Inherent Weight Normalization in Stochastic Neural Networks

Georgios Detorakis
Department of Cognitive Sciences
University of California Irvine
Irvine, CA 92697
`gdetorak@uci.edu`

Sourav Dutta
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556 USA
`sdutta4@nd.edu`

Abhishek Khanna
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556 USA
`akhanna@nd.edu`

Matthew Jerry
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556 USA
`mjerry@alumni.nd.edu`

Suman Datta
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556 USA
`sdatta@nd.edu`

Emre Neftci
Department of Cognitive Sciences
Department of Computer Science
University of California Irvine
Irvine, CA 92697
`eneftci@uci.edu`

Abstract

Multiplicative stochasticity such as Dropout improves the robustness and generalizability of deep neural networks. Here, we further demonstrate that always-on multiplicative stochasticity combined with simple threshold neurons are sufficient operations for deep neural networks. We call such models Neural Sampling Machines (NSM). We find that the probability of activation of the NSM exhibits a self-normalizing property that mirrors Weight Normalization, a previously studied mechanism that fulfills many of the features of Batch Normalization in an online fashion. The normalization of activities during training speeds up convergence by preventing internal covariate shift caused by changes in the input distribution. The always-on stochasticity of the NSM confers the following advantages: the network is identical in the inference and learning phases, making the NSM suitable for online learning, it can exploit stochasticity inherent to a physical substrate such as analog non-volatile memories for in-memory computing, and it is suitable for Monte Carlo sampling, while requiring almost exclusively addition and comparison operations. We demonstrate NSMs on standard classification benchmarks (MNIST and CIFAR) and event-based classification benchmarks (N-MNIST and DVS Gestures). Our results show that NSMs perform comparably or better than conventional artificial neural networks with the same architecture.

1 Introduction

Stochasticity is a valuable resource for computations in biological and artificial neural networks [9, 32, 2]. It affects neural networks in many different ways. Some of them are (i) escaping local minima during learning and inference [1], (ii) stochastic regularization in neural networks [21, 52], (iii)

Bayesian inference approximation with Monte Carlo sampling [9, 16], (iv) stochastic facilitation [31], and (v) energy efficiency in computation and communication [28, 19].

In artificial neural networks, multiplicative noise is applied as random variables that multiply network weights or neural activities (*e.g.* Dropout). In the brain, multiplicative noise is apparent in the probabilistic nature of neural activations [19] and their synaptic quantal release [8, 51]. Analog non-volatile memories for in-memory computing such as resistive RAMs, ferroelectric devices or phase-change materials [54, 23, 15] exhibit a wide variety of stochastic behaviors [42, 34, 54, 33], including set/reset variability [3] and random telegraphic noise [4]. In crossbar arrays of non-volatile memory devices designed for vector-matrix multiplication (*e.g.* where weights are stored in the resistive or ferroelectric states), such stochasticity manifests itself as multiplicative noise.

Motivated by the ubiquity of multiplicative noise in the physics of artificial and biological computing substrates, we explore here Neural Sampling Machines (NSMs): a class of neural networks with binary threshold neurons that rely almost exclusively on multiplicative noise as a resource for inference and learning. We highlight a striking self-normalizing effect in the NSM that fulfills a role that is similar to Weight Normalization during learning [47]. This normalizing effect prevents internal covariate shift as with Batch Normalization [22], stabilizes the weights distributions during learning, and confers rejection to common mode fluctuations in the weights of each neuron.

We demonstrate the NSM on a wide variety of classification tasks, including classical benchmarks and neuromorphic, event-based benchmarks. The simplicity of the NSM and its distinct advantages make it an attractive model for hardware implementations using non-volatile memory devices. While stochasticity there is typically viewed as a disadvantage, the NSM has the potential to exploit it. In this case, the forward pass in the NSM simply boils down to weight memory lookups, additions, and comparisons.

1.1 Related Work

The NSM is a stochastic neural network with discrete binary units and thus closely related to Binary Neural Networks (BNN). BNNs have the objective of reducing the computational and memory footprint of deep neural networks at run-time [14, 44]. This is achieved by using binary weights and simple activation functions that require only bit-wise operations.

Contrary to BNNs, the NSM is stochastic during both inference and learning. Stochastic neural networks are argued to be useful in learning multi-modal distributions and conditional computations [7, 50] and for encoding uncertainty [16].

Dropout and Dropconnect techniques randomly mask a subset of the neurons and the connections during train-time for regularization and preventing feature co-adaptation [21, 52]. These techniques continue to be used for training modern deep networks. Dropout during inference time can be viewed as approximate Bayesian inference in deep Gaussian processes [16], and this technique is referred to as Monte Carlo (MC) Dropout. NSMs are closely related to MC Dropout, with the exception that the activation function is stochastic and the neurons are binary. Similarly to MC Dropout, the "always-on" stochasticity of NSMs can be in principle articulated as a MC integration over an equivalent Gaussian process posterior approximation, fitting the predictive mean and variance of the data. MC Dropout can be used for active learning in deep neural networks, whereby a learner selects or influences the training dataset in a way that optimally minimizes a learning criterion [16, 12].

Taken together, NSM can be viewed as a combination of stochastic neural networks, Dropout and BNNs. While stochastic activations in the binarization function are argued to be inefficient due to the generation of random bits, stochasticity in the NSM, however, requires only one random bit per pass per neuron or per connection. A different approach for achieving zero mean and unit variance is the self-normalizing neural networks proposed in [25]. There, an activation function in non-binary, deterministic networks is constructed mathematically so that outputs are normalized. In contrast, in the NSM unit, normalization in the sense of [47] emerges from the multiplicative noise as a by-product of the central limit theorem. This establishes a connection between exploiting the physics of hardware systems and recent deep learning techniques, while achieving good accuracy on benchmark classification tasks. Such a connection is highly significant for the devices community, as it implies a simple circuit (threshold operations and crossbars) that can exploit (rather than mitigate) device non-idealities such as read stochasticity.

In recurrent neural networks, stochastic synapses were shown to behave as stochastic counterparts of Hopfield networks [38], but where stochasticity is caused by multiplicative noise at the synapses (rather than logistic noise in Boltzmann machines). These were shown to surpass the performances of equivalent machine learning algorithms [20, 36] on certain benchmark tasks.

1.2 Our Contribution

In this article, we demonstrate multi-layer and convolutional neural networks employing NSM layers on GPU simulations, and compare with their equivalent deterministic neural networks. We articulate NSM's self-normalizing effect as a statistical equivalent of Weight Normalization. Our results indicate that a neuron model equipped with a hard-firing threshold (*i.e.*, a Perceptron) and stochastic neurons and synapses:

- Is a sufficient resource for stochastic, binary deep neural networks.
- Naturally performs weight normalization.
- Can outperform standard artificial neural networks of comparable size.

The always-on stochasticity gives the NSM distinct advantages compared to traditional deep neural networks or binary neural networks: The shared forward passes for training and inference in NSM are consistent with the requirement of online learning since an NSM implements weight normalization, which is not based on batches [47]. This enables simple implementations of neural networks with emerging devices. Additionally, we show that the NSM provides robustness to fluctuations and fixed precision of the weights during learning.

1.3 Applications

During inference, the binary nature of the NSM equipped with blank-out noise makes it largely multiplication-free. As with the Binarynet [13] or XNORnet [44], we speculate that they can be most advantageous in terms of energy efficiency on dedicated hardware.

The NSM is of interest for hardware implementations in memristive crossbar arrays, as threshold units are straightforward to implement in CMOS and binary inputs mitigate read and write non-idealities in emerging non-volatile memory devices while reducing communication bandwidth [54]. Furthermore, multiplicative stochasticity in the NSM is consistent with the stochastic properties of emerging nanodevices [42, 34]. Exploiting the physics of nanodevices for generating stochasticity can lead to significant improvements in embedded, dedicated deep learning machines.

