PRINCIPLED WEIGHT INITIALIZATION FOR HYPERNETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Hypernetworks are meta neural networks that generate weights for a main neural network in an end-to-end differentiable manner. Despite extensive applications ranging from multi-task learning to Bayesian deep learning, the problem of optimizing hypernetworks has not been studied to date. We observe that classical weight initialization methods like Glorot & Bengio (2010) and He et al. (2015), when applied directly on a hypernet, fail to produce weights for the mainnet in the correct scale. We develop principled techniques for weight initialization in hypernets, and show that they lead to more stable mainnet weights, lower training loss, and faster convergence.

1 INTRODUCTION

Meta-learning describes a broad family of techniques in machine learning that deals with the problem of learning to learn. An emerging branch of meta-learning involves the use of *hypernetworks*, which are meta neural networks that generate the weights of a main neural network to solve a given task in an end-to-end differentiable manner. Hypernetworks were originally introduced by Ha et al. (2016) as a way to induce weight-sharing and achieve model compression by training the same meta network to learn the weights belonging to different layers in the main network. Since then, hypernetworks have found numerous applications including but not limited to: weight pruning (Liu et al., 2019), neural architecture search (Brock et al., 2017; Zhang et al., 2018), Bayesian neural networks (Krueger et al., 2017; Ukai et al., 2018; Pawlowski et al., 2017; Henning et al., 2018; Deutsch et al., 2019), multi-task learning (Pan et al., 2018; Shen et al., 2017; Klocek et al., 2019; Serrà et al., 2019; Meyerson & Miikkulainen, 2019), continual learning (von Oswald et al., 2019), generative models (Suarez, 2017; Ratzlaff & Fuxin, 2019), ensemble learning (Kristiadi & Fischer, 2019), hyperparameter optimization (Lorraine & Duvenaud, 2018), and adversarial defense (Sun et al., 2017).

Despite the intensified study of applications of hypernetworks, the problem of optimizing them to this day remains significantly understudied. Given the lack of principled approaches to training hypernetworks, prior work in the area is mostly limited to ad-hoc approaches based on trial and error (c.f. Section 3). For example, it is common to initialize the weights of a hypernetwork by sampling a "small" random number. Nonetheless, these ad-hoc methods do lead to successful hypernetwork training primarily due to the use of the Adam optimizer (Kingma & Ba, 2014), which has the desirable property of being invariant to the scale of the gradients. However, even Adam will not work if the loss diverges (i.e. integer overflow) at initialization, which will happen in sufficiently big models. The normalization of badly scaled gradients also results in noisy training dynamics where the loss function suffers from bigger fluctuations during training compared to vanilla stochastic gradient descent (SGD). Wilson et al. (2017) showed that while adaptive optimizers like Adam may exhibit lower training error, they fail to generalize as well to the test set as non-adaptive gradient methods. Moreover, Adam incurs a computational overhead and requires 3X the amount of memory for the gradients compared to vanilla SGD.

Small random number sampling is reminiscent of early neural network research (Rumelhart et al., 1986) before the advent of classical weight initialization methods like Xavier init (Glorot & Bengio, 2010) and Kaiming init (He et al., 2015). Since then, a big lesson learned by the neural network optimization community is that architecture specific initialization schemes are important to the robust training of deep networks, as shown recently in the case of residual networks (Zhang et al., 2019). In fact, weight initialization for hypernetworks was recognized as an outstanding open prob-

lem by prior work (Deutsch et al., 2019) that had questioned the suitability of classical initialization methods for hypernetworks.

Our results We show that when classical methods are used to initialize the weights of hypernetworks, they fail to produce mainnet weights in the correct scale, leading to exploding activations and losses. This is because classical network weights transform one layer's activations into another, while hypernet weights have the added function of transforming the hypernet's activations into the mainnet's weights. Our solution is to develop principled techniques for weight initialization in hypernetworks based on variance analysis. The hypernet case poses unique challenges. For example, in contrast to variance analysis for classical networks, the case for hypernetworks can be asymmetrical between the forward and backward pass. The asymmetry arises when the gradient flow from the mainnet into the hypernet is affected by the biases, whereas in general, this does not occur for gradient flow in the mainnet. This underscores again why architecture specific initialization schemes are essential. We show both theoretically and experimentally that our methods produce hypernet weights in the correct scale. Proper initialization mitigates exploding activations and gradients or the need to depend on Adam. Our experiments reveal that it leads to more stable mainnet weights, lower training loss, and faster convergence.

Section 2 briefly covers the relevant technical preliminaries and Section 3 reviews problems with the ad-hoc methods currently deployed by hypernetwork practitioners. We derive novel weight initialization formulae for hypernetworks in Section 4, empirically evaluate our proposed methods in Section 5, and finally conclude in Section 6.

2 Preliminaries

Definition. *A **hypernetwork** is a meta neural network H with its own parameters ϕ that generates the weights of a main network θ from some embedding e in a differentiable manner: $\theta = H_\phi(e)$. Unlike a classical network, in a hypernetwork, the weights of the main network are not model parameters. Thus the gradients $\Delta\theta$ have to be further backpropagated to the weights of the hypernetwork $\Delta\phi$, and then trained via gradient descent $\phi_{t+1} = \phi_t - \lambda\Delta\phi_t$.*

This fundamental difference suggests that conventional knowledge about neural networks may not apply directly to hypernetworks and novel ways of thinking about weight initialization, optimization dynamics and architecture design for hypernetworks are sorely needed.

2.1 Ricci Calculus

We propose the use of Ricci calculus, as opposed to the more commonly used matrix calculus, as a suitable mathematical language for thinking about hypernetworks. Ricci calculus is useful because it allows us to reason about the derivatives of higher-order tensors with notational ease. For readers not familiar with the index-based notation of Ricci calculus, please refer to Laue et al. (2018) for a good introduction to the topic written from a machine learning perspective.

For a general nth-order tensor $T^{i_1,\ldots,i_n}$, we use d_{i_k} to refer to the dimension of the index set that i_k is drawn from. We include explicit summations where the relevant expressions might be ambiguous, and use Einstein summation convention otherwise. We use square brackets to denote different layers for added clarity, so for example $W[t]$ denotes the t-th weight layer.

2.2 Xavier Initialization

Glorot & Bengio (2010) derived weight initialization formulae for a feedforward neural network by conducting a variance analysis over activations and gradients. For a linear layer $y^i = W^i_j x^j + b^i$, suppose we make the following **Xavier Assumptions** at initialization: **(1)** The W^i_j, x^j, and b^i are all independent of each other. **(2)** $\forall i, j : \mathbb{E}[W^i_j] = 0$. **(3)** $\forall j : \mathbb{E}[x^j] = 0$. **(4)** $\forall i : b^i = 0$.

Then, $\mathbb{E}[y^i] = 0$ and $\mathrm{Var}(y^i) = \mathrm{d}_j \mathrm{Var}(W^i_j)\mathrm{Var}(x^j)$. To keep the variance of the output and input activations the same, i.e. $\mathrm{Var}(y^i) = \mathrm{Var}(x^j)$, we have to sample W^i_j from a distribution whose variance is equal to the reciprocal of the fan-in: $\mathrm{Var}(W^i_j) = \frac{1}{\mathrm{d}_j}$.

If analogous assumptions hold for the backward pass, then to keep the variance of the output and input gradients the same, we have to sample W_j^i from a distribution whose variance is equal to the reciprocal of the fan-out: $\text{Var}(W_j^i) = \frac{1}{\text{d}_i}$.

Thus, the forward pass and backward pass result in symmetrical formulae. Glorot & Bengio (2010) proposed an initialization based on their harmonic mean: $\text{Var}(W_j^i) = \frac{2}{\text{d}_j + \text{d}_i}$.

In general, a feedforward network is non-linear, so these assumptions are strictly invalid. But symmetric activation functions with unit derivative at 0 results in a roughly linear regime at initialization.

2.3 Kaiming Initialization

He et al. (2015) extended Glorot & Bengio (2010)'s analysis by looking at the case of ReLU activation functions, i.e. $y^i = W_j^i \,\text{ReLU}(x^j) + b^i$. We can write $z^j = \text{ReLU}(x^j)$ to get

$$\text{Var}(y^i) = \sum_j \mathbb{E}[(z^j)^2]\text{Var}(W_j^i) = \sum_j \frac{1}{2}\,\mathbb{E}[(x^j)^2]\text{Var}(W_j^i) = \frac{1}{2}\,\text{d}_j \text{Var}(W_j^i)\text{Var}(x^j).$$

This results in an extra factor of 2 in the variance formula. W_j^i have to be symmetric around 0 to enforce Xavier Assumption 3 as the activations and gradients propagate through the layers. He et al. (2015) argued that both the forward or backward version of the formula can be adopted, since the activations or gradients will only be scaled by a depth-independent factor. For convolutional layers, we have to further divide the variance by the size of the receptive field.

'Xavier init' and 'Kaiming init' are terms that are sometimes used interchangeably. Where there might be confusion, we will refer to the forward version as fan-in init, the backward version as fan-out init, and the harmonic mean version as harmonic init.

