Reintroducing Straight-Through Estimators as Principled Methods for Stochastic Binary Networks

Alexander Shekhovtsov[1](✉) and Viktor Yanush[2]

[1] Czech Technical University in Prague, Prague, Czech Republic
shekhole@fel.cvut.cz

[2] Samsung-HSE Laboratory National Research University Higher School of Economics, Moscow, Russia

Abstract. Training neural networks with binary weights and activations is a challenging problem due to the lack of gradients and difficulty of optimization over discrete weights. Many successful experimental results have been achieved with empirical straight-through (ST) approaches, proposing a variety of ad-hoc rules for propagating gradients through non-differentiable activations and updating discrete weights. At the same time, ST methods can be truly derived as estimators in the stochastic binary network (SBN) model with Bernoulli weights. We advance these derivations to a more complete and systematic study. We analyze properties, estimation accuracy, obtain different forms of correct ST estimators for activations and weights, explain existing empirical approaches and their shortcomings, explain how latent weights arise from the mirror descent method when optimizing over probabilities. This allows to reintroduce ST methods, long known empirically, as sound approximations, apply them with clarity and develop further improvements.

1 Introduction

Neural networks with binary weights and activations have much lower computation costs and memory consumption than their real-valued counterparts [18, 26, 45]. They are therefore very attractive for applications in mobile devices, robotics and other resource-limited settings, in particular for solving vision and speech recognition problems [8, 56].

The seminal works that showed feasibility of training networks with binary weights [15] and binary weights and activations [27] used the empirical straight-through gradient estimation approach. In this approach the derivative of a step

We gratefully acknowledge support by Czech OP VVV project "Research Center for Informatics (CZ.02.1.01/0.0/0.0/16019/0000765)".

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-030-92659-5_7.

© Springer Nature Switzerland AG 2021

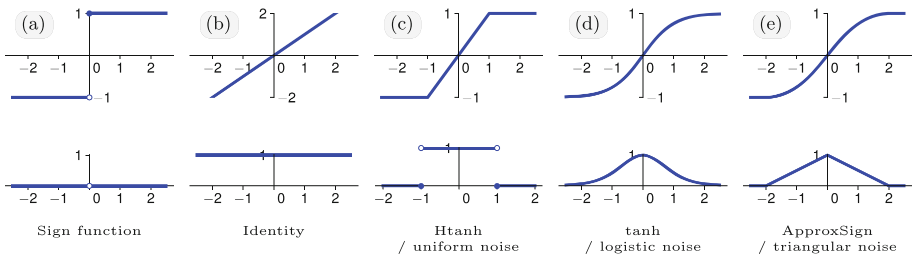

Fig. 1. The sign function and different proxy functions for derivatives used in empirical ST estimators. Variants (c-e) can be obtained by choosing the noise distribution in our framework. Specifically for a real-valued noise z with cdf F, in the upper plots we show $\mathbb{E}_z[\text{sign}(a - z)] = 2F - 1$ and, respectively, twice the density, $2F'$ in the lower plots. Choosing *uniform* distribution for z gives the density $p(z) = \frac{1}{2}\mathbb{1}_{z\in[-1,1]}$ and recovers the common Htanh proxy in (c). The *logistic* noise has cdf $F(z) = \sigma(2z)$, which recovers tanh proxy in (d). The *triangular* noise has density $p(z) = \max(0, |(2 - x)/4|)$, which recovers a scaled version of ApproxSign [34] in (e). The scaling (standard deviation) of the noise in each case is chosen so that $2F'(0) = 1$. The identity ST form in (b) we recover as latent weight updates with mirror descent.

function like sign, which is zero, is substituted with the derivative of some other function, hereafter called a *proxy* function, on the backward pass. One possible choice is to use *identity* proxy, *i.e.*, to completely bypass sign on the backward pass, hence the name *straight-through* [5]. This ad-hoc solution appears to work surprisingly well and the later mainstream research on binary neural networks heavily relies on it [2,6,9,11,18,34,36,45,52,60].

The *de-facto standard* straight-through approach in the above mentioned works is to use deterministic binarization and the clipped identity proxy as proposed by Hubara et al. [27]. However, other proxy functions were experimentally tried, including tanh and piece-wise quadratic ApproxSign [18,34], illustrated in Fig. 1. This gives rise to a diversity of empirical ST methods, where various choices are studied purely experimentally [2,6,52]. Since binary weights can be also represented as a sign mapping of some real-valued *latent weights*, the same type of methods is applied to weights. However, often a different proxy is used for the weights, producing additional unclear choices. The dynamics and interpretation of latent weights are also studied purely empirically [51]. With such obscurity of latent weights, Helwegen et al. [24] argues that "latent weights do not exist" meaning that discrete optimization over binary weights needs to be considered. The existing partial justifications of deterministic straight-through approaches are limited to one-layer networks with Gaussian data [58] or binarization of weights only [1] and do not lead to practical recommendations.

In contrast to the deterministic variant used by the mainstream SOTA, straight-through methods were originally proposed (also empirically) for stochastic autoencoders [25] and studied in models with stochastic binary neurons [5,44]. In the *stochastic binary network* (SBN) model which we consider, all hidden units and/or weights are Bernoulli random variables. The expected loss is a truly differentiable function of parameters (*i.e.*, weight probabilities) and its gradient can be estimated. This framework allows to pose questions such as: "What is the true expected gradient?" and "How far from it is the estimate computed by ST?" Towards computing the true gradient, unbiased gradient estimators were developed [20,55,57], which however have not been applied to networks with deep binary dependencies due to increased variance in deep layers and complexity that grows quadratically with the number of layers [48]. Towards explaining ST methods in SBNs, Tokui & Sato [54] and Shekhovtsov et al. [48] showed how to *derive* ST under linearizing approximations in SBNs. These results however were secondary in these works, obtained from more complex methods. They remained unnoticed in the works applying ST in practice and recent works on its analysis [13,58]. They are not properly related to existing empirical ST variants for activations and weights and did not propose analysis.

The goal of this work is to reintroduce straight-through estimators in a principled way in SBNs, to formalize and systematize empirical ST approaches for activation and weights in shallow and deep models. Towards this goal we review the derivation and formalize many empirical variants and algorithms using the derived method and sound optimization frameworks: we show how different kinds of ST estimators can occur as valid modeling choices or valid optimization choices. We further study properties of ST estimator and its utility for optimization: we theoretically predict and experimentally verify the improvement of accuracy with network width and show that popular modifications such as deterministic ST decrease this accuracy. For deep SBNs with binary weights we demonstrate that several estimators lead to equivalent results, as long as they are applied consistently with the model and the optimization algorithm.

More details on the related work, including alternative approaches for SBNs we discuss in Appendix A.

