# Proximal Mean-field for Neural Network Quantization

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

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

## Abstract

*Compressing large Neural Networks (NN) by quantizing the parameters, while maintaining the performance is highly desirable due to reduced memory and time complexity. In this work, we cast NN quantization as a discrete labelling problem and leverage results from the extensively studied MRF optimization literature. Specifically, we examine relaxations to the discrete labelling problem, leading to an efficient iterative optimization procedure that involves stochastic gradient descent followed by a projection. We prove that our simple projected gradient descent approach is, in fact, equivalent to a proximal version of the well-known mean-field method. These findings allow the decades-old and theoretically grounded research on MRF optimization to be used to design better network quantization schemes. Our experiments on standard classification datasets (MNIST, CIFAR10/100, TinyImageNet) with convolutional and residual architectures evidence that our algorithm obtains fully-quantized networks with accuracies very close to the floating-point reference networks.*

## 1. Introduction

Despite the success of deep neural networks, they are highly overparametrized, resulting in excessive computational and memory requirements. Compressing such large networks by quantizing the parameters, while maintaining the performance, is highly desirable for real-time applications, or for resource-limited devices.

In Neural Network (NN) quantization, the objective is to learn a network while restricting the parameters to take values from a small discrete set (usually binary) representing quantization levels. This can be formulated as a *discrete labelling problem* where each learnable parameter takes a label from the discrete set and the learning objective is to find the label configuration that minimizes the empirical loss. This is an extremely challenging discrete optimization problem because the number of label configurations grows exponentially with the number of parameters in the network and the loss function is highly non-convex.

Over the past 20 years, similar large-scale discrete labelling problems have been extensively studied under the context of Markov Random Field (MRF) optimization, and many efficient approximate algorithms have been developed [2, 7, 12, 33, 43, 44]. In this work, we take inspiration from the rich literature on MRF optimization, and design an efficient approximate algorithm based on the popular mean-field method [44] for NN quantization.

Specifically, we first formulate NN quantization as a discrete labelling problem. Then, we relax the discrete solution space to a convex polytope and introduce an algorithm to iteratively optimize the first-order Taylor approximation of the loss function over the polytope. This approach is in fact a (stochastic) gradient descent method with an additional projection step at each iteration. For a particular choice of projection, we show that our method is equivalent to a proximal version of the well-known mean-field method. Furthermore, we prove that under certain conditions, our algorithm specializes to popular BinaryConnect [11].

The MRF optimization perspective to NN quantization opens up many interesting research directions. In fact, our approach represents the simplest case where the NN parameters are assumed to be independent of each other. However, one can potentially model second-order or even high-order interactions among parameters and use efficient inference algorithms developed and well-studied in the MRF optimization literature. Therefore, we believe, many such algorithms can be transposed into this framework to design better network quantization schemes. Furthermore, in contrast to the existing NN quantization methods [23, 37], we quantize *all* the learnable parameters in the network (including biases) and our formulation can be seamlessly extended to quantization levels beyond binary.

We evaluate the merits of our algorithm on MNIST, CIFAR-10/100, and TinyImageNet classification datasets with convolutional and residual architectures. Our experiments evidence that the quantized networks obtained by our algorithm yield accuracies very close to the floating-

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

point counterparts while consistently outperforming directly comparable baselines.

## 2. Neural Network Quantization

Neural Network (NN) quantization is the problem of learning neural network parameters restricted to a small discrete set representing quantization levels. This primarily relies on the hypothesis that the overparametrization of NNs would make it possible to obtain a quantized network with performance comparable to the floating-point network. To this end, given a dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$, NN quantization problem can be written as:

$$\begin{aligned} \min_{\mathbf{w}} L(\mathbf{w}; \mathcal{D}) &:= \frac{1}{n} \sum_{i=1}^n \ell(\mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)) \;, \\ \mathbf{w} &\in \mathcal{Q}^m \;. \end{aligned} \tag{1}$$

Here, $\ell(\cdot)$ is the input-output mapping composed with a standard loss function (*e.g.*, cross-entropy loss), $\mathbf{w}$ is the $m$ dimensional parameter vector, and $\mathcal{Q}$ with $|\mathcal{Q}| = d$ is a predefined discrete set representing quantization levels (*e.g.*, $\mathcal{Q} = \{-1, 1\}$ or $\mathcal{Q} = \{-1, 0, 1\}$). In Eq. (1), we seek a *fully-quantized network* where all the learnable parameters including biases are quantized. This is in contrast to the previous methods [11, 37] where some parts of the network are not quantized (*e.g.*, biases and last layer parameters).

### 2.1. NN Quantization as Discrete Labelling

NN quantization (1) naturally takes the form of a *discrete labelling problem* where each learnable parameter $w_j$ takes a label $\lambda$ from the discrete set $\mathcal{Q}$. In particular, Eq. (1) is directly related to an MRF optimization problem [25] where the set of random variables corresponds to the set of weights $\mathbf{w}$, the label set is $\mathcal{Q}$, and the energy function is $L(\mathbf{w})$. We refer to Appendix C for a brief overview on MRFs.

An important part of an MRF is the factorization of the energy function that depends on the underlying neighbourhood structure (*i.e.*, interactions among the random variables). While modelling a problem as an MRF, the emphasis is given to the form of the energy function (*e.g.*, submodularity) as well as the form of the interactions (cliques), because both of these aspects determine the complexity of the resulting optimization. In the case of NNs, the energy function (*i.e.*, loss) is a composition of functions which, in general, has a variety of interactions among the random variables. For example, a parameter at the initial layer is related to another parameter at the final layer via the composition of functions. Thus, the energy function does not have an explicit factorization. In fact, optimizing Eq. (1) directly is intractable due to the following inherent problems [28, 34]:

1. The solution space is discrete with exponentially many feasible points ($d^m$ with $m$ in the order of millions).
2. The loss function is highly non-convex and does not satisfy any regularity condition (*e.g.*, submodularity).
3. The loss function does not have an explicit factorization.

This hinders the use of any off-the-shelf discrete optimization algorithm. However, to tackle the aforementioned problems, we take inspiration from the MRF optimization literature [6, 10, 44]. In particular, we first relax the discrete solution space to a convex polytope and then iteratively optimize the first-order Taylor approximation of the loss function over the polytope. Our approach, as will be shown subsequently, falls under the regime of (stochastic) gradient descent methods and is applicable to any loss function. Next we describe these relaxations and the related optimization in detail.

### 2.2. Continuous Relaxation of the Solution Space

The parameter vector $\mathbf{w} \in \mathcal{Q}^m$ can be equivalently represented using indicator variables as follows. Let $u_{j:\lambda} \in \{0, 1\}$ be the indicator variable, where $u_{j:\lambda} = 1$ if and only if $w_j = \lambda \in \mathcal{Q}$. Then, for any $j \in \{1 \dots m\}$, we can write

$$\begin{aligned} w_j &= \sum_{\lambda \in \mathcal{Q}} \lambda\, u_{j:\lambda} \;, \\ \text{s.t.} \quad &\sum_{\lambda \in \mathcal{Q}} u_{j:\lambda} = 1, \quad u_{j:\lambda} \in \{0, 1\} \quad \forall\, \lambda \in \mathcal{Q} \;. \end{aligned} \tag{2}$$

Note, any $w_j$ represented using Eq. (2) belongs to $\mathcal{Q}$. For convenience, by denoting the vector of quantization levels as $\mathbf{q}$, a parameter vector $\mathbf{w} \in \mathcal{Q}^m$ can be written in a matrix vector product as:

$$\begin{aligned} \mathbf{w} &= \mathbf{u}\mathbf{q} \;, \\ \text{s.t.} \quad \mathbf{u} \in \mathcal{V} &= \left\{ \mathbf{u} \;\middle|\; \begin{array}{ll} \sum_\lambda u_{j:\lambda} = 1, & \forall\, j \\ u_{j:\lambda} \in \{0, 1\}, & \forall\, j, \lambda \end{array} \right\} \;. \end{aligned} \tag{3}$$

Here, $\mathbf{u}$ is a $m \times d$ matrix[1] (*i.e.*, each row $\mathbf{u}_j = \{u_{j:\lambda} \mid \lambda \in \mathcal{Q}\}$), and $\mathbf{q}$ is a column vector of dimension $d$. Note that there is a one-to-one correspondence between the sets $\mathcal{V}$ and $\mathcal{Q}^m$. Substituting Eq. (3) in the NN quantization objective (1) results in the variable change from $\mathbf{w}$ to $\mathbf{u}$ as:

$$\min_{\mathbf{w} \in \mathcal{Q}^m} L(\mathbf{w}; \mathcal{D}) = \min_{\mathbf{u} \in \mathcal{V}} \quad \frac{1}{n} \sum_{i=1}^n \ell(\mathbf{u}\mathbf{q}; (\mathbf{x}_i, \mathbf{y}_i)) \;. \tag{4}$$

Even though the above variable change converts the problem from $m$ to $md$ dimensional space, the one-to-one correspondence between $\mathbf{w}$ and $\mathbf{u}$ ensures that the cardinality of the sets $\mathcal{Q}^m$ and $\mathcal{V}$ are exactly the same. The binary constraint $u_{j:\lambda} \in \{0, 1\}$ together with the non-convex loss function $L(\cdot)$ makes the problem NP-hard [34]. By relaxing

[1] To simplify the notation, we denote $\mathbf{u}$ as a matrix flattening of which will give an $md$ dimensional vector.

