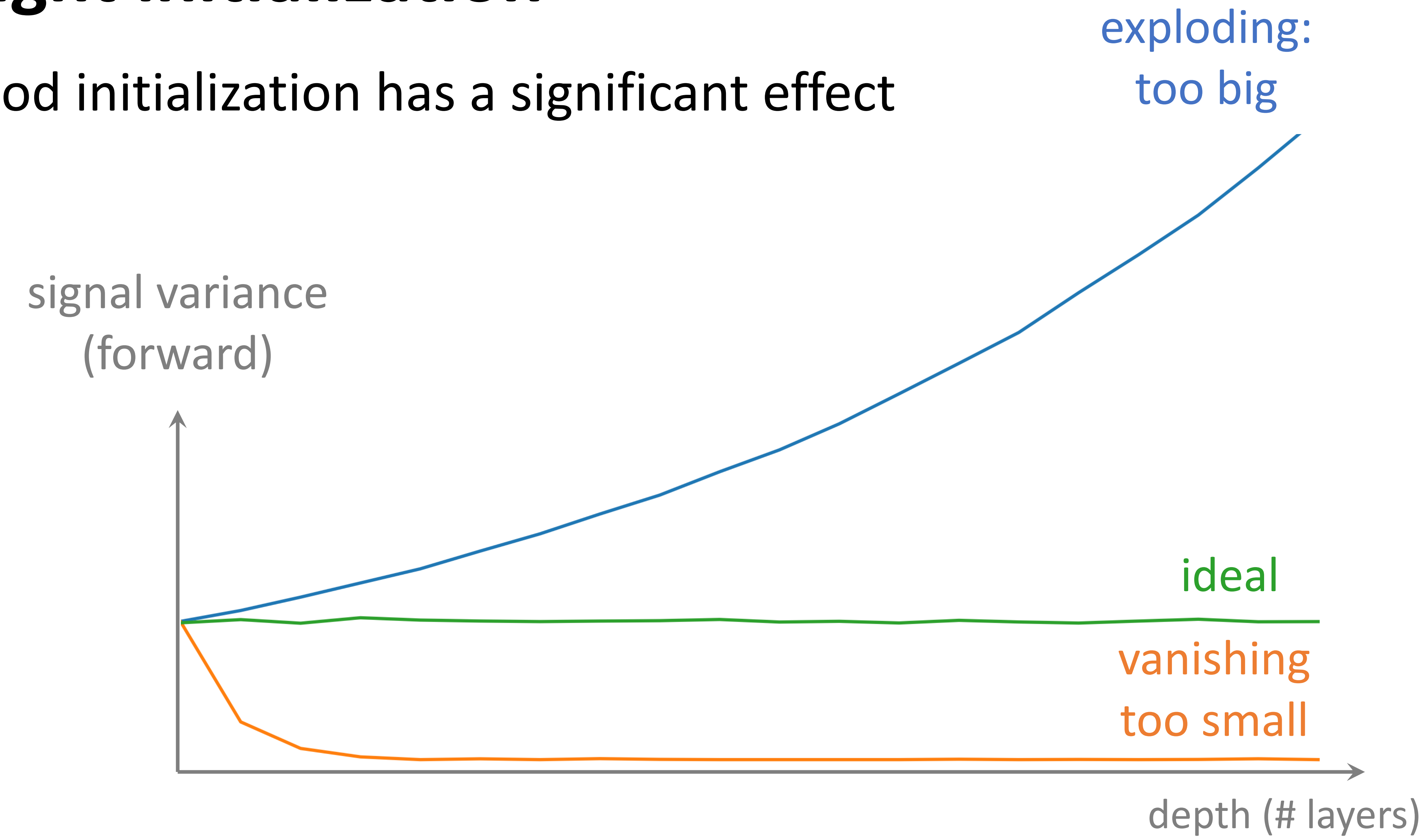
Weight Initialization

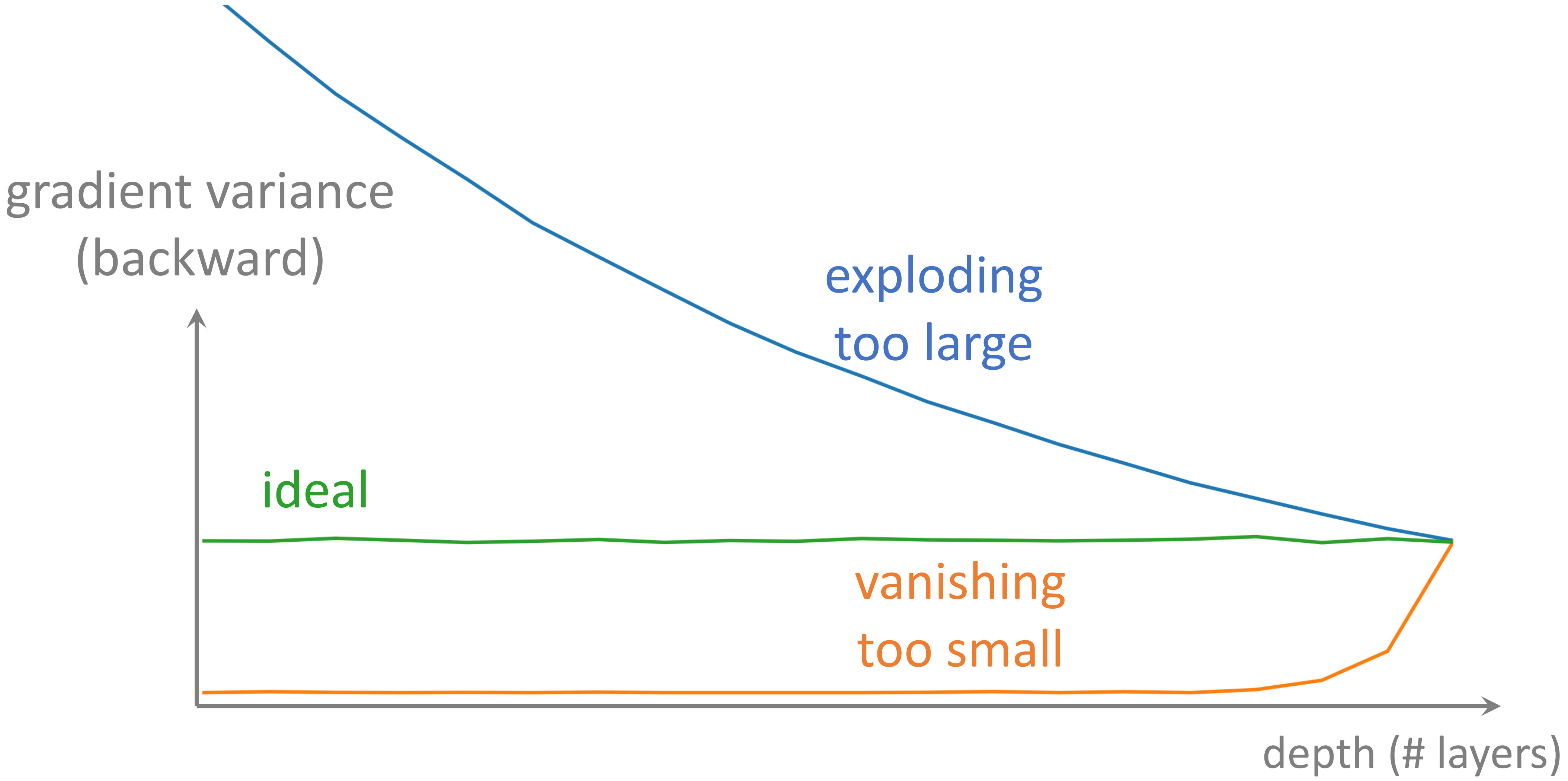
Weight initialization

- Good initialization has a significant effect



Weight initialization

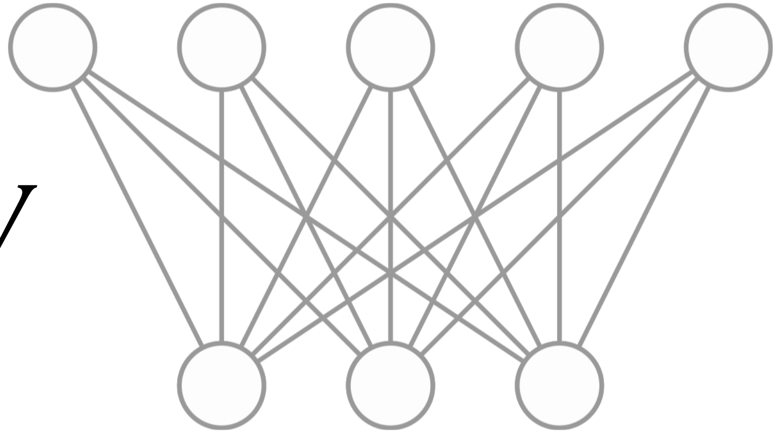
- Good initialization has a significant effect