2 Derivation and Analysis

Notation. We model random states $\boldsymbol{x} \in \{-1,1\}^n$ using the noisy sign mapping:

$$x_i = \operatorname{sign}(a_i - z_i), \tag{1}$$

where z_i are real-valued independent noises with a fixed cdf F and a_i are (input-dependent) parameters. Equivalently to (1), we can say that x_i follows $\{-1,1\}$ valued Bernoulli distribution with probability $p(x_i{=}1) = \mathbb{P}(a_i{-}z_i \geq 0) = \mathbb{P}(z_i \leq a_i) = F(a_i)$. The noise cdf F will play an important role in understanding different schemes. For logistic noise, its cdf F is the logistic sigmoid function σ.

Derivation. Straight-through method was first proposed empirically [25,32] in the context of stochastic autoencoders, highly relevant to date [*e.g.* 16]. In contrast to more recent works applying variants of deterministic ST methods, these earlier works considered stochastic networks. It turns out that in this context it is possible to derive ST estimators exactly in the same form as originally proposed by Hinton. This is why we will first derive, using observations of [48,54], analyze and verify it for stochastic autoencoders.

Algorithm 1: Straight-Through-Activations

```
   /* $a$: preactivation          */
   /* $F$: injected noise cdf     */
   /* $x \in \{-1,1\}^n$          */
1  Forward( $a$ )
2      $p = F(a)$;
3      return $x \sim 2\text{Bernoulli}(p) - 1$;
4  Backward( $\frac{d\mathcal{L}}{dx}$ )
5      return $\frac{d\mathcal{L}}{da} \equiv 2\text{diag}(F'(a))\frac{d\mathcal{L}}{dx}$;
```

Algorithm 2: Straight-Through-Weights

```
   /* $\eta$: latent weights      */
   /* $F$: weight noise cdf       */
   /* $w \in \{-1,1\}^d$          */
1  Forward( $\eta$ )
2      $p = F(\eta)$;
3      return $w \sim 2\text{Bernoulli}(p) - 1$;
4  Backward( $\frac{d\mathcal{L}}{dw}$ )
5      return $\frac{d\mathcal{L}}{d\eta} \equiv 2\frac{d\mathcal{L}}{dw}$;
```

Let $\boldsymbol{y}$ denote observed variables. The *encoder network*, parametrized by $\boldsymbol{\phi}$, computes logits $\boldsymbol{a}(\boldsymbol{y};\boldsymbol{\phi})$ and samples a binary latent state $\boldsymbol{x}$ via (1). As noises $\boldsymbol{z}$ are independent, the conditional distribution of hidden states given observations $p(\boldsymbol{x}|\boldsymbol{y};\boldsymbol{\phi})$ factors as $\prod_i p(x_i|\boldsymbol{y};\boldsymbol{\phi})$. The *decoder* reconstructs observations with $p^{\text{dec}}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$—another neural network parametrized by $\boldsymbol{\theta}$. The autoencoder reconstruction loss is defined as

$$\mathbb{E}_{\boldsymbol{y}\sim\text{data}}\big[\mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x}|\boldsymbol{y};\boldsymbol{\phi})}[-\log p^{\text{dec}}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})]\big]. \tag{2}$$

The main challenge is in estimating the gradient w.r.t. the encoder parameters $\boldsymbol{\phi}$ (differentiation in $\boldsymbol{\theta}$ can be simply taken under the expectation). The problem for a fixed observation $\boldsymbol{y}$ takes the form

$$\tfrac{\partial}{\partial\boldsymbol{\phi}}\mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x};\boldsymbol{\phi})}[\mathcal{L}(\boldsymbol{x})] = \tfrac{\partial}{\partial\boldsymbol{\phi}}\mathbb{E}_{\boldsymbol{z}}[\mathcal{L}(\text{sign}(\boldsymbol{a}-\boldsymbol{z}))], \tag{3}$$

where $p(\boldsymbol{x};\boldsymbol{\phi})$ is a shorthand for $p(\boldsymbol{x}|\boldsymbol{y};\boldsymbol{\phi})$ and $\mathcal{L}(\boldsymbol{x}) = -\log p^{\text{dec}}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$. The reparametrization trick, *i.e.*, to draw one sample of $\boldsymbol{z}$ in (3) and differentiate $\mathcal{L}(\text{sign}(\boldsymbol{a}-\boldsymbol{z}))$ fails: since the loss as a function of $\boldsymbol{a}$ and $\boldsymbol{z}$ is not *continuously differentiable we cannot interchange the gradient and the expectation in $\boldsymbol{z}$*[1]. If we nevertheless attempt the interchange, we obtain that the gradient of $\text{sign}(\boldsymbol{a}-\boldsymbol{z})$ is zero as well as its expectation. Instead, the following steps lead to an unbiased low-variance estimator. From the LHS of (3) we express the derivative as

$$\tfrac{\partial}{\partial\boldsymbol{\phi}}\textstyle\sum_{\boldsymbol{x}}\big(\prod_i p(x_i;\boldsymbol{\phi})\big)\mathcal{L}(\boldsymbol{x}) = \sum_{\boldsymbol{x}}\sum_i\big(\prod_{i'\neq i} p(x_{i'};\boldsymbol{\phi})\big)\big(\tfrac{\partial}{\partial\boldsymbol{\phi}}p(x_i;\boldsymbol{\phi})\big)\mathcal{L}(\boldsymbol{x}). \tag{4}$$

Then we apply *derandomization* [40, ch.8.7], which performs summation over x_i holding the rest of the state $\boldsymbol{x}$ fixed. Because x_i takes only two values, we have

$$\begin{aligned}\textstyle\sum_{x_i}\tfrac{\partial p(x_i;\boldsymbol{\phi})}{\partial\boldsymbol{\phi}}\mathcal{L}(\boldsymbol{x}) &= \tfrac{\partial p(x_i;\boldsymbol{\phi})}{\partial\boldsymbol{\phi}}\mathcal{L}(\boldsymbol{x}) + \tfrac{\partial(1-p(x_i;\boldsymbol{\phi}))}{\partial\boldsymbol{\phi}}\mathcal{L}(\boldsymbol{x}_{\downarrow i})\\ &= \tfrac{\partial}{\partial\boldsymbol{\phi}}p(x_i;\boldsymbol{\phi})\big(\mathcal{L}(\boldsymbol{x})-\mathcal{L}(\boldsymbol{x}_{\downarrow i})\big),\end{aligned} \tag{5}$$

[1] The conditions allow to apply Leibniz integral rule to exchange derivative and integral. Other conditions may suffice, *e.g.*, when using weak derivatives [17].

