complex models more than ever trainable.

1.1.1 Sequential Models

In most common machine learning application, the data distribution is assumed to be independent and identically distributed (i.i.d.) which allows to generate the likelihood function while looking at every sample point independently [16]. But for many applications this property does not hold any more, as the data depends on time and on previous samples. E.g. in speech recognition phoneme distributions are not i.i.d., but depend on the previous phonemes.

The most common classic sequential model is the Markov model [17] where the probability distribution of a sample $\mathbf{x_N}$ can be written as a (first-order)[1] markov chain

$$p(\mathbf{x_1}, ..., \mathbf{x_N}) = p(\mathbf{x_N}) \cdot \prod_{n=2}^{N} p(\mathbf{x_n}|\mathbf{x_{n-1}}),$$
$$p(\mathbf{x}_n|\mathbf{x}_1, ..., \mathbf{x}_{n-1}) = p(\mathbf{x}_n|\mathbf{x}_{n-1})$$

whereas every sample x_N directly depends on its predecessor x_{N-1} [16]. An extension to this model are Hidden Markov Models (HMMs) with discrete states and where the state can not be observed directly, but the output. These models have been used in speech recognition, natural language modelling, on-line handwriting recognition and analysis of biological sequences (e.g. proteins, and DNA) [16].

1.1.2 Feedforward Neural Networks

Artificial Neural Networks are brain-inspired models, which traditionally consist of a set of input neurons $\mathbf{x} \in \mathbb{R}^k$, hidden units $\mathbf{h} \in \mathbb{R}^l$ and output neurons $\mathbf{y} \in \mathbb{R}^m$ and the neurons are connected by synopses (or vertices), each of which is assigned a weight value $W_h(k, l)$ and $W_y(l, m)$ which is the contribution of the input neurons to the output neurons. The activation or transfer function σ_h and σ_y (e.g. Rectified Linear Unit ReLU[2]) is used to determine the state of $\mathbf{h}$ and $\mathbf{y}$ based on its input neurons and introduces non-linearity in the network which enables to learn non-simple tasks. The most basic neural network block is the Multi-Layer Perceptron (MLP) which has at least three layers: an input and output neuron layer and a (fully-connected[3]) hidden layer and can then be represented as the follows:

$$\mathbf{h} = \sigma_h(W_h\mathbf{x} + \mathbf{b}_h)$$
$$\mathbf{y} = \sigma_y(W_y\mathbf{h} + \mathbf{b}_y)$$

Typically several hidden layers are stacked together. Another special case are Convolutional Neural Networks (CNNs), where the spatial dependency (translation invariance and correlation of neighboring pixels) of the input and hidden units is exploited and a convolution kernel is learned per input and output neural layers instead of a weight for every neuron. Feedforward Neural Network have no dependency of previous samples and therefore do not have any cyclic paths in the network which makes them (comparably) simple to train, but are not so common to learn data with sequential dependency.

[1] In a N-order markov chain where every sample's probability distribution depends directly on N previous samples.
[2] The ReLU activation function can be expressed as follows: $\sigma_{ReLU}(x) = max(0, x)$
[3] All input neurons contribute to all output neurons

1.1.3 Recurrent Neural Networks RNN

To also take the sequential property of data (e.g. audio samples) into account, recurrent neural networks introduce recurrent vertices into the network, as illustrated in Fig. 1.1 where as $U_h \in \mathbb{R}^{m \times m}$ are the recurrent weights of a single-layer RNN.

The network can then be written as:

$$\mathbf{h}_t = \sigma_h(W_h \mathbf{x}_t + U_h \mathbf{h}_{t-1} + \mathbf{b}_h)$$
$$\mathbf{y}_t = \sigma_y(W_y \mathbf{h}_t + \mathbf{b}_y)$$

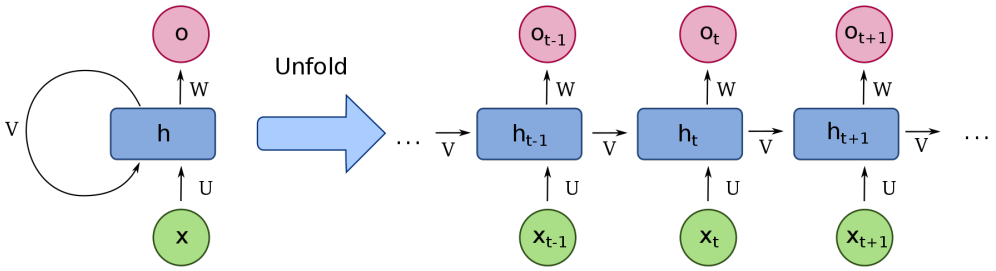


Figure 1.1: RNN cell unfolded in time [12]

Recurrent Neural Network (RNN) can support variable length in sequential model, but suffer from vanishing gradient problem during training making training slow and long-term dependencies hard to train.

RNN have been trained on a large set of applications, e.g. to create automatic image caption [18], text generation like poems, wikipedia article, linux kernels and language translation [19].

1.1.4 Long Short-Term Memory

Hochreiter and Schmidhuber introduced Long Short-Term Memorys (LSTMs) where an additional internal memory cell $\mathbf{c}_t$ and a forget gate $\mathbf{f}_t$ is inserted to a standard RNN. LSTM has been shown to be much less prune to the vanishing gradient problem and can therefore learn much longer time-dependencies [20]. The following formula and Fig. 1.2 show the structure of a typical LSTM cell unfolded in time.

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
g_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ g_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

with the following activation functions:
σ_g: sigmoid function[4]
σ_c: hyperbolic tangent
σ_h: hyperbolic tangent or identity.

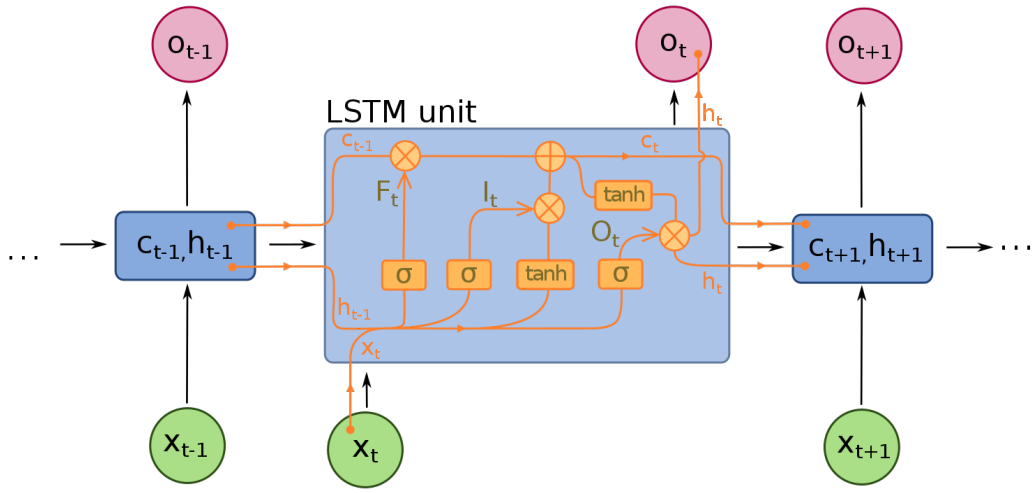


Figure 1.2: LSTM cell unfolded in time [13]

1.1.5 Gated Recurrent Units

Gated Recurrent Unit (GRU) are a simplified version of LSTMs which have no separate output gate and therefore have less computations and parameters. Similar results have been shown than with LSTMs [21], although no extensive study has been conducted yet.