Weight initialization

$\boldsymbol{x}$

(n neurons)



W

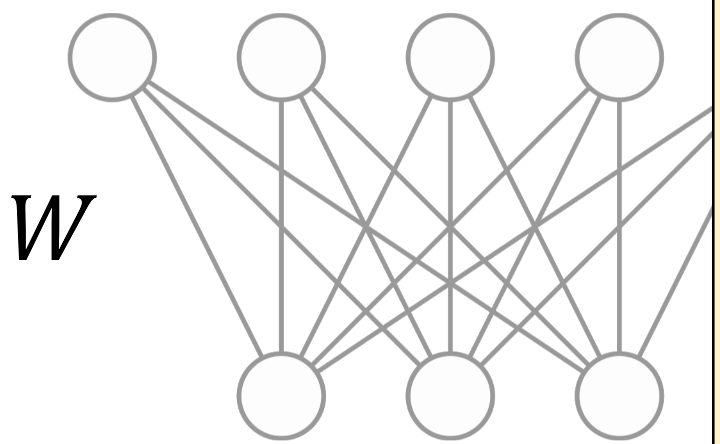
$\boldsymbol{y} = W\boldsymbol{x}$

(n' neurons)

- if linear activation
- one layer: variance scaled by

$$\mathrm{Var}[y] = n\mathrm{Var}[w]\mathrm{Var}[x]$$

Weight initiali



derivation

$$y_i = \sum_j W_{ij} x_j$$

- definition

$$\mathrm{Var}[y_i] = \mathrm{Var}[\sum_j W_{ij} x_j]$$

$$\mathrm{Var}[y_i] = \sum_j \mathrm{Var}[W_{ij} x_j]$$

- independence

$$\mathrm{Var}[y_i] = \sum_j \mathrm{Var}[W_{ij}]\,\mathrm{Var}[x_j]$$

- independence & zero-mean

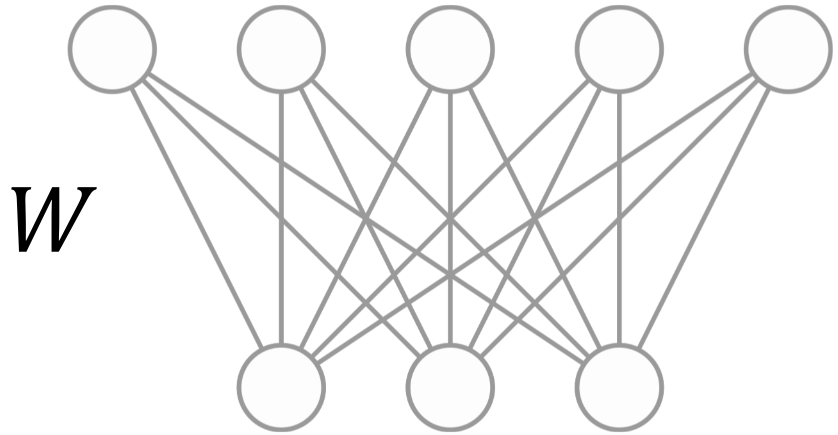
$$\mathrm{Var}[y] = n\mathrm{Var}[w]\mathrm{Var}[x]$$

- identical distributions

Weight initialization

$\boldsymbol{x}$

(n neurons)



$\boldsymbol{y} = W\boldsymbol{x}$

(n' neurons)

- if linear activation
- one layer: variance scaled by

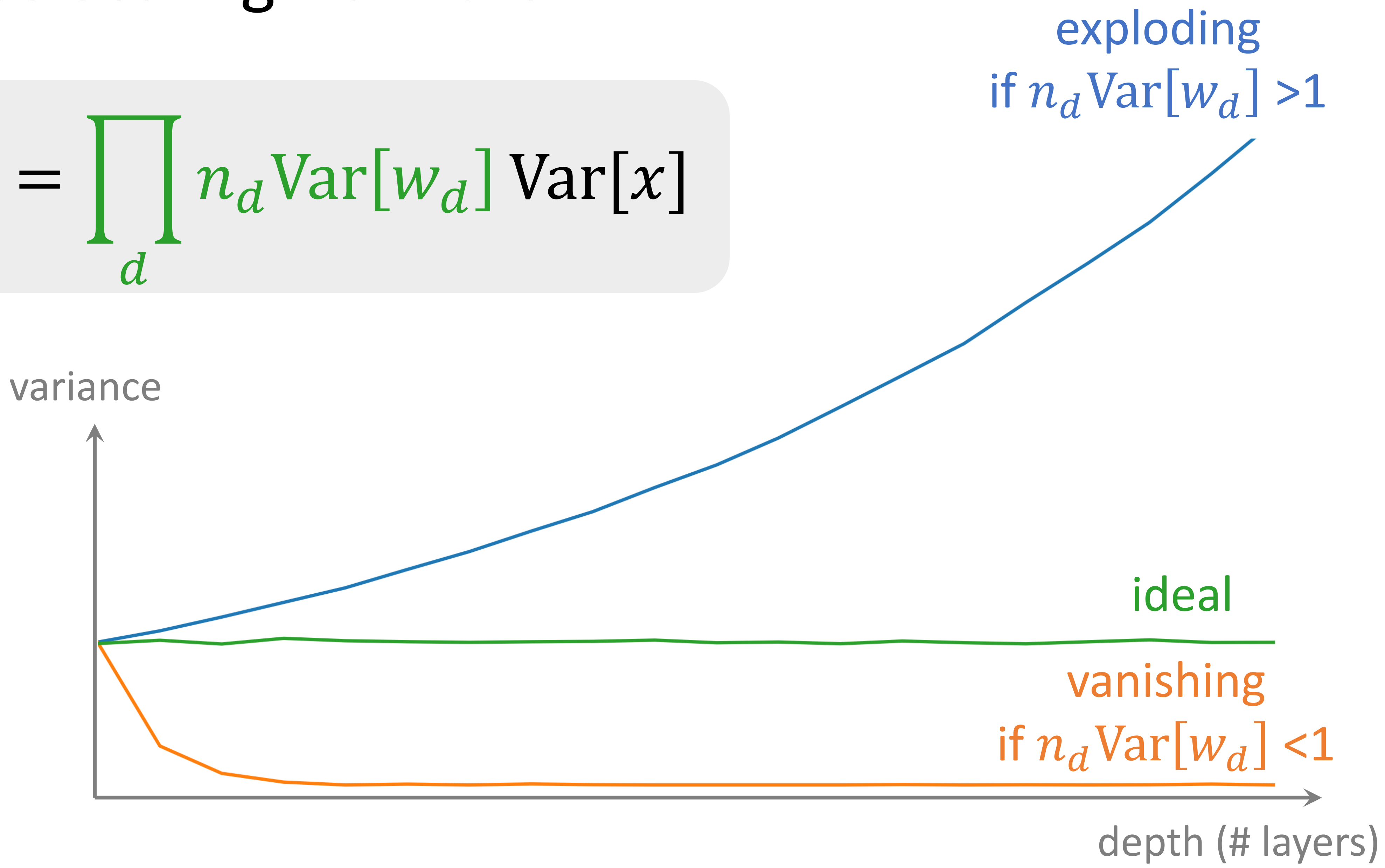
$$\mathrm{Var}[y] = n\mathrm{Var}[w]\mathrm{Var}[x]$$

