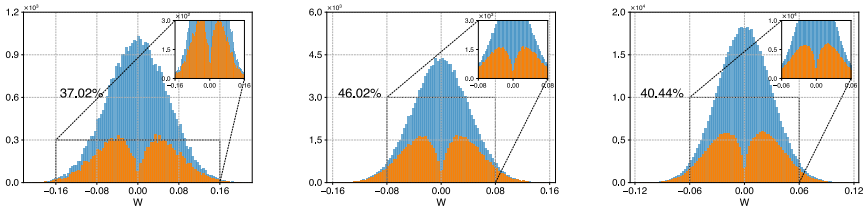
Appendix

In this Appendix we provide the following material:

- Appendix A demonstrates more weight flipping information for vanilla BNNs and OvSW in additional to Fig. 1.
- Appendix B extrapolates the conclusions in Sec. 4.1 to a more generalized scenario.
- Appendix C conducts ablation analysis for different weight initialization methods, including kaiming normal [13] and kaiming uniform [13] and the same weight initialization methods with different std or range.
- Appendix D compares AGS with LARS.
- Appendix E disscusses the difference of OvSW with latent weights (Appendix E.1) and adam optimizer (Appendix E.2) for BNNs.
- Appendix F describes societal impact of OvSW.

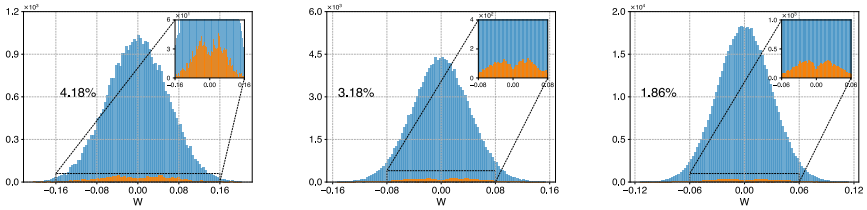
A More Experimental Results

We further demonstrate the weight flip information of the relevant layers based on Fig. 1. As seen in Fig. 5 and Fig. 6, in addition to the weight flipping of layer4.conv2.weight, OvSW also promotes weight flip efficiency for other layers.



(a) layer1.1.conv2.weight **(b)** layer2.1.conv2.weight **(c)** layer3.1.conv2.weight

Fig. 5: Histogram of the initialized weight distribution (blue) and the weights that never update signs throughout training (orange) for Vanilla BNNs. 37.02%, 46.02% and 40.44% represent the ratio of the corresponding orange area to the blue.



(a) layer1.1.conv2.weight **(b)** layer2.1.conv2.weight **(c)** layer3.1.conv2.weight

Fig. 6: Histogram of the initialized weight distribution (blue) and the weights that never update signs throughout training (orange) for OvSW. 4.18%, 3.18% and 1.86% represent the ratio of the corresponding orange area to the blue.

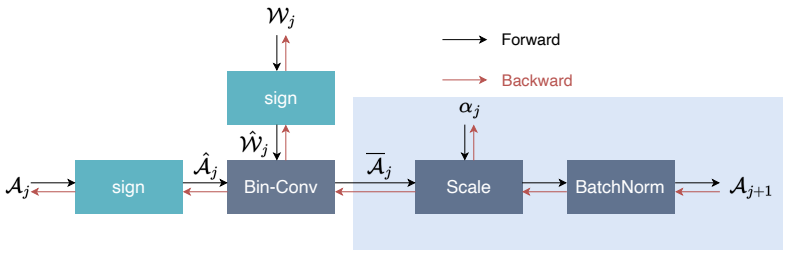


Fig. 7: Forward and backward computation graph for binarized convolutional operation with quantization aware training.

B A More Generalized Scenario

In this section, we further generalize the conclusion in Sec. 4.1 to the more general scenario, where $\mathcal{N}$ and $\mathcal{N}^{'}$ have different α_j and $\alpha_j^{'}$.