[4] sigmoid function: $\sigma_g(x) = \text{sig}(x) = \frac{1}{1+e^{-x}}$

The GRU is illustrated in Fig. 1.3 and formalulated in the following equations:

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h) \\
\sigma_g(x) &= \mathrm{sig}(x), \sigma_h(x) = \tanh(x)
\end{aligned}
$$

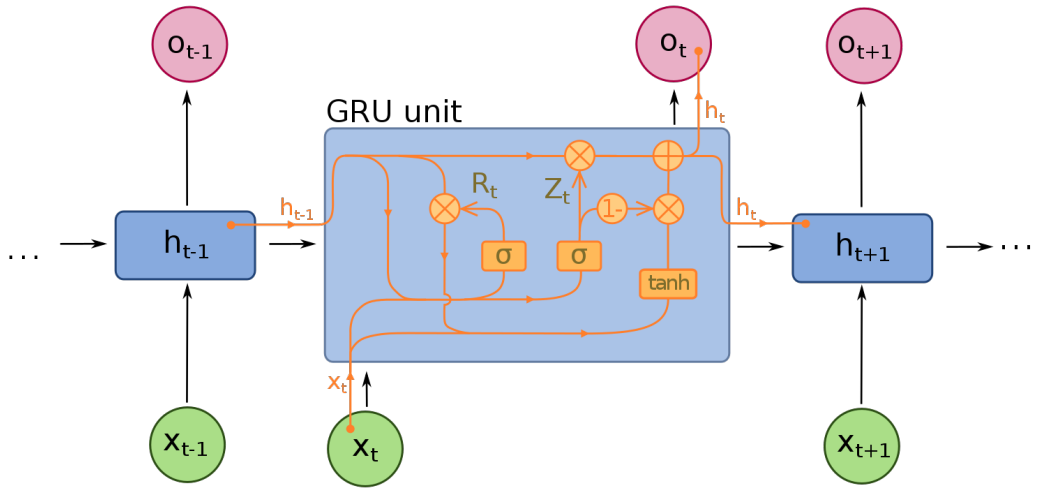


Figure 1.3: GRU cell unfolded in time [14]

1.1.6 Dilated Causal Networks

Recurrent paths make training much harder than in fully-feed forward networks, an alternative presented by Cheema et al. are Dilated Causal Networks, where time is encoded into the 1-d input feature space [15]. Additionally, the data is fed through the network in a logarithmic way, as illustrated in Fig. 1.4 to support a wide range of sequential dependency in the input data. These networks have been used for training motion detection [15] and machine translation [22] and on a set of common sequential model tasks [23] with slightly better performance and with much fast convergence compared to recurrent models.

1.1.7 Spectrogam-based (Convolutional) Neural Networks

Sequential data can also be represented in a spectrogram, where as a sliding window of sequential samples are mapped to the frequency domain using FFT. The resulting time-to-frequency image is than fed to a conventional convolutional network and has been used for speech recognition [24].

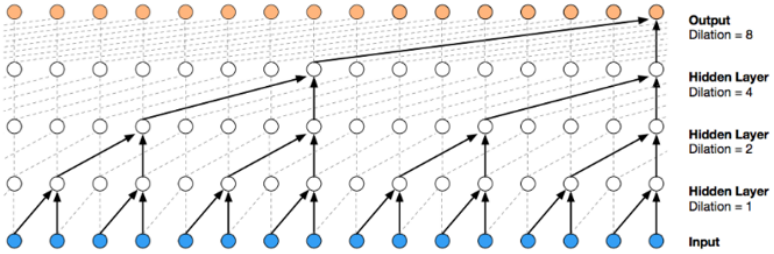


Figure 1.4: Dilated Causal Networks [15]

1.1.8 Reinforcement Learning and Q-Learning

Q-Learning is a reinforcement learning technique. Typically, the model setup includes an agent in a environment, which can have a set of states $s \in S$ and can apply an action of a set $a \in A$. After applying action a, the environment returns a reward $r \in R$ back to the agent.

The action policy learned in Q-Learning is represented by the Q-function $Q : S \times A \to \mathbb{R}$ mapping state action pairs to the expected reward. The learning procedure happens iteratively by updating Q in the following way:

$$Q_{k+1}(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q_k(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot \max_{a \in A} Q(s_{t+1}, a) \right)$$

where as α is the learning rate and γ accounts for future rewards. Typically the agent applies the action which has the highest Q values for the current state (exploitation) or selects a random action (exploration) [25].

In traditional Q-Learning an extensive Q table is learned including all $|S| \cdot |A|$ possible entries. Recent works instead suggest to learn a (deep) neural network to represent the Q function, also known as Deep Q-Network (DQN) [25].

Dueling DQN DDQN
In dueling DQN Q function is split into the state value $V(s)$ and the advantage value $A(s, a)$, where as $Q(s, a) = A(s, a) + V(s)$. Both values are being learned by two separate neural networks [26]. DDQN can better account for cases, where the state is either good or bad independently from the taken action.

1.2 Radio Resource Management (RRM)

Wireless communication is omnipresent nowadays and the desired data total bandwidth is driven by the increasing number of devices (e.g Internet of Things (IoT) end nodes and

smartphones) and the increasing requirements of the applications (e.g. virtual reality, video-telephony). RRM manages the allocation of radio resources (e.g. bands, transmit power, data rates, modulation, error coding schemes, ...) in a multi-cell and multi-user setting which imposes crosstalk from other radio transmitters (Co-channel interference (CCI)), especially as publicly available frequency bands are limited [27].

Implementing the RRM efficiently imposes several challenges: The clients are heterogeneous (e.g. tiny sensor-nodes vs. mobile routers) and clients are moving within and between network cells. Parameter space is high-dimensional and components within the network are varying and state changes are happening stochastically. Especially, theses tasks have to be executed in the frame of milliseconds which exclude compute-intense algorithms [4]. Additionally, radio cells are overlapping and observability and controllability of the entire system is limited [5].