the binary constraints to $u_{j:\lambda} \in [0, 1]$, we obtain the convex hull of the set $\mathcal{V}$ as defined below:

$$\mathcal{S} = \mathrm{conv}(\mathcal{V}) = \left\{ \mathbf{u} \;\middle|\; \begin{array}{ll} \sum_\lambda u_{j:\lambda} = 1, & \forall\, j \\ u_{j:\lambda} \geq 0, & \forall\, j, \lambda \end{array} \right\} . \tag{5}$$

Furthermore, the set $\mathcal{S}$ decomposes over each $j$, and it is in fact the Cartesian product of the probability simplexes of dimension $d$. Thus,

$$\mathcal{S} = \prod_{j=1}^{m} \Delta_j, \quad \text{where } \Delta_j = \left\{ \mathbf{z} \;\middle|\; \begin{array}{ll} \sum_\lambda z_\lambda = 1 & \\ z_\lambda \geq 0, & \forall\, \lambda \end{array} \right\} . \tag{6}$$

Therefore, for a feasible point $\mathbf{u} \in \mathcal{S}$, the vector $\mathbf{u}_j$ for each $j$ ($j$-th row of matrix $\mathbf{u}$) belongs to the probability simplex of dimension $d$. Hence, we can interpret the value $u_{j:\lambda}$ as the probability of assigning the discrete label $\lambda$ to the weight $w_j$. Now, the relaxed optimization can be written as:

$$\min_{\mathbf{u}\in\mathcal{S}} \tilde{L}(\mathbf{u}; \mathcal{D}) := L(\mathbf{u}\mathbf{q}; \mathcal{D}) \ , \tag{7}$$

Note that, for any quantization set $\mathcal{Q}$, if $\mathbf{u} \in \mathcal{V}$, then the loss function $\tilde{L}(\mathbf{u})$ has the same value as the original loss function $L(\mathbf{w})$. Furthermore, the relaxation of $\mathbf{u}$ from $\mathcal{V}$ to $\mathcal{S}$, translates into relaxing $\mathbf{w}$ from $\mathcal{Q}^m$ to the convex region $[q_{\min}, q_{\max}]^m$. Here, $q_{\min}$ and $q_{\max}$ represent the minimum and maximum quantization levels, respectively.

In fact, $\mathbf{u} \in \mathcal{S}$ is an overparametrized representation of $\mathbf{w} \in [q_{\min}, q_{\max}]^m$, and the mapping $\mathbf{u} \to \mathbf{w}$ (*i.e.*, $\mathbf{w} = \mathbf{u}\mathbf{q}$) is many-to-one, and precisely a *surjective*[2] mapping. However, this representation has an interesting probabilistic interpretation that *learning* $\mathbf{u}$ *can be interpreted as learning a discrete probability distribution over the* NN *parameters* $\mathbf{w}$. This interpretation would be useful in drawing the connection between our algorithm and the mean-field method later in Sec. 3.

In addition, it can be shown that any local minimum of Eq. (7) (the relaxed $\mathbf{u}$-space) is also a local minimum of the loss in $[q_{\min}, q_{\max}]^m$ (the relaxed $\mathbf{w}$-space) and vice versa (Proposition 2.1). This essentially means that the variable change from $\mathbf{w}$ to $\mathbf{u}$ does not alter the optimization problem and a local minimum in the $\mathbf{w}$-space can be obtained by optimizing in the $\mathbf{u}$-space.

**Proposition 2.1.** Let $f(\mathbf{w})$ be a continuous function with $\mathbf{w} = g(\mathbf{u}) = \mathbf{u}\mathbf{q}$, where $\mathbf{w} \in [q_{\min}, q_{\max}]^m$. Then a point $\mathbf{u}^k \in \mathcal{S}$ is a local minimum of $f \circ g$, if and only if $\mathbf{w}^k = \mathbf{u}^k\mathbf{q}$ is a local minimum of $f$ in the region $[q_{\min}, q_{\max}]^m$.

*Proof.* It can be easily proved using the properties that the function $g : \mathcal{S} \to [q_{\min}, q_{\max}]^m$ is surjective and continuous. See Appendix A. □

Finally, we would like to point out that the relaxation used while moving from $\mathbf{w}$ to $\mathbf{u}$ space is well studied in the MRF optimization literature and has been used to prove bounds on the quality of the solutions [10, 27]. In the case of NN quantization, in addition to the connection to mean-field (Sec. 3), we believe that this relaxation allows for exploration, which would be useful in the stochastic setting.

### 2.3. First-order Approximation and Optimization

Here we talk about the optimization of $\tilde{L}(\mathbf{u})$ over $\mathcal{S}$, discuss how our optimization scheme allows exploration in the parameter space, and also discuss the conditions when this optimization will lead to a quantized solution in the $\mathbf{w}$ space, which is our prime objective.

Stochastic Gradient Descent (SGD)[3] [39] is the de facto method of choice for optimizing neural networks. In this section, we interpret SGD as a proximal method, which will be useful later to show its difference to our final algorithm. In particular, SGD (or gradient descent) can be interpreted as iteratively minimizing the first-order Taylor approximation of the loss function augmented by a proximal term [35]. In our case, the objective function is the same as SGD but the feasible points are now constrained to form a convex polytope. Thus, at each iteration $k$, the first-order objective can be written as:

$$\mathbf{u}^{k+1} = \operatorname*{argmin}_{\mathbf{u}\in\mathcal{S}} \tilde{L}(\mathbf{u}^k) + \left\langle \mathbf{g}^k, \mathbf{u} - \mathbf{u}^k \right\rangle_F + \frac{1}{2\eta} \left\| \mathbf{u} - \mathbf{u}^k \right\|_F^2 , \tag{8}$$

where $\eta > 0$ is the learning rate and $\mathbf{g}^k := \nabla_{\mathbf{u}} \tilde{L}^k$ is the stochastic (or mini-batch) gradient of $\tilde{L}$ with respect to $\mathbf{u}$ evaluated at $\mathbf{u}^k$. Here, $\langle \cdot, \cdot \rangle_F$ is the Frobenius inner product[4] and $\| \cdot \|_F$ is the Frobenius norm, respectively. In the unconstrained case, by setting the derivative with respect to $\mathbf{u}$ to zero, one can easily verify that the above formulation leads to standard SGD updates. For constrained optimization (as in our case (8)), it is natural to use the stochastic version of Projected Gradient Descent (PGD) [40]. Specifically, at iteration $k$, the projected stochastic gradient update can be written as:

$$\mathbf{u}^{k+1} = P_{\mathcal{S}}\left(\mathbf{u}^k - \eta\, \mathbf{g}^k\right) \ , \tag{9}$$

where $P_{\mathcal{S}}(\cdot)$ denotes the projection to the polytope $\mathcal{S}$. Even though this type of problems can be optimized using projection-free algorithms [3, 15, 29], by relying on PGD, we enable the use of any off-the-shelf first-order optimization algorithms (*e.g.*, Adam [26]). Furthermore, for a particular choice of projection, we show that the PGD update is equivalent to a proximal version of the mean-field method.

[2] A mapping $f(x) : \mathcal{X} \to \mathcal{Y}$ is surjective if $\forall\, y \in \mathcal{Y}$, $\exists\, x \in \mathcal{X}$ such that $f(x) = y$.

[3] The difference between SGD and gradient descent is that the gradients are approximated using a stochastic oracle in the former case.

[4] This is equivalent to vectorizing the matrices and applying the standard inner product.

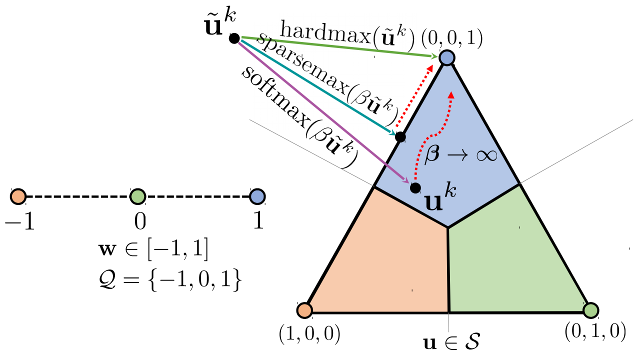
$\tilde{\mathbf{u}}^k$
$\text{hardmax}(\tilde{\mathbf{u}}^k)$
$(0,0,1)$
$\text{sparsemax}(\beta\tilde{\mathbf{u}}^k)$
$\text{softmax}(\beta\tilde{\mathbf{u}}^k)$
$\beta \to \infty$
$\mathbf{u}^k$
$-1$
$0$
$1$
$\mathbf{w} \in [-1,1]$
$\mathcal{Q} = \{-1,0,1\}$
$(1,0,0)$
$\mathbf{u} \in \mathcal{S}$
$(0,1,0)$

Figure 1: *Illustration of* $\mathbf{w}$ *and* $\mathbf{u}$*-spaces, different projections, and exploration with* softmax *when* $m = 1$. *Here each vertex of the simplex corresponds to a discrete quantization level in the* $\mathbf{w}$*-space and the simplex is partitioned based on its vertex association. Given an infeasible point* $\tilde{\mathbf{u}}^k$*, it is projected to the simplex via* softmax *(or* sparsemax*) and when* $\beta \to \infty$*, the projected point would move towards the associated vertex.*

#### 2.3.1 Projection to the Polytope $\mathcal{S}$

From Eq. (6), the polytope $\mathcal{S}$ can be decomposed into $m$ probability simplexes of dimension $d$. Therefore, projection to $\mathcal{S}$ can be decomposed into $m$ independent projections to the $d$-dimensional probability simplexes. Since the objective function (8) is also separable for each $j$, the PGD algorithm has an assumption that the probability of parameter $w_j$ taking a label $\lambda$ (represented by $u_{j:\lambda}$) is independent for each $j$. Such an independence assumption is ubiquitous in NN literature due to its computational efficiency [4, 13]. Thus, for notational convenience, without loss of generality, we assume $m = 1$. Now, for a given updated parameter $\tilde{\mathbf{u}}^{k+1} = \mathbf{u}^k - \eta\,\mathbf{g}^k$ (where $\tilde{\mathbf{u}}^{k+1} \in \mathbb{R}^{\mathrm{d}}$), we discuss three approaches of projecting to the probability simplex $\Delta$. An illustration of these projections is shown in Fig. 1. In this subsection, for brevity, we also ignore the superscript $k+1$.

