# **Stochastic computing structure**

## **3-to-1 majority gate**

To adapt floating-point computations for stochastic computing in convolutional neural networks (CNNs), we must carefully design the network architecture. In stochastic computing, the majority (MAJ) gate with a zoom factor is employed to compress large-scale sequences into a single sequence, replacing traditional mean or sum operations.

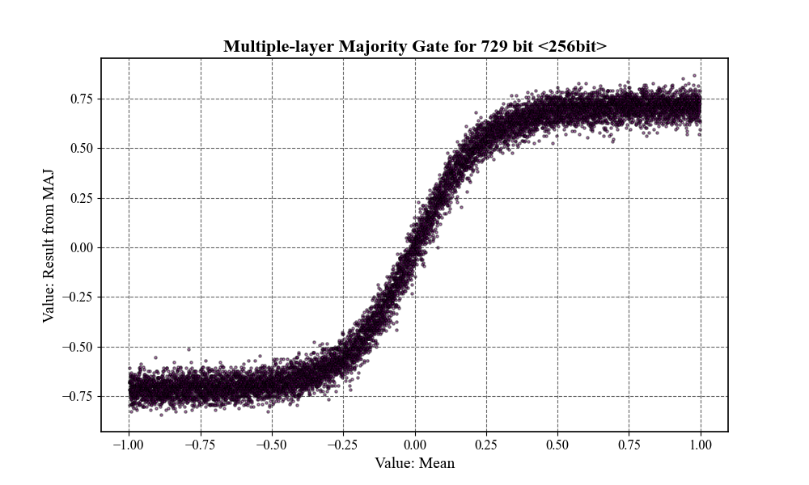
We propose a fundamental 3-to-1 MAJ gate structure, which takes three inputs and produces one output, with an adjustable zoom factor. Let the three inputs be denoted as P1​, P2​, and P3​. The mathematical expression for the 3-to-1 MAJ gate is given by: P1\*P2\*(1-P3)+P1\*(1-P2)\*P3+(1-P1)\*P2\*P3+P1\*P2\*P3.

Larger MAJ structures can be constructed by stacking multiple 3-to-1 MAJ gates in a layered manner, enabling the processing of more complex sequences.

## **Convolution layer**

To perform feature extraction in a stochastic computing framework, we design an equivalent convolution layer. In this scheme, the XNOR operation serves as an effective substitute for multiplication for values in the range [−1,1], allowing us to compute the product of weights and input features. Following the XNOR operation, the results from the convolution kernel and input channels must be compressed into a single output value. To achieve this, we introduce a stacked MAJ (Stack-MAJ) structure.

The 3-to-1 MAJ gate exhibits characteristics similar to the *tanh(x)*  function, polarizing inputs toward −1 or 1. However, after several layers of Stack-MAJ operations, the behavior approximates *tanh(α\*x)* with a large α, leading to aggressive polarization. This can negatively impact training by causing gradient saturation. To mitigate this effect, we introduce a zoom factor: after each MAJ layer, the output is divided by the zoom factor to reduce polarization. As shown in Figure 1, within the range [−0.25,0.25], the Stack-MAJ with a zoom factor of 0.8 exhibits a near-linear relationship, improving training stability.



