FORNAX

Self normalizing neural networks and orthogonal initialization

Krzysztof Kolasiński, 27.06.2017

Content

- SELU activation function
- LSUV
- My benchmarks on CIFAR10
- Another benchmarks

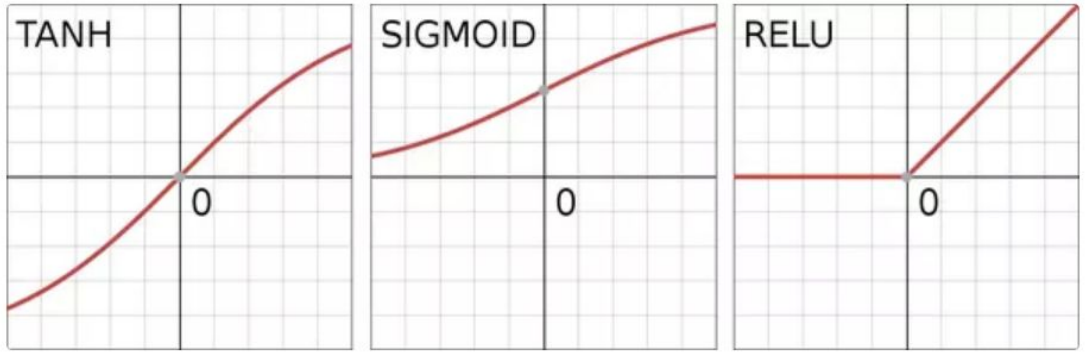
Self Normalizing Neural Networks (22 June 2017)

Self-Normalizing Neural Networks

Günter Klambauer **Thomas Unterthiner** **Andreas Mayr**

Sepp Hochreiter
Institute of Bioinformatics,
Johannes Kepler University Linz, Austria
{klambauer,unterthiner,mayr,hochreit}@bioinf.jku.at

The motivation for unit variance activations in NNs



source: goura

Self Normalizing Neural Networks

We consider dense layer with activation **f** and weight matrix **W**

$$\mathbf{z} = \mathbf{W}\mathbf{x}$$

The activations in higher layer are

$$\mathbf{y} = \mathbf{f}(\mathbf{z})$$

- if **x** are a random variables then **z** as well.
- The authors **assume** that all activations **x** have same mean and variance

$$\mu := \mathrm{E}(x_i) \qquad \nu := \mathrm{Var}(x_i)$$

$$\mathrm{Var}(X) = \mathrm{E}\left[(X - \mu)^2\right]$$

- Same for activations in higher layer

$$\tilde{\mu} := \mathrm{E}(y) \qquad \tilde{\nu} := \mathrm{Var}(y)$$

Self Normalizing Neural Networks

We consider dense layer with activation **f** and weight matrix **W**

$$\mathbf{z} = \mathbf{W}\mathbf{x}$$

The activations in higher layer are

$$\mathbf{y} = \mathbf{f}(\mathbf{z})$$

- if **x** are a random variables then **z** as well.
- The authors **assume** that all activations **x** have same mean and variance

$$\mu := \mathrm{E}(x_i) \qquad \nu := \mathrm{Var}(x_i)$$

- Same for activations in higher layer

$$\tilde{\mu} := \mathrm{E}(y) \qquad \tilde{\nu} := \mathrm{Var}(y)$$

$$\mathrm{Var}(X) = \mathrm{E}\left[(X - \mu)^2\right]$$

- Single activation is defined with weight vector: $z = \mathbf{w}^T\mathbf{x}$
- They define new variables: $\omega := \sum_{i=1}^{n} w_i \qquad \tau := \sum_{i=1}^{n} w_i^2$

Self Normalizing Neural Networks

We consider dense layer with activation **f** and weight matrix **W** $\mathbf{z} = \mathbf{W}\mathbf{x} \quad \mathbf{y} = \mathbf{f}(\mathbf{z})$

$$\mu := \mathrm{E}(x_i) \quad \nu := \mathrm{Var}(x_i) \qquad \tilde{\mu} := \mathrm{E}(y) \quad \tilde{\nu} := \mathrm{Var}(y)$$

- We define general mapping which one layer performs

$$\begin{pmatrix} \tilde{\mu} \\ \tilde{\nu} \end{pmatrix} = g \begin{pmatrix} \mu \\ \nu \end{pmatrix}$$

Normalization techniques like batch, layer, or weight normalization ensure a mapping g that keeps (μ, ν) and $(\tilde{\mu}, \tilde{\nu})$ close to predefined values, typically $(0, 1)$.

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.*

Self Normalizing Neural Networks

How does it work?

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

```python
def selu(x):
    """Scaled Exponential Linear Unit. (Klambauer et al., 2017)
    # Arguments
        x: A tensor or variable to compute the activation function for.
    # References
        - [Self-Normalizing Neural Networks](https://arxiv.org/abs/1706.02515)
    """
    alpha = 1.6732632423543772848170429916717
    scale = 1.0507009873554804934193349852946
    return scale * K.elu(x, alpha)
```

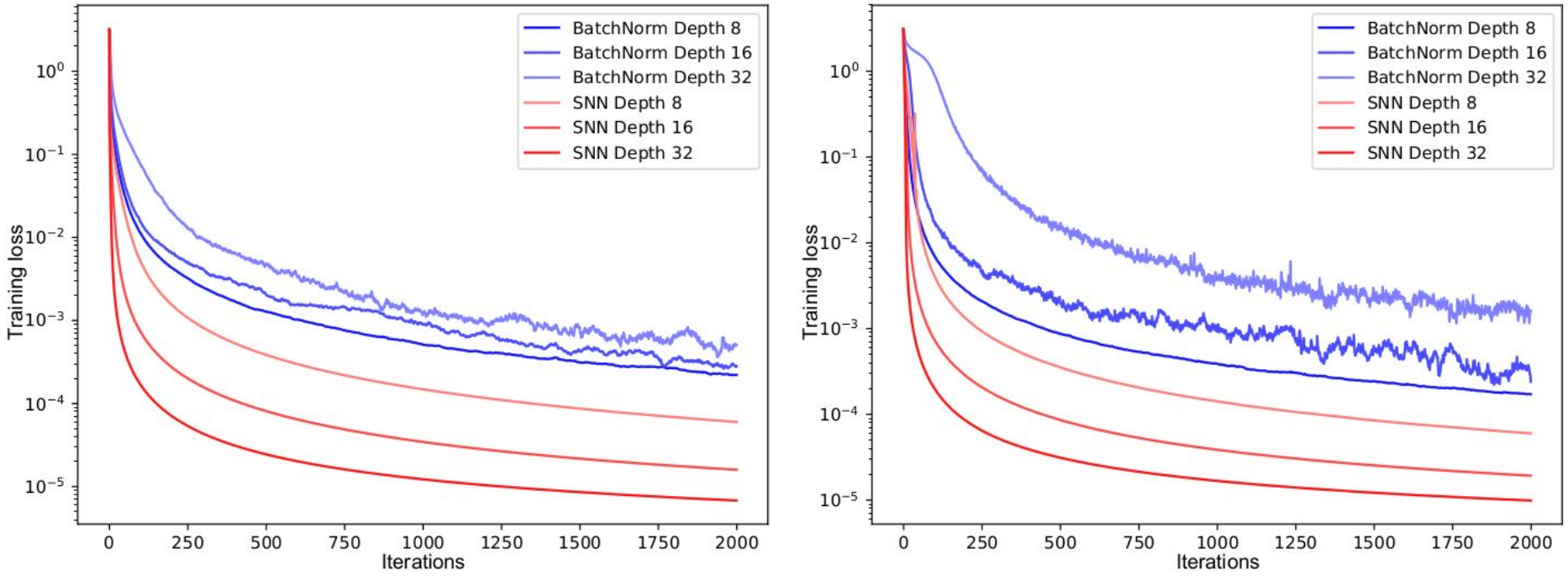


Figure 1: The left panel and the right panel show the training error (y-axis) for feed-forward neural networks (FNNs) with batch normalization (BatchNorm) and self-normalizing networks (SNN) across update steps (x-axis) on the MNIST dataset the CIFAR10 dataset, respectively. We tested networks with 8, 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.

Self Normalizing Neural Networks

How does it work?

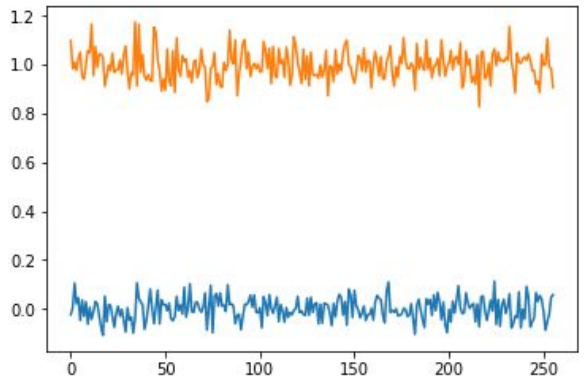
$$\text{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leqslant 0 \end{cases}$$

```python
def selu(x):
    """Scaled Exponential Linear Unit. (Klambauer et al., 2017)
    # Arguments
        x: A tensor or variable to compute the activation function for.
    # References
        - [Self-Normalizing Neural Networks](https://arxiv.org/abs/1706.02515)
    """
    alpha = 1.6732632423543772848170429916717
    scale = 1.0507009873554804934193349852946
    return scale * K.elu(x, alpha)
```

The idea behind self normalization:

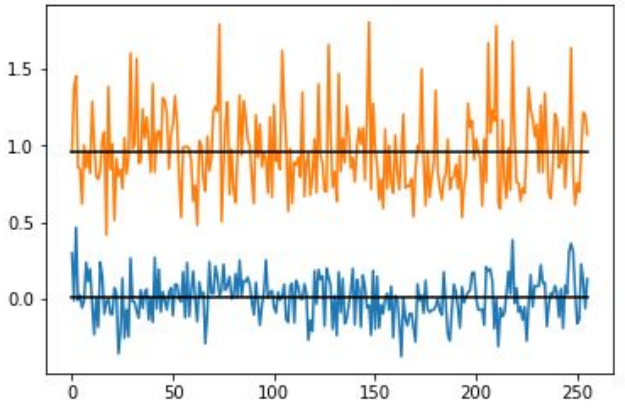
```python
n_hidden = 256
X = np.random.randn(512, n_hidden)

plt.plot(X.mean(0), label="mean")
plt.plot(X.std(0)**2, label="var")

```

```
[<matplotlib.lines.Line2D at 0x7f898ced0438>]
```



Propagate through the network 50 layers

```python
X = np.random.randn(512, n_hidden)

for i in range(50):
    W = np.random.randn(n_hidden, n_hidden)/np.sqrt(n_hidden)
    X_hat = selu(X @ W)
    X = X_hat
```



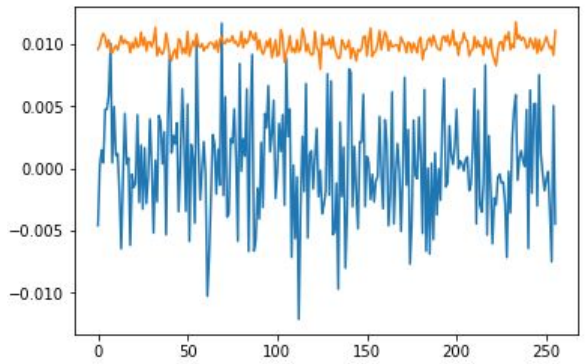
Self Normalizing Neural Networks

How does it work?

