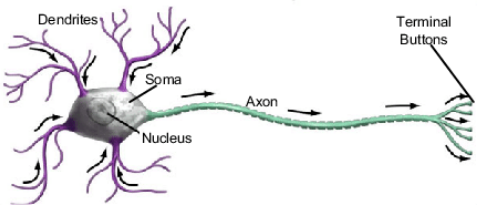
Poupakis Alexandros
Electrical & Computer Engineering School
Technical University of Crete

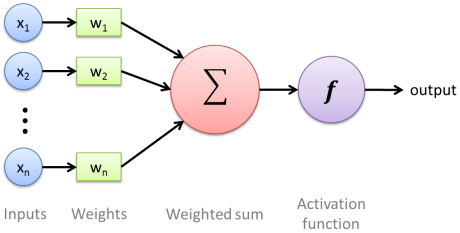
Compression of Weights of Recurrent Neural Network for Speech Recognition Acceleration in Reconfigurable Hardware (FPGA)

Diploma Thesis

Table of Contents

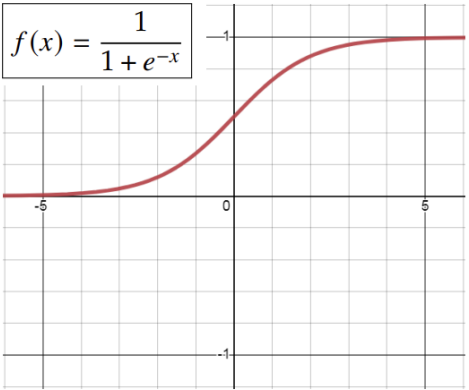
From biology to computing





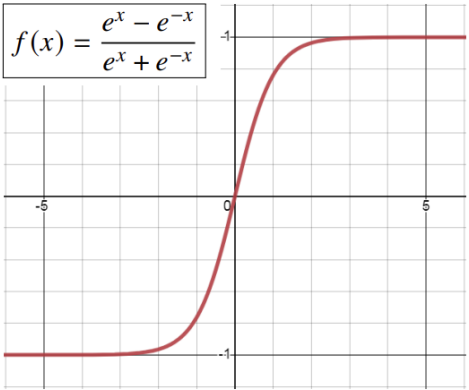
Activation functions

$$f(x) = \frac{1}{1 + e^{-x}}$$



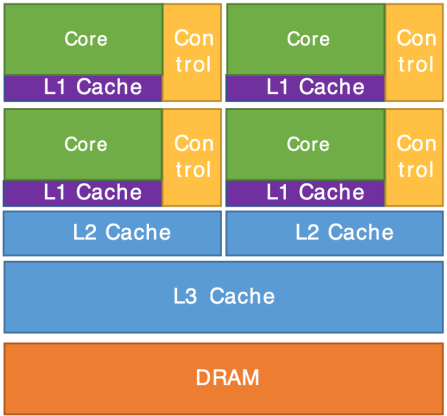
Sigmoid (σ)

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

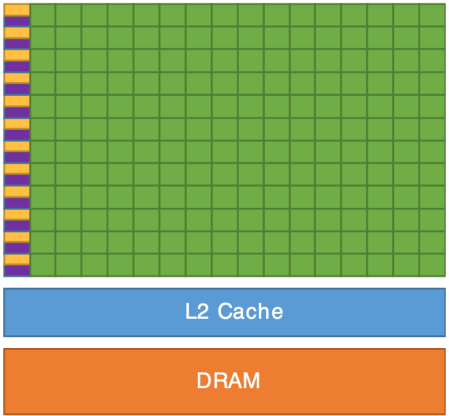


Hyperbolic tangent (tanh)

Computing platforms

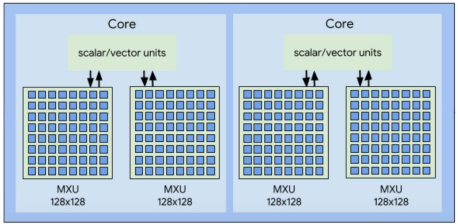


CPU

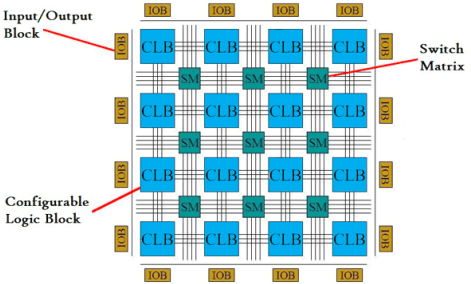


GPU

Computing platforms



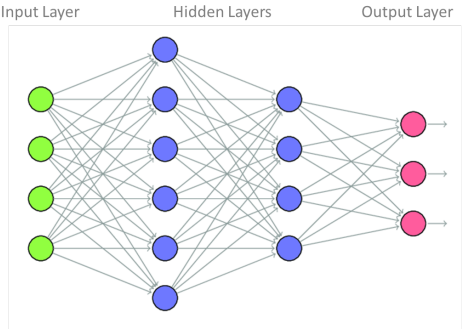
TPU v3



FPGA

Neural Networks

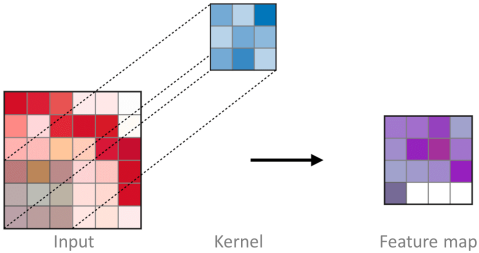
Feedforward Neural Networks



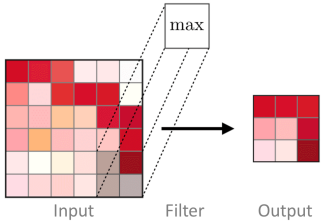
Fully Connected Network

$$\mathbf{h}_1 = f_1\left(\mathbf{x}\mathbf{W}_{ih_1} + \mathbf{b}_{h_1}\right)$$
$$\mathbf{h}_i = f_i\left(\mathbf{h}_{i-1}\mathbf{W}_{h_{i-1}h_i} + \mathbf{b}_{h_i}\right) \quad 1 \leq i \leq k$$
$$\mathbf{o} = g\left(\mathbf{h}_k\mathbf{W}_{h_k o} + \mathbf{b}_o\right)$$

