Self Normalizing Neural Networks Paper Review

https://arxiv.org/pdf/1706.02515.pdf

Authors: Günter Klambauer, Thomas Unterthiner, Andreas Mayr, Sepp Hochreiter

Presenter for this talk: Hashiam Kadhim from DeepLearni.ng

11/10/2017

Outline

1. Problem/Motivation
2. Solution: SNNs
   a. High Level Picture
   b. Mathematical Details
3. Results

DL Excels in:

- Speech Recognition
- Natural Language Processing
- Visual Tasks

In general: data with spatial or temporal structure

DL in Tabular Data

No Neural Network Architectures that take advantage of underlying structure of the data, so we are left with normal Multilayer Perceptrons (MLPs)

DL in Tabular Data: Kaggle Results

Deep™ Learni.ng

Tabular Data in Kaggle

- support vector machines (SVMs) or tree ensemble methods (random forests, gradient boosted trees, etc.) are winning most of the competitions.
- In the rare cases where FNNs have won, the neural nets are usually shallow. Examples include:
  - HIGGS challenge
  - Merck Molecular Activity challenge
  - Tox21 Data challenge

Were all won by MLPs with at most 4 hidden layers

Existing Normalization Techniques

- One of the techniques that is attributed to the great success of CNNs:
  - Batch normalization evolved into a standard to normalize neuron activations to zero mean and unit variance
  - Other normalization schemes exist such as layer normalization and weight normalization

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Deep™ Learni.ng

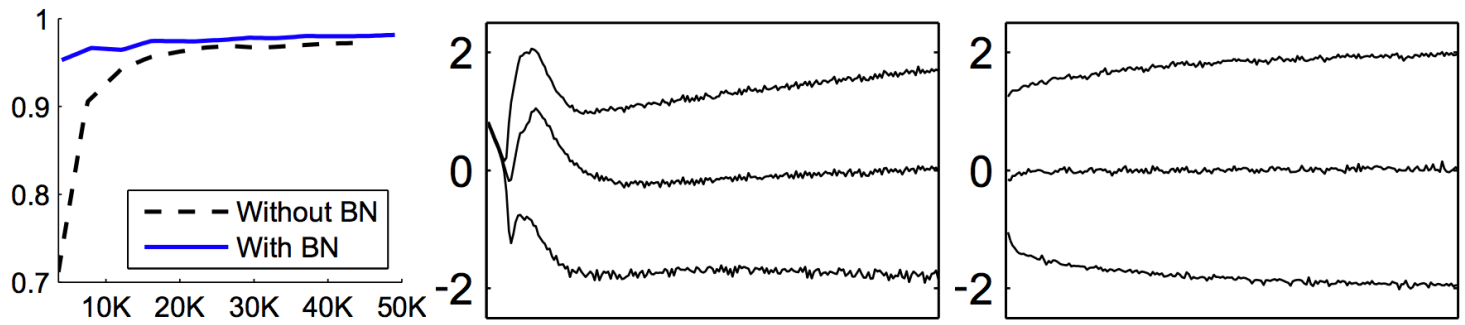


Figure 1: (a) *The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy.* (b, c) *The evolution of input distributions to a typical sigmoid, over the course of training, shown as* $\{15, 50, 85\}$*th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.*

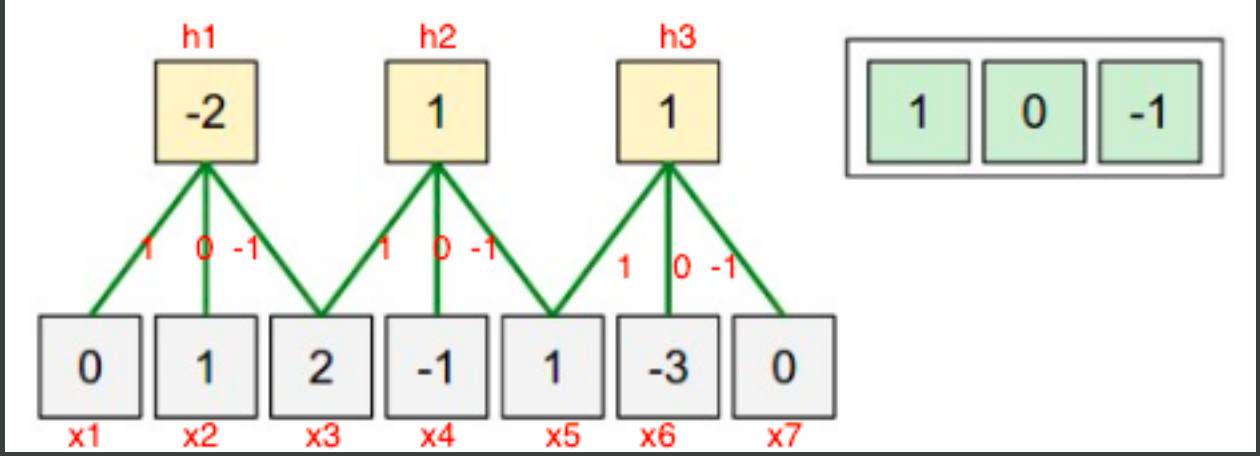
Problems with Normalization

Training with normalization techniques is perturbed by:

