Loss-Aware Binarization Of Deep Networks

Credit: Lu Hou, Quanming Yao, James T. Kwok

Hong Kong University of Science and Technology

Presenter: Ryan McCampbell
https://qdata.github.io/deep2Read

Outline

Outline

Introduction

- Deep neural networks are expensive in space and time
  - Training
  - Inference
  - Weight storage
- Not good for embedded systems and mobile devices
- How can we make them more efficient?

Ideas

- Train neural network then compress
- Train and compress network simultaneously
  - Tensor decomposition
  - Parameter quantization
  - Binarization

Binarization

- Store only one bit for each weight value
- Significantly reduces storage
- Eliminates most multiplications in forward pass

Outline

Prior Work

- BinaryConnect - Transform each weight to ± 1 with the sign function

$$\text{Binarize}(w_l^t) = \text{sign}(w_l^t)$$

- Binary Weight Network - also learn scaling parameter

$$\text{Binarize}(w_l^t) = \alpha_l^t \mathbf{b}_l^t$$

$$\alpha_l^t = \frac{\|w_l^t\|_1}{n_l}, \ \mathbf{b}_l^t = \text{sign}(w_l^t)$$

Prior Work

- Use binarized weights for forward and backward pass
- Use full-precision weights in update, then re-binarize

Prior Work

- Use binarized weights for forward and backward pass
- Use full-precision weights in update, then re-binarize
- **Problem**: This doesn't consider effect of binarization on loss

Outline

Proximal Newton Algorithm

- Proximal Newton algorithm: composite optimization problems where g may not be smooth
$$\min_x f(x) + g(x)$$
- Use second-order information: approximate Hessian $\mathbf{H}$ of f
$$x_{t+1} = \mathrm{argmin}_x \nabla f(x_t)^T (x - x_t) + (x - x_t)^T \mathbf{H} (x - x_t) + g(x)$$

Loss-Aware Binarization

- Minimize $\ell(\hat{w})$ given constraints

$$\hat{w}_l = \alpha_l \mathbf{b}_l,\ \ \alpha_l > 0,\ \ b_l \in \{\pm 1\}_l^n$$

- Define $I_C(\hat{w})$ as indicator function, 1 if constraints hold and ∞ otherwise.

$$\min \ell(\hat{w}) + I_C(\hat{w})$$

- Use proximal Newton method:

$$\min_{\hat{w}^t} \nabla \ell(\hat{w}^{t-1})^T(\hat{w}^t - \hat{w}^{t-1}) + (\hat{w}^t - \hat{w}^{t-1})^T \mathbf{D}^{t-1}(\hat{w}^t - \hat{w}^{t-1})$$

given the constraints, where $\mathbf{D}$ is an approximate diagonal Hessian matrix.

Solution

- Let $\mathbf{d}_l^{t-1} = diag(\mathbf{D}_l^{t-1})$

$$w_l^t = \hat{w}_l^{t-1} - \nabla_l \ell(\hat{w}^{t-1})/\mathbf{d}_l^{t-1}$$

$$\alpha_l^t = \frac{\|\mathbf{d}_l^{t-1} \circ w_l^t\|_1}{\|\mathbf{d}_l^{t-1}\|_1}, \ \mathbf{b}_l^t = sign(w_l^t)$$

- Each iteration: gradient descent with adaptive learning rate $1/\mathbf{d}_l^{t-1}$
- Project to binary

Algorithm

Input: Minibatch $\{(\mathbf{x}_0^t, \mathbf{y}^t)\}$, current full-precision weights $\{\mathbf{w}_l^t\}$, first moment $\{\mathbf{m}_l^{t-1}\}$, second moment $\{\mathbf{v}_l^{t-1}\}$, and learning rate η^t.