3 Review of Current Methods

In the seminal Ha et al. (2016) paper, the authors identified two distinct classes of hypernetworks: dynamic (for recurrent networks) and static (for convolutional networks). They proposed Orthogonal init (Saxe et al., 2013) for the dynamic class, but omitted discussion of initialization for the static class. The static class has since proven to be the dominant variant, covering all kinds of non-recurrent networks (not just convolutional), and thus will be the central object of our investigation.

Through an extensive literature and code review, we found that hypernet practitioners mostly depend on the Adam optimizer, which is invariant to and normalizes the scale of gradients, for training and resort to one of four weight initialization methods:

- M1 Xavier or Kaiming init (as found in Pawlowski et al. (2017); Serrà et al. (2019); von Oswald et al. (2019); Balazevic et al. (2018)).
- M2 Small random values (as found in Krueger et al. (2017); Lorraine & Duvenaud (2018)).
- M3 Kaiming init, but with the output layer scaled by $\frac{1}{10}$ (as found in Ukai et al. (2018)).
- M4 Kaiming init, but with the hypernet embedding set to be a suitably scaled constant (as found in Meyerson & Miikkulainen (2019)).

M1 uses classical neural network initialization methods to initialize hypernetworks. This fails to produce weights for the main network in the correct scale. Consider the following illustrative example of a one-layer linear hypernet generating a linear mainnet with $T+1$ layers, given embeddings sampled from a standard normal distribution and weights sampled entry-wise from a zero-mean distribution. We leave the biases out for now.

$$\begin{gathered} x[t+1]^{i_{t+1}} = W[t]_{i_t}^{i_{t+1}} x[t]^{i_t}, \qquad W[t]_{i_t}^{i_{t+1}} = H[t]_{i_t k_t}^{i_{t+1}} e[t]^{k_t}, \qquad 1 \le t \le T. \\ \text{Var}(x[T+1]^{i_{t+1}}) = \text{Var}(x[1]^{i_1}) \prod_{t=1}^{T} \text{d}_{i_t} \text{Var}(W[t]_{i_t}^{i_{t+1}}) = \text{Var}(x[1]^{i_1}) \prod_{t=1}^{T} \text{d}_{i_t} \text{d}_{k_t} \text{Var}(H[t]_{i_t k_t}^{i_{t+1}}). \end{gathered} \tag{1}$$

In this case, if the variance of the weights in the hypernet $\text{Var}(H[t]_{i_t k_t}^{i_{t+1}})$ is equal to the reciprocal of the fan-in d_{k_t}, then the variance of the activations $\text{Var}(x[T+1]^{i_{t+1}}) = \prod_{t=1}^{T} \text{d}_{i_t}$ explodes. If

it is equal to the reciprocal of the fan-out $\mathrm{d}_{i_{t+1}}$, then the activation variance $\mathrm{Var}(x[T+1]^{i_{t+1}}) = \prod_{t=1}^{T} \frac{\mathrm{d}_{k_t}}{\mathrm{d}_{i_t+1}}$ is likely to vanish, since the size of the embedding vector is typically small relatively to the width of the mainnet weight layer being generated.

Where the fan-in is of a different scale than the fan-out, the harmonic mean has a scale close to that of the smaller number. Therefore, the fan-in, fan-out, and harmonic variants of Xavier and Kaiming init will all result in activations and gradients that scale exponentially with the depth of the mainnet.

M2 and M3 introduce additional hyperparameters into the model, and the ad-hoc manner in which they work is reminiscent of pre deep learning neural network research, before the introduction of classical initialization methods like Xavier and Kaiming init. This ad-hoc manner is not only inelegant and consumes more compute, but will likely fail for deeper and more complex hypernetworks.

M4 proposes to set the embeddings $e[t]^{k_t}$ to a suitable constant ($\mathrm{d}_{i_t}^{-1/2}$ in this case), such that both $W[t]_{i_t}^{i_{t+1}}$ and $H[t]_{i_t k_t}^{i_{t+1}}$ can seem to be initialized with the same variance as Kaiming init. This ensures that the variance of the activations in the mainnet are preserved through the layers, but the restrictions on the embeddings might not be desirable in many applications.

Luckily, the fix appears simple — set $\mathrm{Var}(H[t]_{i_t k_t}^{i_{t+1}}) = \frac{1}{\mathrm{d}_{i_t}\mathrm{d}_{k_t}}$. This results in the variance of the generated weights in the mainnet $\mathrm{Var}(W[t]_{i_t}^{i_{t+1}}) = \frac{1}{\mathrm{d}_{i_t}}$ resembling conventional neural networks initialized with fan-in init. This suggests a general hypernet weight initialization strategy: initialize the weights of the hypernet such that they approximate weights coming from classical neural network initialization. We elaborate on and generalize this intuition in Section 4.

4 HYPERFAN INITIALIZATION

Most hypernetwork architectures use a linear output layer so that gradients can pass from the mainnet into the hypernet directly without any non-linearities. We make use of this fact in developing methods called *hyperfan-in init* and *hyperfan-out init* for hypernetwork weight initialization based on the principle of variance analysis.

4.1 HYPERFAN-IN

Proposition. *Suppose a hypernetwork comprises a linear output layer. Then, the variance between the input and output activations of a linear layer in the mainnet $y^i = W_j^i x^j + b^i$ can be preserved using fan-in init in the hypernetwork with appropriately scaled output layers.*

Case 1. The hypernet generates the weights but not the biases of the mainnet. The bias in the mainnet is initialized to zero. We can write the weight generation in the form $W_j^i = H_{jk}^i h(e)^k + \beta_j^i$ where h computes all but the last layer of the hypernet and (H, β) form the output layer. We make the following **Hyperfan Assumptions** at initialization: **(1)** Xavier assumptions hold for all the layers in the hypernet. **(2)** The H_{jk}^i, $h(e)^k$, β_j^i, x^j, and b^i are all independent of each other. **(3)** $\forall i, j, k : \mathbb{E}[H_{jk}^i] = 0$. **(4)** $\mathbb{E}[x^j] = 0$. **(5)** $\forall i : b^i = 0$.

Use fan-in init to initialize the weights for h. Then, $\mathrm{Var}(h(e)^k) = \mathrm{Var}(e^l)$. If we initialize H with the formula $\mathrm{Var}(H_{jk}^i) = \frac{1}{\mathrm{d}_j \mathrm{d}_k \mathrm{Var}(e^l)}$ and β with zeros, we arrive at $\mathrm{Var}(W_j^i) = \frac{1}{\mathrm{d}_j}$, which is the formula for fan-in init in the mainnet. The Hyperfan assumptions imply the Xavier assumptions hold in the mainnet, thus preserving the input and output activations.

$$\begin{aligned}\mathrm{Var}(y^i) &= \sum_j \mathrm{Var}(W_j^i)\mathrm{Var}(x^j) = \sum_j \sum_k \mathrm{Var}(H_{jk}^i)\mathrm{Var}(h(e)^k)\mathrm{Var}(x^j) \\ &= \sum_j \sum_k \frac{1}{\mathrm{d}_j \mathrm{d}_k \mathrm{Var}(e^l)} \mathrm{Var}(e^l)\mathrm{Var}(x^j) = \mathrm{Var}(x^j).\end{aligned} \tag{2}$$

Case 2. The hypernet generates both the weights and biases of the mainnet. We can write the weight and bias generation in the form $W_j^i = H_{jk}^i h(e[1])^k + \beta_j^i$ and $b^i = G_l^i g(e[2])^l + \gamma^i$ respectively, where h and g compute all but the last layer of the hypernet, and (H, β) and (G, γ)

form the output layers. We modify Hyperfan Assumption 2 so it includes G^i_l, $g(e[2])^l$, and γ^i, and further assume $\text{Var}(x^j) = 1$, which holds at initialization with the common practice of data standardization.

Use fan-in init to initialize the weights for h and g. Then, $\text{Var}(h(e[1])^k) = \text{Var}(e[1]^m)$ and $\text{Var}(g(e[2])^l) = \text{Var}(e[2]^n)$. If we initialize H with the formula $\text{Var}(H^i_{jk}) = \frac{1}{2\text{d}_j\text{d}_k\text{Var}(e[1]^m)}$, G with the formula $\text{Var}(G^i_l) = \frac{1}{2\text{d}_l\text{Var}(e[2]^n)}$, and β, γ with zeros, then the input and output activations in the mainnet can be preserved.

$$\begin{aligned}
\text{Var}(y^i) &= \sum_j \big[\text{Var}(W^i_j)\text{Var}(x^j)\big] + \text{Var}(b^i) \\
&= \sum_j \big[\sum_k \text{Var}(H^i_{jk})\text{Var}(h(e[1])^k)\text{Var}(x^j)\big] + \sum_l \text{Var}(G^i_l)\text{Var}(g(e[2])^l) \\
&= \sum_j \big[\sum_k \frac{1}{2\text{d}_j\text{d}_k\text{Var}(e[1]^m)}\text{Var}(e[1]^m)\text{Var}(x^j)\big] + \sum_l \frac{1}{2\text{d}_l\text{Var}(e[2]^n)}\text{Var}(e[2]^n) \\
&= \frac{1}{2}\text{Var}(x^j) + \frac{1}{2} = \text{Var}(x^j).
\end{aligned} \tag{3}$$

If we initialize G^i_j to zeros, then its contribution to the variance will increase during training, causing exploding activations in the mainnet. Hence, we prefer to introduce a factor of $1/2$ to divide the variance between the weight and bias generation, where the variance of each component is allowed to either decrease or increase during training. This becomes a problem if the variance of the activations in the mainnet deviates too far away from 1, but we found that it works well in practice.