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

```python
def selu(x):
    """Scaled Exponential Linear Unit. (Klambauer et al., 2017)
    # Arguments
        x: A tensor or variable to compute the activation function for.
    # References
        - [Self-Normalizing Neural Networks](https://arxiv.org/abs/1706.02515)
    """
    alpha = 1.6732632423543772848170429916717
    scale = 1.0507009873554804934193349852946
    return scale * K.elu(x, alpha)
```

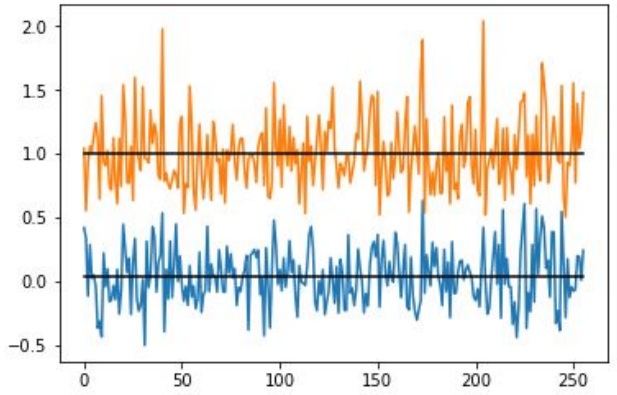
The idea behind self normalization:




```python
X = np.random.randn(512, n_hidden)*0.1
```

Propagate through the network 50 layers

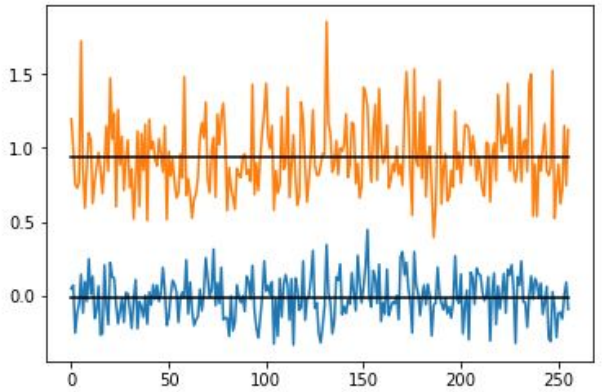
```python
X = np.random.randn(512, n_hidden)

for i in range(50):
    W = np.random.randn(n_hidden, n_hidden)/np.sqrt(n_hidden)
    X_hat = selu(X @ W)
    X = X_hat
```



The network self normalized outputs

Same but W is initialized from uniform distribution (still stable)

```python
for i in range(50):
    # uniform distribution
    W = np.random.rand(n_hidden, n_hidden)
    W = W - W.mean(0)
    N = np.sqrt((W**2).sum(0))
    W = W/N

    X_hat = selu(X @ W)
    X = X_hat
```



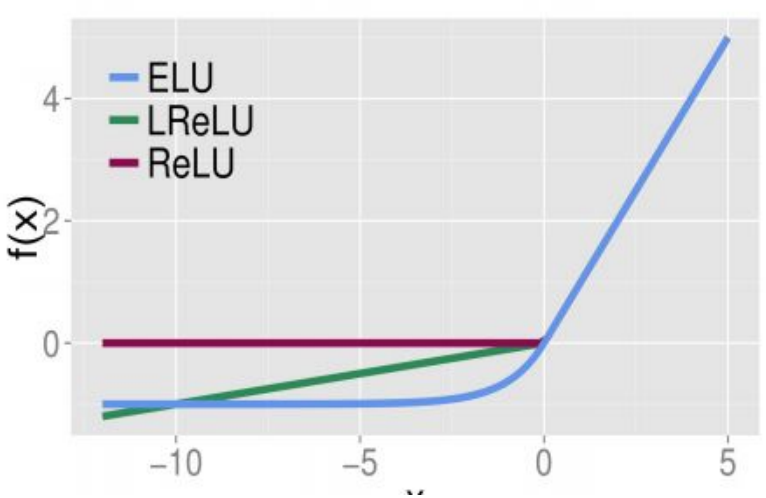
Show this demo

Self Normalizing Neural Networks

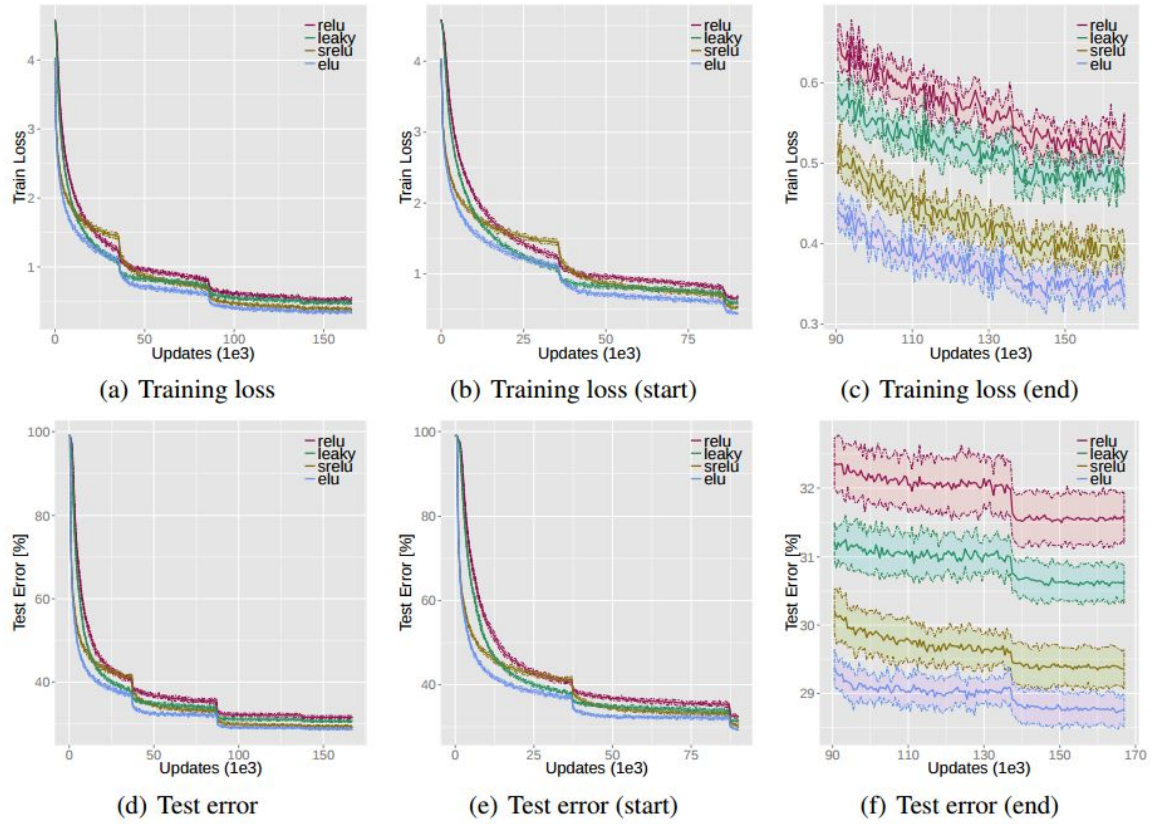
- Constructing SNNs: the authors propose following parametrization for g $\begin{pmatrix} \tilde{\mu} \\ \tilde{\nu} \end{pmatrix} = g \begin{pmatrix} \mu \\ \nu \end{pmatrix}$

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

scaled exponential linear unit



no prelu in their benchmark...



(a) Training loss (b) Training loss (start) (c) Training loss (end)

(d) Test error (e) Test error (start) (f) Test error (end)

Figure 4: Comparison of ReLUs, LReLUs, and SReLUs on CIFAR-100. Panels (a-c) show the training loss, panels (d-f) the test classification error. The ribbon band show the mean and standard deviation for 10 runs along the curve. ELU networks achieved lowest test error and training loss.

arxiv: 1511.07289

Self Normalizing Neural Networks

- Constructing SNNs: the authors propose following parametrization for g $\begin{pmatrix} \tilde{\mu} \\ \tilde{\nu} \end{pmatrix} = g \begin{pmatrix} \mu \\ \nu \end{pmatrix}$

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

Some clues for constructing SNNs (according to authors)

- Won't work with Relu, sigmoid
- must have negative and positive values for controlling mean
- saturation regions to dampen the gradients when activation is too large in lower layer
- a slope larger than 1 to increase the variance when to small in lower layer
- a continuous slope to provide fixed point

They impose initialization condition on the weight vectors:

$$\omega := \sum_{i=1}^{n} w_i \qquad \tau := \sum_{i=1}^{n} w_i^2$$

Derivation of mapping function

- Constructing SNNs: the authors propose following parametrization for g $\begin{pmatrix} \tilde{\mu} \\ \tilde{\nu} \end{pmatrix} = g \begin{pmatrix} \mu \\ \nu \end{pmatrix}$

