Learning Stochastic Binary Neural Networks

Viktor Yanush [1]
Alexander Shekhovtsov [2] Dmitry Molchanov [3] Dmitry Vetrov [1,3]

[1] Samsung HSE Lab

[2]Czech Technical University in Prague

[3]Samsung Research

15.05.2020

Why Binary?

Many papers showed that we do not need much precision:

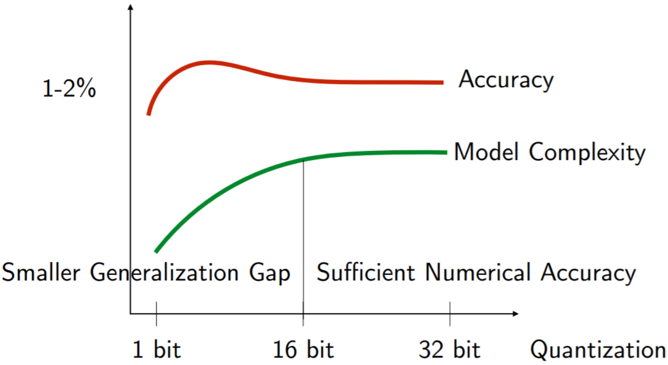
Table 4: ResNet-34 top-1 validation accuracy % and compute cost as precision of activations (A) and weights (W) varies.

Width	Precision	Top-1 Acc. %	Compute cost
1x wide	32b A, 32b W	73.59	1x
	1b A, 1b W	60.54	0.03x
2x wide	4b A, 8b W	74.48	0.74x
	4b A, 4b W	74.52	0.50x
	4b A, 2b W	73.58	0.39x
	2b A, 4b W	73.50	0.39x
	2b A, 2b W	73.32	0.27x
	1b A, 1b W	69.85	0.15x
3x wide	1b A, 1b W	72.38	0.30x

Figure: WRPN: wide reduced-precision networks [Mishra et al., 2017]

Why Binary?

Quantization can be beneficial for generalization:



Why Binary?

- We can easily perform inference on CPU[1]
- Main speed-up comes from convolutions/fully-connected layers
- Batch Norm/Layer Norm etc. can be performed in floats[2]
- First layer usually is small and is not binarized

[1] and sometimes training [Rastegari et al., 2016]
[2] also in 8-bit integers [Simons and Lee, 2019]

Why Stochastic?

We use stochastic BNNs for the following reasons:

- Improved generalization
- Possibility of ensembling
- More well-defined theoretically

Problem setup

Functional view:

$$f(x^0, Z; \theta) = x^n$$

$$x^k = \mathrm{sgn}(a^k(x^{k-1}; \theta^k) - Z^k)$$

Probabilistic view:

$$p(x \mid x^0; \theta) = \prod_{k=1}^{L} p(x^k \mid x^{k-1}; \theta)$$

$$p(x_j^k \mid x^{k-1}; \theta) = F_Z(x_j^k a_j^k(x^{k-1}; \theta))$$

Training objective:

$$\mathcal{L}(\theta) = \mathbb{E}_{x^0 \sim data} \mathbb{E}_Z \left[L(x^n) \right] \to \min_\theta$$

We want to estimate $\nabla \mathcal{L}(\theta)$.

Existing solutions

Existing solutions

Reparameterization trick [Kingma et al., 2015]

Can we use reparameterization trick?

$$\mathcal{L}(\theta) = \mathbb{E}_{x^0 \sim data} \mathbb{E}_Z \left[L(x^n) \right] = \mathbb{E}_{x^0 \sim data} \mathbb{E}_{x^{1 \ldots n}} \left[L(x^n) \right] \to \min_{\theta}$$

No: $\nabla_\theta \mathbb{E}_Z \left[L(x^n) \right] \neq \mathbb{E}_Z \left[\nabla_\theta L(x^n) \right]$.
Example:

$$\frac{\partial \operatorname{sgn}(a - Z)}{\partial a} = 0$$

$$\mathbb{E}_Z \left[\operatorname{sgn}(a - Z) \right] = \mathbb{P}(Z < a) - \mathbb{P}(Z \geq a) = 2F_Z(a) - 1$$

$$\frac{\partial}{\partial a} \mathbb{E}_Z \left[\operatorname{sgn}(a - Z) \right] = 2p_Z(a)$$

