Inherent Weight Normalization in Stochastic Neural Networks

Georgios Detorakis[1], Sourav Dutta[3], Abhishek Khanna[3], Matthew Jerry[3], Suman Datta[3], Emre Neftci[1,2]
[1]Department of Cognitive Sciences, UC Irvine | [2]Department of Computer Science, UC Irvine | [3]Department of Electrical Engineering, University of Notre Dame

Introduction

This work introduces Neural Sampling Machines (NSM), a class of neural networks with binary threshold neurons that rely almost exclusively on multiplicative noise as a resource for inference and learning. 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 [1]. The always-on stochasticity of the NSM can exploit stochasticity inherent to a physical substrate such as analog non-volatile memories for in-memory computing, and is suitable for Monte Carlo sampling, while requiring almost exclusively addition and comparison operations.

Neural Sampling Machines (NSM)

Neurons in the NSM are binary, threshold (sign) units:

$$z_i = \mathrm{sgn}(u_i) = \begin{cases} 1 & \text{if } u_i \geq 0 \\ -1 & \text{if } u_i < 0 \end{cases} \qquad u_i = \sum_{j=1}^{N} \xi_{ij} w_{ij} z_j + b_i + \eta_i, \tag{1}$$

where u_i is the pre-activation of neuron i given by the following equation, and ξ_{ij} and η_i are *iid* multiplicative and additive noise terms.
The probability of the neuron being active is:

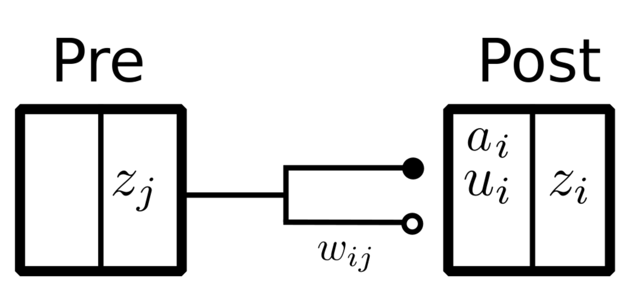
$$P(z_i = 1|\mathbf{z}) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\frac{\mathbb{E}(u_i|\mathbf{z})}{\sqrt{2\mathrm{Var}(u_i|\mathbf{z})}}\right)\right) \tag{2}$$

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

$$P(z_i = 1|\mathbf{z}) = \frac{1}{2}\left(1 + \mathrm{erf}\left(\mathbf{v}_i \cdot \mathbf{z}\right)\right) \qquad \text{with } \mathbf{v}_i = \beta_i \frac{\mathbf{w}_i}{||\mathbf{w}_i||_2}, \tag{3}$$

where β_i captures the parameters of the noise process ξ_i.
To control β_i without changing the distribution governing ξ, the NSM introduces a factor a_i in the preactivation's equation:

$$u_i = \sum_{j=1}^{N} (\xi_{ij} + a_i) w_{ij} z_j + b_i. \tag{4}$$



We consider Gaussian and Bernouilli noise as follows:

Gaussian Noise

$\xi \sim \mathcal{N}(1, \sigma^2)$
$\mathbb{E}(u_i|\mathbf{z}) = (1 + a_i)\sum_j w_{ij} z_j$
$\mathrm{Var}(u_i|\mathbf{z}) = \sigma^2 \sum_j w_{ij}^2$
$\beta_i = \frac{1+a_i}{\sqrt{2\sigma^2}}$

Bernoulli Noise

$\xi \sim Bern(p)$.
$\mathbb{E}(u_i|\mathbf{z}) = (p + a_i)\sum_j w_{ij} z_j$
$\mathrm{Var}(u_i|\mathbf{z}) = p(1-p)\sum_j w_{ij}^2$
$\beta_i = \frac{p+a_i}{\sqrt{2p(1-p)}}$

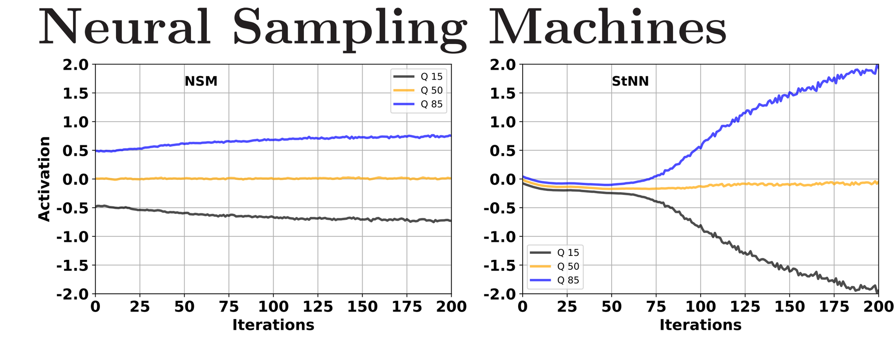
Weight Normalization and Covariate Shift