- Stochastic gradient descent
- Stochastic regularization (such as drop out)
- Estimation of normalization parameters

CNNs' and RNNs' rebuttal to these problems

Weight sharing in order to stabilize learning



Weight sharing doesn't make sense for FNNs

Deep™ Learni.ng

MLPs suffer from these perturbations

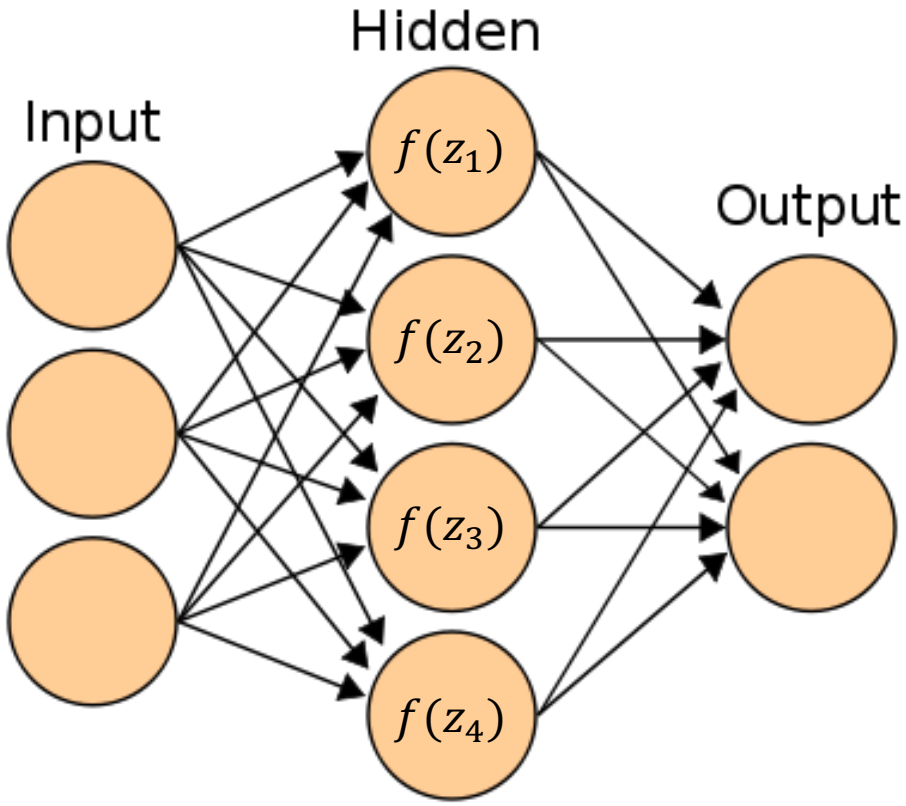
This leads to

- High variance in the training error
- Slower and hindered learning
- Strong regularization such as drop out further increases this variance, leading to divergence in the learning phase
- The authors believe that this sensitivity to perturbations is the reason that FNNs are less successful than RNNs and CNNs.

Deep™ Learni.ng

Enter Self Normalizing Neural Networks

The idea is to create a network where the network itself moves towards points where learning is optimal.



Suppose that f is an activation function for the hidden layers of this network. Then we want $f(\cdot)$ to be normally distributed for each hidden neuron (i.e. we want $f(z_i)$ to be sampled from a normal distribution).

Deep™ Learni.ng

Motivation

Network Properties

- Robust to perturbations
- Lower variance in training error

Big Picture

- Moving towards large AI systems
- Distributions are comparable between SNNs
  - they can communicate with each other and you can plug them together