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

- Assumptions:
  - activations x_i are independent (not full filled in general)
  - and share the same mean and variance

$$\mu := \mathrm{E}(x_i) \qquad \nu := \mathrm{Var}(x_i)$$

$$\omega := \sum_{i=1}^{n} w_i$$

$$\tau := \sum_{i=1}^{n} w_i^2$$

- The mean activation of $z = \boldsymbol{w}^T \boldsymbol{x}$

$$\begin{aligned} \mathrm{E}(z) &= \mathrm{E}\left(\mathbf{w}^T \mathbf{x}\right) = \sum_i \mathrm{E}\left(w_i x_i\right) \\ &= \sum_i w_i \mathrm{E}\left(x_i\right) = \sum_i w_i \mu = \mu \sum_i w_i = \mu\omega \end{aligned}$$

from the assumption

Derivation of mapping function

- Constructing SNNs: the authors propose following parametrization for g $\begin{pmatrix} \tilde{\mu} \\ \tilde{\nu} \end{pmatrix} = g \begin{pmatrix} \mu \\ \nu \end{pmatrix}$

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

- The mean of activations of z is $\mathrm{E}(z) = \mu\omega$

- The variance of activations: $\operatorname{Var}(z) = \mathrm{E}(z^2) - \mathrm{E}(z)^2$

$$\begin{aligned} \mathbf{Var}(X) &= \mathbf{E}\left[(X - \mathbf{E}[X])^2\right] \\ &= \mathbf{E}\left[X^2 - 2X\,\mathbf{E}[X] + \mathbf{E}[X]^2\right] \\ &= \mathbf{E}\left[X^2\right] - 2\,\mathbf{E}[X]\,\mathbf{E}[X] + \mathbf{E}[X]^2 \\ &= \mathbf{E}\left[X^2\right] - \mathbf{E}[X]^2 \end{aligned}$$

- Computing second term

$$\mathrm{E}(z)^2 = (\mu\omega)^2 = \mu^2 \left(\sum_i w_i\right)^2 = \mu^2 \left(\sum_i w_i^2 + \sum_{i,i'} w_i w_{i'}\right)$$

$$z = \boldsymbol{w}^T \boldsymbol{x}$$

$$\omega := \sum_{i=1}^{n} w_i$$

$$\tau := \sum_{i=1}^{n} w_i^2$$

$$\mu := \mathrm{E}(x_i)$$

$$\nu := \operatorname{Var}(x_i)$$

Derivation of mapping function

- Constructing SNNs: the authors propose following parametrization for g $\begin{pmatrix} \tilde{\mu} \\ \tilde{\nu} \end{pmatrix} = g \begin{pmatrix} \mu \\ \nu \end{pmatrix}$

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

- The variance of activations: $\operatorname{Var}(z) = \operatorname{E}(z^2) - \operatorname{E}(z)^2$
- Second term

$$\operatorname{E}(z)^2 = (\mu\omega)^2 = \mu^2 \left(\sum_i w_i \right)^2 = \mu^2 \left(\sum_i w_i^2 + \sum_{i,i'} w_i w_{i'} \right)$$

- First term:

$$\operatorname{E}(z^2) = \operatorname{E}\left(\left(\sum_i w_i x_i \right)^2 \right) = \operatorname{E}\left(\left(\sum_i w_i x_i \right) \left(\sum_{i'} w_{i'} x_{i'} \right) \right)$$

$$= \operatorname{E}\left(\sum_i \sum_{i'} w_i x_i w_{i'} x_{i'} \right) = \operatorname{E}\left(\sum_i w_i^2 x_i^2 \right) + \operatorname{E}\left(\sum_{i \neq i'} w_i x_i w_{i'} x_{i'} \right)$$

$$z = \boldsymbol{w}^T \boldsymbol{x}$$

$$\omega := \sum_{i=1}^n w_i$$

$$\tau := \sum_{i=1}^n w_i^2$$

$$\mu := \operatorname{E}(x_i)$$

$$\nu := \operatorname{Var}(x_i)$$

$$\sum_{i \neq i'} \operatorname{E}(w_i x_i w_{i'} x_{i'}) = \sum_{i \neq i'} w_i w_{i'} \operatorname{E}(x_i x_{i'}) = \sum_{i \neq i'} w_i w_{i'} \operatorname{E}(x_i) \operatorname{E}(x_{i'}) = \mu^2 \sum_{i \neq i'} w_i w_{i'}$$

Derivation of mapping function

- Constructing SNNs: the authors propose following parametrization for g $\begin{pmatrix} \tilde{\mu} \\ \tilde{\nu} \end{pmatrix} = g \begin{pmatrix} \mu \\ \nu \end{pmatrix}$

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

- The variance of activations: $\text{Var}(z) = \text{E}(z^2) - \text{E}(z)^2$

- The final expression is:

$$\text{Var}(z) = \sum_i w_i^2 \text{E}(x_i^2) + \cancel{\mu^2 \sum_{i \neq i'} w_i w_{i'}} - \mu^2 \sum_i w_i^2 - \cancel{\mu^2 \sum_{i \neq i'} w_i w_{i'}}$$

$$\text{Var}(z) = \sum_i w_i^2 \left[\text{E}(x_i^2) - \mu^2\right] = \sum_i w_i^2 \nu = \nu \sum_i w_i^2 = \nu\tau$$

Mean: $\text{E}(z) = \mu\omega$ Variance: $\text{Var}(z) = \nu\tau$

$$z = \boldsymbol{w}^T \boldsymbol{x}$$

$$\omega := \sum_{i=1}^n w_i$$

$$\tau := \sum_{i=1}^n w_i^2$$

$$\mu := \text{E}(x_i)$$

$$\nu := \text{Var}(x_i)$$

Derivation of mapping function

- Constructing SNNs: the authors propose following parametrization for g

$$\begin{pmatrix} \tilde{\mu} \\ \tilde{\nu} \end{pmatrix} = g \begin{pmatrix} \mu \\ \nu \end{pmatrix}$$

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

- The mean and variance:

$$\mathrm{E}(z) = \mu\omega \qquad \operatorname{Var}(z) = \nu\tau$$

- According to CTL the z variable approaches to normal distribution (z is a weighted sum of independent variables)

$$z \sim \mathcal{N}(\mu\omega, \sqrt{\nu\tau}) \text{ with density } p_{\mathrm{N}}(z; \mu\omega, \sqrt{\nu\tau})$$

larger z then more correct is this statement

$$z = \boldsymbol{w}^T \boldsymbol{x}$$

$$\omega := \sum_{i=1}^{n} w_i$$

$$\tau := \sum_{i=1}^{n} w_i^2$$

$$\mu := \mathrm{E}(x_i)$$

$$\nu := \operatorname{Var}(x_i)$$

Derivation of mapping function

- Constructing SNNs: the authors propose following parametrization for g

$$\begin{pmatrix} \tilde{\mu} \\ \tilde{\nu} \end{pmatrix} = g \begin{pmatrix} \mu \\ \nu \end{pmatrix}$$

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

- z is a roughly random variable from normal distribution

$$z \sim \mathcal{N}(\mu\omega, \sqrt{\nu\tau}) \text{ with density } p_{\mathrm{N}}(z; \mu\omega, \sqrt{\nu\tau})$$

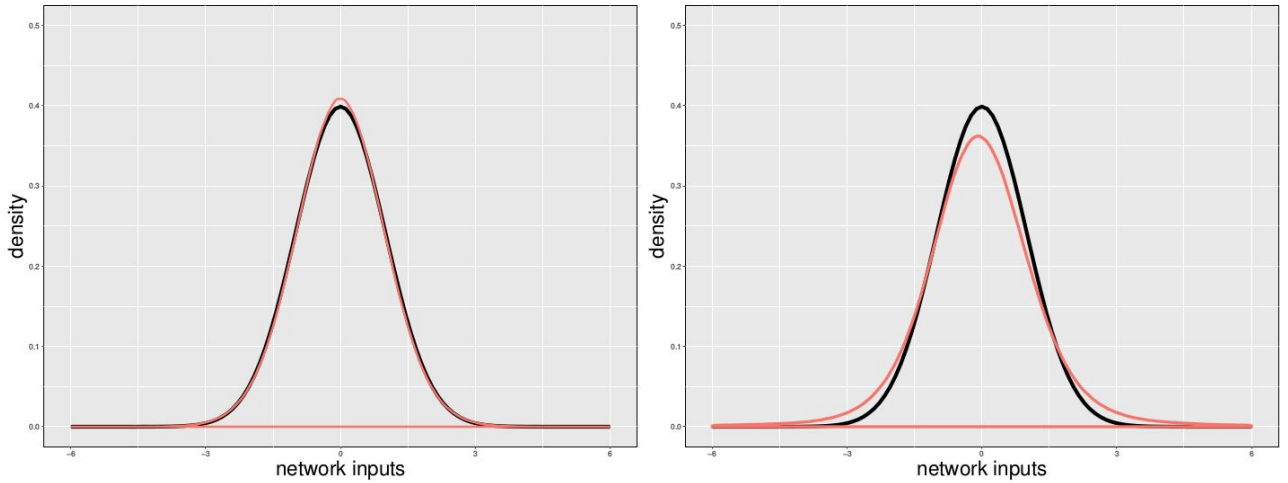


Figure A8: Distribution of network inputs of an SNN for the Tox21 data set. The plots show the distribution of network inputs z of the second layer of a typical Tox21 network. The red curves display a kernel density estimator of the network inputs and the black curve is the density of a standard normal distribution. **Left panel:** At initialization time before learning. The distribution of network inputs is close to a standard normal distribution. **Right panel:** After 40 epochs of learning. The distributions of network inputs is close to a normal distribution.

$$z = \boldsymbol{w}^T \boldsymbol{x}$$

$$\omega := \sum_{i=1}^{n} w_i$$

$$\tau := \sum_{i=1}^{n} w_i^2$$

$$\mu := \mathrm{E}(x_i)$$

$$\nu := \mathrm{Var}(x_i)$$

Derivation of mapping function

- Constructing SNNs: the authors propose following parametrization for g

$$\begin{pmatrix} \tilde{\mu} \\ \tilde{\nu} \end{pmatrix} = g \begin{pmatrix} \mu \\ \nu \end{pmatrix}$$

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

- z is a roughly random variable from normal distribution

$$z \sim \mathcal{N}(\mu\omega, \sqrt{\nu\tau}) \text{ with density } p_{\mathrm{N}}(z; \mu\omega, \sqrt{\nu\tau})$$

- The mean and variance after applying selu is then:

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

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

- Can be integrated analytically

$$\operatorname{Var}(z) = \mathrm{E}\left(z^2\right) - \mathrm{E}(z)^2$$

$$z = \boldsymbol{w}^T \boldsymbol{x}$$

$$\omega := \sum_{i=1}^{n} w_i$$

$$\tau := \sum_{i=1}^{n} w_i^2$$

$$\mu := \mathrm{E}(x_i)$$

$$\nu := \operatorname{Var}(x_i)$$

Derivation of mapping function

```python
def selu(x):
    """Scaled Exponential Linear Unit. (Klambauer et a
    # Arguments
        x: A tensor or variable to compute the activat
    # References
        - [Self-Normalizing Neural Networks](https://a
    """
    alpha = 1.6732632423543772848170429916717
    scale = 1.0507009873554804934193349852946
    return scale * K.elu(x, alpha)
```

- Constructing SNNs: the authors propose following parametrization for g

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

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

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

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

$$z = \boldsymbol{w}^T \boldsymbol{x}$$

$$\omega := \sum_{i=1}^{n} w_i$$

$$\tau := \sum_{i=1}^{n} w_i^2$$

$$\mu := \mathrm{E}(x_i)$$

$$\nu := \mathrm{Var}(x_i)$$

- assume normalized weight vector $\omega = 0$ and $\tau = 1$
- given a fixed point (μ, v) = (0, 1) one may solve this equation for selu parameters:

$$\alpha_{01} \approx 1.6733 \text{ and } \lambda_{01} \approx 1.0507$$

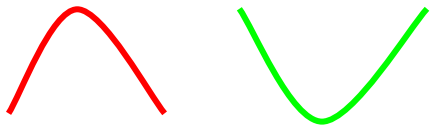
Derivation of mapping function

```
def selu(x):
    """Scaled Exponential Linear Unit. (Klambauer et a
    # Arguments
        x: A tensor or variable to compute the activat
    # References
        - [Self-Normalizing Neural Networks](https://a
    """
    alpha = 1.6732632423543772848170429916717
    scale = 1.0507009873554804934193349852946
    return scale * K.elu(x, alpha)
```

- Constructing SNNs: the authors propose following parametrization for g

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

- Is fixed point (μ, v) = (0, 1) stable and attractive?



From stability theory:

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

stable is if: *the spectral norm of transformation Jacobian is less than one.*

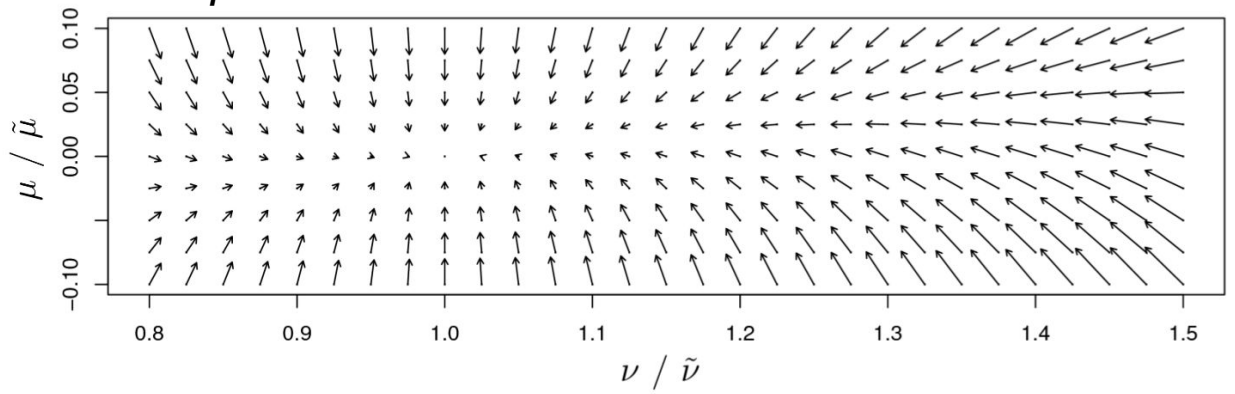


Figure 2: For $\omega = 0$ and $\tau = 1$, the mapping g of mean μ (x-axis) and variance ν (y-axis) to the next layer's mean $\tilde{\mu}$ and variance $\tilde{\nu}$ is depicted. Arrows show in which direction (μ, ν) is mapped by $g : (\mu, \nu) \mapsto (\tilde{\mu}, \tilde{\nu})$. The fixed point of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

$$z = \boldsymbol{w}^T \boldsymbol{x}$$

$$\omega := \sum_{i=1}^{n} w_i$$

$$\tau := \sum_{i=1}^{n} w_i^2$$

$$\mu := \mathrm{E}(x_i)$$

$$\nu := \mathrm{Var}(x_i)$$

Derivation of mapping function

A 1D example for stability theory

- Consider general mapping x -> y

$$y = f(x;\alpha)$$

- In previous example we have fixed (x) and (y) and computed alpha which gives

$$x_\alpha = f(x_\alpha;\alpha)$$

- Stability is when following condition is satisfied

$$|f(x_\alpha+\delta;\alpha)-x_\alpha| < |(x_\alpha+\delta)-x_\alpha|$$

$$\left| \cancel{f(x_\alpha;\alpha)} + \delta \left.\frac{df(x;\alpha)}{dx}\right|_{x=x_\alpha} - \cancel{x_\alpha} \right| < |(x_\alpha+\delta)-x_\alpha|$$

$$|\delta| \left| \left.\frac{df(x;\alpha)}{dx}\right|_{x=x_\alpha} \right| < |\delta| \rightarrow \left| \left.\frac{df(x;\alpha)}{dx}\right|_{x=x_\alpha} \right| < 1$$

Derivation of mapping function

```
def selu(x):
    """Scaled Exponential Linear Unit. (Klambauer et a
    # Arguments
        x: A tensor or variable to compute the activat
    # References
        - [Self-Normalizing Neural Networks](https://a
    """
    alpha = 1.6732632423543772848170429916717
    scale = 1.0507009873554804934193349852946
    return scale * K.elu(x, alpha)
```

- Constructing SNNs: the authors propose following parametrization for g

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

What if weights are not normalized:

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.*

computer assisted proof

$$z = \boldsymbol{w}^T \boldsymbol{x}$$

$$\omega := \sum_{i=1}^{n} w_i$$

$$\tau := \sum_{i=1}^{n} w_i^2$$

$$\mu := \mathrm{E}(x_i)$$

$$\nu := \mathrm{Var}(x_i)$$

New dropout technique

- Standard dropout fits well with Relus since 0 is a default (minimum) value for it. Dropout preserves the mean of activations
- Alpha dropout sets minimum SELU value instead of zero

$$\lim_{x \to -\infty} \mathrm{selu}(x) = -\lambda\alpha = \alpha'$$

- Basic idea is: apply alpha dropout to activations, then apply affine transform of form a*x + b which preserves the mean E(x) and variance Var(x) of activations

Some experiments (LB from 2014 is ~82)

Drug discovery: The Tox21 challenge dataset. The Tox21 challenge dataset comprises about 12,000 chemical compounds whose twelve toxic effects have to be predicted based on their chemical structure. We used the validation sets of the challenge winners for hyperparameter selection (see Section A4) and the challenge test set for performance comparison. We repeated the whole evaluation procedure 5 times to obtain error bars. The results in terms of average AUC are given in Table 2.

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

Some experiments

Astronomy: Prediction of pulsars in the HTRU2 dataset. Since a decade, machine learning methods have been used to identify pulsars in radio wave signals [27]. Recently, the High Time Resolution Universe Survey (HTRU2) dataset has been released with 1,639 real pulsars and 16,259 spurious signals. Currently, the highest AUC value of a 10-fold cross-validation is 0.976 which has

Table 3: Comparison of FNNs and reference methods at HTRU2 in terms of AUC. The first, fourth and seventh column give the method, the second, fifth and eight column the AUC averaged over 10 cross-validation folds, and the third and sixth column the p-value of a paired Wilcoxon test of the AUCs against the best performing method across the 10 folds. FNNs achieve better results than Naive Bayes (NB), C4.5, and SVM. SNNs exhibit the best performance and set a new record.

FNN methods			FNN methods			ref. methods	
method	AUC	p-value	method	AUC	p-value	method	AUC
SNN	0.9803 $\pm$ 0.010						
MSRAinit	0.9791 $\pm$ 0.010	3.5e-01	LayerNorm	0.9762* $\pm$ 0.011	1.4e-02	NB	0.976
WeightNorm	0.9786* $\pm$ 0.010	2.4e-02	BatchNorm	0.9760 $\pm$ 0.013	6.5e-02	C4.5	0.946
Highway	0.9766* $\pm$ 0.009	9.8e-03	ResNet	0.9753* $\pm$ 0.010	6.8e-03	SVM	0.929

Conclusions from paper

Conclusion

We have introduced self-normalizing neural networks for which we have proved that neuron activations are pushed towards zero mean and unit variance when propagated through the network. Additionally, for activations not close to unit variance, we have proved an upper and lower bound on the variance mapping. Consequently, SNNs do not face vanishing and exploding gradient problems. Therefore, SNNs work well for architectures with many layers, allowed us to introduce a novel regularization scheme, and learn very robustly. On 121 UCI benchmark datasets, SNNs have outperformed other FNNs with and without normalization techniques, such as batch, layer, and weight normalization, or specialized architectures, such as Highway or Residual networks. SNNs also yielded the best results on drug discovery and astronomy tasks. The best performing SNN architectures are typically very deep in contrast to other FNNs.

Deep learning with SELUs

Conclusions from paper

- Usage of CLT as a general tool for understanding NN
- Activations may drive outputs of neural network to certain distributions
- What if we can create activation function, which will be able to keep weights of neural network at specific distribution ?

LSUV initialization

Published as a conference paper at ICLR 2016

ALL YOU NEED IS A GOOD INIT

Dmytro Mishkin, Jiri Matas

Center for Machine Perception
Czech Technical University in Prague
Czech Republic {mishkdmy,matas}@cmp.felk.cvut.cz

The authors propose a **Layer-Sequential-Unit Variance** initialization.