Firstly, we rewritten Eq. (3) as:

$$\begin{aligned}\mathcal{A}_{j+1} &= \mathrm{BN}\left(\left(\hat{\mathcal{A}}_j \circledast \hat{\mathcal{W}}_j\right) \odot \alpha_j\right) = \mathrm{BN}\left(\overline{\mathcal{A}}_j \odot \alpha_j\right) \\ &= \mathrm{BN}\left(\mathrm{diag}(\alpha_j)\overline{\mathcal{A}}_j\right) = \mathrm{BN}\left(\Lambda_j \overline{\mathcal{A}}_j\right).\end{aligned} \tag{17}$$

Assuming that for two networks $\mathcal{N}$ and $\mathcal{N}^{'}$, where $\mathcal{A}_j = \mathcal{A}_j^{'}$ and $\mathrm{sign}(\mathcal{W}_j) = \mathrm{sign}(\mathcal{W}_j^{'})$, it is easy to obtain that $\overline{\mathcal{A}}_j$ is equivalent to $\overline{\mathcal{A}}_j^{'}$. Therefore, we pay our attention to $\frac{\partial \mathcal{L}}{\partial \overline{\mathcal{A}}_j}$ and $\frac{\partial \mathcal{L}}{\partial \overline{\mathcal{A}}_j^{'}}$. Assuming that BN in both $\mathcal{N}$ and $\mathcal{N}^{'}$ can estimate the mean and variance of $\Lambda_j \overline{\mathcal{A}}_j$ and $\Lambda_j^{'} \overline{\mathcal{A}}_j^{'}$ with relative precision, and learnable affine parameters $\Gamma_j = \Gamma_j^{'}, \beta_j = \beta_j^{'}$ we can know for the forward propagation:

$$\begin{aligned}\mathcal{A}_{j+1} &= \mathrm{BN}\left(\Lambda_j \overline{\mathcal{A}}_j\right) \\ &\approx \Gamma_j \odot \frac{\Lambda_j \overline{\mathcal{A}}_j - \mu\left(\Lambda_j \overline{\mathcal{A}}_j\right)}{\sigma\left(\Lambda_j \overline{\mathcal{A}}_j\right)} + \beta_j \\ &\approx \Gamma_j^{'} \odot \frac{\Lambda_j^{'} \overline{\mathcal{A}}_j^{'} - \mu\left(\Lambda_j^{'} \overline{\mathcal{A}}_j^{'}\right)}{\sigma\left(\Lambda_j^{'} \overline{\mathcal{A}}_j^{'}\right)} + \beta_j^{'} \\ &= \mathrm{BN}^{'}\left(\Lambda_j^{'} \overline{\mathcal{A}}_j^{'}\right) = \mathcal{A}_{j+1}^{'},\end{aligned} \tag{18}$$

Thus, $\mathcal{N}$ and $\mathcal{N}^{'}$ will have the same output and loss, *i.e.* $\mathcal{L} = \mathcal{L}^{'}$. For the back propagation:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \overline{\mathcal{A}}_j} &= \frac{\partial \mathcal{L}}{\partial \mathcal{A}_{j+1}} \frac{\partial \mathcal{A}_{j+1}}{\partial\left(\Lambda_j \overline{\mathcal{A}}_j\right)} \frac{\partial\left(\Lambda_j \overline{\mathcal{A}}_j\right)}{\partial \overline{\mathcal{A}}_j} \approx \frac{\partial \mathcal{L}}{\partial \mathcal{A}_{j+1}} \frac{\partial \mathcal{A}_{j+1}}{\partial \overline{\mathcal{A}}_j} \Lambda_j^{-1} \Lambda_j \\ &= \frac{\partial \mathcal{L}}{\partial \mathcal{A}_{j+1}^{'}} \frac{\partial \mathcal{A}_{j+1}^{'}}{\partial \overline{\mathcal{A}}_j^{'}} \Lambda_j^{'^{-1}} \Lambda_j^{'} \approx \frac{\partial \mathcal{L}^{'}}{\partial \mathcal{A}_{j+1}^{'}} \frac{\partial \mathcal{A}_{j+1}^{'}}{\partial\left(\Lambda_j^{'} \overline{\mathcal{A}}_j^{'}\right)} \frac{\partial\left(\Lambda_j^{'} \overline{\mathcal{A}}_j^{'}\right)}{\partial \overline{\mathcal{A}}_j^{'}} = \frac{\partial \mathcal{L}^{'}}{\partial \overline{\mathcal{A}}_j^{'}}.\end{aligned} \tag{19}$$