2 Methods

2.1 Neural Sampling Machines (NSM)

We formulate the NSM as a stochastic neural network model that exploits the properties of multiplicative noise to perform inference and learning. For mathematical tractability, we focus on threshold (sign) units, where $\mathrm{sgn}:\mathbb{R}\to[-1,1]$,

$$z_i = \mathrm{sgn}(u_i) = \begin{cases} 1 & \text{if } u_i \geq 0 \\ -1 & \text{if } u_i < 0 \end{cases}, \tag{1}$$

where u_i is the pre-activation of neuron i given by the following equation

$$u_i = \sum_{j=1}^{N} \xi_{ij} w_{ij} z_j + b_i + \eta_i, \tag{2}$$

where ξ_{ij} and η_i represent multiplicative and additive noise terms, respectively. Both ξ and η are independent and identically distributed (*iid*) random variables. w_{ij} is the weight of the connection between neurons i and j, b_i is a bias term, and N is the number of input connections (fan-in) to neuron i. Note that multiplicative noise can be introduced at the synapse (ξ_{ij}), or at the neuron (ξ_i).

Since the neuron is a threshold unit, it follows that $P(z_i = 1|\mathbf{z}) = P(u_i \geq 0|\mathbf{z})$. Thus, the probability that unit i is active given the network state is equal to one minus the cumulative distribution function

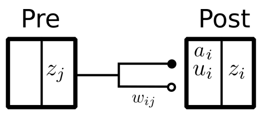

Figure 1: **Blank-out synapse with scaling factors.** Weights are accumulated on u_i as a sum of a deterministic term scaled by α_i (filled discs) and a stochastic term with fixed blank-out probability p (empty discs).

of u_i. Assuming independent random variables u_i, the central limit theorem indicates that the probability of the neuron firing is $P(z_i = 1|\mathbf{z}) = 1 - \Phi(u_i|\mathbf{z})$ (where Φ is the cumulative distribution function of normal distribution) and more precisely

$$P(z_i = 1|\mathbf{z}) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{\mathbb{E}(u_i|\mathbf{z})}{\sqrt{2\mathrm{Var}(u_i|\mathbf{z})}}\right)\right), \tag{3}$$

where $\mathbb{E}(u_i)$ and $\mathrm{Var}(u_i)$ are the expectation and variance of state u_i.

In the case where only independent additive noise is present, equation (2) is rewritten as $u_i = \sum_{j=1}^N w_{ij} z_j + b_i + \eta_i$ and the expectation and variance are given by $\mathbb{E}(u_i|\mathbf{z}) = \sum_{j=1}^N w_{ij} z_j + b_i + \mathbb{E}(\eta)$ and $\mathrm{Var}(u_i|\mathbf{z}) = \mathrm{Var}(\eta)$, respectively. In this case, equation (3) is a sigmoidal neuron with an erf activation function with constant bias $\mathbb{E}(\eta)$ and constant slope $\mathrm{Var}(\eta)$. Thus, besides the sigmoidal activation function, the additive noise case does not endow the network with any extra properties.

In the case of multiplicative noise, equation (2) becomes $u_i = \sum_{j=1}^N \xi_{ij} w_{ij} z_j + b_i$ and its expectation and variance are given by $\mathbb{E}(u_i|\mathbf{z}) = \mathbb{E}(\xi)\sum_{j=1}^N w_{ij} z_j$ and $\mathrm{Var}(u_i|\mathbf{z}) = \mathrm{Var}(\xi)\sum_{j=1}^N w_{ij}^2$, respectively. In this derivation, we have used the fact that the square of a sign function is a constant function ($\mathrm{sgn}^2(x) = 1$). In contrast to the additive noise case, $\mathrm{Var}(u_i|\mathbf{z})$ is proportional to the square of the input weight parameters. The probability of neurons being active becomes:

$$\begin{aligned} P(z_i = 1|\mathbf{z}) &= \frac{1}{2}\left(1 + \mathrm{erf}\left(\mathbf{v}_i \cdot \mathbf{z}\right)\right), \\ \text{with } \mathbf{v}_i &= \beta_i \frac{\mathbf{w}_i}{||\mathbf{w}_i||_2}, \end{aligned} \tag{4}$$

where β_i here is a variable that captures the parameters of the noise process ξ_i. In the denominator, we have used the identity $\sqrt{\sum_j w_{ij}^2} = \sqrt{\sum_j w_{ij}^2 z_j^2} = ||\mathbf{w}_i||_2$, where $||\cdot||_2$ denotes the $L2$ norm of the weights of neuron i. This term has a normalizing effect on the activation function, similar to weight normalization, as discussed below. Note that the self-normalizing effect is not specific to the distribution of the chosen random variables, and holds as long as the random variables are *iid*.

One consequence of multiplicative noise here is that any positive scaling factor applied to $\mathbf{w}_i$ is canceled out by the norm. To counter this problem and control β_i without changing the distribution governing ξ, the NSM introduces a factor a_i in the preactivation's equation:

$$u_i = \sum_{j=1}^N (\xi_{ij} + a_i) w_{ij} z_j + b_i. \tag{5}$$

Thanks to the binary nature of z_i, equation (5) is multiplication-free except for the term involving a_i. Since a_i is defined per neuron, the multiplication operation is only performed once per neuron and time step. In this article, we focus on two relevant cases of noise: Gaussian noise with mean 1 and variance σ^2, $\xi_{ij} \sim \mathcal{N}(1, \sigma^2)$ and Bernoulli (blank-out) noise $\xi_{ij} \sim Bernoulli(p)$, with parameter p. From now on we focus only on the multiplicative noise case.

Gaussian Noise In the case of multiplicative Gaussian Noise, ξ in equation (5) is a Gaussian random variable $\xi \sim \mathcal{N}(1, \sigma^2)$. This means that the expectation and variance are $\mathbb{E}(u_i|\mathbf{z}) = (1+a_i)\sum_j w_{ij} z_j$ and $\mathrm{Var}(u_i|\mathbf{z}) = \sigma^2 \sum_j w_{ij}^2$, respectively. And hence, $\beta_i = \frac{1+a_i}{\sqrt{2\sigma^2}}$.

Bernoulli (Blank-out) Noise Bernoulli ("Blank-out") noise can be interpreted as a Dropout mask on the neurons or a Dropconnect mask on the synaptic weights (see Fig 1), where $\xi_{ij} \in [0, 1]$ in equation (5) becomes a Bernoulli random variable with parameter p. Since the ξ_{ij} are independent, for a given $\mathbf{z}$, a sufficiently large fan-in, and $0 < p < 1$, the sums in equation (5) are Gaussian-distributed with means and variances $\mathbb{E}(u_i|\mathbf{z}) = (p+a_i)\sum_j w_{ij} z_j$ and $\mathrm{Var}(u_i|\mathbf{z}) = p(1-p)\sum_j w_{ij}^2$, respectively. Therefore we obtain: $\beta_i = \frac{p+a_i}{\sqrt{2p(1-p)}}$.

We observed empirically that whether the neuron is stochastic or the synapse is stochastic did not significantly affect the results.

