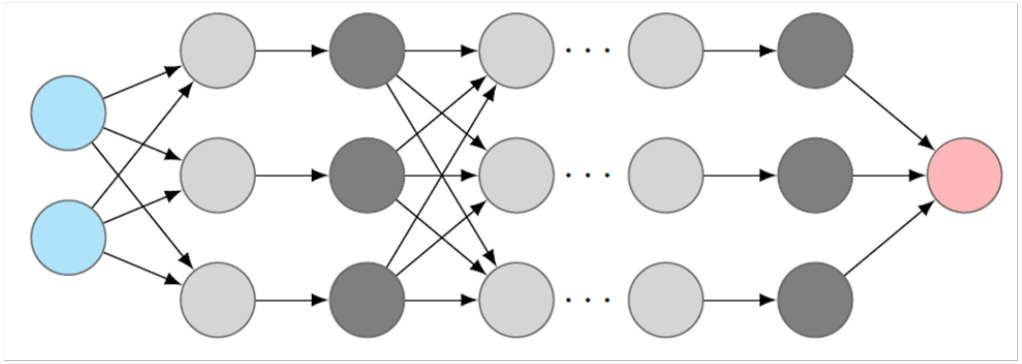
# Training neural networks

*- How to start -*

Lecturer: Michael Möller – michael.moeller@uni-siegen.de
Exercises: Hartmut Bauermeister – hartmut.bauermeister@uni-siegen.de



