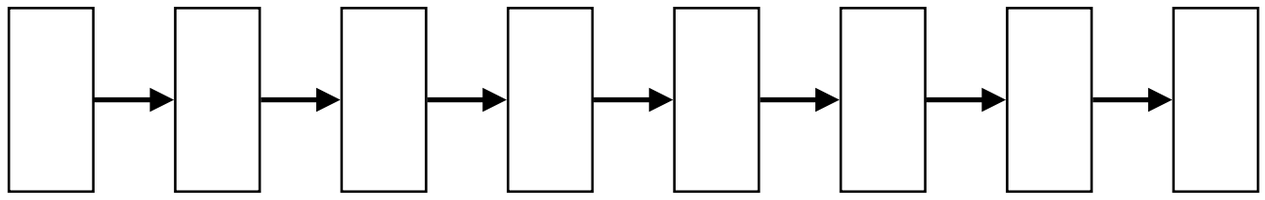
Advanced Deep Learning (BEV033DLA)
Lecture 4
Training Deep Models

Czech Technical University in Prague

- Weight Initialization and Reparameterization
  - Analytic / Data-driven methods
  - Self-Normalizing Neural Networks
  - Batch Normalization
- Residual Networks

Challenges of Training Deep Models

- Vanishing / exploding gradient problem
  - particularly observed and pronounced in deep and recurrent neural networks



Gradient: $(J_1^\mathsf{T})(J_2^\mathsf{T})(J_3^\mathsf{T})\dots(J_L^\mathsf{T})\nabla\mathcal{L}$

- If each J_k is *contractive*, the gradient magnitude *vanishes* with many layers

  Example: sigmoid activation $\mathcal{S}(x) = \frac{1}{1+e^{-x}}$

  $J(x) = \mathcal{S}(x)(1-\mathcal{S}(x)) \leq \frac{1}{4}$

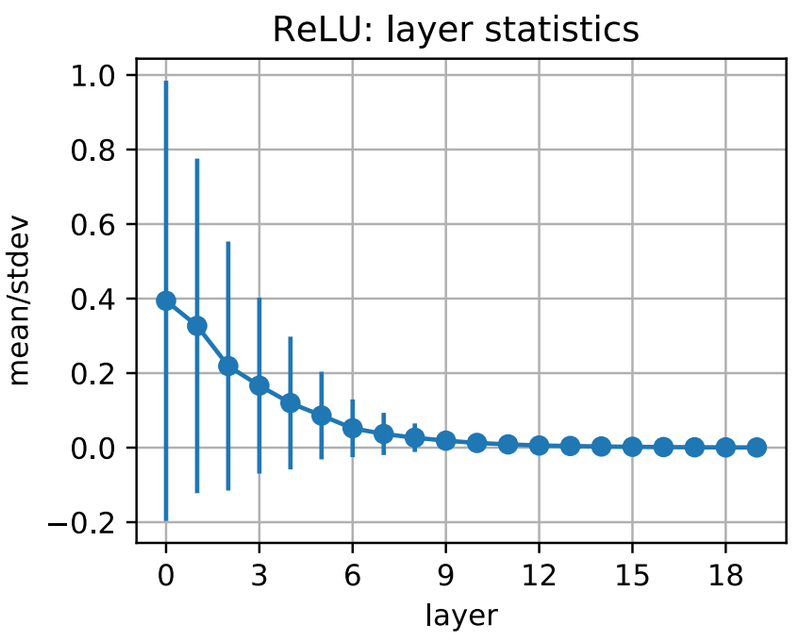
- If each J_k is expansive, the gradient magnitude *explodes*