**Euclidean Projection (Sparsemax):** The standard approach of projecting to a set in the Euclidean space is via sparsemax [32]. Given a scalar $\beta > 0$ (usually $\beta = 1$), sparsemax amounts to finding a point $\mathbf{u}$ in $\Delta$ which is the closest to $\beta\tilde{\mathbf{u}}$, *i.e.*,

$$\mathbf{u} = \text{sparsemax}(\beta\tilde{\mathbf{u}}) = \underset{\mathbf{z}\in\Delta}{\text{argmin}}\ \|\mathbf{z} - \beta\tilde{\mathbf{u}}\|^2\ . \tag{10}$$

As the name suggests, this projection is likely to hit the boundary of the simplex[5], resulting in sparse solutions ($\mathbf{u}$) at every iteration. Please refer to [32] for more detail. As $\beta$ increases, the projected point moves towards a vertex.

**Hardmax Projection:** Compared to sparsemax, hardmax projection is rigid, where a given $\tilde{\mathbf{u}}$ is projected to one of the vertices of the simplex $\Delta$:

$$\mathbf{u} = \text{hardmax}(\tilde{\mathbf{u}})\ , \tag{11}$$

$$u_\lambda = \begin{cases} 1 & \text{if } \lambda = \underset{\mu\in\mathcal{Q}}{\text{argmax}}\ \tilde{u}_\mu \\ 0 & \text{otherwise} \end{cases} \qquad \forall\,\lambda \in \mathcal{Q}\ .$$

Note, by design, hardmax finds extremely sparse projections and it is easy to show that hardmax is exactly the Euclidean projection to the set of vertices of the simplex.

**Softmax Projection:** We now discuss the softmax projection which projects a point to the interior of the simplex, leading to dense solutions. Given a scalar $\beta > 0$, the softmax projection is:

$$\mathbf{u} = \text{softmax}(\beta\tilde{\mathbf{u}})\ , \tag{12}$$

$$u_\lambda = \frac{e^{\beta\tilde{u}_\lambda}}{\sum_{\mu\in\mathcal{Q}} e^{\beta\tilde{u}_\mu}} \quad \forall\,\lambda \in \mathcal{Q}\ .$$

Even though approximate in the Euclidean sense, softmax shares many desirable properties to sparsemax [32] (*e.g.*, preserves the relative order of $\tilde{\mathbf{u}}$) and when $\beta \to \infty$, the projected point moves towards a vertex.

#### 2.3.2 Exploration and Quantization using Softmax

All of the projections discussed above are valid in the sense that the projected point lies in the polytope $\mathcal{S}$. However, our goal is to obtain a quantized solution in the $\mathbf{w}$-space which is equivalent to obtaining a solution $\mathbf{u}$ that is a vertex of the polytope $\mathcal{S}$. Below we provide justifications behind using softmax with a monotonically increasing schedule for $\beta$ in realizing this goal, rather than either sparsemax or hardmax projection.

Recall that, the main reason for relaxing the feasible points to lie within the convex polytope $\mathcal{S}$ is to simplify the optimization problem with the hope that optimizing this relaxation will lead to a better solution. However, in case of hardmax and sparsemax projections, the effective solution space is restricted to be either the set of vertices $\mathcal{V}$ (no relaxation) or the boundary of the polytope (much smaller subset of $\mathcal{S}$). Such restrictions hinder exploration over the polytope and do not fully utilize the potential of the relaxation. In contrast, softmax allows *exploration* over the entire polytope and a monotonically increasing schedule for $\beta$ ensures that the solution gradually approaches a vertex. This interpretation is illustrated in Fig. 1.

**Entropy based view of Softmax:** In fact, softmax can be thought of as a "noisy" projection to the set of vertices $\mathcal{V}$, where the noise is controlled by the hyperparameter $\beta$. We now substantiate this interpretation by providing an entropy based view for the softmax projection.

**Lemma 2.1.** *Let* $\mathbf{u}^k = \text{softmax}(\beta\tilde{\mathbf{u}}^k)$.[6] *Then,*

$$\mathbf{u}^k = \underset{\mathbf{u}\in\mathcal{S}}{\text{argmax}}\ \left\langle \tilde{\mathbf{u}}^k, \mathbf{u} \right\rangle_F + \frac{1}{\beta} H(\mathbf{u})\ , \tag{13}$$

[5] Unless $\beta\tilde{\mathbf{u}}$ when projected to the simplex plane is in $\Delta$, which is rare.

[6] Denotes softmax applied to each $j \in \{1 \ldots m\}$ independently.

*where* $H(\mathbf{u}) = -\sum_{j=1}^{m} \sum_{\lambda\in\mathcal{Q}} u_{j:\lambda} \, \log u_{j:\lambda}$ *is the entropy.*

*Proof.* This can be proved by writing the Lagrangian and setting the derivatives to zero. See Appendix B. □

It is interesting to note that the softmax projection translates into an entropy term in the objective function (13), and for small values of $\beta$, it allows the iterative procedure to explore the optimization landscape. We believe, in the stochastic setting, such an explorative behaviour is crucial, especially in the early stage of training. Furthermore, our empirical results validate this hypothesis that PGD with softmax projection is relatively easy to train and yields consistently better results compared to other PGD variants. Note that, when $\beta \to \infty$, the entropy term vanishes and the softmax function approaches hardmax.

Note, constraining the solution space through a hyperparameter ($\beta$ in our case) has been extensively studied in the optimization literature and one such example is the barrier method [8]. Moreover, even though the softmax based PGD update yields an approximate solution to Eq. (8), in Sec. 3, we prove that it is theoretically equivalent to a proximal version of the mean-field method.

## 3. Softmax based PGD as Proximal Mean-field

Here we discuss the connection between softmax based PGD and the well-known mean-field method [44]. Precisely, we show that the update $\mathbf{u}^{k+1} = \mathrm{softmax}(\beta(\mathbf{u}^k - \eta\,\mathbf{g}^k))$ is actually an *exact fixed point update* of a modified mean-field objective function. This connection bridges the gap between the MRF optimization and the NN quantization literature. We now begin with a brief review of the mean-field method and then proceed with our proof.

**Mean-field Method.** A self-contained overview is provided in Appendix C, but here we review the important details. Given an energy (or loss) function $L(\mathbf{w})$ and the corresponding probability distribution of the form $P(\mathbf{w}) = e^{-L(\mathbf{w})}/Z$, mean-field approximates $P(\mathbf{w})$ using a fully-factorized distribution $U(\mathbf{w}) = \prod_{j=1}^{m} U_j(w_j)$. Here, the distribution $U$ is obtained by minimizing the KL-divergence $\mathrm{KL}(\mathrm{U}\|\mathrm{P})$. Note that, from the probabilistic interpretation of $\mathbf{u} \in \mathcal{S}$ (see Sec. 2.2), for each $j \in \{1\ldots m\}$, the probability $U_j(w_j = \lambda) = u_{j:\lambda}$. Therefore, the distribution $U$ can be represented using the variables $\mathbf{u} \in \mathcal{S}$, and hence, the mean-field objective can be written as:

$$\underset{\mathbf{u}\in\mathcal{S}}{\mathrm{argmin}}\ \mathrm{KL}(\mathbf{u}\|\mathrm{P}) = \underset{\mathbf{u}\in\mathcal{S}}{\mathrm{argmin}}\ \mathbb{E}_{\mathbf{u}}[L(\mathbf{w})] - H(\mathbf{u})\ , \tag{14}$$

where $\mathbb{E}_{\mathbf{u}}[\cdot]$ is expectation over $\mathbf{u}$ and $H(\mathbf{u})$ is the entropy.

In fact, mean-field has been extensively studied in the MRF literature where the energy function $L(\mathbf{w})$ factorizes over small subsets of variables $\mathbf{w}$. This leads to efficient minimization of the KL-divergence as the expectation $\mathbb{E}_{\mathbf{u}}[L(\mathbf{w})]$ can be computed efficiently. However, in a standard neural network, the function $L(\mathbf{w})$ does not have an explicit factorization and direct minimization of the KL-divergence is not straight forward. To simplify the NN loss function one can approximate it using its first-order Taylor approximation which discards the interactions between the NN parameters altogether.

Interestingly, in Theorem 3.1, we show that our softmax based PGD iteratively applies a proximal version of mean-field to the first-order approximation of $L(\mathbf{w})$. At iteration $k$, let $\hat{L}^k(\mathbf{w})$ be the first-order Taylor approximation of $L(\mathbf{w})$. Then, since there are no interactions among parameters in $\hat{L}^k(\mathbf{w})$, and it is linear, our proximal mean-field objective has a closed form solution, which is exactly the softmax based PGD update.

