Neural Arithmetic Logic Units

Andrew Trask, Felix Hill, Scott Reed, Jack Rae, Chris Dyer, Phil Blunsom

November 29, 2018

Table of contents

Introduction

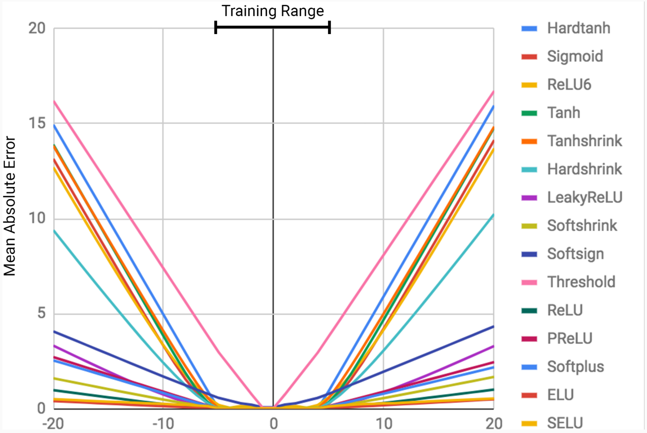
Problem: Neural networks can’t count!

Numerical Extrapolation Failures in Neural Networks

Consider the behavior of various MLPs trained to learn the scalar identity function, i.e., $f(x) = x$

- They trained an autoencoder to take a scalar value as input (e.g., the number 3).
- Then reconstruct the input value as a linear combination of the last hidden layer (3 again).
- They trained 100 models to encode numbers between -5 and 5 and average their ability to encode numbers between -20 and 20.

Numerical Extrapolation Failures in Neural Networks



Numerical Extrapolation Failures in Neural Networks

- All nonlinear functions fail to learn to represent numbers outside of the range seen during training.
- The severity of the failure directly corresponds to the degree of non-linearity within the chosen activation function.
- Some activations learn to be highly linear (such as PReLU) which reduces error somewhat.
- Sharply non-linear functions such as sigmoid and tanh fail consistently.

The Neural Accumulator & Neural Arithmetic Logic Unit

Neural Accumulator (NAC)

The **neural accumulator** (NAC), is a special case of a linear (affine) layer whose transformation matrix $\mathbf{W}$ consists just of -1's, 0's, and 1's; that is, its outputs are additions or subtractions (rather than arbitrary rescalings) of rows in the input vector. This prevents the layer from changing the scale of the representations, meaning that they are consistent throughout the model.

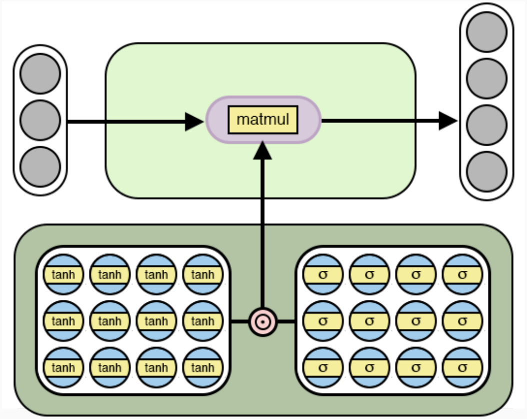
Neural Accumulator (NAC)

Problem: A hard constraint enforcing that every element of W be one of $\{-1, 0, 1\}$ would make learning hard.

Solution: They propose a continuous and differentiable parameterization of W in terms of unconstrained parameters: $\mathsf{W} = \tanh(\hat{\mathsf{W}}) \odot \sigma(\hat{\mathsf{M}})$.

- This form is convenient for learning with gradient descent and produces matrices whose elements are guaranteed to be in $[-1, 1]$ and biased to be close to -1, 0, or 1.
- The model contains no bias vector, and no squashing nonlinearity is applied to the output.

Neural Accumulator (NAC)



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

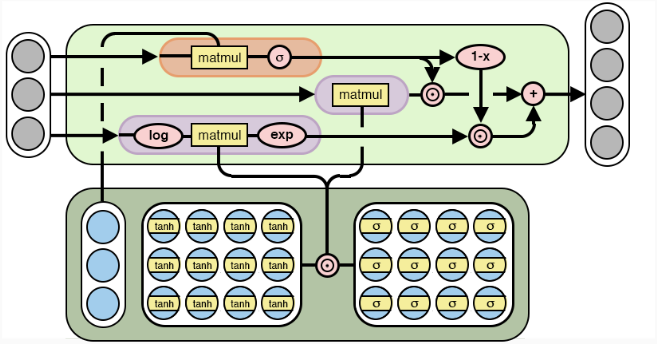
The Neural Arithmetic Logic Unit (NALU)

A similarly robust ability to learn more complex mathematical functions, such as multiplication, may be be desirable. The **neural arithmetic logic unit** (NALU), which learns a weighted sum between two subcells:

1. One capable of addition and subtraction.
2. The other capable of multiplication, division, and power functions such as $\sqrt{x}$.

As with the NAC, there is the same bias against learning to rescale during the mapping from input to output.

The Neural Arithmetic Logic Unit (NALU)



NALU: $y = g \odot a + (1 - g) \odot m \quad m = \exp W(\log(|x| + \epsilon)),\; g = \sigma(Gx)$

The Neural Arithmetic Logic Unit (NALU)

The NALU consists of two NAC cells (the purple cells) interpolated by a learned sigmoidal gate $\mathbf{g}$ (the orange cell), such that if the add/subtract subcell's output value is applied with a weight of 1 (on), the multiply/divide subcell's is 0 (off) and vice versa.