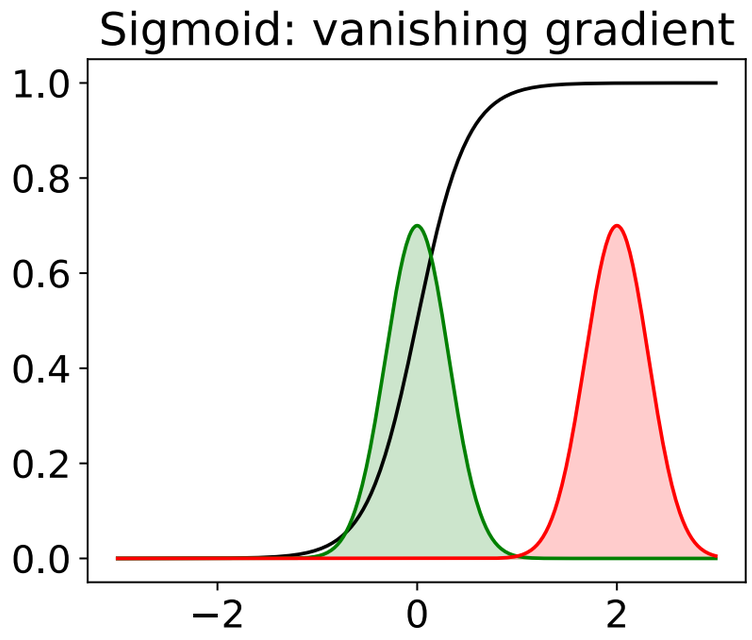
  Example: Linear layer $W^k x$

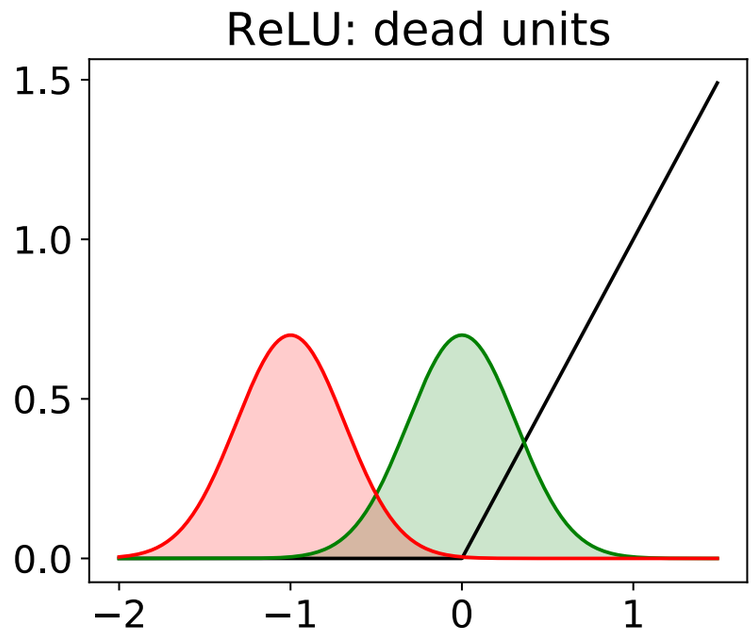
  $J(x) = W$, can be *expansive* if maximum singular value of W is greater than 1

Challenges of Training Deep Models

- Degenerate activation statistics
  - Gradients depend on the operating point
  - **Example**: initialize all weights and biases randomly







Left: node statistics for the layers of a deep FFN with ReLU units with random inputs, all weights initialized from a normal distribution with the same scale.

Middle and right: statistics of pre-activations: if for almost all data points pre-activations are in the region where gradient is close to zero this can lead to vanishing gradients and "dead units"

Challenges of Training Deep Models

- How can we improve training efficiency? (c.f. phenomenon: models that train faster by SGD also generalize better)
  - ReLU instead of sigmoid
  - Weight initialization
  - Batch normalization / RMS normalization
  - Preconditioned / adaptive gradient descent
  - Increase network width
  - Residual connections -- network architecture can make the optimization easier
  - ...

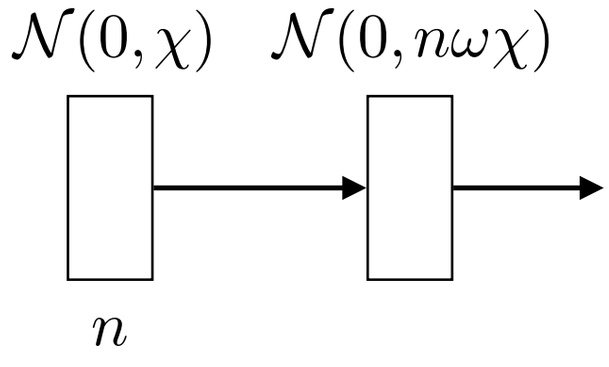
Weight Initialization: Glorot

- **Proper initialization**: Initialize weights/biases so that each neuron has activation statistic (over the dataset) with certain mean and variance

- **Example:** (Glorot & Bengio, 2010) Analyze variance of neuron outputs and backprop gradients under the following simplifying assumptions
  - $\tanh$ activation function $f(x)$ in linear regime, i,e, $f(x) \approx x$
  - Neuron outputs as well as gradient components are i.i.d.

  Start from a single neuron $y = w^T x$, $x \in \mathbb{R}^n$. Assume
  - x_i are i.i.d. with $\mathbb{E}[x_i] = 0$ and $\mathbb{V}[x_i] = \chi$
  - w_i are i.i.d. with $\mathbb{E}[w_i] = 0$ and $\mathbb{V}[w_i] = \omega$

  It follows that $\mathbb{E}[y] = 0$ and $\mathbb{V}[y] = n\omega\chi$.



Weight Initialization: Glorot

- Consider now a feedforward network with $\tanh$ activation and assumptions as above.

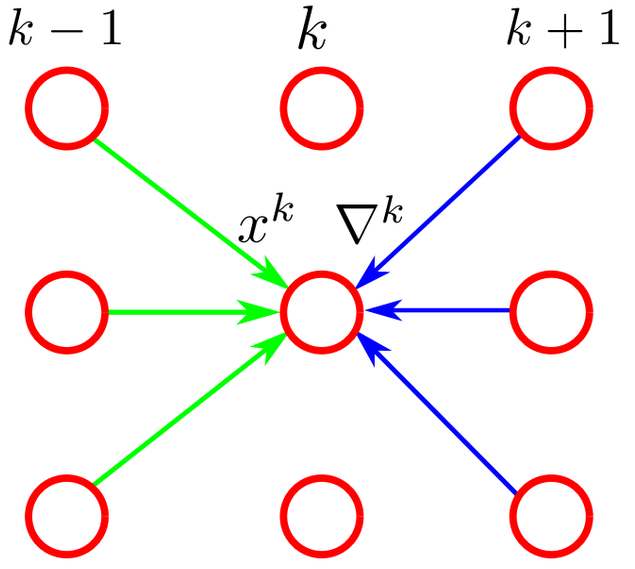
  For layer k with n_k nodes, denote the neuron outputs by x^k and gradients by ∇^k.

  Denote the variance of weights in layer k by ω_k.

