Neural ALU

YU PAN

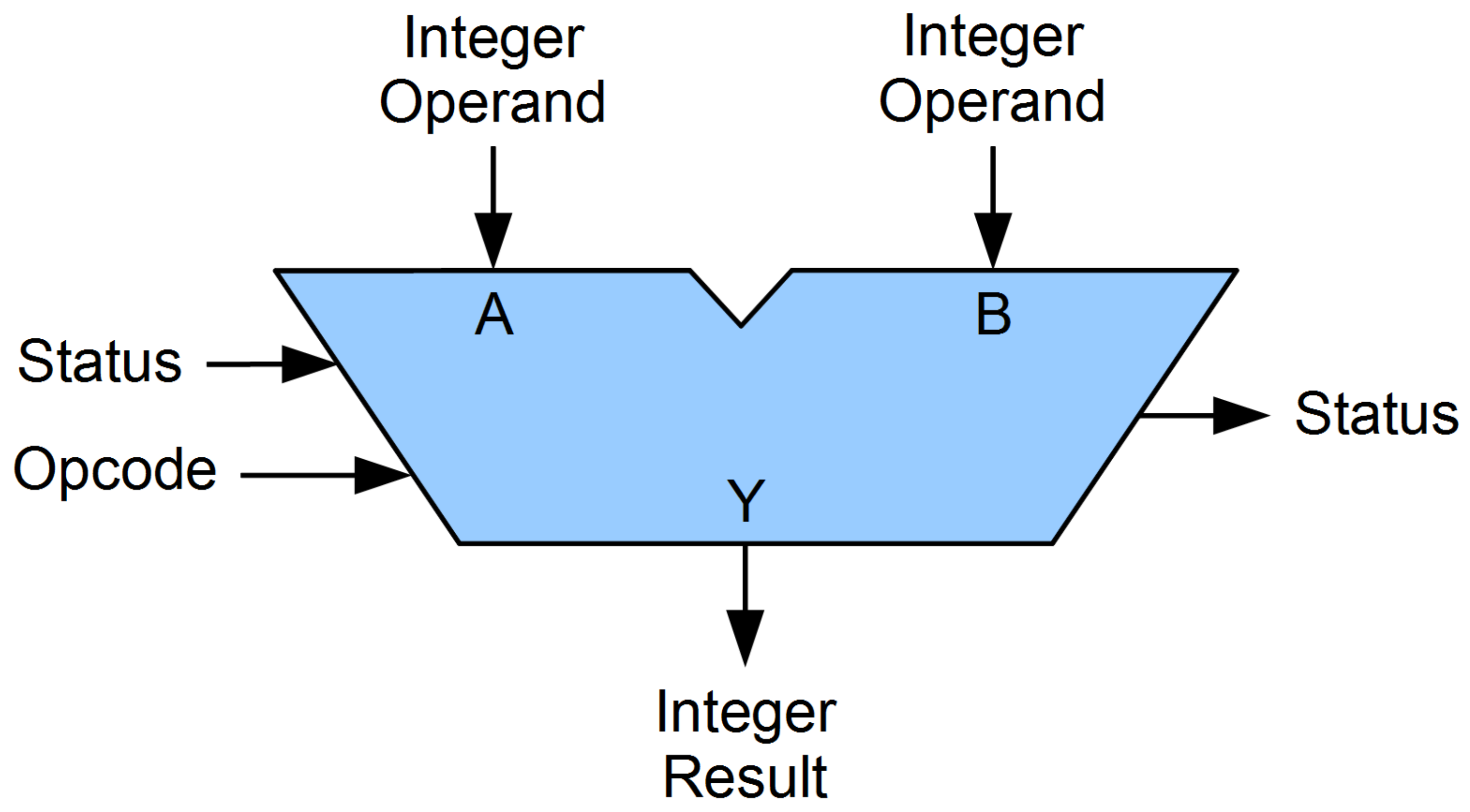
ALU(Arithmetic Logic Units)

Hardware:

An arithmetic logic unit (ALU) is a combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. An ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.

Computer world is based on ALUs!

ALU(Arithmetic Logic Units)



What’s the shortage of the NN?

Do a game:

- 1, 4, 7, _?
- 1, 4, 9, _?
- 2, _, 6, 8?

What’s the shortage of the NN?

Do a game:

- 1, 4, 7, _?
- 1, 4, 9, _?
- 2, _, 6, 8?

Extrapolation

Interpolation

But for a neural network, extrapolation could be difficult!

What’s the shortage of the NN?

Show with an experiment.

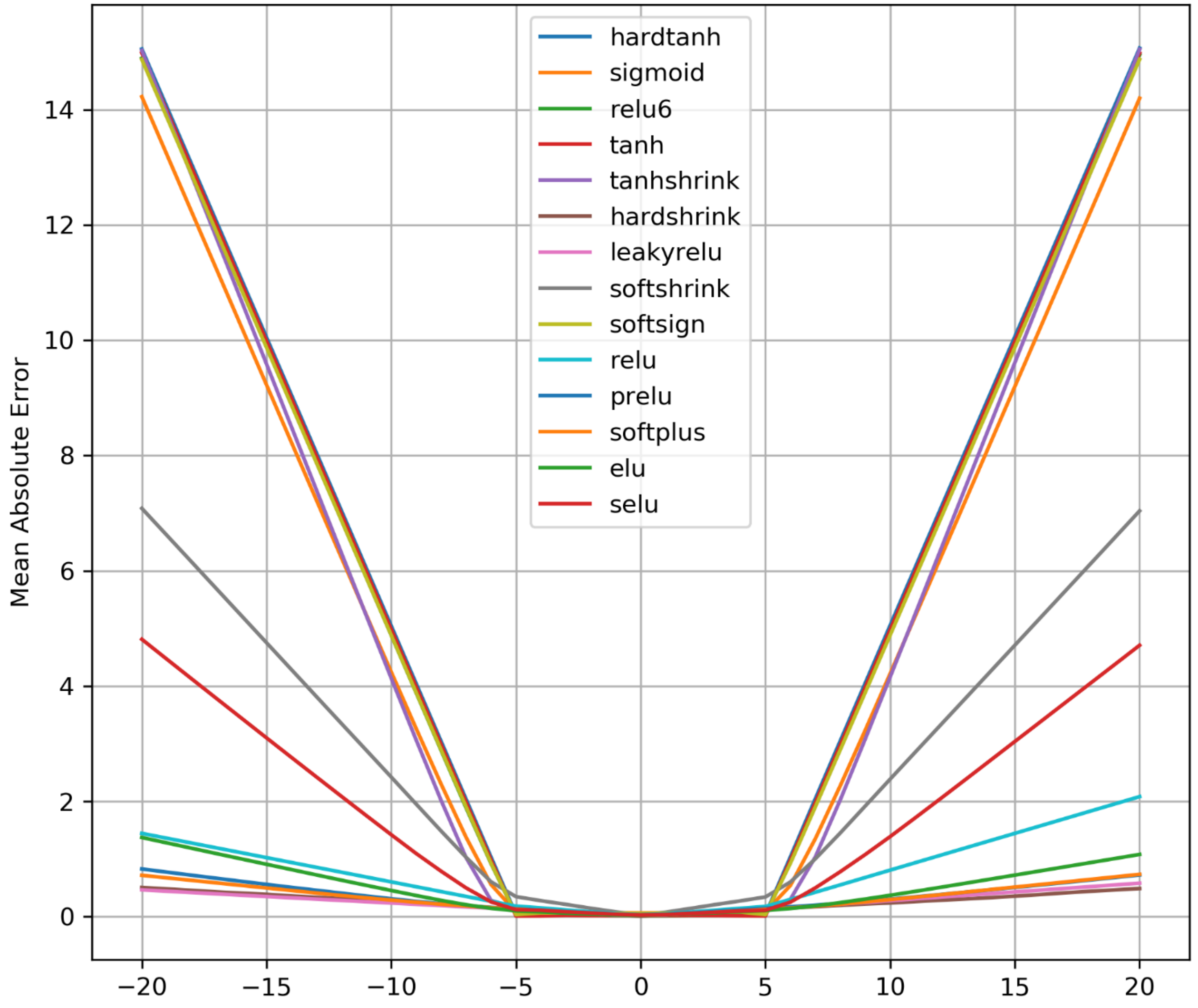
- Construct MLPs of 4 layers with different activation function to learn identity function f(x)=x.
- The range of training data is [-5, 5].
- For test data, it’s [-20, 20].