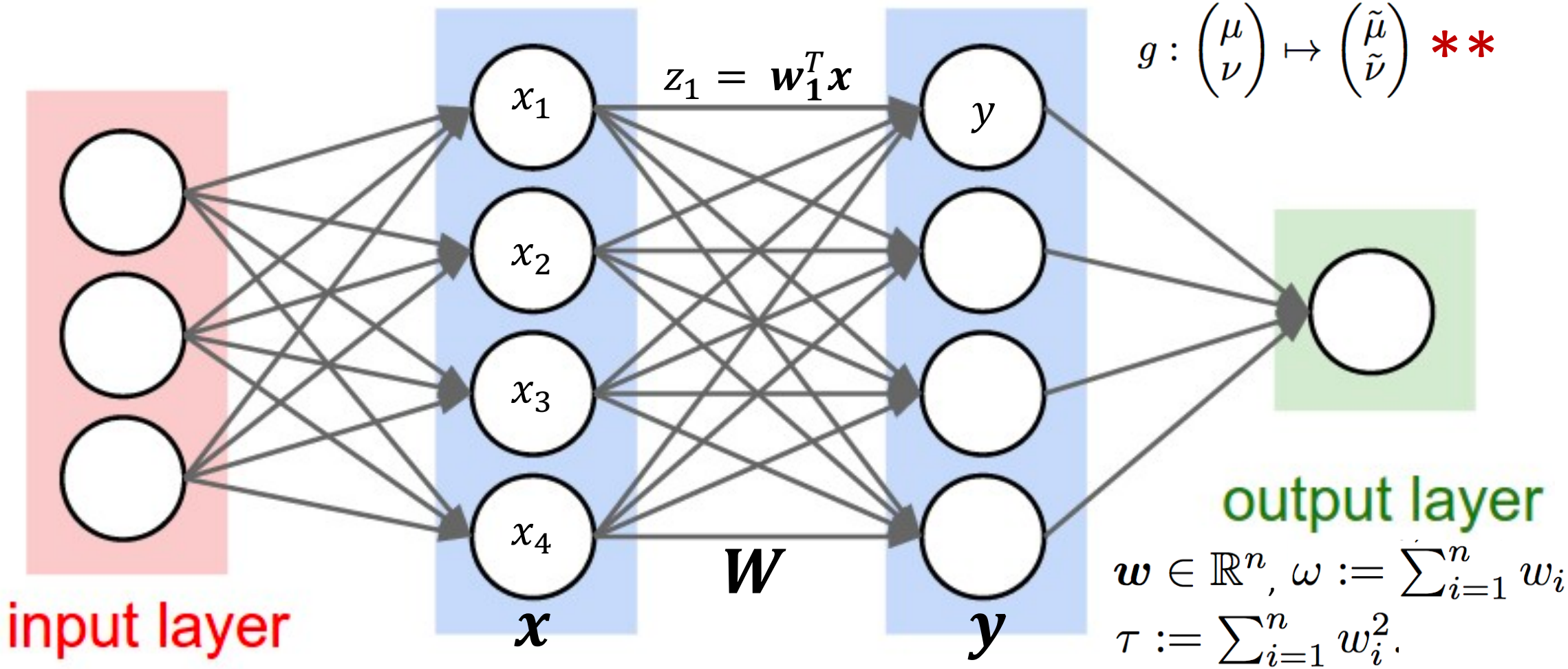
Mathematical Setup

Consider 2 consecutive layers in a neural network with:

- Activation function f
- Weight matrix $\boldsymbol{W}$
- The activations in the lower layer, $\boldsymbol{x}$
- The network inputs into the higher layer, $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$
- The activations of the higher layer, $\boldsymbol{y} = f(\boldsymbol{z})$
- Assume that $\boldsymbol{x}$ is a random variable, hence so is $\boldsymbol{y}$
- Assume that all x_i have mean and variance: $\mu = E(x_i), \nu = \mathrm{VAR}(x_i)$
- Assume that a single activation $y = f(\boldsymbol{w}^T\boldsymbol{x})$ has mean and variance: $\tilde{\mu} = E(y), \tilde{\nu} = \mathrm{VAR}(y)$

Mathematical Setup Continued

$$y = f(z_1)$$
$$\mu = E(x_i), \nu = \mathrm{VAR}(x_i)$$
$$\tilde{\mu} = E(y), \tilde{\nu} = \mathrm{VAR}(y)$$
$$g : \begin{pmatrix} \mu \\ \nu \end{pmatrix} \mapsto \begin{pmatrix} \tilde{\mu} \\ \tilde{\nu} \end{pmatrix} **$$



$$\boldsymbol{w} \in \mathbb{R}^n, \omega := \sum_{i=1}^n w_i$$
$$\tau := \sum_{i=1}^n w_i^2.$$

SNN Definition

Definition 1 (Self-normalizing neural net). *A neural network is self-normalizing if it possesses a mapping* $g : \Omega \mapsto \Omega$ *for each activation* y *that maps mean and variance from one layer to the next and has a stable and attracting fixed point depending on* (ω, τ) *in* Ω. *Furthermore, the mean and the variance remain in the domain* Ω, *that is* $g(\Omega) \subseteq \Omega$, *where* $\Omega = \{(\mu, \nu) \mid \mu \in [\mu_{\min}, \mu_{\max}], \nu \in [\nu_{\min}, \nu_{\max}]\}$. *When iteratively applying the mapping* g, *each point within* Ω *converges to this fixed point.*

