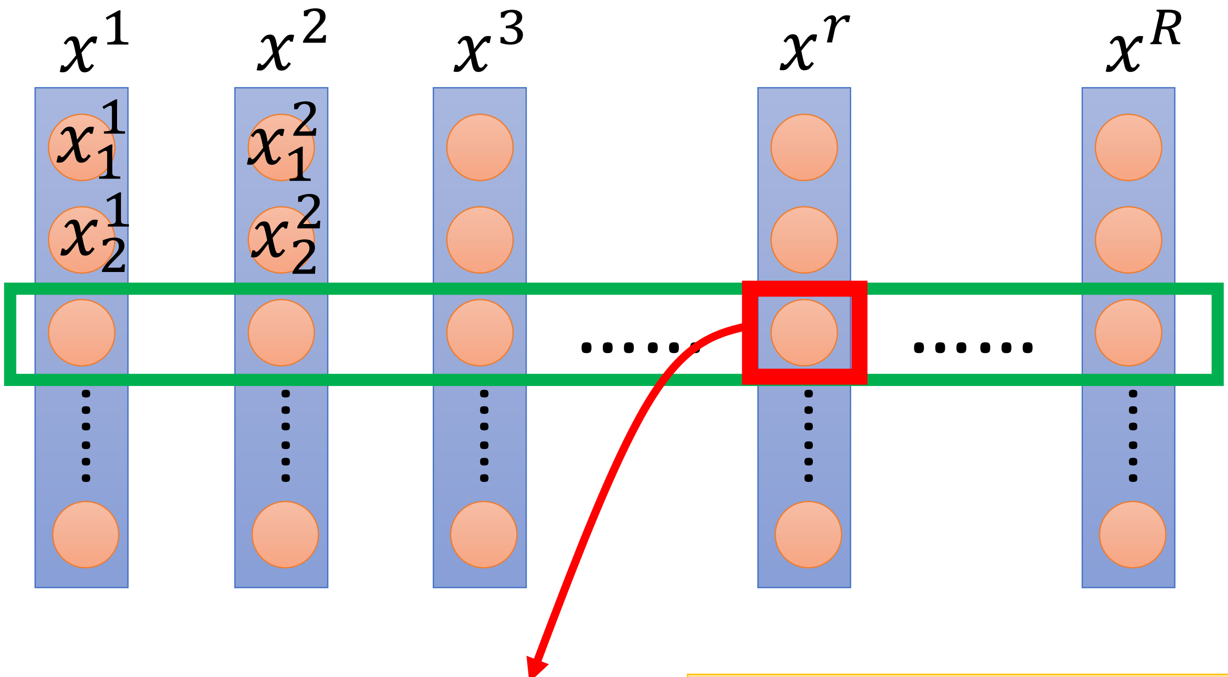
Tips for Training Deep Network

Output

- Training Strategy: Batch Normalization
- Activation Function: SELU
- Network Structure: Highway Network

Batch Normalization

Feature Scaling



For each dimension i:

mean: m_i

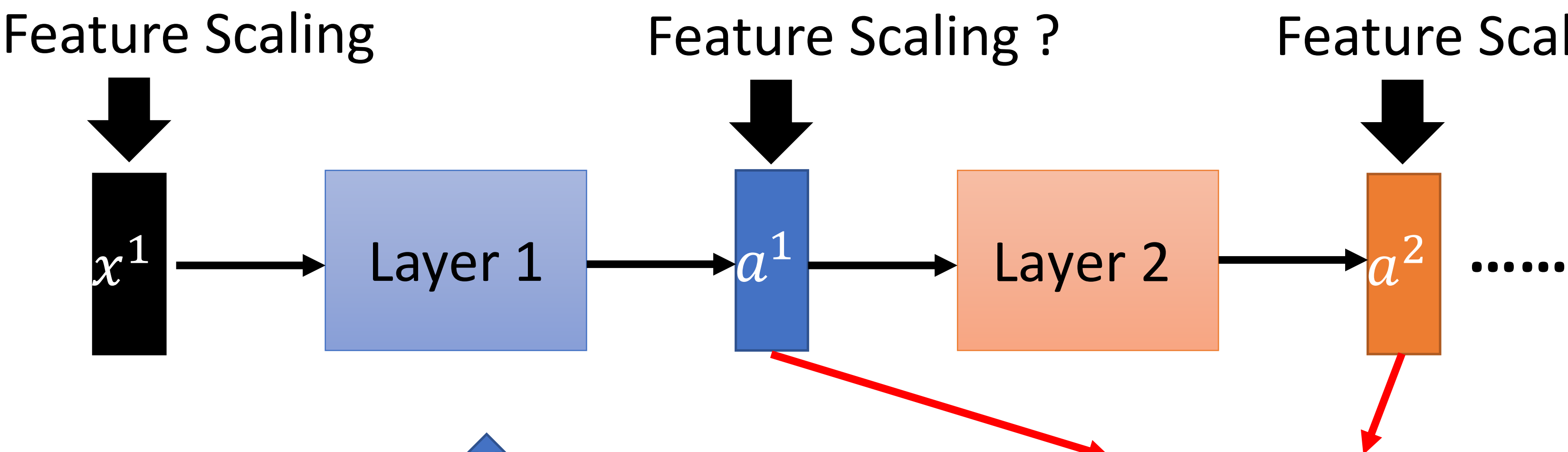
standard deviation: σ_i

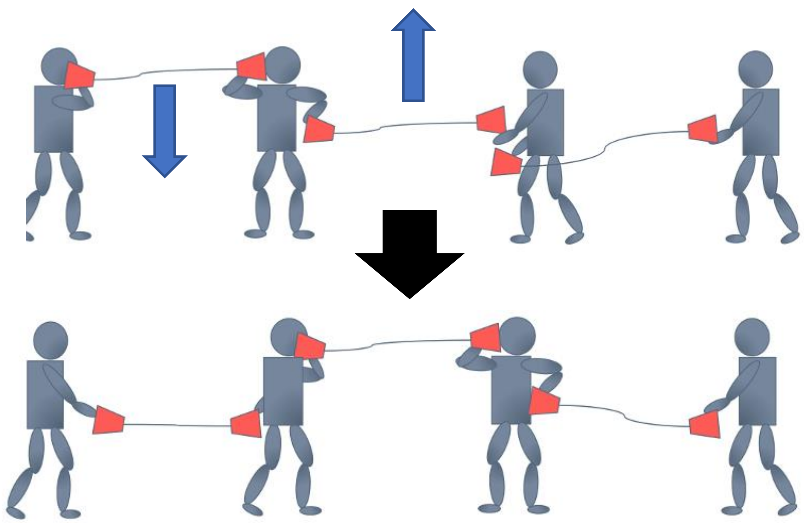
$$x_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$$

The means of all dimensions are 0, and the variances are all 1

In general, gradient descent converges much faster with feature scaling than without it.

How about Hidden Layer?