Wha

Show with

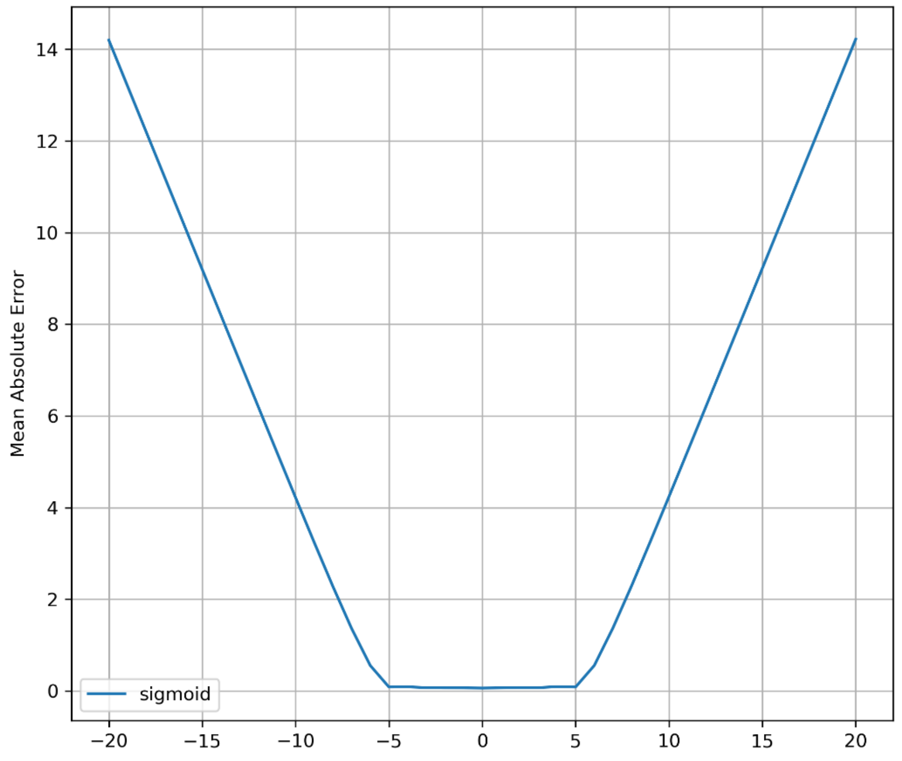
- Construct   unction f(x)=x.
- The range
- For test d

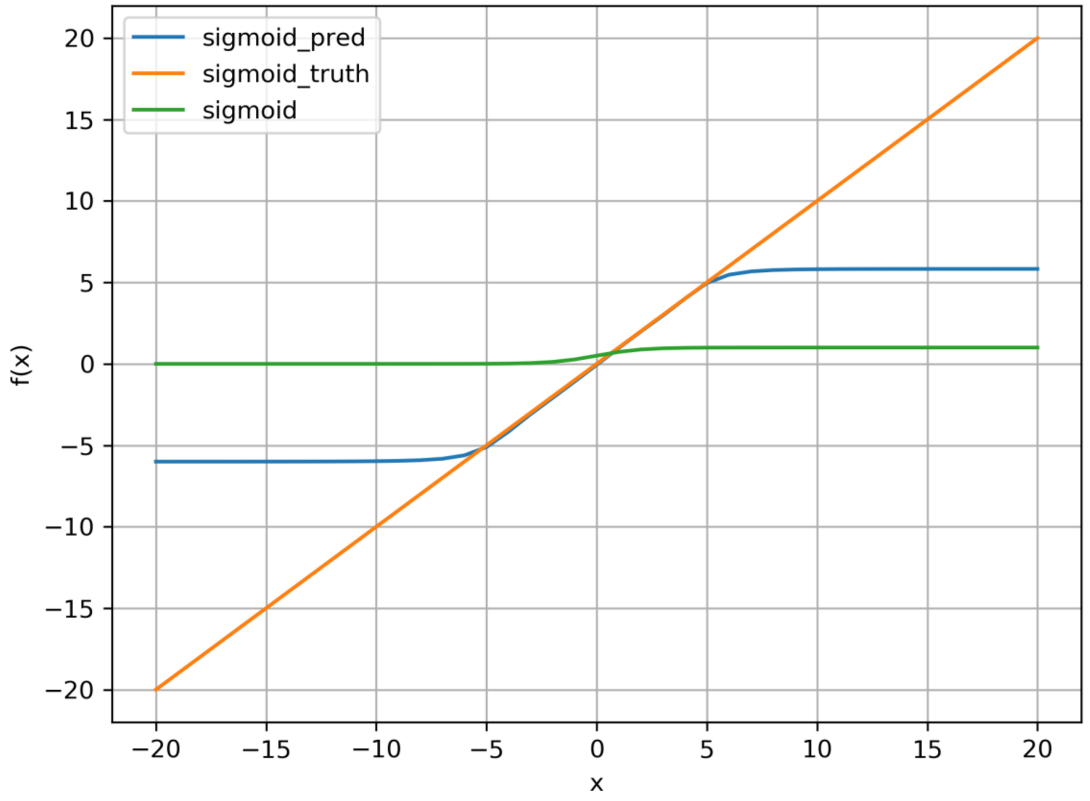
Result:



Using sigmoid for example

Result:





How to solve the numerical extrapolation?

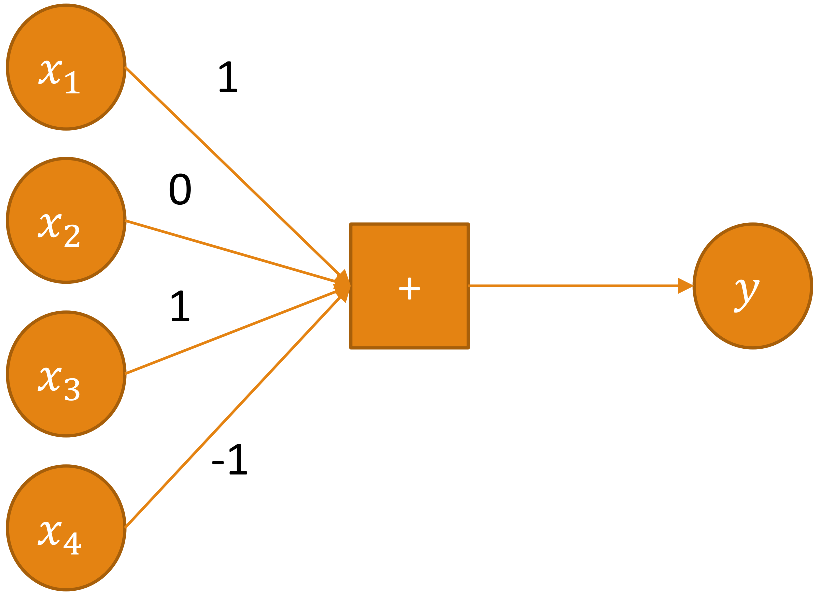
To solve this problem, in this paper, the author proposed two models:

- The NAC(The Neural Accumulator)
- The NALU(The Neural Arithmetic Logic Unit)

The NAC(The Neural Accumulator)

- Addition is always basis of the algorithm.
- If $Y = Wx$, a neural addition without scaling should mean that the matrix W consists just of -1's, 0's and 1's.