- Assuming that the inputs x^0 have zero mean and unit variance
- forward: $\mathbb{V}[x_i^k] = n_{k-1}\omega_k \mathbb{V}[x_j^{k-1}]$

  We want $\mathbb{V}[x_i^k] \approx \mathbb{V}[x_j^{k-1}]$, i.e. $n_{k-1}\omega_k = 1$.
- backward: $\mathbb{V}[\nabla_i^k] = n_{k+1}\omega_{k+1} \mathbb{V}[\nabla_j^{k+1}]$

  We want $\mathbb{V}[\nabla_i^k] \approx \mathbb{V}[\nabla_j^{k+1}]$, i.e. $n_{k+1}w_{k+1} = 1$
- Compromise: set $\omega_k = \frac{2}{n_{k-1}+n_k}$
- Randomly initialize the weights as normal $w_{ij}^k \sim \mathcal{N}(0, \omega_k)$.
- Or as uniform $w_{ij}^k \sim \mathcal{U}(-d, d)$ with $d = \sqrt{\frac{\omega_k}{3}}$ (the variance is then ω_k).



Weight Initialization: General Case

- Consider the typical network structure:

  $a^k = W^k x^{k-1}$ preactivatinos

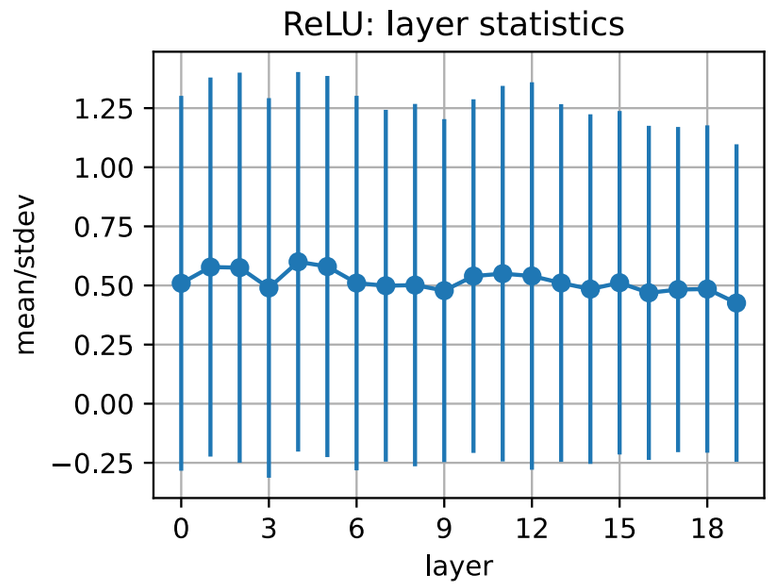
  $x^k = f(a^k)$ non-linear activation function

  $a^{k+1} = W^{k+1} x^k$ preactivatinos of the next layer

- Procedure for general f:
  - Assume pre-activations follow Normal distribution, $a^k \sim \mathcal{N}(0, I)$
  - Assume weights are Normally distributed, $W^{k+1} \sim \mathcal{N}(0, \omega I)$
  - Statistics of a^{k+1} (consider one output with weight vector w for simplicity):
    - Mean $\mathbb{E}[w^\mathsf{T} x^k] = \mathbb{E}[w]^\mathsf{T} E[x^k] = 0$
    - Variance $\mathbb{V}[w^\mathsf{T} x^k] = \mathbb{V}[\sum_i w_i x_i^k] = \sum_i \mathbb{V}[w_i x_i^k] = n_k \omega \mathbb{E}[(x_i^k)^2]$
  - Analytically / numerically compute $\eta = \mathbb{E}_{a \sim \mathcal{N}(0,1)}[f(a)^2]$
  - Initialize variance as $\omega = \frac{1}{n_k \eta}$ or as $\omega = \frac{2}{(n_k + n_{k-1})\eta}$
- $f(a) \approx a$: $\eta = 1$ (Glorot & Bengio, 2010)
- ReLU: $\eta = \frac{1}{2}$ (He initialization, He et al. 2015)

  Sigmoid: $\eta = 0.2928$

  $\tanh$: $\eta = 0.3948$



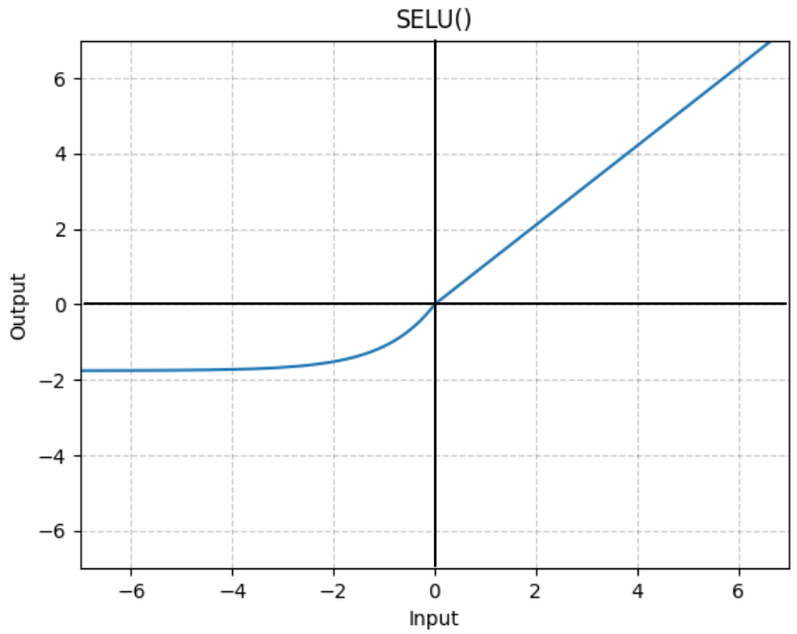
Self-Normalizing Neural Networks

- Scaled Exponential Linear Unit:

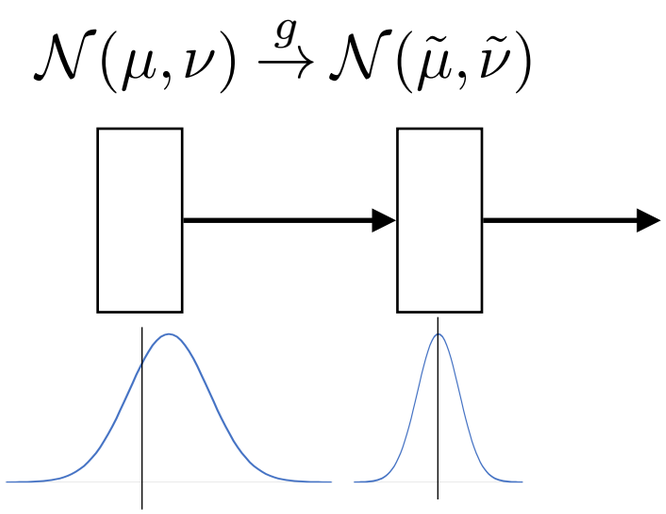
$$\mathrm{SELU}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases}$$

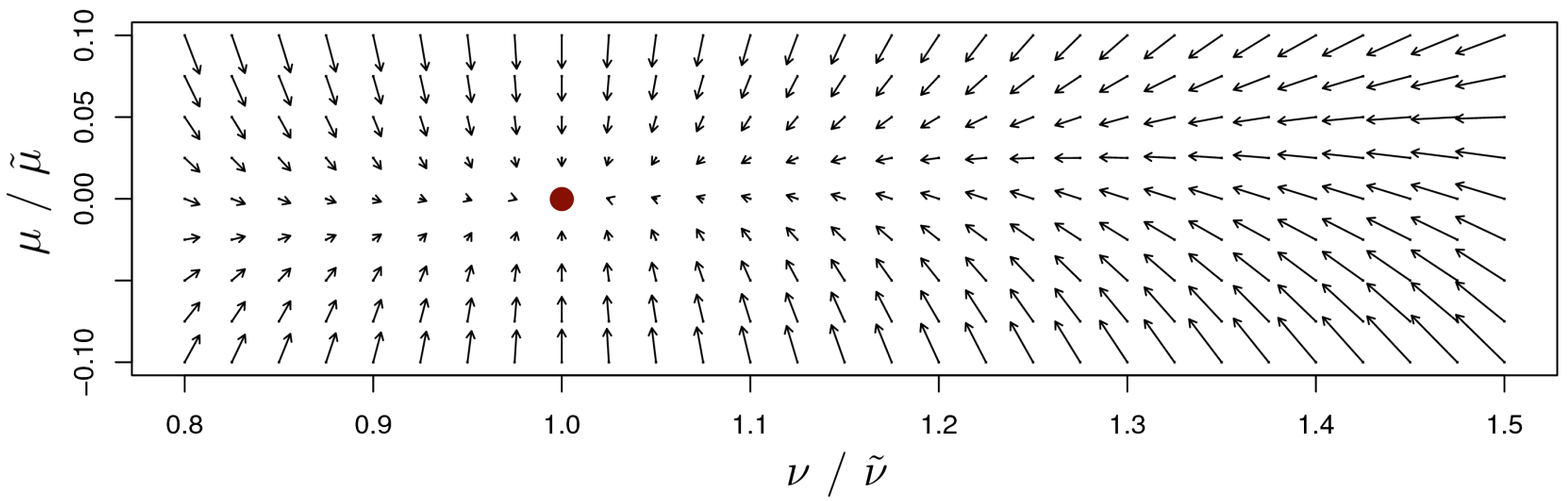
with parameters $\lambda \approx 1.0507$, $\alpha \approx 1.6733$

  - $\eta = \mathbb{E}_{a \sim \mathcal{N}(0,1)}[\mathrm{SELU}(a)^2] = 1$



- **Theorem** (Klambauer et al. 2017): The mapping g of mean and variance statistics (μ, ν) is contractive, there is a stable fixed point.



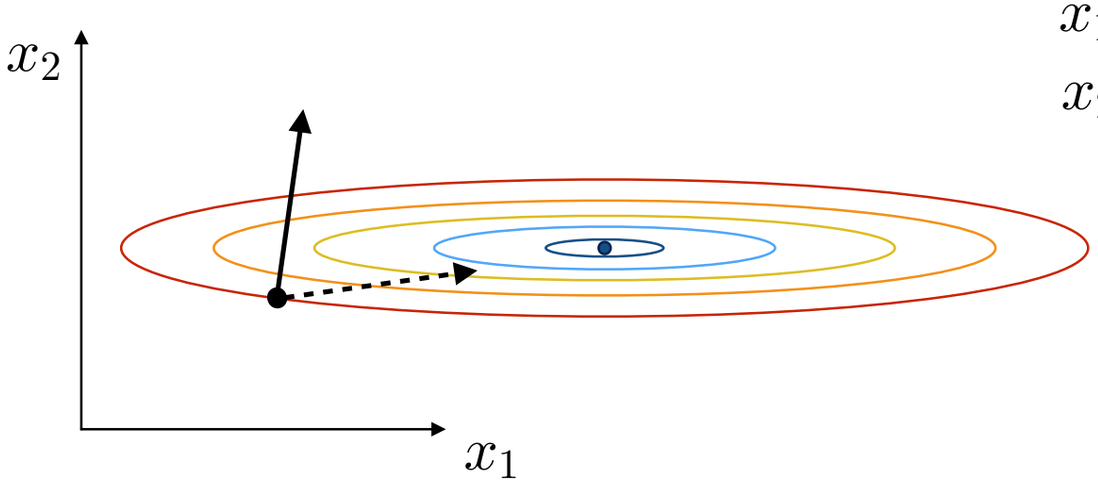


[Klambauer et al. 2017: Self-Normalizing Neural Networks]