Then we can obtain:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{A}_j} = \frac{\partial \mathcal{L}}{\partial \overline{\mathcal{A}}_j} \frac{\partial \overline{\mathcal{A}}_j}{\partial \mathcal{A}_j} = \frac{\partial \mathcal{L}^{'}}{\partial \overline{\mathcal{A}}_j^{'}} \frac{\partial \overline{\mathcal{A}}_j^{'}}{\partial \mathcal{A}_j^{'}} = \frac{\partial \mathcal{L}^{'}}{\partial \mathcal{A}_j^{'}}. \tag{20}$$

From Eq. (18), Eq. (19) and Eq. (20), we observe that due to the existence of BN, the value of α_j and $\alpha_j^{'}$ hardly affects the results of both forward and backward propagation. Thus the flipping efficiency of the weight signs is also independent of the value of α, and it is the relationship between the gradient and the weight distribution that really makes a contribution.

C Ablation Analysis for Weight Initialization

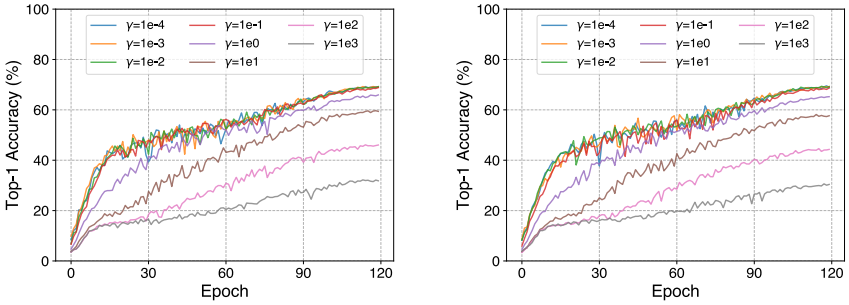
We employ OvSW for two famous weight initialization methods, including kaiming normal (default in this paper) and kaiming uniform, as shown in Eq. (21) and Eq. (22) respectively:

$$\mathcal{W}_j \sim \text{Normal}\big(0, \lambda^2 \text{std}^2\big) \tag{21}$$

$$\mathcal{W}_j \sim \text{Uniform}\big(-\lambda \text{bound}, \lambda \text{bound}\big), \tag{22}$$

where $\text{std} = \frac{\text{gain}}{\sqrt{\text{fan\_in}}}$ and $\text{bound} = \text{gain} \times \sqrt{\frac{3}{\text{fan\_in}}}$. fan_in is computed via $C_{\text{in}}^j \times K_{\text{h}}^j \times K_{\text{w}}^j$. We train binarized ResNet18 for CIFAR100 with 120 epochs to demonstrate weight initialization analysis. Without loss of generality, we set γ to 1 and initialize $\mathcal{W}_j$ via the stand kaiming initialization and then employ $\mathcal{W}_j = \gamma \mathcal{W}_j^{'}$ to simulate Eq. (21) and Eq. (22) in our implementations.

We present the epoch-wise flip rate for eight different settings of γ, which vary from 0.0001 to 1000. The results in Fig. 10 to Fig. 17 and Fig. 18 to Fig. 25 are for kaiming normal and kaiming uniform respectively. As seen, because of the gradients of the BNN being independent of their latent weight distribution, the epoch-wise flip rate and γ have a significant negative correlation. When γ is set to 1000, the epoch-wise flip rate of both distributions undergoes a severe drop, greatly hurting the model's convergence as shown in Fig. 8 and performance as shown in Fig. 9. Meanwhile, we can observe that by reducing the variance or range of the weight distribution to some extent, a significant improvement can be achieved to the BNNs compared to the standard distribution, which demonstrates the reasonableness of AGS again.



(a) Kaiming Normal **(b)** Kaiming Uniform

Fig. 8: Convergence for different weight initialization.