Existing solutions

REINFORCE [Williams, 1992]

$$\mathcal{L}(\theta) = \mathbb{E}_{x^0 \sim data}\mathbb{E}_Z \left[L(x^n)\right] = \mathbb{E}_{x^0 \sim data}\mathbb{E}_{x^{1\dots n}} \left[L(x^n)\right] \to \min_\theta$$

$$\nabla\mathcal{L}(\theta) = \mathbb{E}_{x^0 \sim data}\mathbb{E}_{x^{1\dots n}} \left[L(x^n)\nabla \log p(x \mid x^0; \theta)\right]$$

$$= \mathbb{E}_{x^0 \sim data}\mathbb{E}_{x^{1\dots n}} \left[L(x^n) \sum_{k=1}^{n} \nabla \log p(x^k \mid x^{k-1}; \theta)\right]$$

- Unbiased estimator
- Does not use $\nabla L(x)$, only $L(x)$
- Usually has large variance

Existing solutions

REINFORCE extensions:

- **Augment-REINFORCE-Merge** [Yin and Zhou, 2018] — has quadratic complexity for multiple layers.
- **REBAR** [Tucker et al., 2017] — by default has quadratic complexity for multiple layers. Can be adapted but it introduces bias.
- **RELAX** [Grathwohl et al., 2017] — authors do not write about time complexity but seems the same as REBAR.

Existing solutions

Concrete relaxation [Maddison et al., 2016, Peters and Welling, 2018]

Approach: replace sgn function with smooth approximation $\mathrm{sgn}(a) \approx \sigma(\frac{a}{\tau})$

- Need to tune additional hyperparameter
- Biased estimator
- May introduce significant deviation between training and test-time
- Training can be slow with high temperature

Existing solutions

Straight-Through Estimator [Bengio et al., 2013, Hubara et al., 2016]

Approach: set $\frac{\partial \operatorname{sgn}(a)}{\partial a} = 1$ or $\frac{\partial \operatorname{sgn}(a)}{\partial a} = 1_{[-1,1]}$

- Used in most papers about training deterministic BNNs
- Based on empirical evidence
- Shows good results in practice

Proposed solution

Gradient estimation

Expectation gradient:

$$\frac{\partial}{\partial\theta}\sum_x p(x;\theta)f(x^n,\theta) = \sum_x \frac{\partial}{\partial\theta}p(x;\theta)f(x^n,\theta) + \underbrace{\sum_x p(x;\theta)\frac{\partial}{\partial\theta}f(x^n,\theta)}_{\text{easy part}}$$

Hard part:

$$\begin{aligned}\sum_x \frac{\partial}{\partial\theta}p(x;\theta)f(x^n) &= \sum_{k=1}^{n}\sum_x \frac{p(x)}{p(x^k \mid x^{k-1};\theta)}f(x^n)\frac{\partial}{\partial\theta}p(x^k \mid x^{k-1};\theta) \\ &= \sum_{k=1}^{n}\sum_i\sum_x \frac{p(x)}{p(x_i^k \mid x^{k-1};\theta)}f(x^n)\frac{\partial}{\partial\theta}p(x_i^k \mid x^{k-1};\theta)\end{aligned}$$

Key idea #1

Key idea #1: Partial analytic summation — reduces variance

$$
\begin{aligned}
\sum_x \frac{p(x;\theta)}{p(x_i;\theta)} h(x) &= \sum_{x_{\neq i}} p(x_{\neq i};\theta) \sum_{x_i} h(x) \\
&= \sum_{x_{\neq i}} p(x_{\neq i};\theta) \left(h(x) + h(x_{\downarrow i})\right) \\
&= \left\{ \text{multiply by } 1 = \sum_{x_i} p(x_i;\theta) \right\} \\
&= \sum_x p(x;\theta) \left(h(x) + h(x_{\downarrow i})\right)
\end{aligned}
$$