# Initialization schemes

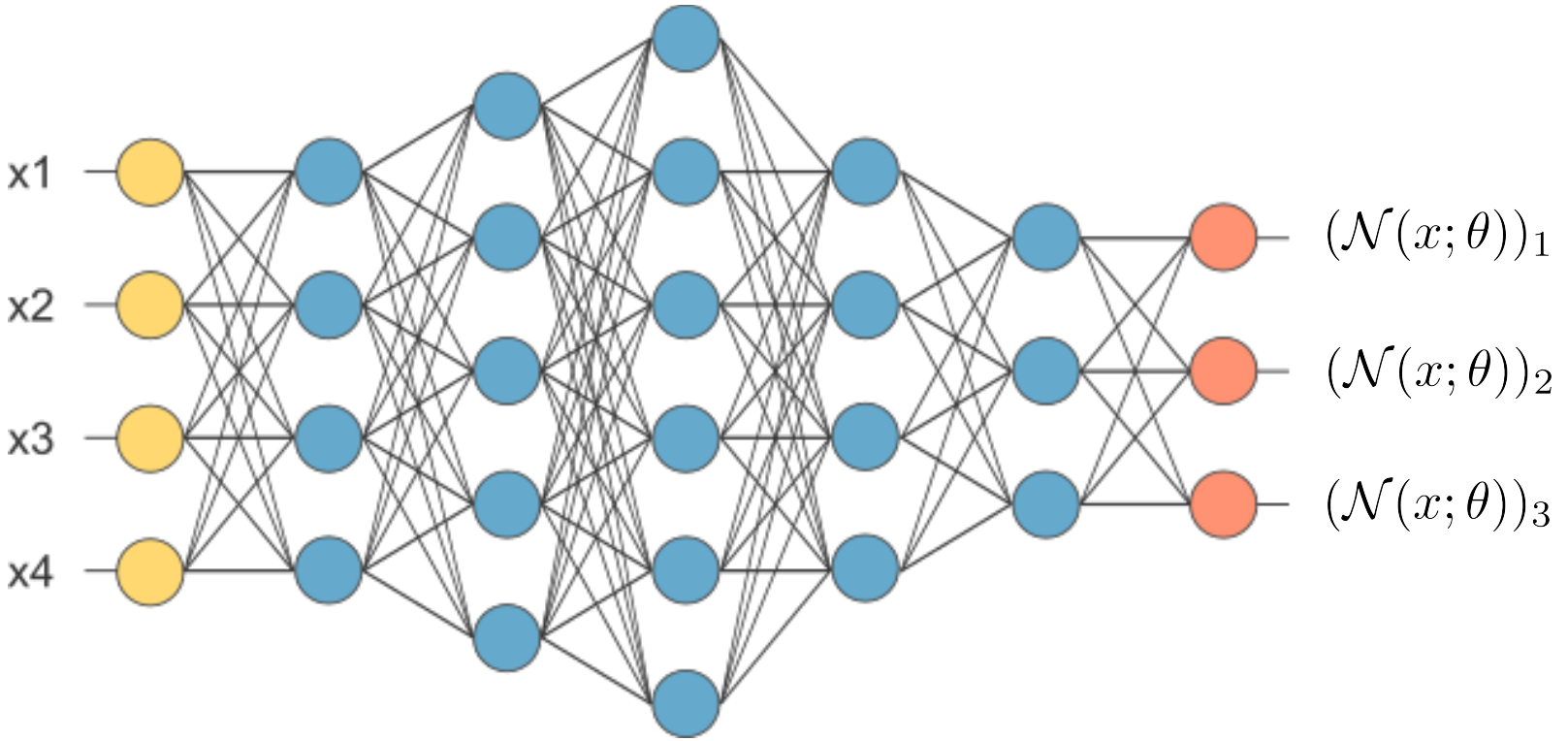



Image taken from https://www.neuraldesigner.com/

$$\mathcal{N}(x;\theta) = \phi^L(\sigma(\phi^{L-1}(\ldots(\sigma(\phi^1(x;\theta^1))\ldots);\theta^{L-1}));\theta^L)$$ *Fully connected network*

$$\phi^i(z;\phi^i) = \phi^i_w z + \phi^i_b, \qquad \forall i \in \{1,\ldots,L\}$$ *Fully connected layer*

$$\sigma(x) = \max(x,0)$$ *Rectified Linear Unit (ReLU)*

# Initialization schemes

We want to minimize an energy like

$$E(\theta) = \frac{1}{N} \sum_{j=1}^{N} \mathcal{L}(\mathcal{N}(x_j; \theta), y_j)$$

with some variant of SGD, e.g. with Adam, which means variants of

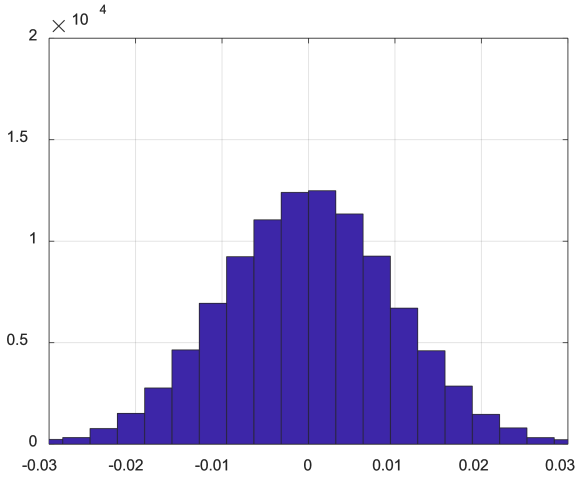
$$\theta(k+1) = \theta(k) - \tau(k) \nabla E_k(\theta(k))$$

are at the heart of the iterative algorithm.

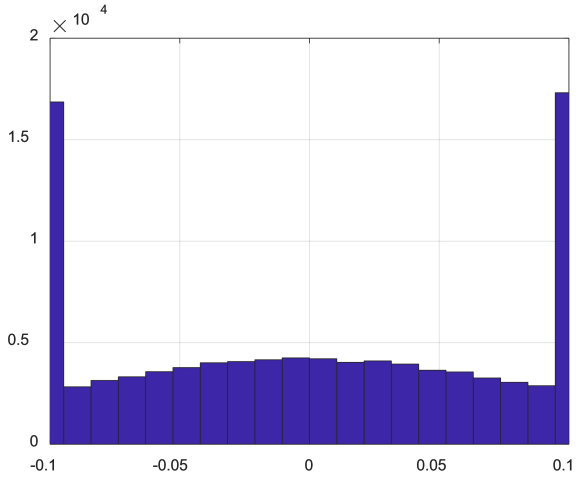
**What should our initial weights $\theta(0)$ be? Is this choice important?**