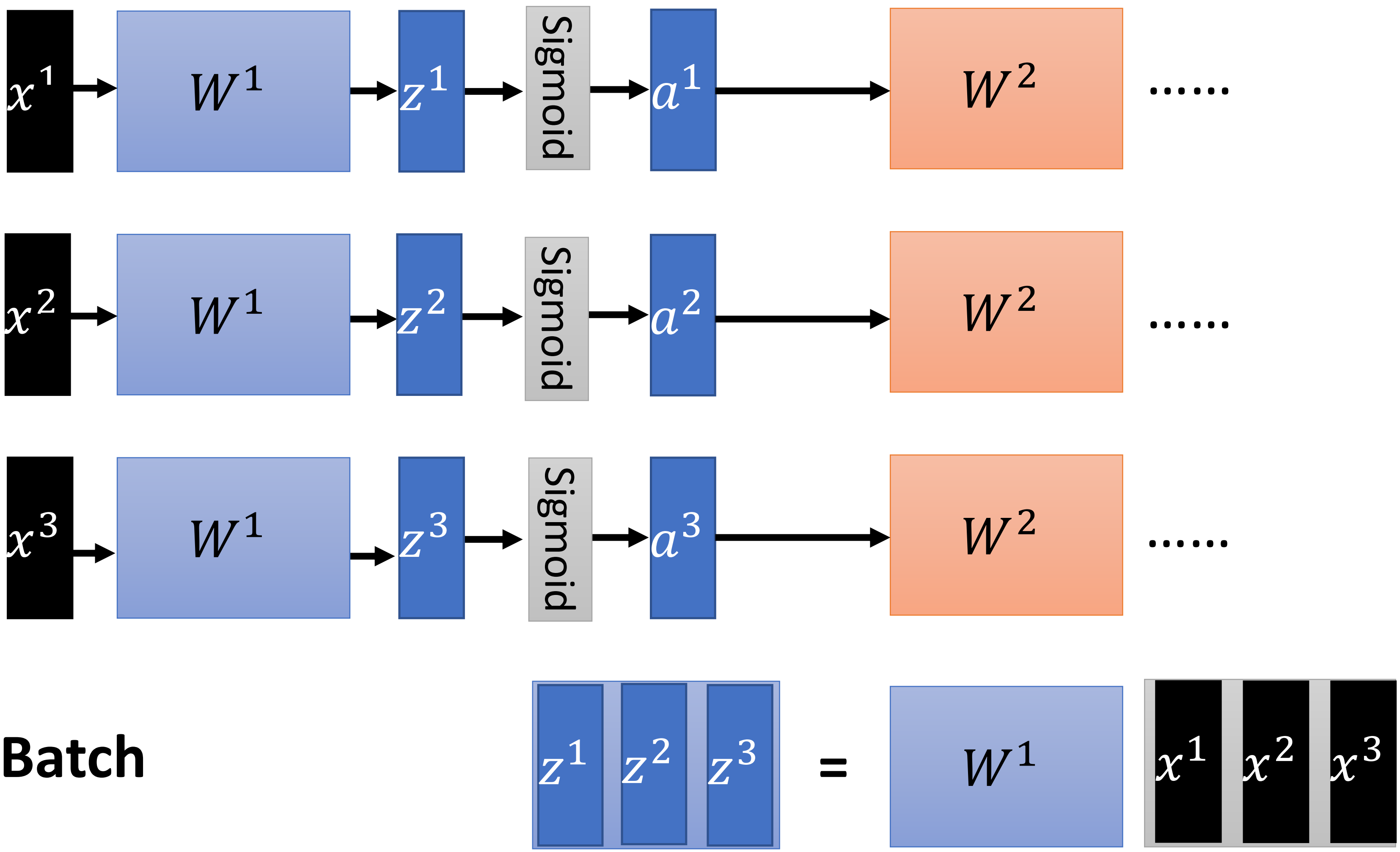
Internal Covariate Shift

Difficulty: their statistics change during the training …

Batch normalization

Smaller learning rate can be helpful, but the training would be slower.

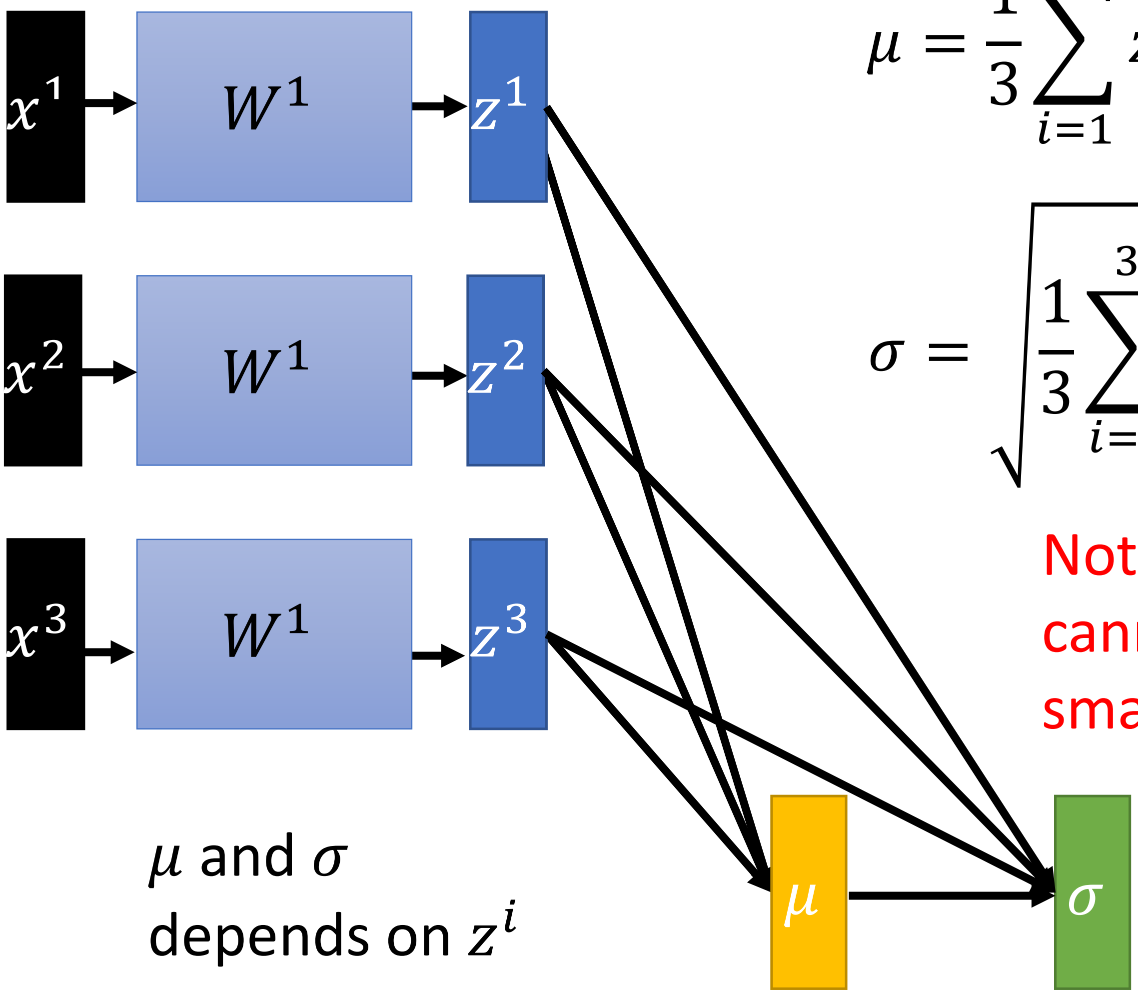
Batch



x^1
W^1
z^1
Sigmoid
a^1
W^2
……
x^2
W^1
z^2
Sigmoid
a^2
W^2
……
x^3
W^1
z^3
Sigmoid
a^3
W^2
……
z^1 z^2 z^3 = W^1 x^1 x^2 x^3