Why:



$$y = x_1 + x_3 - x_4$$

The NAC(The Neural Accumulator)

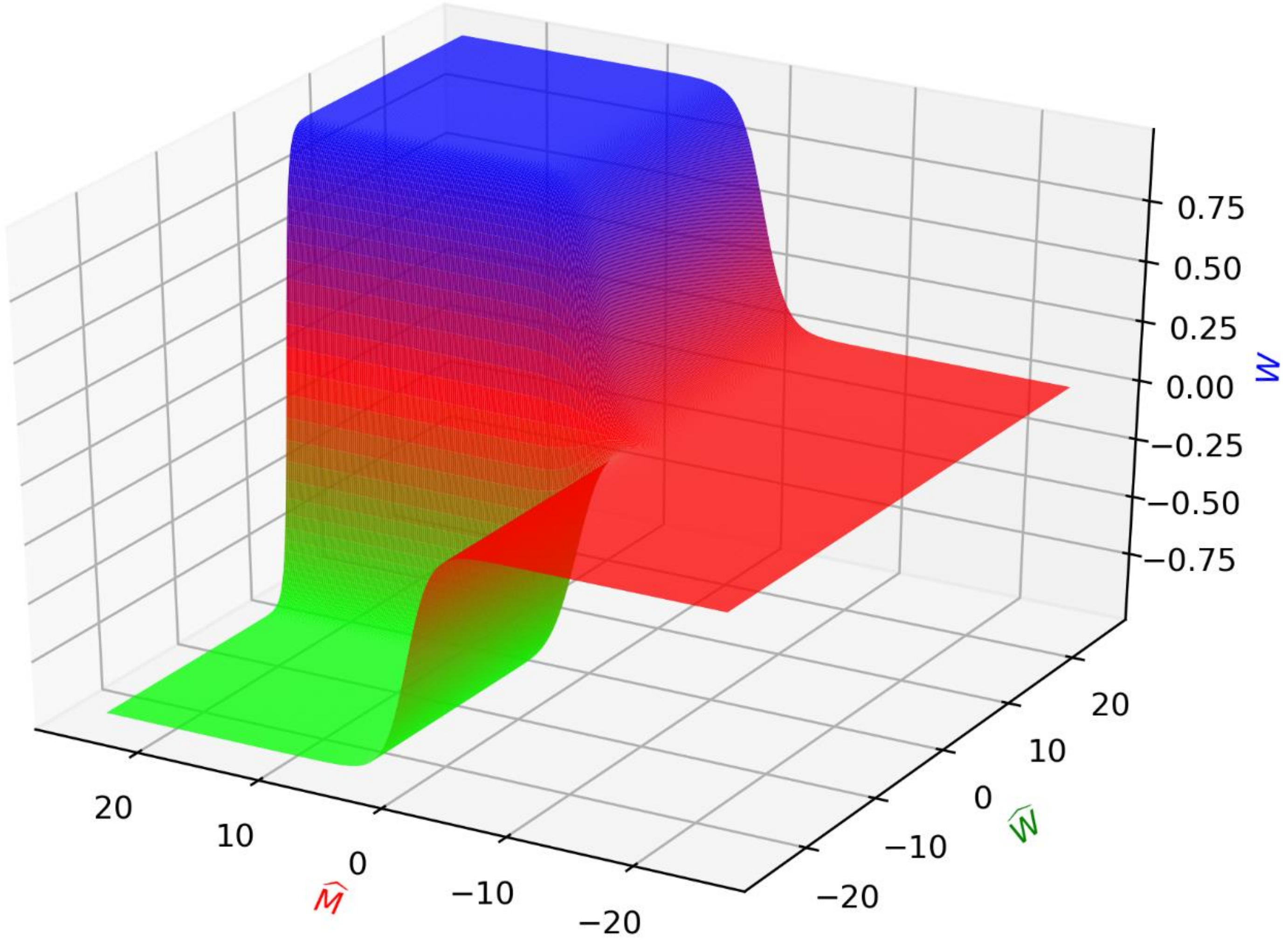
- Enforcing every element of W being one of {-1, 0, 1} would make learning hard.
- So, the paper proposed:

$$W = \tanh(\widehat{W}) \odot \sigma(\widehat{M})$$

- Show:

The N

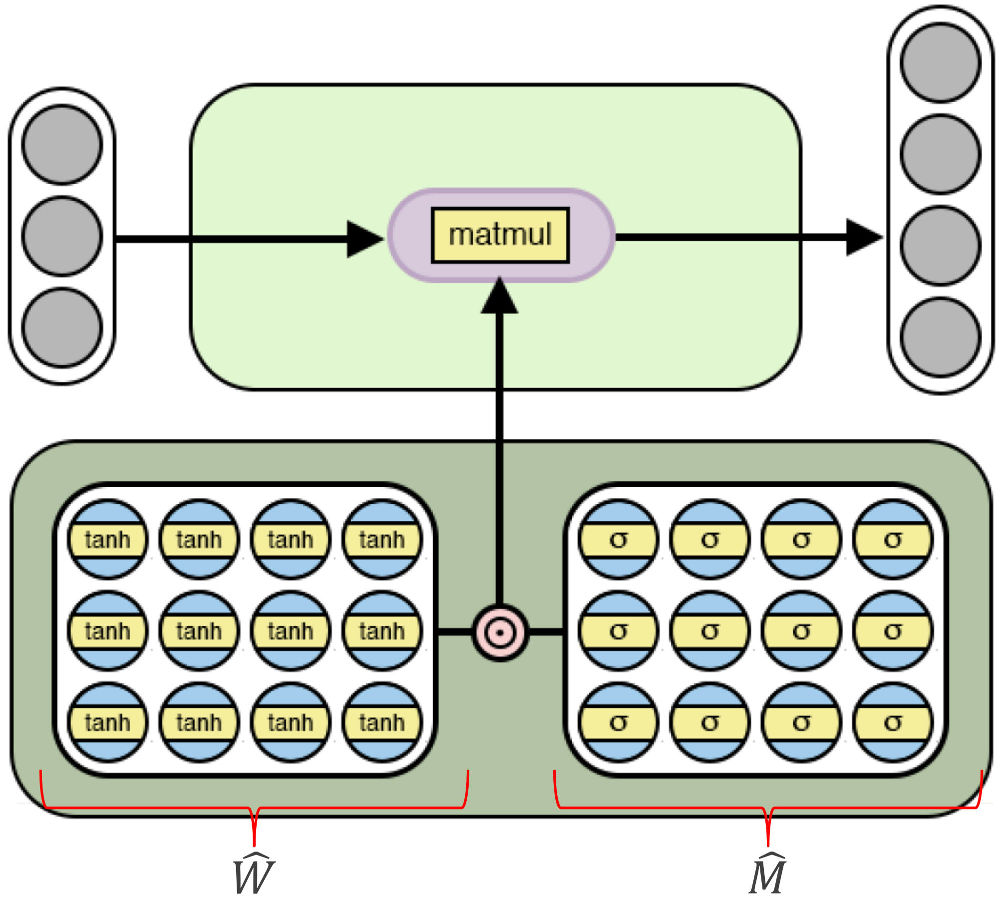
- Enforcing
- So, the pa
- Show:



The NAC(The Neural Accumulator)