Typically, the system has been modelled with full observability and optimizing convex problems with algorithmic [4] approaches like the weighted sum-rate MSE algorithm [28] and fractional programming [29] which are calculated in an iterative way and some of them need to calculate heavy operations (e.g. matrix inversion or singular value decomposition) in every iteration.

As these systems would profit from a reactive learning setting, also reinforcement learning approaches have been applied. Q-Learning with a single agent [30–32] or multi-agent setting with distributed agents [5, 33, 34] or centralized agents [8]. Beside of the classic Q-learning approach [35,36], also deep Q-networks have been presented using feedforward neural networks [3, 5, 6, 8] or LSTMs [2].

Chapter 2

Benchmarks

To evaluate several relevant implementations and extensions for sequential models on RISC-V cores in the RRM field, we have selected several recent publications, which are described in Table 2.1. Even though the optimization objectives of these networks are different (e.g. throughput/spectral efficiency maximization, interference reduction, fair allocation), the network topology and learning procedure are similar, but differ mainly in the kind of network layers (LSTM, feedforward neural networks (MLP and CNN) and the learning method (supervised learning or reinforcement learning with DQN). Not all of the papers have published the exact setup (e.g. number of access nodes or frequency bands) and therefore also not all relevant network parameters (e.g. input and output neurons). In the following, the number of antennas is set to $K = 4$ and the number of frequency bands to $N = 3$ otherwise indicated in the paper.

2.1 Proactive Resource Management in LTE-U Systems: A Deep Learning Perspective [1]

Challita et al. have presented a learning framework in the field of Long Term Evolution in unlicensed spectrum (LTE-U) (4G), where as Small (cell) Base Stations (SBS)[1] are sharing unlicensed bands. The number of bands is rather small, therefore fair and an altruistic policy is needed and should be learned. The SBS are not collaborating directly, but try to achieve long-term fairness measured in average airtime per radio while optimizing proactively dynamic channel selection, carrier aggregation and fractional spectrum access. Challita et al. show that they reach a mixed-strategy Nash equilibrium and can gain up to 28% over a conventional reactive approach. They use a combination of LSTMs

[1] low-powered cellular radio access nodes with a range of 10 m to 1 km, few concurrent connections/sessions

and MLP: The state encoder tries to reconstruct the predicted action sequence and is modeled with a one-layer LSTM with 70 cells, followed by a summarizing fully-connected layer with unknown size and has been set to the same size (i.e. 70) and is followed by the action decoder modelled by a one-layer **70 cell** LSTM with $K = 4$ output neurons.

2.2 Deep Multi-User Reinforcement Learning for Distributed Dynamic Spectrum Access [2]

Naparstek et al. apply Deep Q-Learning to solve the Dynamic Spectrum access (DSA) utilization problem. The time is slotted into fixed-size times slots and each user selects a channel and transmits a packet with a certain attempt probability. After each slot, the user gets an acknowledgement whether the transmission was successful or not. The problem is learned with a DQN approach, where as the network consists of $(2N + 2)$ input neuron where the first $(N + 1)$ neurons is encoding the last channel selected and the other (N) encodes the capacity of the N sub-bands and 1 for the acknowledge signal. These neurons connect to a single-layer LSTM with unknown size (i.e. set to $2N + 2$), and finally fed through two independent linear layers (the value layer and the advantage layer) based on the dueling DQN principle; both layers have $N + 1$ output neurons each. As the exact network topology has not been published, the number of layer is set to 1. Based on double Q learning, the network is trained separately to choose the action and to estimate the Q-value associated to the according action.

2.3 Deep reinforcement learning for resource allocation in V2V communications [3]

Ye et al. elaborate the resource allocation problem in vehicle-to-vehicle and base station-to-vehicle communication setup (e.g. information on traffic safety). Every vehicle is an agent deciding independently to reach an optimal band and power level selection. The system is modeled in a Q-Learning setup where as the reward is determined by latency and reliability metrics, the state is based on the channel information of the corresponding link, previous interference on the link, channel information to the base station, selected sub-channel of neighbors in the previous time slot, the remaining load to transmit and the remaining time to meet latency constraint. The actions are the sub-band to select and transmission power. The Q function is then learned and modelled by a **5 layer fully-connected** neural network with 500,250,120 hidden neurons. There are 6 input neurons and $2N$ (#frequency bands) output neurons.

2.4 Learning to optimize: Training deep neural networks for wireless resource management [4]

H. Sun et al. are solving the general problem of interference channel power control, with K single-antenna transceivers pairs sending data as a Gaussian random variable and independently of each other. Differently, to previous state of the art is learning a MLP to perform the WMMSE algorithm. The input to the network is the magnitude of the channel coefficients and the output are the power allocations. Two models are evaluated: Model 1 is considering all channel coefficient to be Rayleigh fading distributed with zero mean and unit variance which has been used in various resource allocation algorithm. Model 2 considers a multi-cell interfering MAC setting with N regularly placed cells and K randomly distributed users. The proposed network consists of K^2 (Model 1) or $N \times K$ (Model 2) input neurons for the channel coefficients and the output is the set of power allocations K and **3 hidden layer** with **200** neurons each. The presented results have worse accuracy (2-16%) than the baseline WMMSE algorithm, but are up to 33× faster.

2.5 A reinforcement learning approach to power control and rate adaptation in cellular networks [5]

Ghadimi et al. propose a DQN learning approach combined with ensemble learning to optimize downlink power control and rate adaption in cellular networks and to overcome the limitations of missing system observability in previous approaches. Agents are not collaborating and not controlled at a centralized unit. The state is represented by the cell power, average Reference Signal Received Power (RSRP)[2], average interference and cell reward. The action is the increase or decrease of the transmission power by $\{0, \pm 1, \pm 3\}$ DB and the reward is based on the α-fair resource allocation utility function [37]. An ensemble of several (fully-connected forward) DQN with **3 hidden layers** have been trained, but topologies are unknown.

2.6 Deep-Reinforcement Learning Multiple Access for Heterogeneous Wireless Networks [6]

The work of Yu et al. focus on the problem of sharing time slots among multiple of time-slotted heterogeneous network nodes adopting different MAC protocols (DMA, TDMA and ALOHA) objective to sum throughput or α-fairness. All nodes connect to the same base station. The network nodes work independently in a re-inforced way. Possible actions a_t are transmit and wait and reward or channel observation r_t are success, collision

[2] A metric for average user distance.

or idleness. The state is defined as a history of state and action pairs (of $M = 20$ length). The Q function is implemented by a multi-layer fully-connected network with $5M = 100$ input neurons and 2 output neurons and 6 hidden layers, where as all of them have 64 neurons and two residual paths are added between layer 3 and 5 and between 5 and 7. The network output 2 Q values for transmit and wait action.

