Cambricon

An Instruction Set Architecture for Neural Networks

Presented by

韦清 PB15000027

CONTENT

1

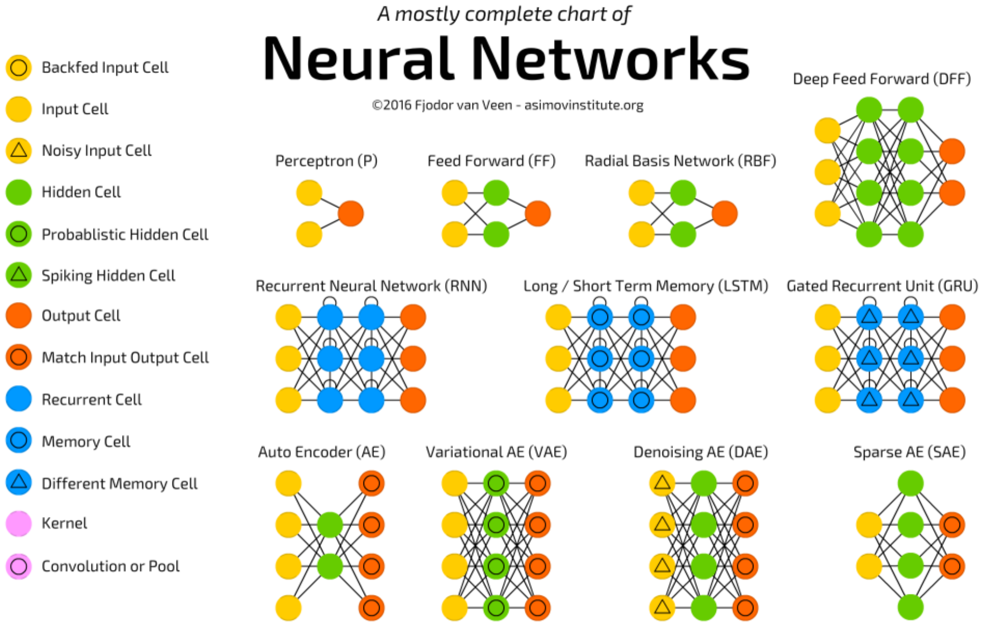
PART ONE

BACKGROUND

Application and prevailing of Neural Networks and
exsiting hardware accelerators for NN techniques.

BACKGROUND

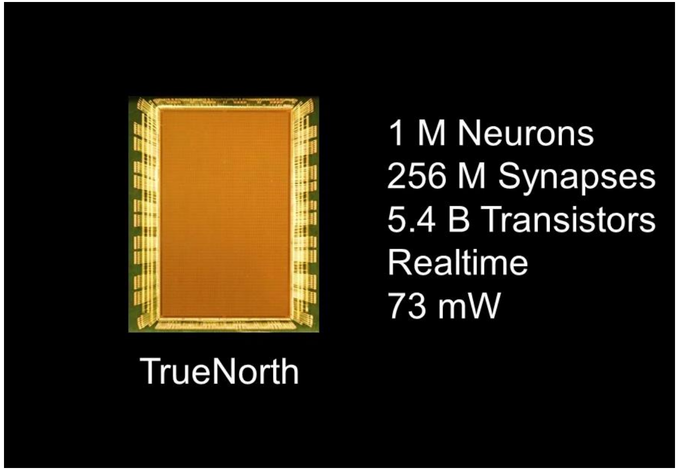
Artificial Neural Network



- Artificial neural networks (ANNs) are computing systems vaguely inspired by the biological neural networks that constitute animal brains.
- Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

BACKGROUND

Existing NN accelerators



Tensor Processing Unit (TPU)

- **30-80x** TOPS/watt vs. 2015 CPUs and GPUs.
- 8 GiB DRAM.
- 8-bit fixed point.
- 256x256 MAC unit.
- Support for data reordering, matrix multiply, activation, pooling, and normalization.

Figure 3. TPU Printed Circuit Board. It can be inserted in the slot for an SATA disk in a server, but the card uses PCIe Gen3 x16.

2

PART TWO

OVERVIEW

Cambricon is a novel domain-specific ISA for NN accelerators, based on a comprehensive analysis of existing NN techniques.

OVERVIEW OF CAMRICON

Design guidelines

01. RISC ISA

- Load-Store architecture
- Simple and short instructions

02. Data-Level Parallelism

DLP enabled by vector/matrix instructions can be more efficient than ILP of traditional scalar instructions, and corresponds to higher code density

Design Guidelines

03. Customized Vector/Matrix Instructions

- For NN techniques, fundamental operations defined in existing algebra libraries are not necessarily effective and efficient choices.
- Many common operations of NN techniques that are not covered by traditional linear algebra libraries

04. Using On-Chip Scratchpad Memory

NN techniques often require intensive, contiguous, and variable-length accesses to vector/matrix data