where $\boldsymbol{x}_{\downarrow i}$ denotes the full state vector $\boldsymbol{x}$ with the sign of x_i flipped. Since this expression is now invariant of x_i, we can multiply it with $1 = \sum_{x_i} p(x_i; \boldsymbol{\phi})$ and express the gradient (4) in the form:

$$\begin{aligned}\textstyle\sum_i \sum_{\boldsymbol{x}_{\neg i}} \big(\prod_{i' \neq i} p(x_{i'}; \boldsymbol{\phi})\big) \sum_{x_i} p(x_i; \boldsymbol{\phi}) \frac{\partial p(x_i;\boldsymbol{\phi})}{\partial \boldsymbol{\phi}} \big(\mathcal{L}(\boldsymbol{x}) - \mathcal{L}(\boldsymbol{x}_{\downarrow i})\big) \\ \textstyle\sum_{\boldsymbol{x}} \big(\prod_{i'} p(x_{i'}; \boldsymbol{\phi})\big) \sum_i \frac{\partial p(x_i;\boldsymbol{\phi})}{\partial \boldsymbol{\phi}} \big(\mathcal{L}(\boldsymbol{x}) - \mathcal{L}(\boldsymbol{x}_{\downarrow i})\big) \\ = \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x};\boldsymbol{\phi})} \textstyle\sum_i \frac{\partial p(x_i,\boldsymbol{\phi})}{\partial \boldsymbol{\phi}} \big(\mathcal{L}(\boldsymbol{x}) - \mathcal{L}(\boldsymbol{x}_{\downarrow i})\big),\end{aligned} \tag{6}$$

where $\boldsymbol{x}_{\neg i}$ denotes all states excluding x_i. To obtain an unbiased estimate, it suffices to take one sample $\boldsymbol{x} \sim p(\boldsymbol{x}; \boldsymbol{\phi})$ and compute the sum in i in (6). This estimator is known as *local expectations* [53] and coincides in this case with GO-gradient [14], RAM [54] and PSA [48].

However, evaluating $\mathcal{L}(\boldsymbol{x}_{\downarrow i})$ for all i may be impractical. A huge simplification is obtained if we assume that the change of the loss $\mathcal{L}$ when only a single latent bit x_i is changed can be approximated via linearization. Assuming that $\mathcal{L}$ is defined as a differentiable mapping $\mathbb{R}^n \to \mathbb{R}$ (*i.e.*, that the loss is built up of arithmetic operations and differentiable functions), we can approximate

$$\mathcal{L}(\boldsymbol{x}) - \mathcal{L}(\boldsymbol{x}_{\downarrow i}) \approx 2 x_i \frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial x_i}, \tag{7}$$

where we used the identity $x_i - (-x_i) = 2x_i$. Expanding the derivative of conditional density $\frac{\partial}{\partial \boldsymbol{\phi}} p(x_i; \boldsymbol{\phi}) = x_i F'(a_i(\boldsymbol{\phi})) \frac{\partial}{\partial \boldsymbol{\phi}} a_i(\boldsymbol{\phi})$, we obtain

$$\frac{\partial p(x_i,\boldsymbol{\phi})}{\partial \boldsymbol{\phi}} \big(\mathcal{L}(\boldsymbol{x}) - \mathcal{L}(\boldsymbol{x}_{\downarrow i})\big) \approx 2F'(a_i(\boldsymbol{\phi})) \frac{\partial a_i(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}} \frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial x_i}. \tag{8}$$

If we now define that $\frac{\partial x_i}{\partial a_i} \equiv 2F'(a_i)$, the summation over i in (6) with the approximation (8) can be written in the form of a chain rule:

$$\textstyle\sum_i 2F'(a_i(\boldsymbol{\phi})) \frac{\partial a_i(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}} \frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial x_i} = \sum_i \frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial x_i} \frac{\partial x_i}{\partial a_i} \frac{\partial a_i(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}}. \tag{9}$$

To clarify, the estimator is already defined by the LHS of (9). We simply want to compute this expression by (ab)using the standard tools, and this is the sole purpose of introducing $\frac{\partial x_i}{\partial a_i}$. Indeed the RHS of (9) is a product of matrices that would occur in standard backpropagation. We thus obtained ST algorithm Algorithm 1. We can observe that *it matches exactly to the one described by Hinton* [25]*: to sample on the forward pass and use the derivative of the noise cdf on the backward pass*, up to the multiplier 2 which occurred due to the use of ± 1 encoding for $\boldsymbol{x}$.

2.1 Analysis

Next we study properties of the derived ST algorithm and its relation to empirical variants. We will denote a modification of Algorithm 1 that does not use sampling in Line 3, but instead computes $x = \mathrm{sign}(a)$, a *deterministic ST*; and a modification that uses derivative of some other function G instead of F in Line 5 as *using a proxy G*.

Invariances. Observe that binary activations (and hence the forward pass) stay invariant under transformations: $\operatorname{sign}(a_i - z_i) = \operatorname{sign}(T(a_i) - T(z_i))$ for any strictly monotone mapping T. Consistently, *the ST gradient by Algorithm 1 is also invariant to T*. In contrast, empirical straight-through approaches, in which the derivative proxy is hand-designed, fail to maintain this property. In particular, rescaling the proxy leads to different estimators.

Furthermore, when applying transform $T = F$ (the noise cdf), the backpropagation rule in line 5 of Algorithm 1 becomes equivalent to using the identity proxy. Hence we see that a common description of ST in the literature as "to back-propagate through the hard threshold function as if it had been the identity function" is also correct, *but only for the case of uniform noise* in $[-1, 1]$. Otherwise, and especially so for deterministic ST, this description is ambiguous because the resulting gradient estimator crucially depends on what transformations were applied under the hard threshold.

ST Variants. Using the invariance property, many works applying randomized ST estimators are easily seen to be equivalent to Algorithm 1: [16,44,49]. Furthermore, using different noise distributions for $\boldsymbol{z}$, we can obtain correct ST analogues for common choices of sign proxies used in empirical ST works as shown in Fig. 1 (c–e). In our framework they correspond to the choice of parametrization of the conditional Bernoulli distribution, which should be understood similarly to how a neural network can be parametrized in different ways.

If the "straight-through" idea is applied informally, however, this may lead to confusion and poor performance. The most cited reference for the ST estimator is Bengio et al. [5]. However, [5, Eq. 13] defines in fact the identity ST variant, incorrectly attributing it to Hinton (see Appendix A). We will show this variant to be less accurate for hidden units, both theoretically and experimentally. Pervez et al. [42] use ± 1 binary encoding but apply ST estimator without coefficient 2. When such estimator is used in VAE, where the gradient of the prior KL divergence is computed analytically, it leads to a significant bias of the total gradient towards the prior. In Fig. 2 we illustrate that the difference in performance may be substantial. We analyze other techniques introduced in FouST in more detail in [47]. An inappropriate scaling by a factor of 2 can be as well detrimental in deep models, where the factor would be applied multiple times (in each layer).