**Theorem 3.1.** *Let* $\mathbf{u}^{k+1} = \mathrm{softmax}(\beta(\mathbf{u}^k - \eta\,\mathbf{g}^k))$ *be the point from the* softmax *based* PGD *update. Then,*

$$\mathbf{u}^{k+1} = \underset{\mathbf{u}\in\mathcal{S}}{\mathrm{argmin}}\ \eta\,\mathbb{E}_{\mathbf{u}}\left[\hat{L}^k(\mathbf{w})\right] - \left\langle \mathbf{u}^k, \mathbf{u}\right\rangle_F - \frac{1}{\beta}H(\mathbf{u})\ , \tag{15}$$

*where* $\hat{L}^k(\mathbf{w})$ *is the first-order Taylor approximation of* $L$ *at* $\mathbf{w}^k = \mathbf{u}^k\mathbf{q}$ *and* $\eta > 0$ *is the learning rate.*

*Proof.* First we will show that, $\mathbb{E}_{\mathbf{u}}[\hat{L}^k(\mathbf{w})] = \left\langle \mathbf{g}^k, \mathbf{u}\right\rangle_F + c$ for some constant $c$. Then the proof follows from Lemma 2.1. See Appendix D. □

The objective function Eq. (15) is the same as the mean-field objective for $\hat{L}^k(\mathbf{w})$ (refer Eq. (14)) except for the term $\langle \mathbf{u}^k, \mathbf{u}\rangle_F$. This, in fact, acts as a proximal term. Note, it is the cosine similarity but subtracted from the loss to enforce proximity. Therefore, it encourages the resulting $\mathbf{u}^{k+1}$ to be closer to the current point $\mathbf{u}^k$ and its influence relative to the loss term is governed by the learning rate $\eta$. Since gradient estimates are stochastic in our case, such a proximal term is highly desired as it encourages the updates to make a smooth transition.

Furthermore, the negative entropy term acts as a convex regularizer and when $\beta \to \infty$ its influence becomes negligible and the update results in a binary labelling $\mathbf{u} \in \mathcal{V}$. In addition, the entropy term in Eq. (15) captures the (in)dependency between the parameters. To encode dependency, the entropy of the fully-factorized distribution can perhaps be replaced with a more complex entropy such as a tree-structured entropy, following the idea of [38]. Furthermore, in place of $\hat{L}^k$, a higher-order approximation can be used. Such explorations go beyond the scope of this paper.

**Remark.** Note that, our update (15) is similar in spirit to that of the theoretically sound mirror-descent algorithm (especially the dual averaging version) when entropy is chosen as the mirror-map (refer Sec. 4.3 of [9]). In fact, at each iteration, both our algorithm and mirror-descent augment

**Algorithm 1** Proximal Mean-Field (PMF)

```
Require: $K, b, \{\eta^k\}, \rho > 1, \mathcal{D}, \tilde{L}$
Ensure: $\mathbf{w}^* \in \mathcal{Q}^m$
$\tilde{\mathbf{u}}^0 \in \mathbb{R}^{m\times d}$,   $\beta \leftarrow 1$                              ▷ Initialization
for $k \leftarrow 0 \dots K$ do
    $\mathbf{u}^k \leftarrow \mathrm{softmax}(\beta\tilde{\mathbf{u}}^k)$                    ▷ Projection (Eq. (12))
    $\mathcal{D}^b = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^b \sim \mathcal{D}$          ▷ Sample a mini-batch
    $\mathbf{g}^k_{\mathbf{u}} \leftarrow \nabla_{\mathbf{u}}\tilde{L}(\mathbf{u}; \mathcal{D}^b)\Big|_{\mathbf{u}=\mathbf{u}^k}$    ▷ Gradient w.r.t. $\mathbf{u}$ at $\mathbf{u}^k$
    $\mathbf{g}^k_{\tilde{\mathbf{u}}} \leftarrow \mathbf{g}^k_{\mathbf{u}} \frac{\partial \mathbf{u}}{\partial \tilde{\mathbf{u}}}\big|_{\tilde{\mathbf{u}}=\tilde{\mathbf{u}}^k}$             ▷ Gradient w.r.t. $\tilde{\mathbf{u}}$ at $\mathbf{u}^k$
    $\tilde{\mathbf{u}}^{k+1} \leftarrow \tilde{\mathbf{u}}^k - \eta^k \mathbf{g}^k_{\tilde{\mathbf{u}}}$        ▷ Gradient descent on $\tilde{\mathbf{u}}$
    $\beta \leftarrow \rho\beta$                                  ▷ Increase $\beta$
end for
$\mathbf{w}^* \leftarrow \mathrm{hardmax}(\tilde{\mathbf{u}}^K)\mathbf{q}$               ▷ Quantization (Eq. (11))
```

the gradient descent objective with a negative entropy term and optimizes over the polytope. However, compared to mirror-descent, our update additionally constitutes a proximal term and an annealing hyperparameter $\beta$ which enables us to gradually enforce a discrete solution. Therefore, to employ mirror-descent, one needs to understand the effects of using adaptive mirror-maps (that depend on $\beta$). Nevertheless, it is interesting to explore the potential of mirror-descent which could facilitate various choices for mirror-map beyond entropy, and allow us to derive different variants of our algorithm.

Algorithm 1 summarizes our approach. Here, similar to the existing methods [23] we store the auxiliary variables $\tilde{\mathbf{u}} \in \mathbb{R}^{m\times d}$ and perform gradient descent on them. In contrast to the existing methods, this is not a necessity but empirically it improves the performance. Finally, since $\beta$ can never be $\infty$, to ensure a fully-quantized network, the final quantization is performed using hardmax. This is equivalent to performing Maximum a Posteriori (MAP) estimate on the learned probability distribution $\mathbf{u} \in \mathcal{S}$. Since, softmax approaches hardmax when $\beta \to \infty$, the fixed points of Algorithm 1 corresponds to the fixed points of PGD with the hardmax projection. However, exploration due to softmax allows our algorithm to converge to fixed points with better validation errors as evidenced by our experiments.

### 3.1. Proximal ICM as a Special Case

For PGD, if hardmax is used instead of the softmax projection, the resulting update is the same as a proximal version of Iterative Conditional Modes (ICM) [6]. In fact, following the proof of Lemma 2.1, it can be shown that the update $\mathbf{u}^{k+1} = \mathrm{hardmax}(\mathbf{u}^k - \eta\,\mathbf{g}^k)$ yields a fixed point of the following equation:

$$\min_{\mathbf{u}\in\mathcal{S}} \eta \left\langle \mathbf{g}^k, \mathbf{u} \right\rangle_F - \left\langle \mathbf{u}^k, \mathbf{u} \right\rangle_F . \tag{16}$$

Notice, this is exactly the same as the ICM objective augmented by the proximal term. In this case, $\mathbf{u} \in \mathcal{V} \subset \mathcal{S}$, meaning, the feasible domain is restricted to be the vertices of the polytope $\mathcal{S}$. Since softmax approaches hardmax when $\beta \to \infty$, this is a special case of proximal mean-field.

**Algorithm 2** One iteration of BinaryConnect (BC) [11]

```
Require: $\tilde{\mathbf{w}}^k, \eta_{\mathbf{w}}, \mathcal{D}, L$
$\mathbf{w}^k \leftarrow \mathrm{sign}(\tilde{\mathbf{w}}^k)$                                  ▷ Projection
$\mathbf{g}^k_{\mathbf{w}} \leftarrow \nabla_{\mathbf{w}} L(\mathbf{w}; \mathcal{D})|_{\mathbf{w}=\mathbf{w}^k}$               ▷ Gradient w.r.t. $\mathbf{w}$
$\mathbf{g}^k_{\tilde{\mathbf{w}}} \leftarrow \mathbf{g}^k_{\mathbf{w}} \frac{\partial \mathbf{w}}{\partial \tilde{\mathbf{w}}}\big|_{\tilde{\mathbf{w}}=\tilde{\mathbf{w}}^k}$                      ▷ Gradient w.r.t. $\tilde{\mathbf{w}}$
$\tilde{\mathbf{w}}^{k+1} \leftarrow \tilde{\mathbf{w}}^k - \eta_{\mathbf{w}}\,\mathbf{g}^k_{\tilde{\mathbf{w}}}$                     ▷ Gradient descent
```

### 3.2. BinaryConnect as Proximal ICM

In this section, considering binary neural networks, *i.e.*, $\mathcal{Q} = \{-1, 1\}$, and non-stochastic setting, we show that the Proximal Iterative Conditional Modes (PICM) algorithm is equivalent to the popular BinaryConnect (BC) method [11]. In these algorithms, the gradients are computed in two different spaces and therefore to alleviate any discrepancy we assume that gradients are computed using the full dataset.

Let $\tilde{\mathbf{w}} \in \mathbb{R}^m$ and $\mathbf{w} \in \mathcal{Q}^m$ be the infeasible and feasible points of BC. Similarly, $\tilde{\mathbf{u}} \in \mathbb{R}^{m\times d}$ and $\mathbf{u} \in \mathcal{V} \subset \mathcal{S}$ be the infeasible and feasible points of our PICM method, respectively. For convenience, we summarize one iteration of BC in Algorithm 2. Now, we show that the update steps in both BC and PICM are equivalent.

**Proposition 3.1.** Consider BC and 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: $\mathbf{w}^k = \mathrm{sign}(\tilde{\mathbf{w}}^k)$ and PICM: $\mathbf{u}^k = \mathrm{hardmax}(\tilde{\mathbf{u}}^k)$ satisfy $\mathbf{w}^k = \mathbf{u}^k\mathbf{q}$.