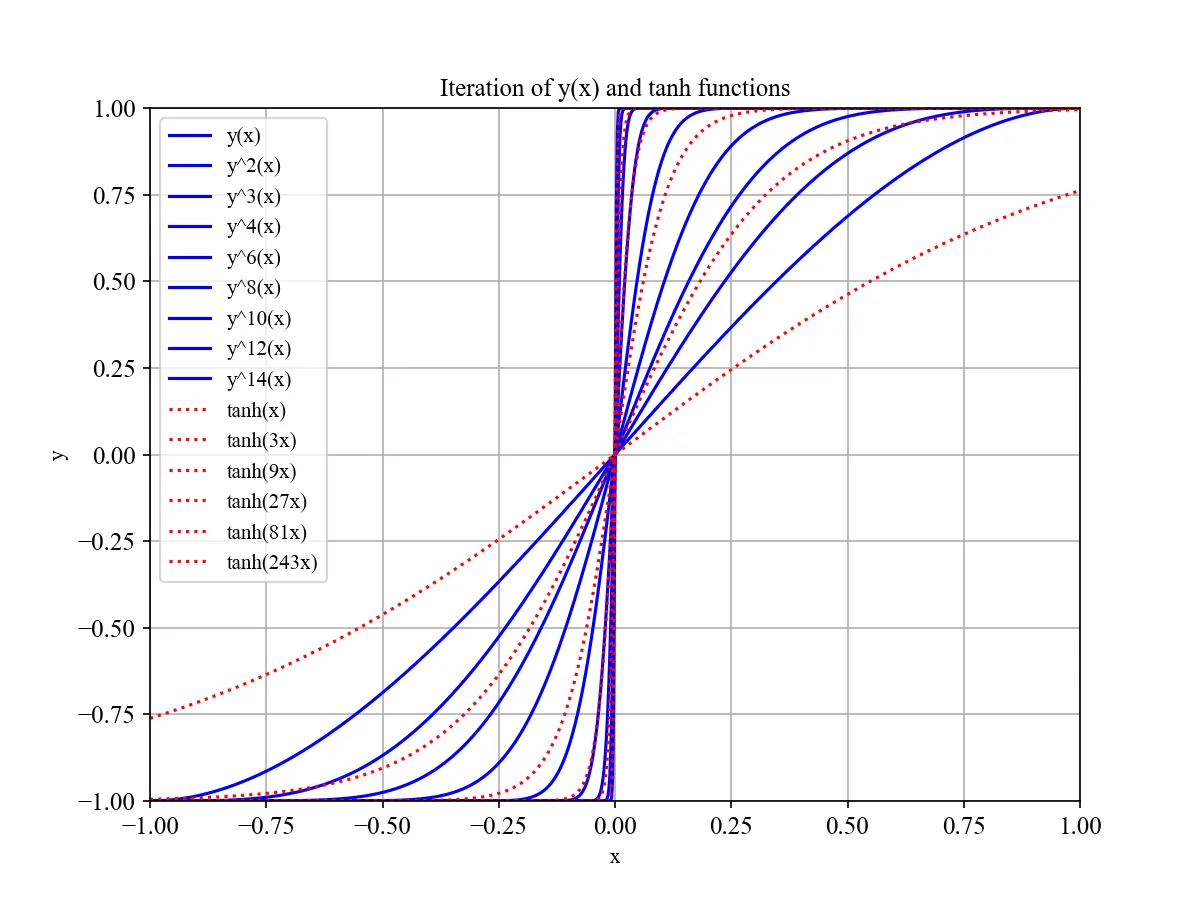
*Figure 1 effect of MAJs with zoom factor as* *0.8*

## **Activation layer**

The activation layer in our design is also implemented using the Stack-MAJ structure. A key requirement in stochastic computing is that all layer inputs and outputs must be constrained to the range [−1,1]. The tanh function is a natural choice for this purpose due to its inherent range restriction. However, because the convolution step compresses data, the gradient magnitude is significantly reduced. For instance, in a convolution layer with 27 input channels and a 3\*3 kernel, the output can be considered an average of 3\*3\*27 (243) values, reducing the gradient by a factor of 1/243 during back-propagation.

To address this gradient vanishing problem, we propose an indexed tanh function, *tanh(α\*x)*, where α is a scaling factor. For the example above, using *tanh(243\*x)* as the activation function after the convolution layer can significantly improve training dynamics by amplifying the gradient.

However, directly implementing *tanh(α\*x)* in stochastic computing is impractical. Instead, we leverage the Self Stack-MAJ structure, which meets two critical requirements: activating convolution outputs and amplifying gradients through multiple zoom operations. Figure 2 illustrates the relationship between the Stack-MAJ (different layers) and the tanh function (different coefficients), demonstrating their similarity.