Key idea #1

Let $q_i^k(x) = \frac{\partial}{\partial\theta} p(x_i^k \mid x^{k-1}; \theta)$. Applying partial summation over x_i^k:

$$\sum_{i,x} \frac{p(x)}{p(x_i^k \mid x^{k-1})} q_i^k(x) f(x^n) = (*)$$

Terms including x_i^k:

$$\begin{aligned} & p(x^{k+1} \mid x^k) q_i^k(x^k) + p(x^{k+1} \mid x_{\downarrow i}^k) q_i^k(x_{\downarrow i}^k) \\ & = (p(x^{k+1} \mid x^k) - p(x^{k+1} \mid x_{\downarrow i}^k)) q_i^k(x^k) \end{aligned}$$

Substituting in $(*)$:

$$(*) = \sum_{i,x} p(x^{1\dots k}) p(x^{k+2\dots n} \mid x^{k+1}) (p(x^{k+1} \mid x^k) - p(x^{k+1} \mid x_{\downarrow i}^k)) q_i^k(x^k) f(x^n)$$

Key idea #2

Key idea #2: Linearization of $p(x^{k+1} \mid x^k)$:

$$p(y \mid x) = \prod_i p(y_i \mid x) = \prod_i (p(y_i \mid \bar{x}) + \underbrace{p(y_i \mid x) - p(y_i \mid \bar{x})}_{\Delta_i})$$

$$\approx p(y \mid \bar{x}) + \sum_i \prod_{i' \neq i} p(y_{i'} \mid \bar{x}) \Delta_i$$

$$= p(y \mid \bar{x}) + \sum_i \frac{p(y \mid \bar{x})}{p(y_{i'} \mid \bar{x})} \Delta_i$$

Key idea #2

Linearizing $p(x^{k+1} \mid x^k_{\downarrow i})$ w.r.t. $\Delta^{k+1}_{i,j} = p(x^{k+1}_j \mid x^k_{\downarrow i}) - p(x^{k+1}_j \mid x^k)$:

$$p(x^{k+1} \mid x^k_{\downarrow i}) \approx p(x^{k+1} \mid x^k) + \sum_j \prod_{j' \neq j} p(x^{k+1}_{j'} \mid x^k) \Delta^{k+1}_{i,j}$$

This approximation is valid when $\Delta^{k+1}_{i,j}$ are small e.g. when model is almost deterministic or there are many units in layer k.

$$p(x^{k+1} \mid x^k_{\downarrow i}) - p(x^{k+1} \mid x^k) = \sum_j \frac{p(x^{k+1} \mid x^k)}{p(x^{k+1}_j \mid x^k)} \Delta^k_{i,j}$$

Key idea #1 + Key idea #2

Substituting $p(x^{k+1} \mid x^k_{\downarrow i}) - p(x^{k+1} \mid x^k)$ in $(*)$:

$$\sum_{i,x} \frac{p(x;\theta)}{p(x_i^k \mid x^{k-1};\theta)} q_i^k(x) f(x^n) \approx \sum_{j,x} \frac{p(x;\theta)}{p(x_j^{k+1} \mid x^k;\theta)} q_j^{k+1}(x) f(x^n),$$

where $q_j^{k+1}(x) = \sum_i q_i^k(x) \Delta_{i,j}^{k+1}$ or $q^{k+1} = q^k \Delta^{k+1}$

Key idea #1 + Key idea #2

Lemma: Let $q_i^k(x)$ depend only on $x^{1\dots k}$ and $q_i^k(x_i^k) = -q_i^k(x_{\downarrow i}^k)$ for all i. Then

$$\sum_{i,x} \frac{p(x;\theta)}{p(x_i^k \mid x^{k-1};\theta)} q_i^k(x) f(x^n) \approx \sum_{j,x} \frac{p(x;\theta)}{p(x_j^{k+1} \mid x^k;\theta)} q_j^{k+1}(x) f(x^n)$$

where $q_j^{k+1}(x) = \sum_i q_i^k(x) \Delta_{i,j}^{k+1}$

$$\Delta_{i,j}^{k+1} = p(x_j^{k+1} \mid x_{\downarrow i}^k) - p(x_j^{k+1} \mid x^k)$$

