**PROXQUANT: Quantized Neural Networks via Proximal Operators**

Anonymous Author(s)

Affiliation

Address

email

Abstract

To make deep neural networks feasible in resource-constrained environments (such as mobile devices), it is beneficial to quantize models by using low-precision weights. One common technique for quantizing neural networks is the straight-through gradient method, which enables back-propagation through the quantization mapping. Despite its empirical success, little is understood about why the straight-through gradient method works.

Building upon a novel observation that the straight-through gradient method is in fact identical to the well-known Nesterov’s dual-averaging algorithm on a quantization constrained optimization problem, we propose a more principled alternative approach, called PROXQUANT, that formulates quantized network training as a regularized learning problem instead and optimizes it via the prox-gradient method. PROXQUANT does back-propagation on the underlying full-precision vector and applies an efficient prox-operator in between stochastic gradient steps to encourage quantizedness. For quantizing ResNets and LSTMs, PROXQUANT outperforms state-of-the-art results on binary quantization and is on par with state-of-the-art on multi-bit quantization. For binary quantization, our analysis shows both theoretically and experimentally that PROXQUANT is more stable than the straight-through gradient method (i.e. BinaryConnect), challenging the indispensability of the straight-through gradient method and providing a powerful alternative.

1 Introduction

In this paper, we formulate the problem of model quantization as a regularized learning problem and propose to solve it with a proximal gradient method. Our contributions are summarized as follows.

- We present a unified framework for defining regularization functionals that encourage binary, ternary, and multi-bit quantized parameters, through penalizing the distance to quantized sets. For binary quantization, the resulting regularizer is a $W$-shaped non-smooth regularizer, which shrinks parameters towards either $-1$ or $1$ in the same way that the $L_1$ norm regularization shrinks parameters towards 0. We demonstrate that the prox-operators for regularizers that come out of our framework often admit linear-time solutions (or linear time approximation heuristics) which result in numerically exact quantized parameters.

- We propose training quantized networks using PROXQUANT (Algorithm[1]) — a stochastic proximal gradient method with a homotopy scheme. Compared with the straight-through gradient method, PROXQUANT has access to additional gradient information at non-quantized points, and its homotopy scheme prevents potential overshoot early in the training. Algorithmically, PROXQUANT involves just adding a simple proximal step with respect to a quantization-inducing regularizer after each stochastic gradient step, thus can be efficiently

implemented under any major deep learning frameworks without incurring significant system overhead and be used as a modular component to add to the training pipeline of any deep networks to result in a quantized network.

- We demonstrate the effectiveness and flexibility of PROXQUANT through systematic experiments on (1) image classification with ResNets (Section 3.1); (2) language modeling with LSTMs (Section 3.2). The PROXQUANT method outperforms the state-of-the-art results on binary quantization and is comparable with the state-of-the-art on ternary and multi-bit quantization.

- For binary nets, we show that BinaryConnect suffers from more optimization instability than PROXQUANT through (1) a theoretical characterization of convergence for BinaryConnect (Section 4.1) and (2) a sign change experiment on CIFAR-10 (Section B). Experimentally, PROXQUANT finds better binary nets that is also closer to the initialization in the sign change metric.

### 2 Quantized net training via regularized learning

We propose the PROXQUANT algorithm, which adds a quantization-inducing regularizer onto the loss and optimizes via the (non-lazy) prox-gradient method with a finite $\lambda$. The prototypical version of PROXQUANT is described in Algorithm 1.

**Algorithm 1 PROXQUANT: Prox-gradient method for quantized net training**

Require: Regularizer $R$ that induces desired quantizedness, initialization $\theta_0$, learning rates $\{\eta_t\}_{t \geq 0}$, regularization strengths $\{\lambda_t\}_{t \geq 0}$

while not converged do

Perform the prox-gradient step

$$\theta_{t+1} = \arg \min_{\theta \in \mathbb{R}^d} \left\{ L(\theta_t) + \left\langle \theta - \theta_t, \nabla L(\theta_t) \right\rangle + \frac{1}{2\eta_t} \|\theta - \theta_t\|_2^2 + \lambda_t R(\theta) \right\}$$

$$= \text{prox}_{\eta_t, \lambda_t R} \left( \theta_t - \eta_t \nabla L(\theta_t) \right).$$

end while

Compared to usual full-precision training, PROXQUANT only adds a prox step after each stochastic gradient step, hence can be implemented straightforwardly upon existing full-precision training. As the prox step does not need to know how the gradient step is performed, our method adapts to other stochastic optimizers as well such as Adam. Further, each iteration is a prox-gradient step over the objective $L(\theta) + \lambda R(\theta)$ with learning rates $\eta_t$, and by choosing $(\eta_t, \lambda_t)$ we obtain a joint control over the speed of training and falling onto the quantized set.