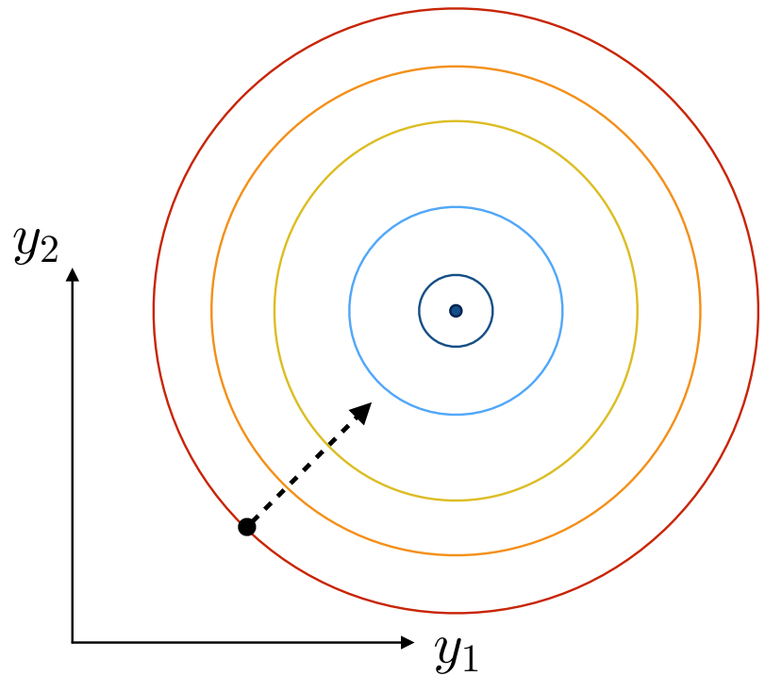
Gradient Descent Under Reparameterization

- Let $f : \mathbb{R}^n \to \mathbb{R}$ and its derivative $J(w) = \frac{\mathrm{d}f(w)}{dw}$

  Gradient descent:
  - $w_{t+1} = w_t - \alpha J\left(w_t\right)^{\mathsf{T}}$
- Make a substitution: $w = A\tilde{w}$ (change of coordinate) and consider GD in $\tilde{w}$:
  - Problem in new coordinates: $\min_{\tilde{w} \in \mathbb{R}^n} f(A\tilde{w})$
  - GD: $\tilde{w}_{t+1} = \tilde{w}_t - \alpha \left(J\left(A\tilde{w}_t\right) A\right)^{\mathsf{T}}$
- Substitute back $\tilde{w} = A^{-1}w$:
  - $A^{-1}w_{t+1} = A^{-1}w_t - \alpha A^{\mathsf{T}} J\left(w_t\right)^{\mathsf{T}}$
  - Effect in original space: $w_{t+1} = w_t - \alpha \left(AA^{\mathsf{T}}\right) J\left(w_t\right)^{\mathsf{T}}$



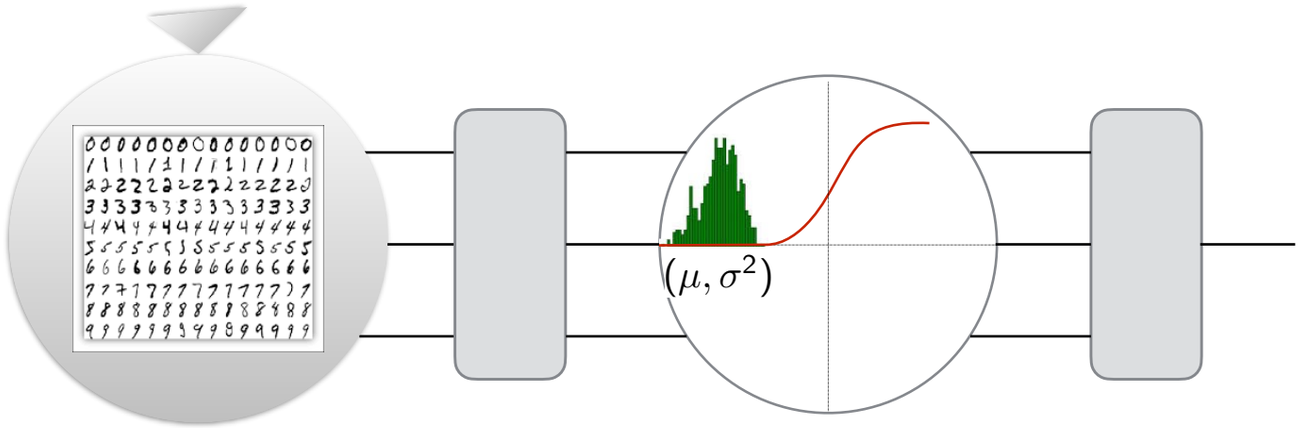
$$x_1 = y_1$$
$$x_2 = 5y_2$$



- **Corollary**: initialization of the weight scale also affects GD dynamics

Batch Normalization

- Can estimate statistics of pre-activations from the data
  - more accurate than using the Gaussian i.i.d. model