2.7 Learning Optimal Resource Allocations in Wireless Systems [7]

Eisen et al. are formulating the general resource allocation problem, as a Lagrangian Dual Problem and show that the duality gap is zero when learned by a neural network (converging to a local minimum). Two problems are looked at: 1) A simple capacity maximization over a set of simple Additive White Gaussian Noise (AWGN) wireless fading channel, where as K users are given a dedicated channel to communicate under the constraint of a total expected power budget and 2) a capacity maximization in an interference channel with K transmitter sending to a single access point. As the 2nd example has a non-convex capacity, it is not solvable with the classic dual approach, but with a neural network. The neural network is built with fully-connected layers and K input neurons, 32 and 16 hidden neurons and K output neurons. The number of users have been set to $K = 4$.

2.8 Deep Reinforcement Learning for Distributed Dynamic Power Allocation in Wireless Networks [8]

Nasir et al. are presenting another model-free but distributed approach for the power allocation scheme on single frequency bands based on deep reinforcement learning. All transmitters collect Channel State Information (CSI) and Quality of Service (QoS) information from several neighbors. The network is learned in a centralized server, observations and training data is collected at the nodes and transmitted to the central unit, and the weights are updated simultaneously on the base stations. The state of the Deep Q network is based on local information (transmit power in the last time slot, its contribution ratio, downlink channel measurement, total interference-plus-noise power at its own receiver) and the interference from its neighbors or to the neighbors. The system is time slotted and actions are taking instantaneously. The actions are the discrete power levels selected and the reward function is defined as the contribution to the collective spectral efficiency minus the penalty of caused interference. The network consists of one input layer with 7 internal states, $6c$ interferer neighbor state and $4c$ interfered neighbor states, which are 57 input states for the use case of 5 agents. The hidden layers consist of 200,100 and 40 neurons and 10 neurons or 10 discrete power levels have been chosen.

2.9 Deep Learning for Radio Resource Allocation in Multi-Cell Networks [9]

Ahmed et al. are looking at the sub-band and power allocation problem in a multi-cell network (with K cells, U users and N sub-bands. Differently to previous approaches, the base stations are exchanging channel quality indicators to their neighbors. Stacked-autoencoder are used and pre-trained with a genetic algorithm, before their encoder parts are stacked to a MLP with $K \cdot K \cdot (N+1) = 100$ input neurons and 4 hidden layers with 1080, 720, 360 and 180 neurons, followed by a softmax layer with 180 output neurons.

2.10 Deep Power Control: Transmit Power Control Scheme based on CNN [10]

Lee et al. are optimizing spectral efficiency (throughput) and energy efficiency in an environment with N single-antenna transceiver pairs. The state is determined by $h_{i,j} = |g_{i,j}|G_{i,j}$ which is composed of the distance related channel gain $G_{i,j}$ and the multipath fading $g_{i,j}$ between the transmitter i and receiver j. After normalization, the N^2 state features are fed to a neural network with 7 convolutional layers with 3×3 kernels and 8 intermediate channels, followed by a single fully-connected layer with N output neurons which are fed to sigmoid activation layer to determine the transmit power. While having full channel information, this approach is slightly better than WMMSE and one order of magnitude faster, in the distributed case where just little information is transmitted and just a part of the channel information is available, the performance is just slightly worse than WMMSE algorithm.

2.11 Deep reinforcement learning for dynamic multichannel access in wireless networks [11]

Wang et al. consider a multichannel access problem, where as there are $N = 16$ correlated channels, each of which have two possible states (good or bad) and their joint distribution is modeled as a (partially observable) Markovian model. They learn a Deep Q network which is learned centralized. A single user at a time can select a channel to transmit a packet and either it is successfully sent (reward = 1) or failed due to bad state (reward = -1). The state of the agent is defined as a set of $M = N = 16$ previous actions and observed channel conditions and the action is the selected channel. The DQN consists of the $M \cdot N \cdot 2 = 512$ input neurons, two hidden layer with 200 neurons each and N output neurons.

2.12 Overview

Table 2.1 gives an overview of the presented benchmark networks comparing the different optimization objectives, network structure and network types.

Ref.	Paper	Optimization Objective	Network structure	Network Typ
[1]	Proactive Resource Management in LTE-U Systems: A Deep Learning Perspective	Throughput and Fairness	10-70-70-70-4	LSTM
[2]	Deep Multi-User Reinforcement Learning for Distributed Dynamic Spectrum Access	Throughput (Decentralized multi-agents)	(2N+2)-(2N+2)-(2N+2)	LSTM-FC, DQN
[3]	Deep reinforcement learning for resource allocation in V2V communications	Interference under latency constraints	6-500-250-120-3N	FC-MLP, DQN
[4]	Learning to optimize: Training deep neural networks for wireless resource management	Throughput	2K-200-200-200-2K	FC-MLP
[5]	A reinforcement learning approach to power control and rate adaptation in cellular networks	Throughput and Power	15-?-10	FC-MLP, DQN
[6]	Deep-Reinforcement Learning Multiple Access for Heterogeneous Wireless Networks	(Sum-)Throughput and Fairness	20-64-64-64-64-64-64-64	FC-MLP, DQN
[7]	Learning Optimal Resource Allocations in Wireless Systems	(Sum-)Throughput	K-32-16-K	FC-MLP
[8]	Deep Reinforcement Learning for Distributed Dynamic Power Allocation in Wireless Networks	Throughput	(10K+7)-200-100-40-1	FC-MLP, DQN
[9]	Deep Learning for Radio Resource Allocation in Multi-Cell Networks	Utility/Throughput	100-1800-720-360-180	FC-MLP (Autoenc.)
[10]	Deep Power Control: Transmit Power Control Scheme b. on CNN	Throughput or Energy Efficiency	10^2-(7l×8·10^2)-10	CNN (3x3)
[11]	Deep reinforcement learning for dynamic multichannel access in wireless networks	#Successfull transmission (Throughput)	512-200-200-16	FC-MLP, DQN

Table 2.1: Benchmark list for different network models used in the RRM setup.

Chapter 3

RISC-V

RISC-V has started as a research project at UC Berkley and become the leading ISA in open source hardware in academia and industry. The RISC-V toolflow is freely available and includes a GNU/GCC software tool chain, GNU/GDB debugger, LLVM compiler etc. RISC-V standard can freely be changed and extended.

For further information on the exact ISA specification the reader is referred to the official RISC-V specification which can be found on `http://riscv.org/specifications/`.