OVERVIEW OF CAMRICON

Performance

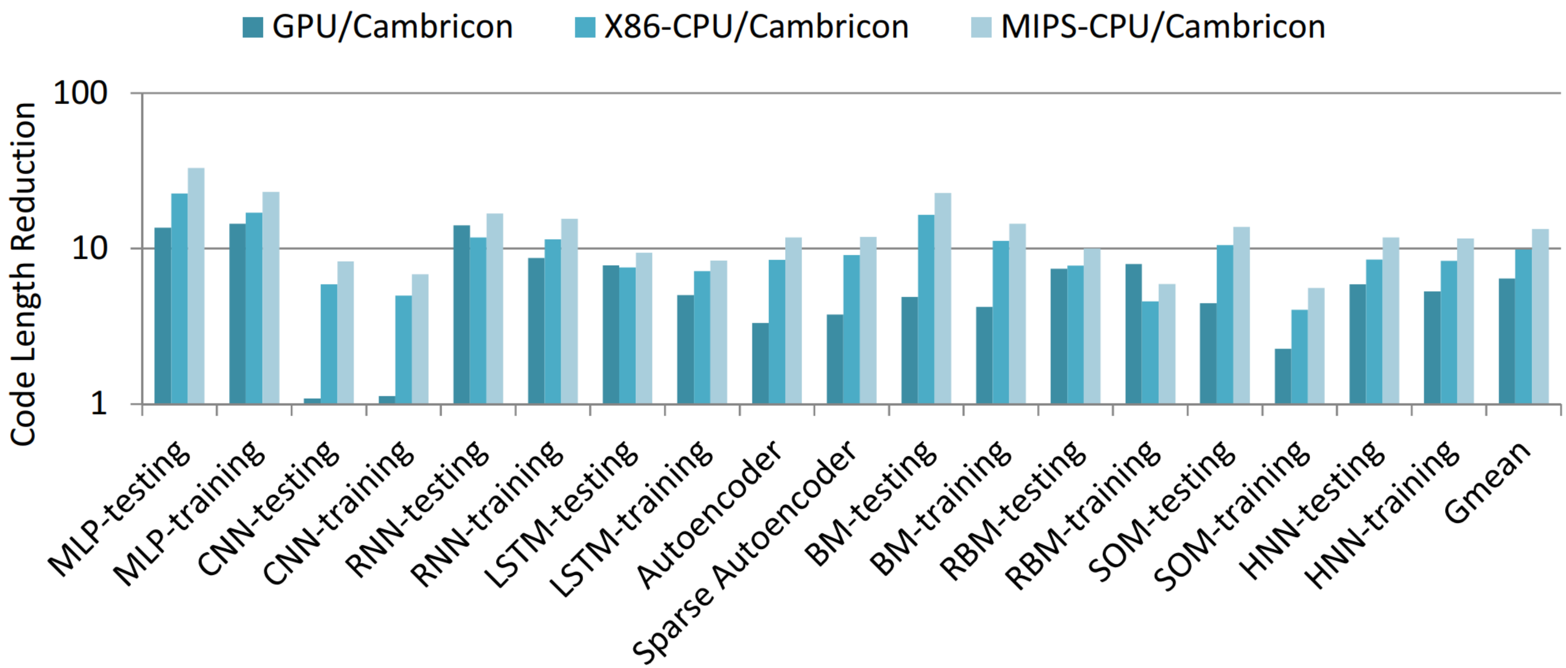
- 10 distinct NN techniques benchmarks:

Technique	Network Structure	Description
MLP	input(64) - H1(150) - H2(150) - Output(14)	Using Multi-Layer Perceptron (MLP) to perform anchorperson detection. [2]
CNN	input(1@32x32) - C1(6@28x28, K: 6@5x5) - S1(6@14x14, K: 2x2) - C2(16@10x10, K: 16@5x5) - S2(16@5x5, K: 2x2) - F(120) - F(84) - output(10)	Convolutional neural network (LeNet-5) for hand-written character recognition. [28]
RNN	input(26) - H(93) - output(61)	Recurrent neural network (RNN) on TIMIT database. [15]
LSTM	input(26) - H(93) - output(61)	Long-short-time-memory (LSTM) neural network on TIMIT database. [15]
Autoencoder	input(320) - H1(200) - H2(100) - H3(50) - Output(10)	A neural network pretrained by auto-encoder on MNIST data set. [49]
Sparse Autoencoder	input(320) - H1(200) - H2(100) - H3(50) - Output(10)	A neural network pretrained by sparse auto-encoder on MNIST data set. [49]
BM	V(500) - H(500)	Boltzmann machines (BM) on MINST data set. [39]
RBM	V(500) - H(500)	Restricted boltzmann machine (RBM) on MINST data set. [39]
SOM	input data(64) - neurons(36)	Self-organizing maps (SOM) based data mining of seasonal flu. [48]
HNN	vector (5), vector component(100)	Hopfield neural network (HNN) on hand-written digits data set. [36]