Batch

Batch normalization



$$\mu = \frac{1}{3}\sum_{i=1}^{3} z^i$$

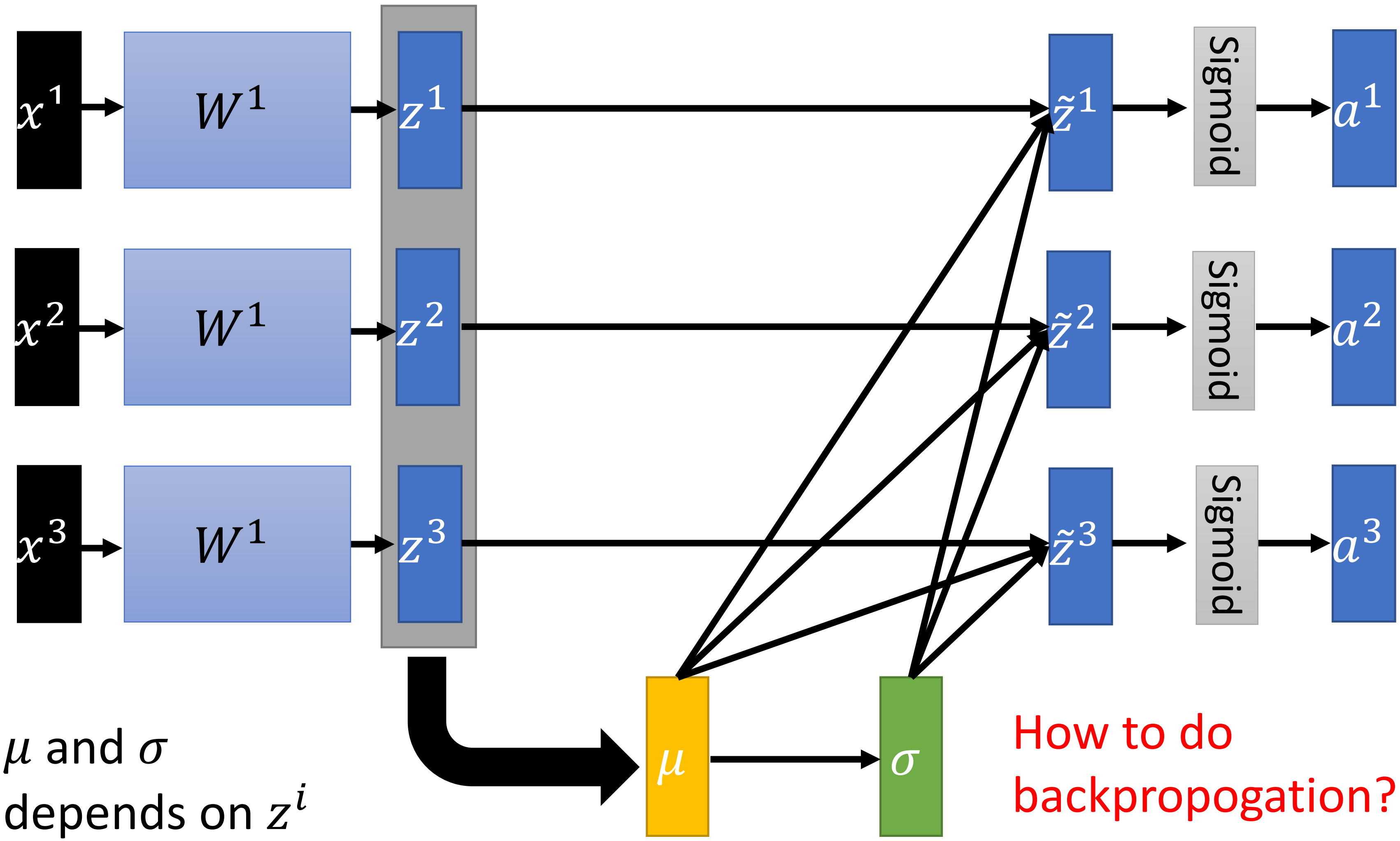
$$\sigma = \sqrt{\frac{1}{3}\sum_{i=1}^{3}\left(z^i - \mu\right)^2}$$

Note: Batch normalization cannot be applied on small batch.

