Proximal Mean-field for Neural Network Quantization Supplementary Material

Thalaiyasingam Ajanthan*[1], Puneet K. Dokania[2], Richard Hartley[1], and Philip H. S. Torr[2]

[1]Australian National University [2]University of Oxford

Here, we provide the proofs of propositions and theorems stated in the main paper and a self-contained overview of the mean-field method. Later in Sec. C, we give the experimental details to allow reproducibility, and more empirical analysis for our Proximal Mean-Field (PMF) algorithm.

A. Mean-field Method

For completeness we briefly review the underlying theory of the mean-field method. For in-depth details, we refer the interested reader to the Chapter 5 of [7]. Furthermore, for background on Markov Random Field (MRF), we refer the reader to the Chapter 2 of [1]. In this section, we use the notations from the main paper and highlight the similarities wherever possible.

Markov Random Field. Let $\mathcal{W} = \{W_1, \dots, W_m\}$ be a set of random variables, where each random variable W_j takes a label $w_j \in \mathcal{Q}$. For a given labelling $\mathbf{w} \in \mathcal{Q}^m$, the energy associated with an Markov Random Field (MRF) can be written as:

$$L(\mathbf{w}) = \sum_{C \in \mathcal{C}} L_C(\mathbf{w}) \,, \tag{19}$$

where $\mathcal{C}$ is the set of subsets (cliques) of $\mathcal{W}$ and $L_C(\mathbf{w})$ is a positive function (factor or clique potential) that depends only on the values w_j for $j \in C$. Now, the joint probability distribution over the random variables can be written as:

$$P(\mathbf{w}) = \frac{1}{Z} \exp(-L(\mathbf{w})) \,, \tag{20}$$

where the normalization constant Z is usually referred to as the partition function. From Hammersley-Clifford theorem, for the factorization given in Eq. (19), the joint probability distribution $P(\mathbf{w})$ can be shown to factorize over each clique $C \in \mathcal{C}$, which is essentially the Markov property. However, this Markov property is not necessary to write Eq. (20) and in turn for our formulation, but since mean-field is usually described in the context of MRFs we provide it here for completeness. The objective of mean-field is to obtain the most probable configuration, which is equivalent to minimizing the energy $L(\mathbf{w})$.

Mean-field Inference. The basic idea behind mean-field is to approximate the intractable probability distribution $P(\mathbf{w})$ with a tractable one. Specifically, mean-field obtains a fully-factorized distribution (*i.e.*, each random variable W_j is independent) closest to the true distribution $P(\mathbf{w})$ in terms of KL-divergence. Let $U(\mathbf{w}) = \prod_{j=1}^{m} U_j(w_j)$ denote a fully-factorized distribution. Recall, the variables $\mathbf{u}$ introduced in Sec. 2.2 represent the probability of each weight W_j taking a label q_λ. Therefore, the distribution U can be represented using the variables $\mathbf{u} \in \Delta^m$, where Δ^m is defined as:

$$\Delta^m = \left\{ \mathbf{u} \;\middle|\; \begin{array}{ll} \sum_\lambda u_{j:\lambda} = 1, & \forall\, j \\ u_{j:\lambda} \geq 0, & \forall\, j, \lambda \end{array} \right\} . \tag{21}$$

The KL-divergence between U and P can be written as:

$$\begin{aligned} \mathrm{KL(U\|P)} &= \sum_{\mathbf{w} \in \mathcal{Q}^m} U(\mathbf{w}) \log \frac{U(\mathbf{w})}{P(\mathbf{w})} \,, \qquad (22) \\ &= \sum_{\mathbf{w} \in \mathcal{Q}^m} U(\mathbf{w}) \log U(\mathbf{w}) - \sum_{\mathbf{w} \in \mathcal{Q}^m} U(\mathbf{w}) \log P(\mathbf{w}) \,, \\ &= -H(U) - \sum_{\mathbf{w} \in \mathcal{Q}^m} U(\mathbf{w}) \log \frac{\exp(-L(\mathbf{w}))}{Z} \,, \quad \text{Eq. (20)} \,, \\ &= -H(U) + \sum_{\mathbf{w} \in \mathcal{Q}^m} U(\mathbf{w}) L(\mathbf{w}) + \log Z \,. \end{aligned}$$

Here, $H(U)$ denotes the entropy of the fully-factorized distribution. Specifically,

$$H(U) = H(\mathbf{u}) = -\sum_{j=1}^{m} \sum_{\lambda=1}^{d} u_{j:\lambda} \log u_{j:\lambda} \,. \tag{23}$$

*Part of the work was done while at the University of Oxford.

