1 Notation

1.1 Mathematical formulation

Consider a linear layer $\mathbb{R}^N \to \mathbb{R}^M$ (where $N \in \mathbb{N}$ for FC nets and $N \in \mathbb{N}_x \times \mathbb{N}_y \times \mathbb{N}_{channel}$ for conv nets).

The pair $(a_0, A), a_0 \in \mathbb{R}^N, A \in \mathbb{R}^{N \times k}$ represents the input zonotope $Z = \{a_0 + A \cdot \varepsilon, \varepsilon \in [-1, 1]^k\}$. We used the notation $A \cdot \varepsilon = (A_n \cdot \varepsilon)_{n \in [N]}$, where $A_n \in \mathbb{R}^k$ and $A_n \cdot \varepsilon \in \mathbb{R}$ is the regular dot-product.

Note that k is the dimension of the zonotope, i.e the number of error terms. We will typically index over $n \in [N]$ and $l \in \{1, ..., k\}$. $n \in [N]$ means $n \in \{1, ..., N\}$ if $N \in \mathbb{N}$, or $(n_x, n_y, n_c) \in [|N_x|] \times [|N_y|] \times [|N_c|]$ for conv nets.

1.2 PyTorch tensor notation

The mathematical objects described above are written resp. as:

- $Z = (a_0, A)$: `Zonotope(a0, A)`
- $\mathbb{R}^N$: tensors of shape `s` ($\simeq N$)
- $a_0 \in \mathbb{R}^N$: a Tensor `a0` of shape `s`
- $A \in \mathbb{R}^{N*k}$: a Tensor `A` of shape `(k, *s)` (we placed `k` first to pass through `torch.nn.Conv2d` layers easily, cf section 2)
- $\varepsilon \in [-1, 1]^k$ is never written explicitly in the code, since we work with zonotopes
- $l \in \{1, ..., k\}$: obviously we index `0<=l<k` instead. But in general it shouldn't be necessary, as everything can be written as vector operations
- $A_n \in \mathbb{R}^N$: `A[n, :]`, where `n` is a tuple of length `len(s)` (i.e 1 for FC nets and 3 for CN nets). But again, it shouldn't be necessary, writing vector operations should suffice.

Tensors used in Conv2d are of shape (N, C_{in}, H, W) and $(N, C_{out}, H_{out}, W_{out})$ where

- N: batch size (in our case, it's the number of error terms)
- C: number of channels
- H, W: dimensions of the layer

2 Convolutional zonotope transformation

We can compute the zonotope approximation of the convolutional layers by considering each ε separately and computing the convolution of the coefficients corresponding to that ε.

Let I be the unknown input tensor of the convolutional layer, K be the filter and A be I's coefficients in the zonotope approximation. K refers to the filter for a single channel. Neurons for a given layer are indexed by $[x, y]$.

$$(I \times K)[x, y] = \sum_{i=0}^{m} \sum_{j=0}^{m} I[x + i - m/2, y + j - m/2] * K[i, j]$$

But $I[x, y] = \sum_{k=1}^{n} A_k[x, y] * \varepsilon_k + A_0[x, y]$. So:

$$(I \times K)[x, y] = \sum_{i=0}^{m} \sum_{j=0}^{m} (\sum_{k=1}^{n} A_k[x+i-m/2, y+j-m/2] * \varepsilon_k + A_0[x+i-m/2, y+j-m/2]) * K[i, j]$$

Now I distribute the multiplication with the filter into the sum and change the order of the summations to get:

$$(I \times K)[x, y] = \sum_{k=1}^{n} \sum_{i=0}^{m} \sum_{j=0}^{m} A_k[x+i-m/2, y+j-m/2] * K[i, j] * \varepsilon_k + A_0[x+i-m/2, y+j-m/2] * K[i, j]$$

And now I take out the ε from the inner summation to get:

$$(I \times K)[x, y] = \sum_{k=1}^{n} (\sum_{i=0}^{m} \sum_{j=0}^{m} A_k[x+i-m/2, y+j-m/2] * K[i, j]) * \varepsilon_k + \sum_{i=0}^{m} \sum_{j=0}^{m} A_0[x+i-m/2, y+j-m/2] * K[i, j]$$

As we can see from the equation, we have a zonotype where the center is given by a convolution over the centers of the input zonotope, A_0. While the coefficients of each ε are just a convolution over the coefficients of that ε in the input:

$$(I \times K)[x, y] = \sum_{k=1}^{n} (A_k \times K)[x, y] * \varepsilon_k + (A_0 \times K)[x, y]$$

It suffices to compute the $k + 1$ convolutions of A and K, which can be done efficiently using pytorch.

3 Loss function

Output layer: $[o_0, o_2, ...o_9]$ (the logits for the MNIST digit classification)

3.1 Sum of violations

Zonotope approximation of verification objective (target t):

$$Z = \sum_{i=0}^{9} \max(o_i - o_t)$$

$Z > 0$ only if one or more of the o_i is greater that o_t.

In particular if we compute the upper bound on Z:

$$L = max_{\varepsilon} Z$$

If $L = 0$ then $o_t \geq o_i \forall i$ which is the property that we want to verify.

Else: $L > 0$ and we could minimize L by gradient descent with respect to lambdas.

In order to do that, we could build the entire Zonotope approximation with pythorch tensors and operators and then use it to compute gradients with respect to the lambdas.

3.2 Other losses

Consider $x = (x_i)_i = (o_i - o_t)_{0 \geq i \geq 9}$ (given target t). The robustness property can be verified from the output zonotope Z if $\forall (o_0, ..., o_9) \in Z, x_i \leq 0 \forall i$

We can take as concrete loss function any function $f(x)$ be such that

$$f(x) \leq 0 \implies \forall i, x_i \leq 0$$

For example:

- the sum of violations $f(x) = x_0^+ + ... + x_9^+$ as described previously
- the max violation $f(x) = max_i x_i$
- any weighted sum of violations with positive weights $w_0 ... w_9 > 0$

An important criterion is that it should be easy to transform zonotopes, i.e to compute $Z' = f^{\#}(Z)$ (where Z' is the zonotope for the scalar loss). For this reason, it seems that the sum of violations is a good choice (though it requires to perform a relu transformations).

Finally, the final loss that is used in the optimizer will be $max_{y \in Z'} y$ i.e the upper bound of Z', same as in the previous subsection.