Feedforward Neural Networks



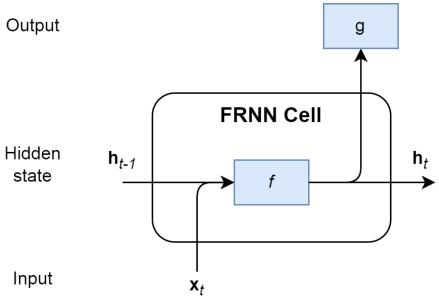
Convolution layer



Pooling layer

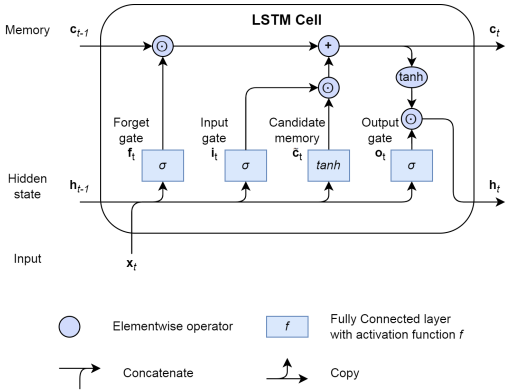
Convolutional Neural Networks

Fully Recurrent Neural Network



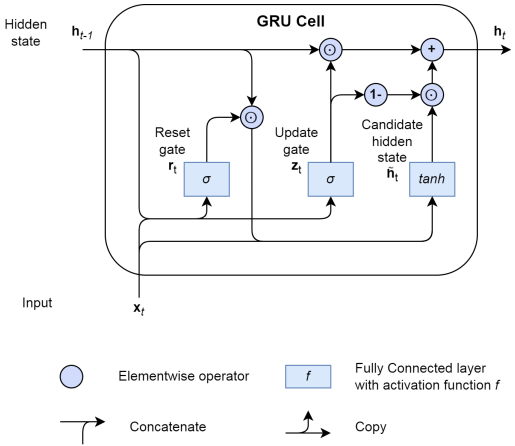
$$\mathbf{h}_t = f\left(\mathbf{x}_t \mathbf{W}_{xh} + \mathbf{h}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h\right)$$
$$\mathbf{o}_t = g\left(\mathbf{h}_t \mathbf{W}_{ho} + \mathbf{b}_o\right)$$

Long Short Term Memory



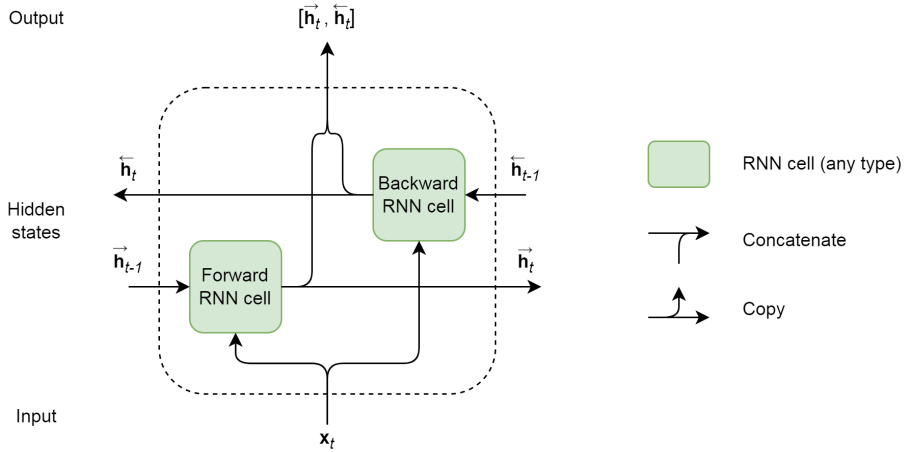
$$\mathbf{i}_t = \sigma\left(\mathbf{x}_t\mathbf{W}_{xi} + \mathbf{h}_{t-1}\mathbf{W}_{hi} + \mathbf{b}_i\right)$$
$$\mathbf{f}_t = \sigma\left(\mathbf{x}_t\mathbf{W}_{xf} + \mathbf{h}_{t-1}\mathbf{W}_{hf} + \mathbf{b}_f\right)$$
$$\mathbf{o}_t = \sigma\left(\mathbf{x}_t\mathbf{W}_{xo} + \mathbf{h}_{t-1}\mathbf{W}_{ho} + \mathbf{b}_o\right)$$
$$\tilde{\mathbf{c}}_t = \tanh\left(\mathbf{x}_t\mathbf{W}_{xc} + \mathbf{h}_{t-1}\mathbf{W}_{hc} + \mathbf{b}_c\right)$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right)$$

Gated Recurrent Unit



$$\mathbf{r}_t = \sigma\left(\mathbf{x}_t \mathbf{W}_{xr} + \mathbf{h}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r\right)$$
$$\mathbf{z}_t = \sigma\left(\mathbf{x}_t \mathbf{W}_{xz} + \mathbf{h}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z\right)$$
$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{x}_t \mathbf{W}_{xh} + \left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right) \mathbf{W}_{hh} + \mathbf{b}_h\right)$$
$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + \left(1 - \mathbf{z}_t\right) \odot \tilde{\mathbf{h}}_t$$

Bidirectional Recurrent Neural Networks



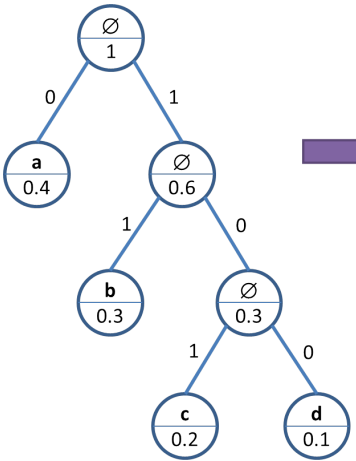
Data Compression

What is Data Compression

- Reduce bits
- Exploit statistical redundancy
- Shannon's entropy limit

$$H(X) = -\sum_{x \in \mathcal{X}} P(x) \log P(x)$$

Huffman Coding



Codebook

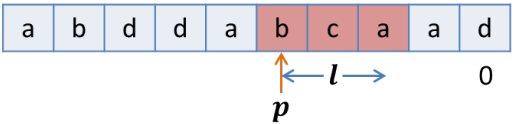
a	0
b	11
c	101
d	100

Sequence:	**b**	**a**	**c**	**b**	**d**
Encoding:	11	0	101	11	100

LZ77 Coding

Encode via triplets: $(\boldsymbol{p}, \boldsymbol{l}, \boldsymbol{c})$

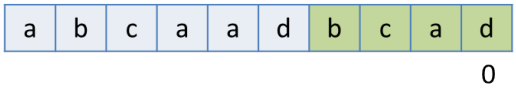
Dictionary



Sequence: ... | **b c a d** | ...

Encoding: ... | (4, 3, d) | ...