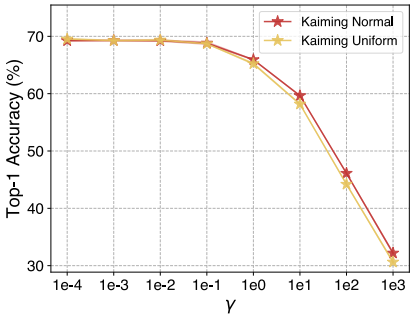
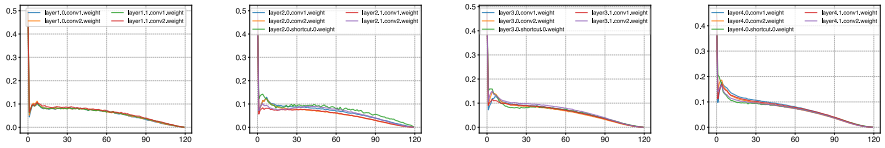
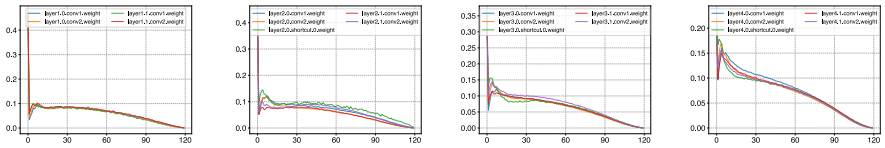


Fig. 9: Mean top-1 accuracy (mean±std) of binarized ResNet18 w.r.t. different values for different γ with different weight initialization methods on CIFAR100.



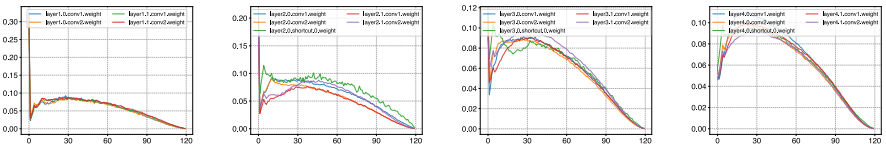
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 10: Epoch-wise flip rate for $\gamma = 0.0001$ (kaiming normal).



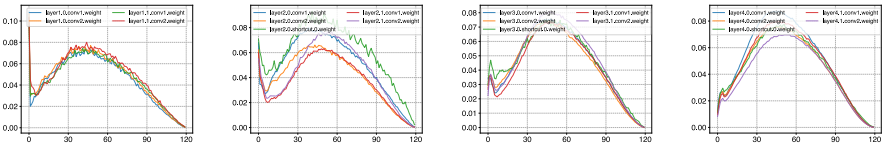
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 11: Epoch-wise flip rate for $\gamma = 0.001$ (kaiming normal).



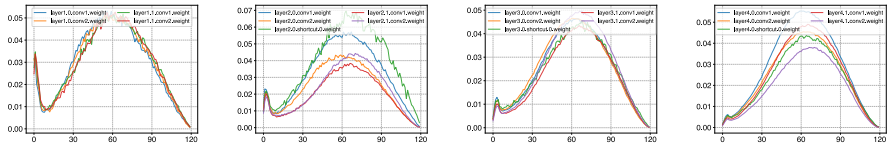
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 12: Epoch-wise flip rate for $\gamma = 0.01$ (kaiming normal).



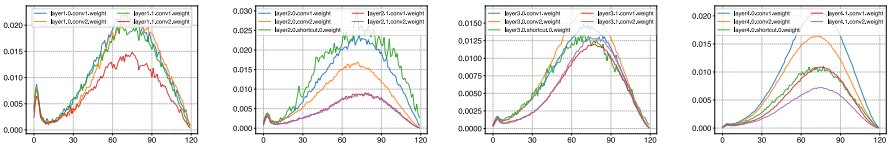
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 13: Epoch-wise flip rate for $\gamma = 0.1$ (kaiming normal).