SNN "Visual" Definition

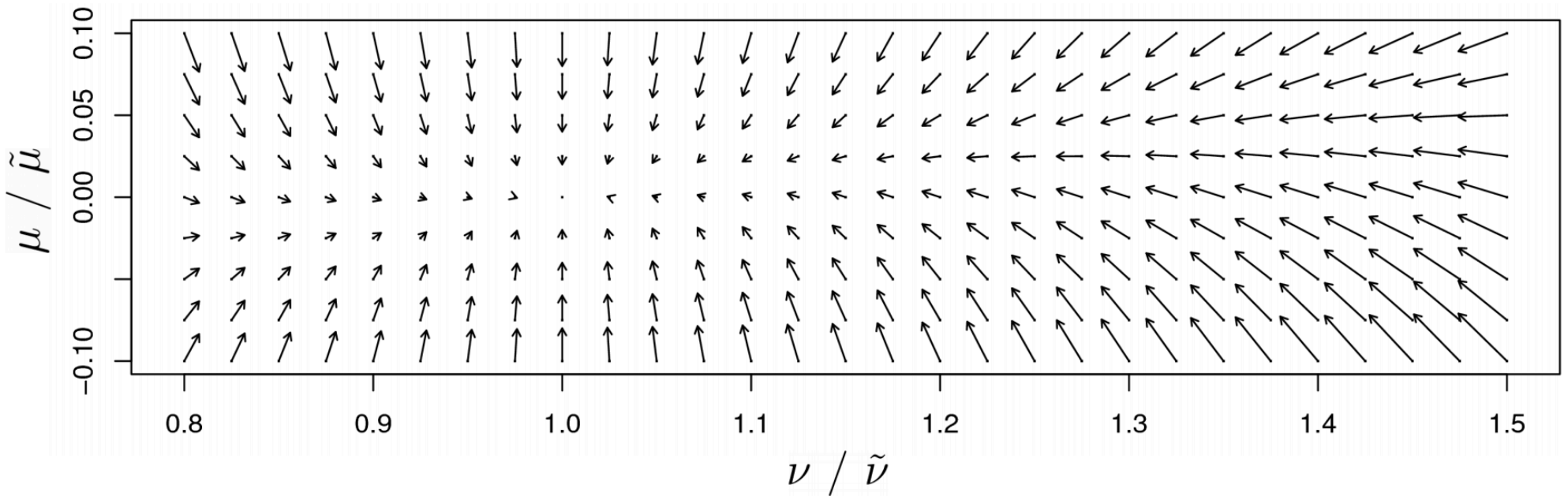


Figure 2: For $\omega = 0$ and $\tau = 1$, the mapping g of mean μ (x-axis) and variance ν (y-axis) to the next layer's mean $\tilde{\mu}$ and variance $\tilde{\nu}$ is depicted. Arrows show in which direction (μ, ν) is mapped by $g : (\mu, \nu) \mapsto (\tilde{\mu}, \tilde{\nu})$. The fixed point of the mapping g is $(0, 1)$.

Important Note

- The normalization effect is observed across layers of a network: in each layer the activations are getting closer to the fixed point.

- The normalization effect can also observed be for two fixed layers across learning steps: perturbations of lower layer activations or weights are damped in the higher layer by drawing the activations towards the fixed point

Constructing an SNN

Design Choices

Activation Function

- (1) negative and positive values for controlling the mean
- (2) saturation regions (derivatives approaching zero) to dampen the variance if it is too large in the lower layer
- (3) a slope larger than one to increase the variance if it is too small in the lower layer
- (4) a continuous curve to ensure a fixed point

Initialization:

- Authors suggest $\omega = 0, \tau = 1$

SNN Activation Function: the Selu

$$\text{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leqslant 0 \end{cases}$$

α and λ are fixed parameters that control the fixed stable point

Deriving the Mean and Variance Mapping Function g

We assume that the x_i are independent (or weakly dependent) from each other but share the same mean μ and variance ν

$$\mu = E(x_i), \nu = \mathrm{VAR}(x_i)$$

$$\boldsymbol{w} \in \mathbb{R}^n,\ \omega := \sum_{i=1}^{n} w_i\,,\ \tau := \sum_{i=1}^{n} w_i^2.$$

$z = \boldsymbol{w}^T\boldsymbol{x} = \sum w_i x_i$ This is where CLT is used (we have a sum of random variables)

$\boldsymbol{E}(\boldsymbol{z}) = \boldsymbol{E}(\boldsymbol{w}^T\boldsymbol{x}) = \mu\omega$ by independence of $\boldsymbol{w}$ and $\boldsymbol{x}$