The algorithm is very simple:

1. Initialize the matrices to be orthonormal
2. Go layer-by-layer and scale their weights such that they will produce unit variance activations

The algorithm:

Algorithm 1 Layer-sequential unit-variance orthogonal initialization. L – convolution or full-connected layer, W_L - its weights, B_L - its output blob., Tol_{var} - variance tolerance, T_i – current trial, T_{max} – max number of trials.

Pre-initialize network with orthonormal matrices as in Saxe et al. (2014)
for each layer L **do**
  while $|Var(B_L) - 1.0| \geq Tol_{var}$ and $(T_i < T_{max})$ **do**
    do Forward pass with a mini-batch
    calculate $Var(B_L)$
    $W_L = W_L \,/\, \sqrt{Var(B_L)}$
  end while
end for

LSUV initialization - some benchmarks from paper

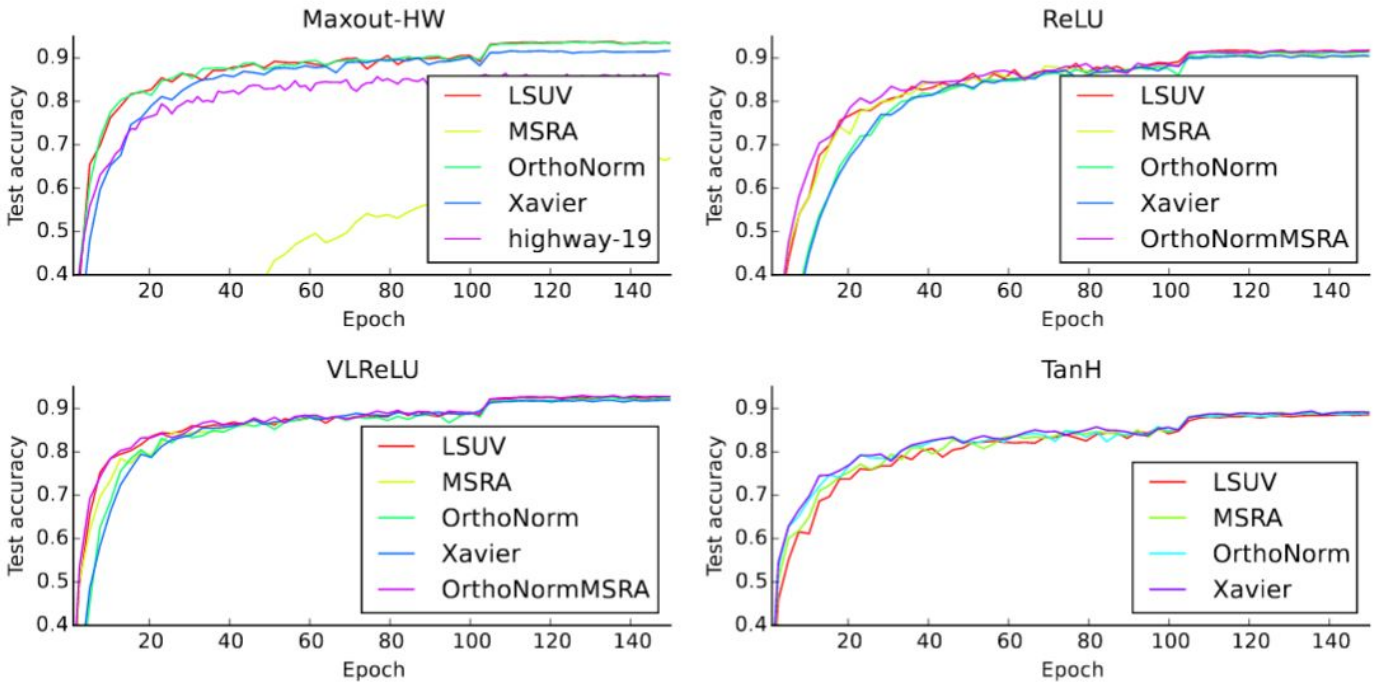


Figure 2: CIFAR-10 accuracy of FitNet-4 with different activation functions. Note that graphs are cropped at 0.4 accuracy. Highway19 is the network from Srivastava et al. (2015).

- In most cases it seems that it provides better start

LSUV initialization - some benchmarks from paper

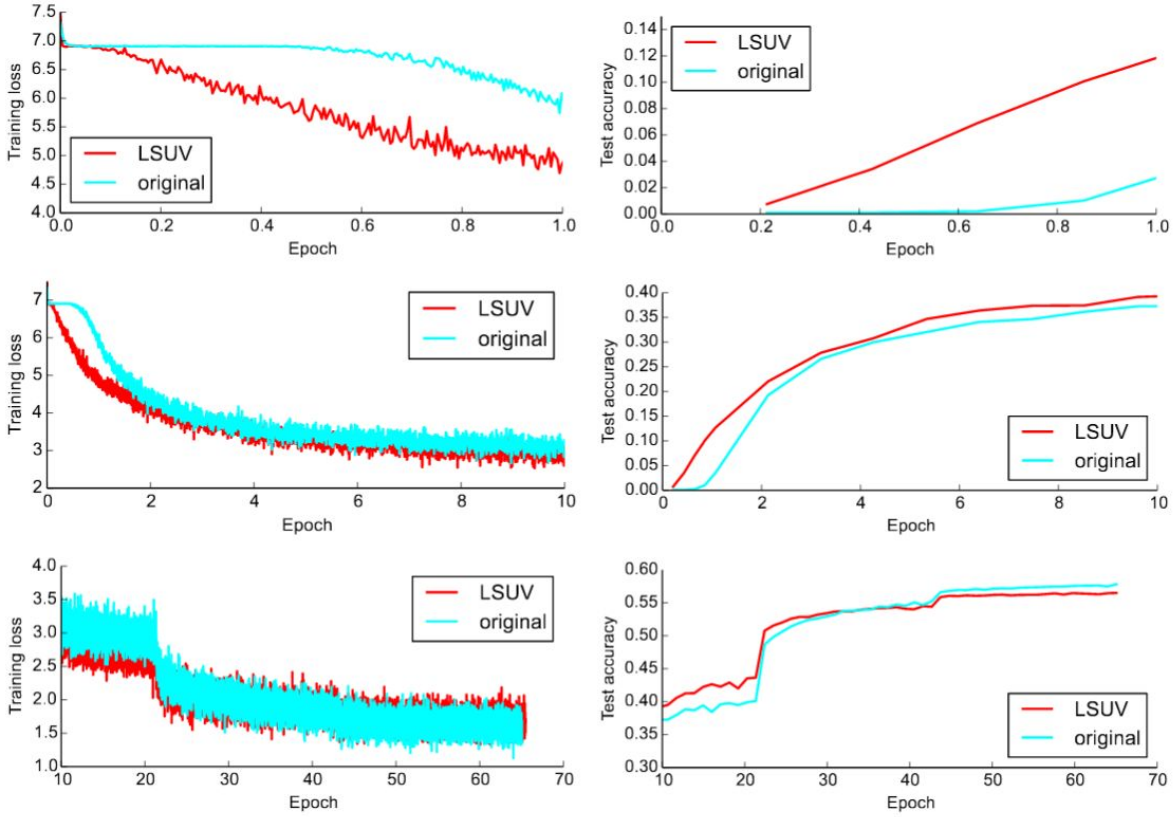


Figure 4: CaffeNet training on ILSVRC-2012 dataset with LSUV and original Krizhevsky et al. (2012) initialization. Training loss (left) and validation accuracy (right). Top – first epoch, middle – first 10 epochs, bottom – full training.

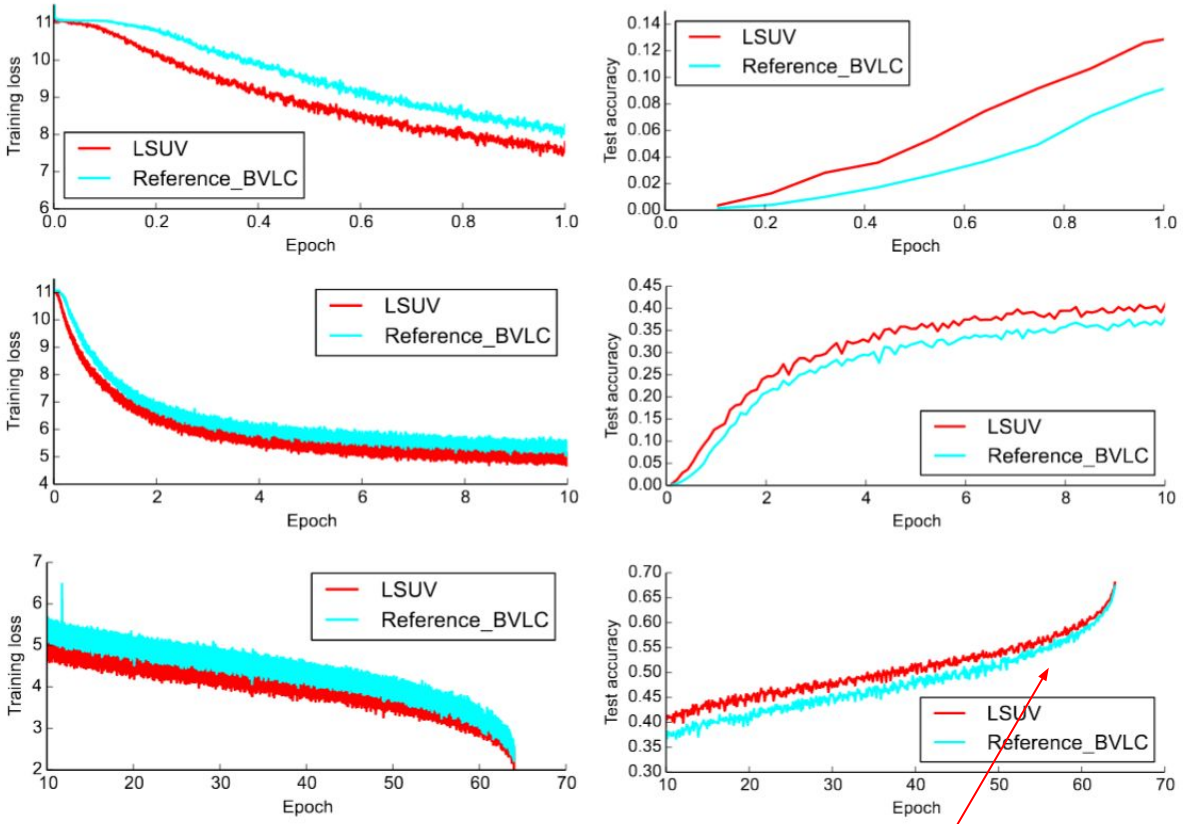


Figure 5: GoogLeNet training on ILSVRC-2012 dataset with LSUV and reference Jia et al. (2014) BVLC initializations. Training loss (left) and validation accuracy (right). Top – first epoch, middle – first ten epochs, bottom – full training

- In most cases it seems that it provides better start

strange behaviour in accuracy???