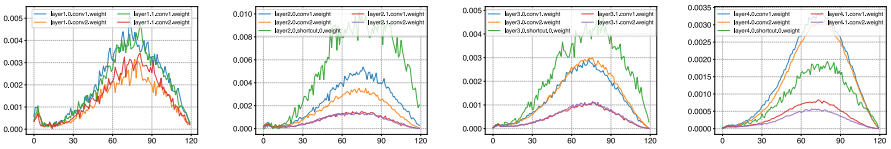
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 14: Epoch-wise flip rate for $\gamma = 1$ (kaiming normal).



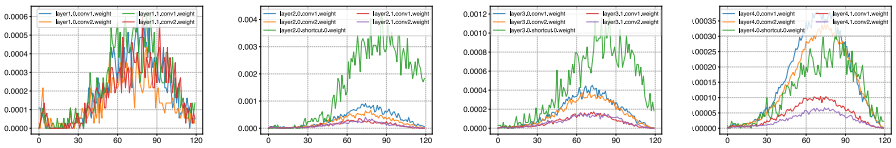
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 15: Epoch-wise flip rate for $\gamma = 10$ (kaiming normal).



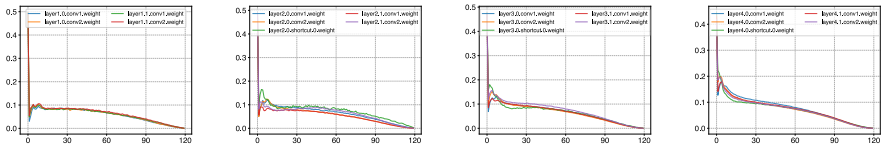
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 16: Epoch-wise flip rate for $\gamma = 100$ (kaiming normal).



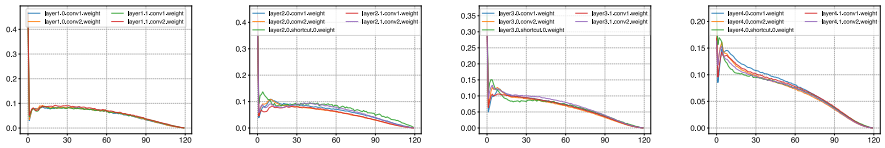
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 17: Epoch-wise flip rate for $\gamma = 1000$ (kaiming normal).



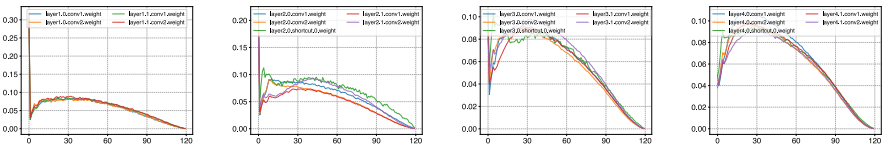
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 18: Epoch-wise flip rate for $\gamma = 0.0001$ (kaiming uniform).



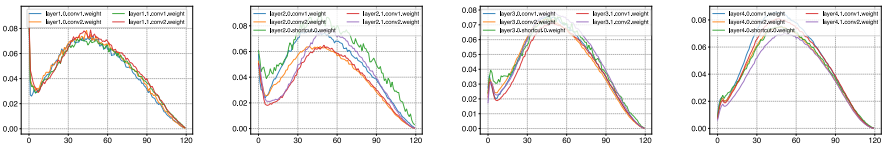
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 19: Epoch-wise flip rate for $\gamma = 0.001$ (kaiming uniform).



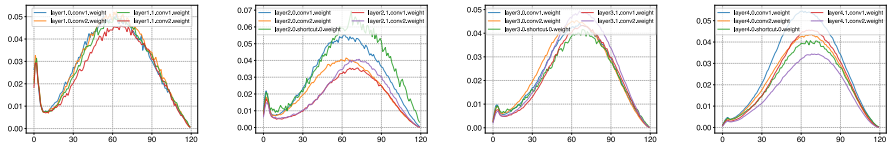
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 20: Epoch-wise flip rate for $\gamma = 0.01$ (kaiming uniform).



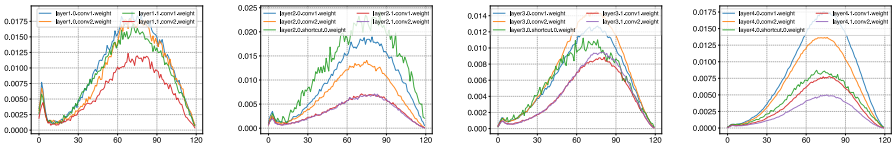
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 21: Epoch-wise flip rate for $\gamma = 0.1$ (kaiming uniform).