Dictionary after decoding (4, 3, d)



Related Work

ASR Neural Network Architectures

Automatic Speech Recognition: human speech $\longrightarrow$ transcribed text

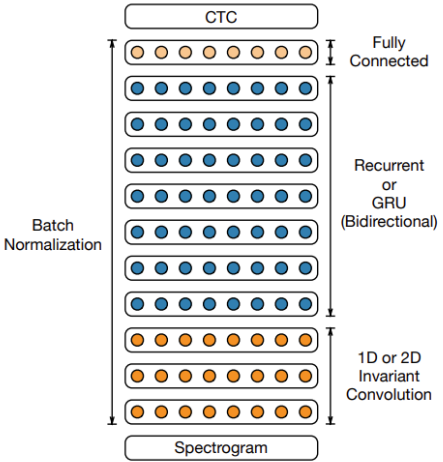
- CTC-based
  - 2014, First attempt with LSTMs
  - 2015, DeepSpeech2
- Transformer-based
  - 2016, Listen, Attend and Spell

Neural Network Compression

- Pruning
  Layer, Channel, Filter, Connection
- Sparse representation
  Quantization, Weight sharing
- Precision reduction
  Low-bit representation, Estimation via integers, Binarization
- Knowledge distillation

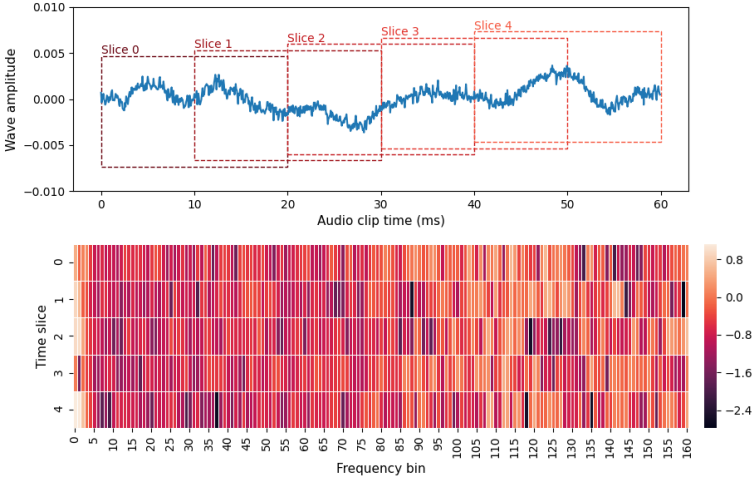
Training & Robustness Analysis

DeepSpeech2 Model



- Input: spectrogram timeseries
- 2 CNN layers
- 5 bidirectional GRU layers
- 1 FC layer
- Output: letter prediction timeseries

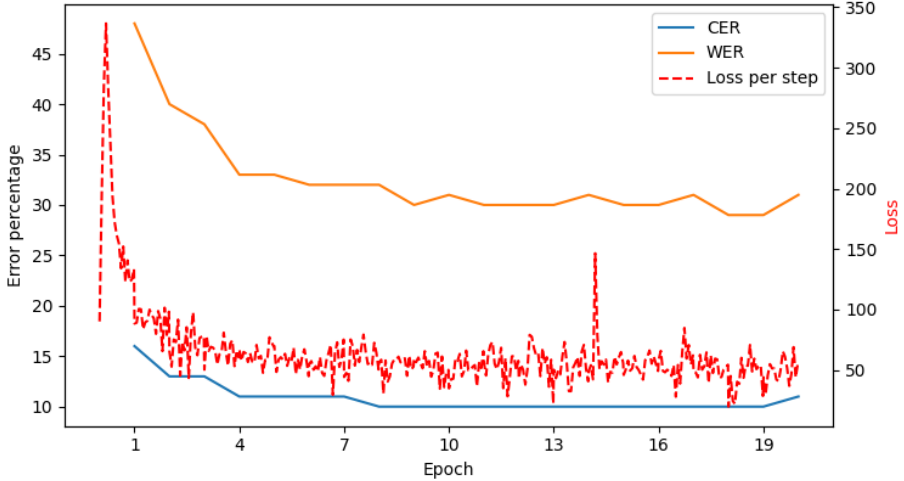
Input Spectrogram Example



Training Parameters

- 2 GTX 2080 Ti GPUs
- LibriSpeech dataset
- 363.5 hours clean speech for training
- 5.5 hours clean speech for validation
- 20 epochs
- Constant learning rate

Training Results



GRU Cell Parameter Categories

Input/Output		Reset gate		Update gate		Candidate state	
Tensor	Size	Tensor	Size	Tensor	Size	Tensor	Size
x_t	$(1, D)$	W_{xr}	(D, H)	W_{xz}	(D, H)	W_{xh}	(D, H)
h_t	$(1, H)$	W_{hr}	(H, H)	W_{hz}	(H, H)	W_{hh}	(H, H)
		Gate kernels				Cand. kernels	
		b_r	$(1, H)$	b_z	$(1, H)$	b_h	$(1, H)$
		Gate biases				Cand. bias	
		r_t	$(1, H)$	z_t	$(1, H)$	h_t	$(1, H)$

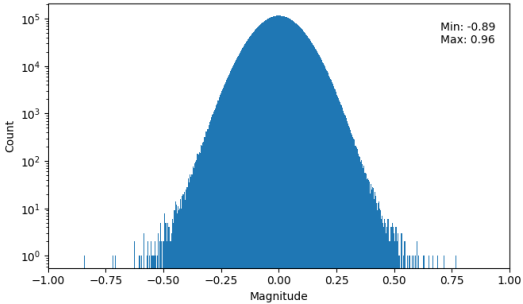
DeepSpeech2 Parameters

Total parameters	CNN	GRU	FC
62683005	250976	62385600	46429
100%	0.40%	99.53%	0.07%

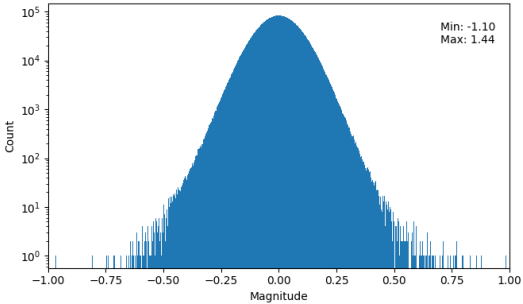
DeepSpeech2 GRU Parameters

Layer	Gate		Candidate	
	Kernel	Bias	Kernel	Bias
GRU 1	10854400	3200	5427200	1600
GRU 2	7680000	3200	3840000	1600
GRU 3	7680000	3200	3840000	1600
GRU 4	7680000	3200	3840000	1600
GRU 5	7680000	3200	3840000	1600
Total	41574400 66.64%	16000 0.03%	20787200 33.32%	8000 0.01%

Weight Distributions



Gate kernels



Candidate kernels

Pruning

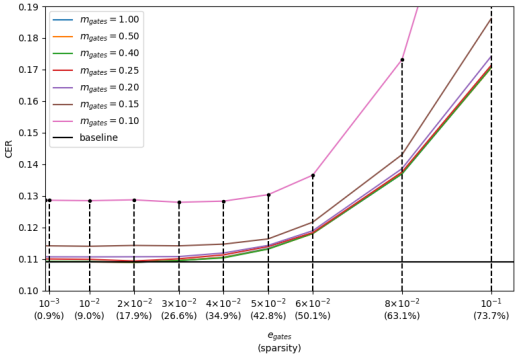
- Reduce the number of weights
- Magnitude pruning
- Weight domain