3.1 The PULP core - RI5CY

The compute platform considered in this project is the PULP platform (Parallel Ultra Low Power)[1]. The project has been started in 2013 and is a joint project between the Integrated Systems Laboratory of ETH Zurich and the Energy-efficient Embedded Systems (EEES) group of the University of Bologna and serves as a platform to explore new and efficient architectures for ultra-low processing. The main goals of the project is to develop an open and scalable hardware and software research platform to bring down the energy efficiency below the envelope of a few milliwatts, but still satisfy the computational demands of smart IoT applications.

The project comes with a comprehensive set of hardware and software IPs which allows to tailor PULP to the preferred compute platform with multiple cores, heterogeneous HW accelerators, interfaces (e.g. I^2C, SPI, JTAG, ...).

The RISCY core is a 4-stage pipeline processor and is illustrated in Fig. 3.1 supporting the RISC-V standard plus custom extensions (i.e. RV32IMFCXpulp ISA). In the execute pipeline stage, there are an arithmetic logic unit, control/stage register, multiplier unit

[1]The project website can be found on `http://www.pulp-platform.org`.

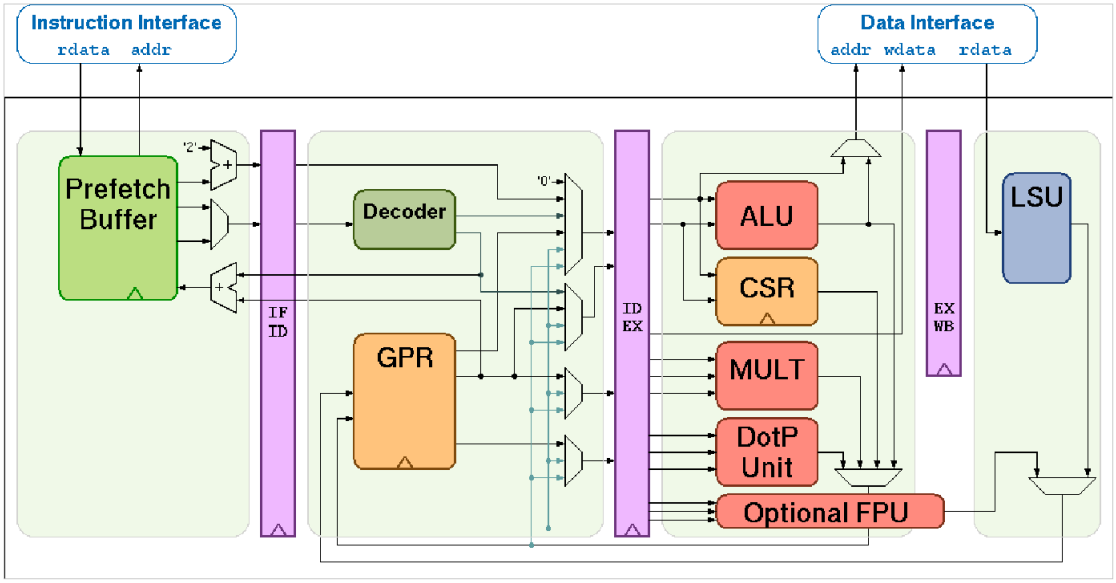


Figure 3.1: Schematics of RI5CY core

and an optional floating point unit. The core has an area of 70 kGE[2] and reaches a coremark/MHz of 3.19.

For the evaluation, we use a simple single-core system, illustrated in Fig. 3.2. The system includes a RI5CY core and 1.5 MB SRAM, where as the 1.2 MB are reserved for data and 300 kB for instructions to avoid access conflicts between data and instructions.

3.2 PULP Toolflow

PULP comes with an entire software toolflow. This includes PULP debug unit which allows to access the core with GDB or JTAG. On-chip performance counter are supported for a variety of metrics: Instructions counts, stalls (memory, branches, jump), statistics on branches (taken, not-taken) and number of memory accesses. Furthermore, the compiler infrastructure is based on the RISC-V GCC toolchain, extended to support custom instructions. Finally, the system can be fully simulated with the virtual platform which allows for very fast simulation and performance assessment compared to a RTL simulation and has a cycle-level divergence of less than 10% and therefore enables fast prototyping.

[2]Gate Equivalent is a metric to describe the technology-independent logic complexity. 1 GE is equal to a 2-NAND gate which is 0.19 μm^2 in GF 22nm FDX.

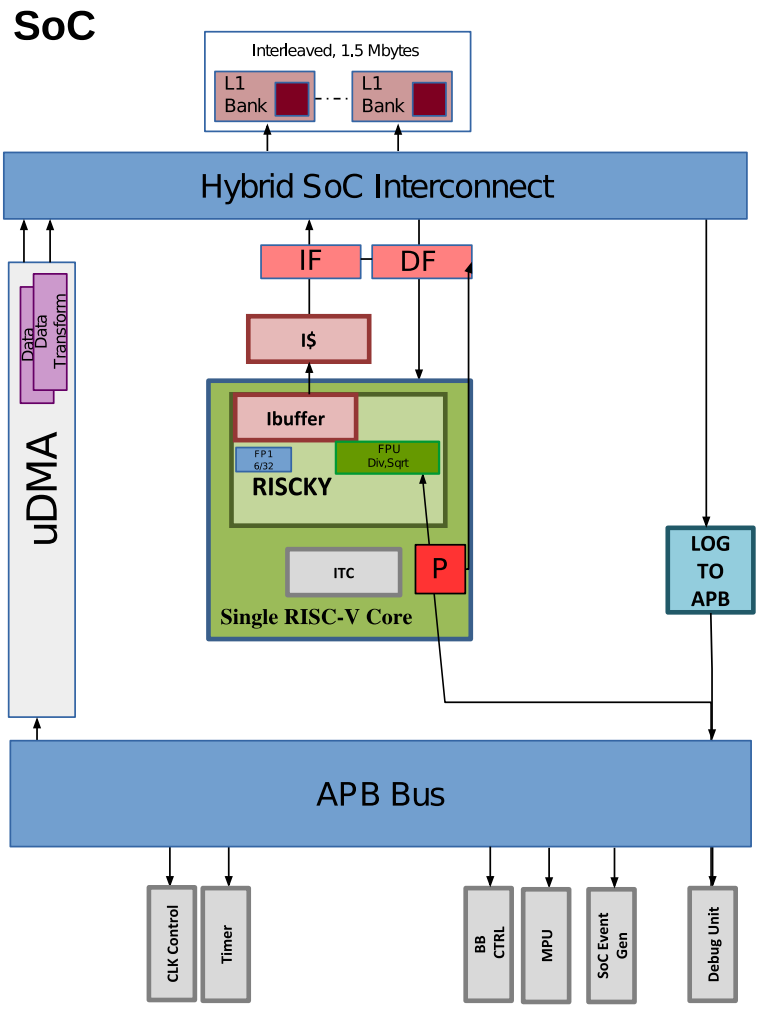


Figure 3.2: Top-Level RISC-V Architecture

3.3 RI5CY Extensions

The standard RISC-V ISA has been extended to speed up the very common execution patterns. The main extensions are described in the following:

Memory reads happen often in bursts which means that consecutive memory addresses have to be accessed. Often these addresses have to be calculated separately which introduces a high overhead of additional stalled cycles. To decrease this overhead RISC-V has been extended with a post increment load and post increment write. In this way the address pointer is incremented in hardware and no additional instructions are needed, which gives in average an overall speed up of 2x for memory accesses.