4.2 HYPERFAN-OUT

A similar derivation can be done for the backward pass using analogous assumptions on gradients flowing

$$\begin{aligned}
\text{in the mainnet:} \quad & \frac{\partial L}{\partial x[t]^{i_t}} = \frac{\partial L}{\partial x[t+1]^{i_{t+1}}} W[t]^{i_{t+1}}_{i_t}, \\
\text{through mainnet weights:} \quad & \frac{\partial L}{\partial W[t]^{i_{t+1}}_{i_t}} = \frac{\partial L}{\partial x[t+1]^{i_{t+1}}} x[t]^{i_t}, \frac{\partial L}{\partial h[t](e)^{k_t}} = \frac{\partial L}{\partial W[t]^{i_{t+1}}_{i_t}} H[t]^{i_{t+1}}_{i_t k_t}, \\
\text{and through mainnet biases:} \quad & \frac{\partial L}{\partial b[t]^{i_{t+1}}} = \frac{\partial L}{\partial x[t+1]^{i_{t+1}}}, \frac{\partial L}{\partial g[t](e)^{l_t}} = \frac{\partial L}{\partial b[t]^{i_{t+1}}} G[t]^{i_{t+1}}_{l_t}.
\end{aligned} \tag{4}$$

If we initialize the output layer H with the analogous hyperfan-out formula $\text{Var}(H[t]^{i_{t+1}}_{i_t k_t}) = \frac{1}{\text{d}_{i_{t+1}}\text{d}_{k_t}\text{Var}(e^{k_t})}$ and the rest of the hypernet with fan-in init, then we can preserve input and output gradients on the mainnet: $\text{Var}(\frac{\partial L}{\partial x[t]^{i_t}}) = \text{Var}(\frac{\partial L}{\partial x[t+1]^{i_{t+1}}})$. However, note that the gradients will shrink when flowing from the mainnet to the hypernet: $\text{Var}(\frac{\partial L}{\partial h[t](e)^{k_t}}) = \frac{\text{d}_{i_t}}{\text{d}_{k_t}\text{Var}(e^{k_t})}\text{Var}(\frac{\partial L}{\partial W[t]^{i_{t+1}}_{i_t}})$, and scaled by a depth-independent factor due to the use of fan-in rather than fan-out init.

In the classical case, the forward version (fan-in init) and the backward version (fan-out init) are symmetrical. This remains true for hypernets if they only generated the weights of the mainnet. However, if they were to also generate the biases, then the symmetry no longer holds, since the biases do not affect the gradient flow in the mainnet but they do so for the hypernet (c.f. Equation 4). Nevertheless, we can initialize G so that it helps hyperfan-out init preserve activation variance on the forward pass as much as possible (keeping the assumption that $\text{Var}(x^j) = 1$ as before):

$$\begin{aligned}
\text{Var}(y^i) &= \sum_j \big[\text{Var}(W^i_j x^j)\big] + \text{Var}(b^i) \\
&= \text{d}_j\text{d}_k\text{Var}(e[1]^m)\text{Var}(H[\text{hyperfan-out}]^i_{jk})\text{Var}(x^j) + \text{d}_l\text{Var}(e[2]^n)\text{Var}(G^i_l) \\
&= \text{d}_j\text{d}_k\text{Var}(e[1]^m)\text{Var}(H[\text{hyperfan-in}]^i_{jk})\text{Var}(x^j) \\
\implies \text{Var}(G^i_l) &= \frac{(1 - \frac{\text{d}_j}{\text{d}_i})}{\text{d}_l\text{Var}(e[2]^n)}.
\end{aligned} \tag{5}$$

We summarize the variance formulae for hyperfan-in and hyperfan-out init in Table 1. It is not uncommon to re-use the same hypernet to generate different parts of the mainnet, as was originally done in Ha et al. (2016). We discuss this case in more detail in Appendix Section A.

Table 1: Hyperfan-in and Hyperfan-out Variance Formulae for $W_j^i = H_{jk}^i h(e[1])^k + \beta_j^i$. If $y^i = \text{ReLU}(W_j^i x^j + b^i)$, then $\mathbb{1}_{\text{ReLU}} = 1$, else if $y^i = W_j^i x^j + b^i$, then $\mathbb{1}_{\text{ReLU}} = 0$. If $b^i = G_l^i g(e[2])^l + \gamma^i$, then $\mathbb{1}_{\text{HBias}} = 1$, else if $b^i = 0$, then $\mathbb{1}_{\text{HBias}} = 0$. We initialize h and g with fan-in init, and $\beta_j^i, \gamma^i = 0$. For convolutional layers, we have to further divide $\text{Var}(H_{jk}^i)$ by the size of the receptive field. Uniform init: $X \sim \mathcal{U}(-\sqrt{3\text{Var}(X)}, \sqrt{3\text{Var}(X)})$. Normal init: $X \sim \mathcal{N}(0, \text{Var}(X))$.

Initialization	**Variance Formula**	**Initialization**	**Variance Formula**
Hyperfan-in	$\text{Var}(H_{jk}^i) = \frac{2^{\mathbb{1}_{\text{ReLU}}}}{2^{\mathbb{1}_{\text{HBias}}} \text{d}_j \text{d}_k \text{Var}(e[1]^m)}$	Hyperfan-out	$\text{Var}(H_{jk}^i) = \frac{2^{\mathbb{1}_{\text{ReLU}}}}{\text{d}_i \text{d}_k \text{Var}(e[1]^m)}$
Hyperfan-in	$\text{Var}(G_l^i) = \frac{2^{\mathbb{1}_{\text{ReLU}}}}{2\text{d}_l \text{Var}(e[2]^n)}$	Hyperfan-out	$\text{Var}(G_l^i) = \max(\frac{2^{\mathbb{1}_{\text{ReLU}}}(1-\frac{\text{d}_j}{\text{d}_i})}{\text{d}_l \text{Var}(e[2]^n)}, 0)$

5 EXPERIMENTS

We evaluated our proposed methods on four sets of experiments involving different use cases of hypernetworks: feedforward networks, continual learning, convolutional networks, and bayesian neural networks. In all cases, we optimize with vanilla SGD and sample from the uniform distribution according to the variance formula given by the init method. More experimental details can be found in Appendix Section B.

5.1 FEEDFORWARD NETWORKS ON MNIST

As an illustrative first experiment, we train a feedforward network with five hidden layers (500 hidden units), a hyperbolic tangent activation function, and a softmax output layer, on MNIST across four different settings: (1) a classical network with Xavier init, (2) a hypernet with Xavier init that generates the weights of the mainnet, (3) a hypernet with hyperfan-in init that generates the weights of the mainnet, (4) and a hypernet with hyperfan-out init that generates the weights of the mainnet.

The use of hyperfan init methods on a hypernetwork reproduces mainnet weights similar to those that have been trained from Xavier init on a classical network, while the use of Xavier init on a hypernetwork causes exploding activations right at the beginning of training (see Figure 1). Observe in Figure 2 that when the hypernetwork is initialized in the proper scale, the magnitude of generated weights stabilizes quickly. This in turn leads to a more stable training regime, as seen in Figure 3. More visualizations of the activations and gradients of both the mainnet and hypernet can be viewed in Appendix Section B.1. Qualitatively similar observations were made when we replaced the activation function with ReLU and Xavier with Kaiming init, with Kaiming init leading to even bigger activations at initialization.

Suppose now the hypernet generates both the weights and biases of the mainnet instead of just the weights. We found that this architectural change leads the hyperfan init methods to take more time (but still less than Xavier init), to generate stable mainnet weights (c.f. Figure 25 in the Appendix).

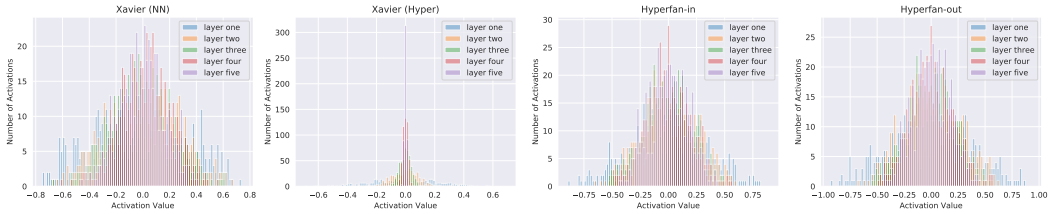

Figure 1: Mainnet Activations before the Start of Training on MNIST.