OVERVIEW OF CAMRICON

Performance

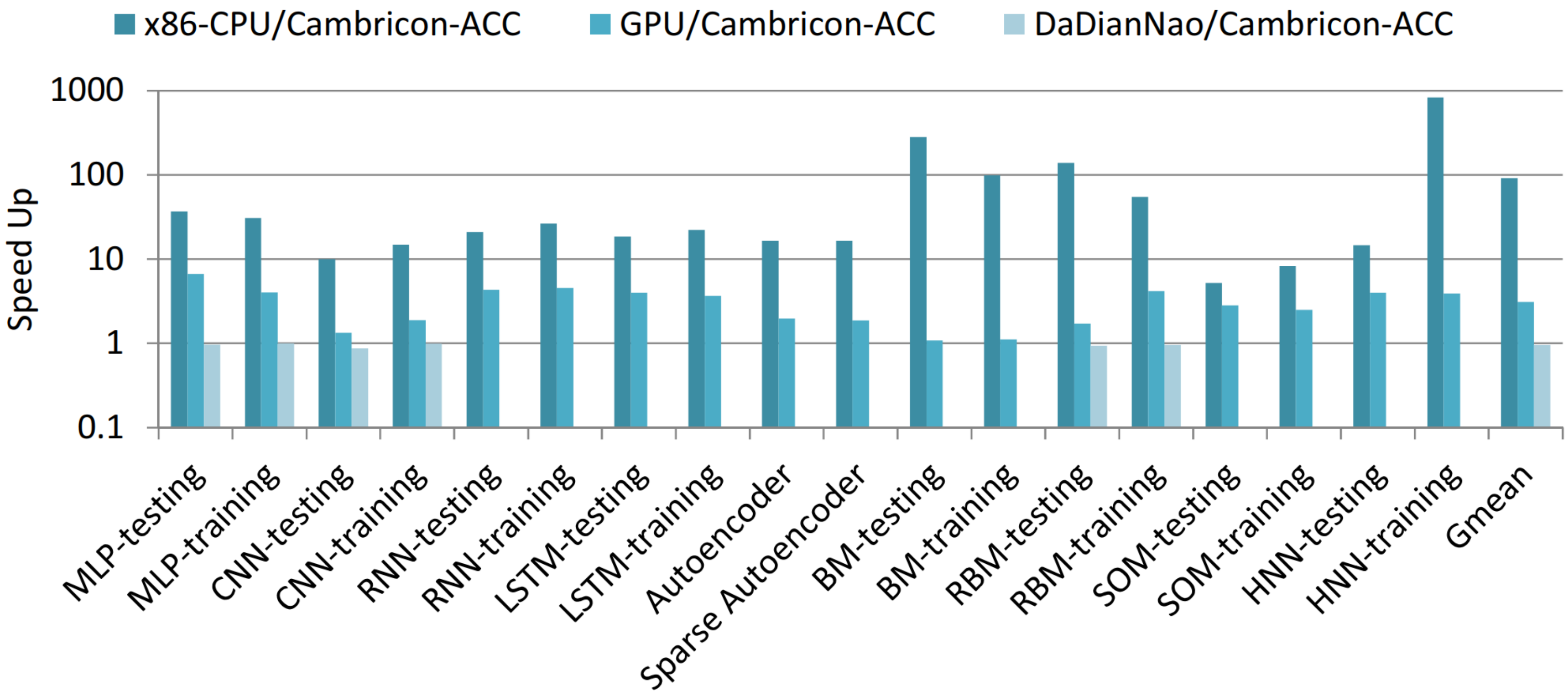
- The reduction of code length against GPU, x86-CPU, and MIPS-CPU



OVERVIEW OF CAMRICON

Performance

- The speedup of Cambricon-ACC against x86-CPU, GPU, and DaDianNao.



3

PART THREE

INSTRUCTION SET

Cambricon is a load-store architecture that integrates scalar, vector, matrix, logical, data transfer, and control instructions

INSTRUCTION SET

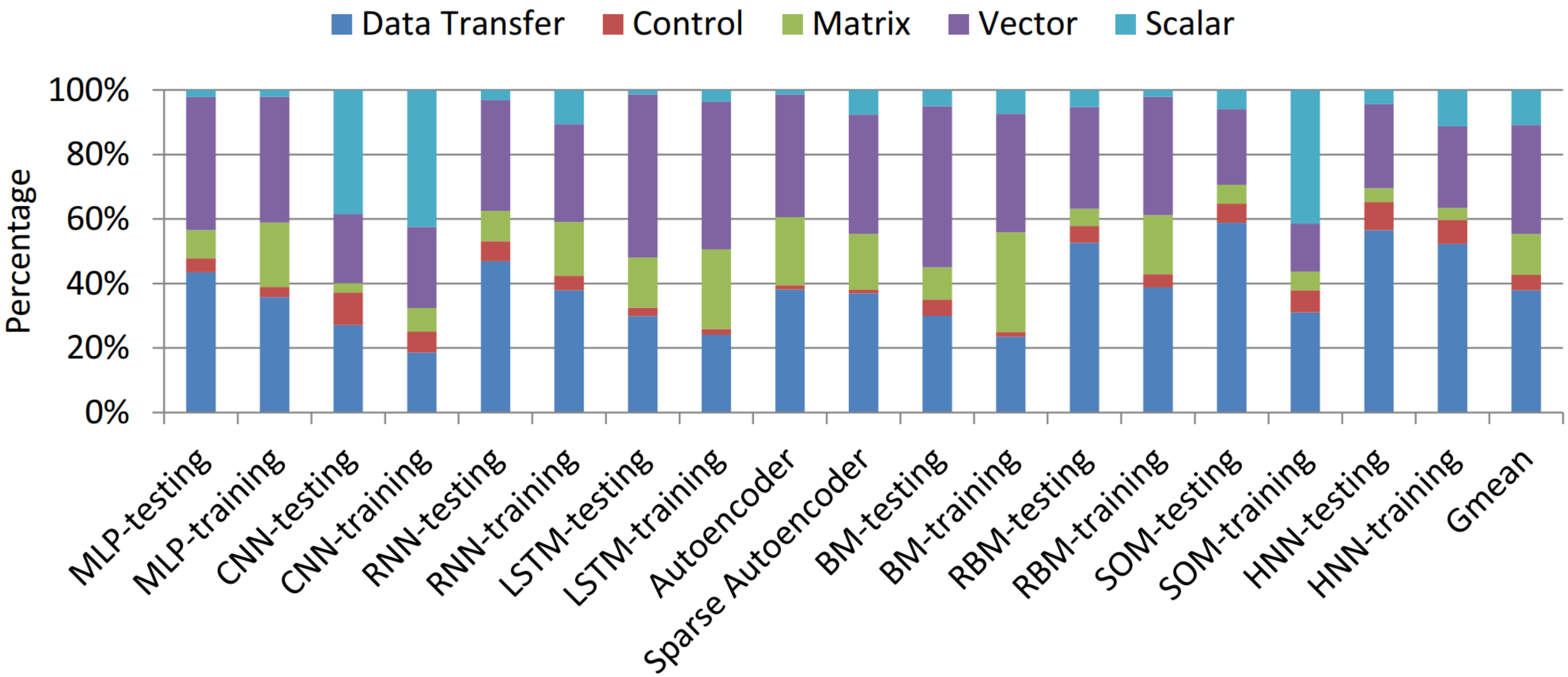
An overview to Cambricon instructions.