Trick for last layer

Applying lemma repeatedly:

$$\sum_{i,x} \frac{p(x;\theta)}{p(x_i^k \mid x^{k-1};\theta)} q_i^k(x) f(x^n) \approx \sum_{j,x} \frac{p(x;\theta)}{p(x_j^n \mid x^{n-1};\theta)} q_j^n(x) f(x^n)$$

Last layer:

$$\sum_{x,j} \frac{p(x;\theta)}{p(x_j^n \mid x^{n-1};\theta)} q_j^n(x) f(x^n) = \sum_{x} p(x;\theta) \sum_{j} q_j^n(x) \left(f(x) - f(x_{\downarrow j})\right)$$

Combining everything together

Denote

$$
\begin{aligned}
d_i^k &= \frac{\partial}{\partial\theta} p(x_i^k \mid x^{k-1}; \theta);\\
df_i &= f(x^n) - f(x^n_{\downarrow i})\\
\Delta^k_{i,j} &= p(x_j^k \mid x^{k-1}_{\downarrow i}; \theta) - p(x_j^k \mid x^{k-1}; \theta)
\end{aligned}
$$

By propagating dependencies from layer k to the last layer:

$$
\sum_x \frac{p(x)}{p(x^k \mid x^{k-1}; \theta)} f(x^n) \frac{\partial}{\partial\theta} p(x^k \mid x^{k-1}; \theta) \approx \sum_x p(x;\theta) d^k \Delta^{k+1} \dots \Delta^n df
$$

Total gradient is found by summing over layers:

$$
\sum_x p(x;\theta) \sum_{k=1}^{n} d^k \Delta^{k+1} \dots \Delta^n df = (d^{n-1} + (\dots (d^2 + d^1 \Delta^2)\Delta^3) \dots)\Delta^n) df
$$

Combining everything together

Finally, gradient can be estimated via

$$
\begin{aligned}
q^1 &= d^1; \\
q^k &= d^k + q^{k-1}\Delta^k \\
\nabla\mathcal{L}(\theta) \approx q^n df &= \underbrace{(d^{n-1} + (\dots(d^2 + d^1\Delta^2)\Delta^3)\dots)\Delta^n)df}_{\text{backprop}}
\end{aligned}
$$

- Not necessary to compute q on forward pass
- We can compute Δ on backward pass

Method discussion

- Computes gradient estimate via one pass through the network
- Estimator is biased but in practice bias is smaller than for existing methods
- Partially computes expectations analytically and reduces variance, but requires n discrete derivative
- Efficient computation of discrete jacobians is different for each layer (e.g. fully connected, convolutional)

Straight-Through Estimator

Proposition: if $a^k(x^{k-1};\theta)$ is multilinear in x, f is differentiable, p_Z is symmetic then we obtain straight-through estimator (STE).

Linearizing $p(x_j^k \mid x^{k-1};\theta)$ and $f(x)$ by x we get:

$$df_i = 2x_i^n \frac{\partial}{\partial x_i^n} f(x^n)$$

$$\Delta_{i,j}^k = 2x_j^k x_i^{k-1} \frac{\partial}{\partial x_i^{k-1}} F_Z(a_j^k(x^{k-1};\theta));$$

$$d_j^k = x_j^k \frac{\partial}{\partial \theta} F_Z(a_j^k(x^{k-1};\theta)).$$

All the x_j^k cancel each other out and we get STE!

Straight-Through Estimator

- Much simpler
- Same computation for any operation
- Works well in practice
- Now theoretically derived

Binary weights

Stochastic binary weights can be handled in the same way:

$$w^k = \mathrm{sgn}(\theta^k - Z_w^k)$$
$$x^k = \mathrm{sgn}(a^k(x^{k-1}, w^k) - Z_a^k)$$

Equivalently

$$w^k \sim \mathrm{Ber}(F_Z(\theta^k)), w^k \in \{-1, 1\}$$
$$x^k \sim \mathrm{Ber}(F_Z(a^k(x^{k-1}, w^k)))$$

We can learn $p^k = F_Z(\theta^k)$ directly with Mirror descent:

$$w^k \sim \mathrm{Ber}(p^k)$$
$$\mathcal{L}(\theta) \to \min_{\theta, p^k \in [0,1]}$$