- many layers: variance scaled by

$$\mathrm{Var}[y] = \prod_d n_d \mathrm{Var}[w_d]\,\mathrm{Var}[x]$$

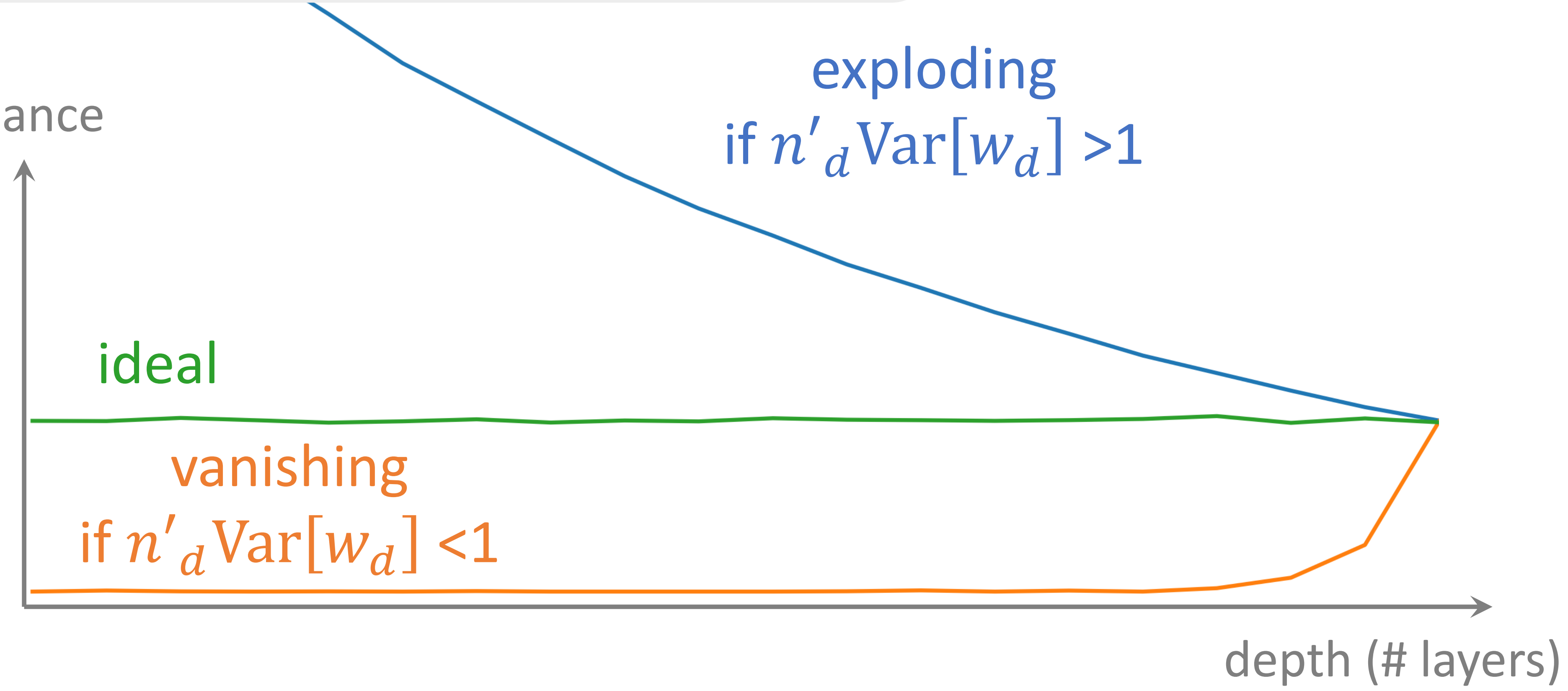
Variance Scaling: forward

$$\mathrm{Var}[y] = \prod_d n_d \mathrm{Var}[w_d]\,\mathrm{Var}[x]$$



Variance Scaling: backward

$$\mathrm{Var}\left[\frac{\partial \mathcal{E}}{\partial x}\right] = \prod_d n'_d \mathrm{Var}[w_d]\ \mathrm{Var}[\frac{\partial \mathcal{E}}{\partial y}]$$



Xavier initialization: torch.nn.init.xavier_normal_

forward:

$$n\mathrm{Var}[w] = 1$$

backward:

$$n'\mathrm{Var}[w] = 1$$

- Gaussian distribution:

$$w \sim \mathcal{N}(\mu = 0, \sigma = \sqrt{1/n})$$

- Uniform distribution:

$$w \sim \mathcal{U}(-a, +a), a = \sqrt{3/n}$$

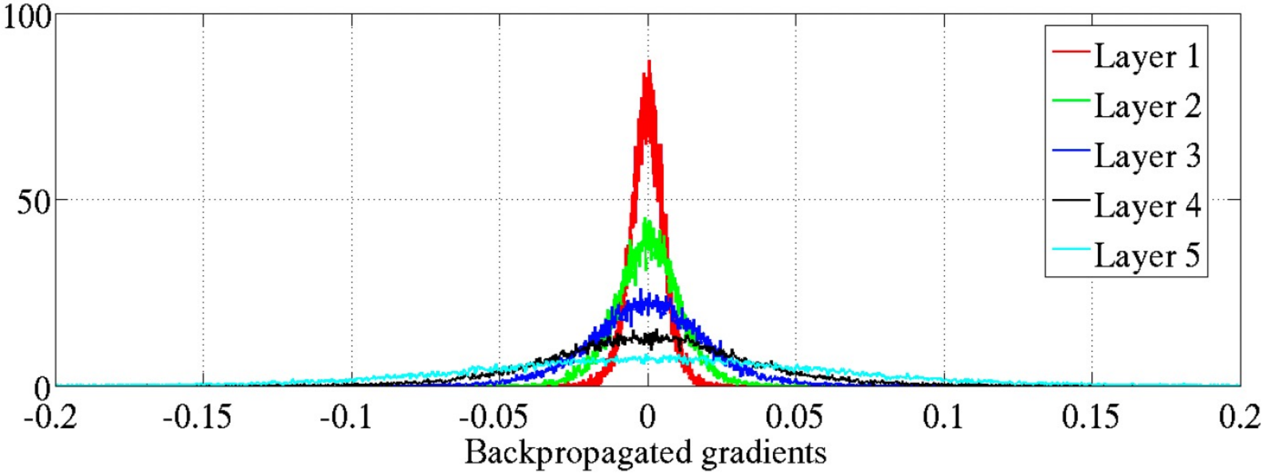
- Consider forward and backward:

  replace n with $(n + n')/2$

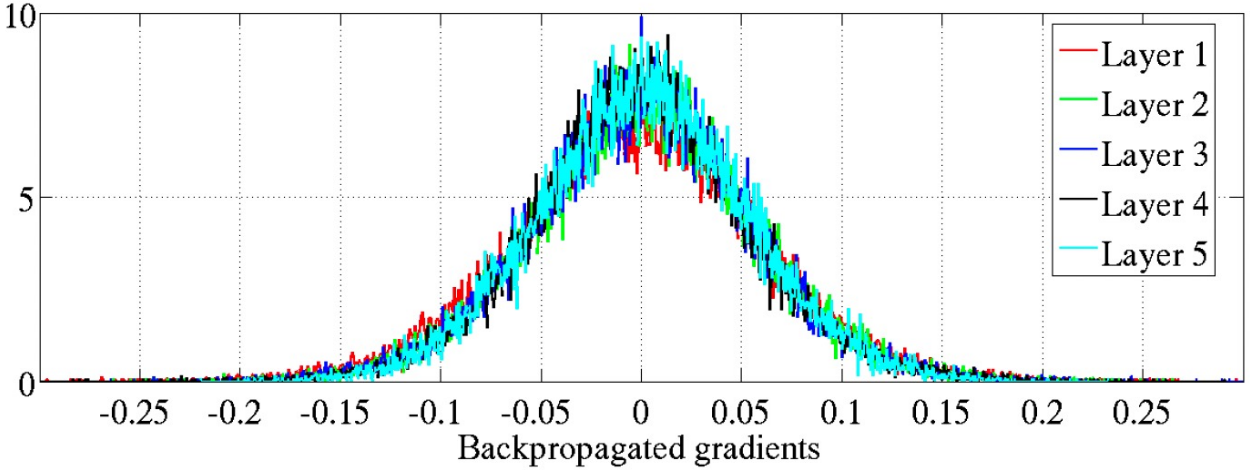
LeCun et al 1998 “Efficient Backprop”

Glorot & Bengio 2010 “Understanding the difficulty of training deep feedforward neural networks”

Xavier initialization: torch.nn.init.xavier_normal_



poor initialization:
earlier layer has smaller gradients



Xavier initialization:
all layers have similar gradient scale

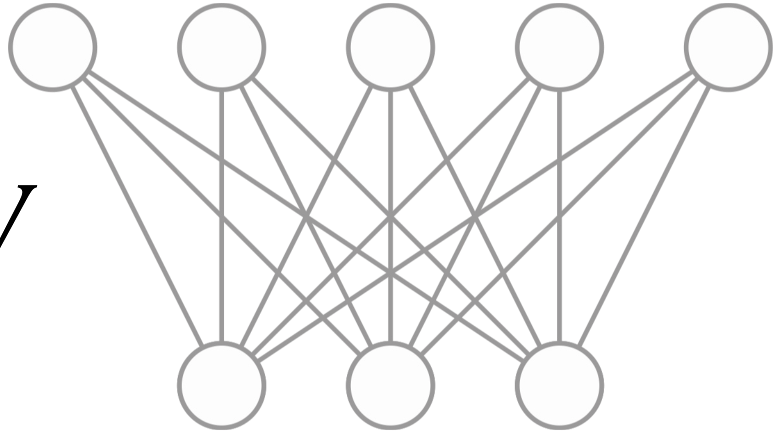
Glorot & Bengio 2010 “Understanding the difficulty of training deep feedforward neural networks”

Weight initialization: ReLU

$\boldsymbol{x}' = \mathrm{ReLU}(\boldsymbol{x})$

(n neurons)

W



$\boldsymbol{y} = W\boldsymbol{x}'$

(n' neurons)

- if ReLU activation
- one layer: variance scaled by

$$\mathrm{Var}[y] = \frac{1}{2} n\mathrm{Var}[w]\mathrm{Var}[x]$$

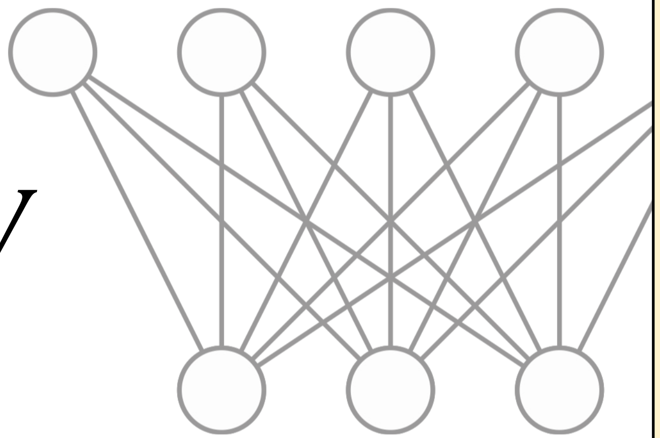
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. ICCV 2015.