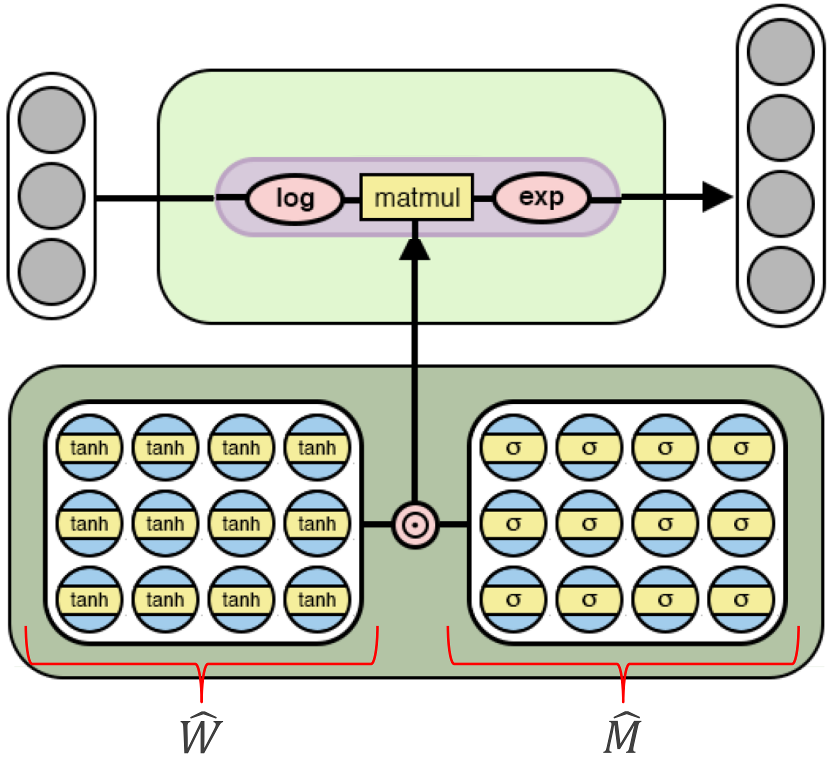
- The structure of the NAC

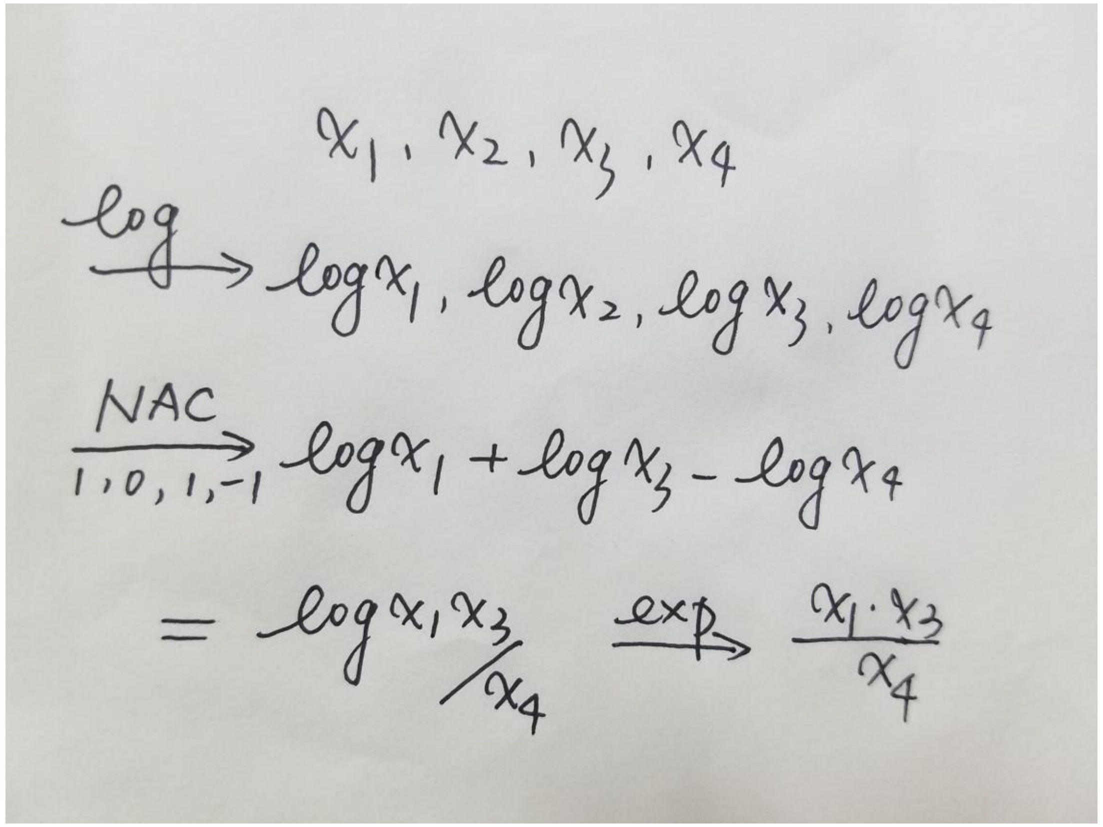
$$W = \tanh(\hat{W}) \odot \sigma(\hat{M})$$



The NAC(The Neural Accumulator)

- To do multiplication, the paper proposed a block:





$$x_1, x_2, x_3, x_4$$

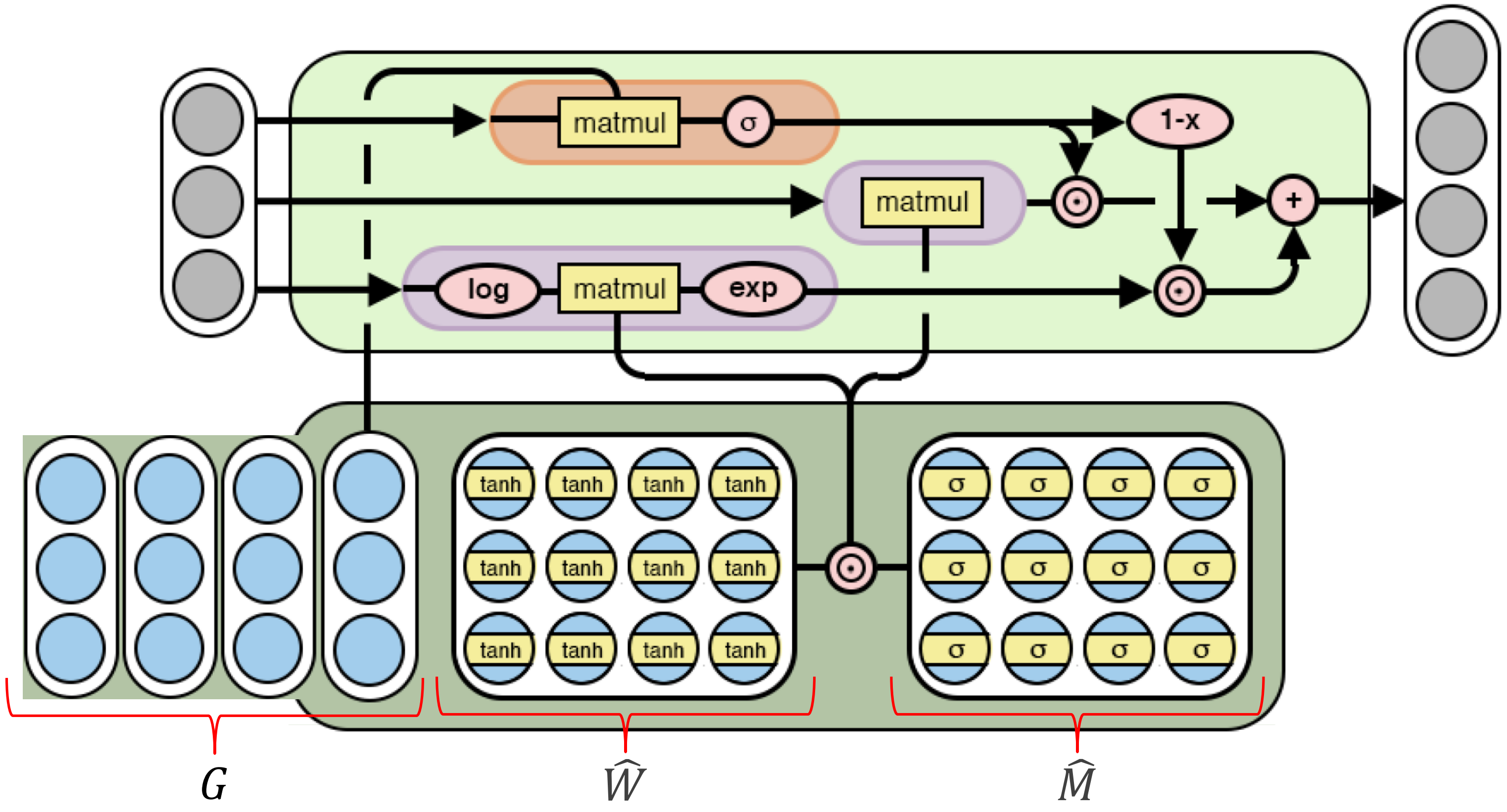
$$\xrightarrow{log} \log x_1, \log x_2, \log x_3, \log x_4$$

$$\xrightarrow[1,0,1,-1]{NAC} \log x_1 + \log x_3 - \log x_4$$

$$= \log x_1 x_3 / x_4 \xrightarrow{exp} \frac{x_1 \cdot x_3}{x_4}$$

The NALU(The Neural Arithmetic Logic Unit)