Mirror descent

Problem:

$$f(x) \to \min_{x \in C}$$

Projected gradient descent:

$$g^k = \nabla f(x^k)$$

$$x^{k+1} = \arg\min_{x \in C} \langle x, g^k \rangle + \frac{1}{2\varepsilon} \|x - x^k\|^2$$

Mirror descent

Problem:

$$f(x) \to \min_{x \in C}$$

Mirror descent:

$$g^k = \nabla f(x^k)$$
$$x^{k+1} = \arg\min_{x \in C} \langle x, g^k \rangle + \frac{1}{\varepsilon} \mathcal{D}_\psi(x, x^k)$$

Mirror descent

Problem:

$$f(x) \to \min_{x \in C}$$

Mirror descent:

$$\begin{aligned} g^k &= \nabla f(x^k) \\ x^{k+1} &= \arg\min_{x \in C} \langle x, g^k \rangle + \frac{1}{\varepsilon} \mathcal{D}_\psi(x, x^k) \end{aligned}$$

Let $\psi : C \to \mathbb{R}$ be strongly convex. Bregman divergence:

$$\mathcal{D}_\psi(p, q) = \psi(p) - \psi(q) - \langle \nabla\psi(q), p - q \rangle$$

MD iteration:

$$\begin{aligned} \nabla\psi(x^{k+1}) &= \nabla\psi(x^k) - \varepsilon g^k \\ x^{k+1} &= (\nabla\psi)^{-1}(\nabla\psi(x^k) - \varepsilon g^k) \end{aligned}$$

Mirror descent

Examples:

- $\mathcal{D}_\psi(p, q) = \frac{1}{2}\|p - q\|^2$, $C = \mathbb{R}^n$

$$x^{k+1} = x^k - \varepsilon g^k$$

- $\mathcal{D}_\psi(p, q) = \mathrm{KL}(p\|q)$, $C = \Delta_n = \{x \in \mathbb{R}^n_+ \mid \sum x_i = 1\}$

$$x_i^{k+1} = \frac{x_i^k \exp(-\varepsilon g_i^k)}{\sum_j x_j^k \exp(-\varepsilon g_j^k)}$$

Binary weights via mirror descent

Optimization problem:

$$\mathcal{L}(\theta) \to \min_{\theta, p^k \in [0,1]}$$

Using mirror descent with KL:

$$p = \arg\min_p \left[\langle p, \frac{\partial \mathcal{L}}{\partial p^k} \rangle + \frac{1}{\varepsilon} \mathrm{KL}(\mathrm{Ber}(p) \| \mathrm{Ber}(p^k)) \right]$$

$$p = \sigma \left(\sigma^{-1}(p^k) - \varepsilon \frac{\partial \mathcal{L}}{\partial p^k} \right)$$

$$\eta = \eta^k - \varepsilon \frac{\partial \mathcal{L}}{\partial p^k}$$

Note: we can use any optimizer here e.g. SGD+Momentum, Adam etc.

Discussion

Many methods using STE for training deterministic BNNs are a special case of proposed scheme:

- XNOR-Net [Rastegari et al., 2016]
- BinaryConnect[Courbariaux et al., 2015, Hubara et al., 2016]
- Bi-Real Net[Liu et al., 2018]

Maximum likelihood training of stochastic BNN also leads to deterministic network:

$$\text{sgn}(\theta - Z) = \text{sgn}\left(\frac{\theta}{\|\theta\|} - \frac{Z}{\|\theta\|}\right) \to \text{sgn}(\theta), \text{ if } \|\theta\| \gg 1$$

In case we really need stochastic BNN we can train Bayesian BNN

Bayesian BNNs

Bayesian BNN

Prior distribution:

$$p(w) = \prod_k p(w^k)$$

$$p(w^k = +1) = p(w^k = -1) = \frac{1}{2}$$

Variational inference:

$$q_\psi(w) = \mathrm{Ber}(w \mid \psi) \approx p(w \mid D) \propto p(D \mid w)p(w)$$

$$ELBO_\beta(\psi) = \mathbb{E}_{q_\psi(w)} \log p(D \mid w) - \beta \underbrace{\mathrm{KL}(q_\psi(w) \| p(w))}_{-\mathrm{H}(q_\psi(w))} \to \max_\psi$$