Furthermore, in Eq. (22), since Z is a constant, it can be removed from the minimization. Hence the final mean-field objective can be written as:

$$\min_U F(U) := \sum_{\mathbf{w}\in\mathcal{Q}^m} U(\mathbf{w})L(\mathbf{w}) - H(U)\;, \qquad (24)$$
$$= \mathbb{E}_U[L(\mathbf{w})] - H(U)\;,$$

where $\mathbb{E}_U[L(\mathbf{w})]$ denotes the expected value of the loss $L(\mathbf{w})$ over the distribution $U(\mathbf{w})$. Note that, the expected value of the loss can be written as a function of the variables $\mathbf{u}$. In particular,

$$E(\mathbf{u}) := \mathbb{E}_U[L(\mathbf{w})] = \sum_{\mathbf{w}\in\mathcal{Q}^m} U(\mathbf{w})L(\mathbf{w})\;, \qquad (25)$$
$$= \sum_{\mathbf{w}\in\mathcal{Q}^m} \prod_{j=1}^{m} u_{j:w_j} L(\mathbf{w})\;.$$

Now, the mean-field objective can be written as an optimization over $\mathbf{u}$:

$$\min_{\mathbf{u}\in\Delta^m} F(\mathbf{u}) := E(\mathbf{u}) - H(\mathbf{u})\;. \qquad (26)$$

Computing this expectation $E(\mathbf{u})$ in general is intractable as the sum is over an exponential number of elements ($|\mathcal{Q}|^m$ elements, where m is usually in the order millions for an image or a neural network). However, for an MRF, the energy function $L(\mathbf{w})$ can be factorized easily as in Eq. (19) (*e.g.*, unary and pairwise terms) and $E(\mathbf{u})$ can be computed fairly easily as the distribution U is also fully-factorized.

In mean-field, the above objective (26) is minimized iteratively using a fixed point update. This update is derived by writing the Lagrangian and setting the derivatives with respect to $\mathbf{u}$ to zero. At iteration k, the mean-field update for each $j \in \{1,\dots,m\}$ can be written as:

$$u^{k+1}_{j:\lambda} = \frac{\exp(-\partial E^k/\partial u_{j:\lambda})}{\sum_\mu \exp(-\partial E^k/\partial u_{j:\mu})} \quad \forall\,\lambda\in\{1,\dots,d\}\;. \qquad (27)$$

Here, $\partial E^k/\partial u_{j:\lambda}$ denotes the gradient of $E(\mathbf{u})$ with respect to $u_{j:\lambda}$ evaluated at $u^k_{j:\lambda}$. This update is repeated until convergence. Once the distribution U is obtained, finding the most probable configuration is straight forward, since U is a product of independent distributions over each random variable W_j. Note that, as most probable configuration is exactly the minimum label configuration, the mean-field method iteratively minimizes the actual energy function $L(\mathbf{w})$.

B. BinaryConnect as Proximal ICM

Proposition B.1. Consider BinaryConnect (BC) and Proximal Iterative Conditional Modes (PICM) with $\mathbf{q} = [-1, 1]^T$ and $\eta_{\mathbf{w}} > 0$. For an iteration $k > 0$, if $\tilde{\mathbf{w}}^k = \tilde{\mathbf{u}}^k\mathbf{q}$ then,

1. the projections in BC: $\mathbf{w}^k = \operatorname{sign}(\tilde{\mathbf{w}}^k)$ and PICM: $\mathbf{u}^k = \operatorname{hardmax}(\tilde{\mathbf{u}}^k)$ satisfy $\mathbf{w}^k = \mathbf{u}^k\mathbf{q}$.

2. let the learning rate of PICM be $\eta_{\mathbf{u}} = \eta_{\mathbf{w}}/2$, then the updated points after the gradient descent step in BC and PICM satisfy $\tilde{\mathbf{w}}^{k+1} = \tilde{\mathbf{u}}^{k+1}\mathbf{q}$.