Also loops with few instructions inside have a high overhead, as the iteration variable has to be incremented and checked if the loop is finished. To reduce this overhead, hardware loops have been introduced. Hardware loops are first configured with the number of iterations and the final instructions in a single cycle. This extension leads to just a 5% area overhead while reaching an average speed up of 2x. Fig. 3.4 shows a simple example where two nop instruction should be executed 100 times: with the special function lp.setupi the number of iterations are set to 100 and the PC pointer is set to the end of the loop.

```
1:    for each row in window
2:        uint32_t a ← &BIN1[FP]
3:        uint32_t b ← &BIN0[FP+offset]
4:        sum += __buitin_popcount(a ^ b)
5:        j++
6:    end for
```

Figure 3.3: Built-in functions allow to explicitly tell the compiler to place special-purpose instructions like popcount.

For some specific applications special bit manipulation instructions have been introduced. E.g. popcount returns the number of 1s in a integer register. These instructions are integrated into the software toolflow and intrinsic function (e.g. Fig. 3.3 shows an example for popcount) have been added.

For very low level optimization the asm directive (standard GCC function) can be used to write assembly code within the C implementation.

Fixed-Point support have been added in the DSP extension (e.g. MAC with normalization and round).

The packet-SIMD extensions allows to pack several data into one 32-bit word. E.g. 2 16-bit words, 4 8-bit words or 8 4-bit words can be packed into a 32 bit register. Furthermore, a set of DSP extensions for pSIMD are supported like MIN, MAX, multiply-accumulate, multiply-subtract, dot product and others. Especially they can calculate ADD, SUB, MUL, MAC with fixed-point support including normalization and rounding. An example

Original RISC-V

```
mv   x5, 0
mv   x4, 100
Lstart:
  addi  x4, x4, -1
  nop
  nop
bne  x4, x5, Lstart
```

HW Loop Ext

```
lp.setupi 100, Lend
nop
Lend: nop
```

Figure 3.4: Hardware Loops: Example how number of instructions are reduced.

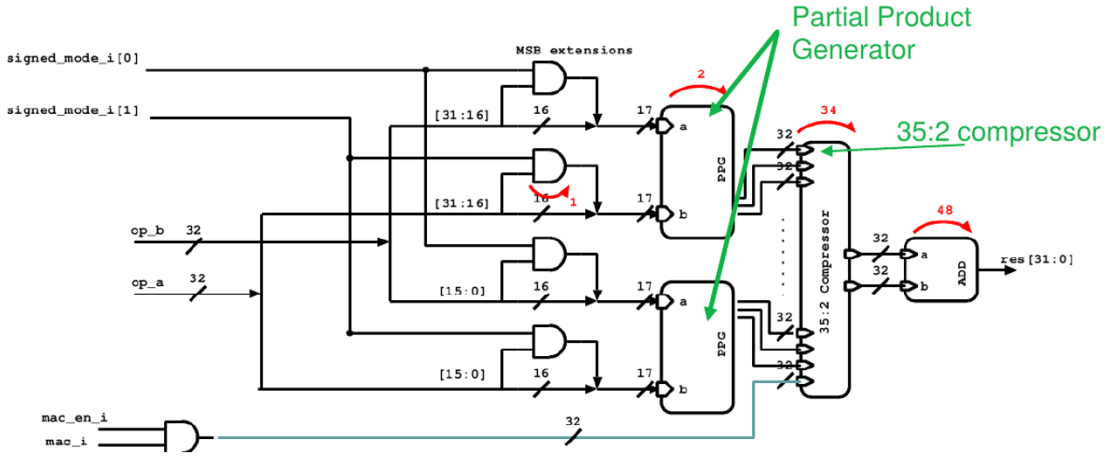


Figure 3.5: Dot-Product with packed SIMD. Upper half and lower half words are multiplied and summed together.

for a 16-bit word dot product is shown in Fig. 3.5, where as the upper 16 bits of both source registers are multiplied and then added to the product of the lower 16-bits.

3.4 Implementation and Evaluation of the Benchmark Suite

To allow for simple deployment of neural networks for sequential input data and general neural networks on RISC-V cores the project has been built in a modular way and split into the ML framework and RISC-V C implementation.

There are plenty of Machine Learning platforms which have been developed by the community in recent years: TensorFlow, pyTorch, Caffe, etc. In this project, we use python as it is more open and extendable and used by the research community. PyTorch is also more flexible in deployment and debugging compared to its main competitor TensorFlow.

We have implemented the networks, as described on pyTorch and it serves as a golden model. We have implemented a toolflow to export these networks to RISC-V compilable C code including all the parameters in the right format (e.g. 16-bit fixed-point numbers or float).

On the RISC-V side, the basic kernels and neural network layers have been implemented and are discussed more in detail in the following section.

3.4.1 Basic Building Blocks

The code for the network layers (linear layer and LSTM) has been decomposed into smaller functions:

- `LinearLayer()`: calculates a matrix-vector multiplication (weights and input neurons) followed by a vector addition (bias).
- `Two-LinearLayersAcc()`: To calculate the LSTM internal nodes (f, i, o, g, h) two linear layers are calculated and accumulated. To minimize writing back partial results to the L1 memory and read them again to accumulate with output of the second linear layer again, the Two-Linear basic block has been added. The output neurons are directly calculated, such that all the input neurons for the first and the second layer are weighted and accumulated, before next output neuron is calculated.
- `HadMulTensor()`: calculates the point-wise multiplications of two Tensors which are needed to calculate the hidden nodes in LSTM and GRU cells.
- `SigTensor()`: calculates the Sigmoid for every element of a Tensor.
- `TanhTensor()`: Hyperbolic Tangent: calculates Hyperbolic Tangent for every element of a Tensor.