Forward Propagation
for $l = 1$ to L **do**
    $\alpha_l^t = \frac{\|\mathbf{d}_l^{t-1} \odot \mathbf{w}_l^t\|_1}{\|\mathbf{d}_l^{t-1}\|_1}$;
    $\mathbf{b}_l^t = \text{sign}(\mathbf{w}_l^t)$;
    rescale the layer-l input: $\tilde{\mathbf{x}}_{l-1}^t = \alpha_l^t \mathbf{x}_{l-1}^t$;
    compute $\mathbf{z}_l^t$ with input $\tilde{\mathbf{x}}_{l-1}^t$ and binary weight $\mathbf{b}_l^t$;
    apply batch-normalization and nonlinear activation to $\mathbf{z}_l^t$ to obtain $\mathbf{x}_l^t$;
end for
compute the loss ℓ using $\mathbf{x}_L^t$ and $\mathbf{y}^t$;
Backward Propagation
initialize output layer's activation's gradient $\frac{\partial \ell}{\partial \mathbf{x}_L^t}$;
for $l = L$ to 2 **do**
    compute $\frac{\partial \ell}{\partial \mathbf{x}_{l-1}^t}$ using $\frac{\partial \ell}{\partial \mathbf{x}_l^t}$, α_l^t and $\mathbf{b}_l^t$;
end for

Algorithm

Update parameters using Adam

for $l = 1$ to L **do**

  compute gradients $\nabla_l \ell(\hat{\mathbf{w}}^t)$ using $\frac{\partial \ell}{\partial \mathbf{x}_l^t}$ and $\mathbf{x}_{l-1}^t$;

  update first moment $\mathbf{m}_l^t = \beta_1 \mathbf{m}_l^{t-1} + (1 - \beta_1)\nabla_l \ell(\hat{\mathbf{w}}^t)$;

  update second moment $\mathbf{v}_l^t = \beta_2 \mathbf{v}_l^{t-1} + (1 - \beta_2)(\nabla_l \ell(\hat{\mathbf{w}}^t) \odot \nabla_l \ell(\hat{\mathbf{w}}^t))$;

  compute unbiased first moment $\hat{\mathbf{m}}_l^t = \mathbf{m}_l^t / (1 - \beta_1^t)$;

  compute unbiased second moment $\hat{\mathbf{v}}_l^t = \mathbf{v}_l^t / (1 - \beta_2^t)$;

  compute current curvature matrix $\mathbf{d}_l^t = \frac{1}{\eta^t}\left(\epsilon \mathbf{1} + \sqrt{\hat{\mathbf{v}}_l^t}\right)$;

  update full-precision weights $\mathbf{w}_l^{t+1} = \mathbf{w}_l^t - \hat{\mathbf{m}}_l^t \oslash \mathbf{d}_l^t$;

  update learning rate $\eta^{t+1} = \text{UpdateRule}(\eta^t, t+1)$;

end for

Further Optimization

- An additional optimization can be obtained by binarizing the activation function
- This reduces additions and multiplications to XNOR bitwise operations

Outline

Results

Table 1: Test error rates (%) for feedforward neural network models.

		MNIST	*CIFAR-10*	*SVHN*
(no binarization)	full-precision	1.190	11.900	2.277
(binarize weights)	BinaryConnect	1.280	**9.860**	2.450
	BWN	1.310	10.510	2.535
	LAB	**1.180**	10.500	**2.354**
(binarize weights and activations)	BNN	1.470	12.870	3.500
	XNOR	1.530	12.620	3.435
	LAB2	**1.380**	**12.280**	**3.362**

Results

Table 2: Test error rates (%) on *SVHN*, for CNNs with different numbers of filters. Number in brackets is the difference between the errors of the binarized scheme and the full-precision network.

	$K = 16$	$K = 32$	$K = 64$	$K = 128$
full-precision	2.738	2.585	2.277	2.146
BinaryConnect	3.200 (0.462)	2.777 (0.192)	2.450 (0.173)	2.315 (0.169)
BWN	3.119 (0.461)	2.743 (0.158)	2.535 (0.258)	2.319 (0.173)
LAB	**3.050** (0.312)	**2.742** (0.157)	**2.354** (0.077)	**2.200** (0.054)

Results

Table 3: Testing cross-entropy values of LSTM.

		War and Peace	*Linux Kernel*
(no binarization)	full-precision	1.268	1.329
(binarize weights)	BinaryConnect	2.942	3.532
	BWN	1.313	1.307
	LAB	**1.291**	**1.305**
(binarize weights and activations)	BNN	3.050	3.624
	XNOR	1.424	1.426
	LAB2	**1.376**	**1.409**

Outline

- This method has similar performance to the full-precision networks for many tasks
- Enables efficient storage and evaluation of deep neural networks
- However it is only slightly better in some cases than BWN
- Binarizing activations reduces performance