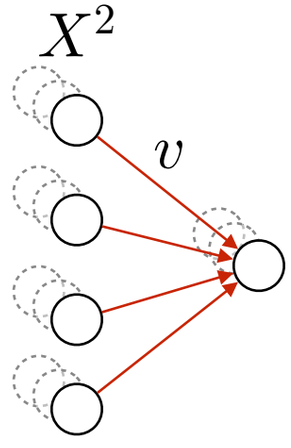


- **Batch normalization** (Joffe & Szegedy, 2015):
  - We actually process data in batches anyhow, let's use the current batch for building the statistics!
  - A – batch of features
  - $BN(A) = \dfrac{A - \mu(A)}{\sigma(A)}\beta + \gamma$
  - μ, σ – mean and standard deviation over batch
  - β, γ – learnable parameters initialized as $\beta = \mathbf{1}, \gamma = \mathbf{0}$.

Batch Normalization

- $BN(A) = \frac{A-\mu(A)}{\sigma(A)}\beta + \gamma$, where A is a batch of activations, μ, σ are batch mean and std.
- **Initialization advantage:** $\beta = 1$, $\gamma = 0 \Rightarrow$ statistics before activations are standardized $\Rightarrow$ activations do not degenerate on average
- **Invariant to the weight scale:** for $A = w^\mathsf{T} X$ we have $\frac{s w^\mathsf{T} X - \mu(s w^\mathsf{T} X)}{\sigma(s w^\mathsf{T} X)} = \frac{w^\mathsf{T} X - \mu(w^\mathsf{T} X)}{\sigma(w^\mathsf{T} X)}$
  - Does the weight scale not matter any more?
  - Steepest descent step for $\tilde{w} = sw$ with learning rate 1 is equivalent to the steepest descent step for w with learning rate $\frac{1}{s^2} \Rightarrow$ *weight initialization scale controls the local learning rate*
- **Reparameterization Advantage**: consider an example:

$$X^2 = \mathrm{BN}(\mathrm{ReLU}(X^1))$$
$$X^3 = v^\mathsf{T} X^2$$
$$\mathcal{L}_b = \mathcal{S}(y_b X^3_b), \quad \mathsf{b} = 1 \ldots \mathsf{B},\ y_b \in \{-1, 1\}$$



  - $\frac{d\mathcal{L}_b}{dv} = \underbrace{\frac{d\mathcal{L}}{dX^3_b}}_{\text{scalar}} X^2_b$
  - X^2 standardized $\Rightarrow$ gradient components have similar scale $\Rightarrow$ good conditioning
  - The idea can be deepened to (approximatelly) relate activation standardization with *Natural Gradient*

LayerNorm, RMSNorm

- Recently, simpler alternatives are more common (in transformers, large-scale training)
- **LayerNorm**:

$$y = \frac{x - \hat{\mu}}{\hat{\sigma}}$$

  - Using mean $\hat{\mu}$ and standard deviation $\hat{\sigma}$ computed over units only (not over batch)
    $\Rightarrow$ no issue with training-test difference
  - Completely similar reasoning as BN (but no generalization boost)
  - Mostly used in CNNs with statistics over spatial dimensions per channel

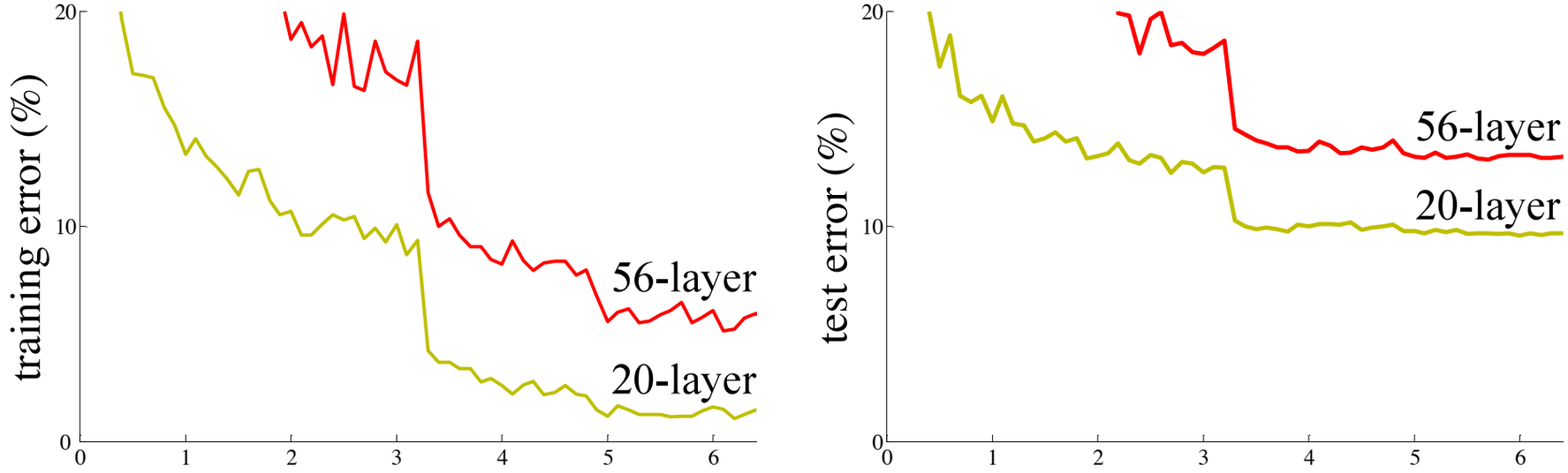
- Root Mean Square Normalization (**RMSNorm**):

$$y = \frac{x}{\sqrt{\frac{1}{n}\sum_i x_i^2}} = \frac{x}{\|x\|_2}\sqrt{n}$$

  - Same as LayerNorm over channels (1D tokens in transformers)
  - No mean subtraction, emphasizing that mostly the scale is important
  - Assume we have followed the statistical initialization model in which $x \sim \mathcal{N}(0, I)$
  - Then $\mathbb{E}[\|x\|_2^2] = \mathbb{E}[\sum_i x_i^2] = \sum_i \mathbb{E}[x_i^2] = \sum_i 1 = n$
  - Therefore, RMSNorm tries to maintain the initialization scale assumptions when $\|x\|$