Bias Analysis. Given a rather crude linearization involved, it is indeed hard to obtain fine theoretical guarantees about the ST method. We propose an analysis targeting understanding the effect of common empirical variants and understanding conditions under which the estimator becomes more accurate. The respective formal theorems are given in Appendix B.

I) When ST is unbiased? As we used linearization as the only biased approximation, it follows that *Algorithm 1 is unbiased if the objective function $\mathcal{L}$ is multilinear in $\boldsymbol{x}$*. A simple counter-example, where ST is biased, is $\mathcal{L}(x) = x^2$. In this case the expected value of the loss is 1, independently of a that determines x; and the true gradient is zero. However the expected ST gradient is $\mathbb{E}[2F'(a)2x] = 4F'(a)(2F(a) - 1)$, which may be positive or negative depend-

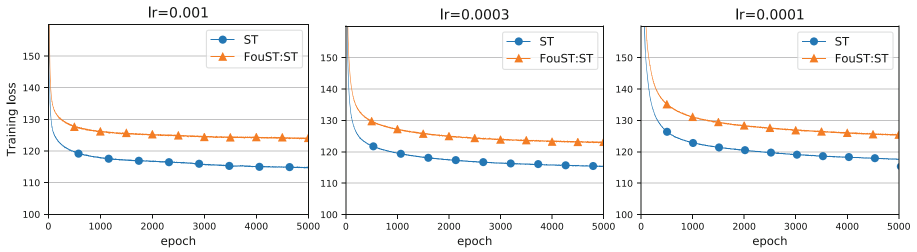

Fig. 2. Training VAE on MNIST, closely following experimental setup [42]. The plots show training loss (negative ELBO) during epochs for different learning rates. The variant of ST algorithm used [42] is misspecified because of the scaling factor and performs substantially worse at for all learning rates. Full experiment specification is given in Appendix D.1.

ing on a. On the other hand, any function of binary variables has an equivalent multilinear expression. In particular, if we consider $\mathcal{L}(\boldsymbol{x}) = \|\boldsymbol{W}\boldsymbol{x}-\boldsymbol{y}\|^2$, analyzed by Yin et al. [58], then $\tilde{\mathcal{L}}(\boldsymbol{x}) = \|\boldsymbol{W}\boldsymbol{x}-\boldsymbol{y}\|^2 - \sum_i x_i^2\|\boldsymbol{W}_{:,i}\|^2 + \sum_i \|\boldsymbol{W}_{:,i}\|^2$ *coincides with* $\mathcal{L}$ *on all binary configurations and is multilinear.* It follows that ST applied to $\tilde{\mathcal{L}}$ gives an unbiased gradient estimate of $\mathbb{E}[\mathcal{L}]$, an immediate improvement compared to [58]. In the special case when $\mathcal{L}$ is linear in $\boldsymbol{x}$, the ST estimator is not only unbiased but has a zero variance, *i.e.*, it is exact.

II) How does using a mismatched proxy in Line 5 of Algorithm 1 affect the gradient in $\boldsymbol{\phi}$? Since $\operatorname{diag}(F')$ occurs in the backward chain, we call estimators that use some matrix $\boldsymbol{\Lambda}$ instead of $\operatorname{diag}(F')$ as *internally rescaled.* We show that *for any* $\boldsymbol{\Lambda} \succcurlyeq 0$, *the expected rescaled estimator has non-negative scalar product with the expected original estimator.* Note that this is not completely obvious as the claim is about the final gradient in the model parameters $\boldsymbol{\phi}$ (*e.g.*, weights of the encoder network in the case of autoencoders). However, if the ST gradient by Algorithm 1 is biased (when $\mathcal{L}$ is not multi-linear) but is nevertheless an ascent direction in expectation, the expected rescaled estimator may fail to be an ascent direction, *i.e.*, to have a positive scalar product with the true gradient.

III) When does ST gradient provide a valid ascent direction? Assuming that all partial derivatives $g_i(\boldsymbol{x}) = \frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial x_i}$ are L-Lipschitz continuous for some L, we can show that *the expected ST gradient is an ascent direction for any network if and only if* $\left|\mathbb{E}_{\boldsymbol{x}}[g_i(\boldsymbol{x})]\right| > L$ *for all* i.

IV) Can we decrease the bias? Assume that the loss function is applied to a linear transform of Bernoulli variables, *i.e.*, takes the form $\mathcal{L}(\boldsymbol{x}) = \ell(\boldsymbol{W}\boldsymbol{x})$. A typical initialization uses random $\boldsymbol{W}$ normalized by the size of the fan-in, *i.e.*, such that $\|\boldsymbol{W}_{k,:}\|_2 = 1\ \forall k$. In this case *the Lipschitz constant of gradients of* $\mathcal{L}$ *scales as* $O(1/\sqrt{n})$, *where* n *is the number of binary variables.* Therefore, *using more binary variables decreases the bias, at least at initialization.*

V) Does deterministic ST give an ascent direction? Let $\boldsymbol{g}^*$ be the deterministic ST gradient for the state $\boldsymbol{x}^* = \operatorname{sign}(\boldsymbol{a})$ and $p^* = p(\boldsymbol{x}^*|\boldsymbol{a})$ be its probability. *We show that deterministic ST gradient forms a positive scalar product* *with the expected ST gradient if* $|g_i^*| \geq 2L(1-p^*)$ *and with the true gradient if* $|g_i^*| \geq 2L(1-p^*)+L$. From this we conclude that deterministic ST positively correlates with the true gradient when $\mathcal{L}$ is multilinear, improves with the number of hidden units in the case described by IV and approaches expected stochastic ST as units learn to be deterministic so that the factor $(1-p^*)$ decreases.

Deep ST. So far we derived and analyzed ST for a single layer model. It turns out that simply applying Algorithm 1 in each layer of a deep model with conditional Bernoulli units gives the correct extension for this case. We will not focus on deriving deep ST here, but remark that it can be derived rigorously by chaining derandomization and linearization steps, discussed above, for each layer [48]. In particular, [48] show that ST can be obtained by making additional linearizations in their (more accurate) PSA method. The insights from the derivation are twofold. First, since derandomization is performed recurrently, the variance for deep layers is significantly reduced. Second, we know which approximations contribute to the bias, they are indeed the linearizations of all conditional Bernoulli probabilities in all layers and of the loss function as a function of the last Bernoulli layer. We may expect that using more units, similarly to property IV, would improve linearizing approximations of intermediate layers increasing the accuracy of deep ST gradient.