LSUV initialization - continuation ….

All You Need is Beyond a Good Init: Exploring Better Solution for Training Extremely Deep Convolutional Neural Networks with Orthonormality and Modulation

Di Xie
xiedi@hikvision.com

Jiang Xiong
xiongjiang@hikvision.com

Shiliang Pu
pushiliang@hikvision.com

Hikvision Research Institute
Hangzhou, China

nel number, respectively. We replace original weight decay regularizer with the orthonormal regularizer:

$$\frac{\lambda}{2}\sum_{i=1}^{D}\|\mathbf{W}_l^T\mathbf{W}_l-\mathbf{I}\|_F^2 \qquad (3)$$

https://arxiv.org/pdf/1703.01827.pdf (6 Mar 2017)

NEURAL PHOTO EDITING WITH INTROSPECTIVE ADVERSARIAL NETWORKS

Andrew Brock, Theodore Lim, & J.M. Ritchie
School of Engineering and Physical Sciences
Heriot-Watt University
Edinburgh, UK
{ajb5, t.lim, j.m.ritchie}@hw.ac.uk

Nick Weston
Renishaw plc
Research Ave, North
Edinburgh, UK
Nick.Weston@renishaw.com

3.5 ORTHOGONAL REGULARIZATION

Orthogonality is a desirable quality in ConvNet filters, partially because multiplication by an orthogonal matrix leaves the norm of the original matrix unchanged. This property is valuable in deep or recurrent networks, where repeated matrix multiplication can result in signals vanishing or exploding. We note the success of initializing weights with orthogonal matrices (Saxe et al., 2014), and posit that maintaining orthogonality throughout training is also desirable. To this end, we propose a simple weight regularization technique, Orthogonal Regularization, that encourages weights to be orthogonal by pushing them towards the nearest orthogonal manifold. We augment our objective with the cost:

$$\mathcal{L}_{ortho} = \Sigma(|WW^T - I|) \qquad (6)$$

Where Σ indicates a sum across all filter banks, W is a filter bank, and I is the identity matrix.

https://arxiv.org/abs/1609.07093 (22 Sep 2016)

On orthogonality and learning recurrent networks with long term dependencies

Eugene Vorontsov [1 2] **Chiheb Trabelsi** [1 2] **Samuel Kadoury** [1 3] **Chris Pal** [1 2]

https://arxiv.org/pdf/1702.00071.pdf (31 Jan 2017)

2. Our Approach

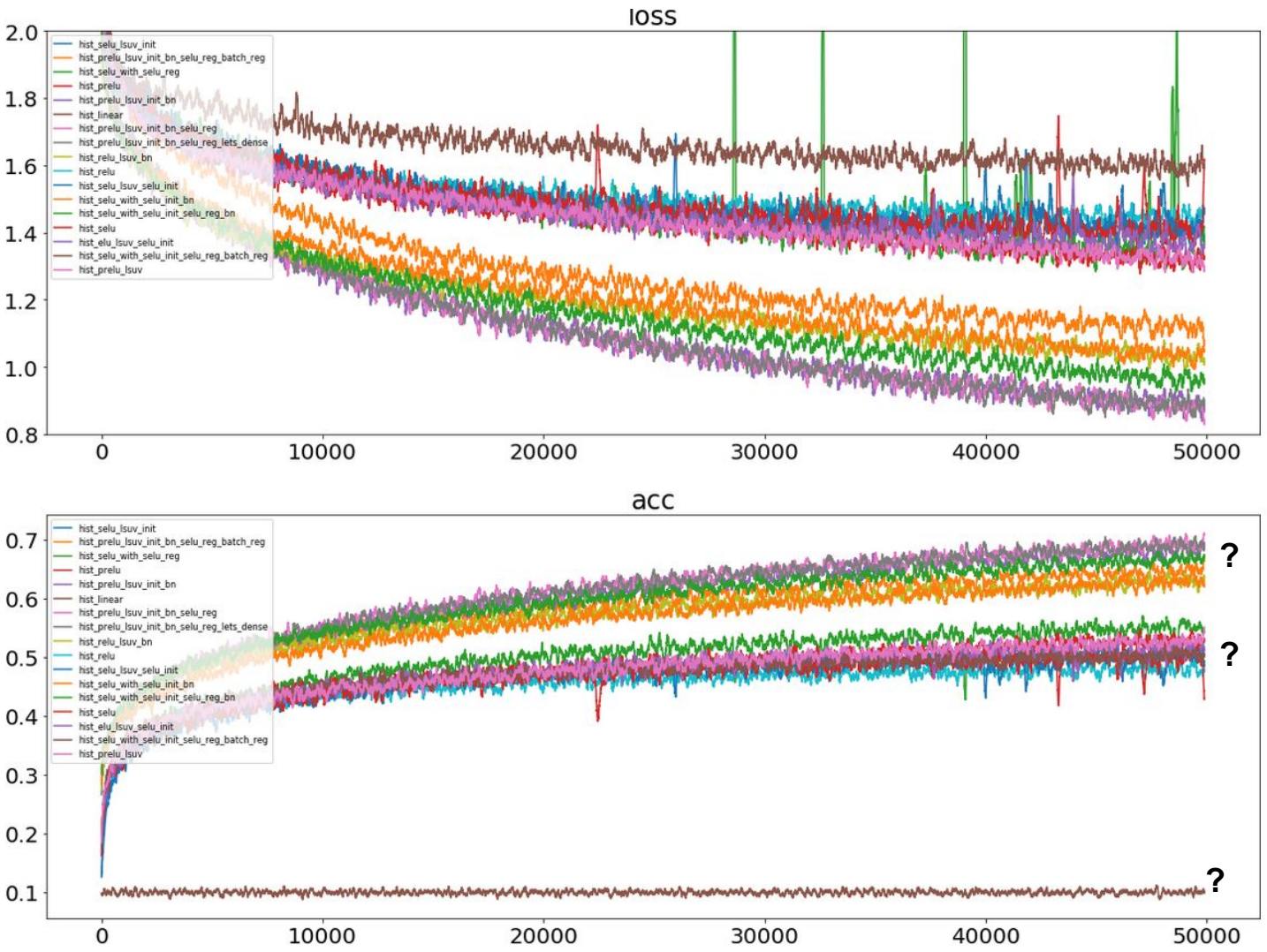
Vanishing and exploding gradients can be controlled to a large extent by controlling the maximum and minimum *gain* of $\mathbf{W}$. The maximum gain of a matrix $\mathbf{W}$ is given by the spectral norm which is given by

$$||\mathbf{W}||_2 = \max\left[\frac{||\mathbf{W}\mathbf{x}||}{||\mathbf{x}||}\right]. \qquad (6)$$

By keeping our weight matrix $\mathbf{W}$ close to orthogonal, one can ensure that it is close to a norm-preserving transformation (where the spectral norm is equal to one, but the minimum gain is also one). One way to achieve this is via a simple soft constraint or regularization term of the form:

$$\lambda\sum_i ||\mathbf{W}_i^T\mathbf{W}_i - \mathbf{I}||^2. \qquad (7)$$

My experiments - FCN 4 layers ~ 1M params

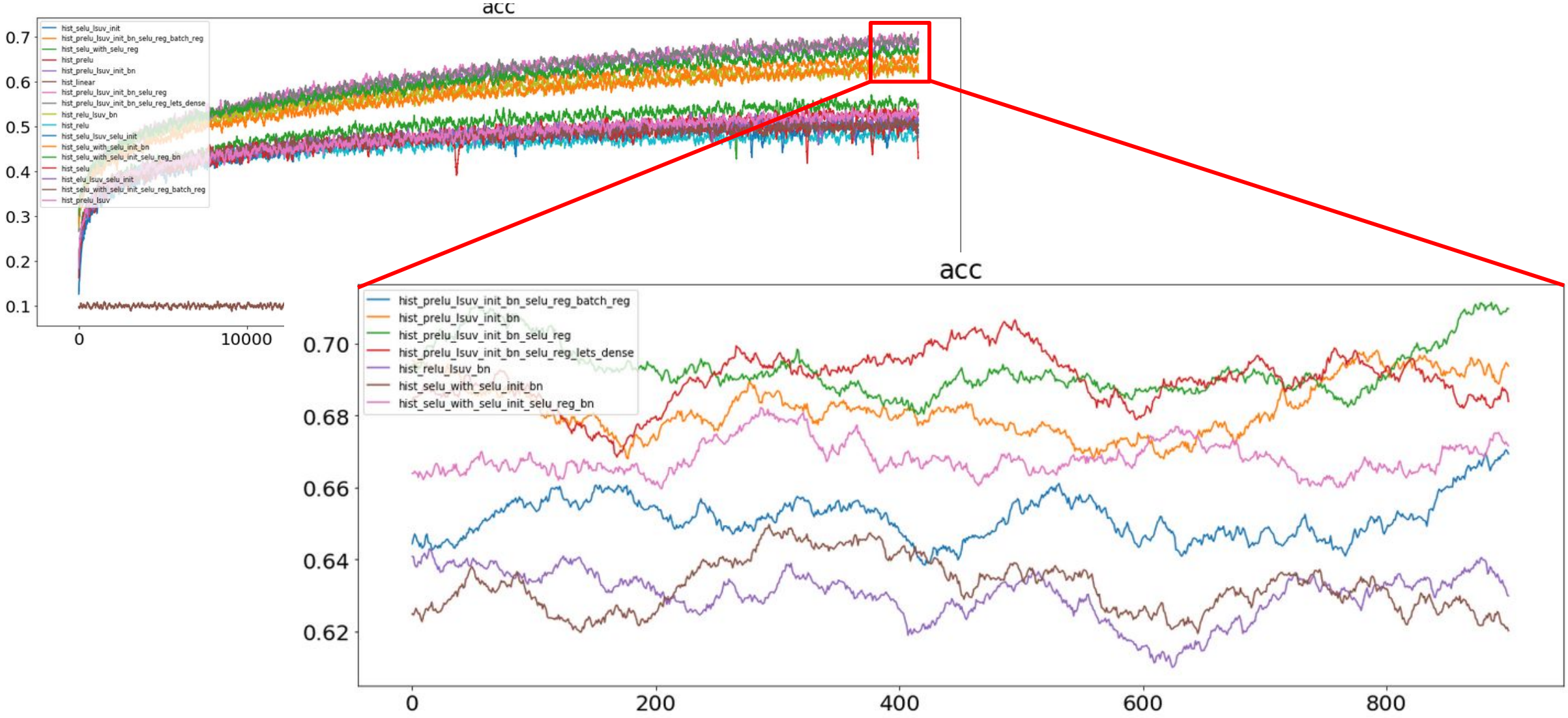


Architecture:

```
model = Sequential()
model.add(Flatten(input_shape=x_train.shape[1:]))

for l in range(1, num_layers+1):
    model.add(dense_layer(n_hidden, activity_regularizer=get_batch_reg(),
                          name=f"l{l}",
                          kernel_initializer=get_init(),
                          kernel_regularizer=get_reg()))
    if advanced_activation == None:
        model.add(Activation(activation, name=f"a{l}"))
    else:
        model.add(advanced_activation(name=f"a{l}"))
    if batch_norm:
        model.add(BatchNormalization())

model.add(Dense(num_classes, kernel_initializer=get_init()))
model.add(Activation('softmax'))

opt = keras.optimizers.Adam(lr=0.002)

model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['categorical_accuracy'])
```

- 4 FC layers, with hidden vectors 256
- Activations: SELU, RELU, PRELU, ELU, LINEAR
- Normalization: Batch, SELU
- Activity regularization: Batch
- Initialization: LSUV, GLOROT, SELU

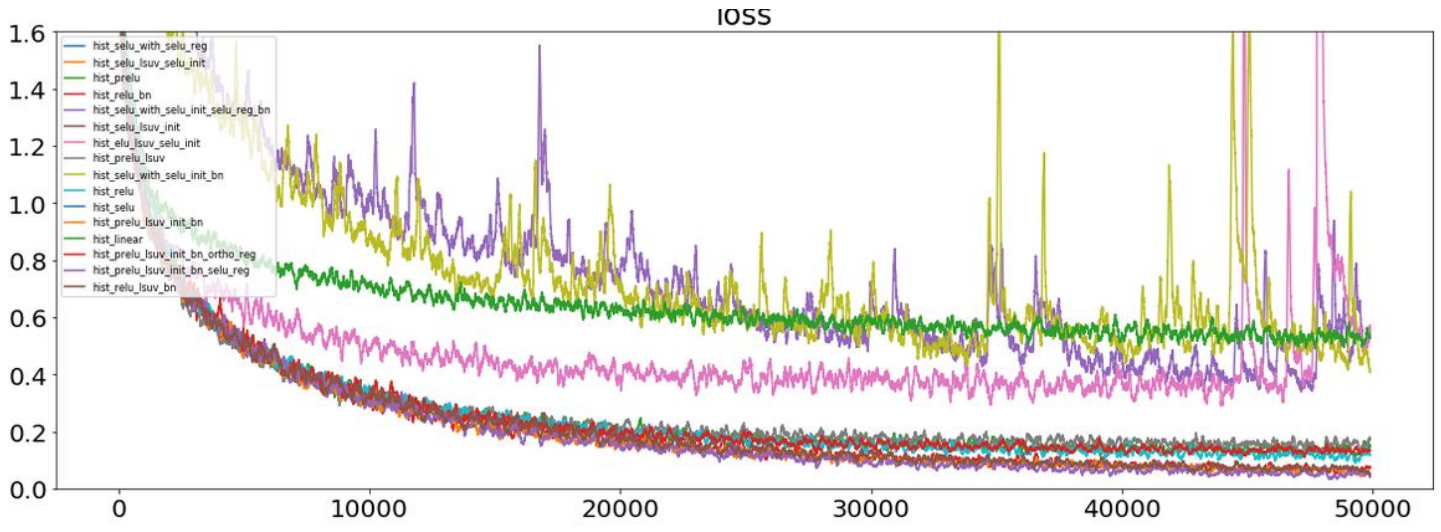
My experiments

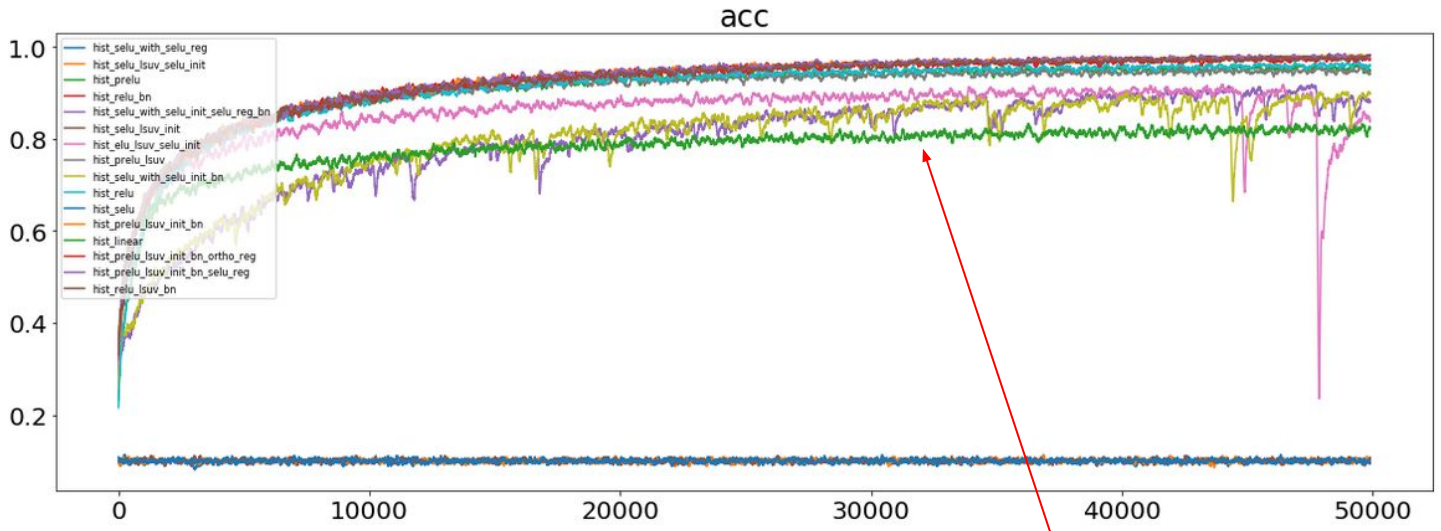


My experiments - ranking

test_acc	train_acc	test_loss	train_loss	prelu	selu	lsuv_init	bn	linear	lsuv_selu_init	batch_reg	selu_init	lets_dense	relu	selu_reg	elu
0.604	0.713	1.224	0.821	X	-	X	X	-	-	-	-	-	-	X	-
0.601	0.684	1.212	0.883	X	-	X	X	-	-	-	-	X	-	X	-
0.597	0.674	1.223	0.957	-	X	-	X	-	-	-	X	-	-	X	-
0.580	0.618	1.204	1.042	-	X	-	X	-	-	-	X	-	-	-	-
0.569	0.696	1.303	0.862	X	-	X	X	-	-	-	-	-	-	-	-
0.568	0.626	1.234	1.047	-	-	X	X	-	-	-	-	-	X	-	-
0.566	0.673	1.434	1.068	X	-	X	X	-	-	X	-	-	-	X	-
0.565	0.548	1.316	1.368	-	X	-	-	-	-	-	X	-	-	X	-
0.528	0.498	1.358	1.403	-	X	X	-	-	-	-	-	-	-	-	-
0.524	0.555	1.361	1.289	X	-	X	-	-	-	-	-	-	-	-	-
0.521	0.523	1.368	1.333	X	-	-	-	-	-	-	-	-	-	-	-
0.512	0.509	1.381	1.362	-	-	-	-	-	X	-	-	-	-	-	X
0.508	0.494	1.596	1.621	-	X	-	-	-	-	X	X	-	-	X	-
0.492	0.490	1.438	1.442	-	X	-	-	-	X	-	-	-	-	-	-
0.481	0.478	1.464	1.446	-	-	-	-	-	-	-	-	-	X	-	-
0.455	0.434	1.515	1.588	-	X	-	-	-	-	-	-	-	-	-	-
0.100	0.092	14.506	14.637	-	-	-	-	X	-	-	-	-	-	-	-