2. if $\eta_{\mathbf{u}} = \eta_{\mathbf{w}}/2$, 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.* Case (1) is simply applying $\tilde{\mathbf{w}}^k = \tilde{\mathbf{u}}^k\tilde{\mathbf{q}}$ whereas case (2) can be proved by writing $\mathbf{w}^k$ as a function of $\tilde{\mathbf{u}}^k$ and then applying chain rule. See Appendix E. □

Since hardmax is a non-differentiable operation, the partial derivative $\partial\mathbf{u}/\partial\tilde{\mathbf{u}} = \partial\,\mathrm{hardmax}\,/\partial\tilde{\mathbf{u}}$ is not defined. However, to allow backpropagation, we write hardmax in terms of the sign function, and used the straight-through-estimator [19] to allow gradient flow similar to binary connect. For details please refer to Appendix E.1.

## 4. Related Work

There is much work on NN quantization focusing on different aspects such as quantizing parameters [11], activations [22], loss aware quantization [20] and quantization for specialized hardware [14], to name a few. Here we give a brief summary of latest works and for a comprehensive survey we refer the reader to [17].

In this work, we consider parameter quantization, which can either be treated as a post-processing scheme [16] or incorporated into the learning process. Popular methods [11, 23] falls into the latter category and optimize the constrained problem using some form of projected stochastic gradient descent. In contrast to projection, quantization can also be enforced using a penalty term [5, 45]. Even though, our method is based on projected gradient descent, by optimizing in the $\mathbf{u}$-space, we provide theoretical insights based on mean-field and bridge the gap between NN quantization and MRF optimization literature.

In contrast, the variational approach can also be used for quantization, where the idea is to learn a posterior probability on the network parameters in a Bayesian framework. In this family of methods, the quantized network can be obtained either via a quantizing prior [1] or using the MAP estimate on the learned posterior [42]. Interestingly, the learned posterior distribution can be used to estimate the model uncertainty and in turn determine the required precision for each network parameter [31]. Note that, even in our seemingly different method, we learn a probability distribution over the parameters (see Sec. 2.2) and it would be interesting to understand the connection between Bayesian methods and our algorithm.

## 5. Experiments

Since neural network binarization is the most popular quantization [11, 37], we set the quantization levels to be binary, *i.e.*, $\mathcal{Q} = \{-1, 1\}$. However, our formulation is applicable to any predefined set of quantization levels given sufficient resources at training time. We would like to point out that, we quantize all learnable parameters, meaning, all quantization algorithms result in 32 times less memory compared to the floating point counterparts.

We evaluate our Proximal Mean-Field (PMF) algorithm on MNIST, CIFAR-10, CIFAR-100 and TinyImageNet[7] classification datasets with convolutional and residual architectures and compare against the BC method [11] and the latest algorithm ProxQuant (PQ) [5]. Note that BC and PQ constitute the closest and directly comparable baselines to PMF. Furthermore, many other methods have been developed based on BC by relaxing some of the constraints, *e.g.*, layer-wise scalars [37], and we believe, similar extensions are possible with our method as well. Our results show that the binary networks obtained by PMF yield accuracies very close to the floating point counterparts while consistently outperforming the baselines.

| Dataset | Image | # class | Train / Val. | $b$ | $K$ |
|---|---|---|---|---|---|
| MNIST | $28 \times 28$ | 10 | 50k / 10k | 100 | 20k |
| CIFAR-10 | $32 \times 32$ | 10 | 45k / 5k | 128 | 100k |
| CIFAR-100 | $32 \times 32$ | 100 | 45k / 5k | 128 | 100k |
| TinyImageNet | $64 \times 64$ | 200 | 100k / 10k | 128 | 100k |

Table 1: *Experiment setup. Here, $b$ is the batch size and $K$ is the total number of iterations used for all the methods.*

### 5.1. Experiment Setup

The details of the datasets and their corresponding experiment setups are given in Table 1. In all the experiments, standard multi-class cross-entropy loss is minimized. MNIST is tested using LeNet-300 and LeNet-5, where the former consists of three fully-connected (FC) layers while the latter is composed of two convolutional and two FC layers. For CIFAR and TinyImageNet, VGG-16 [41] and ResNet-18 [18] architectures adapted for CIFAR dataset are used. In particular, for CIFAR experiments, similar to [30], the size of the FC layers of VGG-16 is set to 512 and no dropout layers are employed. For TinyImageNet, the stride of the first convolutional layer of ResNet-18 is set to 2 to handle the image size [21]. In all the models, batch normalization [24] (with no learnable parameters) and ReLU non-linearity are used. Except for MNIST, standard data augmentation is used (*i.e.*, random crop and horizontal flip) and weight decay is set to $0.0001$ unless stated otherwise.

For all the algorithms, the hyperparameters such as the optimizer and the learning rate (also its schedule) are cross-validated using the validation set[8] and the chosen parameters are given in the supplementary material. For PMF and PGD with sparsemax, the growth-rate $\rho$ in Algorithm 1 (the multiplicative factor used to increase $\beta$) is cross validated between $1.01$ and $1.2$ and chosen values for each experiment are given in supplementary. Furthermore, since the original implementation of BC do not binarize all the learnable parameters, for fair comparison, we implemented BC in our experiment setting based on the publicly available code[9]. However, for PQ we used the original code[10], *i.e.*, *for* PQ, *biases are not binarized.* All methods are trained from a random initialization and the model with the best validation accuracy is chosen for each method. Our algorithm is implemented in PyTorch [36] and the code will be released upon publication.

[7] https://tiny-imagenet.herokuapp.com/

[8] For TinyImageNet, since the ground truth labels for the test set were not available, validation set is used for both cross-validation and testing.

[9] https://github.com/itayhubara/BinaryNet.pytorch

[10] https://github.com/allenbai01/ProxQuant

| Dataset | Architecture | REF (Float) | BC [11] | PQ [5] | Ours |  |  | REF - PMF |
|---|---|---|---|---|---|---|---|---|
|  |  |  |  |  | PICM | PGD | PMF |  |
| MNIST | LeNet-300 | 98.55 | 98.05 | 98.13 | 98.18 | 98.21 | **98.24** | +0.31 |
|  | LeNet-5 | 99.39 | 99.30 | 99.27 | 99.31 | 99.28 | **99.44** | −0.05 |
| CIFAR-10 | VGG-16 | 93.01 | 86.40 | 90.11 | 88.96 | 88.48 | **90.51** | +2.50 |
|  | ResNet-18 | 94.64 | 91.60 | 92.32 | 92.02 | 92.60 | **92.73** | +1.91 |
| CIFAR-100 | VGG-16 | 70.33 | 43.70 | 55.10 | 45.65 | 57.83 | **61.52** | +8.81 |
|  | ResNet-18 | 73.85 | 69.93 | 68.35 | 70.85 | 70.60 | **71.85** | +2.00 |
| TinyImageNet | ResNet-18 | 56.41 | 49.33 | 49.97 | 49.66 | 49.60 | **51.00** | +5.63 |

Table 2: *Classification accuracies on the test set for different methods. Note that our* PMF *algorithm consistently produces better results than other binarization methods and the degradation in performance to the full floating point network (last column) is minimal especially for small datasets. For larger datasets* (e.g.*, CIFAR-100), binarizing* ResNet-18 *results in much smaller degradation compared to VGG-16. Even though,* PICM *and* BC *are theoretically equivalent in the non-stochastic setting,* PICM *yields slightly better accuracies. Note, all binarization methods except* PQ *require exactly* **32** *times less memory compared to single-precision floating points networks at test time.*

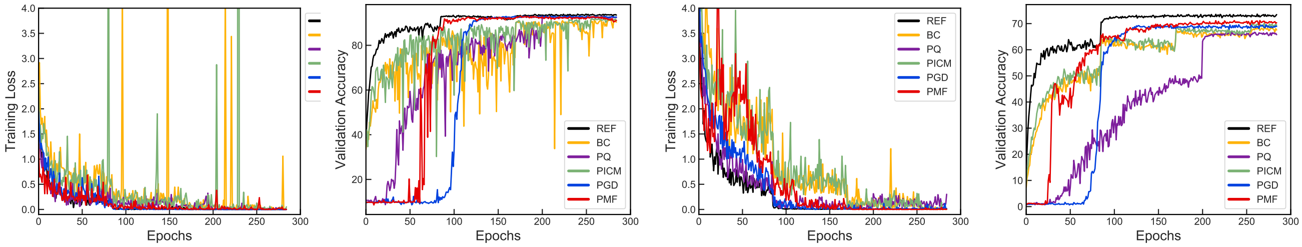

Figure 2: *Training curves for CIFAR-10 (first two) and CIFAR-100 (last two) with* ResNet-18*. For quantization methods, the validation accuracy is always measured with the quantized networks. Specifically, for* PMF *and* PGD*, the* hardmax *projection is applied before the evaluation. Notably, validation accuracy plots clearly illustrate the exploration phase of both* PMF *and* PGD*, during which the accuracies are the worst. However, once $\beta$ is "large enough", the curves closely resembles high-precision reference network while yielding very high accuracies. Furthermore, compared to* BC *and* PICM*, other methods are less noisy suggesting the usefulness of optimizing over a convex domain.*

### 5.2. Results

The classification accuracies (top-1) on the test set of all versions of our algorithm, namely, PMF, PGD (this is PGD with the sparsemax projection), and PICM, the baselines BC and PQ, and the floating point Reference Network (REF) are reported in Table 2. The training curves for CIFAR-10 and CIFAR-100 with ResNet-18 are shown in Fig. 2. Note that our PMF algorithm consistently produces better results than other binarization methods and the degradation in performance to the full floating point reference network is minimal especially for small datasets. For larger datasets (*e.g.*, CIFAR-100), binarizing ResNet-18 results in much smaller degradation compared to VGG-16.