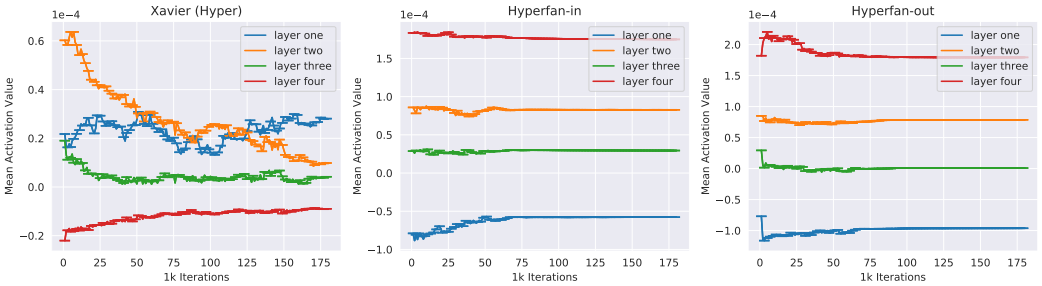

Figure 2: Evolution of Hypernet Output Layer Activations during Training on MNIST. Xavier init results in unstable mainnet weights throughout training, while hyperfan-in and hyperfan-out init result in mainnet weights that stabilize quickly.

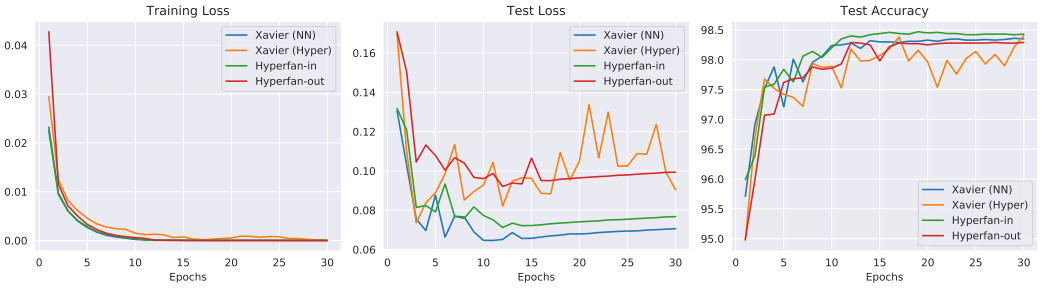

Figure 3: Loss and Test Accuracy Plots on MNIST.

5.2 Continual Learning

Continual learning solves the problem of learning tasks in sequence without forgetting prior tasks. von Oswald et al. (2019) used a hypernetwork to learn embeddings for each task as a way to efficiently regularize the training process to prevent catastrophic forgetting. We compare different initialization schemes on their hypernetwork implementation, which generates the weights and biases of a ReLU mainnet with two hidden layers to solve a sequence of three regression tasks.

In Figure 4, we plot the training loss averaged over 15 different runs, with the shaded area showing the standard error. We observe that the hyperfan methods produce smaller training losses at initialization and during training, eventually converging to a smaller loss for each task.

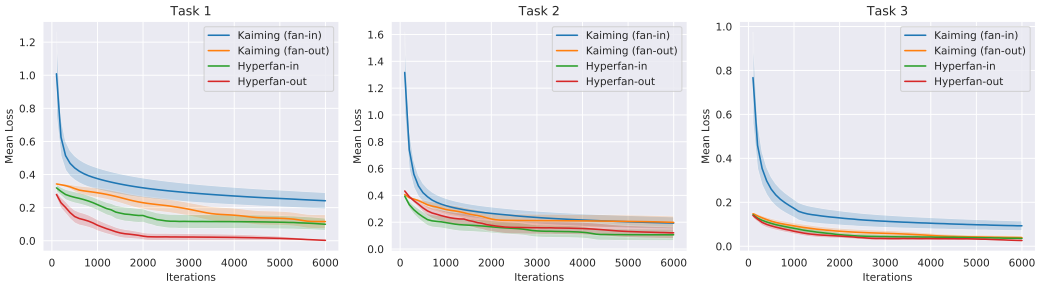

Figure 4: Continual Learning Loss on a Sequence of Regression Tasks.

5.3 CONVOLUTIONAL NETWORKS ON CIFAR-10

Ha et al. (2016) applied a hypernetwork on a convolutional network for image classification on CIFAR-10. We note that our initialization methods do not handle residual connections, which were in their chosen mainnet architecture and are important topics for future study. Instead, we implemented their hypernetwork architecture on a mainnet with the All Convolutional Net architecture (Springenberg et al., 2014) that is composed of convolutional layers and ReLU activation functions.

After searching through a dense grid of learning rates, we failed to enable the fan-in version of Kaiming init to train even with very small learning rates. The fan-out version managed to begin delayed training, starting from around epoch 270 (see Figure 5). By contrast, both hyperfan-in and hyperfan-out init led to successful training immediately. This shows a good init can make it possible to successfully train models that would have otherwise been unamenable to training on a bad init.

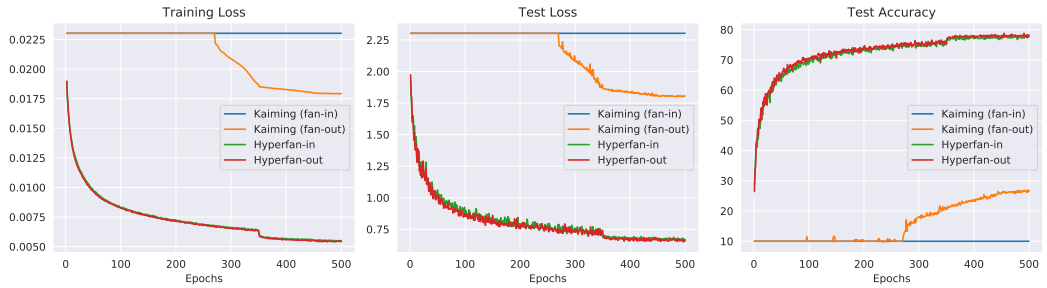

Figure 5: Loss and Test Accuracy Plots on CIFAR-10.

5.4 BAYESIAN NEURAL NETWORKS ON IMAGENET

Bayesian neural networks improve model calibration and provide uncertainty estimation, which guard against the pitfalls of overconfident networks. Ukai et al. (2018) developed a Bayesian neural network by using a hypernetwork to simulate an expressive prior distribution. We trained a similar hypernetwork by applying Ukai et al. (2018)'s methods on ImageNet, but differed in our choice of MobileNet (Howard et al., 2017) as a mainnet architecture that does not have residual connections.

In the work of Ukai et al. (2018), it was noticed that even with the use of batch normalization in the mainnet, classical initialization approaches still led to diverging losses (due to exploding activations, c.f. Section 3). We observe similar results in our experiment (see Figure 6) — the fan-in version of Kaiming init, which is the default initialization in popular deep learning libraries like PyTorch and Chainer, resulted in substantially higher initial losses and led to slower training than the hyperfan methods. This shows that while batch normalization helps, it is not the solution for a bad init that causes exploding activations. Our approach solves this problem in a principled way, and is preferable to trial-and-error based heuristics that Ukai et al. (2018) had to resort to in order to train their model.

Surprisingly, the fan-out version of Kaiming init led to similar results as the hyperfan methods, suggesting that batch normalization might be sufficient to correct the bad initializations that result in vanishing activations. That being said, hypernet practitioners should not expect batch normalization to be the panacea for problems caused by bad initialization, especially in memory-constrained scenarios. In a Bayesian neural network application (especially in hypernet architectures without relaxed weight-sharing), the blowup in the number of parameters limits the use of big batch sizes, which is essential to the performance of batch normalization (Wu & He, 2018). For example, in this experiment, our hypernet model requires 32 times as many parameters as a classical MobileNet.

To the best of our knowledge, the interaction between batch normalization and initialization is not well-understood, even in the classical case, and thus, our findings prompt an interesting direction for future research.

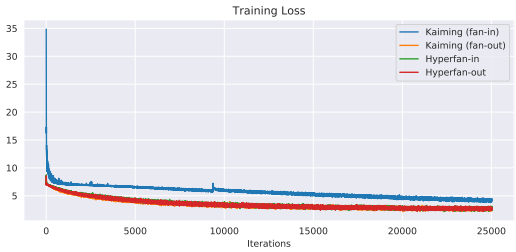

Figure 6: Training Loss on ImageNet.

6 Conclusion

In all our experiments, hyperfan-in and hyperfan-out both lead to successful hypernetwork training with SGD. We did not find a good reason to prefer one over the other (similar to He et al. (2015)'s observation in the classical case for fan-in and fan-out init).

For a long time, the promise of deep nets to learn rich representations of the world was left unfulfilled due to the inability to train these models. The discovery of greedy layer-wise pre-training (Hinton et al., 2006; Bengio et al., 2007) and later, Xavier and Kaiming init, as weight initialization strategies to enable such training was a pivotal achievement that kickstarted the deep learning revolution. This underscores the importance of model initialization as a fundamental step in learning complex representations.

In this work, we developed the first principled weight initialization methods for hypernetworks, a rapidly growing branch of meta-learning. We hope our work will spur momentum towards the development of principled techniques for building and training hypernetworks, and eventually lead to significant progress in learning meta representations. Other non-hypernetwork methods of neural network generation (Stanley et al., 2009; Koutnik et al., 2010) would also do well to consider if their generated weights result in exploding activations and how to avoid that if so.

