1 Introduction

1.1 Abstract

We want to create an algorithm that modifies neural networks by reseting (zero-filling) part of weights. We want the modified network to be more efficient than primary and return similar results.

1.2 Idea

We pessimistically estimate error coming from zero-filling of weights. For given number x, we modify given neural network so that any output varies up to x. We estimate partial derivatives of outputs with respect to weights in network. Algorithm is not optimal. For every layer, we estimate activations of neurons from activations from previous layer. Then we estimate derivatives of weights from derivatives of weights in further layers.

1.3 High-level algorithm

Algorithm consists of 4 phases:

Phase 1: For any neuron, we calculate its range of activation for inputs of neural networks in given range. For simplicity, we assume it is [0, 255]. We do it from the beginning to the end of the network.

Phase 2: For any neuron, we calculate range of partial derivatives of outputs with respect to activation of this neuron. We do it from end to the beginning of network.

Phase 3: For any weight we calculate range of partial derivatives of outputs with respect to this weight.

Phase 4: We choose weights to reset so that error on any output does not exceed the given limit.

1.4 Assumptions

- We assume that for primary and modified network, partial derivative of any output with respect to any weights is constant. This approximation could be examined.
- We consider following types of layers: convolutional, fully-connected, max-pool, relu, softmax, norm.

1.5 Auxiliary assumptions and notation

1.5.1 Auxiliary assumptions

- We realize phases 1, 2 and 3 for any output of network. In any iteration we consider impact of reseting the weights on a specific output of network and thus in description of phases we will assume only one output of network.
- For simplicity of description, we consider 1-dimensional vectors of activation, ranges etc. of neurons in layers. It is different from given implementation but useful for easier understanding of the algorithm.
- We use arithmetical operators "$+$", "$-$", "$\cdot$", "$\div$" and power on intervals (see: interval arithmetic).
- Weight $\neq$ edge. Edges of convolutional layers share weights. Edge has direction from the beginning to the end in direction of flow of data in neural network.

1.5.2 Notation

- $[x, y]$ - interval of real numbers from x to y
- $low(r)$ - lower bound of interval r
- $high(r)$ - upper bound of interval r
- $[E]$ - 1 if expression E is true, 0 otherwise (Iversion bracket)
- $\iota x.(E(x))$ - the only x for which expression $E(x)$ is true
- n^i - i-th layer of neurons
- v_x - x-th element of vector v
- we also regard v as set of its elements.
- $a(x)$ - activation of neuron x
- $range(x)$ - interval of possible values of activation of neuron x
- $input(x)$ - for given neuron x, set of neurons which outputs are x's inputs
- $output(x)$ - for given neuron x, set of neurons which inputs are x's outputs
- $ed\_input(x)$ - for given edge x, neuron with output x
- $ed\_output(x)$ - for given edge x, neuron with input x
- $ed(x)$ - for given weights x, set of edges that share weight x
- $soft\_prev(x)$ - for given neuron x in softmax layer, neuron from previous layer connected to x, impacting x positively

- $soft\_next(x)$ - for given neuron x in layer before softmax, neuron from softmax layer positively impacted by x
- $soft\_prev\_other(x)$ - for given neuron x in softmax layer, $input(x) - soft\_prev(x)$
- $soft\_next\_other(x)$ - for given neuron x in layer before softmax, $output(x) - soft\_next(x)$
- $w(x, y)$ - weights of edge connecting neurons x and y
- L - number of layers in network
- C - output of network
- $\delta(x)$ - for given neuron x, interval of values $\frac{\partial C}{\partial a(x)}$ for possible activations of x

2 Algorithm

2.1 Phase 1

For simplicity of description we assume values on input is activation of 0'th layer of network. We assumed that:

$$\forall_{x \in n^0}(range(x) = [0, 255]) \tag{1}$$

We estimate intervals of activation of neurons of consecutive layers of network, based on estimation of previous layers. For given intervals of activation of neurons from previous layer, we calculate intervals of activation of neurons from next layer so that it is the shortest interval which contains all possible values of activation of a neuron from next layer for any activation of neurons from previous layer in intervals of activation of neurons in previous layer. We do it optimally. For neuron x from following layers:

- convolutional, fully-connected:

$$range(x) := \sum_{y \in input(x)} (w(x, y) \cdot range(y)) \tag{2}$$

- max-pool:

$$range(x) := [max(low(input(x))), max(high(input(x)))] \tag{3}$$

- relu:

$$range(x) := [max(\{0, low(input(x))\}), max(\{0, high(input(x))\})] \tag{4}$$

- softmax:

$$range(x) := [\frac{e^{low(soft\_prev(x))}}{e^{low(soft\_prev(x))} + \sum_{y \in soft\_prev\_other(x)} e^{high(y)}}, \\ \frac{e^{high(soft\_prev(x))}}{e^{high(soft\_prev(x))} + \sum_{y \in soft\_prev\_other(x)} e^{low(y)}}] \tag{5}$$

- norm: Consider two arguments of norm function, *sum* - sum of squares of neighbours of input corresponding to normalized value and v - normalized value. Function norm in monotonic with respect to *sum* so that:

$$(sum1 < sum2) \implies (norm(v, sum1)norm(v, sum2)) \tag{6}$$

Function norm is bitonic with respect to argument v. Let $c := (sum - v * v) * \alpha + k$. Maximum of norm(v, sum) is for $v = sqrt(2 * c/\alpha)$. Minimum is either for the lowest or the highest v from interval.

