Effective Sparsification of Neural Networks with Global Sparsity Constraint

A. Appendix

In this appendix, we present additional MobileNetV1 [15] experiment on ImageNet-1K, the general experimental configurations, proof for equation 3, proof for theorem 1, analysis on the effect of temperature annealing and PyTorch code snippets of ProbMask.

A.1. MobileNetV1 on ImageNet-1K

Dataset	ImageNet		
Ratio	ProbMask	STR	GMP
89%	**65.19**	62.10	61.80
94.1%	**60.10**	23.61	-

Table 4. ProbMask surpasses state-of-the-art methods by 3.09% and 38.49% Top-1 Accuracy, demonstrating the effectiveness and generalizability of ProbMask on lightweight MobileNetV1 [15] architectures. Following the setting of [18], 89% and 94.1% sparsity is chosen to compare at the same pruning rate.

A.2. Experimental Configurations

Dataset	CIFAR	ImageNet
GPUs	1	4
Batch Size	256	256
Epochs	300	100
Weight Optimizer	SGD	SGD
Weight Learning Rate	0.1	0.256
Weight Momentum	0.9	0.875
Probability Optimizer	Adam	Adam
Probability Learning Rate	**6e-3**	**6e-3**
t_1	48	16
t_2	180	60
Warmup	✗	✓
Label Smoothing	✗	0.1

Table 5. The bold-face probability learning rate 6e-3 is the **only** hyperparameter obtained by grid search on CIFAR-10 experiments on a small size network Conv-4 [6] and applied directly to larger datasets and networks. This demonstrates the generality of our proposed ProbMask to different datasets, different networks and different tasks, i.e., pruning networks and finding supermasks. Other hyperparameters are applied following the same practice of previous works [29, 18, 22, 41]. The channels of ResNet32 for CIFAR experiments are doubled following the same practice of [33]. The temperature annealing scheme follows the same practice of [37]

A.3. Proof for equation 3

Proof. The PDF (probability density function) of $\mathrm{Gumbel}(\mu, 1)$ is

$$f(z;\mu) = e^{-(z-\mu)-e^{-(z-\mu)}}. \tag{9}$$

The CDF (cumulative distribution function) of $\mathrm{Gumbel}(\mu, 1)$ is

$$F(z;\mu) = e^{-e^{-(z-\mu)}}. \tag{10}$$

We just need to prove that

$$\forall i,\ P\left(\log(s_i) - \log(1-s_i) + g_{1,i} - g_{2,i} \geq 0\right) = s_i. \tag{11}$$

$g_{1,i}$ and $g_{2,i}$ are two $\mathrm{Gumbel}(0,1)$ random variables sampled for s_i. The probability is taken with respect to $g_{1,i}$ and $g_{2,i}$. s_i can be seen as a constant in the following proof.

Let $z_1 = \log(s_i) + g_{1,i}, z_2 = \log(1-s_i) + g_{2,i}$. Then $z_1 \sim \mathrm{Gumbel}(\log(s_i), 1)$, $z_2 \sim \mathrm{Gumbel}(\log(1-s_i), 1)$.

$$
\begin{aligned}
& P\left(\log(s_i) - \log(1-s_i) + g_{1,i} - g_{2,i} \geq 0\right) && (12)\\
=& P\left(z_2 \leq z_1\right) && (13)\\
=& \int_{-\infty}^{+\infty}\int_{-\infty}^{z_1} f(z_2; \log(1-s_i)) f(z_1; \log(s_i)) dz_2 dz_1 && (14)\\
=& \int_{-\infty}^{+\infty} F(z_1; \log(1-s_i)) f(z_1; \log(s_i)) dz_1 && (15)\\
=& \int_{-\infty}^{+\infty} e^{-e^{-(z_1 - \log(1-s_i))}} \cdot e^{-(z_1 - \log s_i) - e^{-(z_1 - \log s_i)}} dz_1 && (16)\\
=& \int_{-\infty}^{+\infty} e^{-e^{-z_1}(1-s_i) - z_1 + \log s_i - e^{-z_1} s_i} dz_1 && (17)\\
=& s_i \int_{-\infty}^{+\infty} e^{-e^{-z_1} - z_1} dz_1 && (18)\\
=& s_i && (19)
\end{aligned}
$$

$\int_{-\infty}^{+\infty} e^{-e^{-z_1}-z_1} dz_1$ is the integral of a Gumbel(0,1) random variable. □

A.4. Proof for Theorem 1

Proof. The projection from $\boldsymbol{z}$ to set C can be formulated in the following optimization problem:

$$\min_{\boldsymbol{s} \in \mathbb{R}^n} \frac{1}{2}\|\boldsymbol{s} - \boldsymbol{z}\|^2,$$
$$s.t. \mathbf{1}^\top \boldsymbol{s} \leq K \text{ and } 0 \leq \boldsymbol{s}_i \leq 1.$$