Weight initiali

$\boldsymbol{x}' = \mathrm{ReLU}(\boldsymbol{x})$

(n neurons)

W



$\boldsymbol{y} = W\boldsymbol{x}'$

(n' neurons)

derivation

$$y_i = \sum_j W_{ij} x'_j$$

- $x' = \mathrm{ReLU}(x)$

$$\mathrm{Var}[y_i] = \mathrm{Var}[\sum_j W_{ij} x'_j]$$

$$\mathrm{Var}[y_i] = \sum_j \mathrm{Var}[W_{ij} x'_j]$$

$$\mathrm{Var}[y_i] = \sum_j \mathrm{Var}[W_{ij}]\, \mathrm{E}[x'^2_j]$$

$\mathrm{Var}[wx']$
$= E[(wx')^2] - (E[wx'])^2$
$= E[(wx')^2] - (E[w]E[x])^2$
$= E[w^2]E[x'^2] = Var[w]E[x'^2]$
if w is zero-mean but x' is not.

$$\mathrm{Var}[y] = n\mathrm{Var}[w]\frac{1}{2}\mathrm{Var}[x]$$

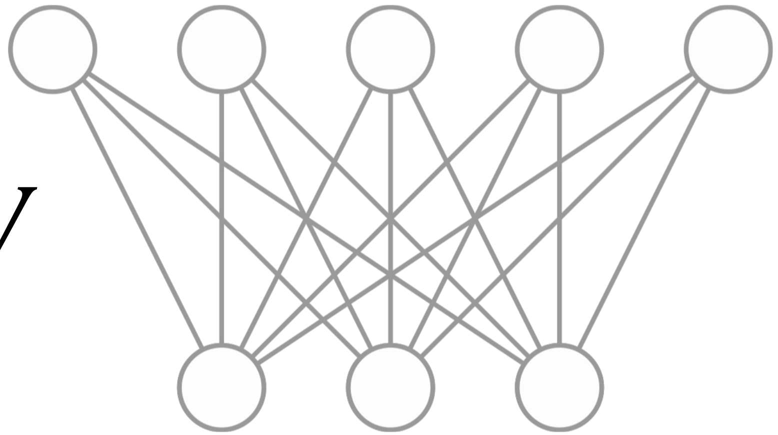
$\mathrm{E}[x'^2] = \mathrm{Var}[x]/2$
if x is zero-mean and symmetric

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". ICCV 2015.

Weight initialization: ReLU

$\boldsymbol{x}' = \mathrm{ReLU}(\boldsymbol{x})$
(n neurons)

W



$\boldsymbol{y} = W\boldsymbol{x}'$
(n' neurons)

- if ReLU activation
- one layer: variance scaled by

$$\mathrm{Var}[y] = \frac{1}{2} n \mathrm{Var}[w] \mathrm{Var}[x]$$

- many layers: variance scaled by

$$\mathrm{Var}[y] = \prod_d \frac{1}{2} n_d \mathrm{Var}[w_d] \, \mathrm{Var}[x]$$

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". ICCV 2015.

Kaiming initialization: torch.nn.init.kaiming_normal_

forward:

$$\frac{1}{2} n \mathrm{Var}[w] = 1$$

backward:

$$\frac{1}{2} n' \mathrm{Var}[w] = 1$$

- Gaussian distribution:

$$w \sim \mathcal{N}(\mu = 0, \sigma = \sqrt{2/n})$$

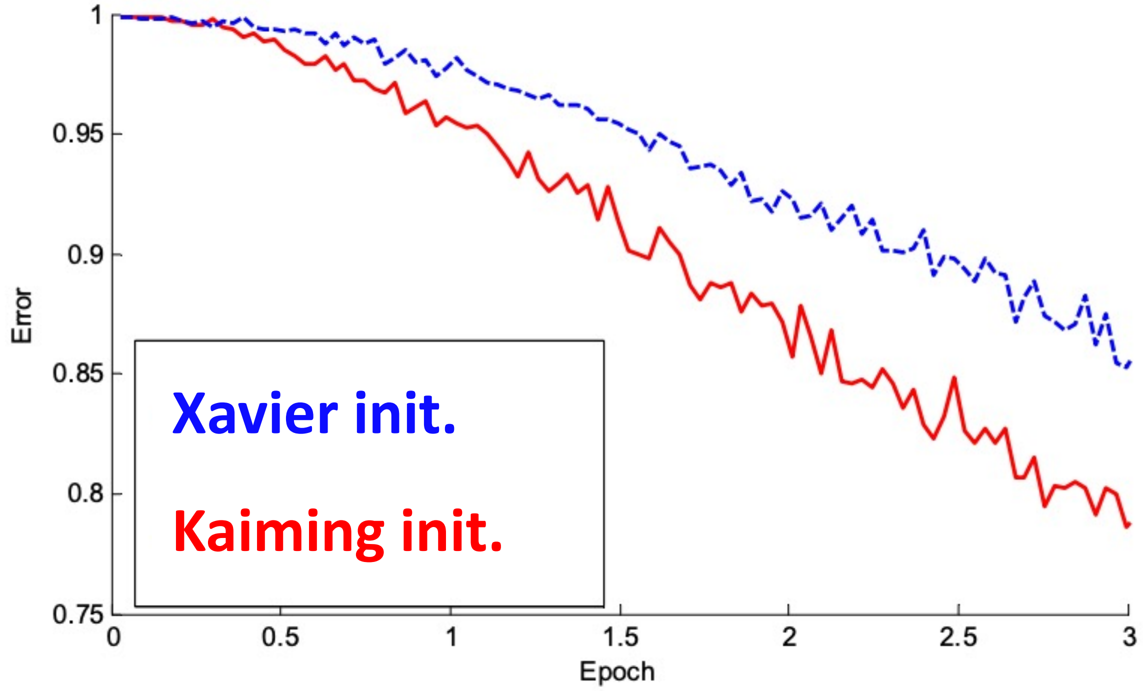
- Uniform distribution:

$$w \sim \mathcal{U}(-a, +a), a = \sqrt{6/n}$$

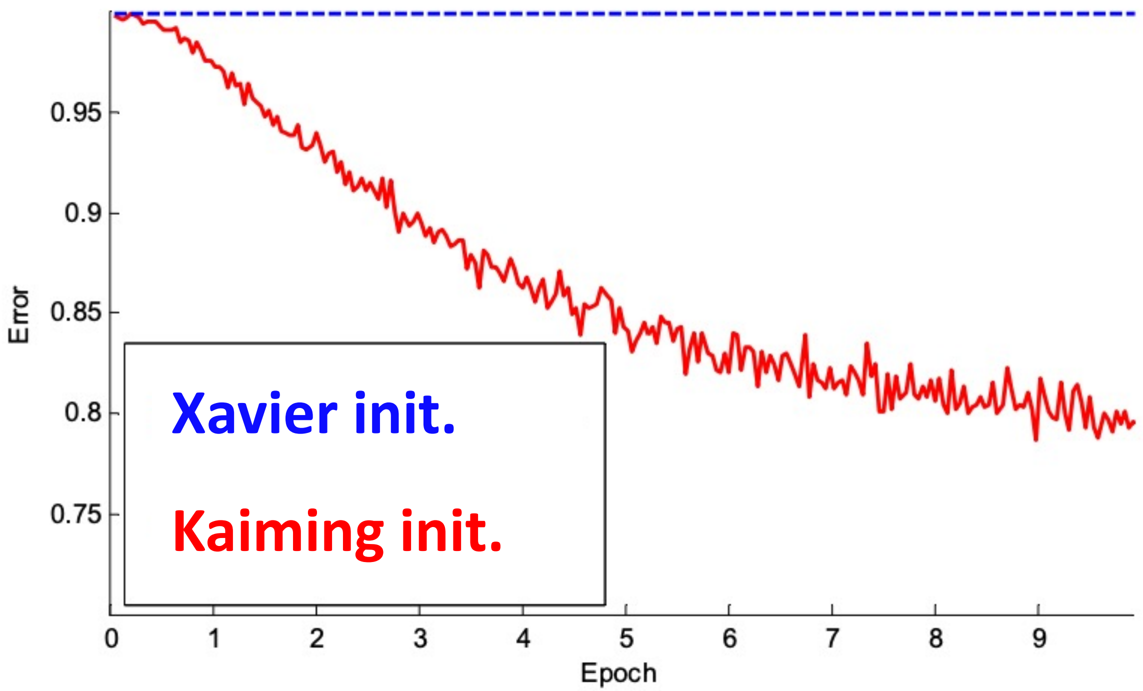
- sufficient to use n or n'