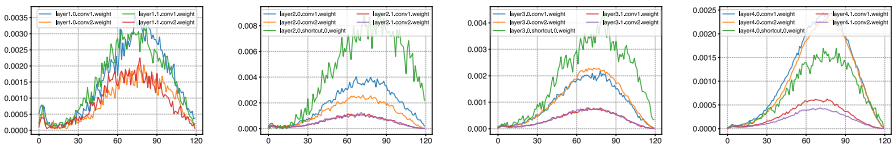
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 22: Epoch-wise flip rate for $\gamma = 1$ (kaiming uniform).



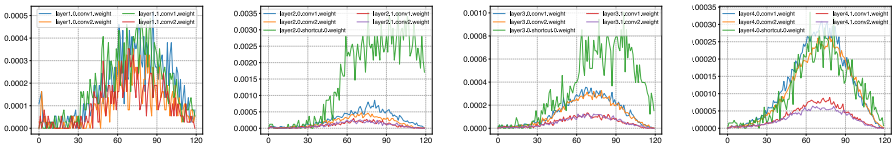
(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 23: Epoch-wise flip rate for $\gamma = 10$ (kaiming uniform).



(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 24: Epoch-wise flip rate for $\gamma = 100$ (kaiming uniform).



(a) layer1 **(b)** layer2 **(c)** layer3 **(d)** layer4

Fig. 25: Epoch-wise flip rate for $\gamma = 1000$ (kaiming uniform).

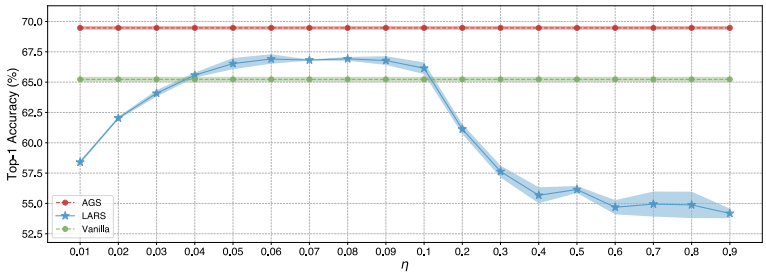


Fig. 26: Mean top-1 accuracy (mean±std) of binarized ResNet18 w.r.t. different values for different η with LARS on CIFAR100.

D AGS vs LARS

In this section, we compare LARS and AGS ($\varphi = 0$ in OvSW) and demonstrate their corresponding results in Fig. 26. As seen, by selecting the appropriate parameter η, LARS is also able to achieve impressive gains by scaling the local learning rate. However, the improvement of AGS is more significant over LARS. By comparing the optimization processes of LARS and OvSW in Algorithm 2 and Algorithm 3, We find that the most essential difference between them is that LARS only scales the learning rate at a single step, and the scaling only acts on the current gradient descent and does not accumulate into the momentum; in contrast, AGS directly modifies the gradient through adaptive gradient scaling, and the gradient not only acts on the current, but also accumulates into the future optimization process through the momentum. From the experimental results in Fig. 26, we can see that compared to LARS, AGS is more conducive to the efficient training of BNNs.

Algorithm 2 SGD with LARS. Example with weight decay and momentum.

Parameters: Learning rate $\beta(t)$, momentum m, weight decay φ, LARS coefficient η, number of steps T
Init: $t = 0, v = 0$. Init weight $\mathcal{W}_j$ for each layer
while $t < T$ for each layer **do**
  $\mathcal{G}_j(t) \leftarrow \frac{\partial \mathcal{L}(t)}{\partial \mathcal{W}_j(t)}$ (obtain a stochastic gradient for the current mini-batch)
  $\lambda_j(t) \leftarrow \frac{\eta \|\mathcal{W}_j(t)\|_F}{\|\mathcal{G}_j(t)\|_F + \varphi \|\mathcal{W}_j(t)\|_F}$ (compute the local LR $\lambda_j(t)$)
  $\mathcal{V}_j(t+1) \leftarrow m\mathcal{V}_j(t) + (\mathcal{G}_j(t) + \varphi \mathcal{W}_j(t))$ (update the momentum)
  $\mathcal{W}_j(t+1) \leftarrow \mathcal{W}_j(t) - \beta(t)\lambda_j(t)\mathcal{V}_j(t)$ (update the weights)
end while

Algorithm 3 SGD with OvSW. Example with weight decay and momentum.