μ and σ depends on z^i

Batch normalization

$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$



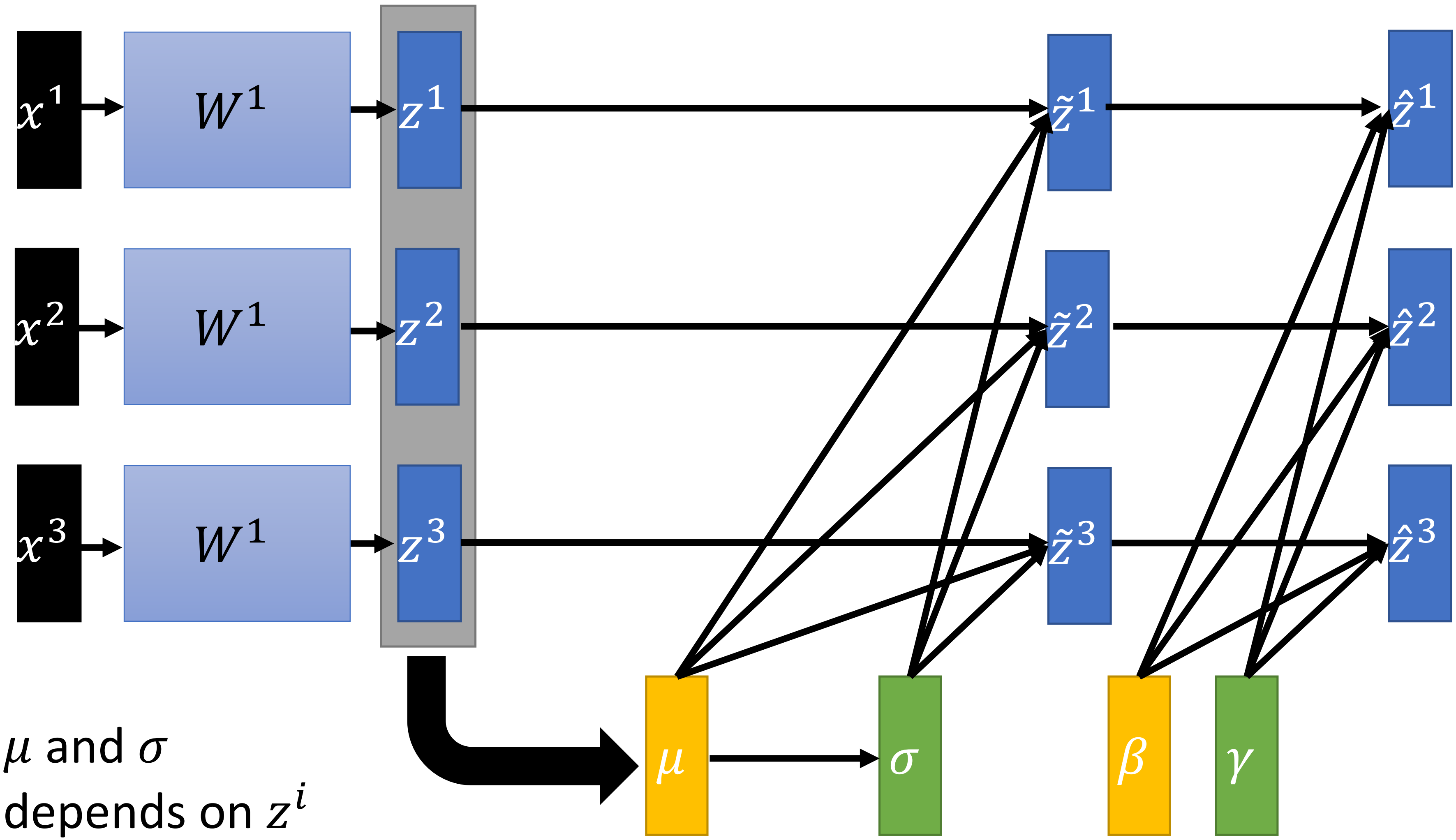
μ and σ depends on z^i

How to do backpropogation?

Batch normalization

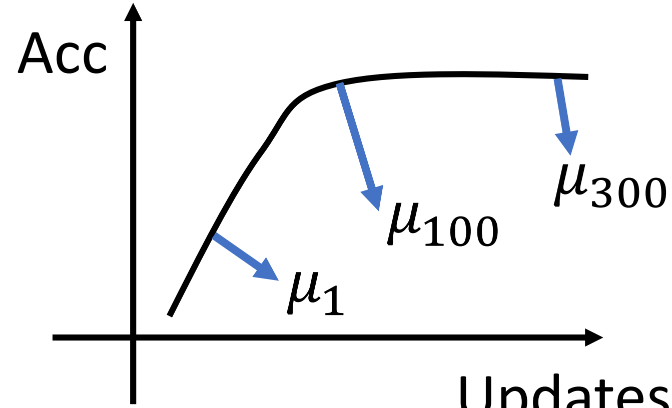
$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$

$$\hat{z}^i = \gamma \odot \tilde{z}^i + \beta$$

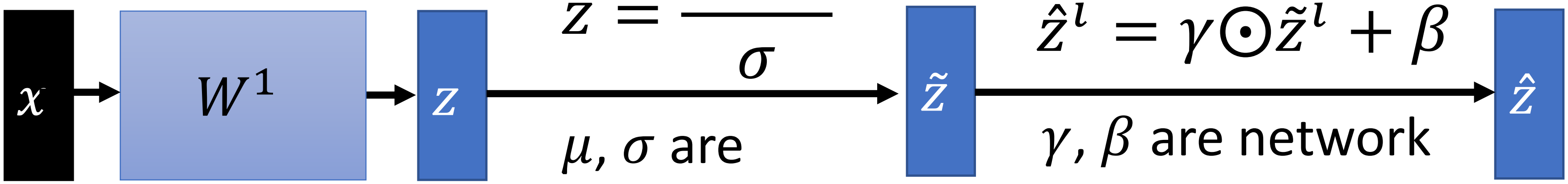


μ and σ depends on z^i

Batch normalization



- At testing stage:



$$\tilde{z} = \frac{z - \mu}{\sigma}$$

μ, σ are from ***batch***

$$\hat{z}^i = \gamma \odot \tilde{z}^i + \beta$$

γ, β are network parameters

We do not have ***batch*** at testing stage.

Ideal solution:

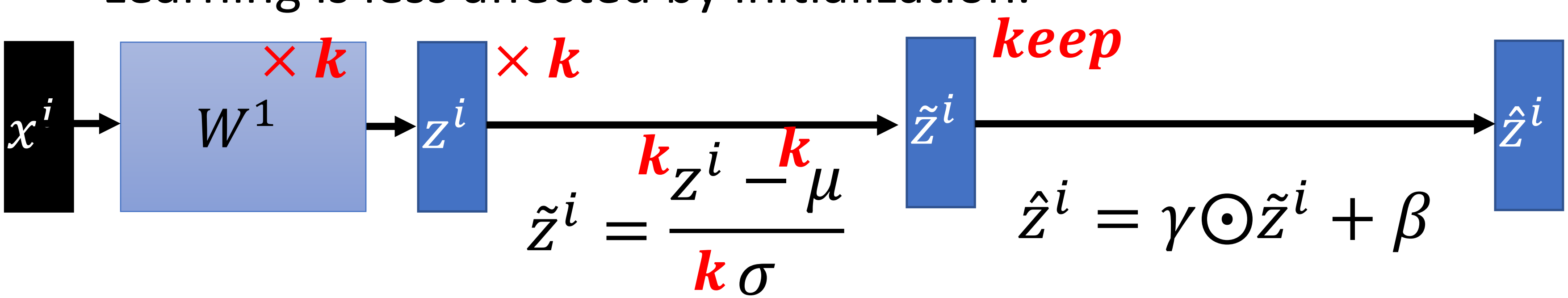
Computing μ and σ using the whole training dataset.

Practical solution:

Computing the moving average of μ and σ of the batches during training.

Batch normalization - Benefit

- BN reduces training times, and make very deep net trainable.
  - Because of less Covariate Shift, we can use larger learning rates.
  - Less exploding/vanishing gradients
    - Especially effective for sigmoid, tanh, etc.
- Learning is less affected by initialization.



- BN reduces the demand for regularization.