Instruction Type		Examples	Operands
Control		jump, conditional branch	register (scalar value), immediate
Data Transfer	Matrix	matrix load/store/move	register (matrix address/size, scalar value), immediate
	Vector	vector load/store/move	register (vector address/size, scalar value), immediate
	Scalar	scalar load/store/move	register (scalar value), immediate
Computational	Matrix	matrix multiply vector, vector multiply matrix, matrix multiply scalar, outer product, matrix add matrix, matrix subtract matrix	register (matrix/vector address/size, s-calar value)
	Vector	vector elementary arithmetics (add, subtract, multiply, divide), vector transcendental functions (exponential, logarithmic), dot product, random vector generator, maximum/minimum of a vector	register (vector address/size, scalar value)
	Scalar	scalar elementary arithmetics, scalar transcendental functions	register (scalar value), immediate
Logical	Vector	vector compare (greater than, equal), vector logical operations (and, or, inverter), vector greater than merge	register (vector address/size, scalar)
	Scalar	scalar compare, scalar logical operations	register (scalar), immediate

INSTRUCTION SET

An overview to Cambricon instructions.

- The percentages of instruction types among all benchmarks:

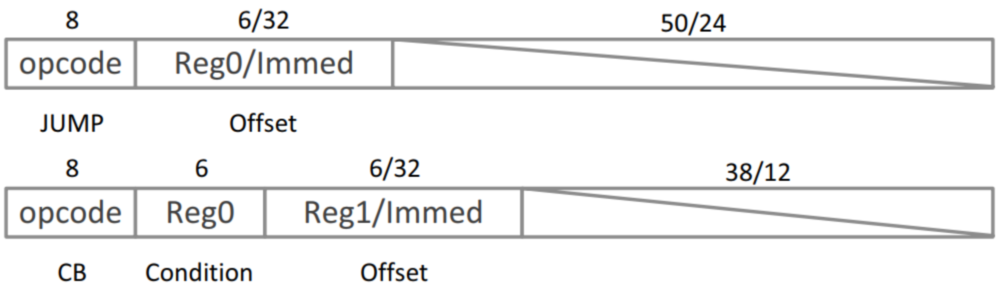


INSTRUCTION SET

Control Instructions

- There are two control instructions in Cambricon: *jump* and *conditional branch*.

- The jump instruction specifies the offset via either an immediate or a GPR value, which will be accumulated to PC.
- The conditional branch instruction specifies the predictor (stored in a GPR) in addition to the offset, and the branch target is determined by a comparison between the predictor and zero.



INSTRUCTION SET

Data Transfer Instructions

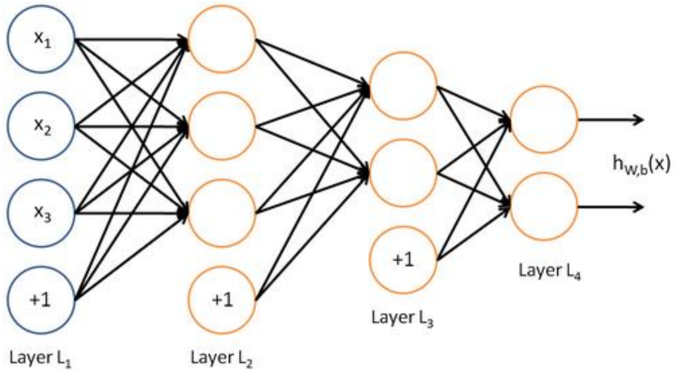
- Data transfer instructions in Cambricon can load/store variable-size data blocks from/to the main memory to/from the on-chip scratchpad memory, or move data between the on-chip scratchpad memory and scalar GPRs.