- The NALU: combine the two block and set a gate to control data flow.



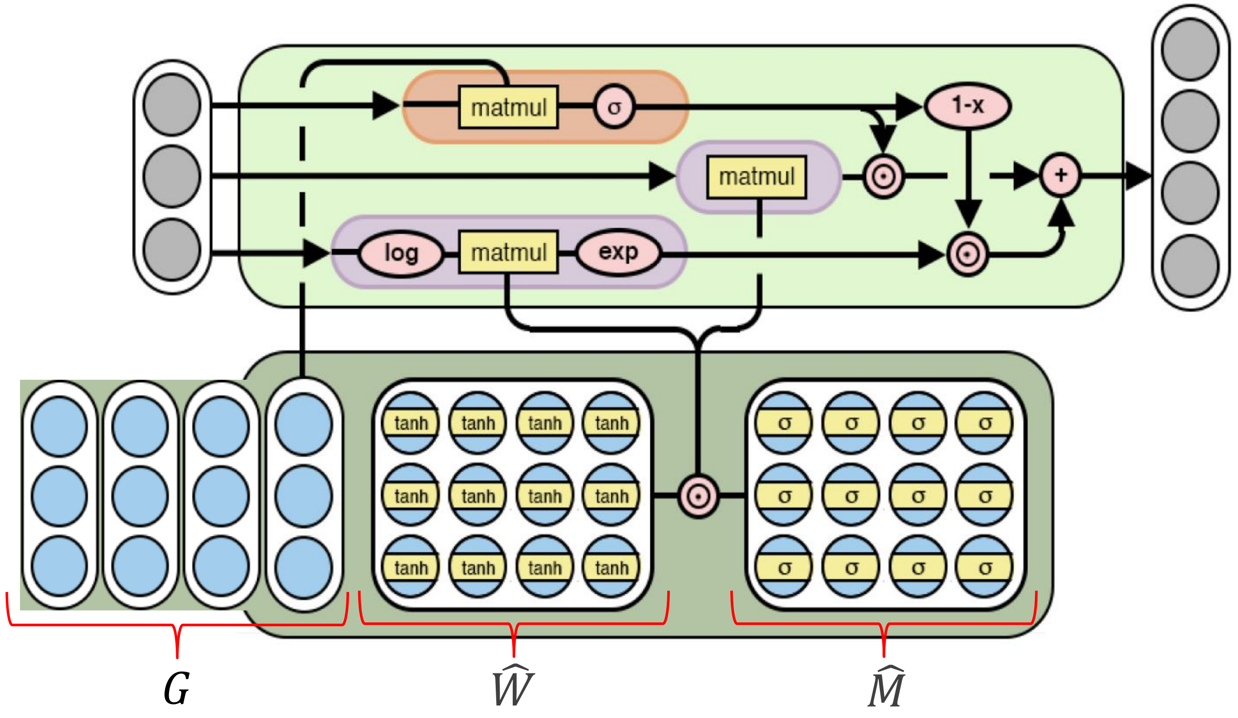
The NALU(The Neural Arithmetic Logic Unit)

- Summary:
- NAC:

$$a = Wx$$
$$W = \tanh(\hat{W}) \odot \sigma(\hat{M})$$

- NALU:

$$Y = g \odot a + (1 - g) \odot m$$
$$m = \exp W(\log(|x| + \epsilon))$$
$$g = \sigma(Gx)$$



The NALU(The Neural Arithmetic Logic Unit)

- Easy to learn addition, subtraction, multiplication, division.
- Easy to solve the learning of the identity function f(x)=x.
- Vectors calculated with mathematical methods achieves numerical extrapolation.

Experiments

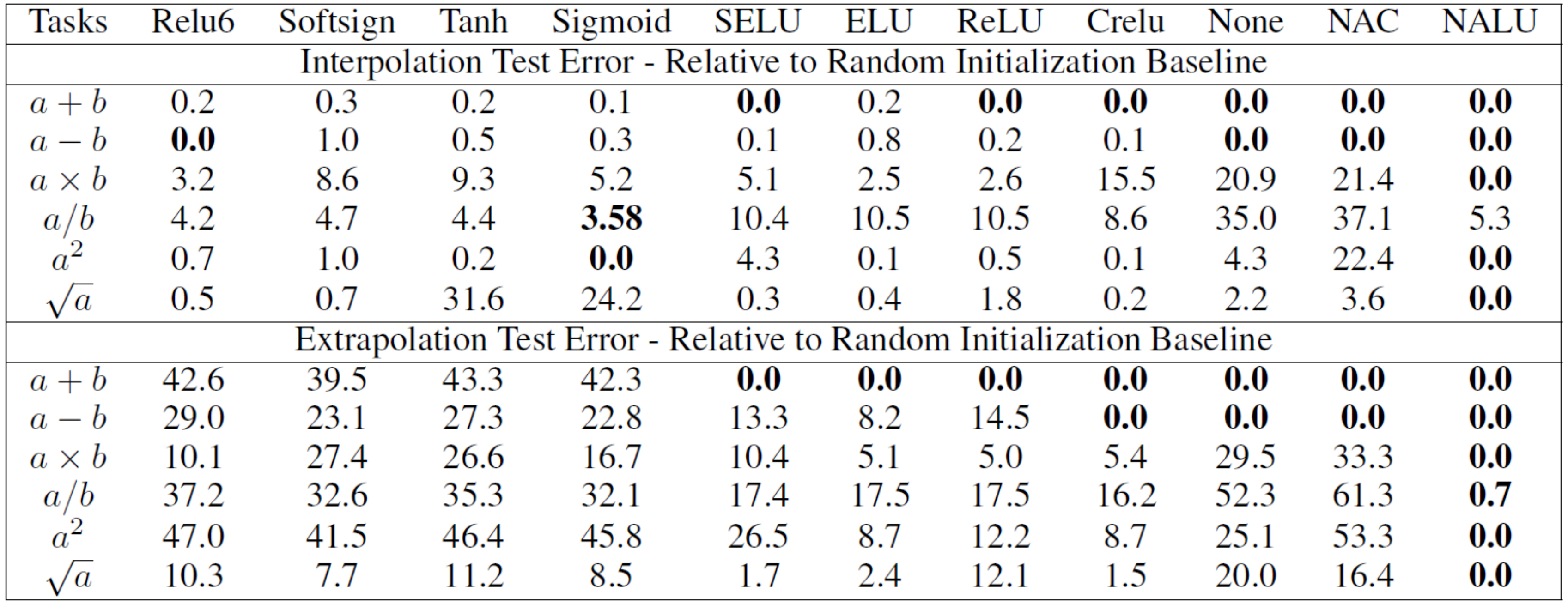
- Simple Function Learning Tasks
- MNIST Counting and Arithmetic Tasks
- Language to Number Translation Tasks
- Learning to Track Time in a Grid-World Environment
- MNIST Parity Prediction Task & Ablation Study

Simple Function Learning Tasks

- Aim: To show the ability of NACs and NALUs to learn arithmetic functions.