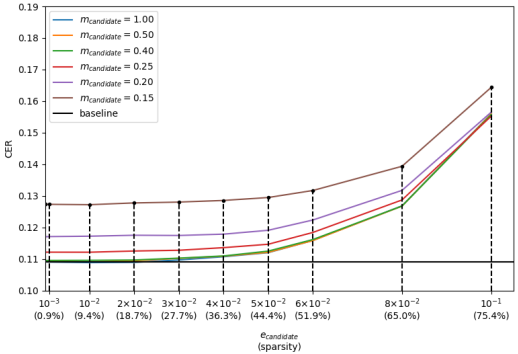
$$\mathbb{W} = [-\mathrm{m}, -e] \cup [e, \mathrm{m}]$$

- Policy

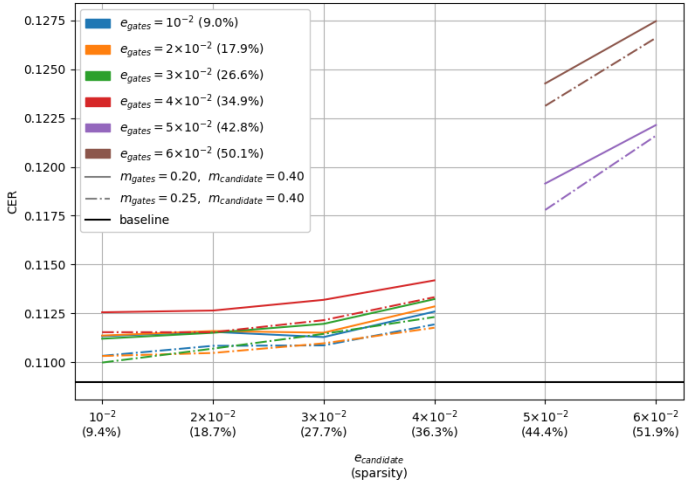
$$w \leftarrow \begin{cases} 0 & \text{if } |w| < e \\ \operatorname{sign}(w) \cdot \mathrm{m} & \text{if } |w| > \mathrm{m} \\ w & \text{otherwise} \end{cases}$$

Individual Global Pruning

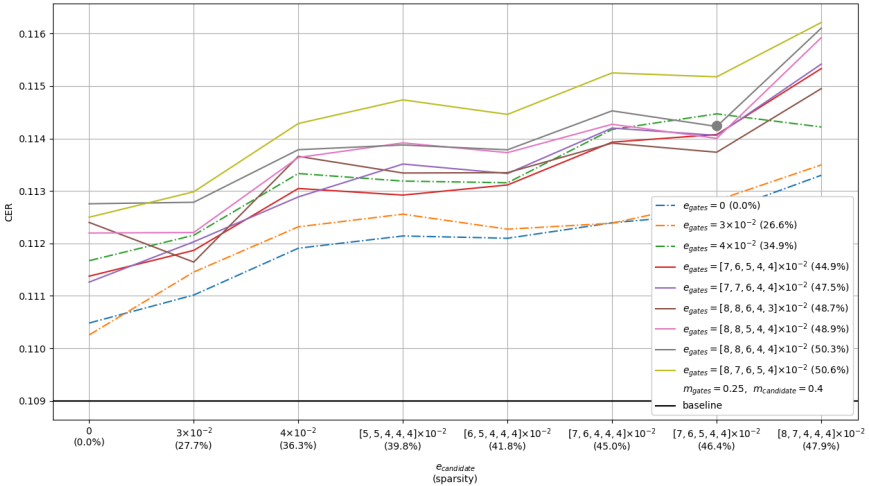




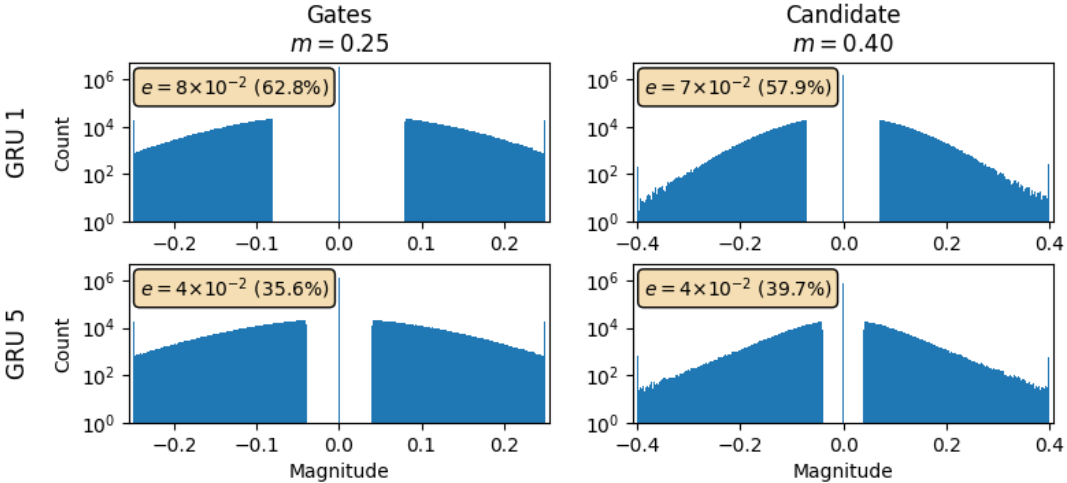
Joint Global Pruning



Joint Layer-wise Pruning



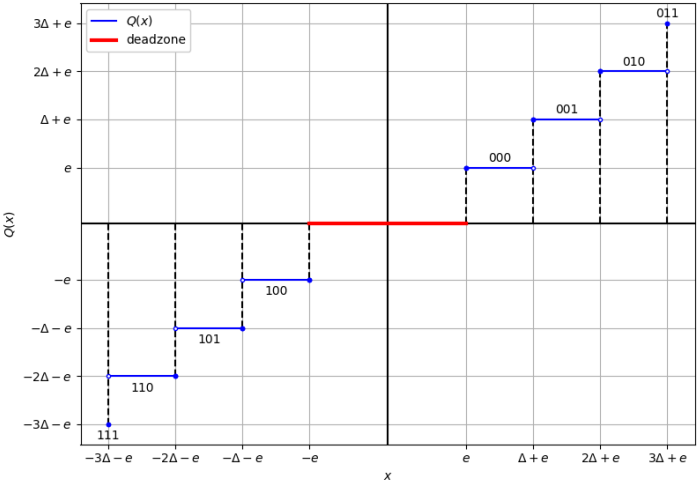
Pruned Weight Distributions



Quantization

- Reduce the number of bits per weight
- Discretize weight domain
- Applied on top of pruning