2.2 NSMs implements Weight Normalization

The key idea in weight normalization [47] is to normalize unit activity by reparameterizing the weight vectors. The reparameterization used there has the form: $\mathbf{v}_i = \beta_i \frac{\mathbf{w}_i}{||\mathbf{w}_i||}$. This is exactly the form obtained by introducing multiplicative noise in neurons (equation (4)), suggesting that NSMs inherently perform weight normalization in the sense of [47]. The authors argue that decoupling the magnitude and the direction of the weight vectors speeds up convergence and confers many of the features of batch normalization. To achieve weight normalization effectively, gradient descent is performed with respect to the scalars β (which are themselves parameterized with a_i) in addition to the weights $\mathbf{w}$:

$$\partial_{\beta_i}\mathcal{L} = \frac{\sum_j w_{ij}\partial_{v_{ij}}\mathcal{L}}{||\mathbf{w}_i||} \tag{6}$$

$$\partial_{w_{ij}}\mathcal{L} = \frac{\beta_i}{||\mathbf{w}_i||}\partial_{v_{ij}}\mathcal{L} - \frac{\mathbf{w}_i\beta_i}{||\mathbf{w}_i||^2}\partial_{\beta_i}\mathcal{L} \tag{7}$$

2.3 NSM Training Procedure

Neural sampling machines (and stochastic neural networks in general) are challenging to train because errors cannot be directly back-propagated through stochastic nodes. This difficulty is compounded by the fact that the neuron state is a discrete random variable, and as such the standard reparametrization trick is not directly applicable [17]. Under these circumstances, unbiased estimators resort to minimizing expected costs through the family of REINFORCE algorithms [53, 7, 35] (also called score function estimator and likelihood-ratio estimator). Such algorithms have general applicability but gradient estimators have impractically high variance and require multiple passes in the network to estimate them [43]. Straight-through estimators ignore the non-linearity altogether [7], but result in networks with low performance. Several work have introduced methods to overcome this issue, such as in discrete variational autoencoders [46], bias reduction techniques for the REINFORCE algorithm [18] and concrete distribution approach (smooth relaxations of discrete random variables) [30] or other reparameterization tricks [48].

Striving for simplicity, here we propagate gradients through the neurons' activation probability function. This approach theoretically comes at a cost in accuracy because the rule is a biased estimate of the gradient of the loss. This is because the gradients are estimated using activation probability. However, it is more efficient than REINFORCE algorithms as it uses the information provided by the gradient back-propagation algorithm. In practice, we find that, provided adequate initialization, the gradients are well behaved and yield good performance while being able to leverage existing automatic differentiation capabilities of software libraries (*e.g.* gradients in Pytorch [40]). In the implementation of NSMs, probabilities are only computed for the gradients in the backward pass, while only binary states are propagated in the forward pass (see SI 4.2).

To assess the impact of this bias, we compare the above training method with Concrete Relaxation which is unbiased [30]. The NSM network is compatible with the binary case of Concrete relaxation. We trained the NSM using BinConcrete units on MNIST data set (Test Error Rate: 0.78%), and observed that the angles between the gradients of the proposed NSM and BinConcrete are close (see SI 4.11).

Unless otherwise stated, and similarly to [47], we use a data-dependent initialization of the magnitude parameters β and the bias parameters over one batch of 100 training samples such that the

Table 1: Classification error on the permutation invariant MNIST task (test set). Error is estimated by averaging test errors over 100 samples (for NSMs) and over the 50 last epochs.

Data set	Network	NSM
PI MNIST	NSM 784–300–300–300–10	1.36 %
PI MNIST	StNN 784–300–300–300–10	1.47 %
PI MNIST	NSM scaled 784–300–300–300–10	1.38 %

preactivations to each layer have zero mean and unit variance over that batch:

$$\beta \leftarrow \frac{1}{\sigma}, \qquad b \leftarrow -\frac{\mu ||w|| \sqrt{2Var(\xi)}}{\sigma}, \tag{8}$$

where μ and σ are feature-wise means and standard deviations estimated over the minibatch. For all classification experiments, we used cross-entropy loss $\mathcal{L}^n = -\sum_i t_i^n \log p_i^n$, where n indexes the data sample and p_i is the Softmax output. All simulations were performed using Pytorch [40]. All NSM layers were built as custom Pytorch layers (for more details about simulations see SI 4.8).[1]

3 Experiments

3.1 Multi-layer NSM Outperforms Standard Stochastic Neural Networks in Speed and Accuracy

In order to characterize the classification abilities of the NSM we trained a fully connected network on the MNIST handwritten digit image database for digit classification. The network consisted of three fully-connected layers of size 300, and a Softmax layer for 10-way classification and all Bernoulli process parameters were set to $p = .5$. The NSM was trained using back-propagation and a softmax layer with cross-entropy loss and minibatches of size 100. As a baseline for comparison, we used the stochastic neural network (StNN) presented in [27] without biases, with a sigmoid activation probability $P_{sig}(z_i = 1|\mathbf{z}) = \mathrm{sigmoid}(\mathbf{w}_i \cdot \mathbf{z})$.

The results of this experiment are shown in Table 1. The 15th, 50th and 85th percentiles of the input distributions to the last hidden layer during training is shown in Fig. 2. The evolution of the distribution in the NSM case is more stable, suggesting that NSMs indeed prevent internal covariate shift.

Both the speed of convergence and accuracy within 200 iterations are higher in the NSM compared to the StNN. The higher performance in the NSM is achieved using inference dynamics that are simpler than the StNN (sign activation function compared to a sigmoid activation function) and using binary random variables.

3.2 Robustness to Weight Fluctuations

The decoupling of the weight matrix as in $\mathbf{v}_i = \beta_i \frac{\mathbf{w}_i}{||\mathbf{w}_i||}$ introduces several additional advantages in learning machines. During learning, the distribution of the weights for a layer tend to remain more stable in NSM compared to the StNN (SI Fig. 4). This feature can be exploited to mitigate saturation at the boundaries of fixed range weight representations (*e.g.* in fixed-point representations or memristors). Another subtle advantage from an implementation point of view is that the probabilities are invariant to positive scaling of the weights, *i.e.* $\frac{\alpha \mathbf{w}_i}{||\alpha \mathbf{w}_i||} = \frac{\mathbf{w}_i}{||\mathbf{w}_i||}$ for $\alpha \geq 0$. Table 1 shows that NSM with weights multiplied by a constant factor .1 (called NSM scaled in the table) during inference did not significantly affect the classification accuracy. This suggests that the NSM can be robust to common mode fluctuations that may affect the rows of the weight matrix. Note that this property does not hold for ANNs with standard activation functions (relu, sigmoid, tanh), and the network performance is lost by such scaling (for more details see SI 4.5).

[1]`https://github.com/nmi-lab/neural_sampling_machines`

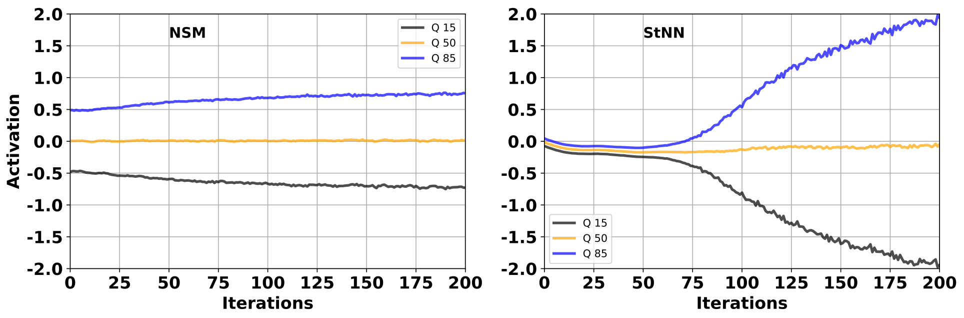

Figure 2: NSM mitigates internal covariate shift. 15th, 50th and 85th percentiles of the input distribution to the last hidden layer (similarly to Fig. 1 in [22]). The internal covariate shift is visible in the StNN as the input distributions change significantly during the learning. The self normalizing effect in NSM performs weight normalization, which is known to mitigate this shift and speed up learning. Each iteration corresponds to one mini-batch update (100 data samples per mini-batch, 20000 data samples total).

3.3 Supervised Classification Experiments: MNIST Variants

We validate the effectiveness of NSMs in supervised classification experiments on MNIST [26], EMNIST [11], N-MNIST [39], and DVS Gestures data sets (See Methods) using convolutional architecture. For all data sets, the inputs were converted to $-1/+1$ binary in a deterministic fashion using the function defined in equation (1). For the MNIST variants we trained all the networks for 200 epochs presenting at each epoch the entire dataset. For testing the accuracy of the networks we used the entire test dataset sampling each minibatch 100 times.