2.2 Phase 2

Let o be activation of neural network, that is activation of last neuron from network (we assume one output of network). We know that:

$$\delta(o) = [1, 1] \tag{7}$$

For any neuron x, we are able to estimate range of its impact on error if we have range of impact on error of neurons from $output(x)$ and correlation between x and neurons from $output(x)$. For better estimation, we consider range of activation of x and range of activation of neurons from $output(x)$. We do not estimate the range optimally. Estimation consists of calculating partial derivative in range of activation in points where it is lowest/highest. Examples:

- conv, fully-connected: (optimally)

$$\delta(x) := \sum_{y \in output(x)} w(y, x) \cdot \delta(y) \tag{8}$$

- max-pool: (optimally)

$$\delta(x) := \sum_{y \in output(x)} ((^{[[max(high(range(input(y)-\{x\})))\leq low(range(x))],}_{[max(low(range(input(y)-\{x\})))\leq high(range(x))]]}) \cdot \delta(y)) \tag{9}$$

- relu: (optimally)

$$\delta(x) := [[0 \leq low(range(x))], [0 \leq high(range(x))]] \cdot \\ \cdot \delta(\iota y.(y \in output(x))) \tag{10}$$

- softmax: For simplicity we assume softmax is the last layer of network. Then one output has partial derivative 1 and any other has 0. Estimation is optimum but requires this assumption.

$$\delta(x) := \delta(\iota y.(y \in output(x)) \tag{11}$$

- norm: (not optimally) Function is relatively complex. For now we do not indicate preferred solution.

2.3 Phase 3

For any weight we calculate range of partial derivative of output with respect to this weight. In convolutional layers a weight occurs in more than one edge:

$$\delta(w) = \sum_{i \in ed(w)} (range(ed\_input(i)) \cdot w \cdot \delta(ed\_output(i))) \tag{12}$$

After processing above we have a list of vectors (one vector for any weight w in network) with values of $\delta(w)$ for all outputs of network.

2.4 Phase 4

The choice of algorithm to pick weights to reset with range of their impact on final classification of particular classes is only a proposal. We encourage to use any other algorithm. Nevertheless phase 4 is not essence of presented algorithm. We have list from phase 3. Let o be number of outputs. Let err be limit of error. We greedily pick consecutive weights to reset and store error coming from reseting them as a vector. The vector is in $|o|$-dimensional space of ranges of real numbers. Range in i'th dimension is range of error collected for i'th output of network. We sort given list by length of vectors in some metric space (e.g. Manhattan, Euclidean). We divide the list into some numer of segments, let it be 10. In every segment, we shuffle elements. Then we go through consecutive segments, every segment once or twice. In segment, we check consecutive vectors. For a vector $v2$ we check whether $v + v2$ is still in limit of error err. If it is then we reset the weight and assign $v := v + v2$.

3 Propagation of error

For layers max-pool, relu, softmax, norm, functions of activation of input have low derivatives ($|f'(x)| \leq 1$). Therefore range of activation on previous layers could decrease geometrically. Also in tests, range of error does not explode on consecutive layers. For fully-connected and convolutional layers, error is raising very fast (in proportion to number of edges). Weights on edges seems to be quite significant so the estimation behaves badly.

4 Conclusion after tests

Algorithm was tested on Lenet on Intel Deep Learning Framework. It fulfills part of exceptations. It gives guarantee of given accuracy of network but resets not many weights. For given error limit 1% resets 0.8% of weights, mainly from last layer. For 4% of reseted weights, decreases number of recognized numbers from 9909/10000 to 9908/10000. Convolutional and fully-connected layers cause geometric increase of ranges so removed weights come from last layers of network. Probably knowledge about ranges of activation and correlation between neuron and neighbours is not enough to precisely predict ranges of activation of next neurons. The same about impact on output of network. According to such estimations, one pixel could dramatically change output of network. Model that would overcome this would be much better.

5 Possible development

- If we do not want to guarantee specific variation on output, we could add layers that calibrate estimation of error so that its average value would be constant for every layer of network. How good would the selection of weights be? It should be quite easy to check.
- Test behavior of algorithm for other networks.
- New model. Such that consider more properties fo neural networks or having wider knowledge about input and correlation between pixels from input.
- Another algorithm that is picking weights, especially such that would prefer reseting weights from convolutional layers.
- Another representation of estimated error (e.g. linear combination of inputs of network)
- Statistic estimations
- To be continued...