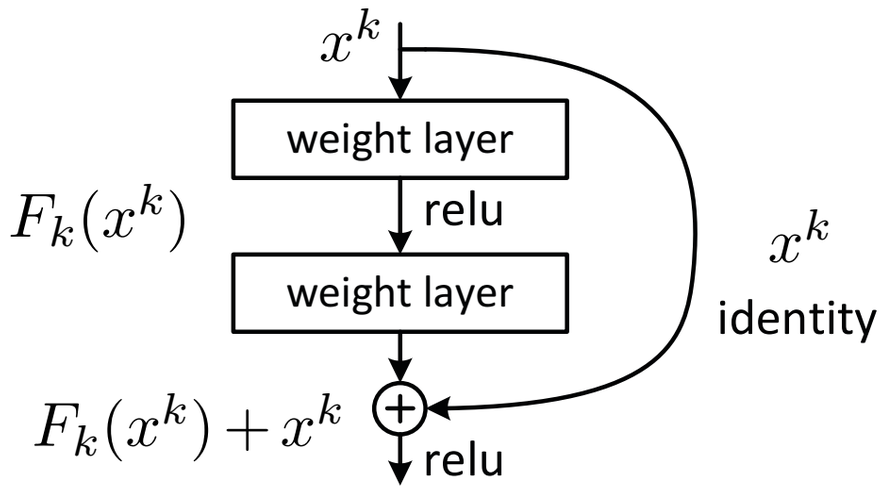
Residual Networks

- By using proper weigh initialization / BN we can learn deep networks with up to 20-30 layers. But for deeper models the training performance becomes a limitation.



Training error and test error on CIFAR-10 with 20-layer and 56-layer "plain" networks

- Idea: introduce skip (shortcut) connections:



[He et al. 2015: Deep Residual Learning for Image Recognition]

Residual Networks

- Residual learning view:

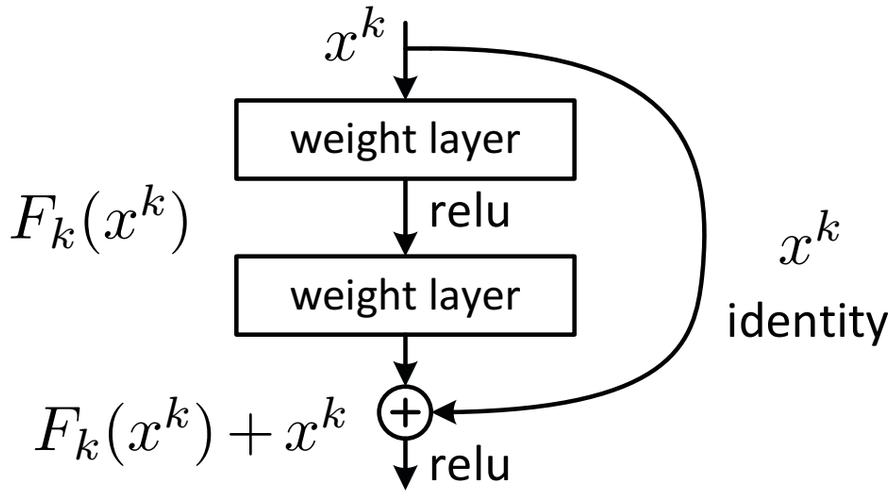
  Consider regression problem of predicting x^* based on x^0:

  $$\min_{\theta_0} \mathbb{E}_{(x^0,x^*)}\left[\left(x^* - F_0(x^0;\theta_0)\right)^2\right]$$

  - Let $x^1 = F^0(x^0;\theta_0)$ current prediction, then $r^1 = x^* - x^1$ is the *residual error* not explained by the model
  - Try to correct it with F_1 based on the current prediction x^1:

    $$\min_{\theta_1} \mathbb{E}_{(x^1,x^*)}\left[\left(\underbrace{(x^* - x^1)}_{r_1} - F_1(x^1;\theta_1)\right)^2\right]$$

  - Corrected predictor: $x^1 + F_1(x^1;\theta_1)$
  - Can also learn this predictor jointly, corresponds to the skip architecture