# Initialization schemes

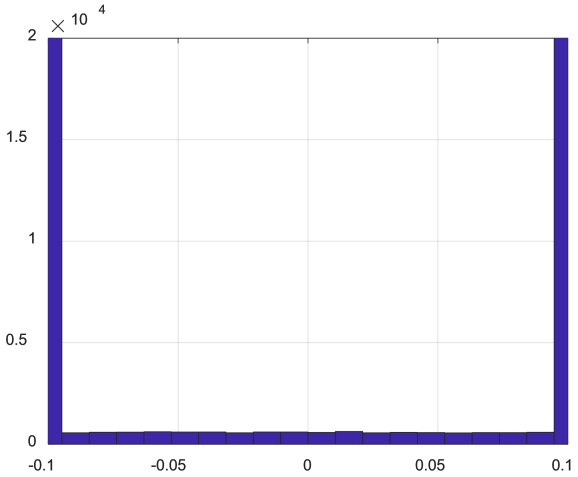
Consider a fully connected network without biases, and all weights initialized with a zero-mean Gaussian distribution of standard deviation 1.




Distribution of the data




Distribution after 1st layer

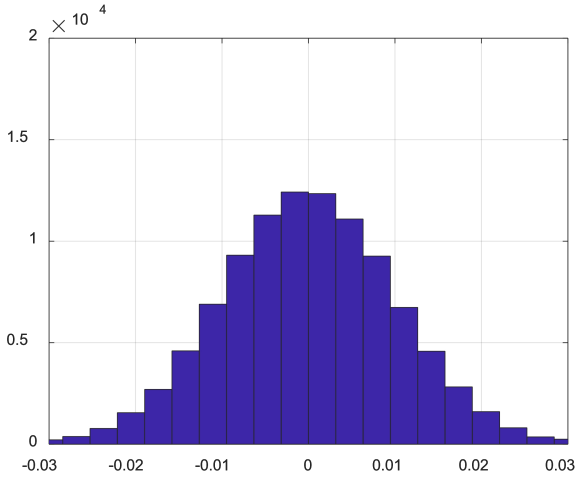



Distribution after 2nd layer

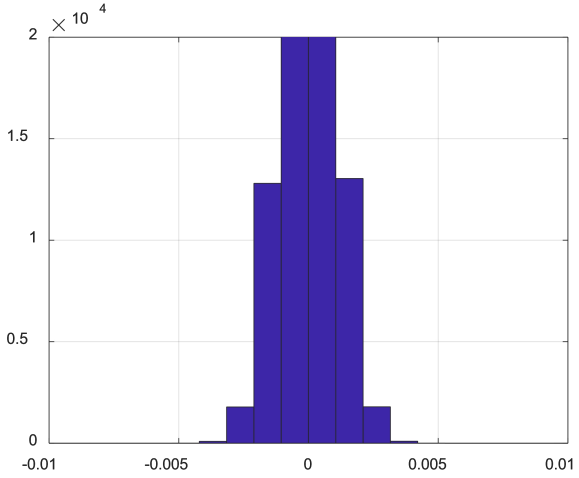
After 10 layers the mean absolute value has increased by a factor of more than 100 million!

# Initialization schemes

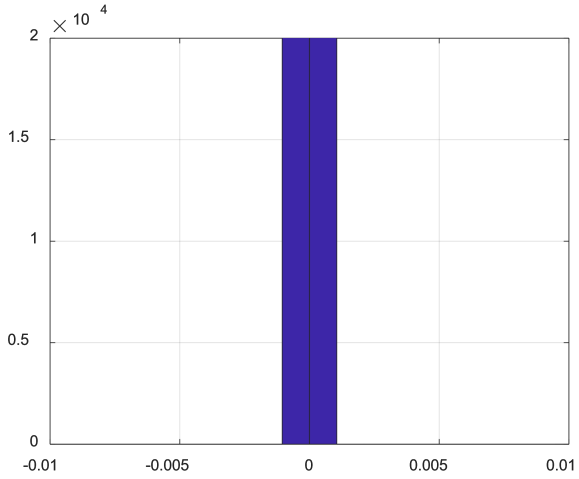
Consider a fully connected network without biases, and all weights initialized with a zero-mean Gaussian distribution of standard deviation 0.01.




Distribution of the data




Distribution after 1st layer




Distribution after 2nd layer

After 10 layers the mean absolute value has decreased by a factor of more than 1 trillion!

Discussion in the lecture: Backpropagation multiplies with the transpose activations. This leads to huge or tiny gradients!