- Implement:
  - Static Task
  - Recurrent Task

Static Task

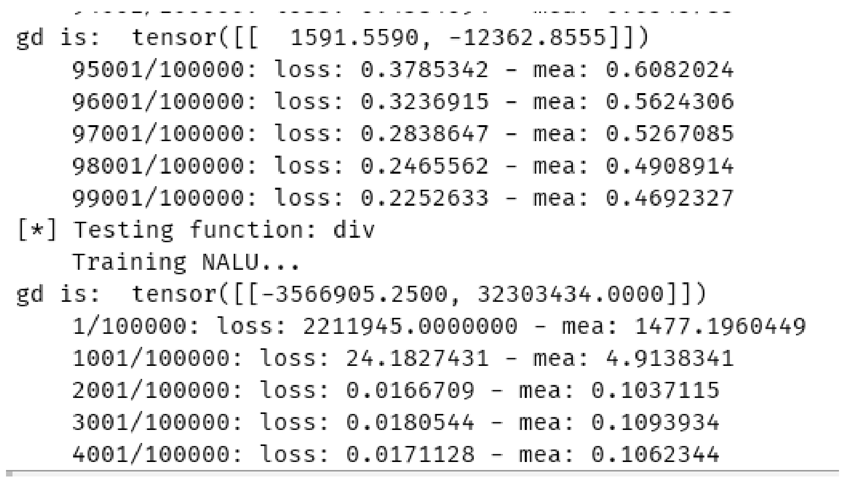


Tasks	Relu6	Softsign	Tanh	Sigmoid	SELU	ELU	ReLU	Crelu	None	NAC	NALU
Interpolation Test Error - Relative to Random Initialization Baseline											
$a+b$	0.2	0.3	0.2	0.1	**0.0**	0.2	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
$a-b$	**0.0**	1.0	0.5	0.3	0.1	0.8	0.2	0.1	**0.0**	**0.0**	**0.0**
$a \times b$	3.2	8.6	9.3	5.2	5.1	2.5	2.6	15.5	20.9	21.4	**0.0**
a/b	4.2	4.7	4.4	**3.58**	10.4	10.5	10.5	8.6	35.0	37.1	5.3
a^2	0.7	1.0	0.2	**0.0**	4.3	0.1	0.5	0.1	4.3	22.4	**0.0**
$\sqrt{a}$	0.5	0.7	31.6	24.2	0.3	0.4	1.8	0.2	2.2	3.6	**0.0**
Extrapolation Test Error - Relative to Random Initialization Baseline											
$a+b$	42.6	39.5	43.3	42.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
$a-b$	29.0	23.1	27.3	22.8	13.3	8.2	14.5	**0.0**	**0.0**	**0.0**	**0.0**
$a \times b$	10.1	27.4	26.6	16.7	10.4	5.1	5.0	5.4	29.5	33.3	**0.0**
a/b	37.2	32.6	35.3	32.1	17.4	17.5	17.5	16.2	52.3	61.3	**0.7**
a^2	47.0	41.5	46.4	45.8	26.5	8.7	12.2	8.7	25.1	53.3	**0.0**
$\sqrt{a}$	10.3	7.7	11.2	8.5	1.7	2.4	12.1	1.5	20.0	16.4	**0.0**

Table 7: Static (non-recurrent) arithmetic error rates. Lower is better. Best models in bold. Scores relative to one randomly initialized model for each task. 100.0 is equivalent to random. 0.0 is perfect accuracy. Raw scores are in the Appendix.

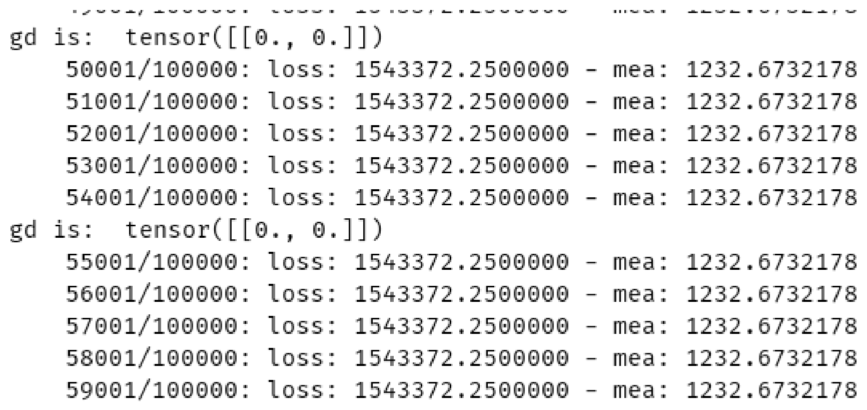
Static Task

- There is a training bug, when learning the multiplication function.



```
gd is:  tensor([[  1591.5590, -12362.8555]])
    95001/100000: loss: 0.3785342 - mea: 0.6082024
    96001/100000: loss: 0.3236915 - mea: 0.5624306
    97001/100000: loss: 0.2838647 - mea: 0.5267085
    98001/100000: loss: 0.2465562 - mea: 0.4908914
    99001/100000: loss: 0.2252633 - mea: 0.4692327
[*] Testing function: div
    Training NALU...
gd is:  tensor([[-3566905.2500, 32303434.0000]])
    1/100000: loss: 2211945.0000000 - mea: 1477.1960449
    1001/100000: loss: 24.1827431 - mea: 4.9138341
    2001/100000: loss: 0.0166709 - mea: 0.1037115
    3001/100000: loss: 0.0180544 - mea: 0.1093934
    4001/100000: loss: 0.0171128 - mea: 0.1062344
```

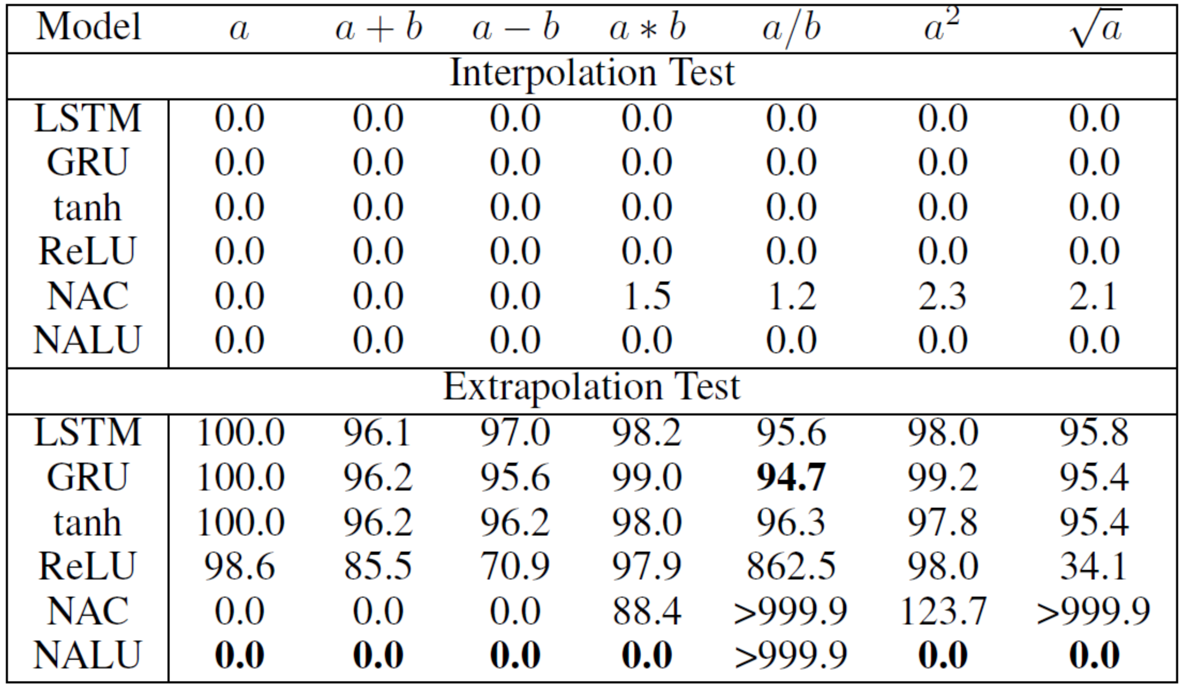
Normal Case



```
gd is:  tensor([[0., 0.]])
    50001/100000: loss: 1543372.2500000 - mea: 1232.6732178
    51001/100000: loss: 1543372.2500000 - mea: 1232.6732178
    52001/100000: loss: 1543372.2500000 - mea: 1232.6732178
    53001/100000: loss: 1543372.2500000 - mea: 1232.6732178
    54001/100000: loss: 1543372.2500000 - mea: 1232.6732178
gd is:  tensor([[0., 0.]])
    55001/100000: loss: 1543372.2500000 - mea: 1232.6732178
    56001/100000: loss: 1543372.2500000 - mea: 1232.6732178
    57001/100000: loss: 1543372.2500000 - mea: 1232.6732178
    58001/100000: loss: 1543372.2500000 - mea: 1232.6732178
    59001/100000: loss: 1543372.2500000 - mea: 1232.6732178
```