Uniform Dead-zone Quantizer



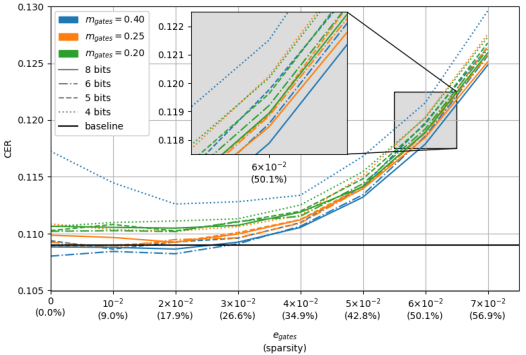
- 2^n quantization levels
- Domain

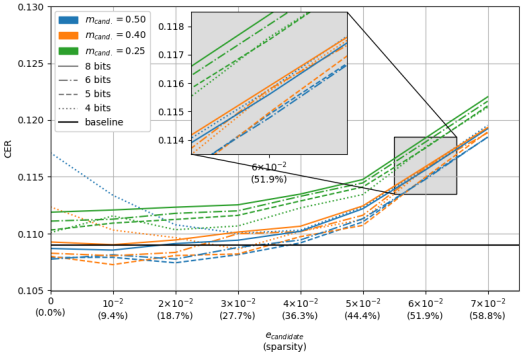
$$x \in [-m, -e] \cup [e, m]$$

- Quantization step size

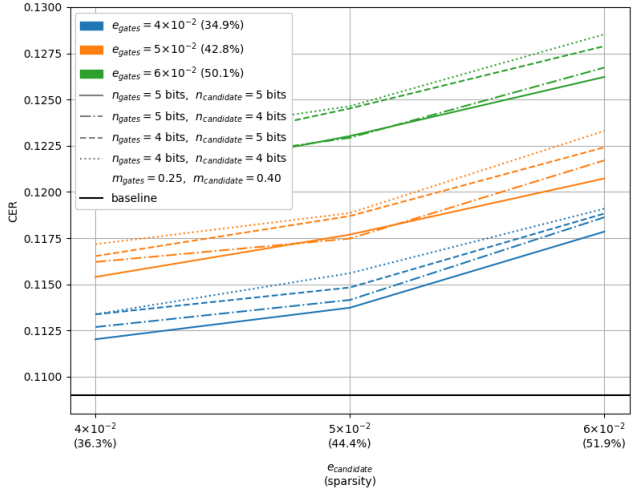
$$\Delta = \frac{m - e}{2^{n-1} - 1}$$

Individual Uniform Quantization with Global Pruning

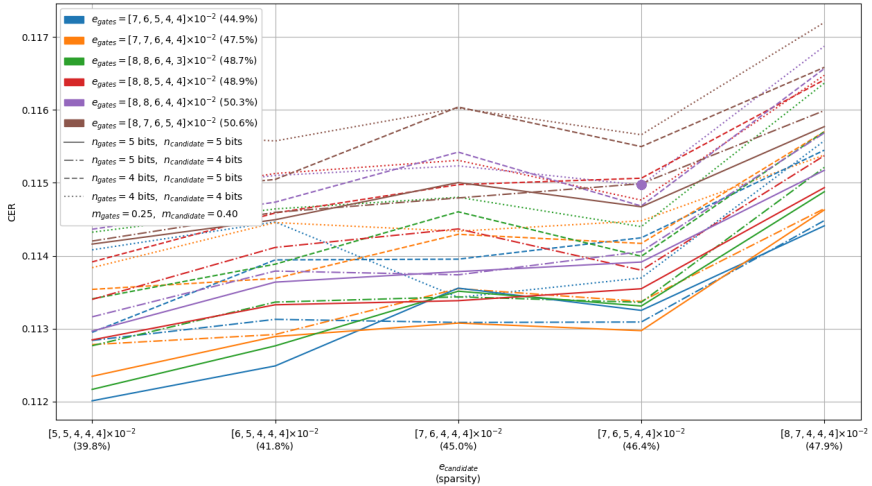




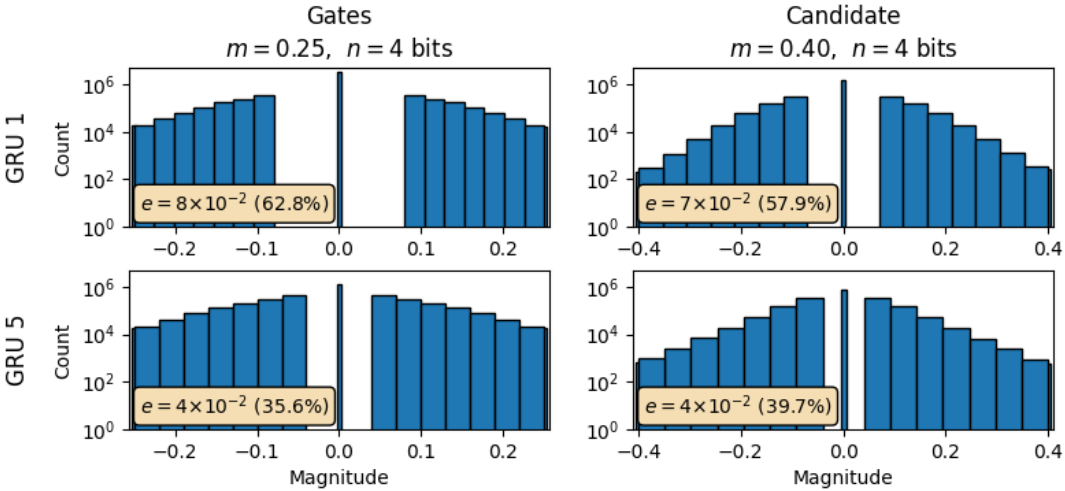
Joint Uniform Quantization with Global Pruning



Joint Uniform Quantization with Layer-wise Pruning



Pruned and Quantized Weight Distributions



K-means Quantization

- K-means quantizer
- Repeated experiments
- Consistently worse compared to Uniform

Robustness Analysis Recap

- Baseline model
  10.90% CER
- Layer-wise pruning
  11.42% CER, 49.0% sparsity
- Uniform dead-zone quantization
  11.49% CER, 49.0% sparsity, 4-bit weights
- Dense matrices $\longrightarrow$ Sparse matrices

Proposed Compression Method

Why use Data Compression?

- Sparse matrix overhead
- 49% sparsity is **very** low!
- The overhead is larger than the quantized weights...

Why invent a novel method?

- Huffman decoding is very slow
- LZ77 variants require a lot of hardware
- We need high-throughout, low-cost decompression

Prerequisites

- Base-symbol alphabet $\mathcal{X}$
- Source output

$$x_0 x_1 x_2 ... \qquad x_i \in \mathcal{X}$$

- New symbol alphabet

$$\mathcal{Z} := \{0 ... |\mathcal{X}| - 1\}$$

- Mapping from $\mathcal{X}$ to $\mathcal{Z}$

$$y_i \mapsto i \qquad \forall y_i \in \mathcal{X}, i \in \mathcal{Z}$$

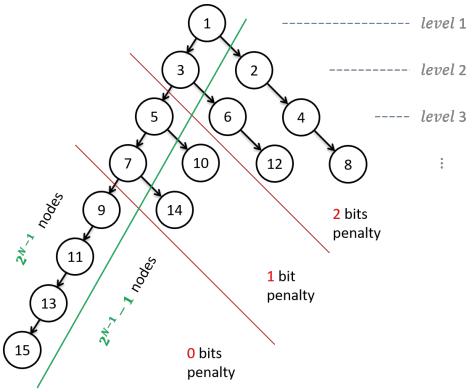
- Transformed source output

$$z_0 z_1 z_2 ... \qquad z_i \in \mathcal{Z},\ y_{z_i} = x_i$$

What does this method do?