$\mathbf{VAR}(\boldsymbol{z}) = \mathbf{VAR}(\boldsymbol{w}^T\boldsymbol{x}) = \nu\tau + \tau\mu^2 + \nu\omega^2 \approx \nu\tau$ assuming μ and ω are near 0

$$g : \begin{pmatrix} \mu \\ \nu \end{pmatrix} \mapsto \begin{pmatrix} \tilde{\mu} \\ \tilde{\nu} \end{pmatrix}$$

Lyapunov Central Limit Theorem with weak dependence of x_i

$y = \mathrm{selu}(z)$

$\tilde{\mu} = E(y)$

$$\tilde{\mu}(\mu, \omega, \nu, \tau) = \int_{-\infty}^{\infty} \mathrm{selu}(z)\, p_{\mathrm{N}}(z; \mu\omega, \sqrt{\nu\tau})\, \mathrm{d}z$$

$\tilde{\nu} = \mathrm{VAR}(y)$

$$\tilde{\nu}(\mu, \omega, \nu, \tau) = \int_{-\infty}^{\infty} \mathrm{selu}(z)^2\, p_{\mathrm{N}}(z; \mu\omega, \sqrt{\nu\tau})\, \mathrm{d}z - (\tilde{\mu})^2\,.$$

Deep Learni.ng

Deriving the Mean and Variance Mapping Function g

These integrals can be analytically computed and lead to following mappings of the moments:

$$\tilde{\mu} = \frac{1}{2}\lambda\left((\mu\omega)\,\mathrm{erf}\left(\frac{\mu\omega}{\sqrt{2}\sqrt{\nu\tau}}\right) + \alpha\, e^{\mu\omega+\frac{\nu\tau}{2}}\,\mathrm{erfc}\left(\frac{\mu\omega+\nu\tau}{\sqrt{2}\sqrt{\nu\tau}}\right) - \alpha\,\mathrm{erfc}\left(\frac{\mu\omega}{\sqrt{2}\sqrt{\nu\tau}}\right) + \sqrt{\frac{2}{\pi}}\sqrt{\nu\tau}e^{-\frac{(\mu\omega)^2}{2(\nu\tau)}} + \mu\omega\right) \tag{4}$$

$$\tilde{\nu} = \frac{1}{2}\lambda^2\left(\left((\mu\omega)^2+\nu\tau\right)\left(2-\mathrm{erfc}\left(\frac{\mu\omega}{\sqrt{2}\sqrt{\nu\tau}}\right)\right) + \alpha^2\left(-2e^{\mu\omega+\frac{\nu\tau}{2}}\,\mathrm{erfc}\left(\frac{\mu\omega+\nu\tau}{\sqrt{2}\sqrt{\nu\tau}}\right)\right.\right. \tag{5}$$

$$\left.\left. + e^{2(\mu\omega+\nu\tau)}\,\mathrm{erfc}\left(\frac{\mu\omega+2\nu\tau}{\sqrt{2}\sqrt{\nu\tau}}\right) + \mathrm{erfc}\left(\frac{\mu\omega}{\sqrt{2}\sqrt{\nu\tau}}\right)\right) + \sqrt{\frac{2}{\pi}}(\mu\omega)\sqrt{\nu\tau}e^{-\frac{(\mu\omega)^2}{2(\nu\tau)}}\right) - (\tilde{\mu})^2$$

Fixing $\mu, \tilde{\mu}, \omega = 0$ and $\nu, \tilde{\nu}, \tau = 1$, we can solve for λ and α in our selu activation function

They find that $\lambda = 1.0507$ and $\alpha = 1.6733$

Jacobian Insures Fixed Point Stable and Attracting

$$\mathcal{J}(\mu,\nu) = \begin{pmatrix} \partial\frac{\mu^{\text{new}}(\mu,\nu)}{\partial\mu} & \partial\frac{\mu^{\text{new}}(\mu,\nu)}{\partial\nu} \\ \partial\frac{\nu^{\text{new}}(\mu,\nu)}{\partial\mu} & \partial\frac{\nu^{\text{new}}(\mu,\nu)}{\partial\nu} \end{pmatrix}, \qquad \mathcal{J}(0,1) = \begin{pmatrix} 0.0 & 0.088834 \\ 0.0 & 0.782648 \end{pmatrix}. \tag{6}$$

The spectral norm of $\mathcal{J}(0,1)$ (its largest singular value) is $0.7877 < 1$. That means g is a contraction mapping around the fixed point $(0,1)$ (the mapping is depicted in Figure 2). Therefore, $(0,1)$ is a stable fixed point of the mapping g.