Strange Case

Recurrent Task

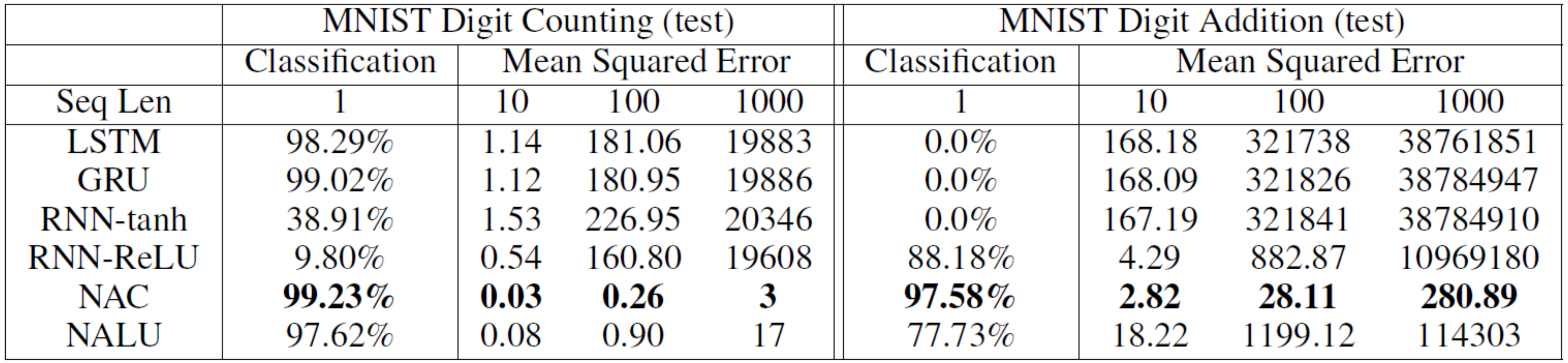


Model	a	$a+b$	$a-b$	$a*b$	a/b	a^2	$\sqrt{a}$
Interpolation Test							
LSTM	0.0	0.0	0.0	0.0	0.0	0.0	0.0
GRU	0.0	0.0	0.0	0.0	0.0	0.0	0.0
tanh	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ReLU	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NAC	0.0	0.0	0.0	1.5	1.2	2.3	2.1
NALU	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Extrapolation Test							
LSTM	100.0	96.1	97.0	98.2	95.6	98.0	95.8
GRU	100.0	96.2	95.6	99.0	**94.7**	99.2	95.4
tanh	100.0	96.2	96.2	98.0	96.3	97.8	95.4
ReLU	98.6	85.5	70.9	97.9	862.5	98.0	34.1
NAC	0.0	0.0	0.0	88.4	>999.9	123.7	>999.9
NALU	**0.0**	**0.0**	**0.0**	**0.0**	>999.9	**0.0**	**0.0**

Table 9: Recurrent use of NALU and NAC compared to modern recurrent architectures, evaluated using Mean Squared Error relative to a randomly initialized LSTM. 100.0 is equivalent to random. >100.0 is worse than random. 0 is perfect accuracy. Raw scores in appendix.

MNIST Counting and Arithmetic Tasks

- Aim: To discover whether backpropagation can learn the representation of non-numeric inputs to NACs/NALUs.

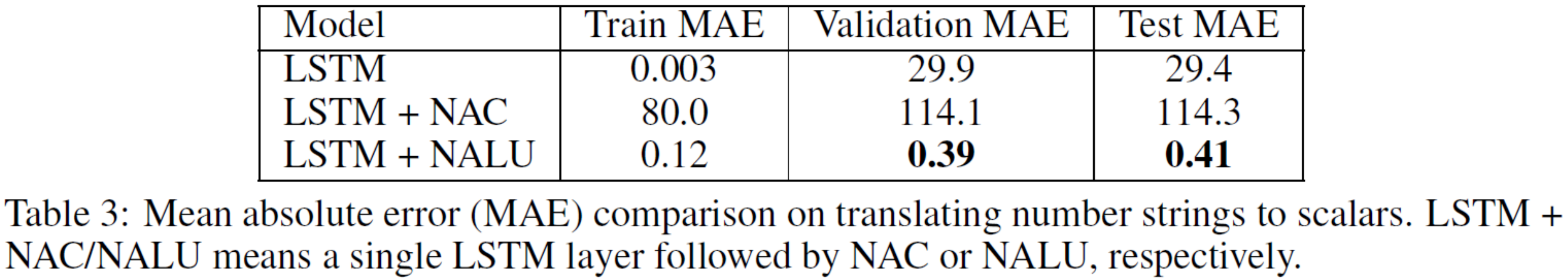


	MNIST Digit Counting (test)				MNIST Digit Addition (test)			
	Classification	Mean Squared Error			Classification	Mean Squared Error		
Seq Len	1	10	100	1000	1	10	100	1000
LSTM	98.29%	1.14	181.06	19883	0.0%	168.18	321738	38761851
GRU	99.02%	1.12	180.95	19886	0.0%	168.09	321826	38784947
RNN-tanh	38.91%	1.53	226.95	20346	0.0%	167.19	321841	38784910
RNN-ReLU	9.80%	0.54	160.80	19608	88.18%	4.29	882.87	10969180
NAC	**99.23%**	**0.03**	**0.26**	**3**	**97.58%**	**2.82**	**28.11**	**280.89**
NALU	97.62%	0.08	0.90	17	77.73%	18.22	1199.12	114303

Table 10: Accuracy of the MNIST Counting & Addition tasks for series of length 1, 10, 100, and 1000.

Language to Number Translation Tasks

- Aim: To make clear whether representations of number words are learned in a systematic way.

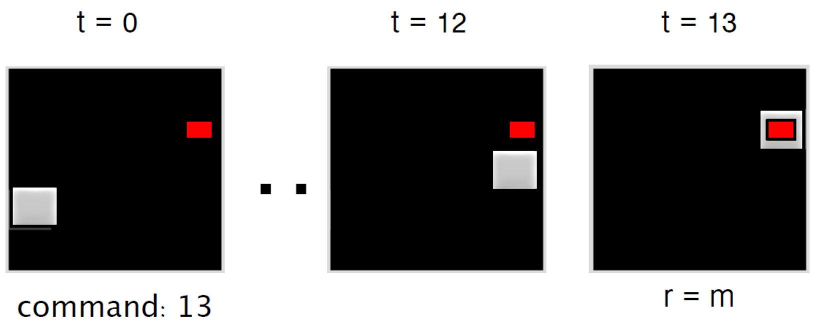


Model	Train MAE	Validation MAE	Test MAE
LSTM	0.003	29.9	29.4
LSTM + NAC	80.0	114.1	114.3
LSTM + NALU	0.12	**0.39**	**0.41**

Table 3: Mean absolute error (MAE) comparison on translating number strings to scalars. LSTM + NAC/NALU means a single LSTM layer followed by NAC or NALU, respectively.