References

Ivana Balazevic, Carl Allen, and Timothy M Hospedales. Hypernetwork knowledge graph embeddings. *arXiv preprint arXiv:1808.07018*, 2018.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pp. 153–160, 2007.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

Lior Deutsch, Erik Nijkamp, and Yu Yang. A generative model for sampling high-performance and diverse weights for neural networks. *arXiv preprint arXiv:1905.02898*, 2019.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Christian Henning, Johannes von Oswald, João Sacramento, Simone Carlo Surace, Jean-Pascal Pfister, and Benjamin F Grewe. Approximating the predictive distribution via adversarially-trained hypernetworks. In *Bayesian Deep Learning Workshop, NeurIPS (Spotlight)*, volume 2018, 2018.

Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Sylwester Klocek, Łukasz Maziarka, Maciej Wołczyk, Jacek Tabor, Marek Śmieja, and Jakub Nowak. Hypernetwork functional image representation. *arXiv preprint arXiv:1902.10404*, 2019.

Jan Koutnik, Faustino Gomez, and Jürgen Schmidhuber. Evolving neural networks in compressed weight space. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 619–626. ACM, 2010.

Agustinus Kristiadi and Asja Fischer. Predictive uncertainty quantification with compound density networks. *arXiv preprint arXiv:1902.01080*, 2019.

David Krueger, Chin-Wei Huang, Riashat Islam, Ryan Turner, Alexandre Lacoste, and Aaron Courville. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

Sören Laue, Matthias Mitterreiter, and Joachim Giesen. Computing higher order derivatives of matrix and tensor expressions. In *Advances in Neural Information Processing Systems*, pp. 2755–2764, 2018.

Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. *arXiv preprint arXiv:1903.10258*, 2019.

Jonathan Lorraine and David Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*, 2018.

Elliot Meyerson and Risto Miikkulainen. Modular universal reparameterization: Deep multi-task learning across diverse domains. *arXiv preprint arXiv:1906.00097*, 2019.

Zheyi Pan, Yuxuan Liang, Junbo Zhang, Xiuwen Yi, Yong Yu, and Yu Zheng. Hyperst-net: Hypernetworks for spatio-temporal forecasting. *arXiv preprint arXiv:1809.10889*, 2018.

Nick Pawlowski, Andrew Brock, Matthew CH Lee, Martin Rajchl, and Ben Glocker. Implicit weight uncertainty in neural networks. *arXiv preprint arXiv:1711.01297*, 2017.

Neale Ratzlaff and Li Fuxin. Hypergan: A generative model for diverse, performant neural networks. *arXiv preprint arXiv:1901.11058*, 2019.

David E Rumelhart, Geoffrey E Hintont, and Ronald J Williams. Learning representations by back-propagating errors. *NATURE*, 323:9, 1986.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Joan Serrà, Santiago Pascual, and Carlos Segura. Blow: a single-scale hyperconditioned flow for non-parallel raw-audio voice conversion. *arXiv preprint arXiv:1906.00794*, 2019.

Falong Shen, Shuicheng Yan, and Gang Zeng. Meta networks for neural style transfer. *arXiv preprint arXiv:1709.04111*, 2017.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

Joseph Suarez. Language modeling with recurrent highway hypernetworks. In *Advances in neural information processing systems*, pp. 3267–3276, 2017.

Zhun Sun, Mete Ozay, and Takayuki Okatani. Hypernetworks with statistical filtering for defending adversarial examples. *arXiv preprint arXiv:1711.01791*, 2017.

Kenya Ukai, Takashi Matsubara, and Kuniaki Uehara. Hypernetwork-based implicit posterior estimation and model averaging of cnn. In *Asian Conference on Machine Learning*, pp. 176–191, 2018.

Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F Grewe. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pp. 4148–4158, 2017.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018.

Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.

APPENDIX

A RE-USING HYPERNET WEIGHTS

A.1 FOR MAINNET WEIGHTS OF THE SAME SIZE

For model compression or weight-sharing purposes, different parts of the mainnet might be generated by the same hypernet function. This will cause some assumptions of independence in our analysis to be invalid. Consider the example of the same hypernet being used to generate multiple different mainnet weight layers of the same size, i.e. $H[t]^{i_{t+1}}_{i_t k} = H[t+1]^{i_{t+2}}_{i_{t+1} k}, \mathrm{d}_{i_{t+1}} = \mathrm{d}_{i_{t+2}} = \mathrm{d}_{i_t}$. Then, $x[t+1]^{i_{t+1}} = H[t]^{i_{t+1}}_{i_t k_t} e[t]^{k_t} x[t]^{i_t} \not\perp\!\!\!\perp W[t+1]^{i_{t+2}}_{i_{t+1}} = H[t+1]^{i_{t+2}}_{i_{t+1} k} e[t+1]^{k_{t+1}}$.

The relaxation of some of these independence assumptions does not always prove to be a big problem in practice, because the correlations introduced by repeated use of H can be minimized with the use of flat distributions like the uniform distribution. It can even be helpful, since the re-use of the same hypernet for different layers causes the gradient flowing through the hypernet output layer to be the sum of the gradients from the weights of these layers: $\frac{\partial L}{\partial h(e)^k} = \sum_t \frac{\partial L}{\partial W[t]^{i_{t+1}}_{i_t}} H^{i_{t+1}}_{i_t k}$, thus combating the shrinking effect.

A.2 FOR MAINNET WEIGHTS OF DIFFERENT SIZES

Similar reasoning applies if the same hypernet was used to generate differently sized subsets of weights in the mainnet. However, we encourage avoiding this kind of hypernet architecture design if not otherwise essential, since it will complicate the initialization formulae listed in Table 1.

Consider Ha et al. (2016)'s hypernetwork architecture. Their two-layer hypernet generated weight chunks of size (K, n, n) for a main convolutional network where $K = 16$ was found to be the highest common factor among the size of mainnet layers, and $n^2 = 9$ was the size of the receptive field. We simplify the presentation by writing i for i_t, j for j_t, k for $k_{t,m}$, and l for $l_{t,m}$.

$$
\begin{aligned}
W[t]^i_j &= \begin{cases} H^{i(\text{mod } K)}_k \alpha[t][j + \lfloor \frac{i}{K} \rfloor \mathrm{d}_j]^k + \beta^{i(\text{mod } K)} & \text{if } i \text{ is divisible by } K \\ \delta_{j(\text{mod } K) j(\text{mod } K)} \left[H^{j(\text{mod } K)}_k \alpha[t][i + \lfloor \frac{j}{K} \rfloor \mathrm{d}_i]^k + \beta^{j(\text{mod } K)} \right] & \text{if } j \text{ is divisible by } K \end{cases} \\
\alpha[t][m_t]^k &= G[t][m_t]^k_l e[t][m_t]^l + \gamma[t][m_t]^k
\end{aligned}
\tag{6}
$$

Because the output layer (H, β) in the hypernet was re-used to generate mainnet weight matrices of different sizes (i.e. in general, $i_t \neq i_{t+1}, j_t \neq j_{t+1}$), G effectively becomes the output layer that we want to be considering for hyperfan-in and hyperfan-out initialization.

Hence, to achieve fan-in in the mainnet $\mathrm{Var}(W[t]^i_j) = \frac{1}{\mathrm{d}_j}$, we have to use fan-in init for H (i.e. $\mathrm{Var}(H^{i(\text{mod } K)}_k) = \frac{1}{\mathrm{d}_k} \neq \frac{1}{\mathrm{d}_j \mathrm{d}_k \mathrm{Var}(e[t][m_t]^l)}$), and hyperfan-in init for G (i.e. $\mathrm{Var}(G[t][m_t]^k_l) = \frac{1}{\mathrm{d}_j \mathrm{d}_l \mathrm{Var}(e[t][m_t]^l)}$).

Analogously, to achieve fan-out in the mainnet $\mathrm{Var}(W[t]^i_j) = \frac{1}{\mathrm{d}_i}$, we have to use fan-in init for H (i.e. $\mathrm{Var}(H^{i(\text{mod } K)}_k) = \frac{1}{\mathrm{d}_k} \neq \frac{1}{\mathrm{d}_i \mathrm{d}_k \mathrm{Var}(e[t][m_t]^l)}$), and hyperfan-out init for G (i.e. $\mathrm{Var}(G[t][m_t]^k_l) = \frac{1}{\mathrm{d}_i \mathrm{d}_l \mathrm{Var}(e[t][m_t]^l)}$).

B MORE EXPERIMENTAL DETAILS

B.1 FEEDFORWARD NETWORKS ON MNIST

The networks were trained on MNIST for 30 epochs with batch size 10 using a learning rate of 0.0005 for the hypernets and 0.01 for the classical network. The hypernets had one linear layer with embeddings of size 50 and different hidden layers in the mainnet were all generated by the same hypernet output layer with a different embedding, which was randomly sampled from $\mathcal{U}(-\sqrt{3}, \sqrt{3})$ and fixed. We use the mean cross entropy loss for training, but the summed cross entropy loss for testing.