My experiments - Vgg like CNN



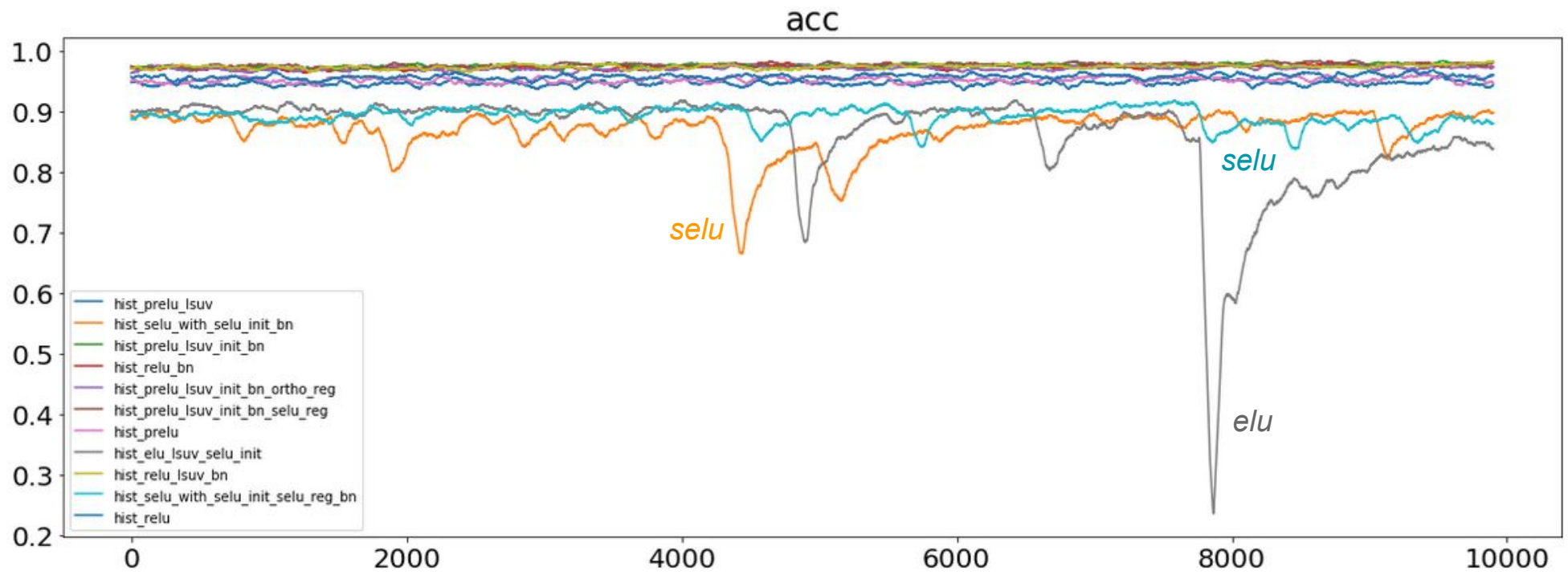


? :D

Architecture:

```
for l in range(2, num_layers+1):

    model.add(Conv2D(2*l*n_hidden, (3, 3),
                     padding='same',
                     kernel_initializer=get_init(),
                     kernel_regularizer=get_reg(2*l*n_hidden)))

    if advanced_activation == None:
        model.add(Activation(activation))
    else:
        model.add(advanced_activation(name=f"aux_a{l}"))

    model.add(Conv2D(2*l*n_hidden, (3, 3),
                     padding='same',
                     name=f"l{l}",
                     kernel_initializer=get_init(),
                     kernel_regularizer=get_reg(2*l*n_hidden)))

    if advanced_activation == None:
        model.add(Activation(activation, name=f"a{l}"))
    else:
        model.add(advanced_activation(name=f"a{l}"))
    if batch_norm:
        model.add(BatchNormalization())


    model.add(MaxPooling2D())

model.add(Flatten())
model.add(Dense(num_classes, kernel_initializer=get_init()))
```

My experiments - Vgg like CNN about 2M params



My experiments - Vgg like CNN about 2M params

	test_acc	train_acc	test_loss	train_loss	linear	bn	lsuv_init	selu	elu	lets_conv	ortho_reg	prelu	selu_reg	lsuv_selu_init	selu_init	relu
12	0.909	0.980	0.406	0.046	-	X	X	-	-	X	-	X	X	-	-	-
15	0.908	0.980	0.398	0.041	-	X	X	-	-	-	-	X	X	-	-	-
14	0.904	0.975	0.463	0.128	-	X	X	-	-	-	X	X	-	-	-	-
16	0.902	0.980	0.433	0.056	-	X	X	-	-	-	-	-	-	-	-	X
3	0.898	0.975	0.466	0.076	-	X	-	-	-	-	-	-	-	-	-	X
11	0.898	0.985	0.460	0.047	-	X	X	-	-	-	-	X	-	-	-	-
2	0.880	0.954	0.523	0.143	-	-	-	-	-	-	-	X	-	-	-	-
9	0.879	0.963	0.492	0.119	-	-	-	-	-	-	-	-	-	-	-	X
7	0.878	0.950	0.512	0.157	-	-	X	-	-	-	-	X	-	-	-	-
8	0.863	0.900	0.466	0.401	-	X	-	X	-	-	-	-	-	-	X	-
6	0.840	0.837	0.672	0.558	-	-	-	-	X	-	-	-	-	X	-	-
4	0.813	0.878	0.751	0.568	-	X	-	X	-	-	-	-	X	-	X	-
13	0.811	0.814	0.599	0.555	X	-	-	-	-	-	-	-	-	-	-	-
10	0.100	0.097	14.506	14.557	-	-	-	X	-	-	-	-	-	-	-	-
1	0.100	0.091	14.506	14.647	-	-	-	X	-	-	-	-	-	X	-	-
5	0.100	0.110	14.506	14.350	-	-	X	X	-	-	-	-	-	-	-	-
0	0.100	0.102	14.443	14.412	-	-	-	X	-	-	-	-	X	-	X	-

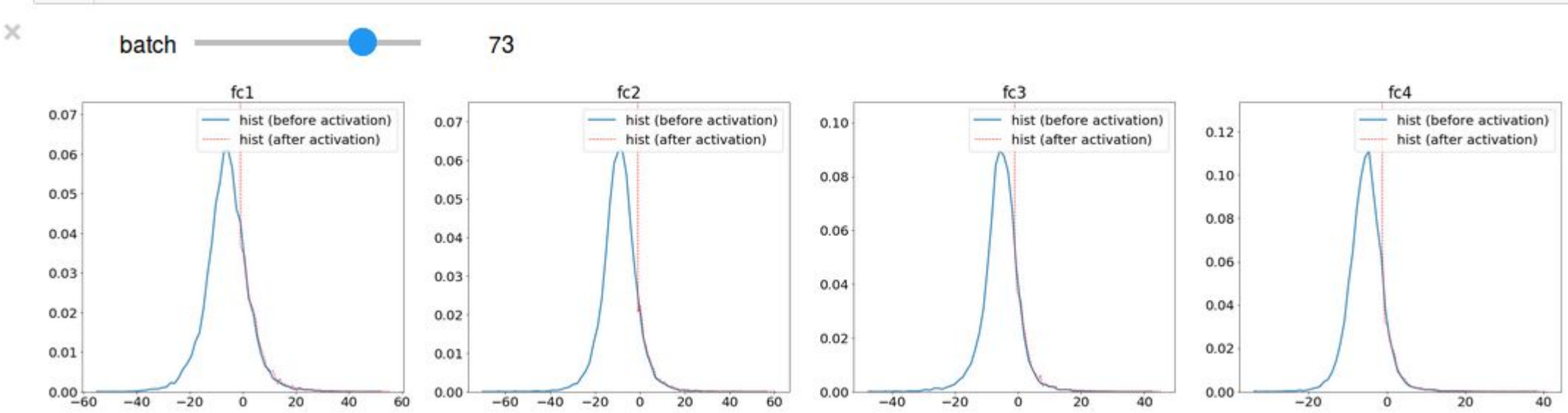
CIFAR-10 49 results collected

Units: accuracy %

Classify 32x32 colour images.

Result	Method	Venue	Detail
96.53%	Fractional Max-Pooling	arXiv 2015	Detail
95.59%	Striving for Simplicity: The All Convolutional Net	ICLR 2015	Detail
94.16%	All you need is a good init	ICLR 2016	Detail
94%	Lessons learned from manually classifying CIFAR-10	unpublished 2011	Detail
93.95%	Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree	AISTATS 2016	Detail
93.72%	Spatially-sparse convolutional neural networks	arXiv 2014	
93.63%	Scalable Bayesian Optimization Using Deep Neural Networks	ICML 2015	
93.57%	Deep Residual Learning for Image Recognition	arXiv 2015	Detail
93.45%	Fast and Accurate Deep Network Learning by Exponential Linear Units	arXiv 2015	Detail
93.34%	Universum Prescription: Regularization using Unlabeled Data	arXiv 2015	
93.25%	Batch-normalized Maxout Network in Network	arXiv 2015	Detail

My experiments - Vgg like CNN about 2M params



Histograms of activations???

Another benchmarks

Systematic evaluation of CNN advances on the ImageNet

Dmytro Mishkin[a], Nikolay Sergievskiy[b], Jiri Matas[a]

[a]*Center for Machine Perception, Faculty of Electrical Engineering, Czech Technical University in Prague. Karlovo namesti, 13. Prague 2, 12135*
[b]*ELVEES NeoTek, Proyezd 4922, 4 build. 2, Zelenograd, Moscow, Russian Federation, 124498*

https://arxiv.org/abs/1606.02228

- Super cool benchmark github project!!!
- https://github.com/ducha-aiki/caffenet-benchmark

Activations

Name	Accuracy	LogLoss	Comments
ReLU	0.470	2.36	With LRN layers
ReLU	0.471	2.36	No LRN, as in rest
TanH	0.401	2.78	
1.73TanH(2x/3)	0.423	2.66	As recommended in Efficient BackProp, LeCun98
ArcSinH	0.417	2.71	
VLReLU	0.469	2.40	y=max(x,x/3)
RReLU	0.478	2.32	
Maxout	0.482	2.30	sqrt(2) narrower layers, 2 pieces. Same complexity, as for ReLU
Maxout	*0.517*	*2.12*	same width layers, 2 pieces
PReLU	0.485	2.29	
ELU	0.488	2.28	alpha=1, as in paper
ELU	0.485	2.29	alpha=0.5

Questions?

References

- https://en.wikipedia.org/wiki/Activation_function - nice list of differenct activation functions
- https://arxiv.org/pdf/1511.07289.pdf - elu paper
- https://calculatedcontent.com/2017/06/16/normalization-in-deep-learning/ - quite well written blog about normalization in NNs
- https://github.com/NVIDIA/DIGITS/blob/master/examples/weight-init/README.md - comparison of different initialization schemes and their effect on learning process
- https://www.reddit.com/r/MachineLearning/comments/6n9bkm/p_understanding_visualizing_selfnormalizing/ - discussion about SELU on reddit
- https://gist.github.com/eamartin/d7f1f71e5ce54112fe05e2f2f17ebedf - some insights on how SELUs works. Actually very nice jupyter notebook.
- https://www.reddit.com/r/MachineLearning/comments/6g5tg1/r_selfnormalizing_neural_networks_improved_elu/dise43y/ - reddit discussion

References

- https://arxiv.org/pdf/1511.06422.pdf - All you need is a good init
- https://arxiv.org/pdf/1703.01827.pdf - All You Need is Beyond a Good Init
- https://github.com/ducha-aiki/caffenet-benchmark - cool benchmark on different configurations of regularization and activations functions
- https://arxiv.org/abs/1606.02228 - Systematic evaluation of CNN advances on the ImageNet
- https://arxiv.org/abs/1706.02515 - Self-Normalizing Neural Networks

FORNAX

WWW.FORNAX.AI