- Vector LOAD (VLOAD): load a vector with the size of v_size from the main memory to the vector scratchpad memory.
- Vector STORE (VSTORE), Matrix LOAD (MLOAD), Matrix STORE (MSTORE)

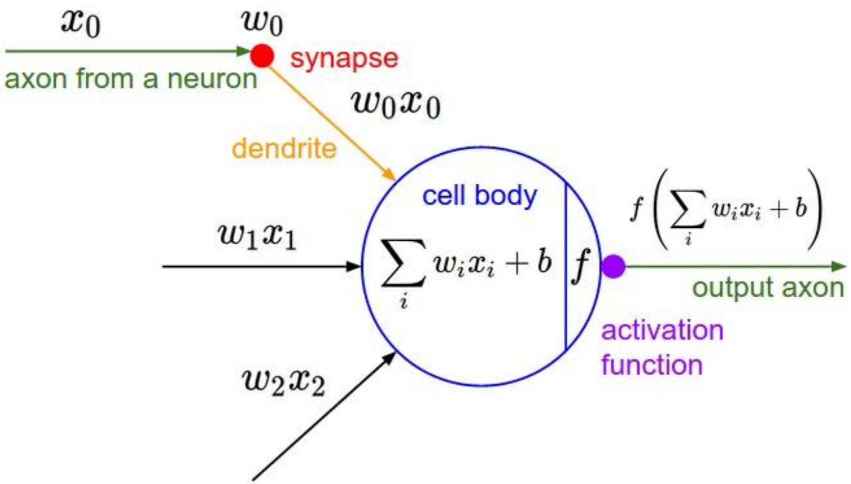
8	6	6	6	32	6
opcode	Reg0	Reg1	Reg2	Immed	
VLOAD	Dest_addr	V_size	Src_base	Src_offset	

INSTRUCTION SET

Matrix Instructions



$$\mathbf{y} = \mathbf{f}(W\mathbf{x} + \mathbf{b})$$



INSTRUCTION SET

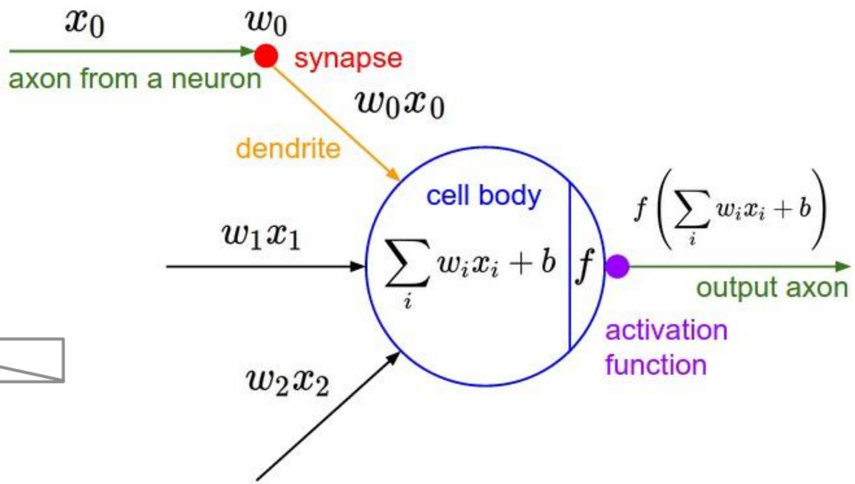
Matrix Instructions

- To compute $\boldsymbol{Wx}$, we need a matrix-mult-vector instruction.

  - Input Matrix size: Vout_size x Vin_size
  - Input Vector size: Vin_size
  - Output Vector size: Vout_size

8	6	6	6	6	6	26
opcode	Reg0	Reg1	Reg2	Reg3	Reg4	
MMV	Vout_addr	Vout_size	Min_addr	Vin_addr	Vin_size	

$$\mathbf{y} = \mathbf{f}(W\mathbf{x} + \mathbf{b})$$



INSTRUCTION SET

Matrix Instructions

- To compute the gradient vector in Back Propagation, we need a vector-mult-matrix instruction.

- Input Matrix size: Vin_size x Vout_size
- Input Vector size: Vin_size
- Output Vector size: Vout_size

8	6	6	6	6	6	26
opcode	Reg0	Reg1	Reg2	Reg3	Reg4	
VMM	Vout_addr	Vout_size	Min_addr	Vin_addr	Vin_size	

$$J(\theta) = \frac{1}{2}(\boldsymbol{X}\theta - \boldsymbol{Y})^T(\boldsymbol{X}\theta - \boldsymbol{Y})$$

$$\frac{\partial}{\partial\theta}J(\theta) = \boldsymbol{X}^T(\boldsymbol{X}\theta - \boldsymbol{Y})$$

INSTRUCTION SET

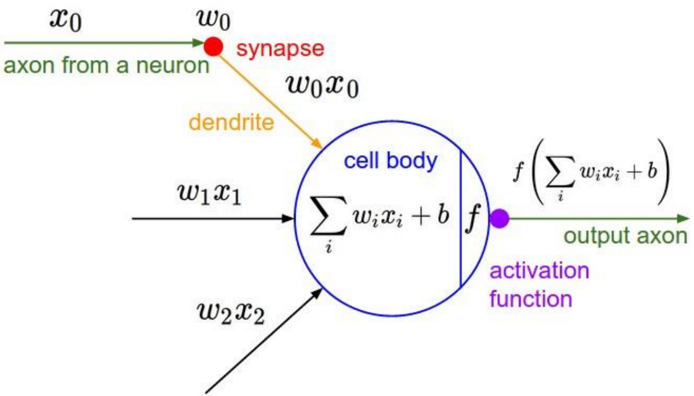
Matrix Instructions

- In training an NN, the weight matrix W often needs to be incrementally updated with
  $$W = W + \eta \Delta W,$$
- where η is the learning rate, and ΔW is estimated as the outer product of two vectors.