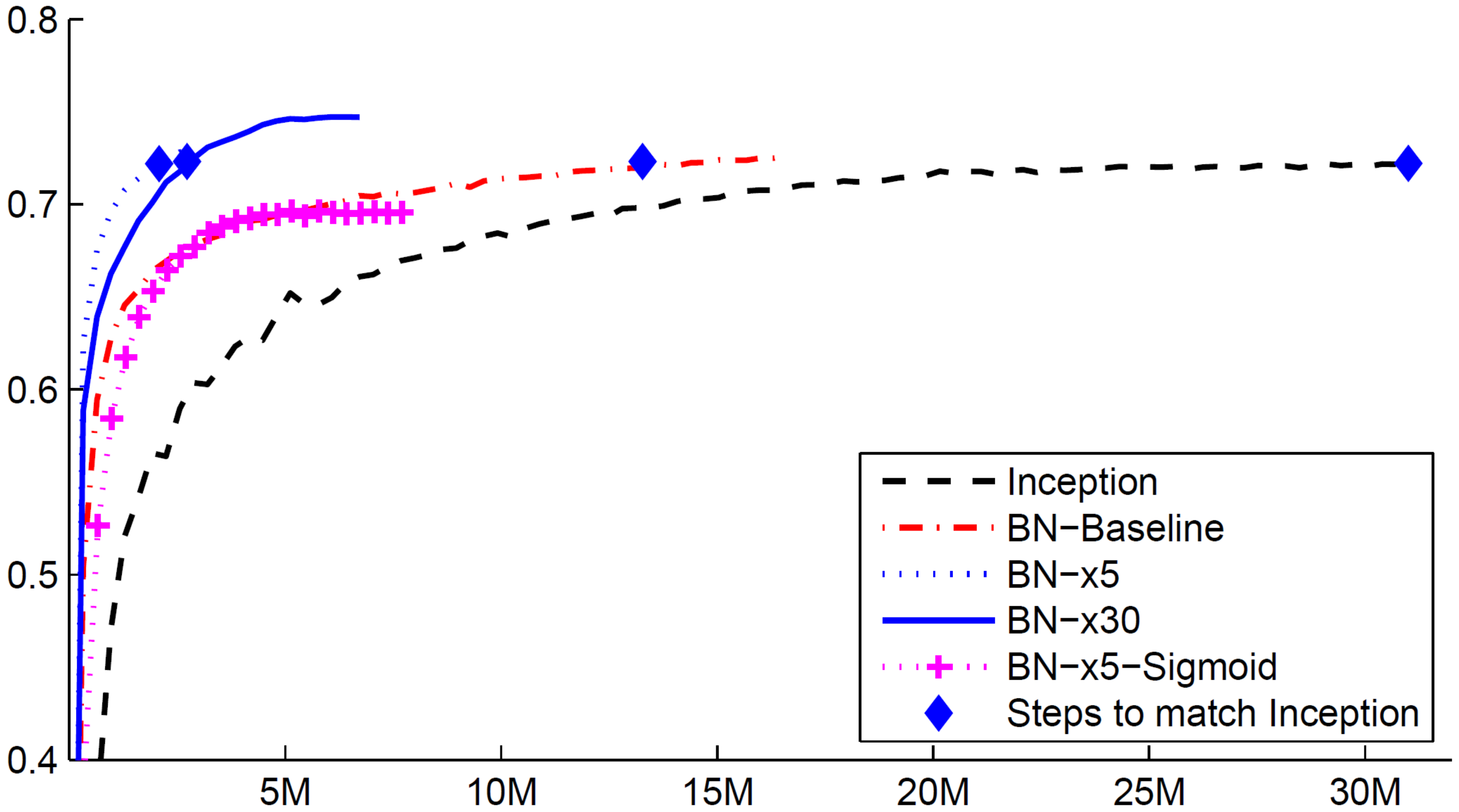


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

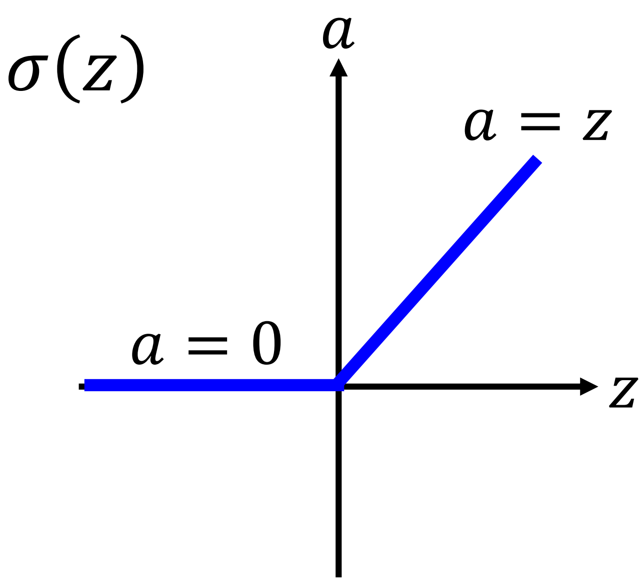
To learn more ……

- Batch Renormalization
- Layer Normalization
- Instance Normalization
- Weight Normalization
- Spectrum Normalization

Activation Function: SELU

ReLU

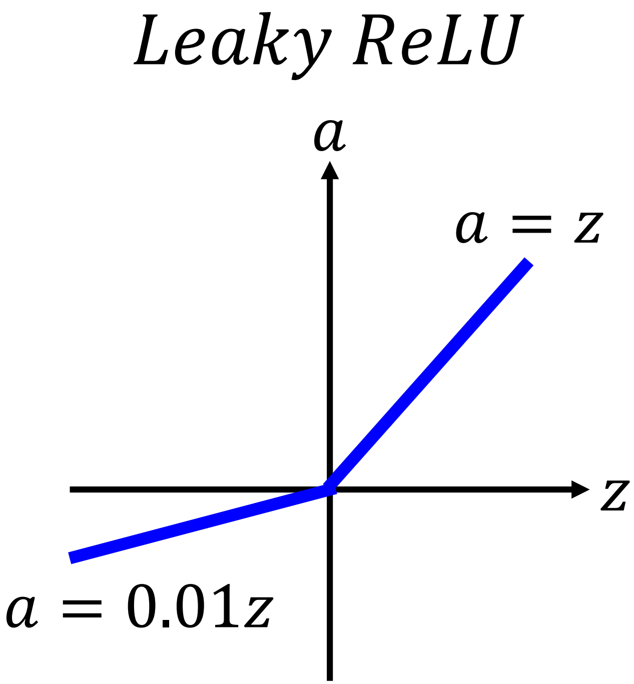
- Rectified Linear Unit (ReLU)