- Compress constant-length sequences of symbols in $\mathcal{Z}$
- Store overlapping sequences in a tree
- Provide variable-length encodings and mathematical decoding
- Retrieve sequences by traversing the tree upwards

Data Structure



- N-bit tree
- Descendant / ancestor functions

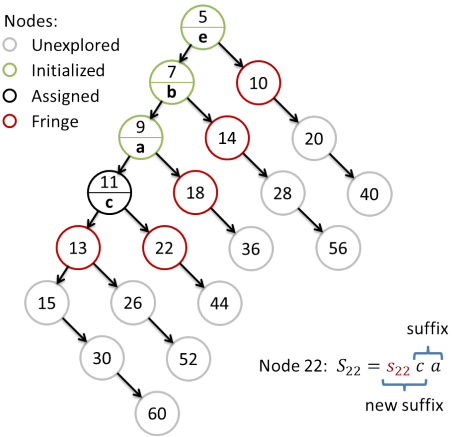
$$children(n) := \begin{cases} \{2n, n+2\} & \text{if } n \text{ is odd} \\ \{2n\} & \text{if } n \text{ is even} \end{cases}$$

$$parent(n) := \begin{cases} n-2 & \text{if } n \text{ is odd} \\ n/2 & \text{if } n \text{ is even} \end{cases}$$

- Penalty groups

$$PGs := \lceil \log_2 N \rceil + 1$$

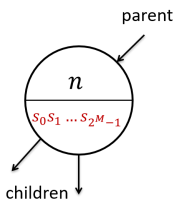
How do we assign symbols to nodes?



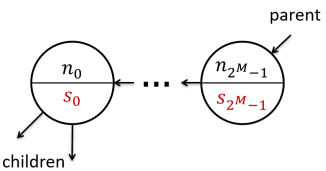
- Per penalty group
- Maximizing overlap
- Comparing the node's number of children to the new suffix's available symbols
- Constructing the most frequent (sub)sequence

Generalization

2^M symbols per node:

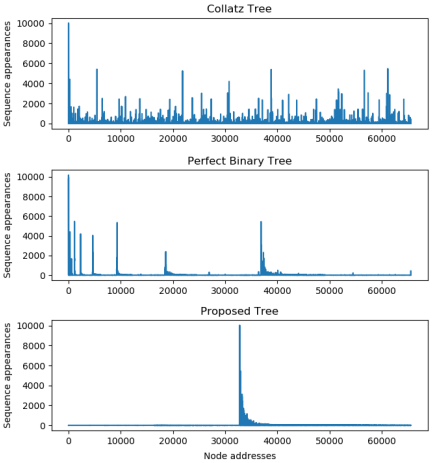


1 symbol per node equivalent expansion:

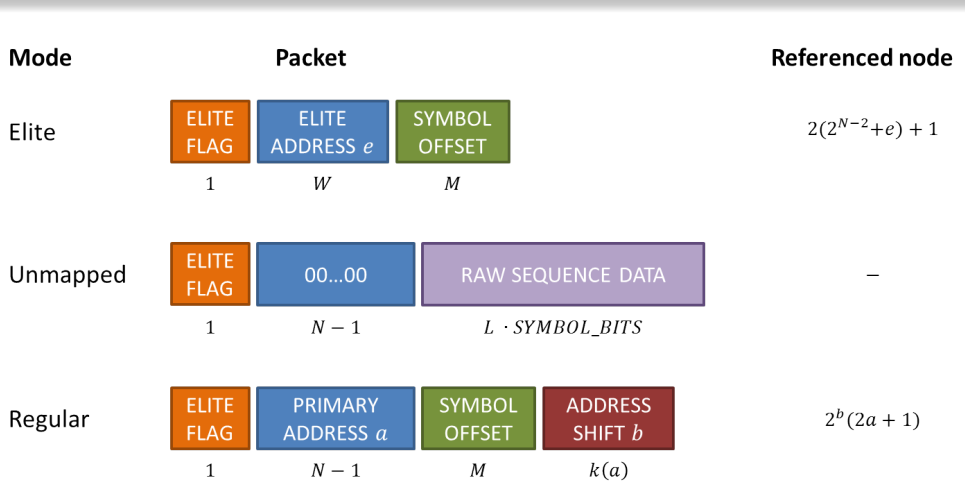


- 2^M symbols per node
- Single memory access
- Use the same assignment algorithm

Benefit of our Tree



Tree Addressing Modes



Decompression Throughput

- Depending on the symbol offset, the cycles to decompress a packet vary

$$\left\lceil \frac{L}{2^M} \right\rceil \text{ or } \left\lceil \frac{L}{2^M} \right\rceil + 1$$

- Expected cycles per packet

$$E_D = \frac{L + 2^M - 1}{2^M}$$

- Average decompression throughput

$$\mathcal{T} := \frac{L}{E_D} \frac{symbols}{cycle} = \frac{2^M \cdot L}{L + 2^M - 1} \frac{symbols}{cycle}$$

Sparse Matrix Encoding

a	0	0	-a	0
b	a	0	0	0
0	-b	0	0	c
0	0	0	a	0

Raw sparse matrix

⇒

0	0	3	0	5	2	2	1

Zeros (run-length encoding) vector

a	b	a	-b	-a	a	c	∅

Weights vector

Column-wise encoding

Encoded Matrix Statistics

Layer	Kernel	Size (each)	Memory (MB)		Sparsity
			Weights	Zeros	
1	Gates	3392×1600	3.147	3.933	63%
	Candidate	3392×800			58%
2	Gates	2400×1600	2.409	3.012	62%
	Candidate	2400×800			50%
3	Gates	2400×1600	3.018	3.773	50%
	Candidate	2400×800			43%
4	Gates	2400×1600	3.691	4.615	35%
	Candidate	2400×800			36%
5	Gates	2400×1600	3.635	4.544	36%
	Candidate	2400×800			39%
Total			**15.900**	**19.877**	

Compression Results for Zeros

		Proposed method				Entropy limit	
Layer	L	$\mathcal{T}$ $\left(\frac{\text{symbols}}{\text{cycle}}\right)$	Compr. data (MB)	Memory reduction	Tree (KiB)	Compr. data (MB)	Memory reduction
1	6	1.72	2.204	44.0%	80	1.908	51.5%
2	6	1.72	1.538	48.9%	80	1.379	54.2%
3	7	1.75	1.598	57.6%	80	1.412	62.6%
4	9	1.80	1.610	65.1%	80	1.325	71.3%
5	9	1.80	1.666	63.3%	80	1.329	70.7%
Total			**8.616**	**56.7%**	**400**	**7.353**	**63.0%**

