Bitwise Neural Networks

- Minje Kim, Paris Smaragdis

- ICML 2015

Overview

- every input node, output node, and weight, is represented by a single bit, XNOR and bit counting operations are used instead of multiplication
- Binary: weight & activation & input & output

Feedforward in Bitwise Neural Networks

- Forward和backward中都用XNOR操作进行替换设计

$$a_i^l = b_i^l + \sum_j^{K^{l-1}} w_{i,j}^l \otimes z_j^{l-1}, \tag{1}$$

$$z_i^l = \mathrm{sign}(a_i^l), \tag{2}$$

$$\mathbf{z}^l \in \mathbb{B}^{K^l}, \mathbf{W}^l \in \mathbb{B}^{K^l \times K^{l-1}}, \mathbf{b}^l \in \mathbb{B}^{K^l}, \tag{3}$$

We can check the prediction error $\mathcal{E}$ by measuring the bit-wise agreement of target vector $\mathbf{t}$ and the output units of L-th layer using XNOR as a multiplication operator,

$$\mathcal{E} = \sum_i^{K^{L+1}} (1 - t_i \otimes z_i^{L+1})/2, \tag{4}$$

but this error function can be tentatively replaced by involving a softmax layer during the training phase.

- The XNOR operation is a faster substitute of binary multiplication.

Training Bitwise Neural Networks

- Step1. Real-valued Networks with Weight Compression
- We train a real-valued network that takes either bitwise inputs or real-valued inputs ranged between -1 and +1 .

$$a_i^l = \tanh(\bar{b}_i^l) + \sum_j^{K^{l-1}} \tanh(\bar{w}_{i,j}^l)\bar{z}_j^{l-1}, \qquad (5)$$

$$\bar{z}_i^l = \tanh\left(a_i^l\right), \qquad (6)$$

Forward

backward

Weight compression needs some changes in the backpropagation procedure. In a hidden layer we calculate the error,

$$\delta_j^l(n) = \Big(\sum_i^{K^{l+1}} \tanh(\bar{w}_{i,j}^{l+1})\delta_i^{l+1}(n)\Big) \cdot \Big(1 - \tanh^2\left(a_j^l\right)\Big).$$

$$\nabla\bar{w}_{i,j}^l = \Big(\sum_n \delta_i^l(n)\bar{z}_j^{l-1}\Big) \cdot \Big(1 - \tanh^2\left(\bar{w}_{i,j}^l\right)\Big),$$

$$\nabla\bar{b}_i^l = \Big(\sum_n \delta_i^l(n)\Big) \cdot \Big(1 - \tanh^2\left(\bar{b}_i^l\right)\Big),$$

gradient

Training Bitwise Neural Networks

- Step2. Training BNN with Noisy Backpropagation
- 1. Setup a sparsity parameter which says the proportion of the zeros after the binarization.
- 2. Then, we divide the parameters into three groups: +1 , 0 , or -1

in (1) and (2). Then, during noisy backpropagation the errors and gradients are calculated using those binarized weights and signals as well:

$$\delta_j^l(n) = \sum_i^{K^{l+1}} w_{i,j}^{l+1} \delta_i^{l+1}(n),$$
$$\nabla \bar{w}_{i,j}^l = \sum_n \delta_i^l(n) z_j^{l-1}, \quad \nabla \bar{b}_i^l = \sum_n \delta_i^l(n). \qquad (7)$$

- gradients can get too small to update the binary parameters, we instead update their corresponding real-valued parameters.

$$\bar{w}_{i,j}^l \leftarrow \bar{w}_{i,j}^l - \eta \nabla \bar{w}_{i,j}^l, \quad \bar{b}_{i,j}^l \leftarrow \bar{b}_i^l - \eta \nabla \bar{b}_i^l, \qquad (8)$$

- At the end of each update we binarize them again with beta.

Experiments

Networks	Bipolar	0 or 1	Fixed-point (2bits)
Floating-point networks (64bits)	1.17%	1.32%	1.36%
BNN	1.33%	1.36%	1.47%