The layers are implemented as follows:

- `LinearLayer()`: Calculates a fully connected layer (or MLP) which is identical to the basis function Linear.
- `RNNLayer()`: Calculates the RNN output based on the input feature map and set of weights and pointers (assigned by pointer). The hidden nodes are calculated and stored at the same address (in-place by pointer).
- `LSTMLayer()`: Calculates the LSTM layer, hidden nodes are calculated in-place (by pointer) and the internal nodes are calculated with the `Two-LinearLayersAcc()` function and hidden and cell nodes are calculated with the Hadamard product.

- `InferNetwork()`: Takes a list of layers which themselves include all relevant parameters (by pointer) and calculates the output of the network by layer-wise application of the right functions (`LSTMLayer()`, `LinearLayer()`, ...). Organizes pointers to input and output feature map in a ping-pong manner (pointer of the inputs neurons are set to the previous pointer of the output neurons).

3.4.2 Results

We have used the built-in performance counter and traces of the virtual platform to evaluate the performance. Tab. 3.1 shows the number of parameters, it can be seen that most of the networks have below 150 k parameters, where as Ahmed et al. [9] and Lee et al. [10] have several million parameters, which will be needed to stored in off-chip memory. The benchmark suite requires 5.7 MB memory to store all parameters for 16-bit words.

[1]	30k	
[2]	1k	
[3]	160k	
[4]	85k	
[5]	(0k)	entire topology unknown
[6]	24k	
[7]	1k	
[8]	37k	
[9]	1'174k	// large network
[10]	(4'032k)	CNN, external memory needed
[11]	150k	
total	1657k	

Table 3.1: Number of Parameters

To understand the number of active cycles and stalls, we have analyzed the stalls appearing while running the benchmark suite. Table 3.2 gives an overview. It can be seen that most of stalls are during memory accesses (24.6% in average). The first two networks have small numbers of hidden nodes, which introduces a high overhead when a branch is taken and leads up to 7.5% overhead in cycles for branches. The overall cycle per instruction (CPI) is 1.33.

Tab. 3.3 shows the number of cycles in the different basic function blocks. For the networks with no other than fully-connected layers, just the function `LinearLayer()` is executed and therefore not added in the table. For the networks containing LSTMs and RNNs, Tab. 3.3 shows the function split with the amount of cycles in every instruction. It can be seen that Sigmoid and Hyperbolic tangent take a lot of cycles within the LSTMs (e.g. 42% in [2]) when calculated as Taylor expansion with 4 elements (row 1 and 3)

	MLP	LSTM	#cycles	active	data stalls	branch stalls	others	CPI
[1]	1	2	279.7 k	210.5 k	21.4%	2.5%	0.9%	1.33
[2]	2	1	11.4 k	8.7 k	13.1%	7.2%	3.0%	1.30
[3]	4	0	1282.3 k	963.9 k	24.8%	0.0%	0.0%	1.33
[4]	8	0	1361.3 k	1024.1 k	24.8%	0.0%	0.0%	1.33
[6]	6	0	188.1 k	141.9 k	24.5%	0.0%	0.0%	1.33
[7]	3	0	6.6 k	5.1 k	22.3%	0.2%	0.3%	1.30
[8]	4	0	291.6 k	219.6 k	24.7%	0.0%	0.0%	1.33
[11]	4	0	1196.4 k	898.8 k	24.9%	0.0%	0.0%	1.33
Total	32	3	4617.4 k	3472.7 k	24.6%	0.2%	0.1%	1.33
	FPS@50MHz		10.8					

Table 3.2: Instruction and Cycle Count for the selected Benchmark Suite

and even with piece-wise linear approximation with 8 pieces (row 2 and 4) up to 33%. Therefore, these functions seem are good candidates for further analysis on instruction acceleration.

network	approxim. model for sig/tanh	#cycles	MLP	LSTM	`LinearLayer()`	`LSTMLayer()`	`TwoLin...()`	`SigTensor()`	`TanhTensor()`	Oth.
[1]	taylor	279.7 k	1	2	14.4%	85.6%	70.4%	8.2%	5.8%	1.1%
[1]	linear	267.8 k	1	2	15.0%	85.0%	73.5%	6.2%	4.1%	1.2%
[2]	taylor	11.4 k	2	1	6.2%	93.8%	47.7%	24.5%	17.4%	4.1%
[2]	linear	10.1 k	2	1	7.1%	92.9%	54.5%	20.3%	13.3%	4.7%

Table 3.3: Cycle statistic for basic blocks with sigmoid/hyperbolic tangent calculated either with taylor or piecewise linear approximation

Table 3.4 shows the histogram of the instruction during execution of the benchmark suite with existing extensions. The code is calculating 480 kMAC. There is just limited register re-use for data and weights and therefore for every MAC 2 data items have to be loaded, leading to 964 k instructions for load (p.lh). The overhead for shifting back the multiplication results and additions for the fixed-point representation is quite high with a total of 960 k instructions (30%), which should be improved by using the intrinsic SIMD data types.

Table 3.5 shows the instruction count if all the extension (SIMD and hardware loops and load/store increment) are turned off. It can be seen that 2.2x more cycles are needed, especially a lot of cycles are used to calculate addresses (i.e. addi is a common instruction used for this), furthermore the branch instruction appears on the top instruction as

Instr.	#	Instruction description
p.lh	964k	read 2 half words
c.add	494k	addition (16-bit compressed instr.)
mul	489k	32bit to 32bit multiplication
p.exths	480k	sign extension half-word
c.srai	480k	signed arithmetic right shift immediate
c.mv	22k	register to register copy
other	267k	other instructions
	3'199k	total instructions
	4'294k	total #cycles

Table 3.4: Instruction and Cycle Count for the Benchmark Suite with extensions.

looping is common in the calculation of neural networks and this overhead is much lower in the version with the extensions turned on.

c.addi	1'425k	add immediate (mostly for address calculation)
lh	1'418k	(single)load half word
bne	713k	branch not equal
c.add	712k	addition
c.slli	708k	shift left immediate
c.srai	708k	signed arithmetic right shift immediate (16-bit compressed instruction)
mul	707k	32bit to 32bit multiplication
srai	707k	signed arithmetic right shift immediate
c.mv	12k	register to register copy
others	41k	other instructions
	7'151k	total instructions
	9'283k	total #cycles

Table 3.5: Instruction and Cycle Count for the Benchmark Suite with no extensions

3.4.3 Next Steps

For the next steps in the project, we want to analyze more in details exact instruction patterns. E.g. instructions which follow each other very often and could be merged to a single instruction. Furthermore hyperbolic tangent and sigmoid take a high ratio of the instructions within LSTMs, GRU and RNN and the number of instruction can be reduced using piece-wise linear approximation, but we have to analyze the error we introduce by this approximation and how many pieces should be used.