Parameters: Learning rate $\beta(t)$, momentum m, weight decay φ, λ for AGS, τ for SAD, $(\mathcal{S}(t), m, \sigma)$ for flipping state detection, number of steps T
Init: $t = 0, v = 0$. Init weight $\mathcal{W}_j$ for each layer
while $t < T$ for each layer **do**
  $\mathcal{G}_j(t) \leftarrow \frac{\partial \mathcal{L}(t)}{\partial \mathcal{W}_j(t)}$ (obtain a stochastic gradient for the current mini-batch)
  $\overline{\mathcal{G}}_j(t) \leftarrow$ Eq. (14) (scale the gradient adaptively)
  $\overline{\mathcal{G}}_j(t) \leftarrow$ Eq. (16) (silence awareness decaying)
  $\mathcal{V}_j(t+1) \leftarrow m\mathcal{V}_j(t) + \left(\overline{\mathcal{G}}_j(t) + \varphi \mathcal{W}_j(t)\right)$ (update the momentum)
  $\mathcal{W}_j(t+1) \leftarrow \mathcal{W}_j(t) - \beta(t)\mathcal{V}_j(t)$ (update the weights)
  $\mathcal{S}_j(t) \leftarrow$ Eq. (15) (update state for the weights)
end while

E Discussion for Latent Weights and Optimizer

E.1 Latent Weights

We first discuss the similarities and differences between ours and Helwegen *et al*. [17]. Similarly, both of us find that the gradient of the weights in the BNNs is independent of the magnitude of the weights. Subsequently, Helwegen *et al*. design a binary optimizer (BOP), which determines the flipping of the weight signs by comparing the exponential moving average of gradients with a predefined threshold, which is independent to the magnitude of weights and gradients. While this approach avoids the problem that inappropriate weight distributions can lead to inefficient weight signs flipping, they ignore the fact that the weight magnitudes also play a role in sign changes during the optimization process. OvSW constructed a correlation between the gradient distribution and the weight distribution via AGS, which is essentially the same as BOP in facilitating weight signs flipping. Meanwhile, the optimization takes the role of weight magnitude into account and thus achieves better results than BOP. Apart from these, another advantage of OvSW is that SAD detects "silent weights" to further enhance the efficiency of weight signs flipping.

E.2 Optimizer

OvSW emploies SGD to train BNNs, which is different from the previous state-of-the-art BNNs that uses Adam [21] as the optimizer, including ReActNet [33], AdamBNN [32], RoBNN [51], ReBNN [50]. From the perspective of weight signs flipping, we believe this is due to $\widehat{m_t}/\left(\sqrt{\widehat{v_t}}+\epsilon\right)$ in Adam adaptively scales the gradients and facilitates the flipping efficiency. However, Adam needs to preserve the first momentum and second momentum of the gradient during training, leading to additional storage. In the mixed precision training scenario, a model with parameter number Ψ and Adam optimizer will consume 16Ψ [41] storage for model and optimizer. Even though OvSW introduce an auxiliary variable $\mathcal{S}$, it cause 14Ψ [41] storage, which is 12.5% less than Adam. At the same time, OvSW can be simply and effectively implemented on GPUs, improving the performance of BNN with little or no degradation of training speed.

F Societal Impact

Increasing model size can result in tremendous resource consumption and carbon emissions during both training end inference. OvSW can improve performance and accelerate the convergence efficiency of BNNs while introducing negligible memory and computational overhead. It can facilitate the deployment of BNNs. On the other hand, it also enables efficient training for BNNs under memory and computational resources constraints. Both have far-reaching potential for the promotion of green AI.