### 3 Experiments

We evaluate the performance of PROXQUANT on two tasks: image classification with ResNets, and language modeling with LSTMs. On both tasks, we show that the default straight-through gradient method is not the only choice, and our PROXQUANT can achieve the same and often better results.

#### 3.1 Image classification on CIFAR-10

**Problem setup** We perform image classification on the CIFAR-10 dataset, which contains 50000 training images and 10000 test images of size 32x32. We apply a commonly used data augmentation strategy (pad by 4 pixels on each side, randomly crop to 32x32, do a horizontal flip with probability 0.5, and normalize). Our models are ResNets \cite{he2016deep} of depth 20, 32, and 44 with weights quantized to binary or ternary.
**Method** We use PROXQUANT with suitable regularizers in the binary case and the ternary case, which we respectively denote as PQ-B and PQ-T. The training is initialized at pre-trained full-precision nets (warm-start). For the regularization strength we use the homotopy method $\lambda_t = \lambda \cdot t$ with $\lambda = 10^{-4}$. We initialize at pre-trained full-precision networks and use the Adam optimizer with constant learning rate 0.01. To accelerate training in the final stage, we do a hard quantization $\theta \mapsto q(\theta)$ at epoch 400 and keeps training till the 600-th epoch to stabilize the BatchNorm layers.

We compare with BinaryConnect (BC) for binary nets and Trained Ternary Quantization (TTQ) [Zhu et al., 2016] for ternary nets. For BinaryConnect, we haven’t found reported results with ResNets on CIFAR-10, and we train with the recommended Adam optimizer with learning rate decay [Courbariaux et al., 2015] (initial learning rate 0.01, multiply by 0.1 at epoch 81 and 122, hard-quantize at epoch 400), which we find leads to the best result for BinaryConnect.

**Result** The top-1 classification errors are reported in Table 1. For binary nets, our PROXQUANT-Binary consistently yields better results than BinaryConnect. For ternary nets, our results are comparable with the reported results of TTQ, and the best performance of our method over 4 runs (from the same initialization) is slightly better than TTQ.

### Table 1: Top-1 classification error of quantized ResNets on CIFAR-10. Performance is reported in mean(std) over 4 runs, where for PQ-T we report in addition the best of 4 (Bo4).

<table>
<thead>
<tr>
<th>Model</th>
<th>Full-Precision (32)</th>
<th>BC (1)</th>
<th>PQ-B (ours) (1)</th>
<th>TTQ (2)</th>
<th>PQ-T (ours) (2)</th>
<th>PQ-T (Bo4) (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-20</td>
<td>8.06</td>
<td>9.49 (0.22)</td>
<td><strong>9.15</strong> (0.21)</td>
<td>8.87</td>
<td><strong>8.40</strong> (0.13)</td>
<td>8.22</td>
</tr>
<tr>
<td>ResNet-32</td>
<td>7.25</td>
<td>8.66 (0.36)</td>
<td><strong>8.40</strong> (0.23)</td>
<td>7.63</td>
<td>7.65 (0.15)</td>
<td>7.53</td>
</tr>
<tr>
<td>ResNet-44</td>
<td>6.96</td>
<td>8.26 (0.24)</td>
<td><strong>7.79</strong> (0.06)</td>
<td>7.02</td>
<td>7.05 (0.08)</td>
<td>6.98</td>
</tr>
</tbody>
</table>

**3.2 Language modeling with LSTMs**

See Appendix A for details.

**4 Stability analysis of binary quantization**

**4.1 Convergence characterization for BinaryConnect**

We now show that BinaryConnect has a very stringent convergence condition. Consider the BinaryConnect method with batch gradients:

$$s_t = \text{sign}(\theta_t), \quad \theta_{t+1} = \theta_t - \eta_t \nabla L(s_t).$$

**Definition 4.1 (Fixed point and convergence).** We say that $s \in \{\pm 1\}^d$ is a fixed point of the BinaryConnect algorithm, if $s_0 = s$ in eq. (3) implies that $s_t = s$ for all $t = 1, 2, \ldots$. We say that the BinaryConnect algorithm converges if there exists $t < \infty$ such that $s_t$ is a fixed point.