3 Latent Weights Do Exist!

Responding to the work "Latent weights do not exist: Rethinking binarized neural network optimization" [24] and the lack of formal basis to introduce latent weights in the literature (*e.g.*, [27]), we show that such weights can be formally defined in SBNs and that several empirical update rules do in fact correspond to sound optimization schemes: projected gradient descent, mirror descent, variational Bayesian learning.

Let $\boldsymbol{w}$ be ± 1-Bernoulli weights with $p(w_i{=}1) = \theta_i$, let $\mathcal{L}(\boldsymbol{w})$ be the loss function for a fixed training input. Consistently with the model for activations (1), we can define $w_i = \operatorname{sign}(\eta_i - z_i)$ in order to model weights w_i using parameters $\eta_i \in \mathbb{R}$ which we will call *latent weights*. It follows that $\theta_i = F_z(\eta_i)$. We need to tackle two problems in order to optimize $\mathbb{E}_{\boldsymbol{w}\sim p(\boldsymbol{w}\,|\,\boldsymbol{\theta})}[\mathcal{L}(\boldsymbol{w})]$ in probabilities $\boldsymbol{\theta}$: i) how to estimate the gradient and ii) how to handle constraints $\boldsymbol{\theta} \in [0,1]^m$.

Projected Gradient. A basic approach to handle constraints is the *projected gradient descent*:

$$\boldsymbol{\theta}^{t+1} := \operatorname{clip}(\boldsymbol{\theta}^t - \varepsilon \boldsymbol{g}^t, 0, 1), \tag{10}$$

where $\boldsymbol{g}^t = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{w}\sim p(\boldsymbol{w}\,|\,\boldsymbol{\theta}^t)}[\mathcal{L}(\boldsymbol{w})]$ and $\operatorname{clip}(\boldsymbol{x}, a, b) := \max(\min(\boldsymbol{x}, b), a)$ is the projection. Observe that for the uniform noise distribution on $[-1,1]$ with $F(z) = \operatorname{clip}(\frac{z+1}{2}, 0, 1)$, we have $\theta_i = p(w_i{=}1) = F(\eta_i) = \operatorname{clip}(\frac{\eta_i+1}{2}, 0, 1)$. Because this F is linear on $[-1,1]$, the update (10) can be equivalently reparametrized in $\boldsymbol{\eta}$ as

$$\boldsymbol{\eta}^{t+1} := \operatorname{clip}(\boldsymbol{\eta}^t - \varepsilon' \boldsymbol{h}^t, -1, 1), \tag{11}$$

where $\boldsymbol{h}^t = \nabla_{\boldsymbol{\eta}} \mathbb{E}_{\boldsymbol{w} \sim p(\boldsymbol{w} | F(\boldsymbol{\eta}))}[\mathcal{L}(\boldsymbol{w})]$ and $\varepsilon' = 4\varepsilon$. The gradient in the latent weights, $\boldsymbol{h}^t$, can be estimated by Algorithm 1 and simplifies by expanding $2F' = 1$. We obtained that *the emperically proposed method of Hubara et al.* [27, Alg.1] *with stochastic rounding and with real-valued weights identified with* $\boldsymbol{\eta}$ *is equivalent to PGD on* $\boldsymbol{\eta}$ *with constraints* $\eta \in [-1,1]^m$ *and ST gradient by Algorithm 1.*

Mirror Descent. As an alternative approach to handle constraints $\boldsymbol{\theta} \in [0,1]^m$, we study the application of mirror descent (MD) and connect it with the identity ST update variants. A step of MD is found by solving the following proximal problem:

$$\boldsymbol{\theta}^{t+1} = \min_{\boldsymbol{\theta}} \left[\langle \boldsymbol{g}^t, \boldsymbol{\theta} - \boldsymbol{\theta}^t \rangle + \tfrac{1}{\varepsilon} D(\boldsymbol{\theta}, \boldsymbol{\theta}^t) \right]. \tag{12}$$

The divergence term $\frac{1}{\varepsilon} D(\boldsymbol{\theta}, \boldsymbol{\theta}^t)$ weights how much we trust the linear approximation $\langle \boldsymbol{g}^t, \boldsymbol{\theta} - \boldsymbol{\theta}^t \rangle$ when considering a step from $\boldsymbol{\theta}^t$ to $\boldsymbol{\theta}$. When the gradient is stochastic we speak of *stochastic mirror descent* (SMD) [3,59]. A common choice of divergence to handle probability constraints is the KL-divergence $D(\theta_i, \theta_i^t) = \mathrm{KL}(\mathrm{Ber}(\theta_i), \mathrm{Ber}(\theta_i^t)) = \theta_i \log(\frac{\theta_i}{\theta_i^t}) + (1-\theta_i) \log(\frac{1-\theta_i}{1-\theta_i^t})$. Solving for a stationary point of (12) gives

$$0 = g_i^t + \tfrac{1}{\varepsilon}\big(\log(\tfrac{\theta_i}{1-\theta_i}) - \log(\tfrac{\theta_i^t}{1-\boldsymbol{\theta}_i^t}) \big). \tag{13}$$

Observe that when $F = \sigma$ we have $\log(\frac{\theta_i}{1-\theta_i}) = \eta_i$. Then the MD step can be written in the well-known convenient form using the latent weights $\boldsymbol{\eta}$ (natural parameters of Bernoulli distribution):

$$\boldsymbol{\theta}^t := \sigma(\boldsymbol{\eta}^t); \qquad \boldsymbol{\eta}^{t+1} := \boldsymbol{\eta}^t - \varepsilon \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^t). \tag{14}$$

We thus have obtained the rule where on the forward pass $\boldsymbol{\theta} = \sigma(\boldsymbol{\eta})$ defines the sampling probability of $\boldsymbol{w}$ and on the backward pass the derivative of σ, that otherwise occurs in Line 5 of Algorithm 1, *is bypassed exactly as if the identity proxy was used.* We define such ST rule for optimization in weights as Algorithm 2. Its correctness is not limited to logistic noise. We show that for any strictly monotone noise distribution F there is a corresponding divergence function D:

Proposition 1. *Common SGD in latent weights* $\boldsymbol{\eta}$ *using the* identity straight-through-weights *Algorithm 2 implements SMD in the weight probabilities* $\boldsymbol{\theta}$ *with the divergence corresponding to* F.

Proof in Appendix C. Proposition 1 reveals that although Bernoulli weights can be modeled the same way as activations using the injected noise model $\boldsymbol{w} = \mathrm{sign}(\boldsymbol{\eta} - \boldsymbol{z})$, *the noise distribution* F *for weights correspond to the choice of the optimization proximity scheme.*

Despite generality of Proposition 1, we view the KL divergence as a more reliable choice in practice. Azizan et al. [3] have shown that the optimization with SMD has an inductive bias to find the closest solution to the initialization point as measured by the divergence used in MD, which has a strong impact on generalization. This suggests that MD with KL divergence will prefer higher entropy solutions, making more diverse predictions. It follows that SGD on latent weights with logistic noise and identity straight-through Algorithm 2 enjoys the same properties.