Weight normalization [1] normalizes unit activity by decoupling the magnitude and the direction of the weight vector

$$\mathbf{v}_i = \beta_i \frac{\mathbf{w}_i}{||\mathbf{w}_i||},$$

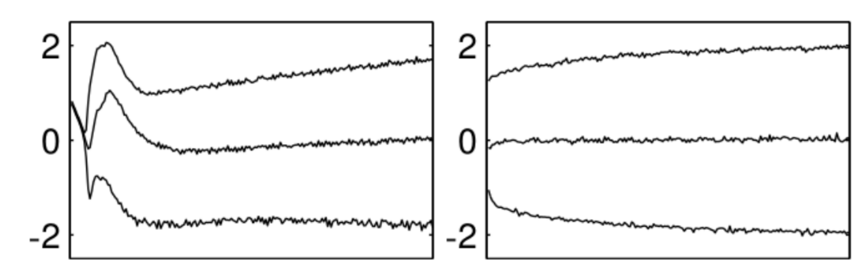
and training them separately. This speeds up convergence and confers many of the features of batch normalization. This is exactly the form obtained by introducing multiplicative noise.

Neural Sampling Machines

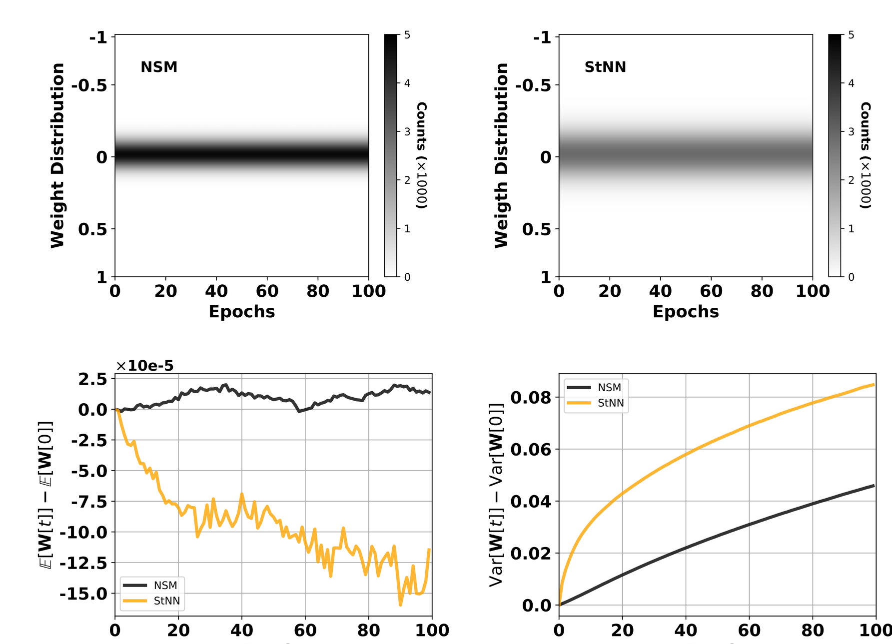


This suggests that NSMs inherently perform weight normalization in the sense of [1]. The 15th, 50th and 85th percentiles of the input distribution in the last hidden layer indeed behave similarly to batch normalization.

MLP with Batch Normalization [2]

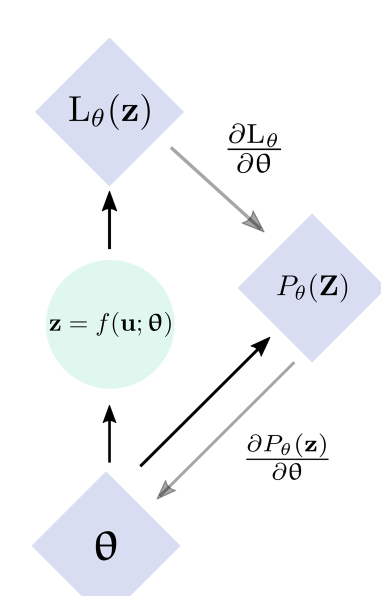


Without BN   With BN



Evolution of weights distribution of the third layer ($\mathbf{W}_3$) is shown for NSM and StNN (top row). The weights distribution remains contracted around its mean (low variance) for the NSM. Similarly, NSM's variance does not increase significantly (bottom row).

Implementation



Gradients were computed through equation (2). In the forward pass equation (1) is computed along with the probability (2) and the activity u passes through the non-linearity before propagates to the next layer. In the backward pass only the gradient of equation (2) is used to propagate the gradients.

Dataset	Network Model
*MNIST	CNN (32c5–p2–64c5–p2–1024–10)
CIFAR	AllConvNet (12 Layers) [1]
DVS Gesture	AllConvNet (18 Layers)

Results

Classification Benchmarks

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%

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

Comparison with Other Training Methods

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

bNSM: Bernouilli NSM
gNSM: Gaussian NSM
BD: Binary (sign non-linearity) Deterministic
STE: bNSM with Straight Through Estimator
wBD: BD with weight normalization
SN: Stochastic network (noisy rectifier [3])
BN: Deterministic Binary network trained with gradients estimated on erf

Conclusions

Motivated by the ubiquity of multiplicative noise in the physics of artificial and biological computing substrates, we explored Neural Sampling Machines. A striking self-normalizing effect fulfills a role that is similar to Weight Normalization during learning [1] similar to Batch Normalization [2]. 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 that can exploit (rather than mitigate) device non-idealities such as read stochasticity.

References

[1] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.
[2] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
[3] Y. Bengio, N. Léonard, et al. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.