**Theorem 4.1.** Assume that the learning rates satisfy $\sum_{t=0}^{\infty} \eta_t = \infty$, then $s \in \{\pm 1\}^d$ is a fixed point for BinaryConnect eq. (3) if and only if $\text{sign}(\nabla L(s)[i]) = -s[i]$ for all $i \in [d]$ such that $\nabla L(\theta)[i] \neq 0$. Such a point may not exist, in which case BinaryConnect does not converge for any initialization $\theta_0 \in \mathbb{R}^d$.

In a sign change experiment on CIFAR-10 (see Appendix B), we are going to see that BinaryConnect indeed fails to converge to a fixed sign pattern, corroborating Theorem 4.1.

**References**


A Experiments on LSTMs

Problem setup We perform language modeling with LSTMs [Hochreiter and Schmidhuber [1997] on the Penn Treebank (PTB) dataset [Marcus et al. [1993], which contains 929K training tokens, 73K validation tokens, and 82K test tokens. Our model is a standard one-hidden-layer LSTM with embedding dimension 300 and hidden dimension 300. We train quantized LSTMs with the encoder, transition matrix, and the decoder quantized to $k$-bits for $k \in \{1, 2, 3\}$. The quantization is performed in a row-wise fashion, so that each row of the matrix has its own codebook $\{a_1, \ldots, a_k\}$.

Method We compare our multi-bit PROXQUANT to the state-of-the-art alternating minimization algorithm with straight-through gradients [Xu et al., 2018]. Training is initialized at a pre-trained full-precision LSTM. We use the SGD optimizer with initial learning rate 20.0 and decay by a factor of 1.2 when the validation error does not improve over an epoch. We train for 80 epochs with batch size 20, BPTT 30, dropout with probability 0.5, and clip the gradient norms to 0.25. The regularization rate $\lambda$ is tuned by finding the best performance on the validation set. In addition to multi-bit quantization, we also report the results for binary LSTMs (weights in $\{-1, +1\}$), comparing BinaryConnect and our PROXQUANT-Binary.

Result We report the perplexity-per-word (PPW, lower is better) in Table 2. The performance of PROXQUANT is comparable with the Straight-through gradient method. On Binary LSTMs, PROXQUANT-Binary beats BinaryConnect by a large margin. These results demonstrate that PROXQUANT offers a powerful alternative for training recurrent networks.

<table>
<thead>
<tr>
<th>Method / Number of Bits</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>FP (32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BinaryConnect</td>
<td>419.1</td>
<td>-</td>
<td>-</td>
<td>88.5</td>
</tr>
<tr>
<td>PROXQUANT-Binary (ours)</td>
<td>321.8</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ALT Straight-through$^1$</td>
<td>104.7</td>
<td>90.2</td>
<td>86.1</td>
<td></td>
</tr>
<tr>
<td>ALT-PROXQUANT (ours)</td>
<td>106.2</td>
<td>90.0</td>
<td>87.2</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: PPW of quantized LSTM on Penn Treebank.

B Sign change experiment

We experimentally compare the training dynamics of PROXQUANT-Binary and BinaryConnect through the sign change metric. The sign change metric between any $\theta_1$ and $\theta_2$ is the proportion of their different signs, i.e. the (rescaled) Hamming distance:

$$\text{SignChange}(\theta_1, \theta_2) = \frac{\|\text{sign}(\theta_1) - \text{sign}(\theta_2)\|_1}{2d} \in [0, 1].$$

In $\mathbb{R}^d$, the space of all full-precision parameters, the sign change is a natural distance metric that represents the closeness of the binarization of two parameters.

Recall in our CIFAR-10 experiments (Section 3.1), for both BinaryConnect and PROXQUANT, we initialize at a good full-precision net $\theta_0$ and stop at a converged binary network $\hat{\theta} \in \{\pm 1\}^d$. We are interested in $\text{SignChange}(\theta_0, \hat{\theta})$ along the training path, as well as $\text{SignChange}(\theta_1, \hat{\theta})$, i.e. the distance of the final output model to the initialization.

As PROXQUANT converges to higher-performance solutions than BinaryConnect, we expect that if we run both methods from a same warm start, the sign change of PROXQUANT should be higher than that of BinaryConnect, as in general one needs to travel farther to find a better net.