- Cambricon provides an **Outer-Product (OP)** instruction, a **Matrix-Mult-Scalar (MMS)** instruction, and a **Matrix-Add-Matrix (MAM)** instruction to collaboratively perform the weight updating.

- In addition, Cambricon also provides a **Matrix-Subtract-Matrix (MSM)** instruction to support the weight updating in Restricted Boltzmann Machine (RBM)

$$W = W + \eta \Delta W$$



INSTRUCTION SET

Matrix Instructions

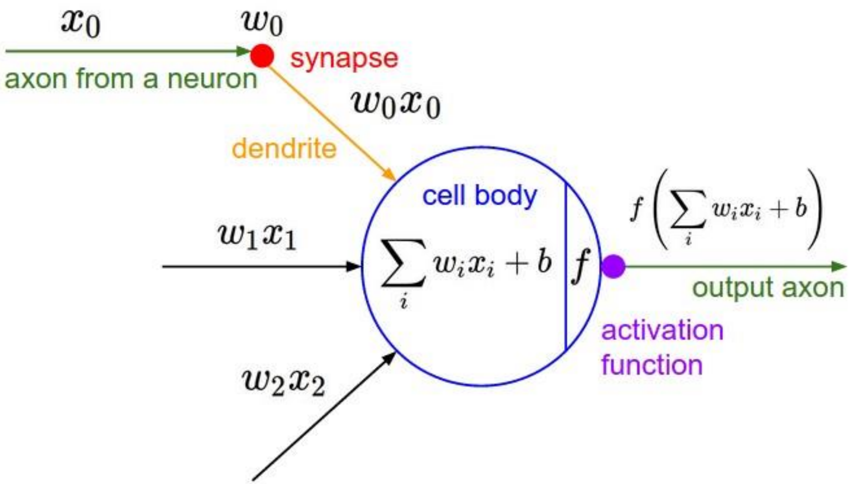
- Matrix Mult Vector (MMV)
- Vector Mult Matrix (VMM)
- Outer Product (OP)
- Matrix Mult Scalar (MMS)
- Matrix Add Matrix (MAM)
- Matrix Subtract Matrix (MSM)

INSTRUCTION SET

Vector Instructions

- To compute $W\boldsymbol{x} + \boldsymbol{b}$, we need a Vector-Add-Vector (VAV) instruction.

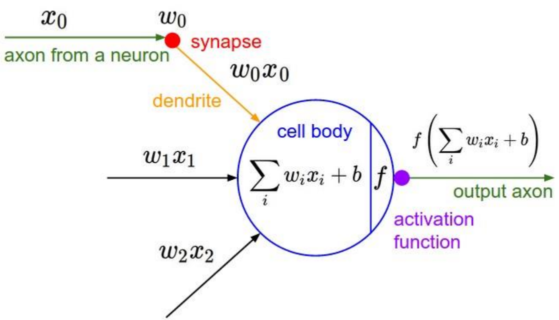
$$\mathbf{y} = \mathbf{f}(W\mathbf{x} + \mathbf{b})$$



INSTRUCTION SET

Vector Instructions

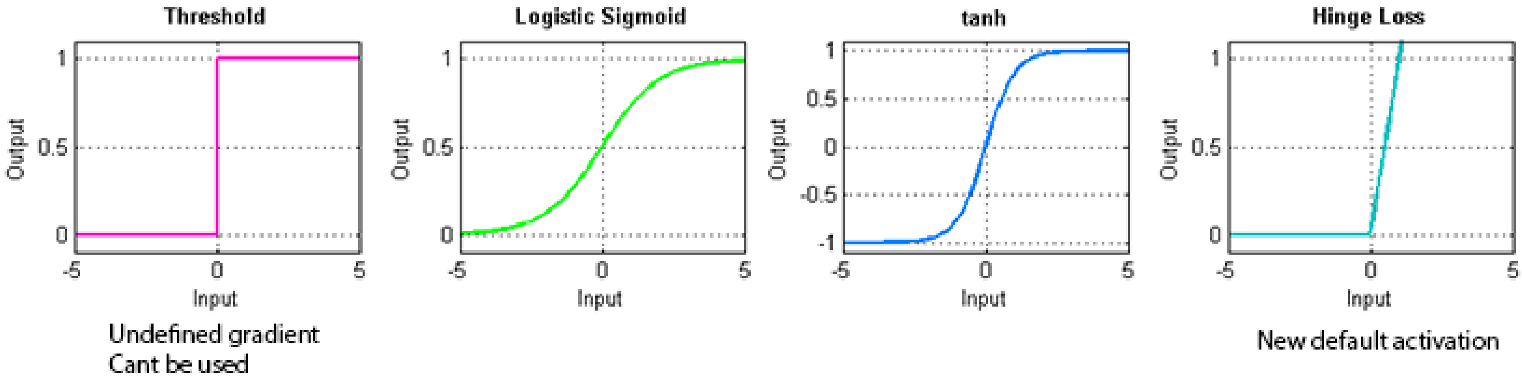
$$\mathbf{y} = \mathbf{f}(W\mathbf{x} + \mathbf{b})$$



- To compute $\mathbf{f}(W\boldsymbol{x} + \boldsymbol{b})$:

($\mathbf{f}$ is the element-wise version of the activation function f.)