# Initialization schemes

We will see that for zero-mean random variables $x \in \mathbb{R}^n$ and $v \in \mathbb{R}^n$ in which all entries of v and all entries of x have the same variance it holds that

$$\mathrm{var}(v^T x) = n\ \mathrm{var}(v_i)\ \ \mathrm{var}(x_i)$$

This would motivate the choice $\mathrm{var}(\theta^i) = \frac{1}{n}$ for a fully connected layer $\theta^i \in \mathbb{R}^{n \times n}$

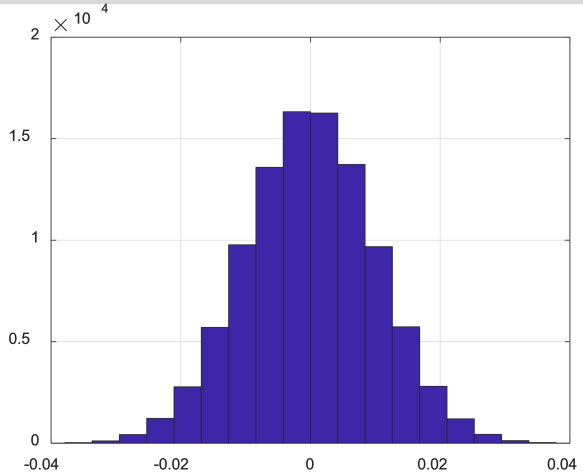
We choose $\mathrm{var}(\theta^i) = \frac{2}{n}$ for a fully connected layer with ReLU activations, "because the ReLU eliminates half of the data".

We choose $\mathrm{var}(\theta^i) = \frac{4}{n_i + n_{i+1}}$ for a fully connected layer $\theta^i \in \mathbb{R}^{n_{i+1} \times n_i}$ (with ReLU )

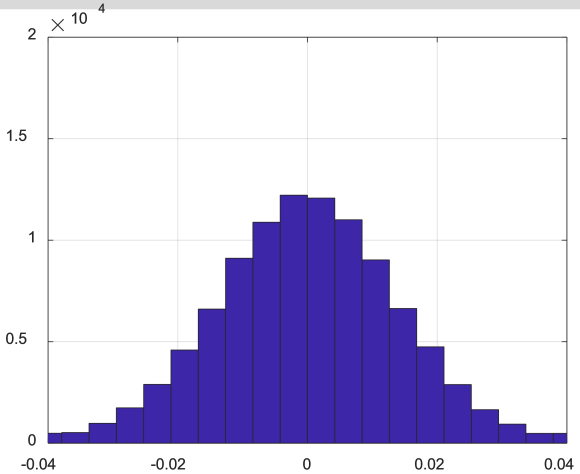
*He et al., "Delving Deep into Rectifiers", 2015*

*Glorot and Bengio, "Understanding the difficulty of training deep feedforward neural networks", 2010.*