LeCun et al 1998 “Efficient Backprop”
Glorot & Bengio 2010 “Understanding the difficulty of training deep feedforward neural networks”

Kaiming initialization: torch.nn.init.kaiming_normal_



22-layer VGG w/ ReLU :
better init converges faster



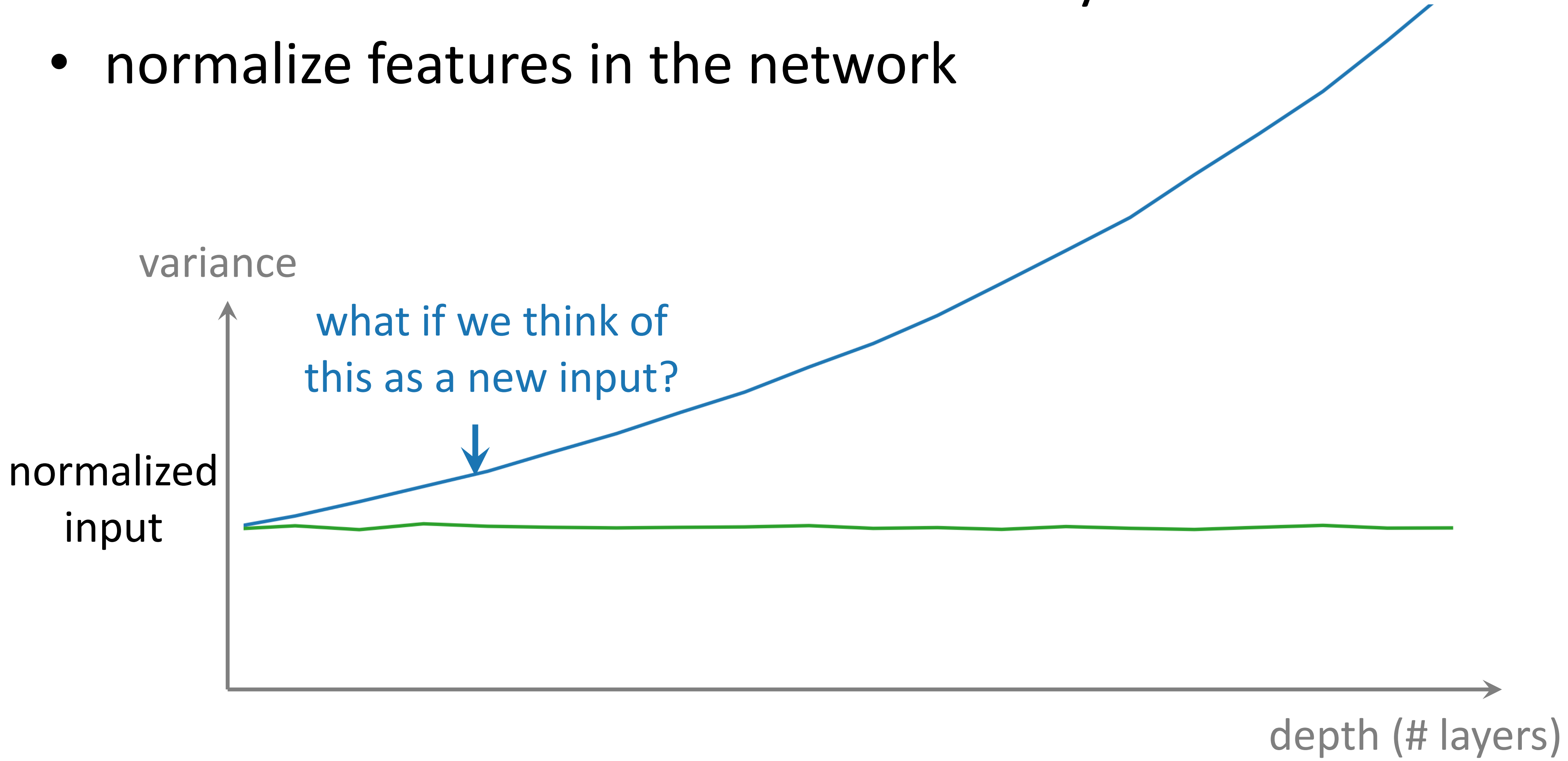
30-layer VGG w/ ReLU:
better init enables training

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". ICCV 2015.

Normalization Modules

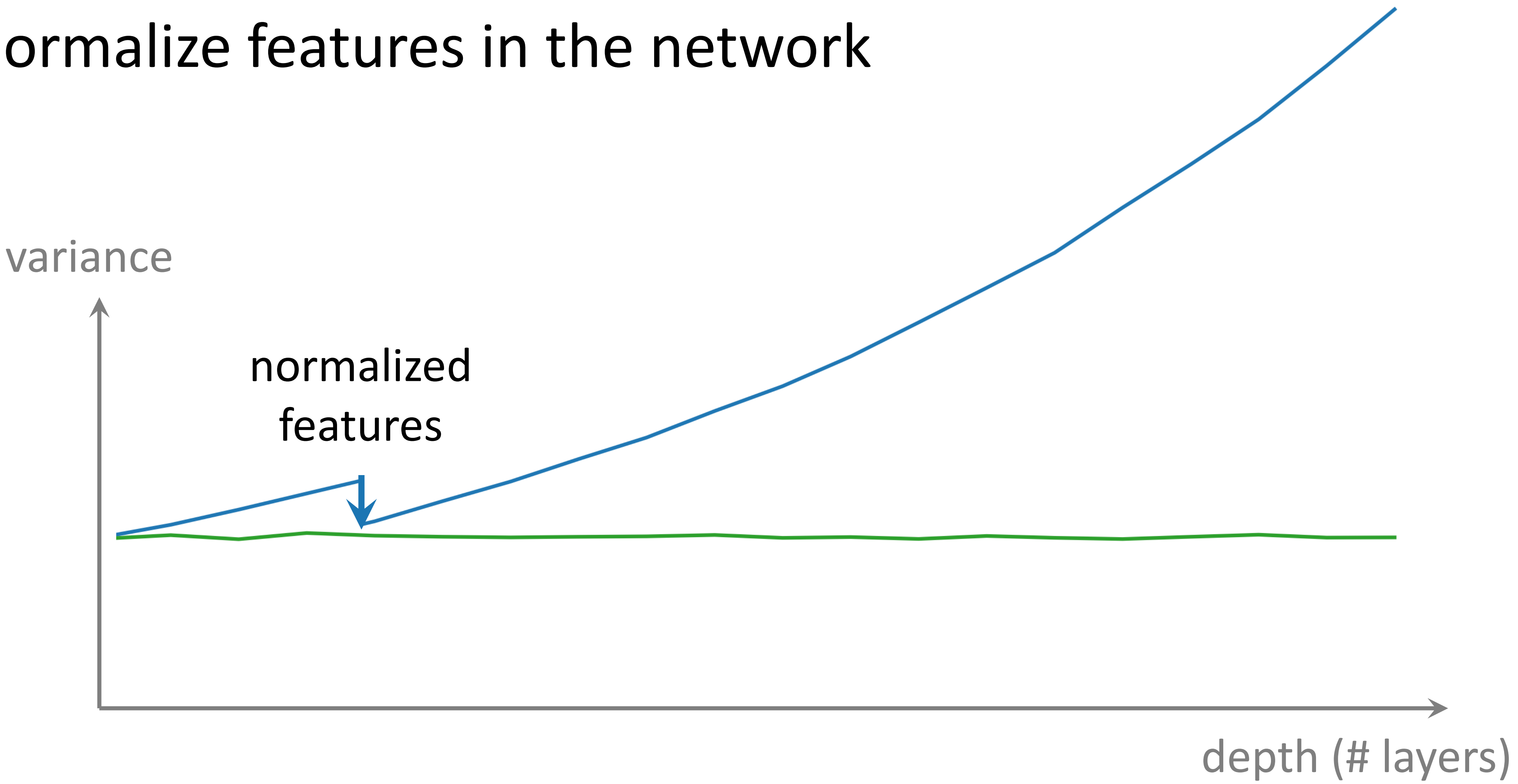
Normalization Modules

- We want to maintain variance for all layers
- normalize features in the network