Then we solve the problem with Lagrangian multiplier method.

$$L(\boldsymbol{s}, v) = \frac{1}{2}\|\boldsymbol{s} - \boldsymbol{z}\|^2 + v(\mathbf{1}^\top \boldsymbol{s} - K) \tag{20}$$
$$= \frac{1}{2}\|\boldsymbol{s} - (\boldsymbol{z} - v\mathbf{1})\|^2 + v(\mathbf{1}^\top \boldsymbol{z} - K) - \frac{n}{2}v^2. \tag{21}$$

with $v \geq 0$ and $0 \leq \boldsymbol{s}_i \leq 1$. Minimize the problem with respect to $\boldsymbol{s}$, we have

$$\tilde{\boldsymbol{s}} = \mathbf{1}_{\boldsymbol{z}-v\mathbf{1}\geq 1} + (\boldsymbol{z} - v\mathbf{1})_{1>\boldsymbol{z}-v\mathbf{1}>0} \tag{22}$$

Then we have

$$\begin{aligned} g(v) =& L(\tilde{\boldsymbol{s}}, v) \\ =& \frac{1}{2}\|[\boldsymbol{z} - v\mathbf{1}]_- + [\boldsymbol{z} - (v+1)\mathbf{1}]_+\|^2 \\ &+ v(\mathbf{1}^\top \boldsymbol{z} - s) - \frac{n}{2}v^2 \\ =& \frac{1}{2}\|[\boldsymbol{z} - v\mathbf{1}]_-\|^2 + \frac{1}{2}\|[\boldsymbol{z} - (v+1)\mathbf{1}]_+\|^2 \\ &+ v(\mathbf{1}^\top \boldsymbol{z} - s) - \frac{n}{2}v^2, v \geq 0. \\ g'(v) =& \mathbf{1}^\top [v\mathbf{1} - \boldsymbol{z}]_+ + \mathbf{1}^\top [(v+1)\mathbf{1} - \boldsymbol{z}]_- \\ &+ (1^T \boldsymbol{z} - s) - nv \\ =& \mathbf{1}^\top \min(1, \max(0, \boldsymbol{z} - v\mathbf{1})) - K, v \geq 0. \end{aligned}$$

It is easy to verify that $g'(v)$ is a monotone decreasing function with respect to v and we can use a bisection method solve the equation $g'(v) = 0$ with solution v_1^*. Then we get that $g(v)$ increases in the range of $(-\infty, v_1^*]$ and decreases in the range of $[v_1^*, +\infty)$. The maximum of g(v) is achieved at 0 if $v_1^* \leq 0$ and v_1^* if $v_1^* > 0$. Then we set $v_2^* = max(0, v_1^*)$. Finally we have

$$\boldsymbol{s}^* = \mathbf{1}_{\boldsymbol{z}-v_2^*\mathbf{1}\geq 1} + (\boldsymbol{z} - v_2^*\mathbf{1})_{1>\boldsymbol{z}-v_2^*\mathbf{1}>0} \tag{23}$$
$$= \min(1, \max(0, \boldsymbol{z} - v_2^*\mathbf{1})). \tag{24}$$

□

A.5. Temperature Annealing

Thanks to the ℓ_1 norm and cube $[0,1]^n$ in our constraint, most probabilities will converge to 0 or 1 at the end of training, which is shown in

$$\boldsymbol{s} = \min(1, \max(0, \boldsymbol{z} - v_2^*\mathbf{1})).$$

Traditional temperature annealing starts with a relative high value, i.e., 1 to have a smooth relaxation and gradually decrease to a small value to make relaxation close to the original objective function. In this section we analyze how the temperature annealing contributes to the training process, especially helping probabilities converge to 0 or 1.

Firstly consider the gradient:

$$\nabla_{s_i}\mathcal{L}\left(\boldsymbol{w}, \sigma\Big(\frac{\log(\frac{\boldsymbol{s}}{\mathbf{1}-\boldsymbol{s}}) + \boldsymbol{g_1} - \boldsymbol{g_0}}{\tau}\Big)\right) \tag{25}$$
$$= \nabla_{\sigma_i}\mathcal{L}\left(\boldsymbol{w}, \sigma\Big(\frac{\log(\frac{\boldsymbol{s}}{\mathbf{1}-\boldsymbol{s}}) + \boldsymbol{g_1} - \boldsymbol{g_0}}{\tau}\Big)\right) S, \tag{26}$$

where $S = \nabla_{s_i}\sigma(\frac{\log(\frac{s_i}{1-s_i}) + g_{1,i} - g_{0,i}}{\tau})$. We can see that the larger the magnitude $|S|$, z_i (step 8 in Algorithm 1) will vary more greatly.