Learning to Track Time in a Grid-World Environment

- Aim: In this task, we test whether a NAC can be used “internally” by an RL-trained agent to develop more systematic generalization to quantitative changes in its environment.



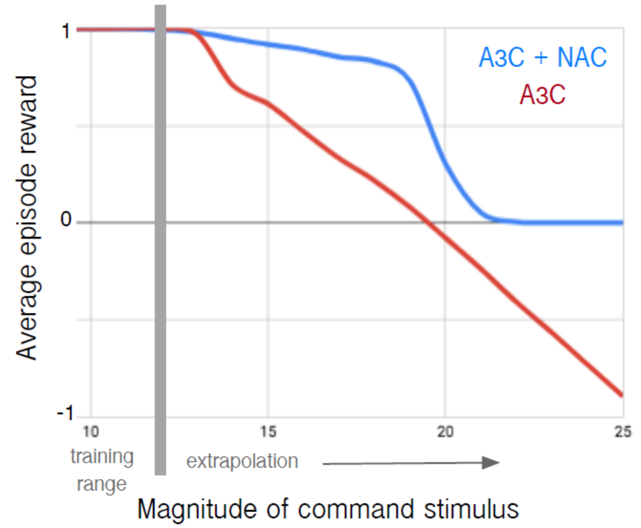
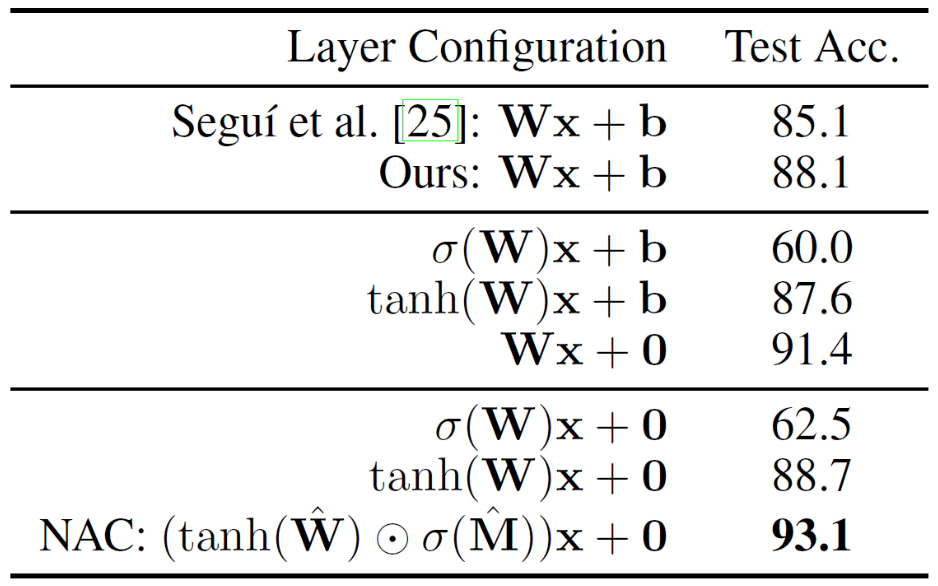


Figure 5: (above) Frames from the gridworld time tracking task. The agent (gray) must move to the destination (red) at a specified time. (below) NAC improves extrapolation ability learned by A3C agents for the dating task.

MNIST Parity Prediction Task & Ablation Study

- Aim: To show performance of replacing fully-connected layer.



Layer Configuration	Test Acc.
Seguí et al. [25]: $\mathbf{W}\mathbf{x} + \mathbf{b}$	85.1
Ours: $\mathbf{W}\mathbf{x} + \mathbf{b}$	88.1
$\sigma(\mathbf{W})\mathbf{x} + \mathbf{b}$	60.0
$\tanh(\mathbf{W})\mathbf{x} + \mathbf{b}$	87.6
$\mathbf{W}\mathbf{x} + \mathbf{0}$	91.4
$\sigma(\mathbf{W})\mathbf{x} + \mathbf{0}$	62.5
$\tanh(\mathbf{W})\mathbf{x} + \mathbf{0}$	88.7
NAC: $(\tanh(\hat{\mathbf{W}}) \odot \sigma(\hat{\mathbf{M}}))\mathbf{x} + \mathbf{0}$	**93.1**

Table 4: An ablation study between an affine layer and a NAC on the MNIST parity task.

Disadvantage

- There are still some big problems in the model.
- Bad writing.

Bad Writing

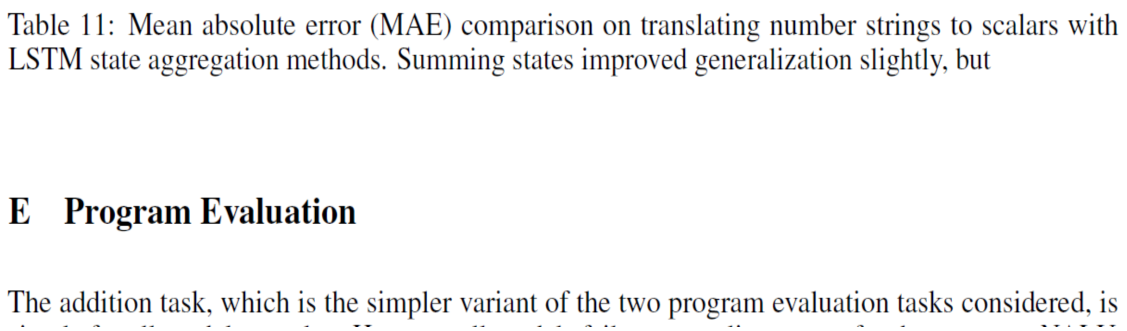
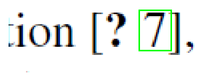


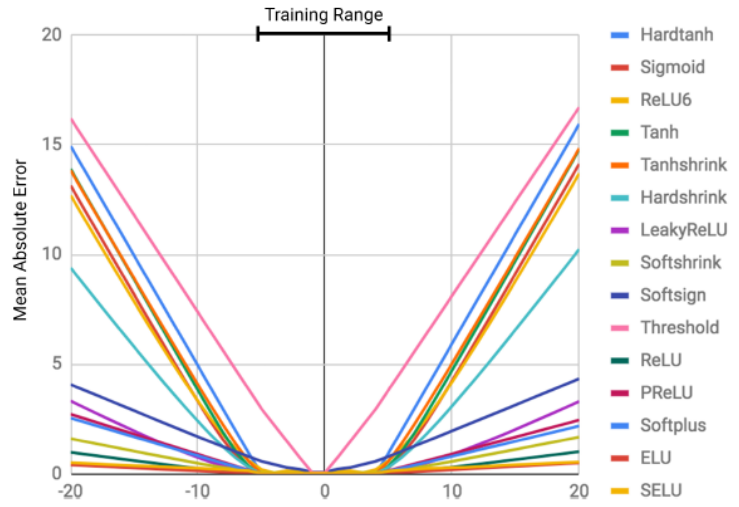
Table 11: Mean absolute error (MAE) comparison on translating number strings to scalars with LSTM state aggregation methods. Summing states improved generalization slightly, but

E Program Evaluation

The addition task, which is the simpler variant of the two program evaluation tasks considered, is



ion [? 7],



Future

- To find the max width of extrapolation range.
- To solve the division problem.
- To solve the gradient problem.