Variational Bayesian Learning. Extending the results above, we study the variational Bayesian learning formulation and show the following:

Proposition 2. *Common SGD in latent weights* $\boldsymbol{\eta}$ with a weight decay and identity straight-through-weights *Algorithm 2 is equivalent to optimizing a factorized variational approximation to the weight posterior* $p(\boldsymbol{w}|data)$ *using a composite SMD method.*

Proof in Appendix C.2. As we can see, powerful and sound learning techniques can be obtained in a form of simple update rules using identity straight-through estimators. Therefore, identity-ST is fully rehabilitated in this context.

4 Experiments

Stochastic Autoencoders. Previous work has demonstrated that Gumbel-Softmax (biased) and ARM (unbiased) estimators give better results than ST on training variational autoencoders with Bernoulli latents [16,29,57]. However, only the test performance was revealed to readers. We investigate in more detail what happens during training. Except of studying the training loss under the same training setup, we measure the gradient approximation accuracy using ARM with 1000 samples as the reference.

We train a simple yet realistic variant of stochastic autoencoder for the task of text retrieval with binary representation on *20newsgroups* dataset. The autoencoder is trained by minimizing the reconstruction loss (2). Please refer to Appendix D.2 for full specification of the model and experimental setup.

For each estimator we perform the following protocol. First, we train the model with this estimator using Adam with $lr = 0.001$ for 1000 epochs. We then switch the estimator to ARM with 10 samples and continue training for 500 more epochs (denoted as ARM-10 correction phase). Figure 3 top shows the training performance for different number of latent bits n. It is seen (esp. for 8 and 64 bits) that some estimators (esp. ST and det_ST) appear to make no visible progress, and even increase the loss, while switching them to ARM makes a rapid improvement. Does it mean that these estimators are bad and ARM is very good? An explanation of this phenomenon is offered in Fig. 5. The rapid improvement by ARM is possible because these estimators have accumulated a significant bias due to a systematic error component, which nevertheless can be easily corrected by an unbiased estimator.

To measure the bias and alignment of directions, as theoretically analyzed in Sect. 2.1, we evaluate different estimators at the same parameter points located

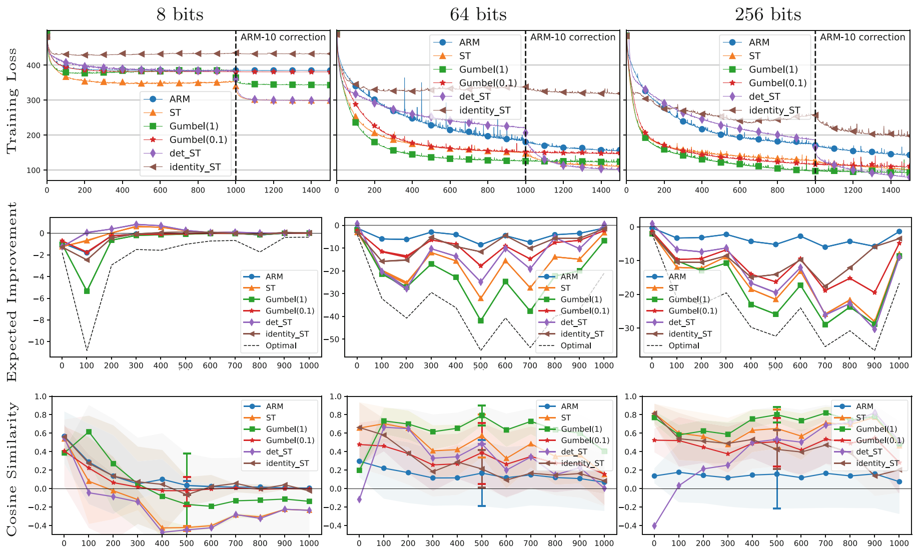

Fig. 3. Comparison of the training performance and gradient estimation accuracy for a stochastic autoencoder with different number of latent Bernoulli units (bits). *Training Loss:* each estimator is applied for 1000 epochs and then switched to ARM-10 in order to correct the accumulated bias. *Expected improvement*: lower is better (measures expected change of the loss), the dashed line shows the maximal possible improvement knowing the true gradient. *Cosine similarity*: higher is better, close to 1 means that the direction is accurate while below 0 means the estimated gradient is not an ascent direction; error bars indicate empirical 70% confidence intervals using 100 trials.

along the learning trajectory of the reference ARM estimator. At each such point we estimate the true gradient $\boldsymbol{g}$ by ARM-1000. To measure the quality of a candidate 1-sample estimator $\tilde{\boldsymbol{g}}$ we compute the *expected cosine similarity* and the *expected improvement*, defined respectively as:

$$\mathrm{ECS} = \mathbb{E}\Big[\langle \boldsymbol{g}, \tilde{\boldsymbol{g}}\rangle / (\|\boldsymbol{g}\|\|\tilde{\boldsymbol{g}}\|)\Big], \qquad \mathrm{EI} = -\mathbb{E}[\langle \boldsymbol{g}, \tilde{\boldsymbol{g}}\rangle] / \sqrt{\mathbb{E}[\|\tilde{\boldsymbol{g}}\|^2]}, \tag{15}$$

The expectations are taken over 100 trials and all batches. A detailed explanation of these metrics is given in Appendix D.2. These measurements, displayed in Fig. 3 for different bit length, clearly show that with a small bit length biased estimators consistently run into producing wrong directions. *Identity ST and deterministic ST clearly introduce an extra bias to ST*. However, when we increase the number of latent bits, the accuracy of all biased estimators improves, confirming our analysis **IV**, **V**.

The practical takeaways are as follows: 1) biased estimators may perform significantly better than unbiased but might require a correction of the systematically accumulated bias; 2) with more units the ST approximation clearly improves and the bias has a less detrimental effect, requiring less correction; 3) Algorithm 1 is more accurate than other ST variants in estimating the true gradient.

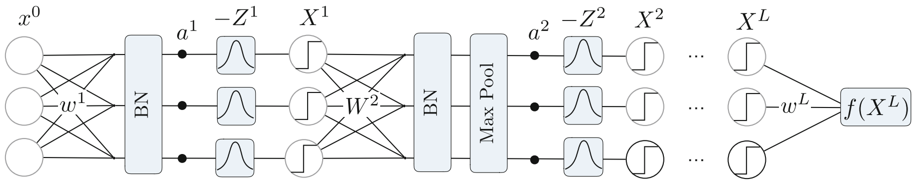

Fig. 4. Stochastic Binary Network: first and last layer have real-valued weights. BN layers have real-valued scale and bias parameters that can adjust scaling of activations relative to noise. Z are independent injected noises with a chosen distribution. Binary weights W_{ij} are random ± 1 Bernoulli(θ_{ij}) with learnable probabilities θ_{ij}. In experiments we consider SBN with a convolutional architecture same as [15,27]: (2×128C3) – MP2 – (2×256C3) – MP2 – (2×512C3) – MP2 – (2×1024FC) – 10FC – softmax.