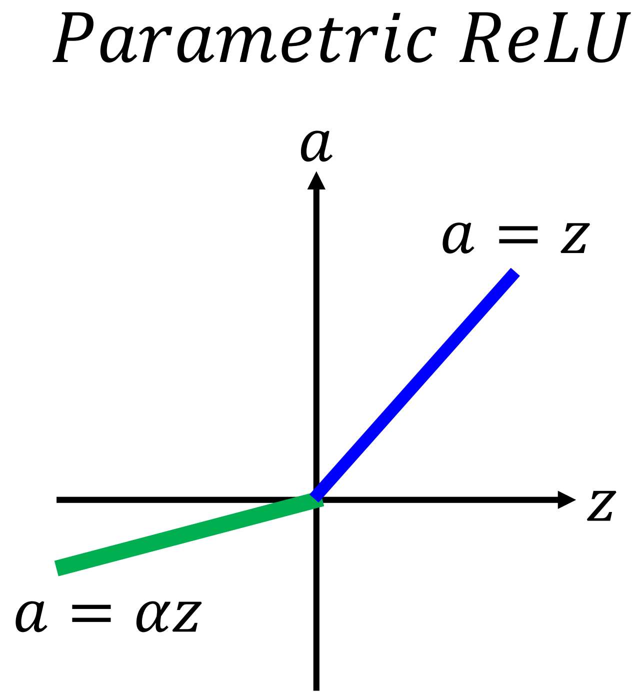


<u>Reason:</u>

1. Fast to compute
2. Biological reason
3. Infinite sigmoid with different biases
4. Vanishing gradient problem

ReLU - variant





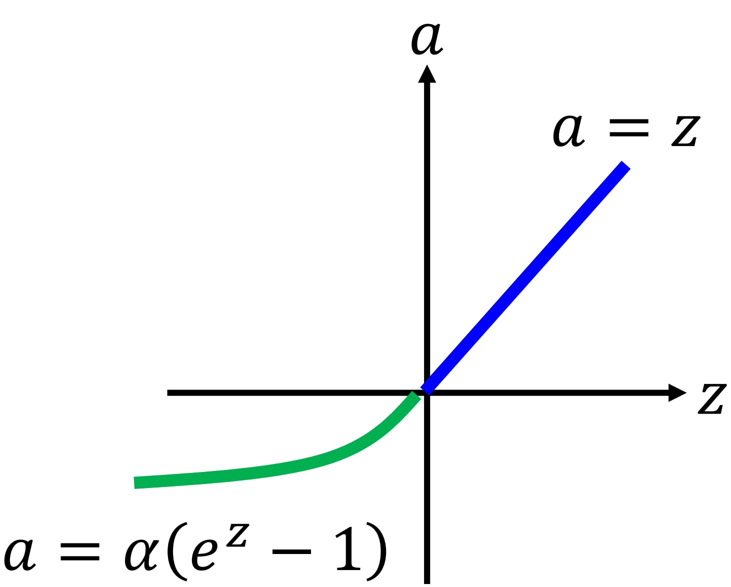
α also learned by gradient descent

(1) Definition of scaled exponential linear units (SELUs)

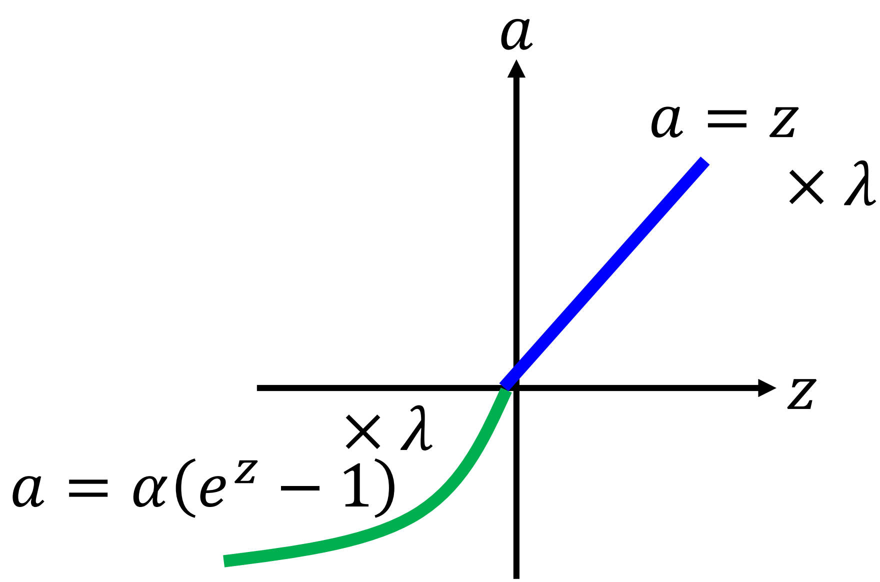
In [3]:
```
def selu(x):
    with ops.name_scope('elu') as scope:
        alpha = 1.6732632423543772848170429916717
        scale = 1.0507009873554804934193349852946
        return scale*tf.where(x>=0.0, x, alpha*tf.nn.elu(x))
```

https://github.com/bioinf-jku/SNNs

Exponential Linear Unit (ELU)



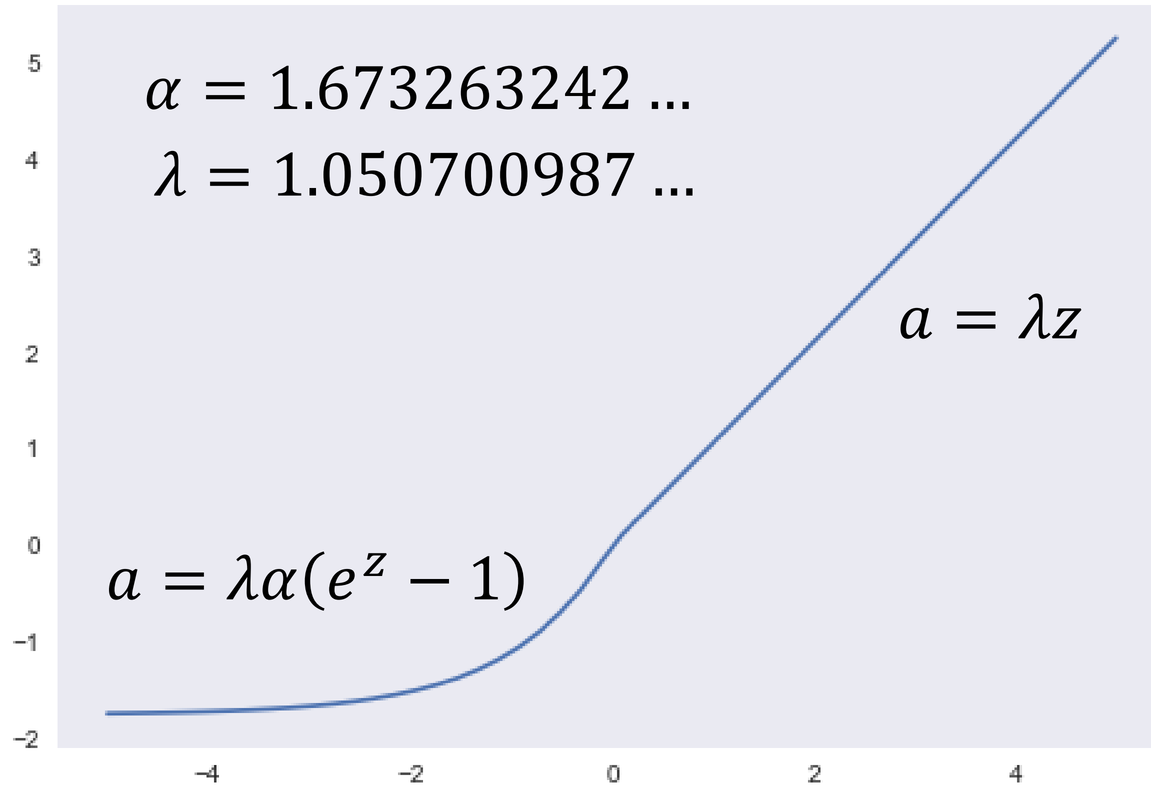
Scaled ELU (SELU)



$$\alpha = 1.6732632423543772848170429916717$$

$$\lambda = 1.0507009873554804934193349852946$$

SELU



Positive and negative values

➡ The whole ReLU family has this property except the original ReLU.