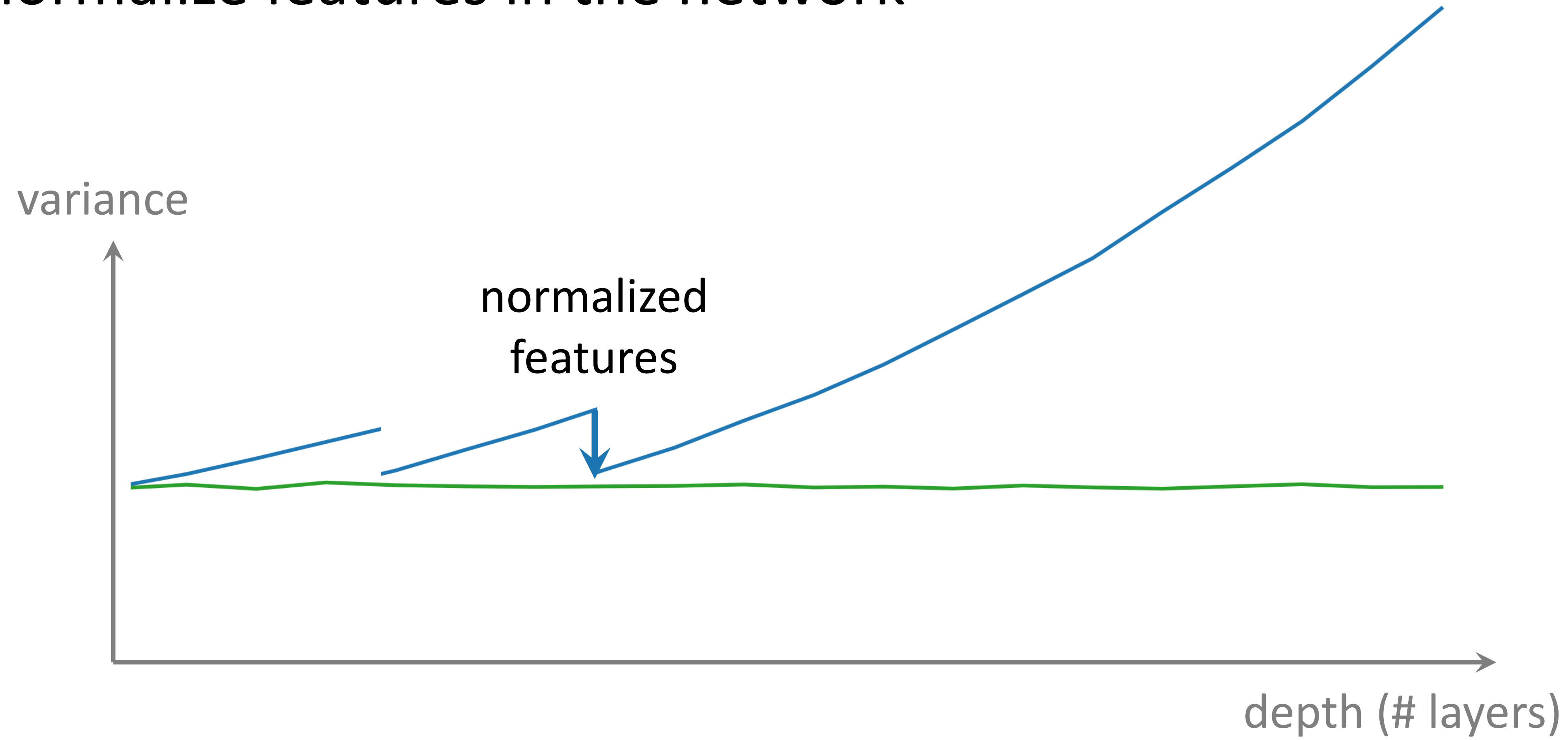
Normalization Modules

- We want to maintain variance for all layers
- normalize features in the network



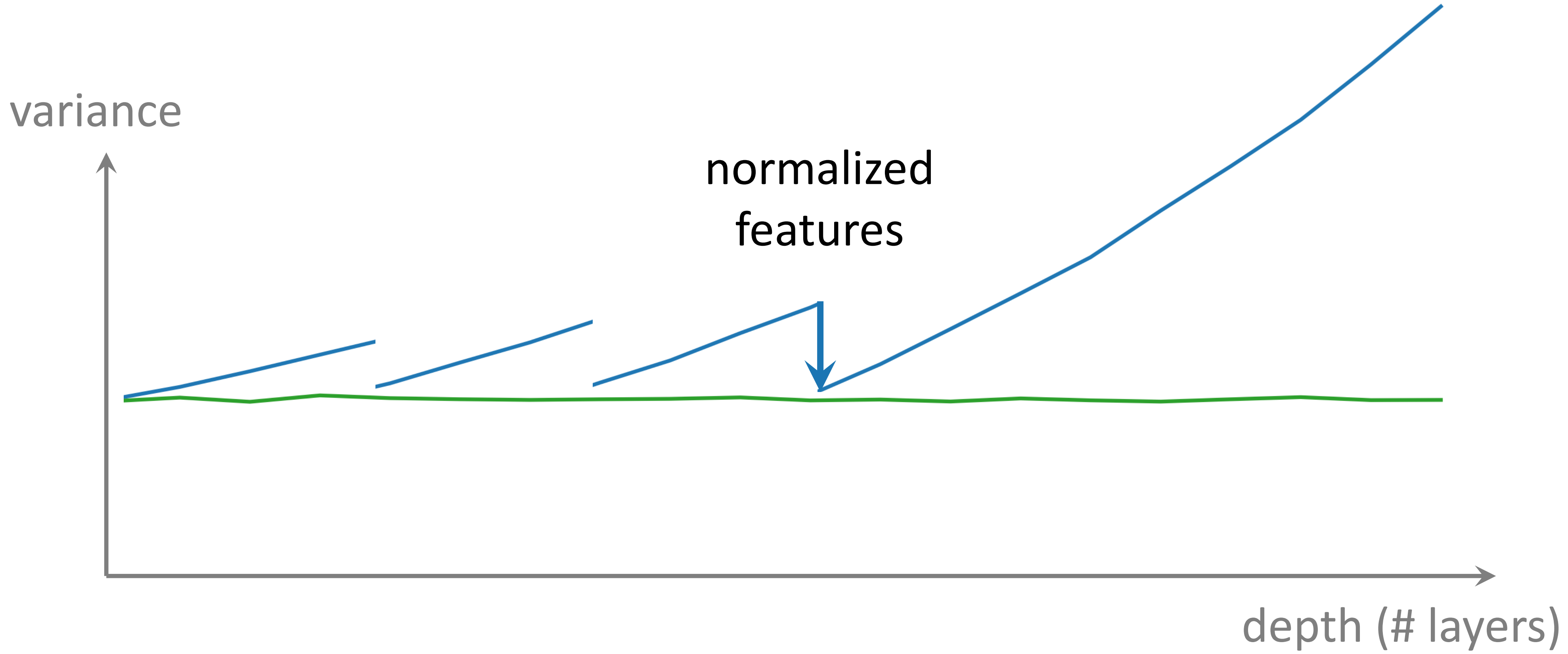
Normalization Modules

- We want to maintain variance for all layers
- normalize features in the network