Classification with Deep SBN. In this section we verify Algorithm 1 with different choice of noises in a deep network and verify optimization in binary weight probabilities using SGD on latent weights with Algorithm 2. We consider CIFAR-10 dataset and use the SBN model illustrated in Fig. 4. The SBN model, its initialization and the full learning setup is detailed in Appendix D.3. We trained this SBN with three choices of noise distributions corresponding to proxies used by prior work as in Fig. 1 (c–e). Table 1 shows the test results in comparison with baselines.

We see that training with different choices of noise distributions, corresponding to different ST rules, all achieves similar results. This is in contrast to empirical studies advocating specific proxies and is allowed by the consistency of the model, initialization and training. The identity ST applied to weights, implementing SMD updates, works well. Comparing to empirical ST baselines (all except Peters & Welling), we see that there is no significant difference in the 'det' column indicating that our derived ST method is on par with the well-guessed baselines. If the same networks are tested in the stochastic mode ('10-sample' column), there is a clear boost of performance, indicating an advantage of SBN models. Out of the two experiments of Hubara et al., randomized training (rand.) also appears better confirming advantage of stochastic ST. In the stochastic mode, there is a small gap to Peters & Welling, who use a different estimation method and pretraining. Pretraining a real valued network also seem important, *e.g.*, [19] report 91.7% accuracy with VGG-Small using pretraining and a smooth transition from continuous to binarized model. When our method is applied with an initialization from a pretrained model, improved results (92.6% 10-sample test accuracy) can be obtained with even a smaller network [35]. There are however even more superior results in the literature, *e.g.*, using neural architecture search with residual real connections, advanced data augmentation techniques and model distillation [10] achieve 96.1%.

The takeaway message here is that ST can be considered in the context of deep SBN models as a simple and robust method if the estimator matches the model and is applied correctly. Since we achieve experimentally near 100% training accuracy in all cases, the optimization fully succeeds and thus the bias of ST is tolerable.

Table 1. Test accuracy for different methods on CIFAR-10 with the same/similar architecture. SBN can be tested either with zero noises (*det*) or using an ensemble of several samples (we use *10-sample*). Standard deviations are given w.r.t. to 4 trials with random initialization. The two quotations for Hubara et al. [27] refer to their result with Torch7 implementation using randomized Htanh and Theano implementation using deterministic Htanh, respectively.

Method	det	10-sample
Stochastic training		
Our SBN, logistic noise	89.6 ± 0.1	90.6 ± 0.2
Our SBN, uniform noise	89.7 ± 0.2	90.5 ± 0.2
Our SBN, triangular noise	89.5 ± 0.2	90.0 ± 0.3
Hubara et al. [27] (rand.)	89.85	-
Peters & Welling [43]	88.61	16-sample: 91.2
Deterministic training		
Rastegari et al. [45]	89.83	-
Hubara et al. [27] (det.)	88.60	-

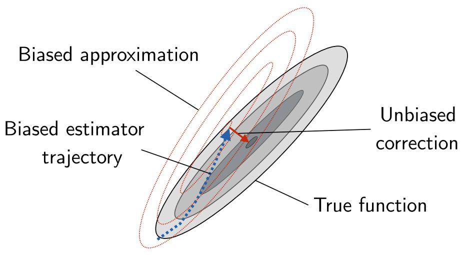

Fig. 5. Schematic explanation of the optimization process using a biased estimator followed by a correction with an unbiased estimator. Initially, the biased estimator makes good progress, but then the value of the true loss function may start growing while the optimization steps nevertheless come closer to the optimal location in the parameter space.

5 Conclusion

We have put many ST methods on a solid basis by deriving and explaining them from the first principles in one framework. It is well-defined what they estimate and what the bias means. We obtained two different main estimators for propagating activations and weights, bringing the understanding which function they have, what approximations they involve and what are the limitations imposed by these approximations. The resulting methods in all cases are strikingly simple, no wonder they have been first discovered empirically long ago. We showed how our theory leads to a useful understanding of bias properties and to reasonable choices that allow for a more reliable application of these methods. We hope that researchers will continue to use these simple techniques, now with less guesswork and obscurity, as well as develop improvements to them.

References

1. Ajanthan, T., Gupta, K., Torr, P.H., Hartley, R., Dokania, P.K.: Mirror descent view for neural network quantization. arXiv preprint arXiv:1910.08237 (2019)
2. Alizadeh, M., Fernandez-Marques, J., Lane, N.D., Gal, Y.: An empirical study of binary neural networks' optimisation. In: ICLR (2019)
3. Azizan, N., Lale, S., Hassibi, B.: A study of generalization of stochastic mirror descent algorithms on overparameterized nonlinear models. In: ICASSP, pp. 3132–3136 (2020)