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



INSTRUCTION SET

Vector Instructions

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

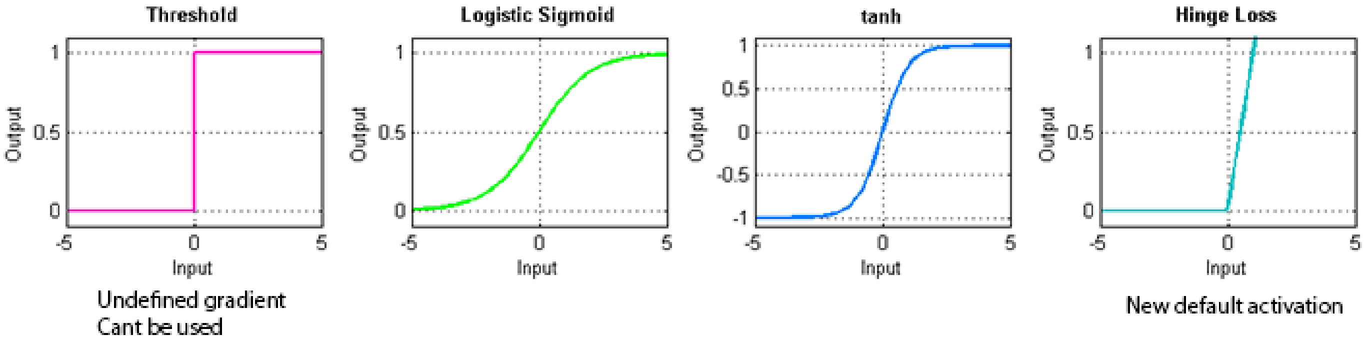
- Computing the exponential e^{a_i} for each element (a_i, i = 1,…,n) in the input vector $\boldsymbol{a}$.
  -> **Vector-Exponential (VEXP)** instruction: for elementwise exponential of a vector

- Adding the constant 1 to each element of the vector (e^{a_i},…, e^{a_n}).
  -> **Vector-Add-Scalar (VAS)** instruction

- Dividing e^{a_i} by $1 + e^{a_i}$ for each vector index i = 1,…,n.
  -> **Vector-Div-Vector (VDV)** instruction: for element-wise division between vectors

INSTRUCTION SET

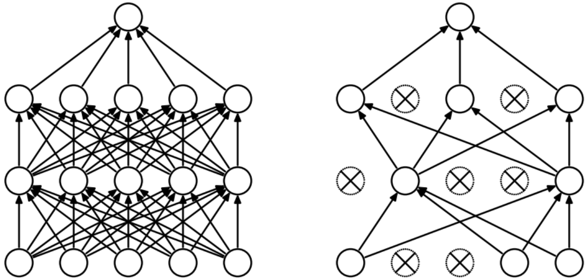
Vector Instructions



- **Vector-Mult-Vector (VMV)**, **Vector-Sub-Vector (VSV)**, **Vector-Logarithm (VLOG)**
- Use CORDIC technique to calculate transcendental functions (e.g. logarithmic, trigonometric and anti-trigonometric functions) using addition, subtract, shift and table-lookup operations.

INSTRUCTION SET

Vector Instructions



(a) Standard Neural Net (b) After applying dropout.

Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

- **Random-Vector (RV)**: generates a vector of random numbers obeying the uniform distribution at the interval [0,1].

- Given uniform random vectors, we can further generate random vectors obeying other distributions (e.g., Gaussian distribution) using the Ziggurat algorithm.

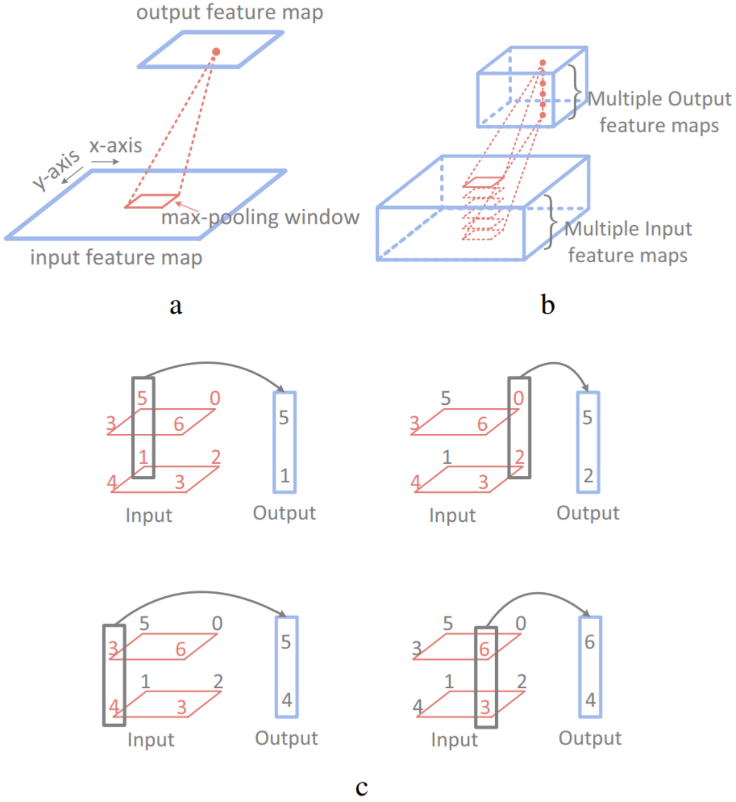
INSTRUCTION SET

Logical Instructions

- **Vector-Greater-Than-Merge (VGTM)**
  - Vout[i] = (Vin0[i] > Vin1[i])?Vin0[i] : Vin1[i]