Back to the visualization

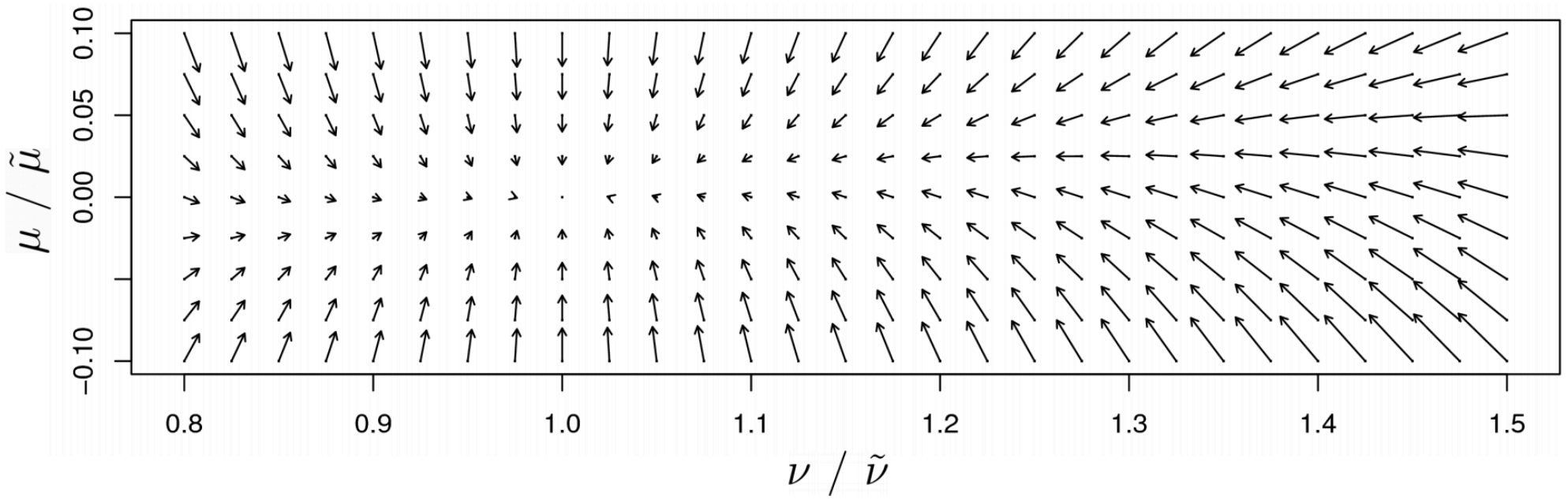


Figure 2: For $\omega = 0$ and $\tau = 1$, the mapping g of mean μ (x-axis) and variance ν (y-axis) to the next layer's mean $\tilde{\mu}$ and variance $\tilde{\nu}$ is depicted. Arrows show in which direction (μ, ν) is mapped by $g : (\mu, \nu) \mapsto (\tilde{\mu}, \tilde{\nu})$. The fixed point of the mapping g is $(0, 1)$.

Main Theorem

Theorem 1 (Stable and Attracting Fixed Points). *We assume* $\alpha = \alpha_{01}$ *and* $\lambda = \lambda_{01}$. *We restrict the range of the variables to the following intervals* $\mu \in [-0.1, 0.1]$, $\omega \in [-0.1, 0.1]$, $\nu \in [0.8, 1.5]$, *and* $\tau \in [0.95, 1.1]$, *that define the functions' domain* Ω. *For* $\omega = 0$ *and* $\tau = 1$, *the mapping Eq.* (3) *has the stable fixed point* $(\mu, \nu) = (0, 1)$, *whereas for other* ω *and* τ *the mapping Eq.* (3) *has a stable and attracting fixed point depending on* (ω, τ) *in the* (μ, ν)*-domain:* $\mu \in [-0.03106, 0.06773]$ *and* $\nu \in [0.80009, 1.48617]$. *All points within the* (μ, ν)*-domain converge when iteratively applying the mapping Eq.* (3) *to this fixed point.*

i.e. If (μ, ω, ν, τ) is close to (0,0,1,1) then $(\tilde{\mu}, \tilde{\nu})$ is close to (0,1)

Deep™ Learni.ng

Other Theorems

Theorem 2 (Decreasing ν). *For $\lambda = \lambda_{01}$, $\alpha = \alpha_{01}$ and the domain Ω^+: $-1 \leqslant \mu \leqslant 1$, $-0.1 \leqslant \omega \leqslant 0.1$, $3 \leqslant \nu \leqslant 16$, and $0.8 \leqslant \tau \leqslant 1.25$, we have for the mapping of the variance $\tilde{\nu}(\mu, \omega, \nu, \tau, \lambda, \alpha)$ given in Eq. (5): $\tilde{\nu}(\mu, \omega, \nu, \tau, \lambda_{01}, \alpha_{01}) < \nu$.*

i.e. variance is bounded from above (no more exploding gradients)