Saturation region ➡ ELU also has this property

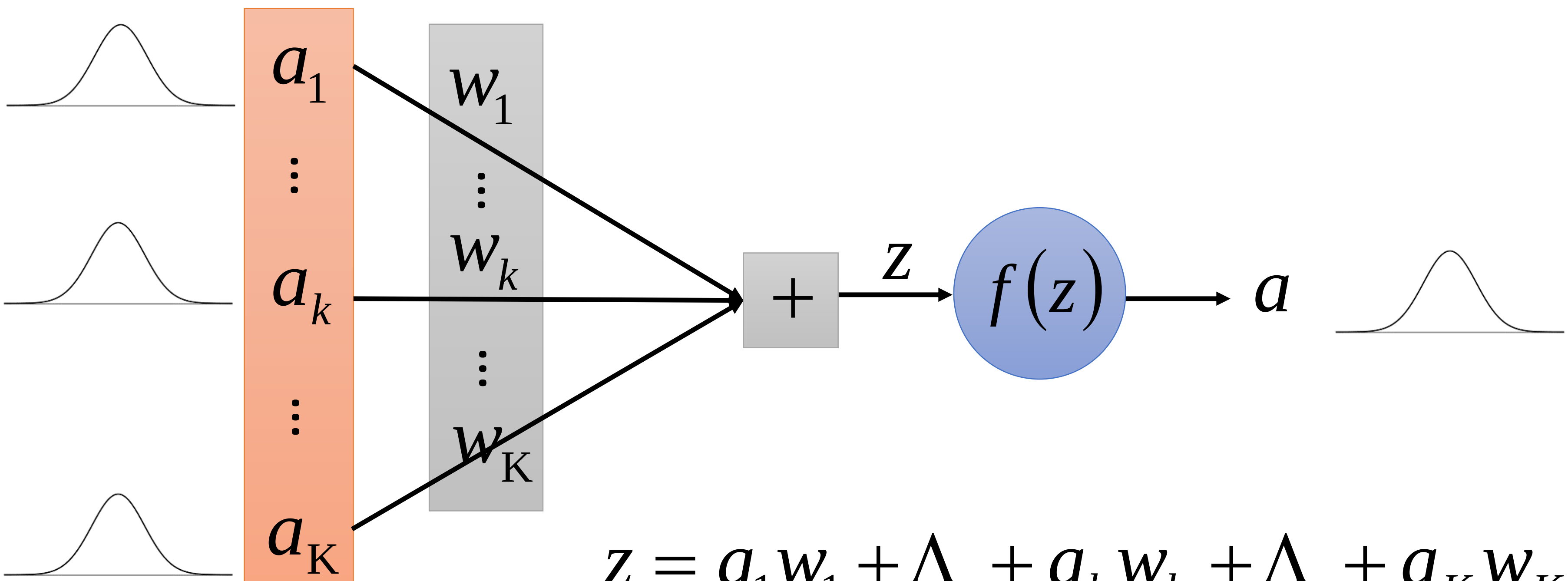
Slope larger than 1 ➡ Only SELU also has this property

SELU

$$\mu_z = E[z]$$

$$= \sum_{k=1}^{K} \underbrace{E[a_k]}_{\mu} w_k = \mu \underbrace{\sum_{k=1}^{K} w_k}_{=0} = \mu \cdot K \underbrace{\mu_w}_{=0}$$

The inputs are i.i.d random variables with mean $\underbrace{\mu}_{=0}$ and variance $\underbrace{\sigma^2}_{=1}$.



$$z = a_1 w_1 + \Lambda + a_k w_k + \Lambda + a_K w_K$$

Do not have to be Gaussian

SELU

$$\mu_z = 0 \qquad \mu_w = 0$$

$$\sigma_z^2 = E[(z-\mu_z)^2] = E[z^2]$$

$$= E[(a_1 w_1 + a_2 w_2 + \cdots)^2]$$

$$= \sum_{k=1}^{K} (w_k)^2 \sigma^2 = \underbrace{\sigma^2}_{=1} \cdot \underbrace{K\sigma_w^2}_{=1} = 1$$

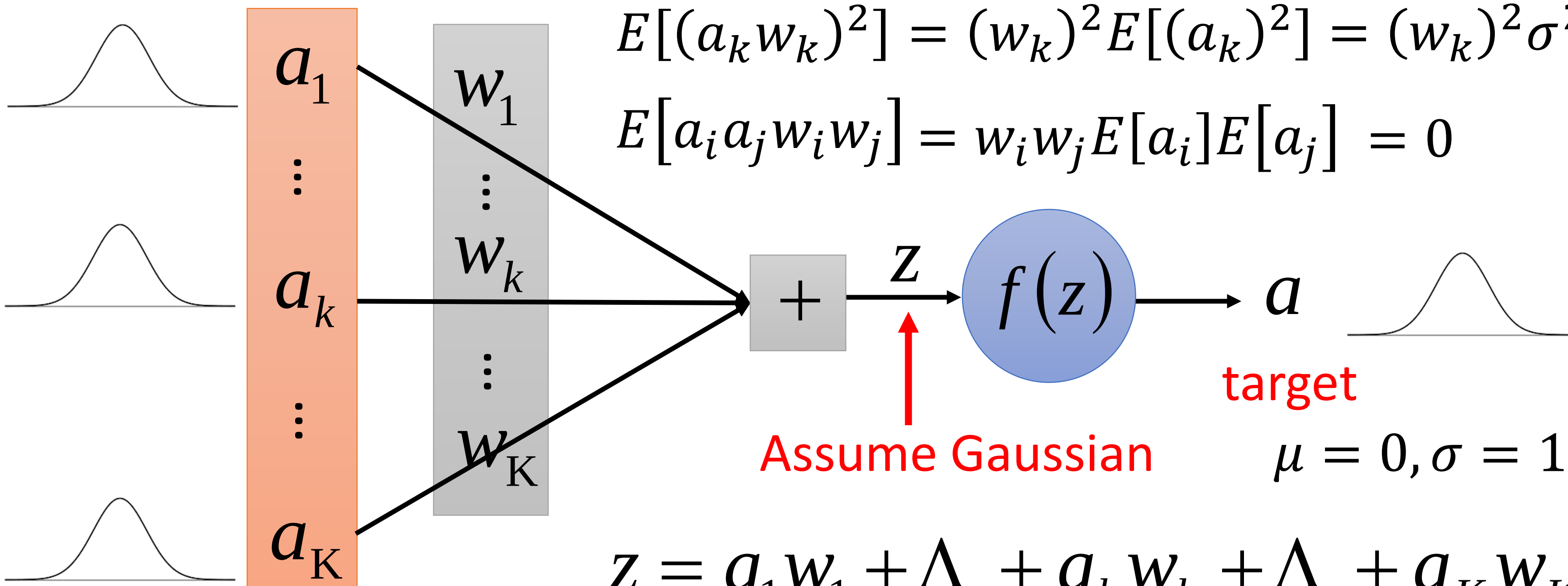
The inputs are i.i.d random variables with mean μ and variance σ^2.