Currently, the networks in [9] and [10] are not implemented because their parameters don't fit on-chip memory leading to external memory accesses. We will also implement

these networks with a virtually larger memory (i.e. ideal memory reference) and with external memory accesses (e.g. hyperRAM) and DMA transfer.

There is still a high overhead in the fixed-point operations, with the existing SIMD data types these fixed-point shifting and sign-extension should be reduced. Then we will explore low-bit data types and floating-point operations before we will finally extend the ISA based on the instruction patterns found in the evaluation. These ISA extensions will be implemented in RTL. Furthermore, the assembler and compiler will be extended such that simple deployment from networks in C is enabled on the extended RISC-V core.

Bibliography

[1] U. Challita, L. Dong, and W. Saad, "Proactive Resource Management in LTE-U Systems: A Deep Learning Perspective," 2 2017. [Online]. Available: http://arxiv.org/abs/1702.07031

[2] O. Naparstek and K. Cohen, "Deep Multi-User Reinforcement Learning for Distributed Dynamic Spectrum Access," 4 2017. [Online]. Available: http://arxiv.org/abs/1704.02613

[3] H. Ye and G. Y. Li, "Deep Reinforcement Learning for Resource Allocation in V2V Communications," *ieeexplore.ieee.org*, 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8422586/http://arxiv.org/abs/1711.00968

[4] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for wireless resource management," in *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 7 2017, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/8227766/

[5] E. Ghadimi, F. Davide Calabrese, G. Peters, and P. Soldati, "A reinforcement learning approach to power control and rate adaptation in cellular networks," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 5 2017, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/document/7997440/

[6] Y. Yu, T. Wang, and S. C. Liew, "Deep-Reinforcement Learning Multiple Access for Heterogeneous Wireless Networks," 11 2017. [Online]. Available: http://arxiv.org/abs/1712.00162

[7] M. Eisen, C. Zhang, L. Chamon, D. L. a. p. a. ..., and U. 2018, "Learning Optimal Resource Allocations in Wireless Systems," *arxiv.org*. [Online]. Available: https://arxiv.org/abs/1807.08088

[8] Y. S. Nasir and D. Guo, "Deep Reinforcement Learning for Distributed Dynamic Power Allocation in Wireless Networks," 8 2018. [Online]. Available: http://arxiv.org/abs/1808.00490

[9] K. I. Ahmed, H. Tabassum, and E. Hossain, "Deep Learning for Radio Resource Allocation in Multi-Cell Networks," 8 2018. [Online]. Available: http://arxiv.org/abs/1808.00667

[10] W. Lee, M. Kim, and D.-H. Cho, "Deep Power Control: Transmit Power Control Scheme Based on Convolutional Neural Network," *IEEE Communications Letters*, vol. 22, no. 6, pp. 1276–1279, 6 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8335785/

[11] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, "Deep reinforcement learning for dynamic multichannel access in wireless networks," *IEEE Transactions on Cognitive Communications and Networking*, 2018.

[12] Wikimedia Commons, "Recurrent Neural Network unfold," 2005.

[13] ——, "Recurrent Neural Network unfold," 2005.

[14] ——, "Gate Recurrent Unit," 2017.

[15] N. Cheema, S. Hosseini, J. Sprenger, E. Herrmann, H. Du, K. Fischer, and P. Slusallek, "Dilated Temporal Fully-Convolutional Network for Semantic Segmentation of Motion Capture Data," *arXiv preprint arXiv:1806.09174*, 2018. [Online]. Available: https://arxiv.org/abs/1806.09174

[16] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[17] A. A. Markov, "Research on common case of trials, connected in chain," *Zapiski Akademii Nauk po Fiziko-Matematicheskomu Otdeleniyu*, vol. 8, 1910.

[18] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3128–3137.

[19] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1017–1024.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[21] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[22] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu, "Neural machine translation in linear time," *arXiv preprint arXiv:1610.10099*, 2016. [Online]. Available: https://arxiv.org/pdf/1610.10099.pdf

[23] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[24] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4277–4280.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and others, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[26] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[27] N. D. Tripathi, J. H. Reed, and H. F. VanLandingham, *Radio resource management in cellular systems*. Springer Science & Business Media, 2006, vol. 618.

[28] Q. Shi, M. Razaviyayn, Z.-Q. Luo, and C. He, "An iteratively weighted MMSE approach to distributed sum-utility maximization for a MIMO interfering broadcast channel," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3060–3063.

[29] M. Naeem, K. Illanko, A. Karmokar, A. Anpalagan, and M. Jaseemuddin, "Optimal power allocation for green cognitive radio: fractional programming approach," *IET Communications*, vol. 7, no. 12, pp. 1279–1286, 2013.

[30] A. Galindo-Serrano and L. Giupponi, "Distributed Q-learning for aggregated interference control in cognitive radio networks," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 4, pp. 1823–1834, 2010.

[31] K.-L. A. Yau, P. Komisarczuk, and P. D. Teal, "A context-aware and intelligent dynamic channel selection scheme for cognitive radio networks," in *Cognitive Radio Oriented Wireless Networks and Communications, 2009. CROWNCOM'09. 4th International Conference on*. IEEE, 2009, pp. 1–6.

[32] T. K. Vu, M. Bennis, M. Debbah, M. Latva-aho, and C. S. Hong, "Ultra-Reliable Communication in 5G mmWave Networks: A Risk-Sensitive Approach," *IEEE Communications Letters*, vol. 22, no. 4, pp. 708–711, 2018.

[33] F. D. Calabrese, L. Wang, E. Ghadimi, G. Peters, and P. Soldati, "Learning radio resource management in 5G networks: Framework, opportunities and challenges," *arXiv preprint arXiv:1611.10253*, 2016.

[34] R. Amiri, H. Mehrpouyan, L. Fridman, R. K. Mallik, A. Nallanathan, and D. Matolak, "A Machine Learning Approach for Power Allocation in HetNets Considering QoS," *arXiv preprint arXiv:1803.06760*, 2018.

[35] M. Bennis and D. Niyato, "A Q-learning based approach to interference avoidance in self-organized femtocell networks," in *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*. IEEE, 2010, pp. 706–710.

[36] M. Simsek, A. Czylwik, A. Galindo-Serrano, and L. Giupponi, "Improved decentralized Q-learning algorithm for interference reduction in LTE-femtocells," in *Wireless Advanced (WiAd), 2011*. IEEE, 2011, pp. 138–143.

[37] C. Touati, E. Altman, and J. Galtier, "Utility based fair bandwidth allocation," *Proc. of IASTED NPDPA*, pp. 126–131, 2002.