Variational inference in Bayesian BNN

Optimization problem:

$$\mathbb{E}_{q_\psi(w)} \frac{1}{N} \sum_{i=1}^{N} l_i(x_i, w) + \frac{\beta}{N} \mathrm{H}(q_\psi(w)) \to \max_{\psi}$$

Composite mirror descent iteration:

$$\psi = \arg\min_{\psi} \left[\langle \psi, g^k \rangle + \frac{1}{\varepsilon} \mathrm{KL}(\mathrm{Ber}(\psi) \| \mathrm{Ber}(\psi^k)) - \lambda \mathrm{H}(\mathrm{Ber}(\psi)) \right]$$

$$\eta = \frac{1}{\varepsilon\lambda + 1} \eta^k - \frac{\varepsilon}{\varepsilon\lambda + 1} g^k \approx \eta^k - \varepsilon(g^k + \lambda\eta^k)$$

$\lambda\eta^k$ — *logit decay* (as in weight decay)

Bayesian BNN II

We also tried to place prior on both weights and probabilities:

$$p(w, \theta) = p(w \mid \theta)p(\theta) = \mathrm{Ber}(w \mid \theta)\,\mathrm{Beta}(\theta \mid a, a)$$

Why?

- Marginal prior is the same
  $p(w) = \int \mathrm{Ber}(w \mid \theta)\,\mathrm{Beta}(\theta \mid a, a)d\theta = \mathrm{Ber}(w \mid \frac{1}{2})$
- More expressive variational distribution
- Better control over entropy of the model

Bayesian BNN II

Joint KL divergence:

$$
\begin{aligned}
&\mathrm{KL}(q(w,\theta)\|p(w,\theta \mid D)) \\
&\qquad = -\mathbb{E}_{q(\theta)}\left[\mathbb{E}_{q(w|\theta)} \log p(D \mid w) - \mathrm{KL}(q(w \mid \theta)\|p(w \mid \theta))\right] \\
&\qquad\quad + \mathrm{KL}(q(\theta)\|p(\theta)) + \log p(D)
\end{aligned}
$$

Variational distribution:

$$
q(w,\theta) = q(w \mid \theta)q(\theta) = p(w \mid \theta)\,\text{logit-Normal}(\theta \mid \mu, \sigma)
$$

Experiments

Networks:

- VGG-16
- ResNet-18

Datasets:

- CIFAR-10

Experiments:

- Training via maximum likelihood and variational inference
- Inference via different testing modes for weights and activations

Results

Method: Training mode	Method: Testing mode	Single model accuracy	Ensemble accuracy
Binary weights and activations			
Hubara et al. [Hubara et al., 2016]		89.85%	-
XNOR-Net [Rastegari et al., 2016]		89.83%	-
Maximum likelihood	tanh-S-S	87.78%	89.67%
	tanh-S-M	88.09%	**90.07%**
	tanh-M-S	88.63%	89.73%
	tanh-M-M	89.31%	-
Weight prior	tanh-S-S	88.84%	**90.69%**
	tanh-S-M	89.39%	**90.80%**
	tanh-M-S	89.29%	**91.00%**
	tanh-M-M	89.61%	-
Joint prior	tanh-S-S	85.63%	88.92%
	tanh-S-M	86.04%	88.93%
	tanh-M-S	86.14%	89.54%
	tanh-M-M	**90.07%**	-

Results

Method: Training mode	Method: Testing mode	Single model accuracy	Ensemble accuracy
Binary weights and real activations			
BinaryConnect [Courbariaux et al., 2015]		90.10%	-
Maximum likelihood	tanh-S-E	89.59%	**90.29%**
	tanh-M-E	**90.24%**	-
Weight prior	tanh-S-E	**90.81%**	**91.22%**
	tanh-M-E	**91.23%**	-
Joint prior	tanh-S-E	87.31%	89.93%
	tanh-M-E	89.98%	-

