Verifying neural networks with precise DeepPoly relaxation

Peshal Agarwal & Pushpak Pati
December 18, 2020

Abstract

We aim to improve the ReLU abstract transformation of DeepPoly by learning the slope of the polyhedra by gradient backpropagation. We perform multiple gradient updation steps to verify the network for any given input and the magnitude of maximum perturbation.

1 Implementation

We define abstract transformers for each of the layers in the network (see `code/layers_fc.py` and `code/layers_conv.py`). For each of the transformers we store the coefficients (l_x, u_x) and the intercepts (l_c, u_c) for the lower and upper bounds, as well as the concrete lower (l) and upper (u) bound values. Note that to obtain precise bounds of the neuron values, we need to back substitute. We pass a list of coefficients, intercepts of lower and upper bounds, and $l$and u of all the previous layers to each transformer to facilitate back substitution whenever required.

1.1 Normalization

We do not set any coefficients or intercepts rather simply set the concrete lower and upper bounds as: $l \leftarrow (l-\mu)/\sigma$ and $u \leftarrow (u-\mu)/\sigma$

1.2 Linear

The coefficients (l_x, u_x) and intercepts (l_c, u_c) of the lower and upper bounds are the weight and bias of the current layer. We calculate l and u with back substitution.

1.3 ReLU

For strictly negative neurons ($u < 0$), we set $l_x \leftarrow 0$, $u_x \leftarrow 0$, $l_c \leftarrow 0$, and $u_c \leftarrow 0$.
For strictly positive neurons ($l > 0$), we set $l_x \leftarrow 1$, $u_x \leftarrow 1$, $l_c \leftarrow 0$, and $u_c \leftarrow 0$.
For crossing neurons ($l < 0$ and $u > 0$), we set $l_c \leftarrow 0$. We set two slopes (l_x, u_x) as learnable parameters while ensuring the range between 0 and 1. u_c is updated as per the current value of slope (u_x). We initialize the u_x as defined in the slides and initialize l_x as 1. Note that we do not need to back substitute till the input layer to calculate l and u for this layer since ReLU acts individually on each neuron. We can directly use values of l and u from the previous layer, and coefficients (u_x) and intercepts (u_c) of the current layer to compute l and u. Specifically, $l \leftarrow max(0, l)$ and $u \leftarrow u * u_x + u_c$.

1.4 Convolution

We flatten the input and the output dimensions and calculate the corresponding affine transform for the convolution operation. The coefficients and intercepts of the affine transform act as coefficient and intercept for the lower and upper bound equations. We then perform back substitution to obtain l and u, exactly same as in the Linear layer.

1.5 Verification

We create a set of nine lower and upper bounds coefficients and intercepts by subtracting the logits of the incorrect classes from the correct class logit. This allows us to reuse the back substitution function to calculate the final nine lower bounds. The loss is defined as the absolute sum of all the negative lower bounds and is minimized using `Adam` optimizer (see `code/deeppoly_fc.py` and `code/deeppoly_conv.py`). We output "Verified" as soon as the loss is zero and "Not Verified" otherwise.