NSM models with Gaussian noise (gNSM) and Bernouilli noise (bNSM) converged to similar or better accuracy compared to the architecturally equivalent deterministic models. The results for MNIST, EMNIST and N-MNIST are given in Table 2, where we compare with the deterministic counterpart convolutional neural network (see Table 6 in the SI). In addition we compared with a binary (sign non-linearity) deterministic network (BD), a binary deterministic network with weight normalization (wBD), a stochastic network (noisy rectifier [7]) (SN), and a deterministic binary network (BN). We trained the first three networks using a Straight-Through Estimator [7] (STE) and the latter one using erf function in the backward pass only (*i.e.*, gradients computed on the erf function). The architecture of all these four networks is the same as the NSM's. The training process was the same as for the NSM networks and the results are given in Table 2. From these results we can conclude that NSM training procedure provides better performance than the STE and normalization of binary deterministic networks trained with a similar way as NSM (*e.g.*, BN).

3.4 Supervised Classification Experiments: CIFAR10/100

We tested the NSM on the CIFAR10 and CIFAR100 dataset of natural images. We used the model architecture described in [47] and added an extra input convolutional layer to convert RGB intensities into binary values. The NSM non-linearities are sign functions given by equation (1). We used the Adam [24] optimizer, with initial learning rate 0.0003 and we trained for 200 epochs using a batch size of 100 over the entire CIFAR10/100 data sets (50K/10K images for training and testing respectively). The test error was computed after each epoch and by running 100 times each batch (MC samples) with different seeds. Thus classification was made on the average over the MC samples. After 100 epochs we started decaying the learning rate linearly and we changed the first moment from 0.9 to 0.5. The results are given in Table 5. For the NSM networks we tried two different types of initialization. First, we initialized the weights with the values of the already trained deterministic network weights. Second and in order to verify that the initialization does not affect dramatically the training, we initialized the NSM without using any pre-trained weights. In both cases the performance of the NSM was similar as it is indicated in Table 5. We compared with the counterpart deterministic implementation using the exact same parameters and same additional input convolutional layer.

Table 2: (Top) Classification error on MNIST datasets. Error is estimated by averaging test errors over 100 samples (for NSMs), 5 runs, and over the 10 last epochs. Prefix, d-deterministic, b-Bernouilli, g-Gaussian. (Bottom) Comparison of networks on MNIST classification task. The NSM variations Bernoulli (bNSM) and Gaussian (gNSM) are compared with an NSM trained with a Straight-Through Estimator instead of the proposed training algorithm, a deterministic binary (sign non-linearity) network (BD), a BD with weight normalization enabled (wBD), a stochastic network (noisy rectifier) (SN) and a binary network (BN). For more details see section 3.3 in the main text.

Dataset	**dCNN**	**bNSM**	**gNSM**
MNIST	0.880%	0.775 %	0.805%
EMNIST	6.938%	6.185 %	6.256%
NMNIST	0.927%	0.689 %	0.701%

Model	bNSM	gNSM	bNSM (STE)	BD	wBD	SN	BN
Error	0.775	0.805	2.13	3.11	2.72	2.05	1.10

Table 3: Classification error on CIFAR10/CIFAR100. Error is estimated by sampling 100 times each mini-batch (MC samples) and finally averaging over all 100 samples (for NSMs), 5 runs and over the 10 last epochs. Prefix, d-deterministic, b-blank-out, g-Gaussian. The * indicates a network that has not been initialized with pre-trained weights (see main text).

Dataset	**Model**	**Error**
CIFAR10/100	bNSM	9.98% / 34.85%
CIFAR10/100	gNSM	10.35% / 34.84%
CIFAR10/100	dCNN	10.47% / 34.37%
CIFAR10/100	bNSM*	9.94% / 35.19%
CIFAR10/100	gNSM*	9.81% / 34.93%

Table 4: Classification error on DVS Gestures data set. Error is estimated by averaging test errors over 100 samples and over the 10 last epochs. Prefix, d-deterministic, b-blank-out, g-Gaussian.

Dataset	**Model**	**Error**
DVS Gestures	IBM EEDN	8.23%
DVS Gestures	bNSM	8.56%
DVS Gestures	gNSM	8.83%
DVS Gestures	dCNN	9.16%

3.5 Supervised Classification Experiments: DVS Gestures

Binary neural networks, such as the NSM are particularly suitable for discrete or binary data. Neuromorphic sensors such as Dynamic Vision Sensors (DVS) that output streams of events fall into this category and can transduce visual or auditory spatiotemporal patterns into parallel, microsecond-precise streams of events [29].

Amir *et al.* recorded DVS Gesture data set using a Dynamical Vision Sensor (DVS), comprising 1342 instances of a set of 11 hand and arm gestures, collected from 29 subjects under 3 different lighting conditions. Unlike standard imagers, the DVS records streams of events that signal the temporal intensity changes at each of its 128×128 pixels. The unique features of each gesture are embedded in the stream of events. To process these streams, we closely follow the pre-processing in [5], where event streams were downsized to 64×64 and binned in frames of $16\mathrm{ms}$. The input of the neural was formed by 6 frames (channels) and only ON (positive polarity) events were used. Similarly to [5], 23 subjects are used for the training set, and the remaining 6 subjects are reserved for testing. We note that the network used in this work is much smaller than the one used in [5].

We adapted a model based on the all convolutional networks of [49]. Compared to the original model, our adaptation includes an additional group of three convolutions and one pooling layer to account for the larger image size compared to the CIFAR10 data set used in [49] and a number of output classes that matches those of the DVS Gestures data set (11 classes). See SI Tab. 7 for a detailed listing of the layers. We trained the network for 200 epochs using a batch size 100. For the NSM network we initialized the weights using the converged weights of the deterministic network. This makes learning more robust and causes a faster convergence.

We find that the smaller models of [49] (in terms of layers and number of neurons) are faster to train and perform equally well when executed on GPU compared to the EEDN used in [5]. The models reported in Amir *et al.* were optimized for implementation in digital neuromorphic hardware, which strongly constrains weights, connectivity and neural activation functions in favor of energetic efficiency.

4 Conclusions

Stochasticity is a powerful mechanism for improving the computational features of neural networks, including regularization and Monte Carlo sampling. This work builds on the regularization effect of stochasticity in neural networks, and demonstrates that it naturally induces a normalizing effect on the activation function. Normalization is a powerful feature used in most modern deep neural networks [22, 45, 47], and mitigates internal covariate shift. Interestingly, this normalization effect may provide an alternative mechanism for divisive normalization in biological neural networks [10].

Our results demonstrate that NSMs can (i) outperform standard stochastic networks on standard machine learning benchmarks on convergence speed and accuracy, and (ii) perform close to deterministic feed-forward networks when data is of discrete nature. This is achieved using strictly simpler inference dynamics, that are well suited for emerging nanodevices, and argue strongly in favor of *exploiting* stochasticity in the devices for deep learning. Several implementation advantages accrue from this approach: it is an online alternative to batch normalization and dropout, it mitigates saturation at the boundaries of fixed range weight representations, and it confers robustness against certain spurious fluctuations affecting the rows of the weight matrix.

Although feed-forward passes in networks can be implemented free of multiplications, the weight update rule is more involved as it requires multiplications, calculating the row-wise $L2$-norms of the weight matrices, and the derivatives of the erf function. However, these terms are shared for all connections fanning into a neuron, such that the overhead in computing them is reasonably small. Furthermore, based on existing work, we speculate that approximating the learning rule either by hand [37] or automatically [6] can lead to near-optimal learning performances while being implemented with simple primitives.

References

[1] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[2] Maruan Al-Shedivat, Rawan Naous, Gert Cauwenberghs, and Khaled Nabil Salama. Memristors empower spiking neurons with stochasticity. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 5(2):242–253, 2015.