Theorem 3 (Increasing ν). *We consider $\lambda = \lambda_{01}$, $\alpha = \alpha_{01}$ and the domain Ω^-: $-0.1 \leqslant \mu \leqslant 0.1$, and $-0.1 \leqslant \omega \leqslant 0.1$. For the domain $0.02 \leqslant \nu \leqslant 0.16$ and $0.8 \leqslant \tau \leqslant 1.25$ as well as for the domain $0.02 \leqslant \nu \leqslant 0.24$ and $0.9 \leqslant \tau \leqslant 1.25$, the mapping of the variance $\tilde{\nu}(\mu, \omega, \nu, \tau, \lambda, \alpha)$ given in Eq. (5) increases: $\tilde{\nu}(\mu, \omega, \nu, \tau, \lambda_{01}, \alpha_{01}) > \nu$.*

i.e. variance is bounded from below (no more vanishing gradients)

New Drop Out Technique Proposed

Through an affine transformation of the surviving inputs, the proposed drop out technique aims to keep the original values of the mean and variance of the inputs.

Results

We compare SNNs to other deep networks at different benchmarks. Hyperparameters such as number of layers (blocks), neurons per layer, learning rate, and dropout rate, are adjusted by grid-search for each dataset on a separate validation set (see Section A4). We compare the following FNN methods:

- **"MSRAinit":** FNNs without normalization and with ReLU activations and "Microsoft weight initialization" [17].
- **"BatchNorm":** FNNs with batch normalization [20].
- **"LayerNorm":** FNNs with layer normalization [2].
- **"WeightNorm":** FNNs with weight normalization [32].
- **"Highway":** Highway networks [35].
- **"ResNet":** Residual networks [16] adapted to FNNs using residual blocks with 2 or 3 layers with rectangular or diavolo shape.
- **"SNNs":** Self normalizing networks with SELUs with $\alpha = \alpha_{01}$ and $\lambda = \lambda_{01}$ and the proposed dropout technique and initialization strategy.