Proof. 1. In the binary case ($\mathcal{Q} = \{-1, 1\}$), for each $j \in \{1,\dots,m\}$, the hardmax projection can be written as:

$$u^k_{j:-1} = \begin{cases} 1 & \text{if } \tilde{u}^k_{j:-1} \geq \tilde{u}^k_{j:1} \\ 0 & \text{otherwise} \end{cases}\;,$$
$$u^k_{j:1} = 1 - u^k_{j:-1}\;. \qquad (28)$$

Now, multiplying both sides by $\mathbf{q}$, and substituting $\tilde{w}^k_j = \tilde{\mathbf{u}}^k_j\mathbf{q}$,

$$\mathbf{u}^k_j\mathbf{q} = \begin{cases} -1 & \text{if } \tilde{w}^k_j = -1\,\tilde{u}^k_{j:-1} + 1\,\tilde{u}^k_{j:1} \leq 0 \\ 1 & \text{otherwise} \end{cases}\;,$$
$$w^k_j = \operatorname{sign}(\tilde{w}^k_j)\;. \qquad (29)$$

Hence, $\mathbf{w}^k = \operatorname{sign}(\tilde{\mathbf{w}}^k) = \operatorname{hardmax}(\tilde{\mathbf{u}}^k)\mathbf{q}$.

2. Since $\mathbf{w}^k = \mathbf{u}^k\mathbf{q}$ from case (1) above, by chain rule the gradients $\mathbf{g}^k_{\mathbf{w}}$ and $\mathbf{g}^k_{\mathbf{u}}$ satisfy,

$$\mathbf{g}^k_{\mathbf{u}} = \mathbf{g}^k_{\mathbf{w}}\,\frac{\partial \mathbf{w}}{\partial \mathbf{u}} = \mathbf{g}^k_{\mathbf{w}}\,\mathbf{q}^T\;. \qquad (30)$$

Similarly, from case (1) above, for each $j \in \{1,\dots,m\}$,

$$w^k_j = \operatorname{sign}(\tilde{w}^k_j) = \operatorname{sign}(\tilde{\mathbf{u}}^k_j\,\mathbf{q}) = \operatorname{hardmax}(\tilde{\mathbf{u}}^k_j)\,\mathbf{q}\;,$$
$$\frac{\partial w_j}{\partial \tilde{\mathbf{u}}_j} = \frac{\partial \operatorname{sign}}{\partial \tilde{\mathbf{u}}_j} = \frac{\partial \operatorname{sign}}{\partial \tilde{w}_j}\frac{\partial \tilde{w}_j}{\partial \tilde{\mathbf{u}}_j} = \frac{\partial \operatorname{hardmax}}{\partial \tilde{\mathbf{u}}_j}\,\mathbf{q}\;. \qquad (31)$$

Here, the partial derivatives are evaluated at $\tilde{\mathbf{u}} = \tilde{\mathbf{u}}^k$ but omitted for notational clarity. Moreover, $\frac{\partial w_j}{\partial \tilde{\mathbf{u}}_j}$ is a d-dimensional column vector, $\frac{\partial \operatorname{sign}}{\partial \tilde{w}_j}$ is a scalar, and $\frac{\partial \operatorname{hardmax}}{\partial \tilde{\mathbf{u}}_j}$ is a $d \times d$ matrix. Since, $\frac{\partial \tilde{w}_j}{\partial \tilde{\mathbf{u}}_j} = \mathbf{q}$ (similar to Eq. (30)),

$$\frac{\partial w_j}{\partial \tilde{\mathbf{u}}_j} = \frac{\partial \operatorname{sign}}{\partial \tilde{w}_j}\,\mathbf{q} = \frac{\partial \operatorname{hardmax}}{\partial \tilde{\mathbf{u}}_j}\,\mathbf{q}\;. \qquad (32)$$

Now, consider the $\mathbf{g}^k_{\tilde{\mathbf{u}}}$ for each $j \in \{1, \dots, m\}$,

$$\mathbf{g}^k_{\tilde{\mathbf{u}}_j} = \mathbf{g}^k_{\mathbf{u}_j} \frac{\partial \mathbf{u}_j}{\partial \tilde{\mathbf{u}}_j} = \mathbf{g}^k_{\mathbf{u}_j} \frac{\partial \,\text{hardmax}}{\partial \tilde{\mathbf{u}}_j} , \tag{33}$$