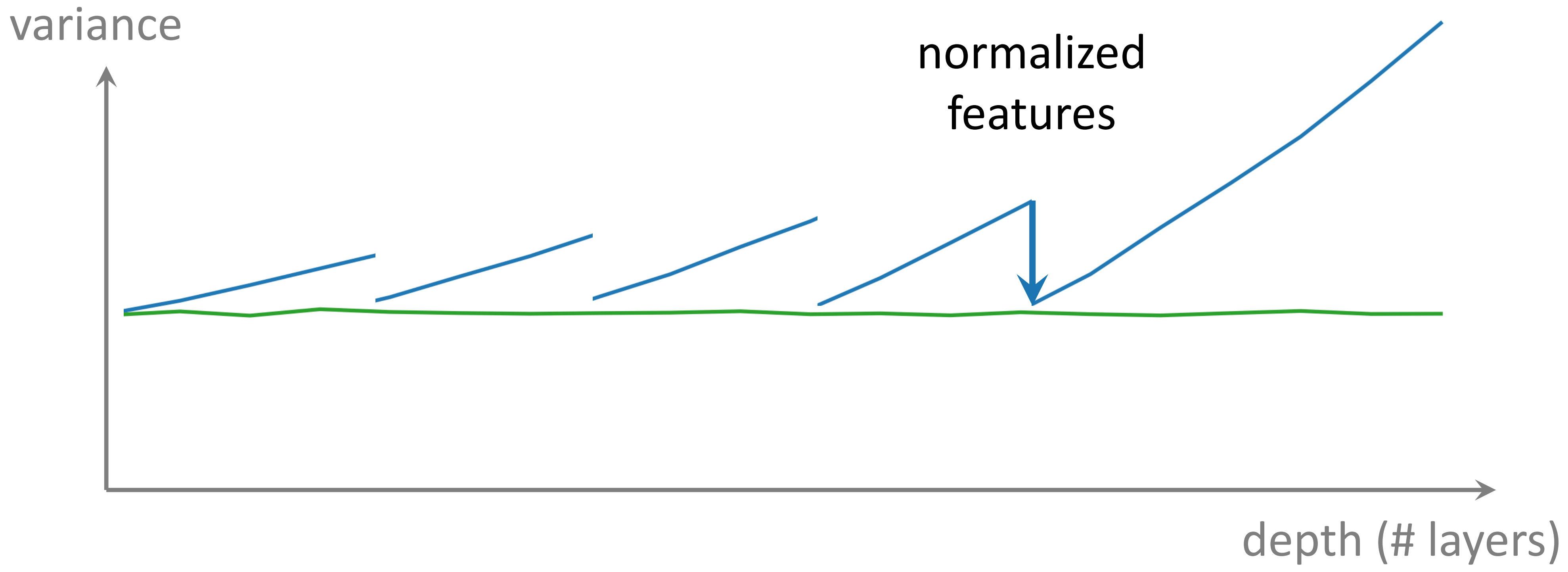
Normalization Modules

- We want to maintain variance for all layers
- normalize features in the network



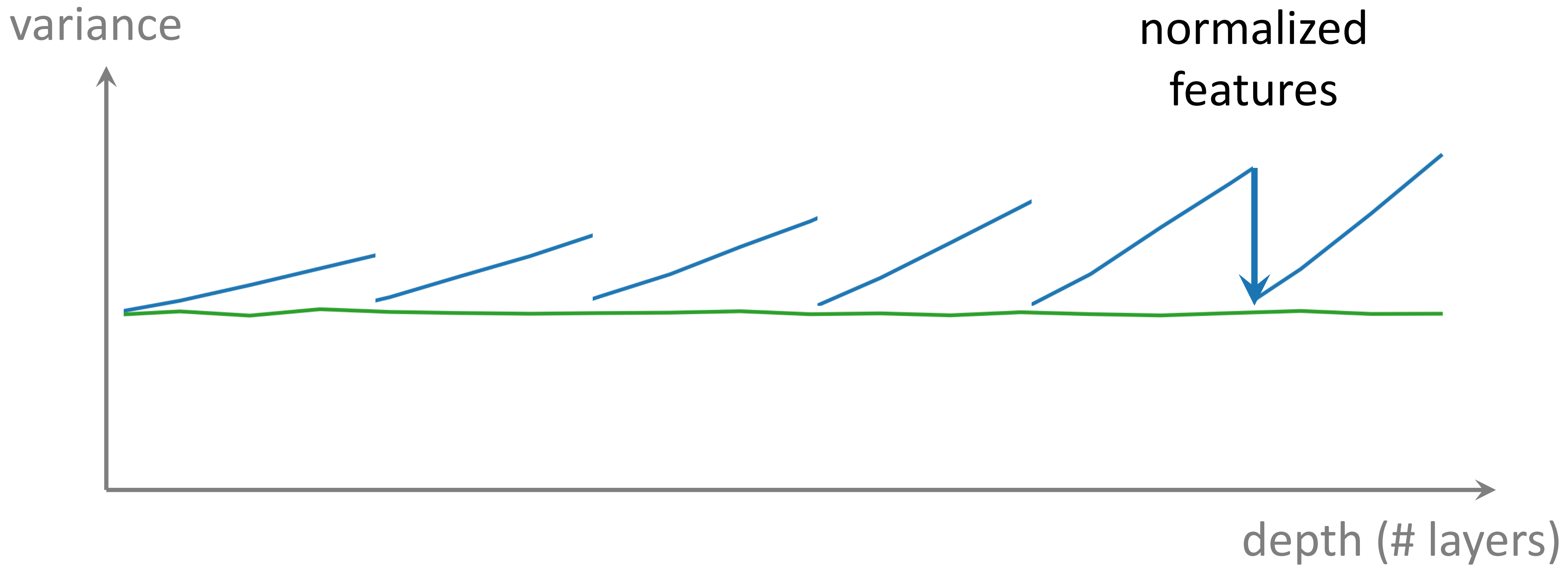
Normalization Modules

- We want to maintain variance for all layers
- normalize features in the network



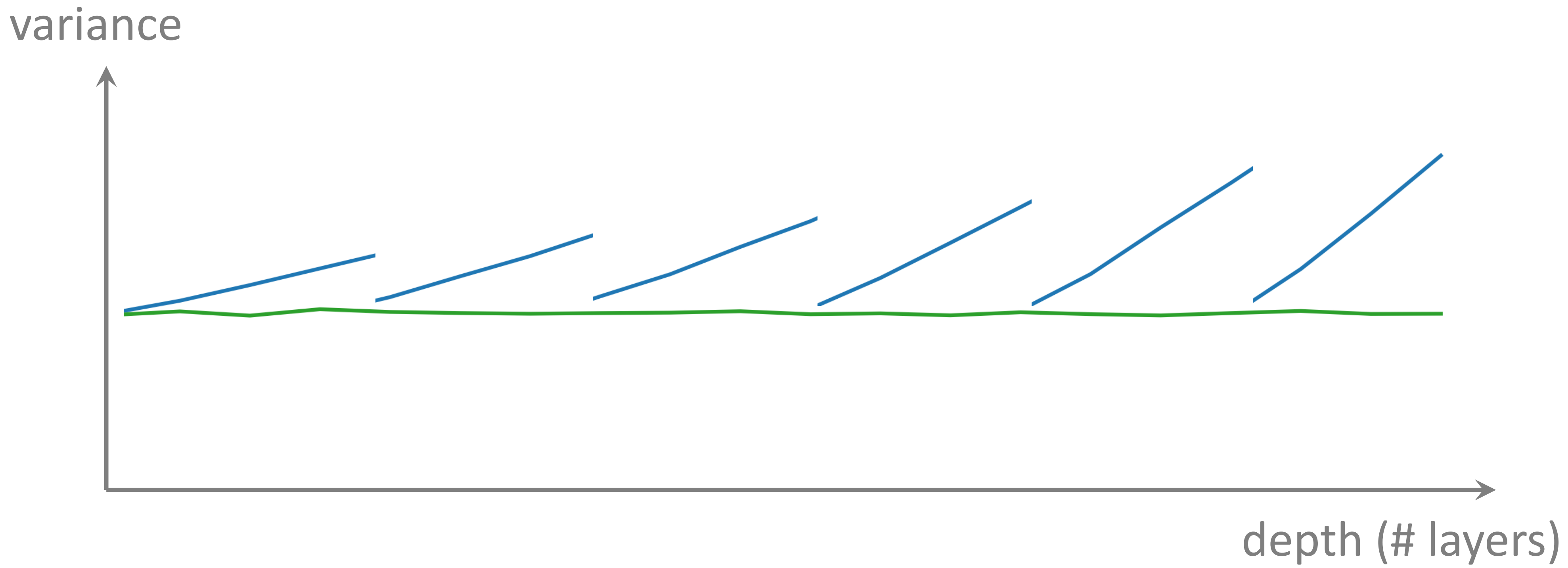
Normalization Modules

- We want to maintain variance for all layers
- normalize features in the network