Take $x = \frac{1}{\tau} \in [1, +\infty)$, $r = log(s_i) - log(1 - s_i) + g_{1,i} - g_{0,i}$. We have

$$S = \frac{\sigma(rx)(1 - \sigma(rx))x}{s_i(1 - s_i)} \tag{27}$$

Since S is an even function w.r.t r, we just consider the case $r > 0$. Then we take the gradient w.r.t to x.

$$\nabla_x \left(\sigma(rx)(1 - \sigma(rx))x\right) \tag{28}$$
$$= \sigma(rx)(1 - \sigma(rx))(rx - 2rx\sigma(rx) + 1) \tag{29}$$

Take $y = rx, y > 0$ since $x > 0$ and $r > 0$. The solution to $y - 2y\sigma(y) + 1 = 0$ is around 1.55. S is a monotonically increasing function for $x \in (0, \frac{1.55}{r}]$ and monotonically decreasing function for $x \in [\frac{1.55}{r}, +\infty)$.

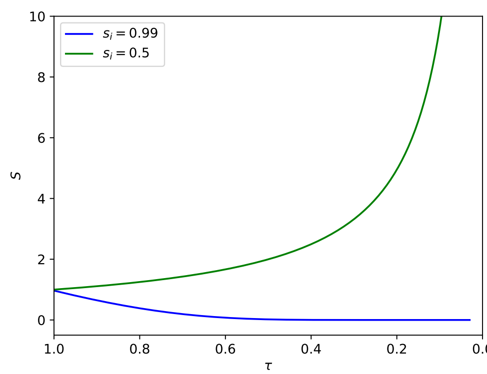

Figure 8. The value of S changes with τ approching zero.

Now we analyze S using two special cases. Take $g = g_{1,i} - g_{2,i}$, $g \sim Logistic(0, 1)$. Take $s_i = 0.99$ and $g = 0.04$ for example. $r = \log(99) + 0.1 \approx 4.63$. S is monotonically decreasing function for $x \in [1, +\infty)$. Take $s_i = 0.5$ and $g = 0.04$ for example. S would increase as τ decreases to 0.03, since we set $\tau = 0.97(1 - t/T) + 0.03$. We plot the corresponding graph in Figure 8.

From the above two examples, we know that for probabilities around 0.5, S becomes larger in the training process, potentially making $|z_i|$ large and finally make probability come close to 0 or 1 after projection. For probabilities close to 0 or 1, S becomes smaller in the training process, making them stay close to 0 or 1 at the end of training.

```python
class ProbMaskConv(nn.Conv2d):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.scores = nn.Parameter(torch.Tensor(self.weight.size()))     # Probability
        self.subnet = None                                              # Mask
        self.scores.data = (torch.ones_like(self.scores)* parser_args.score_init_constant)

    def forward(self, x):                  # Sample a mask and forward propagation
        if not parser_args.discrete:       # Training
            eps = 1e-20
            temp = parser_args.T
            uniform0 = torch.rand_like(self.scores)
            uniform1 = torch.rand_like(self.scores)
            noise = -torch.log(torch.log(uniform0 + eps) / torch.log(uniform1 + eps) + eps)
            self.subnet = torch.sigmoid((torch.log(self.scores + eps) - torch.log(1.0 - self.scores + eps) + noise) * temp)
        else:                              # Testing
            self.subnet = (torch.rand_like(self.scores) < self.scores).float()
        w = self.weight * self.subnet
        x = F.conv2d(x, w, self.bias, self.stride, self.padding, self.dilation, self.groups)
        return x
```

Listing 1. PyTorch Code Snippets for ProbMaskConv.

```python
def constrainScoreByWhole(model):
    total = 0
    for n, m in model.named_modules():
        if hasattr(m, "scores"):
            total += m.scores.nelement()
    v = solveV(model, total)                 # Calculate v_2^* in Theorem 1
    for n, m in model.named_modules():
        if hasattr(m, "scores"):
            m.scores.sub_(v).clamp_(0, 1)    # Do the projection

def solveV(model, total):                    # Solve solution to Equation 7 with bisection search
    k = total * parser_args.prune_rate
    a, b = 0, 0
    for n, m in model.named_modules():
        if hasattr(m, "scores"):
            b = max(b, m.scores.max())
    def f(v):
        s = 0
        for n, m in model.named_modules():
            if hasattr(m, "scores"):
                s += (m.scores - v).clamp(0, 1).sum()
        return s - k
    if f(0) < 0:
        return 0
    itr = 0
    while (1):
        itr += 1
        v = (a + b) / 2
        obj = f(v)
        if abs(obj) < 1e-3 or itr > 20:
            break
        if obj < 0:
            b = v
        else:
            a = v
    v = max(0, v)
    return v
```

Listing 2. PyTorch Code Snippets for Projection in Theorem 1