The superior performance of PMF against BC, PICM and PGD empirically validates the hypothesis that performing "noisy" projection via softmax and annealing the noise is indeed beneficial in the stochastic setting. Furthermore, even though PICM and BC are theoretically equivalent in the non-stochastic setting, PICM yields slightly better accuracies in all our experiments. We conjecture that this is due to the fact that in PICM, the training is performed on a larger network (*i.e.*, in the $\mathbf{u}$-space).

To further consolidate our implementation of BC, we quote the accuracies reported in the original papers here. In [11], the top-1 accuracy on CIFAR-10 with a modified VGG type network is 90.10%. In the same setting, even with additional layer-wise scalars, (Binary Weight Network (BWN) [37]), the corresponding accuracy is 90.12%. For comprehensive results on network quantization we refer the reader to Table 5 of [17]. Note that, in all the above cases, the last layer parameters and biases in all layers were not binarized.

### 6. Discussion

In this work, we have formulated NN quantization as a discrete labelling problem and introduced a projected stochastic gradient descent algorithm to optimize it. By showing our approach as a proximal mean-field method, we have also provided an MRF optimization perspective to the NN quantization problem. This connection to MRF opens up many interesting research directions primarily on considering dependency between the neural network parameters to derive better network quantization schemes. Furthermore, our PMF approach learns a probability distribution over the network parameters, which is similar in spirit to Bayesian deep learning methods. Therefore, we believe, it is interesting to explore the connection between Bayesian methods and our algorithm, which can potentially drive research in both the fields.

## 7. Acknowledgements

This work was supported by the ERC grant ERC-2012-AdG 321162-HELIOS, EPSRC grant Seebibyte EP/M013774/1, EPSRC/MURI grant EP/N019474/1 and the Australian Research Council Centre of Excellence for Robotic Vision (project number CE140100016). We would also like to acknowledge the Royal Academy of Engineering and FiveAI.

# Appendices

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. F, we give the experimental details to allow reproducibility, and empirically analyse the effect of updating auxiliary variables for our PMF algorithm.

## A. Relationship between w-space and u-space

**Proposition A.1.** Let $f(\mathbf{w})$ be a continuous function with $\mathbf{w} = g(\mathbf{u}) = \mathbf{u}\mathbf{q}$, where $\mathbf{w} \in [q_{\min}, q_{\max}]^m$. Then a point $\mathbf{u}^k \in \mathcal{S}$ is a local minimum of $f \circ g$, if and only if $\mathbf{w}^k = \mathbf{u}^k\mathbf{q}$ is a local minimum of $f$ in the region $[q_{\min}, q_{\max}]^m$.

*Proof.* We will prove this by contradiction. Let $\bar{\mathbf{w}}$ be a local minimum around the neighbourhood of $\mathbf{w}^k$. Since, the function $g : \mathcal{S} \to [q_{\min}, q_{\max}]^m$ is surjective and continuous, there exists a $\bar{\mathbf{u}}$ such that $\bar{\mathbf{w}} = \bar{\mathbf{u}}\mathbf{q}$ in the neighbourhood of $\mathbf{u}^k$, and it satisfies $f \circ g(\bar{\mathbf{u}}) < f \circ g(\mathbf{u}^k)$. This is a contradiction, hence, if $\mathbf{u}^k$ is a local minimum of $f \circ g$, then $\mathbf{w}^k$ is a local minimum of $f$ in the region $[q_{\min}, q_{\max}]^m$. Similarly, from $\mathbf{w}$-space to the $\mathbf{u}$-space can be proved. □

## B. Entropy based view of Softmax

Recall the softmax update for $\tilde{\mathbf{u}}_j^k$ for $j \in \{1 \dots m\}$:

$$\mathbf{u}_j^k = \text{softmax}(\beta\tilde{\mathbf{u}}_j^k)\;, \quad \text{where} \tag{17}$$
$$u_{j:\lambda}^k = \frac{e^{\beta\tilde{u}_{j:\lambda}^k}}{\sum_{\mu\in\mathcal{Q}} e^{\beta\tilde{u}_{j:\mu}^k}} \quad \forall\,\lambda \in \mathcal{Q}\;.$$

**Lemma B.1.** *Let* $\mathbf{u}^k = \text{softmax}(\beta\tilde{\mathbf{u}}^k)$. *Then,*

$$\mathbf{u}^k = \underset{\mathbf{u}\in\mathcal{S}}{\text{argmax}}\ \left\langle \tilde{\mathbf{u}}^k, \mathbf{u}\right\rangle_F + \frac{1}{\beta}H(\mathbf{u})\;, \tag{18}$$

*where* $H(\mathbf{u}) = -\sum_{j=1}^m \sum_{\lambda\in\mathcal{Q}} u_{j:\lambda} \log u_{j:\lambda}$ *is the entropy.*

*Proof.* Now, ignoring the condition $u_{j:\lambda} \geq 0$ for now, the Lagrangian of Eq. (18) with dual variables $z_j$ with $j \in \{1 \dots m\}$ can be written as:

$$F(\mathbf{u}, \mathbf{z}) = \beta\left\langle \tilde{\mathbf{u}}^k, \mathbf{u}\right\rangle_F + H(\mathbf{u}) + \sum_j z_j \left(1 - \sum_\lambda u_{j:\lambda}\right)\;. \tag{19}$$

Note that the objective function is multiplied by $\beta > 0$. Now, differentiating $F(\mathbf{u}, \mathbf{z})$ with respect to $\mathbf{u}$ and setting the derivatives to zero:

$$\frac{\partial F}{u_{j:\lambda}} = \beta\,\tilde{u}_{j:\lambda}^k - 1 - \log u_{j:\lambda} - z_j = 0\;, \tag{20}$$
$$\log u_{j:\lambda} = -1 - z_j + \beta\,\tilde{u}_{j:\lambda}^k\;,$$
$$u_{j:\lambda} = e^{-1-z_j}\, e^{\beta\tilde{u}_{j:\lambda}^k}\;.$$

Since $\sum_\mu u_{j:\mu} = 1$,

$$\sum_\mu u_{j:\mu} = 1 = \sum_\mu e^{-1-z_j}\, e^{\beta\tilde{u}_{j:\mu}^k}\;, \tag{21}$$
$$e^{-1-z_j} = \frac{1}{\sum_\mu e^{\beta\tilde{u}_{j:\mu}^k}}\;.$$

Substituting in Eq. (21),

$$u_{j:\lambda} = \frac{e^{\beta\tilde{u}_{j:\lambda}^k}}{\sum_\mu e^{\beta\tilde{u}_{j:\mu}^k}}\;. \tag{22}$$

Note that, $u_{j:\lambda} \geq 0$ for all $j \in \{1 \dots m\}$ and $\lambda \in \mathcal{Q}$, and therefore, $\mathbf{u}$ satisfies Eq. (18) which is exactly the softmax update (17). Hence, the proof is complete. □

## C. 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 [44]. Furthermore, for background on Markov Random Field (MRF), we refer the reader to the Chapter 2 of [2]. In this section, we use the notations from the main paper and highlight the similarities wherever possible.

**Markov Random Field.** Let $\mathcal{W} = \{W_1, \ldots, W_m\}$ 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 MRF can be written as:

$$L(\mathbf{w}) = \sum_{C \in \mathcal{C}} L_C(\mathbf{w}) \,, \tag{23}$$

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:

$$P(\mathbf{w}) = \frac{1}{Z} e^{-L(\mathbf{w})} \,, \tag{24}$$

where the normalization constant $Z$ is usually referred to as the partition function. From Hammersley-Clifford theorem, for the factorization given in Eq. (23), 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. (24) 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 $\lambda$. Therefore, the distribution $U$ can be represented using the variables $\mathbf{u} \in \mathcal{S}$, where $\mathcal{S}$ is defined as:

$$\mathcal{S} = \left\{ \mathbf{u} \;\middle|\; \begin{array}{ll} \sum_\lambda u_{j:\lambda} = 1, & \forall\, j \\ u_{j:\lambda} \geq 0, & \forall\, j, \lambda \end{array} \right\} . \tag{25}$$

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

$$\begin{aligned} \mathrm{KL}(\mathrm{U}\|\mathrm{P}) &= \sum_{\mathbf{w} \in \mathcal{Q}^m} U(\mathbf{w}) \log \frac{U(\mathbf{w})}{P(\mathbf{w})} \,, \qquad (26)\\ &= \sum_{\mathbf{w} \in \mathcal{Q}^m} U(\mathbf{w}) \log U(\mathbf{w}) - \sum_{\mathbf{w} \in \mathcal{Q}^m} U(\mathbf{w}) \log P(\mathbf{w}) \,, \\ &= -H(U) - \sum_{\mathbf{w} \in \mathcal{Q}^m} U(\mathbf{w}) \log \frac{e^{-L(\mathbf{w})}}{Z} \,, \quad \text{Eq. (24)} \,, \\ &= -H(U) + \sum_{\mathbf{w} \in \mathcal{Q}^m} U(\mathbf{w}) L(\mathbf{w}) + \log Z \,. \end{aligned}$$

Here, $H(U)$ denotes the entropy of the fully-factorized distribution, which is exactly the one used in Theorem D.1. Specifically,

$$H(U) = H(\mathbf{u}) = -\sum_{j=1}^{m} \sum_{\lambda \in \mathcal{Q}} u_{j:\lambda} \log u_{j:\lambda} \,. \tag{27}$$