However, we find that this is not the case: PROXQUANT produces binary nets with both lower sign changes and higher performances, compared with BinaryConnect. This finding is consistent in

---

$^1$We thank Xu et al. [2018] for sharing the implementation of this method through a personal communication. There is a very clever trick not mentioned in their paper: after computing the alternating quantization $q_{\text{alt}}(\theta)$, they multiply by a constant 0.3 before taking the gradient; in other words, their quantizer is a rescaled alternating quantizer: $\theta \mapsto 0.3q_{\text{alt}}(\theta)$. This scaling step gives a significant gain in performance – without scaling the PPW is $\{116.7, 94.3, 87.3\}$ for $\{1, 2, 3\}$ bits. In contrast, our PROXQUANT does not involve a scaling step and achieves better PPW than this unscaled ALT straight-through method.
Figure 1: SignChange($\theta_0, \theta_t$) against $t$ (epoch) for BinaryConnect and PROXQUANT, over 4 runs starting from the same full-precision ResNet-20. PROXQUANT has significantly lower sign changes than BinaryConnect while converging to better models. (a) The first conv layer of size $16 \times 3 \times 3 \times 3$; (b) The last conv layer of size $64 \times 64 \times 3 \times 3$; (c) The fully connected layer of size $64 \times 10$; (d) The validation top-1 error of the binarized nets (with moving average smoothing).

all layers, across different warm starts, and across different runs from each same warm start (see Figure 1 and Table 3 in Appendix B.1). This shows that for every warm start position, there is a good binary net nearby which can be found by PROXQUANT but not BinaryConnect, suggesting that BinaryConnect, and in general the straight-through gradient method, suffers from higher optimization instability than PROXQUANT. This result here is also consistent with Theorem 4.1: the signs in BinaryConnect never stop changing until we manually freeze the signs at epoch 400.

B.1 Detailed sign change results on ResNet-20

Table 3: Performances and sign changes on ResNet-20 in mean(std) over 3 full-precision initializations and 4 runs per (initialization x method). Sign changes are computed over all quantized parameters in the net.

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Method</th>
<th>Top-1 Error(%)</th>
<th>Sign change</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP-Net 1 (8.06)</td>
<td>BC</td>
<td>9.489 (0.223)</td>
<td>0.383 (0.006)</td>
</tr>
<tr>
<td></td>
<td>PQ-B</td>
<td><strong>9.146 (0.212)</strong></td>
<td><strong>0.276 (0.020)</strong></td>
</tr>
<tr>
<td>FP-Net 2 (8.31)</td>
<td>BC</td>
<td>9.745 (0.422)</td>
<td>0.381 (0.004)</td>
</tr>
<tr>
<td></td>
<td>PQ-B</td>
<td><strong>9.444 (0.067)</strong></td>
<td><strong>0.288 (0.002)</strong></td>
</tr>
<tr>
<td>FP-Net 3 (7.73)</td>
<td>BC</td>
<td>9.383 (0.211)</td>
<td>0.359 (0.001)</td>
</tr>
<tr>
<td></td>
<td>PQ-B</td>
<td><strong>9.084 (0.241)</strong></td>
<td><strong>0.275 (0.001)</strong></td>
</tr>
</tbody>
</table>

Table 4: Performances and sign changes on ResNet-20 in raw data over 3 full-precision initializations and 4 runs per (initialization x method). Sign changes are computed over all quantized parameters in the net.

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Method</th>
<th>Top-1 Error(%)</th>
<th>Sign change</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP-Net 1 (8.06)</td>
<td>BC</td>
<td>9.664, 9.430, 9.198, 9.663</td>
<td>0.386, 0.377, 0.390, 0.381</td>
</tr>
<tr>
<td></td>
<td>PQ-B</td>
<td>9.058, 8.901, 9.388, 9.237</td>
<td>0.288, 0.247, 0.284, 0.285</td>
</tr>
<tr>
<td>FP-Net 2 (8.31)</td>
<td>BC</td>
<td>9.456, 9.530, 9.623, 10.370</td>
<td>0.376, 0.379, 0.382, 0.386</td>
</tr>
<tr>
<td></td>
<td>PQ-B</td>
<td>9.522, 9.474, 9.410, 9.370</td>
<td>0.291, 0.287, 0.289, 0.287</td>
</tr>
<tr>
<td>FP-Net 3 (7.73)</td>
<td>BC</td>
<td>9.107, 9.558, 9.338, 9.328</td>
<td>0.360, 0.357, 0.359, 0.360</td>
</tr>
<tr>
<td></td>
<td>PQ-B</td>
<td>9.284, 8.866, 9.301, 8.884</td>
<td>0.275, 0.276, 0.276, 0.275</td>
</tr>
</tbody>
</table>