We show activation and gradient plots for two cases: (i) the hypernet generates only the weights of the mainnet, and (ii) the hypernet generates both the weights and biases of the mainnet. (i) covers Figures 3, 1, 7, 8, 9, 10, 11, 12, 2, 13, 14, 15, and 16. (ii) covers Figures 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, and 29.

The activations and gradients in our plots were calculated by averaging across a fixed held-out set of 300 examples drawn randomly from the test set.

In Figures 1, 8, 9, 11, 12, 13, 14, 16, 18, 20, 21, 23, 24, 26, 27, and 29, the y axis shows the number of activations/gradients, while the x axis shows the value of the activations/gradients. The value of activations/gradients from the hypernet output layer correspond to the value of mainnet weights.

In Figures 2, 7, 10, 15, 19, 22, 25, and 28, the y axis shows the mean value of the activations/gradients, while each increment on the x axis corresponds to a measurement that was taken every 1000 training batches, with the bars denoting one standard deviation away from the mean.

B.1.1 HYPERNET GENERATES ONLY THE MAINNET WEIGHTS

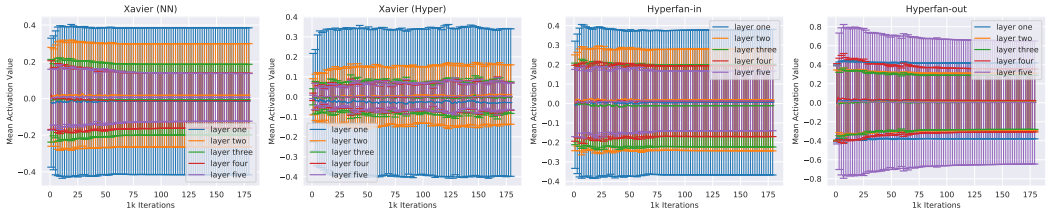

Figure 7: Evolution of Mainnet Activations during Training on MNIST.

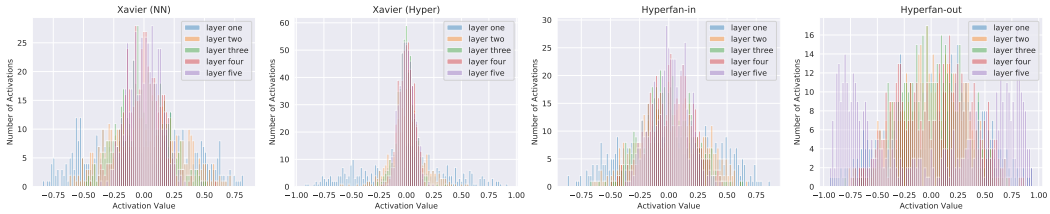

Figure 8: Mainnet Activations at the End of Training on MNIST.

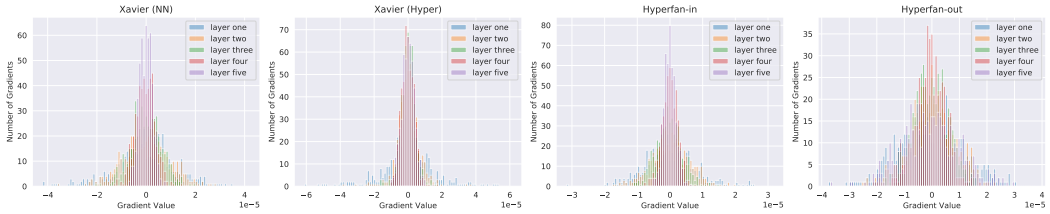

Figure 9: Mainnet Gradients before the Start of Training on MNIST.

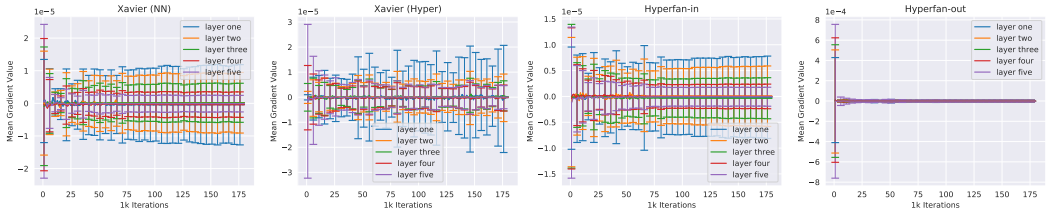

Figure 10: Evolution of Mainnet Gradients during Training on MNIST.

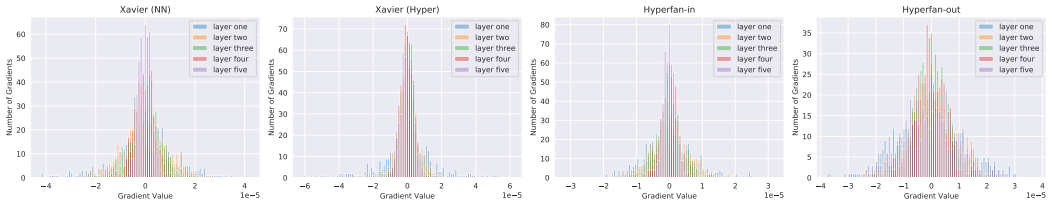

Figure 11: Mainnet Gradients at the End of Training on MNIST.

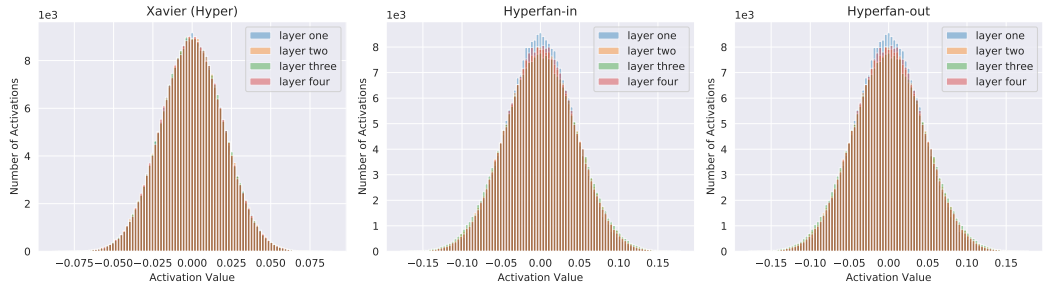

Figure 12: Hypernet Output Layer Activations before the Start of Training on MNIST.

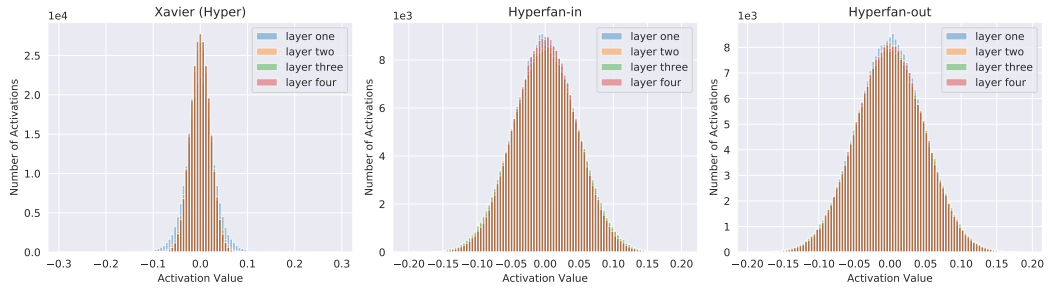

Figure 13: Hypernet Output Layer Activations at the End of Training on MNIST.

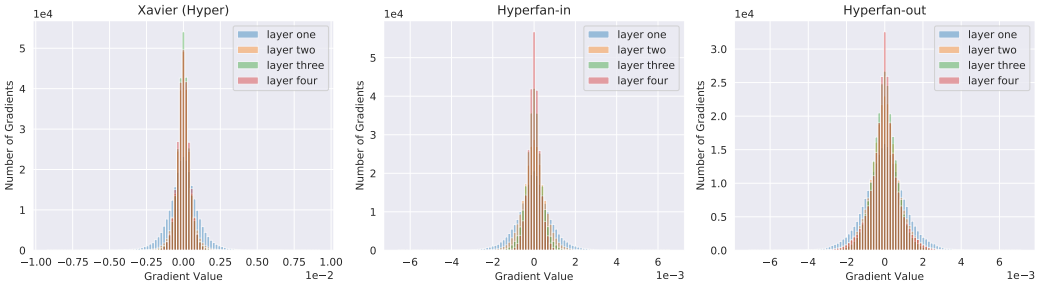

Figure 14: Hypernet Output Layer Gradients before the Start of Training on MNIST.

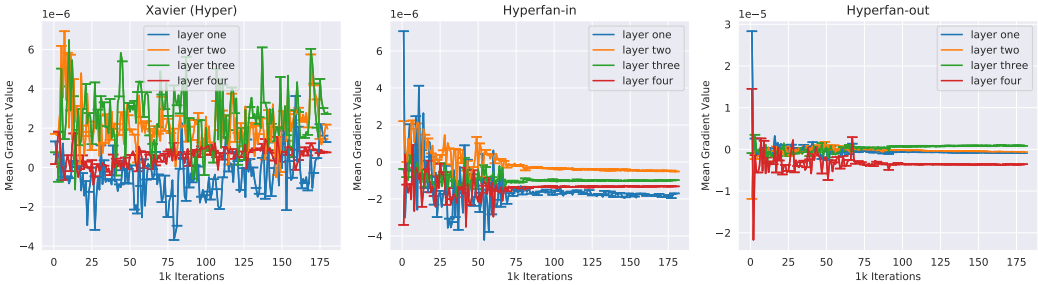

Figure 15: Evolution of Hypernet Output Layer Gradients during Training on MNIST.

