Results on Interval Arithmetic Applied to Neural Networks

Lieutenant Commander Guillaume Berthelot

August 2024

We consider interval arithmetic applied to neural networks.

Theorem 1. *Let a neural network be given. Let A_ϵ be a tensor of noise symbols for interval arithmetic. Let f be the function such that the image of the tensor through the network is given by $f(a_{\epsilon_i})$ for each element of A_ϵ.*

If there exists a basis of dimension n on which the projection of A_ϵ is injective and if there exists a matrix W representing f on this basis, i.e., for all i belonging to $\{1, \ldots, n\}$, $f(a_{\epsilon_i}) = W a_{\epsilon_i} e_i$,

Then,

$$Z^+ = |W||A_\epsilon|$$

is a bounding vector for interval arithmetic. If C is the center vector, then the bounds are given by $C \pm Z^+$.

Proof. Let p be the number of elements in A_ϵ, we have

$$Z^+ = \sum_{i=1}^{p} |f(a_{\epsilon_i})| = \sum_{i=1}^{p} |W a_{\epsilon_i} e_i| = |W| \sum_{i=1}^{p} |a_{\epsilon_i} e_i|$$

hence the result. □

Lemma 1. *There exists a canonical basis in which the so-called linear operations of neural networks are representable by a matrix. This is the basis generated by the flattening operation.*

Lemma 2. *If p is the tensor of approximation coefficients of an activation function, then the image of p in the canonical basis is the flattening of p.*

Corollary 1. *If $L_1, R_1, L_2, R_2, \ldots, L_n, R_n$ is a sequence of linear layers and activations, and if A_ϵ is a tensor of noise symbols for interval arithmetic that projects injectively onto the canonical basis of L_1,*

If $p_1, p_2, \ldots, p_n$ are the approximation tensors projected in their respective canonical bases, then the image A_s of A_ϵ for interval arithmetic is given by

$$Z_s = |W_r||A_\epsilon| = |((p_n W_n) \otimes (p_{n-1} W_{n-1}) \otimes \cdots \otimes (p_1 W_1))||A_\epsilon|$$

In other words, it is possible to reduce the computation of interval arithmetic applied to the entire network to a matrix product.

Proof. Let L_n, R_n be the last two linear and activation layers,

Let A_{n-1} be the tensor of input symbols, and A_s the image of the input tensor by the linear approximation of the L_n, R_n pair. Then $A_s = f_n(p_n A_{n-1}) = p_n W_n A_{n-1}$ then recursively $= p_n W_n p_{n-1} W_{n-1} A_{n-2} =$

$$\ldots = ((p_n W_n) \otimes (p_{n-1} W_{n-1}) \otimes \cdots \otimes (p_1 W_1)) A_\epsilon$$

But A_ϵ projects injectively onto its canonical space.

Hence the result. □

Theorem 2. *Let d be the noise tensor generated by the approximation of an activation layer. Then d projects injectively onto the canonical basis.*

Proof. The approximation operation is defined. □

By applying the previous results, it is possible to reduce the evaluation of interval arithmetic to a matrix product.

Theorem 3. *Let $L_1, R_1, L_2, R_2, \ldots, L_n, R_n$ be a sequence of linear layers and activations, let j be an intermediate layer L_j. Then the bounds for interval arithmetic for the activation layer R_j are given by*

$$C_j \pm \sum_{l=1}^{j} |W_{r_i}||A_{d_i}|$$

where

$$W_{r_i} = (W_j) \otimes (p_{j-1} W_{j-1}) \otimes \cdots \otimes (p_i W_i) \quad \forall i \in \{1, \ldots, j\}$$

Proof. This result is immediate. □

Corollary 2. *Let a network consist of linear layers and activation functions. Algorithm 1 is an affine approximation algorithm for this network in polynomial time.*

Algorithm 1 Affine Approximation Algorithm for the Network

Step 1:
for each linear layer **do**
    Calculate W_l
    Create an empty list L_W
end for
Step 2:
Choose an input x
for each dimension of x **do**
    Establish a noise level δ
end for
Create the vector A_δ
Create a list A with the first element A_δ
Initialize a unit approximation vector p
for each layer of the network, in increasing order **do**
    if linear layer **then**
        Calculate the center
        for each element of the list L_W **do**
            Multiply it by $p \times W_l$ on the left side
        end for
        Add W_l to the list L_W
        Create a copy $|L_W|$
        for each pair of elements $(|W_L|, |A|)$ **do**
            Stack the sum of the products: result Z^+
        end for
        Overwrite p with a unit vector of the output dimension
    else if activation layer **then**
        Calculate the bounds and store the result
        Define p, q, d
        Shift the center
        Add A_d to the list A
    end if
end for