$$\begin{aligned}
\mathbf{g}^k_{\tilde{\mathbf{u}}_j}\, \mathbf{q} &= \mathbf{g}^k_{\mathbf{u}_j} \frac{\partial \,\text{hardmax}}{\partial \tilde{\mathbf{u}}_j} \mathbf{q} , \quad \text{multiplying by } \mathbf{q} , \\
&= g^k_{w_j}\, \mathbf{q}^T \frac{\partial \,\text{hardmax}}{\partial \tilde{\mathbf{u}}_j} \mathbf{q} , \quad \text{Eq. (30)} , \\
&= g^k_{w_j}\, \mathbf{q}^T \frac{\partial \,\text{sign}}{\partial \tilde{w}_j} \mathbf{q} , \quad \text{Eq. (32)} , \\
&= g^k_{w_j} \frac{\partial \,\text{sign}}{\partial \tilde{w}_j} \mathbf{q}^T \mathbf{q} , \\
&= g^k_{\tilde{w}_j}\, \mathbf{q}^T \mathbf{q} , \quad \tfrac{\partial \,\text{sign}}{\partial \tilde{w}_j} = \tfrac{\partial w_j}{\partial \tilde{w}_j} , \\
&= 2\, g^k_{\tilde{w}_j} , \quad \mathbf{q} = [-1, 1]^T .
\end{aligned}$$

Now, consider the gradient descent step for $\tilde{\mathbf{u}}$, with $\eta_{\mathbf{u}} = \eta_{\mathbf{w}}/2$,

$$\begin{aligned}
\tilde{\mathbf{u}}^{k+1} &= \tilde{\mathbf{u}}^k - \eta_{\mathbf{u}}\, \mathbf{g}^k_{\tilde{\mathbf{u}}} , \\
\tilde{\mathbf{u}}^{k+1}\, \mathbf{q} &= \tilde{\mathbf{u}}^k\, \mathbf{q} - \eta_{\mathbf{u}}\, \mathbf{g}^k_{\tilde{\mathbf{u}}}\, \mathbf{q} , \\
&= \tilde{\mathbf{w}}^k - \eta_{\mathbf{u}}\, 2\, \mathbf{g}^k_{\tilde{\mathbf{w}}} , \\
&= \tilde{\mathbf{w}}^k - \eta_{\mathbf{w}}\, \mathbf{g}^k_{\tilde{\mathbf{w}}} , \\
&= \tilde{\mathbf{w}}^{k+1} .
\end{aligned} \tag{34}$$

Hence, the proof is complete. □

Note that, in the implementation of BC, the auxiliary variables $\tilde{\mathbf{w}}$ are clipped between $[-1, 1]$ as it does not affect the sign function. In the $\mathbf{u}$-space, this clipping operation would translate into a projection to the polytope Δ^m, meaning $\tilde{\mathbf{w}} \in [-1, 1]$ implies $\tilde{\mathbf{u}} \in \Delta^m$, where $\tilde{\mathbf{w}}$ and $\tilde{\mathbf{u}}$ are related according to $\tilde{\mathbf{w}} = \tilde{\mathbf{u}}\mathbf{q}$. Even in this case, Proposition B.1 holds, as the assumption $\tilde{\mathbf{w}}^k = \tilde{\mathbf{u}}^k\mathbf{q}$ is still satisfied.

B.1. Approximate Gradients through Hardmax

In previous works [3, 6], to allow back propagation through the sign function, the straight-through-estimator [4] is used. Precisely, the partial derivative with respect to the sign function is defined as:

$$\frac{\partial \,\text{sign}(r)}{\partial r} := \mathbb{1}[|r| \leq 1] . \tag{35}$$

To make use of this, we intend to write the projection function hardmax in terms of the sign function. To this end, from Eq. (28), for each $j \in \{1, \dots, m\}$,

$$u^k_{j:-1} = \begin{cases} 1 & \text{if } \tilde{u}^k_{j:-1} - \tilde{u}^k_{j:1} \geq 0 \\ 0 & \text{otherwise} \end{cases} , \tag{36}$$

$$u^k_{j:1} = 1 - u^k_{j:-1} . \tag{37}$$

Dataset	Architecture	PMF wo $\tilde{\mathbf{u}}$	PMF
MNIST	LeNet-300	96.74	**98.24**
	LeNet-5	98.78	**99.44**
CIFAR-10	VGG-16	80.18	**90.51**
	ResNet-18	87.36	**92.73**