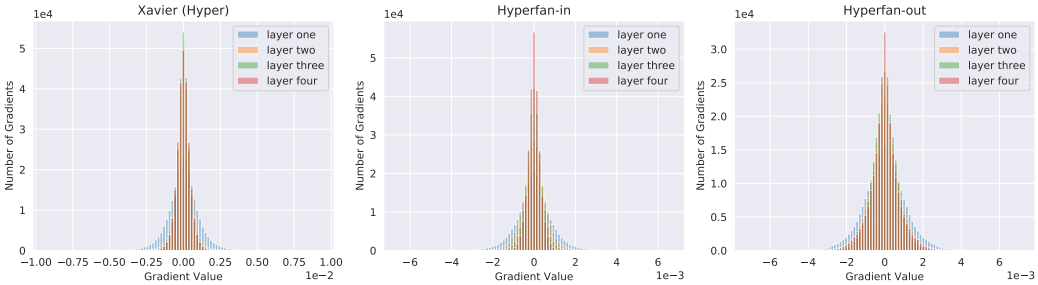

Figure 16: Hypernet Output Layer Gradients at the End of Training on MNIST.

B.1.2 HYPERNET GENERATES BOTH MAINNET WEIGHTS AND BIASES

Figure 17: Loss and Test Accuracy Plots on MNIST.

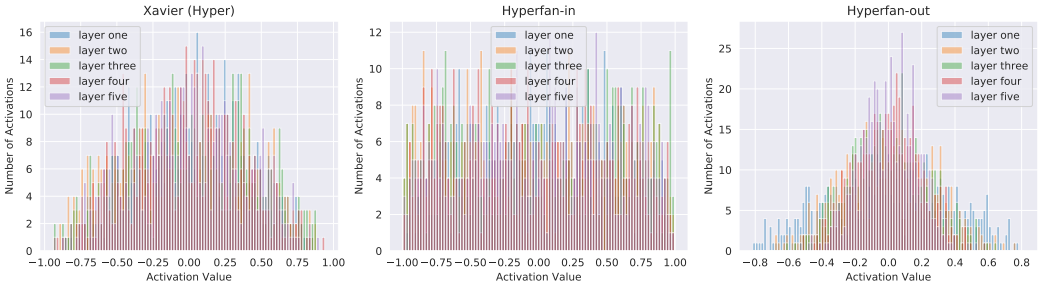

Figure 18: Mainnet Activations before the Start of Training on MNIST.

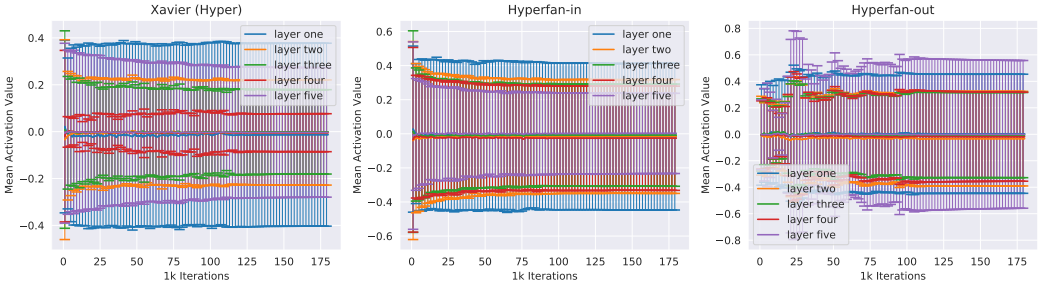

Figure 19: Evolution of Mainnet Activations during Training on MNIST.

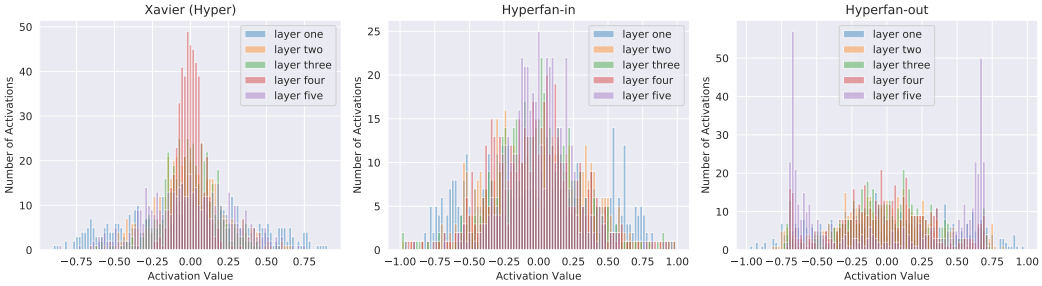

Figure 20: Mainnet Activations at the End of Training on MNIST.

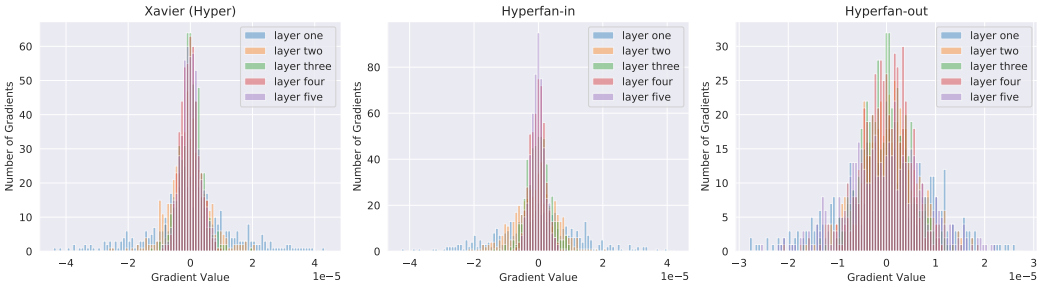

Figure 21: Mainnet Gradients before the Start of Training on MNIST.

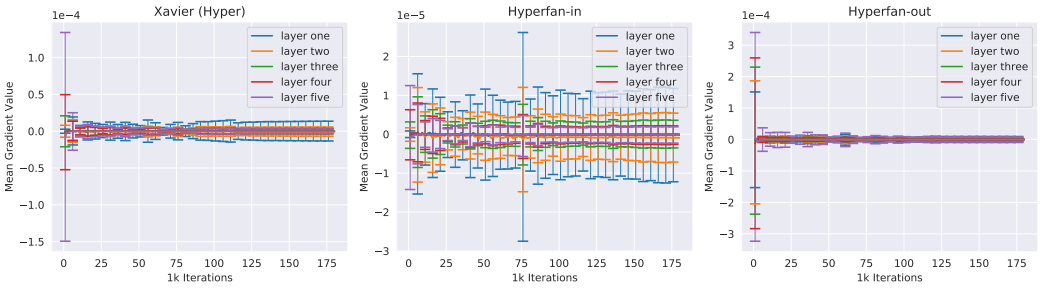

Figure 22: Evolution of Mainnet Gradients during Training on MNIST.

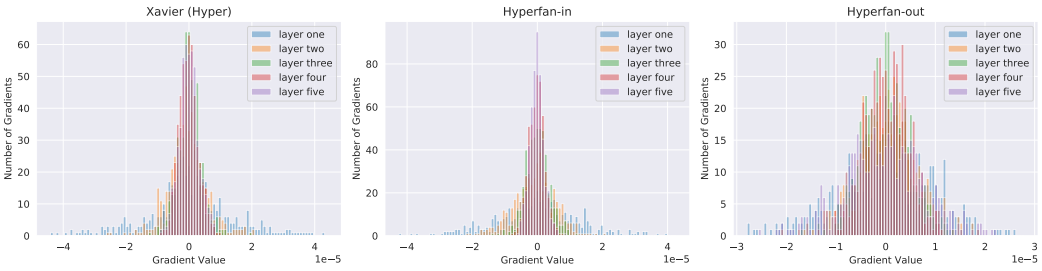

Figure 23: Mainnet Gradients at the End of Training on MNIST.

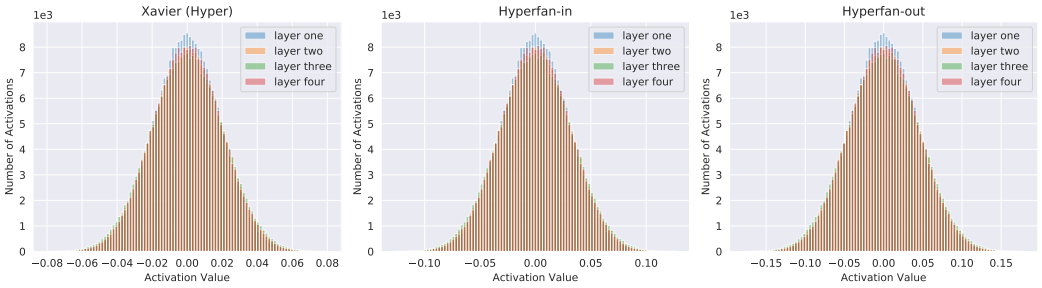

Figure 24: Hypernet Output Layer Activations before the Start of Training on MNIST.