[3] Stefano Ambrogio, Simone Balatti, Antonio Cubeta, Alessandro Calderoni, Nirmal Ramaswamy, and Daniele Ielmini. Statistical fluctuations in hfo x resistive-switching memory: part i-set/reset variability. *IEEE Transactions on electron devices*, 61(8):2912–2919, 2014.

[4] Stefano Ambrogio, Simone Balatti, Antonio Cubeta, Alessandro Calderoni, Nirmal Ramaswamy, and Daniele Ielmini. Statistical fluctuations in hfo x resistive-switching memory: Part ii—random telegraph noise. *IEEE Transactions on Electron Devices*, 61(8):2920–2927, 2014.

[5] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7243–7252, 2017.

[6] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[7] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[8] Tiago Branco and Kevin Staras. The probability of neurotransmitter release: variability and feedback control at single synapses. *Nature Reviews Neuroscience*, 10(5):373, 2009.

[9] L. Buesing, J. Bill, B. Nessler, and W. Maass. Neural dynamics as sampling: A model for stochastic computation in recurrent networks of spiking neurons. *PLoS Computational Biology*, 7(11):e1002211, 2011.

[10] Matteo Carandini and David J Heeger. Normalization as a canonical neural computation. *Nature Reviews Neuroscience*, 13(1):nrn3136, 2011.

[11] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

[12] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. In *Advances in neural information processing systems*, pages 705–712, 1995.

[13] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[14] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[15] S. B. Eryilmaz, E. Neftci, S. Joshi, S. Kim, M. BrightSky, H. L. Lung, C. Lam, G. Cauwenberghs, and H. S. P. Wong. Training a probabilistic graphical model with resistive switching electronic synapses. *IEEE Transactions on Electron Devices*, 63(12):5004–5011, Dec 2016.

[16] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[18] Shixiang Gu, Sergey Levine, Ilya Sutskever, and Andriy Mnih. Muprop: Unbiased backpropagation for stochastic neural networks. *arXiv preprint arXiv:1511.05176*, 2015.

[19] Julia J Harris, Renaud Jolivet, and David Attwell. Synaptic energy use and supply. *Neuron*, 75(5):762–777, 2012.

[20] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[21] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[23] Matthew Jerry, Pai-Yu Chen, Jianchi Zhang, Pankaj Sharma, Kai Ni, Shimeng Yu, and Suman Datta. Ferroelectric fet analog synapse for acceleration of deep neural network training. In *2017 IEEE International Electron Devices Meeting (IEDM)*, pages 6–2. IEEE, 2017.

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

[25] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[27] Dong-Hyun Lee, Saizheng Zhang, Antoine Biard, and Yoshua Bengio. Target propagation. *arXiv preprint arXiv:1412.7525*, 2014.

[28] William B Levy and Robert A Baxter. Energy-efficient neuronal computation via quantal synaptic failures. *The Journal of Neuroscience*, 22(11):4746–4755, 2002.

[29] S.-C. Liu and T. Delbruck. Neuromorphic sensory systems. *Current Opinion in Neurobiology*, 20(3):288–295, 2010.

[30] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[31] Mark D McDonnell and Lawrence M Ward. The benefits of noise in neural systems: bridging theory and experiment. *Nature Reviews Neuroscience*, 12(7), 2011.

[32] Rubén Moreno-Bote. Poisson-like spiking in circuits with probabilistic synapses. *PLoS computational biology*, 10(7):e1003522, 2014.

[33] Halid Mulaosmanovic, Thomas Mikolajick, and Stefan Slesazeck. Random number generation based on ferroelectric switching. *IEEE Electron Device Letters*, 39(1):135–138, 2017.

[34] Rawan Naous, Maruan AlShedivat, Emre Neftci, Gert Cauwenberghs, and Khaled Nabil Salama. Memristor-based neural networks: Synaptic versus neuronal stochasticity. *AIP Advances*, 6(11):111304, 2016.

[35] Radford M Neal. Learning stochastic feedforward networks. *Department of Computer Science, University of Toronto*, page 64, 1990.

[36] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs. Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in Neuroscience*, 7(272), Jan. 2014.

[37] Emre Neftci, Charles Augustine, Somnath Paul, and Georgios Detorakis. Event-driven random back-propagation: Enabling neuromorphic deep learning machines. In *2017 IEEE International Symposium on Circuits and Systems*, May 2017.

[38] Emre O Neftci, Bruno Umbria Pedroni, Siddharth Joshi, Maruan Al-Shedivat, and Gert Cauwenberghs. Stochastic synapses enable efficient brain-inspired learning machines. *Frontiers in Neuroscience*, 10(241), 2016.

[39] Garrick Orchard, Ajinkya Jayawant, Gregory K. Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9, nov 2015.

[40] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[41] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[42] Damien Querlioz, Olivier Bichler, Adrien Francis Vincent, and Christian Gamrat. Bioinspired programming of memory devices for implementing an inference engine. *Proceedings of the IEEE*, 103(8):1398–1416, 2015.

[43] Tapani Raiko, Mathias Berglund, Guillaume Alain, and Laurent Dinh. Techniques for learning binary stochastic feedforward neural networks. *arXiv preprint arXiv:1406.2989*, 2014.

[44] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[45] Mengye Ren, Renjie Liao, Raquel Urtasun, Fabian H Sinz, and Richard S Zemel. Normalizing the normalizers: Comparing and extending network normalization schemes. *arXiv preprint arXiv:1611.04520*, 2016.

[46] Jason Tyler Rolfe. Discrete variational autoencoders. *arXiv preprint arXiv:1609.02200*, 2016.

[47] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.

[48] Oran Shayer, Dan Levi, and Ethan Fetaya. Learning discrete weights using the local reparameterization trick. *arXiv preprint arXiv:1710.07739*, 2017.

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

[50] Yichuan Tang and Ruslan Salakhutdinov. A new learning algorithm for stochastic feedforward neural nets. In *ICML'2013 Workshop on Challenges in Representation Learning*, 2013.

[51] B Walmsley, FR Edwards, and DJ Tracey. The probabilistic nature of synaptic transmission at a mammalian excitatory central synapse. *Journal of Neuroscience*, 7(4):1037–1046, 1987.

[52] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

[53] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[54] S. Yu, Z. Li, P. Chen, H. Wu, B. Gao, D. Wang, W. Wu, and H. Qian. Binary neural network with 16 mb rram macro chip for classification and online training. In *2016 IEEE International Electron Devices Meeting (IEDM)*, pages 16.2.1–16.2.4, Dec 2016.

Supplementary Information

4.1 Table of Abbreviations

Table 5: Abbreviations used in the main text and in the SI.

Abbreviation	**Definition**
NSM	Neural Sampling Machine
b/gNSM	Bernoulli/Gaussian NSM
S2M	Synaptic Sampling Machine
StNN	Stochastic Neural Network
BNN	Binary Neural Networks
DVS	Dynamic Vision Sensor
BD	Deterministic Binary Network with sgn as non-linearity
wBD	Same as BD with weight normalization enabled
SN	Stochastic network with noisy rectifier
BN	Binary network
STE	Straight-Through Estimator

4.2 Computation of Gradients in NSM Computational Graph

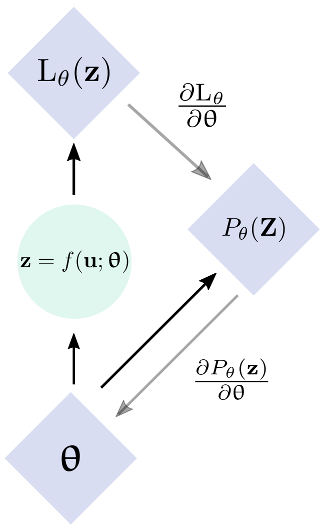
$\mathrm{L}_\theta(\mathbf{z})$
$\frac{\partial \mathrm{L}_\theta}{\partial \theta}$
$P_\theta(\mathbf{Z})$
$\mathbf{z} = f(\mathbf{u};\boldsymbol{\theta})$
$\frac{\partial P_\theta(\mathbf{z})}{\partial \theta}$
θ

Figure 3: Gradient estimation in NSM computation graph. For the NSM network the gradient $\nabla_\theta L(x)$ is computed via back-propagation on the probability $P_\theta(\mathbf{z})$ only in the backward pass (see equation (3) and main text). The light-green node indicates a stochastic discrete node that propagates the activity of units to the next layer only in the forward pass. The parameters here are $\boldsymbol{\theta} = (\mathbf{w}, \boldsymbol{\beta})$ (see main text).