*Figure 2 relationship of Stack-MAJ and tanh function*

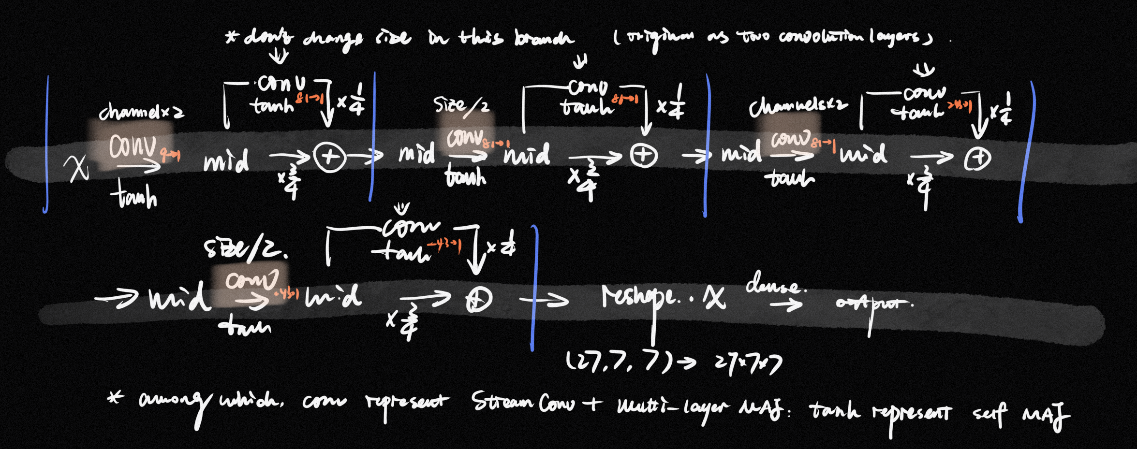
Based on this, we derive a suitable activation expression. For a convolution layer with n = kernel size \* in-channels, the ideal activation is tanh(n\*x). We approximate this by applying 3\*log3(n) layers of 3-to-1 MAJ gates to implement the Self Stack-MAJ. Since stochastic sequences are inherently random, we introduce bit displacement at each layer to generate distinct representations of the same value. This allows us to input three identical stochastic sequences into the MAJ gate, producing the Self Stack-MAJ result.

## **Fully-connected layer**

The fully-connected layer follows a structure similar to the convolution layer, consisting of a multiplication step and an addition step. We implement the multiplication using XNOR operations and the addition using the Stack-MAJ structure, ensuring compatibility with the stochastic computing framework.

# **CNN structure**

To address the challenges of gradient vanishing and the [−1,1] value constraint in stochastic computing, we incorporate residual connections into our CNN architecture. Residual networks help mitigate gradient degradation by allowing the network to learn identity functions, ensuring that the gradient can flow through the network more effectively. The detailed architecture of our proposed CNN is illustrated in Figure 3.



*Figure 3 Network Structure*

*(\*adjustment: different channel numbers, different residual connection weight, more convolution blocks. Check code for details.)*

# **Training method**

The use of Stack-MAJ to amplify gradients enables the network to adapt to a relatively high initial learning rate. We set the initial learning rate to 0.001 and reduce it by half if the total loss does not decrease below the recorded minimum loss for five consecutive epochs. Whenever a new minimum loss is achieved, we update the best model checkpoint.

# **Result**

After training for 100 epochs, we obtained a best model with a floating-point test accuracy of 98.59%. This accuracy is notably high given the depth of the network, demonstrating the effectiveness of our design.

# **Problems**

Despite the promising results, we encountered a significant issue: the accuracy for fully stochastic sequences was unexpectedly low. To investigate, we conducted experiments with hybrid implementations. When the first convolution layer was computed using stochastic computing (512-bit sequences) and the remaining layers used floating-point computation, the accuracy reached 97.3%, indicating that the convolution operation functions correctly in isolation. However, when the first two convolution layers were computed using stochastic computing (512-bit sequences) with the remaining layers in floating-point, the accuracy dropped dramatically to approximately 20%.

However, a fully stochastic multi-convolution network achieved relatively high accuracy in separate experiments. This discrepancy suggests the presence of a potential bug in the convolution implementation when transitioning between stochastic and floating-point computations!

# **Promotions (maybe)**

1. The maj3 function iteration can be made adjustable to create the similar effect of tanh(α\*x) with a gradually increasing a. Potentially improving performance.
2. The parameter of addition can be learnable. In this case, we use mid\*1/2 + x\*1/2 for residual connection. But if the network goes deeper, this fixed weighting may exacerbate gradient vanishing. Introducing learnable parameters for the residual combination could enable the construction of deeper networks, such as VGGNet.
3. The current fully-connected layer uses a simple summation in the dense operation. Replacing this with the Stack-MAJ component could improve consistency and performance in the stochastic computing framework.