Results

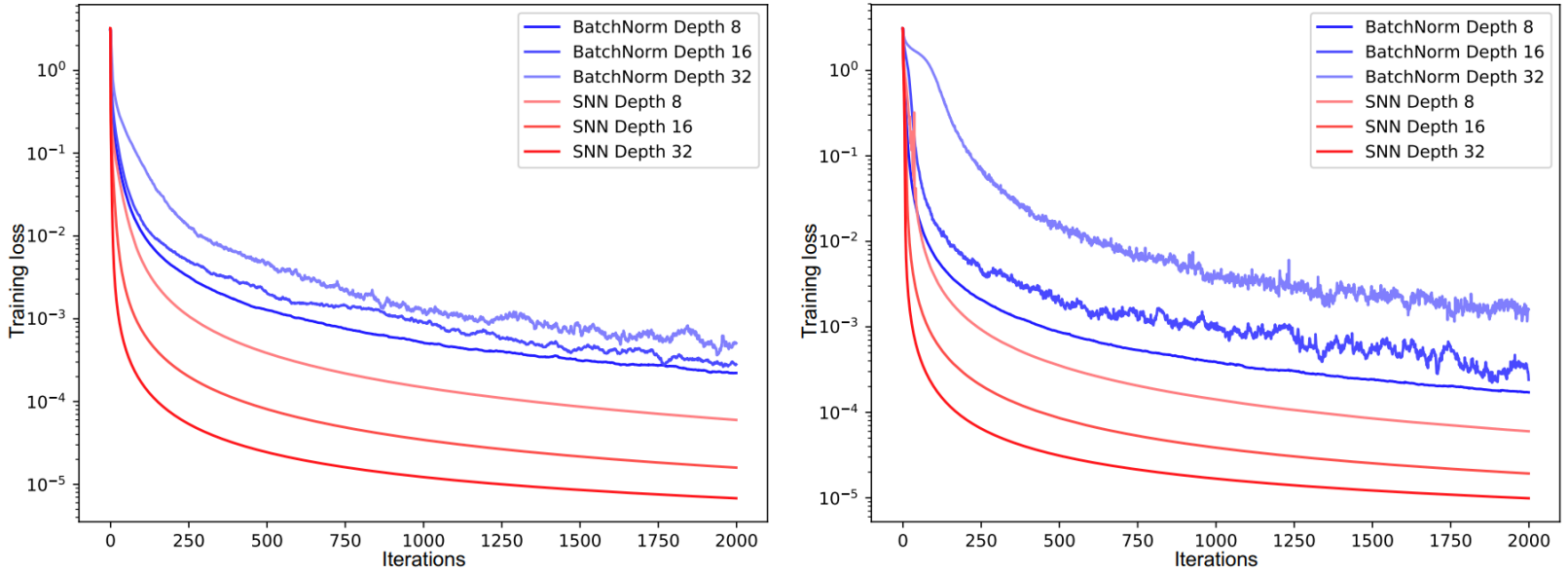


Figure 1: The left panel and the right panel show the training error (y-axis) for feed-forward neural networks (FNNs) with batch normalization (BatchNorm) and self-normalizing networks (SNN) across update steps (x-axis) on the MNIST dataset the CIFAR10 dataset, respectively. We tested networks with 8, 16, and 32 layers and learning rate 1e-5. FNNs with batch normalization exhibit high variance due to perturbations. In contrast, SNNs do not suffer from high variance as they are more robust to perturbations and learn faster.

Results

Table 1: **Left:** Comparison of seven FNNs on 121 UCI tasks. We consider the average rank difference to rank 4, which is the average rank of seven methods with random predictions. The first column gives the method, the second the average rank difference, and the last the p-value of a paired Wilcoxon test whether the difference to the best performing method is significant. SNNs significantly outperform all other methods. **Right:** Comparison of 24 machine learning methods (ML) on the UCI datasets with more than 1000 data points. The first column gives the method, the second the average rank difference to rank 12.5, and the last the p-value of a paired Wilcoxon test whether the difference to the best performing method is significant. Methods that were significantly worse than the best method are marked with "*". The full tables can be found in Table A11, Table A12 and Table A13. SNNs outperform all competing methods.

FNN method comparison			ML method comparison		
Method	avg. rank diff.	p-value	Method	avg. rank diff.	p-value
SNN	-0.756		SNN	-6.7	
MSRAinit	-0.240*	2.7e-02	SVM	-6.4	5.8e-01
LayerNorm	-0.198*	1.5e-02	RandomForest	-5.9	2.1e-01
Highway	0.021*	1.9e-03	MSRAinit	-5.4*	4.5e-03
ResNet	0.273*	5.4e-04	LayerNorm	-5.3	7.1e-02
WeightNorm	0.397*	7.8e-07	Highway	-4.6*	1.7e-03
BatchNorm	0.504*	3.5e-06	...	...	...

Results

Table 2: Comparison of FNNs at the Tox21 challenge dataset in terms of AUC. The rows represent different methods and the columns different network depth and for ResNets the number of residual blocks ("na": 32 blocks were omitted due to computational constraints). The deeper the networks, the more prominent is the advantage of SNNs. The best networks are SNNs with 8 layers.

	#layers / #blocks						
method	2	3	4	6	8	16	32
SNN	*83.7* $\pm$ 0.3	**84.4** $\pm$ 0.5	**84.2** $\pm$ 0.4	**83.9** $\pm$ 0.5	**84.5** $\pm$ 0.2	**83.5** $\pm$ 0.5	**82.5** $\pm$ 0.7
Batchnorm	80.0 $\pm$ 0.5	79.8 $\pm$ 1.6	77.2 $\pm$ 1.1	77.0 $\pm$ 1.7	75.0 $\pm$ 0.9	73.7 $\pm$ 2.0	76.0 $\pm$ 1.1
WeightNorm	*83.7* $\pm$ 0.8	82.9 $\pm$ 0.8	82.2 $\pm$ 0.9	*82.5* $\pm$ 0.6	*81.9* $\pm$ 1.2	78.1 $\pm$ 1.3	56.6 $\pm$ 2.6
LayerNorm	**84.3** $\pm$ 0.3	*84.3* $\pm$ 0.5	*84.0* $\pm$ 0.2	*82.5* $\pm$ 0.8	80.9 $\pm$ 1.8	78.7 $\pm$ 2.3	78.8 $\pm$ 0.8
Highway	83.3 $\pm$ 0.9	83.0 $\pm$ 0.5	82.6 $\pm$ 0.9	82.4 $\pm$ 0.8	80.3 $\pm$ 1.4	80.3 $\pm$ 2.4	79.6 $\pm$ 0.8
MSRAinit	82.7 $\pm$ 0.4	81.6 $\pm$ 0.9	81.1 $\pm$ 1.7	80.6 $\pm$ 0.6	80.9 $\pm$ 1.1	80.2 $\pm$ 1.1	80.4 $\pm$ 1.9
ResNet	82.2 $\pm$ 1.1	80.0 $\pm$ 2.0	80.5 $\pm$ 1.2	81.2 $\pm$ 0.7	81.8 $\pm$ 0.6	*81.2* $\pm$ 0.6	na

Deep
Learni.ng

Contact us

129 Spadina Ave.
Suite 600
Toronto, ON
M5V 2L3

www.deeplearni.ng
deploy@deeplearni.ng