4.3 NSM in Convolutional Neural Networks

CNN perform state-of-the-art in several visual, auditory and natural language tasks by assuming prior structure to the connectivity and the weight matrices [26, 17]. The NSM with stochastic neurons can be similarly extended to the convolution operation as follows (bias parameters omitted):

$$\begin{aligned} u_{ijk} &= Conv(\mathbf{w}_k, \mathbf{z}) \\ &= \sum_{q=1}^{Q}\sum_{m=1}^{H}\sum_{n=1}^{V}(\xi_{i+m,j+n,q} + a_{ijk}) w_{m,n,q,k} z_{i+m,j+n,q} \end{aligned} \tag{9}$$

where Q is the number of input channels and H, V are height and width of the filter, respectively. In the case of neural stochasticity, existing software libraries of the convolution can be used. In contrast, synaptic stochasticity, requires modification of such libraries due to the sharing of the filter parameters. While it is possible to do so, we have not observed significant differences in using neural or synaptic stochasticity. Therefore only neural stochasticity is used for convolution operations. Similarly to the case without convolutions, the activation probability becomes:

$$\begin{aligned} P(z_{ijk} = 1|\mathbf{z})\frac{1}{2}\left(1 + \operatorname{erf}\left(Conv(\mathbf{v}_k, \mathbf{z})\right)\right), \\ \text{with } \mathbf{v}_i = \beta_{ijk}\frac{\mathbf{w}_k}{||\mathbf{w}_k||}, \end{aligned} \tag{10}$$

where,

$$||\mathbf{w}_k|| = \sqrt{\sum_{m=1}^{H}\sum_{n=1}^{V}\sum_{q=1}^{Q} w_{m,n,q,k}^2}. \tag{11}$$

4.4 Derivation of Gradients (Equations (6) and (7))

In this section we derive equations (6) and (7). Therefore, if we differentiate through $\mathbf{v}_i = \beta_i \frac{\mathbf{w}_i}{||\mathbf{w}_i||}$, we obtain equation (6) from

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}_i} &= \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\frac{\partial \mathbf{v}_i}{\partial \boldsymbol{\beta}_i} \\ &= \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\frac{\mathbf{w}_i}{||\mathbf{w}_i||} \\ &= \frac{\sum_j w_{ij}\partial_{v_{ij}}\mathcal{L}}{||\mathbf{w}_i||}. \end{aligned} \tag{12}$$

And it is obvious that equation (6) is equation (12). For obtaining equation (7) we have

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{w}_i} &= \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\frac{\partial \mathbf{v}_i}{\partial \mathbf{w}_i} \\ &= \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\frac{\beta_i \partial \frac{\mathbf{w}_i}{||\mathbf{w}_i||}}{\partial \mathbf{w}_i} \\ &= \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\left(\beta_i \frac{\frac{\partial \mathbf{w}_i}{\partial \mathbf{w}_i}||\mathbf{w}_i|| - \mathbf{w}_i\frac{\partial||\mathbf{w}_i||}{\partial \mathbf{w}_i}}{||\mathbf{w}_i||^2}\right) \\ &= \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\left(\beta_i \frac{||\mathbf{w}_i|| - \mathbf{w}_i\frac{\sum_j w_{ij}}{||\mathbf{w}_i||}}{||\mathbf{w}_i||^2}\right) \\ &= \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\frac{\beta_i||\mathbf{w}_i||}{||\mathbf{w}_i||^2} - \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\frac{\beta_i \mathbf{w}_i \sum_j w_{ij}}{||\mathbf{w}_i||^2||\mathbf{w}_i||} \\ &= \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\frac{\beta_i}{||\mathbf{w}_i||} - \frac{\beta_i}{||\mathbf{w}_i||^2}\frac{\partial \mathcal{L}}{\partial \mathbf{v}_i}\frac{\mathbf{w}_i}{||\mathbf{w}_i||}\sum_j w_{ij} \\ &= \frac{\beta_i}{||\mathbf{w}_i||}\partial_{v_{ij}}\mathcal{L} - \frac{\beta_i \mathbf{w}_i}{||\mathbf{w}_i||^2}\frac{\sum_j w_{ij}\partial_{v_{ij}}\mathcal{L}}{||\mathbf{w}_i||} \\ &= \frac{\beta_i}{||\mathbf{w}_i||}\partial_{v_{ij}}\mathcal{L} - \frac{\beta_i}{||\mathbf{w}_i||^2}\mathbf{w}_i \partial_{\beta_i}\mathcal{L}. \end{aligned} \tag{13}$$

Therefore, we have derived equation (7) (which is equation (13)).

4.5 Robustness to Weight Fluctuations

The decoupling of the weight matrix (*i.e.*, $\mathbf{v}_i = \beta_i \frac{\mathbf{w}_i}{||\mathbf{w}_i||}$) introduces a robustness to weights fluctuation. During learning, the distribution of the weights for each layer tends to remain more stable in NSM compared to StNN. See for instance Figure 4, where in the top row the evolution of weights distribution of the third layer ($\mathbf{W}_3$) is shown for the NSM and the StNN, respectively. It is apparent that the distribution of NSM weights is more narrow and remains concentrated around its mean (low variance). On the other hand, the variance of the weight distribution in larger in the StNN. The same results are illustrated in the two bottom panels where the mean of the weights of the third layer over training is subtracted from the mean of the initial weights. We observe that the NSM is more robust and the mean remains almost steady (left panel) in comparison to StNN. The same phenomenon is observed also in the case of standard deviation (right panel), where the NSM's standard deviation increases slightly in comparison to StNNs.

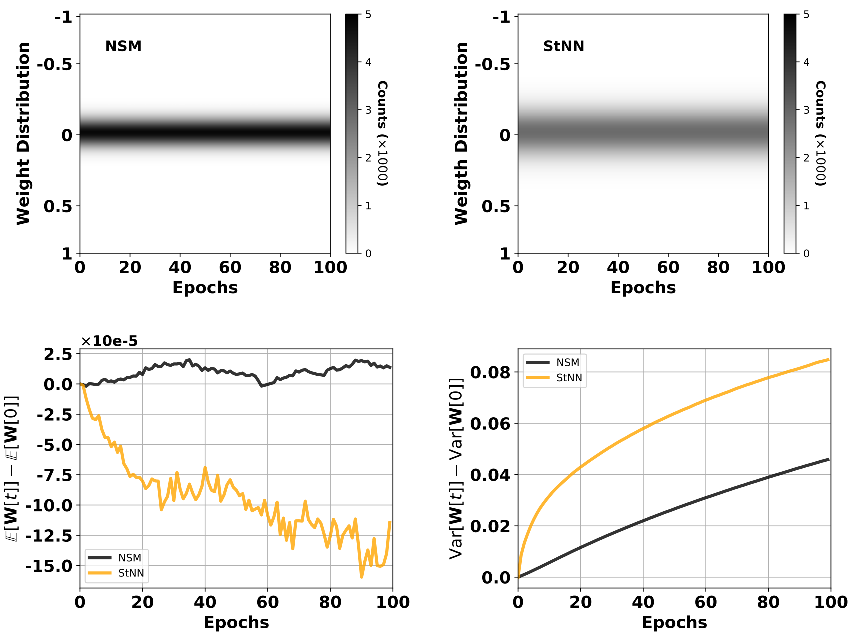

Figure 4: Evolution of W_3 (*i.e.*, weights of the third layer) weight distributions during learning, normalized to initial values (top row). In the NSM, the scale of the weights is controlled by the factors β_i. This renders the weights during learning more stable (left panel, top row) compared to the sigmoid neural network (right panel, top row), which tends to grow at a faster rate. The mean of NSM remains close to zero (black line, bottom left panel) in comparison to the mean of the StNN (yellow line). Similar to the mean, the variance of NSM (black line, bottom right panel) grows slower and remains smaller than that of StNN (yellow line, bottom right panel).