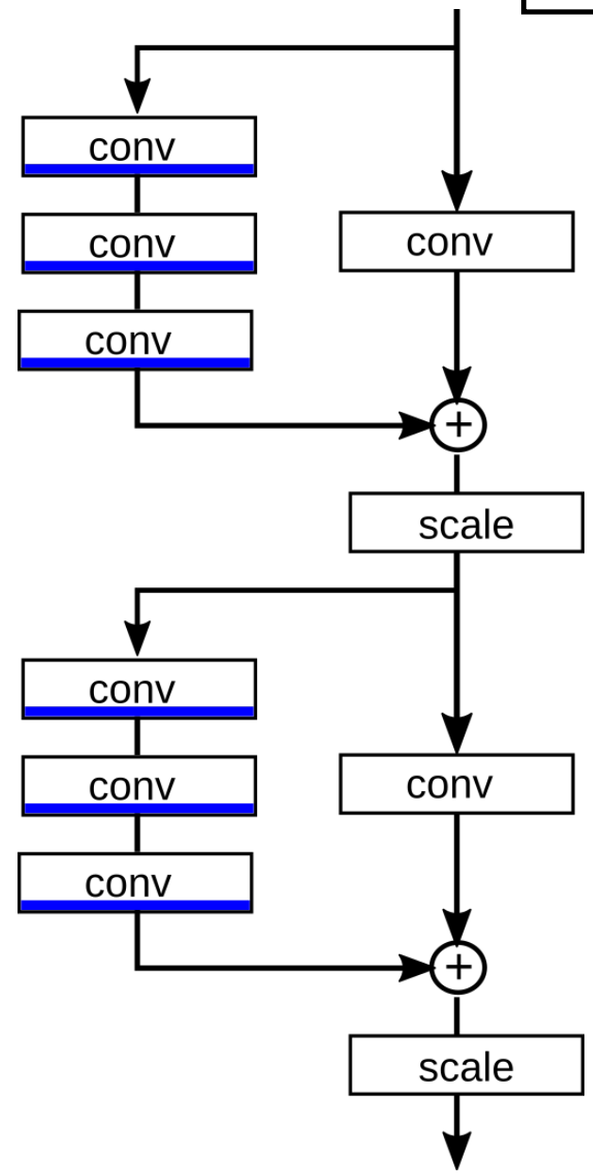
Residual Networks

- Highway architecture view:

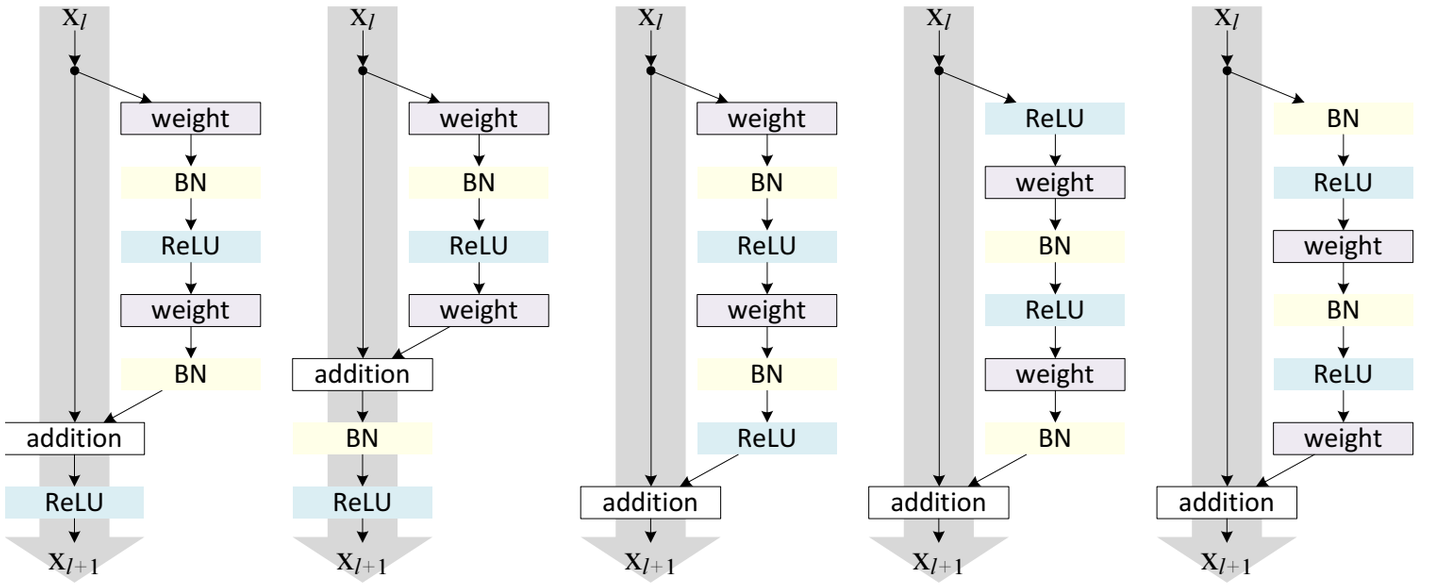
  We have two linked networks:

  - a "highway" network with few layers,
  - a very deep network which adds "corrections" the former.

  This improves trainability of the network. It becomes possible to train networks with 100 and more layers.



- Variants (pre-activation) and further...



- Attempts for theoretical explanation:
  - analyze the gradients statistic. Resnets have rather uniform distribution of gradients,
  - The limit of large width (NTK) shows better learning properties than with FFNs

Depth vs Width

- Zagoruyko and Komodakis (2016): Wide Residual Networks

Width factor

depth	k	# params	CIFAR-10	CIFAR-100
40	1	0.6M	6.85	30.89
40	2	2.2M	5.33	26.04
40	4	8.9M	4.97	22.89
40	8	35.7M	4.66	-
28	10	36.5M	**4.17**	20.50
28	12	52.5M	4.33	**20.43**
22	8	17.2M	4.38	21.22
22	10	26.8M	4.44	20.75
16	8	11.0M	4.81	22.07
16	10	17.1M	4.56	21.59

Test error of depth x width combinations

- Consistent gains from increasing width
- Need to consider the resource usage again
- Can be made sparse / distilled after training and even retrained after sparsening (c.f lottery ticket hypothesis)