Table 3: *Comparison of* PMF *with and without storing the auxiliary variables* $\tilde{\mathbf{u}}$*. Storing the auxiliary variables and updating them is in fact improves the overall performance. However, even without storing* $\tilde{\mathbf{u}}$*,* PMF *obtains reasonable performance, indicating the usefulness of our relaxation.*

Hence, the projection $\text{hardmax}(\tilde{\mathbf{u}}^k)$ for each j can be written as:

$$u^k_{j:-1} = (\text{sign}(\tilde{u}^k_{j:-1} - \tilde{u}^k_{j:1}) + 1)/2 , \tag{38}$$

$$u^k_{j:1} = (1 - \text{sign}(\tilde{u}^k_{j:-1} - \tilde{u}^k_{j:1}))/2 . \tag{39}$$

Now, using Eq. (35), we can write:

$$\frac{\partial \mathbf{u}_j}{\partial \tilde{\mathbf{u}}_j}\bigg|_{\tilde{\mathbf{u}}_j = \tilde{\mathbf{u}}^k_j} = \frac{1}{2} \begin{bmatrix} \mathbb{1}[|v^k_j| \leq 1] & -\mathbb{1}[|v^k_j| \leq 1] \\ -\mathbb{1}[|v^k_j| \leq 1] & \mathbb{1}[|v^k_j| \leq 1] \end{bmatrix} , \tag{40}$$

where $v^k_j = \tilde{u}^k_{j:-1} - \tilde{u}^k_{j:1}$.

C. Experimental Details

To enable reproducibility, we first give the hyperparameter settings used to obtain the results reported in the main paper in Table 4.

C.1. Proximal Mean-field Analysis

To analyse the effect of storing the auxiliary variables $\tilde{\mathbf{u}}$ in Algorithm 1, we evaluate PMF with and without storing $\tilde{\mathbf{u}}$, meaning the variables $\mathbf{u}$ are updated directly. The results are reported in Table 3. Storing the auxiliary variables and updating them is in fact improves the overall performance. However, even without storing $\tilde{\mathbf{u}}$, PMF obtains reasonable performance, indicating the usefulness of our continuous relaxation. Note that, if the auxiliary variables are not stored in BC, it is impossible to train the network as the quantization error in the gradients are catastrophic and single gradient step is not sufficient to move from one discrete point to the next.

C.2. Effect of Annealing Hyperparameter β

In Algorithm 1, the annealing hyperparameter is gradually increased by a multiplicative scheme. Precisely, β is updated according to $\beta = \rho\beta$ for some $\rho > 1$. Such a multiplicative continuation is a simple scheme suggested in Chapter 17 of [5] for penalty methods. To examine the

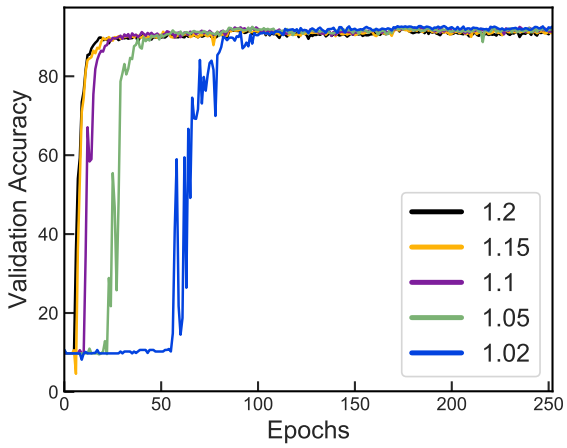

Figure 3: PMF *results on CIFAR10 with ResNet-18 by varying* ρ *values. While* PMF *is robust to a range of* ρ *values, the longer exploration phase in lower values of* ρ *tend to yield slightly better final accuracies.*

sensitivity of the continuation parameter ρ, we report the behaviour of PMF on CIFAR-10 with ResNet-18 for various values of ρ in Fig. 3.

C.3. Multi-bit Quantization

To test the performance of PMF for quantization levels beyond binary, we ran PMF for 2-bit quantization with $\mathcal{Q} = \{2, 1, 1, 2\}$ on CIFAR-10 with the same hyperparameters as in the binary case and the results are, ResNet-18: 92.88% and VGG-16: 91.27%, respectively. We believe, the improvements over binary ($+0.15\%$ and $+0.76\%$) even without hyperparameter tuning shows the merits of PMF for NN quantization. Note, similar to existing methods [2], we can also obtain different quantization levels for each weight w_j, which would further improve the performance.