Furthermore, in Eq. (26), since $Z$ is a constant, it can be removed from the minimization. Hence the final mean-field objective can be written as:

$$\begin{aligned} \min_U F(U) &:= \sum_{\mathbf{w} \in \mathcal{Q}^m} U(\mathbf{w}) L(\mathbf{w}) - H(U) \,, \qquad (28)\\ &= \mathbb{E}_U[L(\mathbf{w})] - H(U) \,, \end{aligned}$$

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,

$$\begin{aligned} E(\mathbf{u}) := \mathbb{E}_U[L(\mathbf{w})] &= \sum_{\mathbf{w} \in \mathcal{Q}^m} U(\mathbf{w}) L(\mathbf{w}) \,, \qquad (29)\\ &= \sum_{\mathbf{w} \in \mathcal{Q}^m} \prod_{j=1}^{m} u_{j:w_j} L(\mathbf{w}) \,. \end{aligned}$$

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

$$\min_{\mathbf{u} \in \mathcal{S}} F(\mathbf{u}) := E(\mathbf{u}) - H(\mathbf{u}) \,. \tag{30}$$

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. (23) (*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 (30) 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. The derivation is very similar to the proof of Lemma B.1, and at iteration $k$, the mean-field update for each $j \in \{1 \ldots m\}$ can be written as:

$$u^{k+1}_{j:\lambda} = \frac{e^{-\frac{\partial E^k}{\partial u_{j:\lambda}}}}{\sum_\mu e^{-\frac{\partial E^k}{\partial u_{j:\mu}}}} \quad \forall\, \lambda \in \mathcal{Q} \,. \tag{31}$$

Here, $\frac{\partial E^k}{\partial u_{j:\lambda}}$ 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 $L(\mathbf{w})$.

## D. Softmax based PGD as Proximal Mean-field

Recall the softmax based PGD update $\mathbf{u}^{k+1} = \operatorname{softmax}(\beta(\mathbf{u}^k - \eta\,\mathbf{g}^k))$ for each $j \in \{1\ldots m\}$ can be written as:

$$u^{k+1}_{j:\lambda} = \frac{e^{\beta\left(u^k_{j:\lambda} - \eta\, g^k_{j:\lambda}\right)}}{\sum_{\mu\in\mathcal{Q}} e^{\beta\left(u^k_{j:\mu} - \eta\, g^k_{j:\mu}\right)}} \quad \forall\,\lambda \in \mathcal{Q}\,. \tag{32}$$

Here, $\eta > 0$, and $\beta > 0$.

**Theorem D.1.** *Let* $\mathbf{u}^{k+1} = \operatorname{softmax}(\beta(\mathbf{u}^k - \eta\,\mathbf{g}^k))$ *be the point from the* softmax *based* PGD *update. Then,*

$$\mathbf{u}^{k+1} = \operatorname*{argmin}_{\mathbf{u}\in\mathcal{S}} \eta\,\mathbb{E}_{\mathbf{u}}\left[\hat{L}^k(\mathbf{w})\right] - \left\langle \mathbf{u}^k, \mathbf{u}\right\rangle_F - \frac{1}{\beta}H(\mathbf{u})\,, \tag{33}$$

*where* $\hat{L}^k(\mathbf{w})$ *is the first-order Taylor approximation of* $L$ *at* $\mathbf{w}^k = \mathbf{u}^k\mathbf{q}$ *and* $\eta > 0$ *is the learning rate.*

*Proof.* We will first prove that $\mathbb{E}_{\mathbf{u}}\left[\hat{L}^k(\mathbf{w})\right] = \left\langle \mathbf{g}^k_{\mathbf{u}}, \mathbf{u}\right\rangle_F + c$ for some constant $c$. From the definition of $\hat{L}^k(\mathbf{w})$,

$$\begin{aligned}\hat{L}^k(\mathbf{w}) &= L(\mathbf{w}^k) + \left\langle \mathbf{g}^k_{\mathbf{w}}, \mathbf{w} - \mathbf{w}^k\right\rangle\,, \\ &= \left\langle \mathbf{g}^k_{\mathbf{w}}, \mathbf{w}\right\rangle + c\,,\end{aligned} \tag{34}$$

where $c$ is a constant that does not depend on $\mathbf{w}$. Now, the expectation over $\mathbf{u}$ can be written as:

$$\begin{aligned}\mathbb{E}_{\mathbf{u}}\left[\hat{L}^k(\mathbf{w})\right] &= \mathbb{E}_{\mathbf{u}}\left[\left\langle \mathbf{g}^k_{\mathbf{w}}, \mathbf{w}\right\rangle\right] + c\,, \\ &= \left\langle \mathbf{g}^k_{\mathbf{w}}, \mathbb{E}_{\mathbf{u}}[\mathbf{w}]\right\rangle + c\,, \\ &= \left\langle \mathbf{g}^k_{\mathbf{w}}, \mathbf{u}\mathbf{q}\right\rangle + c\,, \quad \text{Definition of } \mathbf{u}\,.\end{aligned} \tag{35}$$

We will now show that $\left\langle \mathbf{g}^k_{\mathbf{w}}, \mathbf{u}\mathbf{q}\right\rangle = \left\langle \mathbf{g}^k_{\mathbf{u}}, \mathbf{u}\right\rangle_F$. To see this, let us consider an element $j \in \{1\ldots m\}$,

$$\begin{aligned}g^k_{w_j}\left\langle \mathbf{u}_j, \mathbf{q}\right\rangle &= g^k_{w_j}\left\langle \mathbf{q}, \mathbf{u}_j\right\rangle\,, \\ &= g^k_{w_j}\mathbf{q}^T\mathbf{u}_j\,, \\ &= g^k_{\mathbf{u}_j}\mathbf{u}_j\,, \quad \mathbf{g}^k_{\mathbf{u}} = \mathbf{g}^k_{\mathbf{w}}\,\mathbf{q}^T\,.\end{aligned} \tag{36}$$

From the above equivalence, Eq. (33) can now be written as:

$$\begin{aligned}\mathbf{u}^{k+1} &= \operatorname*{argmin}_{\mathbf{u}\in\mathcal{S}} \eta\left\langle \mathbf{g}^k_{\mathbf{u}}, \mathbf{u}\right\rangle_F - \left\langle \mathbf{u}^k, \mathbf{u}\right\rangle_F - \frac{1}{\beta}H(\mathbf{u})\,, \\ &= \operatorname*{argmax}_{\mathbf{u}\in\mathcal{S}} \left\langle \mathbf{u}^k - \eta\,\mathbf{g}^k_{\mathbf{u}}, \mathbf{u}\right\rangle_F + \frac{1}{\beta}H(\mathbf{u})\,.\end{aligned} \tag{37}$$

Now, from Lemma B.1 we can write $\mathbf{u}^{k+1} = \operatorname{softmax}(\beta(\mathbf{u}^k - \eta\,\mathbf{g}^k))$. Hence, the proof is complete. □

## E. BinaryConnect as Proximal ICM

**Proposition E.1.** Consider BC and 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. if $\eta_{\mathbf{u}} = \eta_{\mathbf{w}}/2$, 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 $j \in \{1\ldots m\}$, the hardmax projection can be written as:

$$\begin{aligned}u^k_{j:-1} &= \begin{cases} 1 & \text{if } \tilde{u}^k_{j:-1} \ge \tilde{u}^k_{j:1} \\ 0 & \text{otherwise}\end{cases}\,, \\ u^k_{j:1} &= 1 - u^k_{j:-1}\,.\end{aligned} \tag{38}$$

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

$$\begin{aligned}\mathbf{u}^k_j\mathbf{q} &= \begin{cases} -1 & \text{if } \tilde{w}^k_j = -1\,\tilde{u}^k_{j:-1} + 1\,\tilde{u}^k_{j:1} \le 0 \\ 1 & \text{otherwise}\end{cases}\,, \\ w^k_j &= \operatorname{sign}(\tilde{w}^k_j)\,.\end{aligned} \tag{39}$$

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,

$$\mathbf{g}^k_{\mathbf{u}} = \mathbf{g}^k_{\mathbf{w}}\,\frac{\partial\mathbf{w}}{\partial\mathbf{u}} = \mathbf{g}^k_{\mathbf{w}}\,\mathbf{q}^T\,. \tag{40}$$

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

$$\begin{aligned}w^k_j &= \operatorname{sign}(\tilde{w}^k_j) = \operatorname{sign}(\tilde{\mathbf{u}}^k_j\,\mathbf{q}) = \operatorname{hardmax}(\tilde{\mathbf{u}}^k_j)\,\mathbf{q}\,, \\ \frac{\partial w_j}{\partial\tilde{\mathbf{u}}_j} &= \frac{\partial\operatorname{sign}}{\partial\tilde{\mathbf{u}}_j} = \frac{\partial\operatorname{sign}}{\partial\tilde{w}_j}\frac{\partial\tilde{w}_j}{\partial\tilde{\mathbf{u}}_j} = \frac{\partial\operatorname{hardmax}}{\partial\tilde{\mathbf{u}}_j}\,\mathbf{q}\,.\end{aligned} \tag{41}$$

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. (40)),

$$\frac{\partial w_j}{\partial\tilde{\mathbf{u}}_j} = \frac{\partial\operatorname{sign}}{\partial\tilde{w}_j}\,\mathbf{q} = \frac{\partial\operatorname{hardmax}}{\partial\tilde{\mathbf{u}}_j}\,\mathbf{q}\,. \tag{42}$$