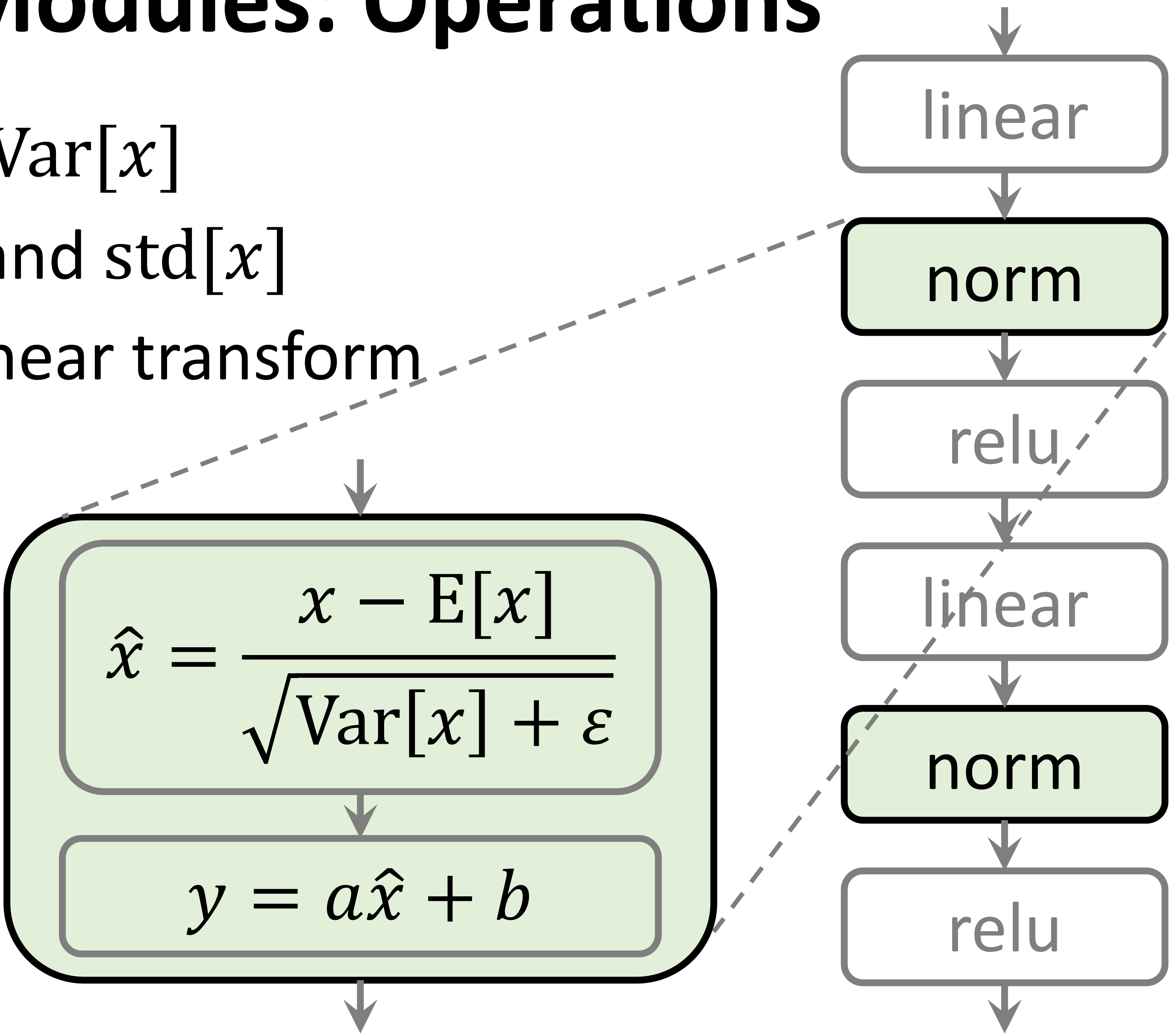
Normalization Modules

- We want to maintain variance for all layers
- normalize features in the network
- train end-to-end by BackProp



Normalization Modules: Operations

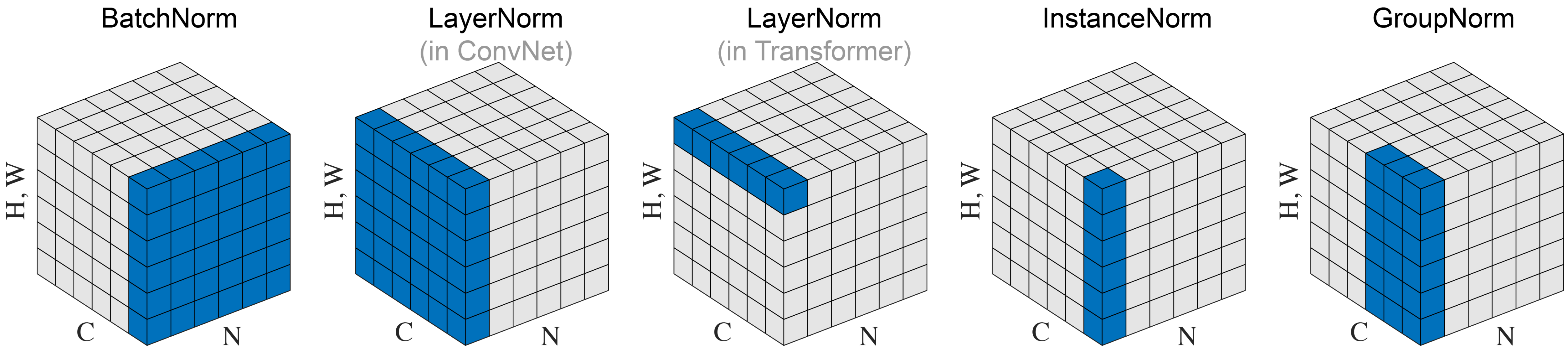
1. compute $\mathrm{E}[x]$ and $\mathrm{Var}[x]$
2. normalize by $\mathrm{E}[x]$ and $\mathrm{std}[x]$
3. compensate by a linear transform



Ioffe & Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. ICML 2015

Normalization Modules: Variants

differ in support sets of $\mathrm{E}[x], \mathrm{Var}[x]$



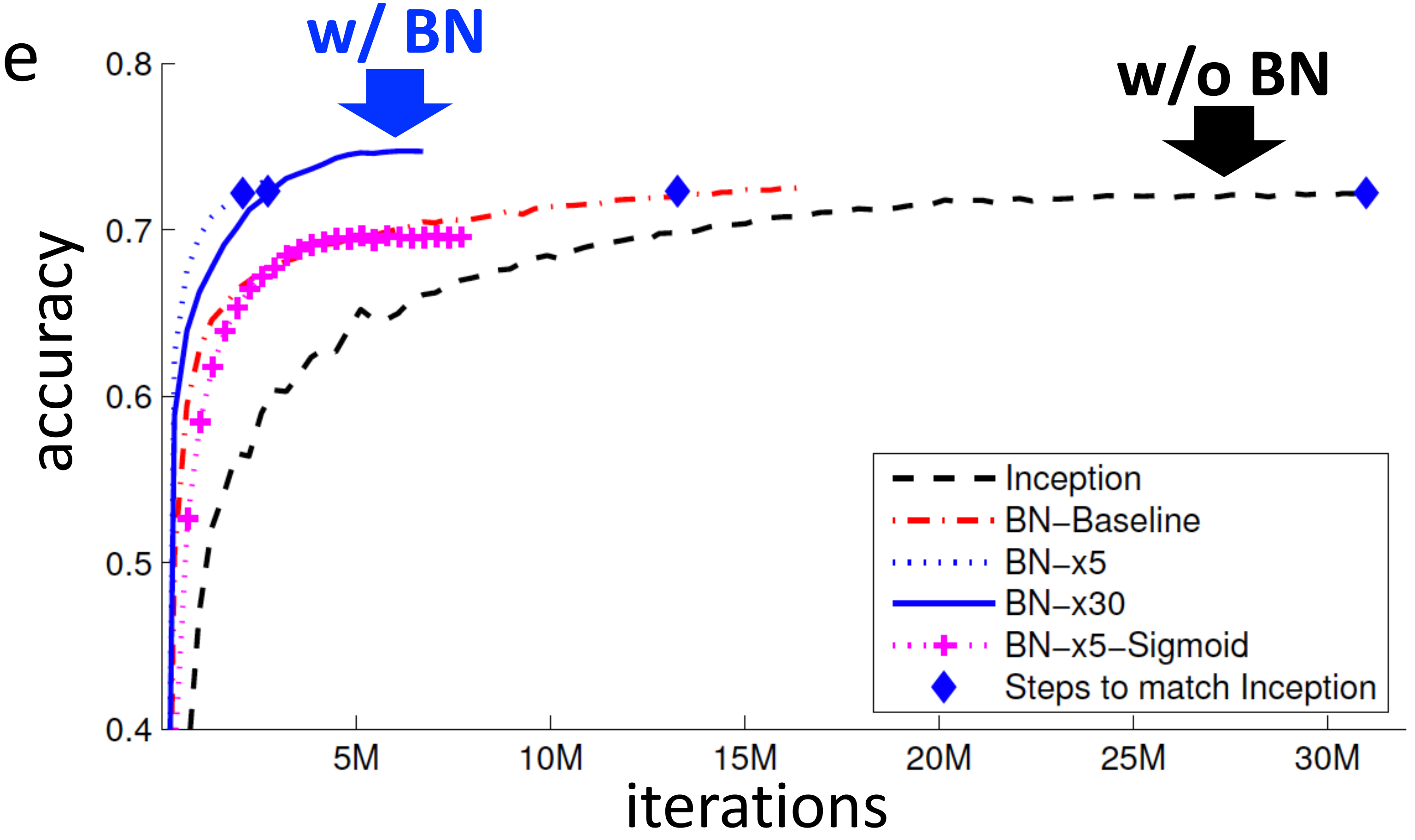
Ioffe & Szegedy, 2015; Ba, Kiros, Hinton, 2016; Ulyanov, Vedaldi, Lempitsky, 2016; Wu & He, 2018

Normalization Modules: Effects

- **Enable training models that are otherwise not trainable**

- Speed up convergence

- Improve accuracy



Ioffe & Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". ICML 2015