References

[1] Thalaiyasingam Ajanthan. *Optimization of Markov random fields in computer vision*. PhD thesis, Australian National University, 2017. 1

[2] Yu Bai, Yu-Xiang Wang, and Edo Liberty. Proxquant: Quantized neural networks via proximal operators. *ICLR*, 2019. 4

[3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *NIPS*, 2015. 3

[4] Geoffrey Hinton. Neural networks for machine learning. *Coursera, video lectures*, 2012. 3

[5] Jorge Nocedal and Stephen Wright. Numerical optimization. *Springer*, 2006. 3

[6] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *ECCV*, 2016. 3

[7] Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 2008. 1

Hyperparameter	MNIST with LeNet-300/5					TinyImageNet with ResNet-18				
	REF	BC/PICM	PQ	PGD	PMF	REF	BC/PICM	PQ	PGD	PMF
learning_rate	0.001	0.001	0.01	0.001	0.001	0.1	0.0001	0.01	0.1	0.001
lr_decay	step	step	-	step	step	step	step	step	step	step
lr_interval	7k	7k	-	7k	7k	60k	30k	30k	30k	30k
lr_scale	0.2	0.2	-	0.2	0.2	0.2	0.2	0.2	0.2	0.2
momentum	-	-	-	-	-	0.9	-	-	0.95	-
optimizer	Adam	Adam	Adam	Adam	Adam	SGD	Adam	Adam	SGD	Adam
weight_decay	0	0	0	0	0	0.0001	0.0001	0.0001	0.0001	0.0001
ρ (ours) or reg_rate (PQ)	-	-	0.001	1.2	1.2	-	-	0.0001	1.01	1.02
	CIFAR-10 with VGG-16					CIFAR-10 with ResNet-18				
	REF	BC/PICM	PQ	PGD	PMF	REF	BC/PICM	PQ	PGD	PMF
learning_rate	0.1	0.0001	0.01	0.0001	0.001	0.1	0.0001	0.01	0.1	0.001
lr_decay	step	step	-	step	step	step	step	-	step	step
lr_interval	30k	30k	-	30k	30k	30k	30k	-	30k	30k
lr_scale	0.2	0.2	-	0.2	0.2	0.2	0.2	-	0.2	0.2
momentum	0.9	-	-	-	-	0.9	-	-	0.9	-
optimizer	SGD	Adam	Adam	Adam	Adam	SGD	Adam	Adam	SGD	Adam
weight_decay	0.0005	0.0001	0.0001	0.0001	0.0001	0.0005	0.0001	0.0001	0.0001	0.0001
ρ (ours) or reg_rate (PQ)	-	-	0.0001	1.05	1.05	-	-	0.0001	1.01	1.02
	CIFAR-100 with VGG-16					CIFAR-100 with ResNet-18				
	REF	BC/PICM	PQ	PGD	PMF	REF	BC/PICM	PQ	PGD	PMF
learning_rate	0.1	0.01	0.01	0.0001	0.0001	0.1	0.0001	0.01	0.1	0.001
lr_decay	step	multi-step	-	step	step	step	step	step	step	multi-step
lr_interval	30k	20k - 80k, every 10k	-	30k	30k	30k	30k	30k	30k	30k - 80k, every 10k
lr_scale	0.2	0.5	-	0.2	0.2	0.1	0.2	0.2	0.2	0.5
momentum	0.9	0.9	-	-	-	0.9	-	-	0.95	0.95
optimizer	SGD	SGD	Adam	Adam	Adam	SGD	Adam	Adam	SGD	SGD
weight_decay	0.0005	0.0001	0.0001	0.0001	0.0001	0.0005	0.0001	0.0001	0.0001	0.0001
ρ (ours) or reg_rate (PQ)	-	-	0.0001	1.01	1.05	-	-	0.0001	1.01	1.05

Table 4: *Hyperparameter settings used for the experiments. Here, if* lr_decay == step*, then the learning rate is multiplied by* lr_scale *for every* lr_interval *iterations. On the other hand, if* lr_decay == multi-step*, the learning rate is multiplied by* lr_scale *whenever the iteration count reaches any of the milestones specified by* lr_interval*. Here, ρ denotes the growth rate of β (refer Algorithm 1) and β is multiplied by ρ every 100 iterations.*