- The first NAC (the smaller purple subcell) computes the accumulation vector $\mathbf{a}$, which stores results of the NALU's addition/subtraction operations.
- The second NAC (the larger purple subcell) operates in log space and is capable of learning to multiply and divide, storing its results in $\mathbf{m}$ (ϵ prevents $\log 0$).

This cell can learn arithmetic functions consisting of multiplication, addition, subtraction, division, and power functions in a way that extrapolates to numbers outside of the range observed during training.

Experiments

Simple Function Learning Tasks

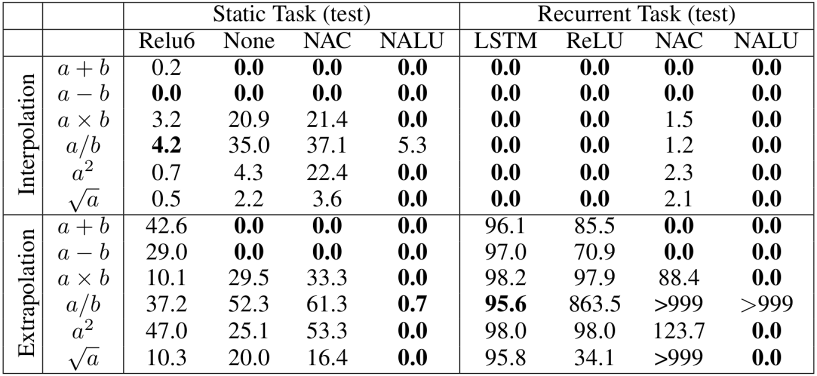
They have two task variants:

1. **The static tasks**: The inputs are presented all at once as a single vector.
2. **The recurrent tasks**: Inputs are presented sequentially over time

- Inputs are randomly generated, and for the target, two values (a and b) are computed as a sum over regular parts of the input.
- An operation (e.g., $a \times b$) is then computed providing the training (or evaluation) target.

The model is trained end-to-end by minimizing the squared loss, and evaluation looks at performance of the model on held-out values from within the training range (interpolation) or on values from outside of the training range (extrapolation).

Simple Function Learning Tasks



		Static Task (test)				Recurrent Task (test)			
		Relu6	None	NAC	NALU	LSTM	ReLU	NAC	NALU
Interpolation	$a+b$	0.2	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
	$a-b$	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
	$a \times b$	3.2	20.9	21.4	**0.0**	**0.0**	**0.0**	1.5	**0.0**
	a/b	**4.2**	35.0	37.1	5.3	**0.0**	**0.0**	1.2	**0.0**
	a^2	0.7	4.3	22.4	**0.0**	**0.0**	**0.0**	2.3	**0.0**
	$\sqrt{a}$	0.5	2.2	3.6	**0.0**	**0.0**	**0.0**	2.1	**0.0**
Extrapolation	$a+b$	42.6	**0.0**	**0.0**	**0.0**	96.1	85.5	**0.0**	**0.0**
	$a-b$	29.0	**0.0**	**0.0**	**0.0**	97.0	70.9	**0.0**	**0.0**
	$a \times b$	10.1	29.5	33.3	**0.0**	98.2	97.9	88.4	**0.0**
	a/b	37.2	52.3	61.3	**0.7**	**95.6**	863.5	>999	>999
	a^2	47.0	25.1	53.3	**0.0**	98.0	98.0	123.7	**0.0**
	$\sqrt{a}$	10.3	20.0	16.4	**0.0**	95.8	34.1	>999	**0.0**

Scores are scaled relative to a randomly initialized model for each task such that 100.0 is equivalent to random, 0.0 is perfect accuracy, and >100 is worse than a randomly initialized model.

Language to Number Translation Tasks

It is not clear whether representations of number words are learned in a systematic way. To test this, they created a new translation task which translates a text number expression (e.g., `five hundred and fifteen`) into a scalar representation (515).

- They trained and tested using numbers from 0 to 1000.
- The training set consists of the numbers 0–19 in addition to a random sample from the rest of the interval, adjusted to make sure that each unique token is present at least once in the training set.
- There are 169 examples in the training set, 200 in validation, and 631 in the test test.

All networks trained on this dataset start with a token embedding layer, followed by encoding through an LSTM, and then a linear layer, NAC, or NALU.

Language to Number Translation Tasks

“three	hundred	and	thirty	four”
3.05	299.9	301.3	330.1	334

“seven	hundred	and	two”
6.98	699.9	701.3	702.2

“eighty	eight”
79.6	88

“twenty	seven	and	eighty”
18.2	27.0	29.1	106.1

Here are the intermediate states of the NALU on randomly selected test examples. Without supervision, the model learns to track sensible estimates of the unknown number up to the current token. This allows the network to predict given tokens it has never seen before in isolation, such as e.g. `eighty`, since it saw `eighty one`, `eighty four` and `eighty seven` during training.

Language to Number Translation Tasks

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**

- They observed that both baseline LSTM variants overfit severely to the 169 training set numbers and generalize poorly.
- The LSTM + NAC performs poorly on both training and test sets.
- The LSTM + NALU achieves the best generalization performance by a wide margin, suggesting that the multiplier is important for this task.

Thank you!