4.6 Training NSMs with BinConcrete

This section details how the NSM can be trained using the BinConcrete distribution instead of propagating gradients through the activation probability function (see main text and SI 4.2). In the forward pass, the probability is computed using equation (3) and then passed to the BinConcrete [30] given by the following equation

$$X = \sigma\Big(\frac{L + \log(\alpha)}{\lambda}\Big), \tag{14}$$

where α is the probability we have already computed, σ is the sigmoid function, L is the Logistic distribution $(\log(U) - \log(1-U)$, where U is the uniform distribution in the $[0, 1]$) and λ is the temperature term. In our experiments, we assume that $\lambda = 1$. In the backward pass, the gradients are computed through equation (14) instead of equation (3).

4.7 N-MNIST

The N-MNIST data set uses the same digits as contained in MNIST [39]. The digits were presented to an event-based camera that detects temporal contrast (ATIS), and their output was recorded. The data set consists of binary files, each containing the information of a single digit. Each file contains four arrays of equal length describing: the x coordinate and y coordinates of an event, the polarity (on or off) and the timestamp of the event. For this network, only the positive polarity events were extracted. Ten 34×34 frames of zeros were created for each digit and the maximum timestamp was divided by 10 to obtain the frame length. For each digit, an entry in the frame corresponding to the x and y coordinates of the events extracted inside the designated frame time was changed from 0 to 1. This was repeated for each of the ten frames. Test error results were obtained averaging test errors across the last 5 epochs and over X separate runs with different seed values.

4.8 Simulation Details

The source code for this work is written in Python and Pytorch [41] and it is available online under the GPL license at `https://github.com/nmi-lab/neural_sampling_machines`. We ran all the experiments on two machines:

1. A Ryzen ThreadRipper with 64GB physical memory running Arch Linux, Python $3.7.4$, Pytorch $1.2.0$ and GCC $9.1.0$, equipped with three Nvidia GeForce GTX 1080 Ti GPUs.
2. A Intel i7 with 64GB physical memory running Arch Linux, Python $3.7.3$, Pytorch $1.0.1$, and GCC $8.2.1$, equipped with two Nvidia GeForce RTX 2080 Ti GPUs.

4.9 MNIST, EMNIST, NMNIST Neural Networks

Table 6: Convolutional neural network used for MNIST, EMNIST, NMNIST data sets.

Layer Type	# Channels	x, y dimension
Raw Input	1	28
5×5 Conv	32	24
2×2 Max Pooling (stride 2)	32	12
5×5 Conv	64	8
2×2 Max Pooling (stride 2)	64	4
1024×512 FC	1024	1
Softmax output	10	1

4.10 DVS Gestures Neural Network

Table 7: All convolutional neural network used for the DVS Gestures dataset.

Layer Type	# Channels & Dimensions	
Input (ON events)	6	64×64
3×3 Conv	96	64×64
3×3 Conv	96	64×64
3×3 Conv	96	64×64
2×2 Max Pooling (stride 2)	96	32×32
3×3 Conv	192	32×32
3×3 Conv	192	32×32
3×3 Conv	192	32×32
2×2 Max Pooling (stride 2)	192	16×16
3×3 Conv	256	16×16
3×3 Conv	256	16×16
3×3 Conv	256	16×16
2×2 Max Pooling (stride 2)	256	8×8
3×3 Conv	256	8×8
1×1 Conv	256	8×8
1×1 Conv	256	8×8
Global average pool	256	1×1
Softmax	11	1×1

4.11 Weights Statistics for MNIST Classification

In this section we provide some statistics on the weights of the convolutional neural network used in the MNIST classification task. The network architecture is given in SI 4.9 and the results of the classification task are given in Table 2 in main text. First, we provide the histogram of the weights after training on the MNIST data set for three different types of networks. An NSM network, an NSM trained using the BinConcrete distribution and a deterministic network with sigmoid function as non-linearity. For more details about the networks see the main text, and SI 4.9 and 4.6. Histograms are illustrated in Figure 5. Then we measured the expected value of the weights for each layer and for each network as well as the mean gradients of the weights. Those results are shown in Figures 6 and 7, respectively. Finally, we show the angles between the gradients of NSM weights and NSM trained with BinConcrete (yellow) and NSM and Deterministic (orange) in Figure 8. These results indicate that the NSM and the NSM with BinConcrete express similar behavior during training. On the other hand, the deterministic network has larger weights and develops larger gradient steps (blue lines in Figures 6 and 7).

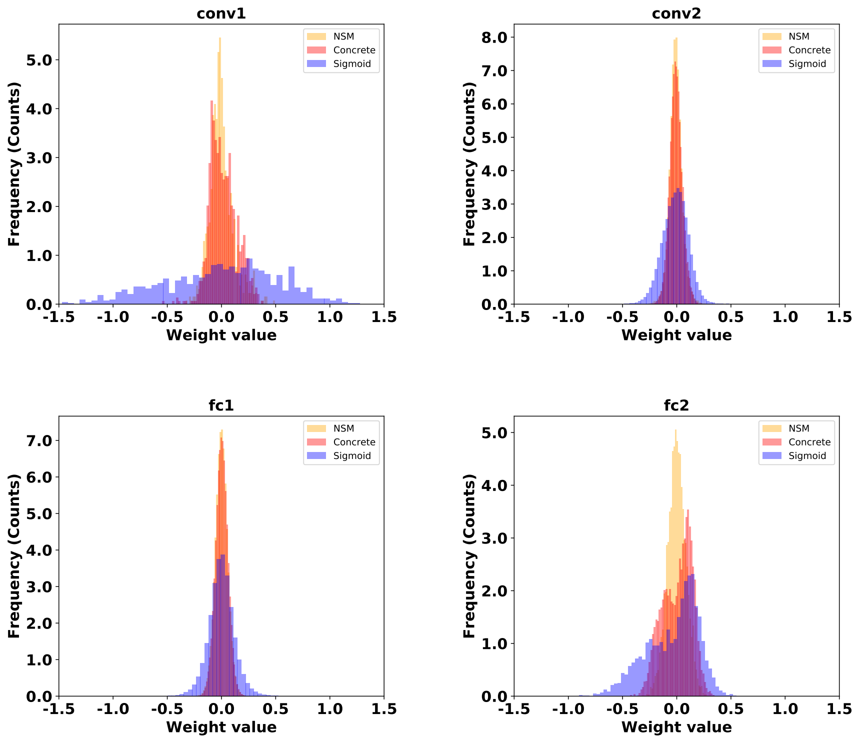

Figure 5: Histograms of Weights on MNIST Classification. The weights of four layers, convolutional layers 1 and 2 and fully connected layers 1 and 2 (see SI 4.9) for three different neural networks, NSM (yellow), NSM trained with Concrete Distribution (red, see SI 4.6), and Deterministic one with sigmoid as non-lineariry.