Compression Results for Signed Weights

		Proposed method				Entropy limit	
Layer	L	$\mathcal{T}$ $\left(\frac{\text{symbols}}{\text{cycle}}\right)$	Compr. data (MB)	Memory reduction	Tree (KiB)	Compr. data (MB)	Memory reduction
1	4	1.60	2.775	11.8%	32	2.637	16.2%
2	4	1.60	2.119	12.0%	32	2.015	16.4%
3	4	1.60	2.658	11.9%	32	2.528	16.2%
4	4	1.60	3.250	11.9%	32	3.084	16.4%
5	4	1.60	3.209	11.7%	32	3.044	16.3%
Total			**14.011**	**11.9%**	**160**	**13.308**	**16.3%**

Compression Results for Sign-separated Weights

		Proposed method				Entropy limit	
Layer	L	$\mathcal{T}$ $\left(\frac{\text{symbols}}{\text{cycle}}\right)$	Compr. data (MB)	Memory reduction	Tree (KiB)	Compr. data (MB)	Memory reduction
1	6	1.72	2.766	12.1%	48	2.618	16.8%
2	6	1.72	2.103	12.7%	48	2.001	16.9%
3	6	1.72	2.634	12.7%	48	2.513	16.7%
4	6	1.72	3.220	12.8%	48	3.068	16.9%
5	6	1.72	3.191	12.2%	48	3.025	16.8%
Total			**13.914**	**12.5%**	**240**	**13.224**	**16.8%**

Compression Recap

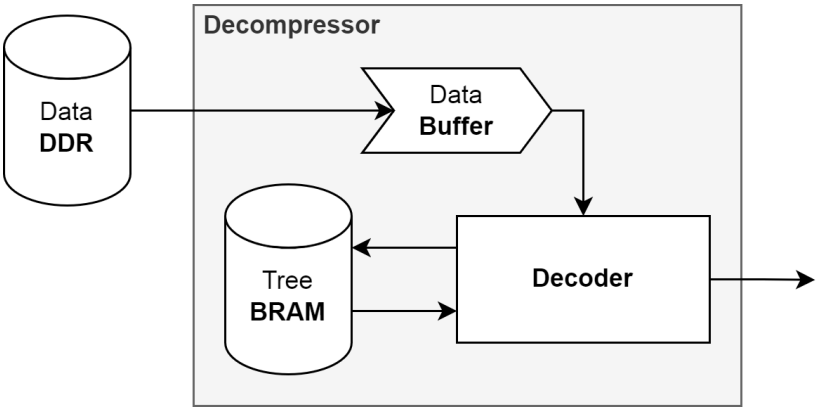
Description	Methods used	Data type	Memory (MB)	Reduction
Original	-	32-bit float	249.5	-
Pruned & quantized	LMP+Uniform	4-bit weights 5-bit zeros	35.7	7x
Compressed	LMP+Uniform +PATH	4-bit weights 5-bit zeros	22.5	11x

Hardware Architecture

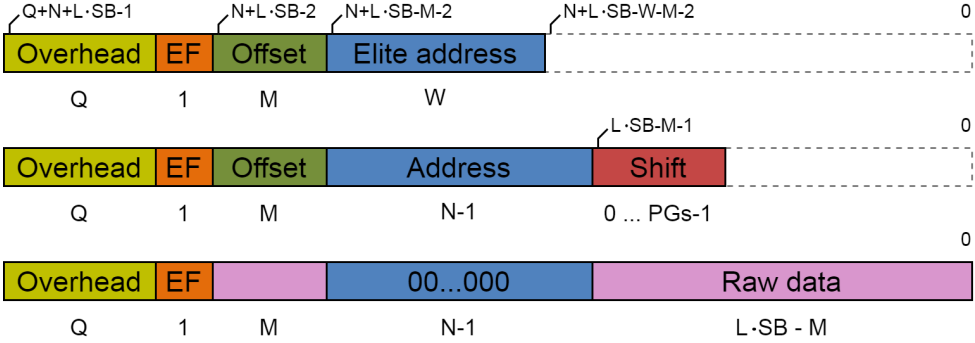
FPGA Resources

- LookUp Tables (LUTs)
- Flip Flops (FFs)
- Block RAMs (BRAMs)
- External memory (DDR)

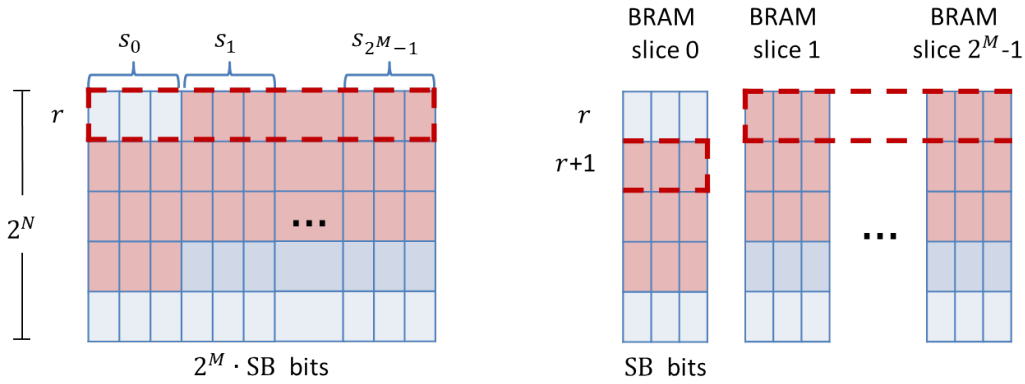
Decompressor Architecture



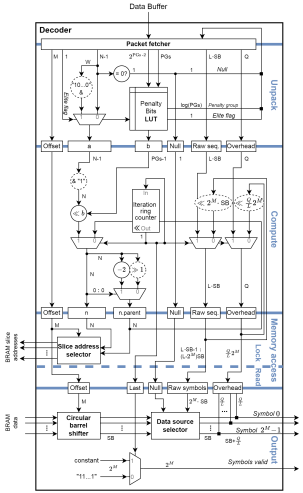
Packet Layouts



BRAM Organization

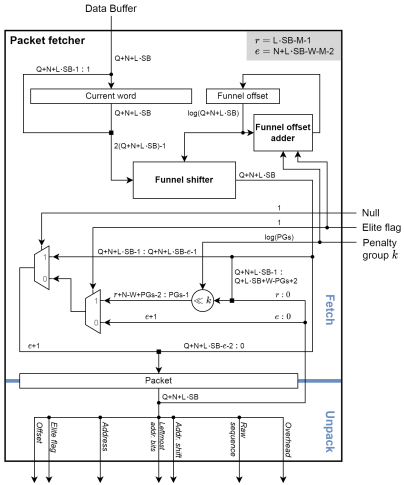


Decoder Pipeline



- Fetch
  Aligns packets from buffer
- Unpack
  Extracts packet fields
- Compute
  Calculates node address
- Memory Access
  Reads symbols from sliced memory
- Output
  Outputs the rotated read symbols or the raw symbols