- **Vector-Greater-than (VGT)**
- **Vector-Equal instruction (VE)**
- **Vector AND/OR/NOT instructions(VAND/VOR/VNOT)**

8	6	6	6	6	32
opcode	Reg0	Reg1	Reg2	Reg3	
VGTM	Vout_addr	Vout_size	Vin0_addr	Vin1_addr	



4

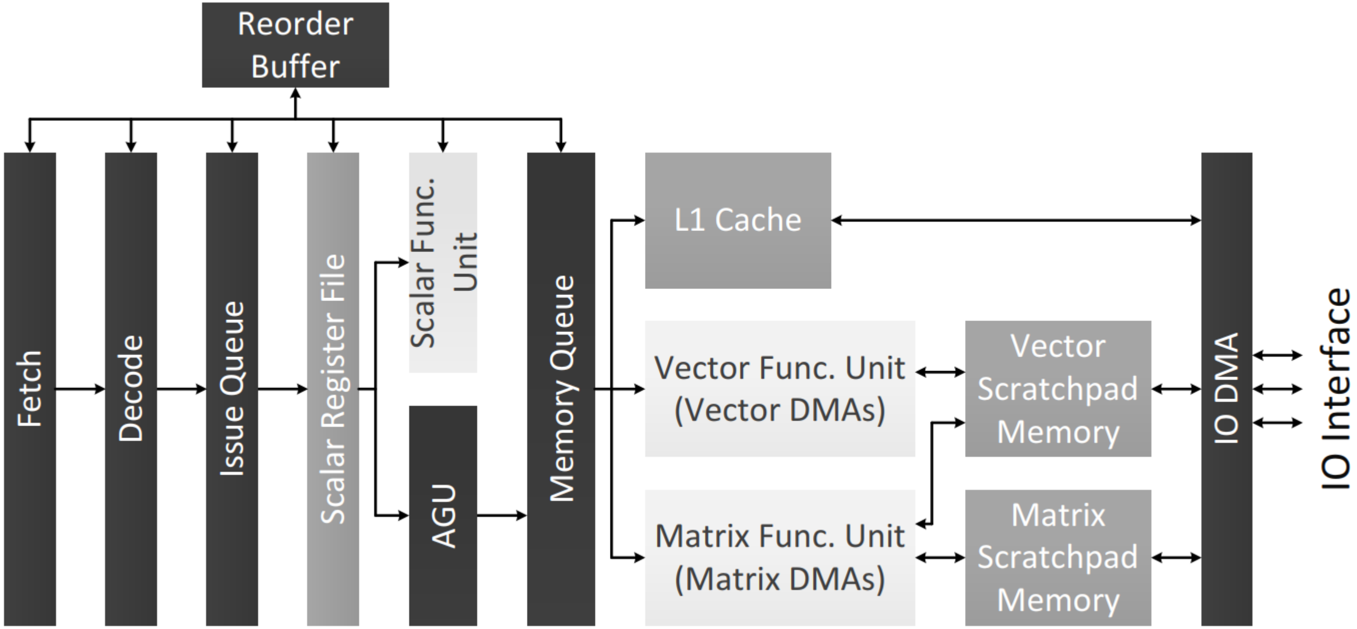
PART THREE

PROTOTYPE ACCELERATOR

Present a prototype accelerator of Cambricon

PROTOTYPE ACCELERATOR

Architecture



7 pipeline stages: *fetching, decoding, issuing, register reading, execution, writing back, committing*

THANKS FOR YOUR WATCHING

weiq618@mail.ustc.edu.cn

References

* Liu, Shaoli, et al. "Cambricon: An instruction set architecture for neural networks." ACM SIGARCH Computer Architecture News. Vol. 44. No. 3. IEEE Press, 2016.
* Hennessy, J.L., Patterson, D.A. Computer Architechture: A Quantitative Approach, Fifth Edition.
* Rosenblatt, Frank. x. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books, Washington DC, 1961
* Rumelhart, David E., Geoffrey E. Hinton, and R. J. Williams. "Learning Internal Representations by Error Propagation". David E. Rumelhart, James L. McClelland, and the PDP research group. (editors), Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundation. MIT Press, 1986.
* Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function Mathematics of Control, Signals, and Systems, 2(4), 303–314.
* Deep Learning; Ian Goodfellow, Yoshua Bengio, Aaaron Courville; MIT Press; 2016, p 196.
* V. Kantabutra. On hardware for computing exponential and trigonometric functions. Computers, IEEE Transactions on, 1996.
* G Marsaglia and W W. Tsang. The ziggurat method for generating random variables. Journal of statistical software, 2000.
* Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: A simple way to prevent neural networks from overfitting[J]. The Journal of Machine Learning Research, 2014, 15(1): 1929-1958.