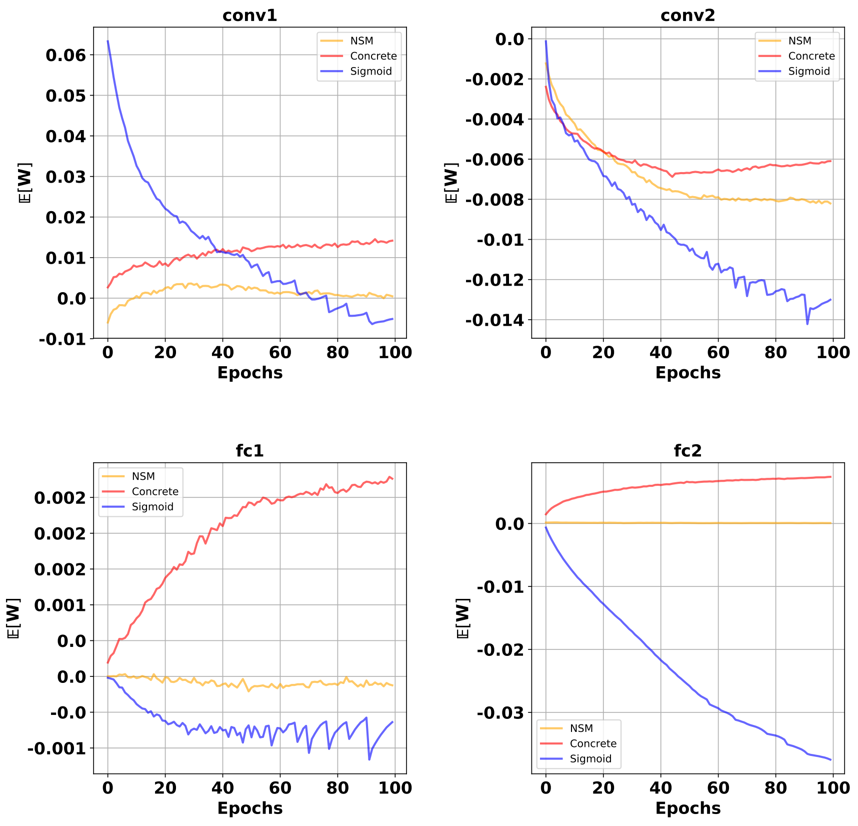

Figure 6: Mean of Weights on MNIST Classification. Mean weights of the four layers of the neural network used in main text for MNIST classification. Convolutional layers 1 and 2 and fully connected layers 1 and 2 (see SI 4.9). Weights of three different neural networks are presented here, NSM (yellow), NSM trained with BinConcrete Distribution (red, see SI 4.6), and a Deterministic one with sigmoid as non-linearity (blue).

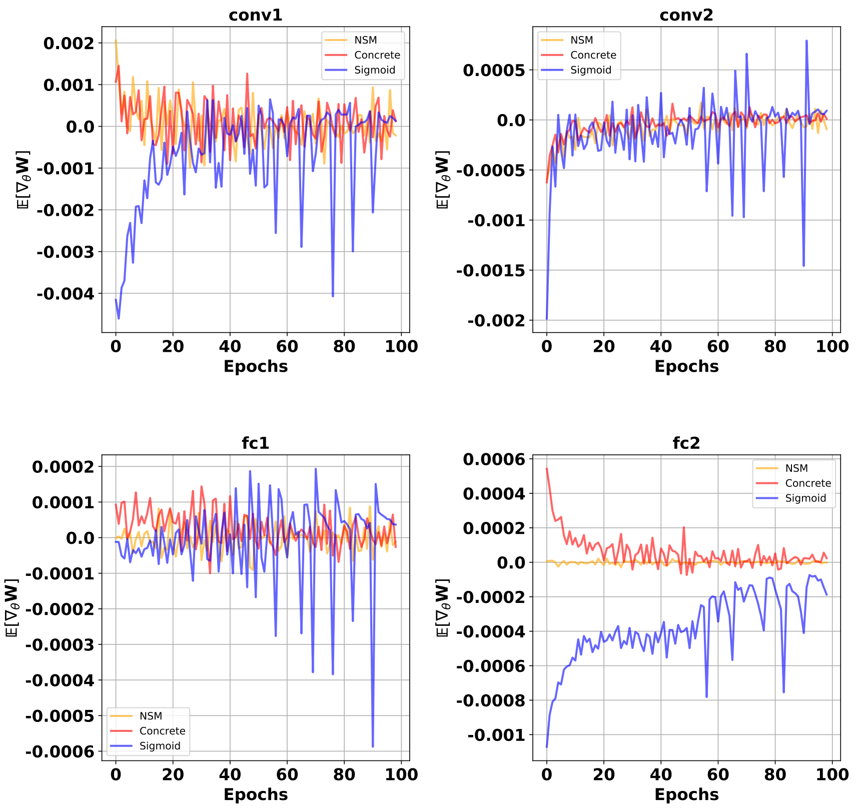

Figure 7: Mean of Weights Gradients on MNIST Classification. Mean of weights gradients of the four layers of the neural network used in main text for MNIST classification. Convolutional layers 1 and 2 and fully connected layers 1 and 2 (see SI 4.9). Weights of three different neural networks are presented here, NSM (yellow), NSM trained with BinConcrete Distribution (red, see SI 4.6), and a Deterministic one with sigmoid as non-linearity (blue).

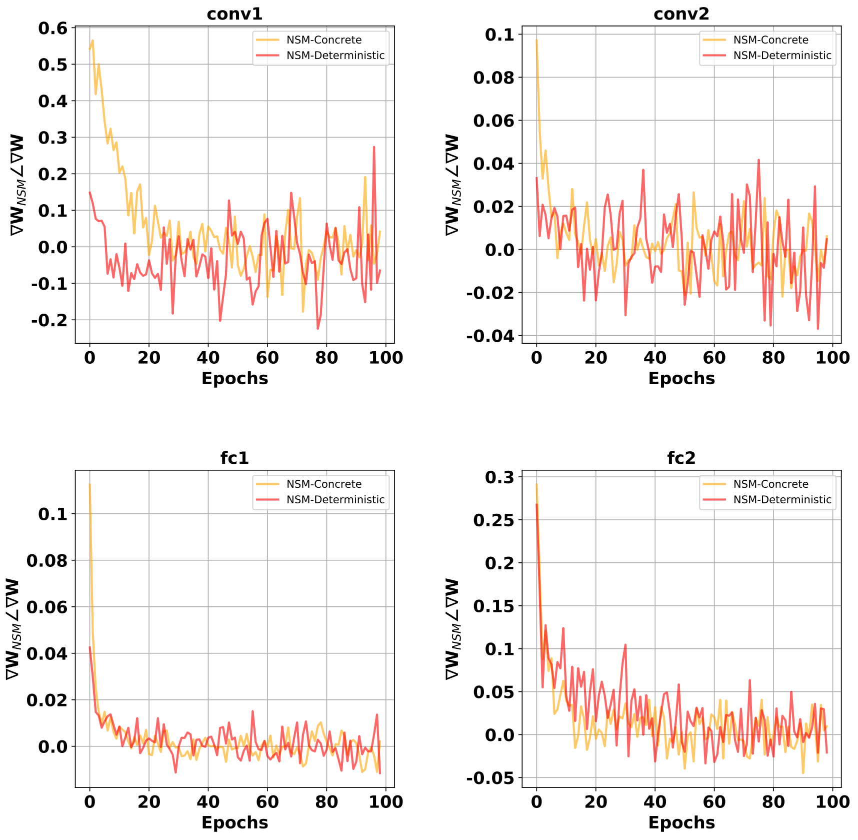

Figure 8: Angles (cosine similarity) of Gradients on MNIST Classification. The angles of gradients of weights of three networks are compared with each other. Four layers, convolutional layers 1 and 2 and fully connected layers 1 and 2 (see SI 4.9) are shown in this figure. Two cases are illustrated in this figure, (i) NSM against NSM trained with BinConcrete (yellow, see SI 4.6), (ii) NSM against a Deterministic network with sigmoid as non-linearity (red).