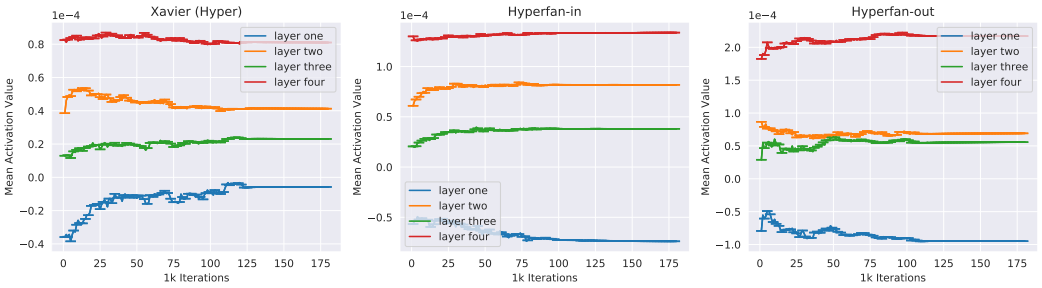

Figure 25: Evolution of Hypernet Output Layer Activations during Training on MNIST.

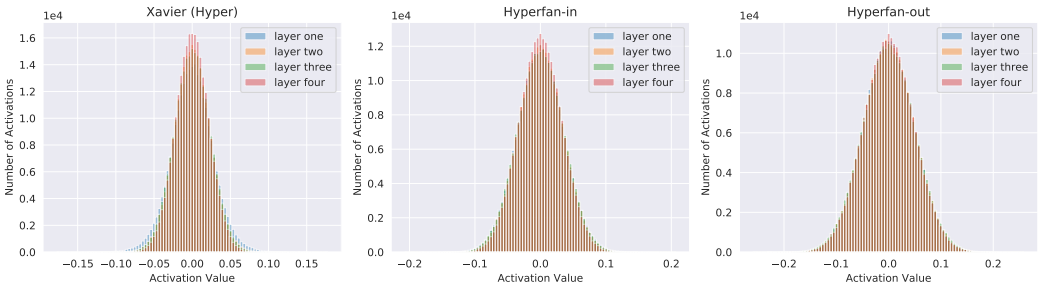

Figure 26: Hypernet Output Layer Activations at the End of Training on MNIST.

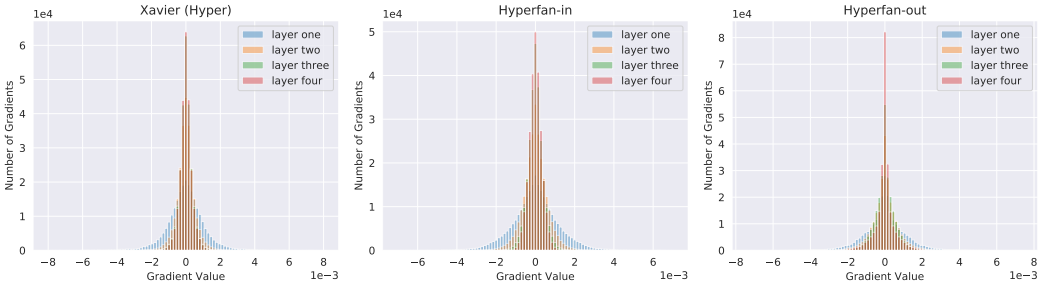

Figure 27: Hypernet Output Layer Gradients before the Start of Training on MNIST.

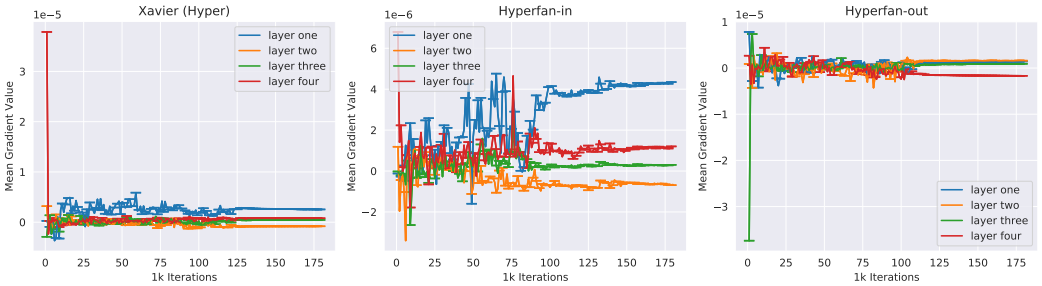

Figure 28: Evolution of Hypernet Output Layer Gradients during Training on MNIST.

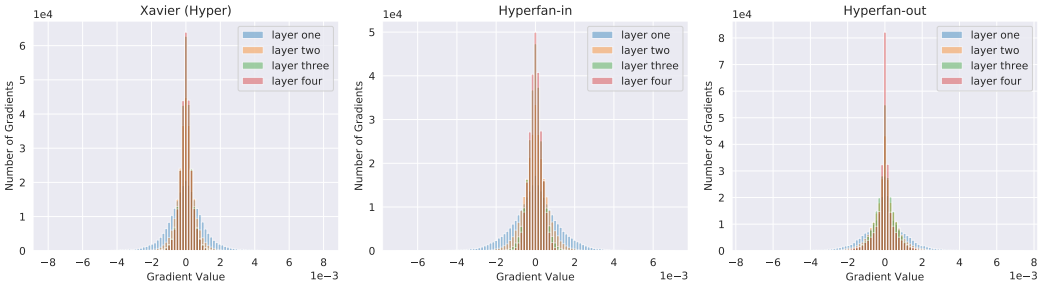

Figure 29: Hypernet Output Layer Gradients at the End of Training on MNIST.

B.1.3 REMARK ON THE COMBINATION OF FAN-IN AND FAN-OUT INIT

Glorot & Bengio (2010) proposed to use the harmonic mean of the two different initialization formulae derived from the forward and backward pass. He et al. (2015) commented that either version suffices for convergence, and that it does not really matter given that the difference between the two will be a depth-independent factor.

We experimented with the harmonic, geometric, and arithmetic means of the two different formulae in both the classical and the hypernet case. There was no indication of any significant benefit from taking any of the three different means in both cases. Thus, we confirm and concur with He et al. (2015)'s original observation that either the fan-in or the fan-out version suffices.

B.2 CONTINUAL LEARNING

The mainnet is a feedforward network with two hidden layers (10 hidden units) and the ReLU activation function. The weights and biases of the mainnet are generated from a hypernet with two hidden layers (10 hidden units) and trainable embeddings of size 2 sampled from $\mathcal{U}(-\sqrt{3}, \sqrt{3})$. We keep the same continual learning hyperparameter β_{output} value of 0.005 and pick the best learning rate for each initialization method from $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. Notably, Kaiming (fan-in) could only be trained from learning rate 10^{-5}, with losses diverging soon after initialization using the other learning rates. Each task was trained for 6000 training iterations using batch size 32, with Figure 4 plotted from losses measured at every 100 iterations.

B.3 CONVOLUTIONAL NETWORKS ON CIFAR-10

The networks were trained on CIFAR-10 for 500 epochs starting with an initial learning rate of 0.0005 using batch size 100, and decaying with $\gamma = 0.1$ at epochs 350 and 450. The hypernet is composed of two layers (50 hidden units) with separate embeddings and separate input layers but shared output layers. The weight generation happens in blocks of $(96, 3, 3)$ where $K = 96$ is the highest common factor between the different sizes of the convolutional layers in the mainnet and $n = 3$ is the size of the convolutional filters (see Appendix Section A.2 for a more detailed explanation on the hypernet architecture). The embeddings are size 50 and fixed after random sampling from $\mathcal{U}(-\sqrt{3}, \sqrt{3})$. We use the mean cross entropy loss for training, but the summed cross entropy loss for testing.

B.4 BAYESIAN NEURAL NETWORK ON IMAGENET

Ukai et al. (2018) showed that a Bayesian neural network can be developed by using a hypernetwork to express a prior distribution without substantial changes to the vanilla hypernetwork setting. Their methods simply require putting $\mathcal{L}_2$-regularization on the model parameters and sampling from stochastic embeddings. We trained a linear hypernet to generate the weights of a MobileNet mainnet architecture (excluding the batch normalization layers), using the block-wise sampling strategy described in Ukai et al. (2018), with a factor of 0.0005 for the $\mathcal{L}_2$-regularization. We initialize fixed embeddings of size 32 sampled from $\mathcal{U}(-\sqrt{3}, \sqrt{3})$, and sample additive stochastic noise coming from $\mathcal{U}(-0.1, 0.1)$ at the beginning of every mini-batch training. Due to expensive computational requirements involved in training on ImageNet, we trained it with batch size 256 and learning rate 0.1 for 5 epochs, or equivalently, 25025 iterations.