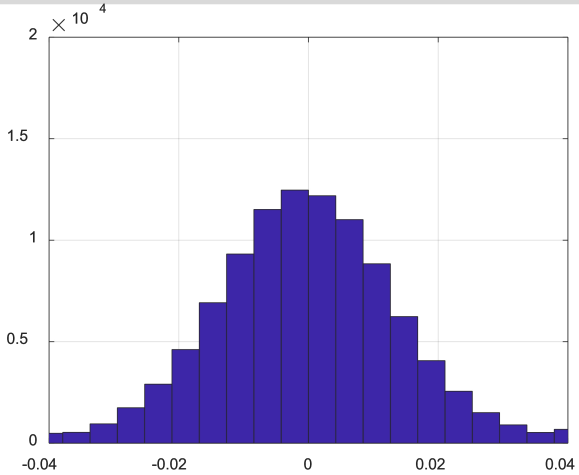
# Initialization schemes



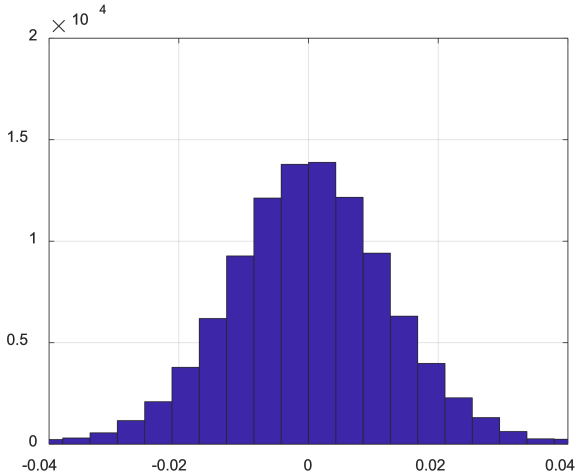

Distribution of the data



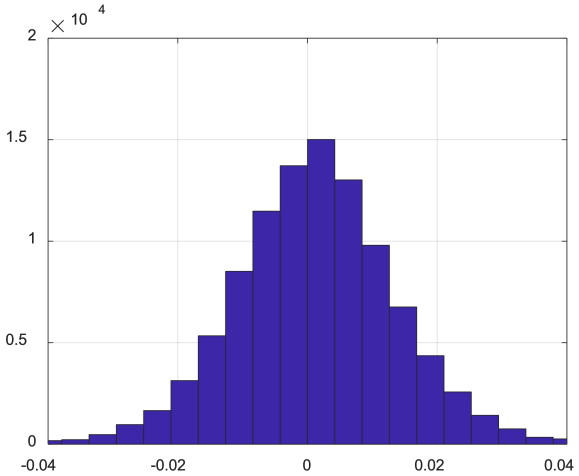

Distribution after 2nd layer



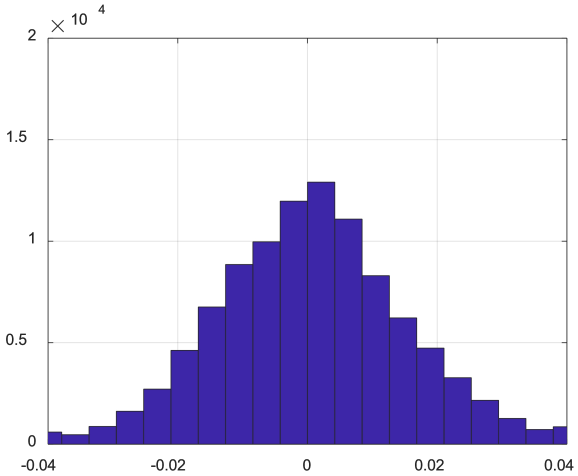

Distribution after 4th layer




Distribution after 6th layer

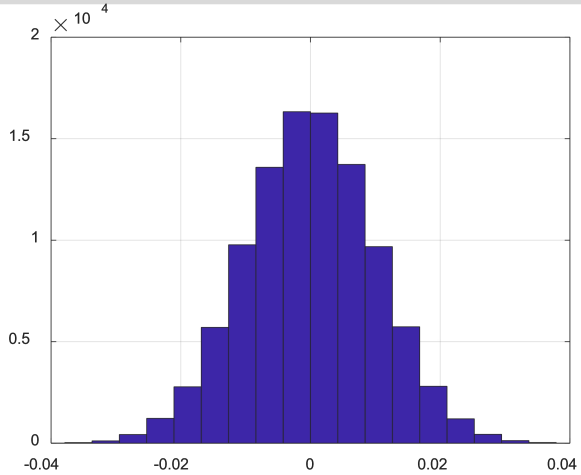



Distribution after 8th layer




Distribution after 10th layer

# Initialization schemes

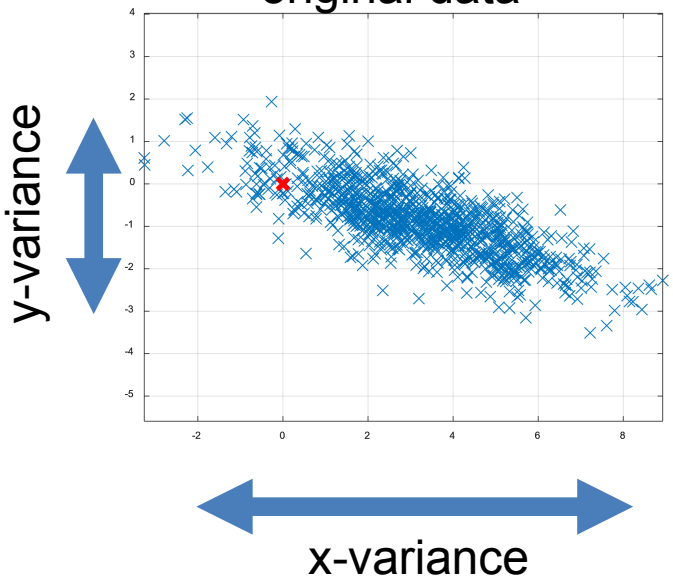



Distribution of the data

**Careful: We assumed a zero-mean distribution of the input data!**

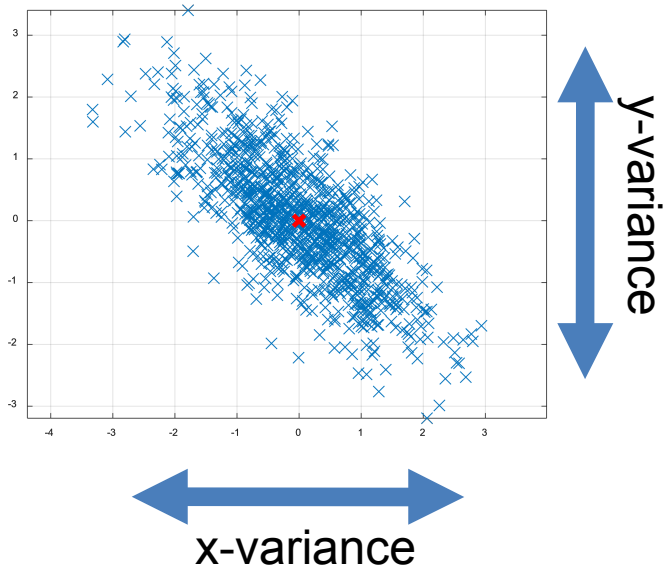
Weight initialization (and possibly the entire training) fails if the data already has a significant mean!

original data




preprocessing

zero-centered normalized data



# Batch normalization

**If we can just normalize our input data, can we also normalize the output or our favorite layers?**


from www.amazon.com

For a current minibatch, collect all activations of the neuron you'd like to normalize, i.e. $z \in \mathbb{R}^{1 \times |I(k)|}$ and compute

$\mu = \text{mean}(z)$ minibatch mean

$\sigma^2 = \text{var}(z)$ minibatch variance

Output a normalized version $\frac{z - \mu}{\sqrt{\sigma^2 + \epsilon}}$

One can differentiate through these operations!

https://arxiv.org/pdf/1502.03167.pdf, *Batch normalization,* Ioffe, Szegedy, 2015

# Batch normalization

**But maybe the layer does not like being normalized…**

Allow to “learn undoing” the normalization!

$\mu = \mathrm{mean}(z)$ minibatch mean