Results

Method		Single model accuracy
Training mode	Testing mode	
Real weights and activations		
VGG-16		92.64%
Maximum likelihood	tanh-E-E	90.75%
Weight prior	tanh-E-E	92.33%
Joint prior	tanh-E-E	90.63%

Conclusion

- We can use STE for binary weights and activations
- If we use STE, forward pass should be connected to backward pass
- Maximum Likelihood $\Rightarrow$ deterministic BNN
- Bayesian BNNs are better for ensembling
- Stochastic BNNs have many inference modes with different computation/accuracy tradeoff

ResNet-18 binary

Method: Training mode	Method: Testing mode	Single model accuracy	Ensemble accuracy
Binary weights and activations			
Bethge et al. [Bethge et al., 2018]		87.6%	-
Maximum likelihood	tanh-S-S	85.92%	89.28%
	tanh-S-M	87.34%	89.52%
	tanh-M-S	88.03%	89.31%
	tanh-M-M	88.36%	-
Weight prior	tanh-S-S	87.23%	90.95%
	tanh-S-M	87.87%	91.04%
	tanh-M-S	88.67%	90.45%
	tanh-M-M	89.32%	-
Joint prior	tanh-S-S	79.44%	81.64%
	tanh-S-M	79.39%	81.90%
	tanh-M-S	79.88%	81.88%
	tanh-M-M	81.73%	-

ResNet-18 real

Method: Training mode	Method: Testing mode	Single model accuracy
Real weights and activations		
ResNet-18		93.02%
Maximum likelihood	tanh-E-E	90.12%
Weight prior	tanh-E-E	91.5%
Joint prior	tanh-E-E	82.02%

Bengio, Y., Léonard, N., and Courville, A. (2013).
Estimating or propagating gradients through stochastic neurons for conditional computation.
arXiv preprint arXiv:1308.3432.

Bethge, J., Bornstein, M., Loy, A., Yang, H., and Meinel, C. (2018).
Training competitive binary neural networks from scratch.
arXiv preprint arXiv:1812.01965.

Courbariaux, M., Bengio, Y., and David, J.-P. (2015).
Binaryconnect: Training deep neural networks with binary weights during propagations.
In *Advances in neural information processing systems*, pages 3123–3131.

II

Grathwohl, W., Choi, D., Wu, Y., Roeder, G., and Duvenaud, D. (2017).
Backpropagation through the void: Optimizing control variates for black-box gradient estimation.
arXiv preprint arXiv:1711.00123.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016).
Binarized neural networks.
In *Advances in neural information processing systems*, pages 4107–4115.

Kingma, D. P., Salimans, T., and Welling, M. (2015).
Variational dropout and the local reparameterization trick.
In *Advances in neural information processing systems*, pages 2575–2583.

III

Liu, Z., Wu, B., Luo, W., Yang, X., Liu, W., and Cheng, K.-T. (2018).
Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm.
In *Proceedings of the European conference on computer vision (ECCV)*, pages 722–737.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2016).
The concrete distribution: A continuous relaxation of discrete random variables.
arXiv preprint arXiv:1611.00712.

Mishra, A., Nurvitadhi, E., Cook, J. J., and Marr, D. (2017).
Wrpn: wide reduced-precision networks.
arXiv preprint arXiv:1709.01134.

IV

Peters, J. W. and Welling, M. (2018).
Probabilistic binary neural networks.
arXiv preprint arXiv:1809.03368.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016).
Xnor-net: Imagenet classification using binary convolutional neural networks.
In *European conference on computer vision*, pages 525–542. Springer.

Simons, T. and Lee, D.-J. (2019).
A review of binarized neural networks.
Electronics, 8(6):661.

V

Tucker, G., Mnih, A., Maddison, C. J., Lawson, J., and Sohl-Dickstein, J. (2017).
Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models.
In *Advances in Neural Information Processing Systems*, pages 2627–2636.

Williams, R. J. (1992).
Simple statistical gradient-following algorithms for connectionist reinforcement learning.
Machine learning, 8(3-4):229–256.

Yin, M. and Zhou, M. (2018).
Arm: Augment-reinforce-merge gradient for stochastic binary networks.