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

$$\begin{aligned}
\mathbf{g}^k_{\tilde{\mathbf{u}}_j} &= \mathbf{g}^k_{\mathbf{u}_j} \frac{\partial \mathbf{u}_j}{\partial \tilde{\mathbf{u}}_j} = \mathbf{g}^k_{\mathbf{u}_j} \frac{\partial\, \mathrm{hardmax}}{\partial \tilde{\mathbf{u}}_j} \;, && (43)\\
\mathbf{g}^k_{\tilde{\mathbf{u}}_j}\, \mathbf{q} &= \mathbf{g}^k_{\mathbf{u}_j} \frac{\partial\, \mathrm{hardmax}}{\partial \tilde{\mathbf{u}}_j}\, \mathbf{q} \;, \quad \text{multiplying by } \mathbf{q} \;,\\
&= g^k_{w_j}\, \mathbf{q}^T \frac{\partial\, \mathrm{hardmax}}{\partial \tilde{\mathbf{u}}_j}\, \mathbf{q} \;, \quad \text{Eq. (40)} \;,\\
&= g^k_{w_j}\, \mathbf{q}^T \frac{\partial\, \mathrm{sign}}{\partial \tilde{w}_j}\, \mathbf{q} \;, \quad \text{Eq. (42)} \;,\\
&= g^k_{w_j} \frac{\partial\, \mathrm{sign}}{\partial \tilde{w}_j}\, \mathbf{q}^T\, \mathbf{q} \;,\\
&= g^k_{\tilde{w}_j}\, \mathbf{q}^T\, \mathbf{q} \;, \quad \frac{\partial\, \mathrm{sign}}{\partial \tilde{w}_j} = \frac{\partial w_j}{\partial \tilde{w}_j} \;,\\
&= 2\, g^k_{\tilde{w}_j} \;, \quad \mathbf{q} = [-1, 1]^T \;.
\end{aligned}$$

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

$$\begin{aligned}
\tilde{\mathbf{u}}^{k+1} &= \tilde{\mathbf{u}}^k - \eta_{\mathbf{u}}\, \mathbf{g}^k_{\tilde{\mathbf{u}}} \;, && (44)\\
\tilde{\mathbf{u}}^{k+1}\, \mathbf{q} &= \tilde{\mathbf{u}}^k\, \mathbf{q} - \eta_{\mathbf{u}}\, \mathbf{g}^k_{\tilde{\mathbf{u}}}\, \mathbf{q} \;,\\
&= \tilde{\mathbf{w}}^k - \eta_{\mathbf{u}}\, 2\, \mathbf{g}^k_{\tilde{\mathbf{w}}} \;,\\
&= \tilde{\mathbf{w}}^k - \eta_{\mathbf{w}}\, \mathbf{g}^k_{\tilde{\mathbf{w}}} \;,\\
&= \tilde{\mathbf{w}}^{k+1} \;.
\end{aligned}$$

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 $\mathcal{S}$, meaning $\tilde{\mathbf{w}} \in [-1, 1]$ implies $\tilde{\mathbf{u}} \in \mathcal{S}$, 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 E.1 holds, as the assumption $\tilde{\mathbf{w}}^k = \tilde{\mathbf{u}}^k\mathbf{q}$ is still satisfied.

## E.1. Approximate Gradients through Hardmax

In previous works [11, 37], to allow back propagation through the sign function, the straight-through-estimator [19] is used. Precisely, the partial derivative with respect to the sign function is defined as:

$$\frac{\partial\, \mathrm{sign}(r)}{\partial r} := \mathbb{1}[|r| \le 1] \;. \qquad (45)$$

To make use of this, we intend to write the projection function hardmax in terms of the sign function. To this end, from Eq. (38), for each $j \in \{1 \dots m\}$,

$$u^k_{j:-1} = \begin{cases} 1 & \text{if } \tilde{u}^k_{j:-1} - \tilde{u}^k_{j:1} \ge 0 \\ 0 & \text{otherwise} \end{cases} \;, \qquad (46)$$

$$u^k_{j:1} = 1 - u^k_{j:-1} \;. \qquad (47)$$

| 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 $\mathrm{hardmax}(\tilde{\mathbf{u}}^k)$ for each $j$ can be written as:

$$u^k_{j:-1} = \frac{\mathrm{sign}(\tilde{u}^k_{j:-1} - \tilde{u}^k_{j:1}) + 1}{2} \;, \qquad (48)$$

$$u^k_{j:1} = \frac{1 - \mathrm{sign}(\tilde{u}^k_{j:-1} - \tilde{u}^k_{j:1})}{2} \;. \qquad (49)$$

Now, using Eq. (45), we can write:

$$\left.\frac{\partial \mathbf{u}_j}{\partial \tilde{\mathbf{u}}_j}\right|_{\tilde{\mathbf{u}}_j = \tilde{\mathbf{u}}^k_j} = \frac{1}{2} \begin{bmatrix} \mathbb{1}[|\upsilon^k_j| \le 1] & -\mathbb{1}[|\upsilon^k_j| \le 1] \\ -\mathbb{1}[|\upsilon^k_j| \le 1] & \mathbb{1}[|\upsilon^k_j| \le 1] \end{bmatrix} \;, \qquad (50)$$

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

# F. Experimental Details

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

## F.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.

# References

[1] J. Achterhold, J. M. Kohler, A. Schmeink, and T. Genewein. Variational network quantization. *ICLR*, 2018. 7

[2] T. Ajanthan. *Optimization of Markov random fields in computer vision*. PhD thesis, Australian National University, 2017. 1, 9

| 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 |
| $\rho$ (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 |
| $\rho$ (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,<br>every 10k | - | 30k | 30k | 30k | 30k | 30k | 30k | 30k - 80k,<br>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 |
| $\rho$ (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, $\rho$ denotes the growth rate of $\beta$ (refer Algorithm 1) and $\beta$ is multiplied by $\rho$ every 100 iterations.*

[3] T. Ajanthan, A. Desmaison, R. Bunel, M. Salzmann, P. H. S. Torr, and M. P. Kumar. Efficient linear programming for dense CRFs. *CVPR*, 2017. 3

[4] S.-I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 1998. 4

[5] Y. Bai, Y.-X. Wang, and E. Liberty. Proxquant: Quantized neural networks via proximal operators. *ICLR*, 2019. 7, 8

[6] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society.*, 1986. 2, 6

[7] A. Blake, P. Kohli, and C. Rother. *Markov random fields for vision and image processing*. Mit Press, 2011. 1

[8] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2009. 5

[9] S. Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 2015. 5

[10] C. Chekuri, S. Khanna, J. Naor, and L. Zosin. A linear programming formulation and approximation algorithms for the metric labeling problem. *SIAM Journal on Discrete Mathematics*, 2004. 2, 3

[11] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. *NIPS*, 2015. 1, 2, 6, 7, 8, 12

[12] P. K. Dokania and P. K. Mudigonda. Parsimonious labeling. *ICCV*, 2015. 1

[13] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2011. 4

[14] S. K. Esser, R. Appuswamy, P. A. Merolla, J. V. Arthur, and D. S. Modha. Backpropagation for energy-efficient neuromorphic computing. *NIPS*, 2015. 6

[15] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 1956. 3

[16] Y. Gong, L. Liu, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014. 7

[17] Y. Guo. A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*, 2018. 7, 8

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, 2016. 7

[19] G. Hinton. Neural networks for machine learning. *Coursera, video lectures*, 2012. 6, 12

[20] L. Hou, Q. Yao, and J. T. Kwok. Loss-aware binarization of deep networks. *ICLR*, 2017. 6

[21] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get m for free. *ICLR*, 2017. 7

[22] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. *NIPS*, 2016. 6

[23] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *JMLR*, 2017. 1, 6, 7

[24] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015. 7

[25] R. Kindermann and J. L. Snell. *Markov Random Fields and Their Applications*. American Mathematical Society, 1980. 2

[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 3

[27] J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: metric labeling and Markov random fields. *Journal of the ACM*, 2002. 3

[28] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *PAMI*, 2004. 2

[29] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate Frank-Wolfe optimization for structural SVMs. *ICML*, 2012. 3

[30] N. Lee, T. Ajanthan, and P. H. S. Torr. SNIP: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018. 7

[31] C. Louizos, K. Ullrich, and M. Welling. Bayesian compression for deep learning. *NIPS*, 2017. 7

[32] A. Martins and R. Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. *ICML*, 2016. 4

[33] P. K. Mudigonda. *Combinatorial and convex optimization for probabilistic models in computer vision*. PhD thesis, Oxford Brookes University, 2008. 1

[34] G. L. Nemhauser and L. A. Wolsey. *Integer programming and combinatorial optimization*. Springer, 1988. 2

[35] N. Parikh and S. P. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 2014. 3

[36] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. 2017. 7

[37] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnornet: Imagenet classification using binary convolutional neural networks. *ECCV*, 2016. 1, 2, 7, 8, 12

[38] P. Ravikumar, A. Agarwal, and M. J. Wainwright. Message-passing for graph-structured linear programs: proximal projections, convergence and rounding schemes. *ICML*, 2008. 5

[39] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 1951. 3

[40] L. Rosasco, S. Villa, and B. C. Vũ. Convergence of stochastic proximal gradient algorithm. *arXiv preprint arXiv:1403.5074*, 2014. 3

[41] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015. 7

[42] D. Soudry, I. Hubara, and R. Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. *NIPS*, 2018. 7

[43] O. Veksler. *Efficient graph-based energy minimization methods in computer vision*. PhD thesis, Cornell University New York, USA, 1999. 1

[44] M. J. Wainwright, M. I. Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 2008. 1, 2, 5, 9

[45] P. Yin, S. Zhang, J. Lyu, S. Osher, Y. Qi, and J. Xin. Binaryrelax: A relaxation approach for training deep neural networks with quantized weights. *SIIMS*, 2018. 7