$\sigma^2 = \mathrm{var}(z)$ minibatch variance

Output $\mathrm{BN}(z; \theta_\sigma^{BN}, \theta_\mu^{BN}) = \theta_\sigma^{BN} \cdot \left( \frac{z - \mu}{\sqrt{\sigma^2 + \epsilon}} \right) + \theta_\mu^{BN}$

These **batch normalization layers** are frequently used in state-of-the-art networks!

They are typically placed before the activation function, and seem to greatly ease the difficulties in initializing and training neural networks with various architectures!

# Batch normalization

**And how do the work during inference?**

$\mu = \mathrm{mean}(z)$ minibatch mean << *There is no minibatch during inference*

$\sigma^2 = \mathrm{var}(z)$ minibatch variance

Output $\mathrm{BN}(z; \theta_\sigma^{BN}, \theta_\mu^{BN}) = \theta_\sigma^{BN} \cdot \left( \frac{z - \mu}{\sqrt{\sigma^2 + \epsilon}} \right) + \theta_\mu^{BN}$

One trains a moving average of means and variances of the batches during training.

These numbers are fixed and used as $\mu$ and $\sigma$ during inference.

# Summary

Pay attention to how you set up your training! Particularly

- It usually makes sense to preprocess your data. Make it zero-mean. Possibly normalize.
- Choose the initial weights wisely. Avoid exploding or vanishing activations.
- Utilize batch-norm to make your network less dependent on the initial weights and network architecture.
- Further reading: Self-normalizing neural networks (Klambauer et al. 2017) pose an interesting alternative (using scaled exponential linear units SELUS), see https://papers.nips.cc/paper/6698-self-normalizing-neural-networks.pdf