4. Bai, Y., Wang, Y.-X., Liberty, E.: ProxQuant: quantized neural networks via proximal operators. In: ICLR (2019)
5. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013)
6. Bethge, J., Yang, H., Bornstein, M., Meinel, C.: Back to simplicity: how to train accurate BNNs from scratch? CoRR, abs/1906.08637 (2019)
7. Boros, E., Hammer, P.: Pseudo-Boolean optimization. Discret. Appl. Math. **1–3**(123), 155–225 (2002)
8. Bulat, A., Tzimiropoulos, G.: Binarized convolutional landmark localizers for human pose estimation and face alignment with limited resources. In: ICCV, October 2017
9. Bulat, A., Tzimiropoulos, G., Kossaifi, J., Pantic, M.: Improved training of binary networks for human pose estimation and image recognition. arXiv (2019)
10. Bulat, A., Martinez, B., Tzimiropoulos, G.: BATS: binary architecture search (2020)
11. Bulat, A., Martinez, B., Tzimiropoulos, G.: High-capacity expert binary networks. In: ICLR (2021)
12. Chaidaroon, S., Fang, Y.: Variational deep semantic hashing for text documents. In: SIGIR Conference on Research and Development in Information Retrieval, pp. 75–84 (2017)
13. Cheng, P., Liu, C., Li, C., Shen, D., Henao, R., Carin, L.: Straight-through estimator as projected Wasserstein gradient flow. arXiv preprint arXiv:1910.02176 (2019)
14. Cong, Y., Zhao, M., Bai, K., Carin, L.: GO gradient for expectation-based objectives. In: ICLR (2019)
15. Courbariaux, M., Bengio, Y., David, J.-P.: BinaryConnect: training deep neural networks with binary weights during propagations. In: NeurIPS, pp. 3123–3131 (2015)
16. Dadaneh, S.Z., Boluki, S., Yin, M., Zhou, M., Qian, X.: Pairwise supervised hashing with Bernoulli variational auto-encoder and self-control gradient estimator. arXiv, abs/2005.10477 (2020)
17. Dai, B., Guo, R., Kumar, S., He, N., Song, L.: Stochastic generative hashing. In: ICML 2017, pp. 913–922 (2017)
18. Esser, S.K., et al.: Convolutional networks for fast, energy-efficient neuromorphic computing. Proc. Natl. Acad. Sci. **113**(41), 11441–11446 (2016)
19. Gong, R., et al.: Differentiable soft quantization: bridging full-precision and low-bit neural networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), October 2019
20. Grathwohl, W., Choi, D., Wu, Y., Roeder, G., Duvenaud, D.: Backpropagation through the void: optimizing control variates for black-box gradient estimation. In: ICLR (2018)
21. Graves, A.: Practical variational inference for neural networks. In: NeurIPS, pp. 2348–2356 (2011)
22. Gregor, K., Danihelka, I., Mnih, A., Blundell, C., Wierstra, D.: Deep autoregressive networks. In: ICML (2014)
23. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. In: ICCV, pp. 1026–1034 (2015)
24. Helwegen, K., Widdicombe, J., Geiger, L., Liu, Z., Cheng, K.-T., Nusselder, R.: Latent weights do not exist: rethinking binarized neural network optimization. In: NeurIPS, pp. 7531–7542 (2019)

25. Hinton, G.: Lecture 15D - Semantic hashing: 3:05–3:35 (2012). https://www.cs.toronto.edu/~hinton/coursera/lecture15/lec15d.mp4
26. Horowitz, M.: Computing's energy problem (and what we can do about it). In: International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), pp. 10–14 (2014)
27. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks. In: NeurIPS, pp. 4107–4115 (2016)
28. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: ICML, vol. 37, pp. 448–456 (2015)
29. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with Gumbel-Softmax. In: ICLR (2017)
30. Khan, E., Rue, H.: Learning algorithms from Bayesian principles. Draft v. 0.7, August 2020
31. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: ICLR (2015)
32. Krizhevsky, A., Hinton, G.E.: Using very deep autoencoders for content-based image retrieval. In: ESANN (2011)
33. Lin, W., Khan, M.E., Schmidt, M.: Fast and simple natural-gradient variational inference with mixture of exponential-family approximations. In: ICML, vol. 97, June 2019
34. Liu, Z., Wu, B., Luo, W., Yang, X., Liu, W., Cheng, K.-T.: Bi-real net: enhancing the performance of 1-bit CNNs with improved representational capability and advanced training algorithm. In: ECCV, pp. 722–737 (2018)
35. Livochka, A., Shekhovtsov, A.: Initialization and transfer learning of stochastic binary networks from real-valued ones. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops (2021)
36. Martínez, B., Yang, J., Bulat, A., Tzimiropoulos, G.: Training binary neural networks with real-to-binary convolutions. In: ICLR (2020)
37. Meng, X., Bachmann, R., Khan, M.E.: Training binary neural networks using the Bayesian learning rule. In: ICML (2020)
38. Nanculef, R., Mena, F.A., Macaluso, A., Lodi, S., Sartori, C.: Self-supervised Bernoulli autoencoders for semi-supervised hashing. CoRR, abs/2007.08799 (2020)
39. Nemirovsky, A.S., Yudin, D.B.: Problem complexity and method efficiency in optimization (1983)
40. Owen, A.B.: Monte Carlo theory, methods and examples (2013)
41. Paszke, A., et al.: Pytorch: an imperative style, high-performance deep learning library. In: NeurIPS, pp. 8024–8035 (2019)
42. Pervez, A., Cohen, T., Gavves, E.: Low bias low variance gradient estimates for Boolean stochastic networks. In: ICML, vol. 119, pp. 7632–7640, 13–18 July 2020
43. Peters, J.W., Welling, M.: Probabilistic binary neural networks. arXiv preprint arXiv:1809.03368 (2018)
44. Raiko, T., Berglund, M., Alain, G., Dinh, L.: Techniques for learning binary stochastic feedforward neural networks. In: ICLR (2015)
45. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet classification using binary convolutional neural networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 525–542. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_32
46. Roth, W., Schindler, G., Fröning, H., Pernkopf, F.: Training discrete-valued neural networks with sign activations using weight distributions. In: European Conference on Machine Learning (ECML) (2019)
47. Shekhovtsov, A.: Bias-variance tradeoffs in single-sample binary gradient estimators. In: GCPR (2021)

48. Shekhovtsov, A., Yanush, V., Flach, B.: Path sample-analytic gradient estimators for stochastic binary networks. In: NeurIPS (2020)
49. Shen, D., et al.: NASH: toward end-to-end neural architecture for generative semantic hashing. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15–20 2018, Volume 1: Long Papers, pp. 2041–2050 (2018)
50. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. JMLR **15**, 1929–1958 (2014)
51. Sun, Z., Yao, A.: Weights having stable signs are important: finding primary subnetworks and kernels to compress binary weight networks (2021)
52. Tang, W., Hua, G., Wang, L.: How to train a compact binary neural network with high accuracy? In: AAAI (2017)
53. Titsias, M.K., Lázaro-Gredilla, M.: Local expectation gradients for black box variational inference. In: NeurIPS, pp. 2638–2646 (2015)
54. Tokui, S., Sato, I.: Evaluating the variance of likelihood-ratio gradient estimators. In: ICML, pp. 3414–3423 (2017)
55. Tucker, G., Mnih, A., Maddison, C.J., Lawson, J., Sohl-Dickstein, J.: REBAR: low-variance, unbiased gradient estimates for discrete latent variable models. In: NeurIPS (2017)
56. Xiang, X., Qian, Y., Yu, K.: Binary deep neural networks for speech recognition. In: INTERSPEECH (2017)
57. Yin, M., Zhou, M.: ARM: augment-REINFORCE-merge gradient for stochastic binary networks. In: ICLR (2019)
58. Yin, P., Lyu, J., Zhang, S., Osher, S., Qi, Y., Xin, J.: Understanding straight-through estimator in training activation quantized neural nets. arXiv preprint arXiv:1903.05662 (2019)
59. Zhang, S., He, N.: On the convergence rate of stochastic mirror descent for nonsmooth nonconvex optimization. arXiv, Optimization and Control (2018)
60. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: DoReFa-Net: training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160 (2016)