Packet Fetcher

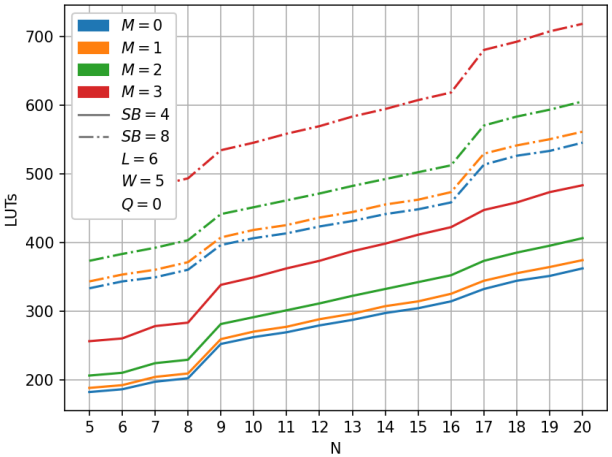


Resource Cost Model

- Analytical LUT and FF cost for every component of the decoder
- Equations are based on the design parameters

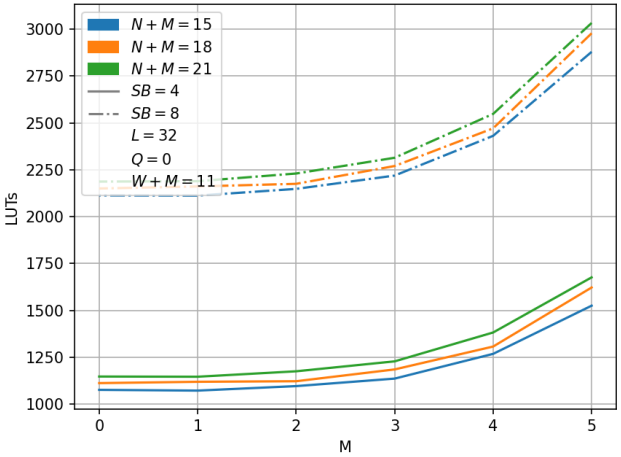
L	Sequence length
N	Address width
M	2^M symbols per node
W	Elite window
SB	Bits per symbol
Q	Packet overhead

Decoder's LUT Cost



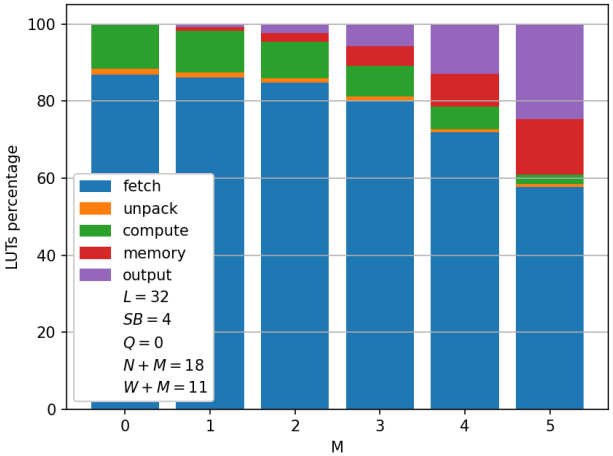
Linear w.r.t N

Decoder's LUT Cost with Constrained Tree Size



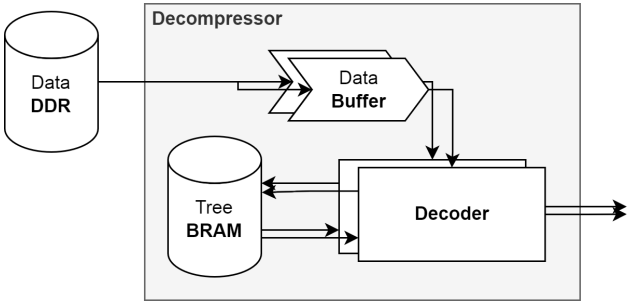
Exponential w.r.t M

Decoder's LUT Cost per Stage



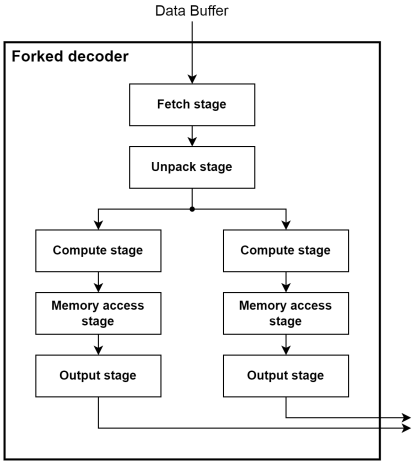
Fetch stage is by far the most expensive

Two Decoders



- Use both BRAM ports
- Full decoder logic duplication

Forked Pipeline Decoder



- Use both BRAM ports
- Reuse the most expensive stage
- Suitable when $\mathrm{E_D} > 1$

Decoder's Cost for Zeros

Layer	$M = 0$			$M = 1$			$M = 2$		
	LUTs	FFs	$\mathcal{T}\left(\frac{\text{symbols}}{\text{cycle}}\right)$	LUTs	FFs	$\mathcal{T}\left(\frac{\text{symbols}}{\text{cycle}}\right)$	LUTs	FFs	$\mathcal{T}\left(\frac{\text{symbols}}{\text{cycle}}\right)$
1	355	237	1.00	350	251	2.00	371	282	3.00
2	355	237	1.00	350	251	2.00	371	282	3.00
3	387	257	1.00	377	271	1.75	405	302	3.50
4	447	297	1.00	437	311	1.80	465	342	3.00
5	447	297	1.00	437	311	1.80	465	342	3.00

Hypothetical Comparison with Snappy Architectures

	Literature				This thesis			
Design	**LUTs**	**FFs**	**BRAMs**	$\mathcal{T}$	**LUTs**	**FFs**	**BRAMs**	$\mathcal{T}$
Xilinx	15.3K	16.5K	48	3.3	1.1K	0.8K	18	4
Qiao et al. 2018	91K	8.9K	32	15	1.4K	1.1K	18	16
Fang et al. 2020	56K		50	30.9	2.8K	2.0K	18	32

Hypothetical Comparison with Deflate Architectures

	Literature				This thesis			
Design	**LUTs**	**FFs**	**BRAMs**	$\mathcal{T}$	**LUTs**	**FFs**	**BRAMs**	$\mathcal{T}$
CAST Inc.	4.3K		9	3	1.1K	0.8K	18	4
Lewdon et al. 2020	15.7K	9.1K	15	2.2	1.1K	0.8K	18	4
Gao et al. 2022	308K	196K	128	62.4	10.2K	5.9K	18	64

Conclusions & Future Work

Conclusions

- Inference acceleration needs weight compression
- Proposed compression method is within 9.5% of the entropy limit
- Decompressor’s architecture is fast and resource efficient

Future Work

- Non-fluctuating decompression throughput
- Sequence-parallel decompressor architecture
- Extend method to variable-length sequences
- Cost-benefit analysis of weight decompressor in inference accelerators

Thesis Contributions

- Comprehensive robustness analysis of DeepSpeech2
- Novel compression method suitable for FPGAs
- Low-cost decompressor architecture
- 11x reduction of GRU kernel's memory using network and weight compression
- 4-65x lower LUT cost compared to other decompressors for the same hypothetical throughput