=0

=1

$$E[(a_k w_k)^2] = (w_k)^2 E[(a_k)^2] = (w_k)^2 \sigma^2$$

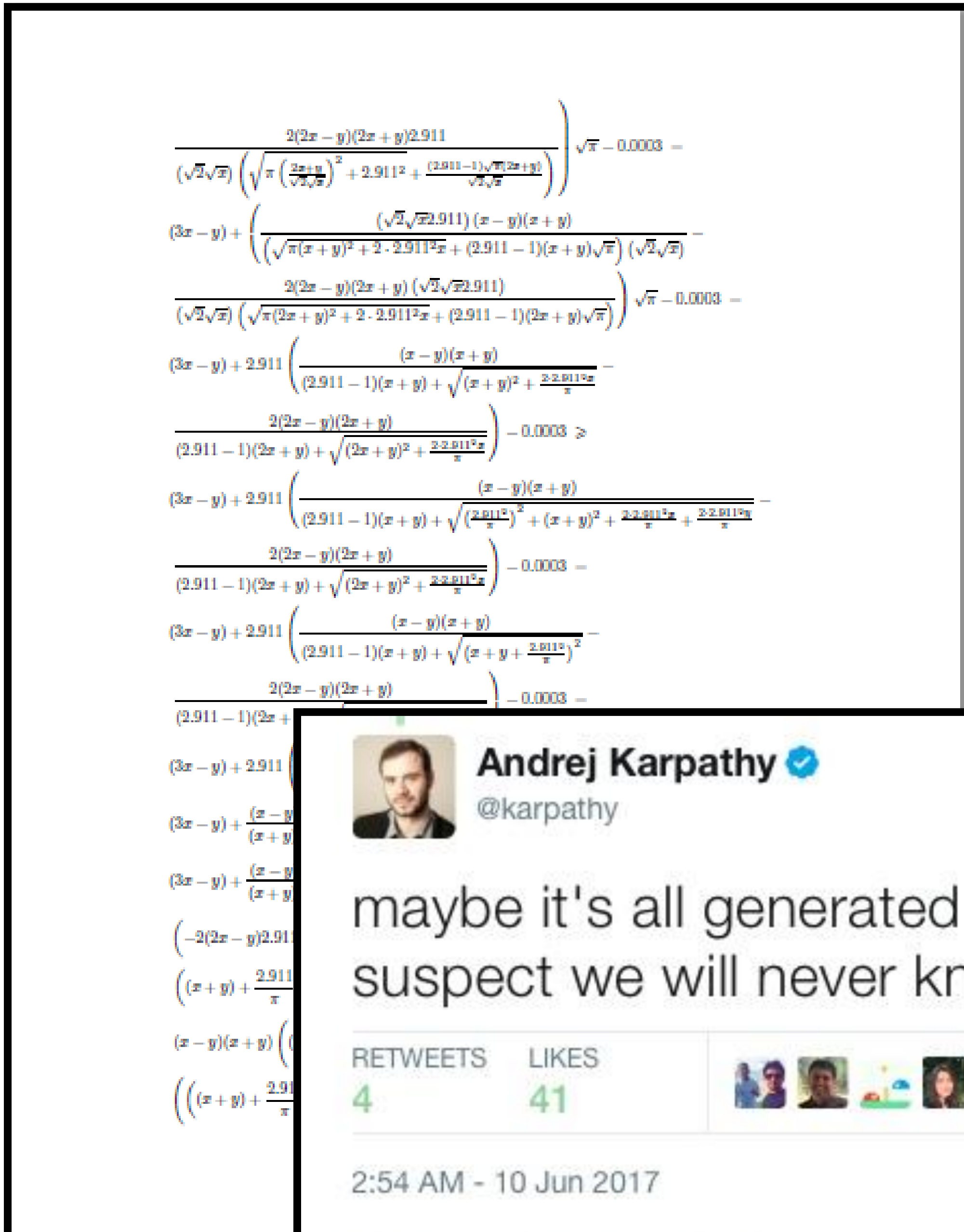
$$E\left[a_i a_j w_i w_j\right] = w_i w_j E[a_i] E\left[a_j\right] = 0$$



$$\mu = 0, \sigma = 1$$

$$z = a_1 w_1 + \Lambda + a_k w_k + \Lambda + a_K w_K$$

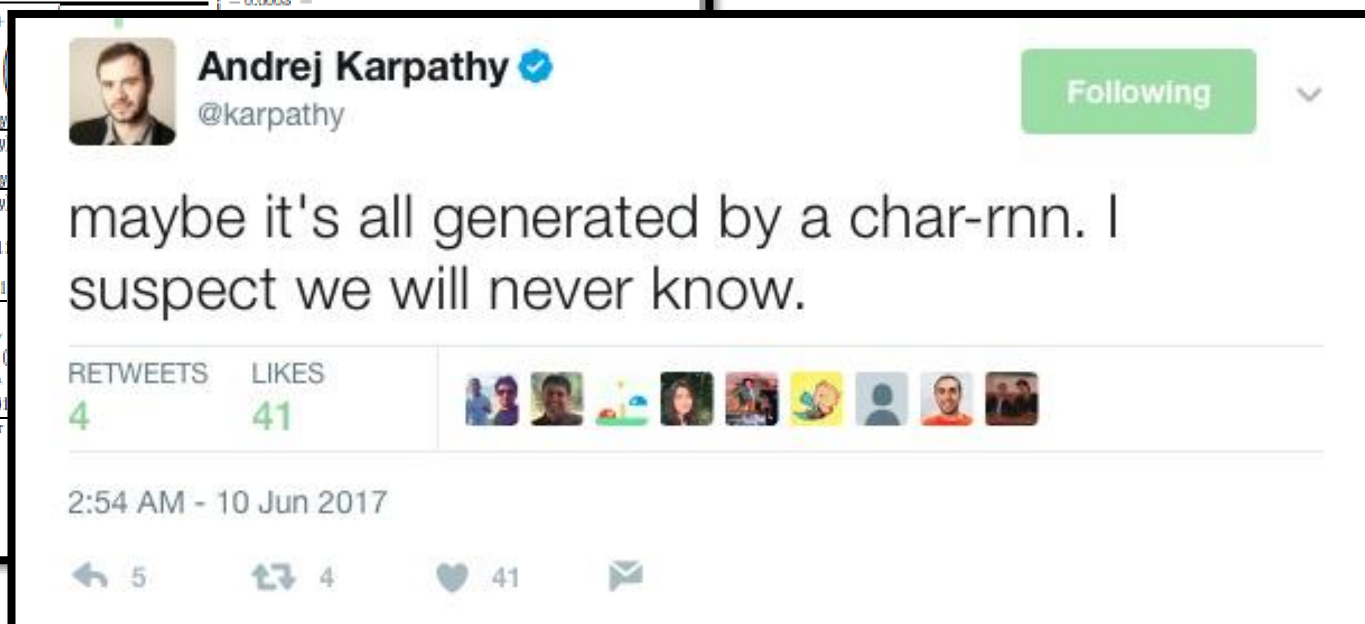
Demo

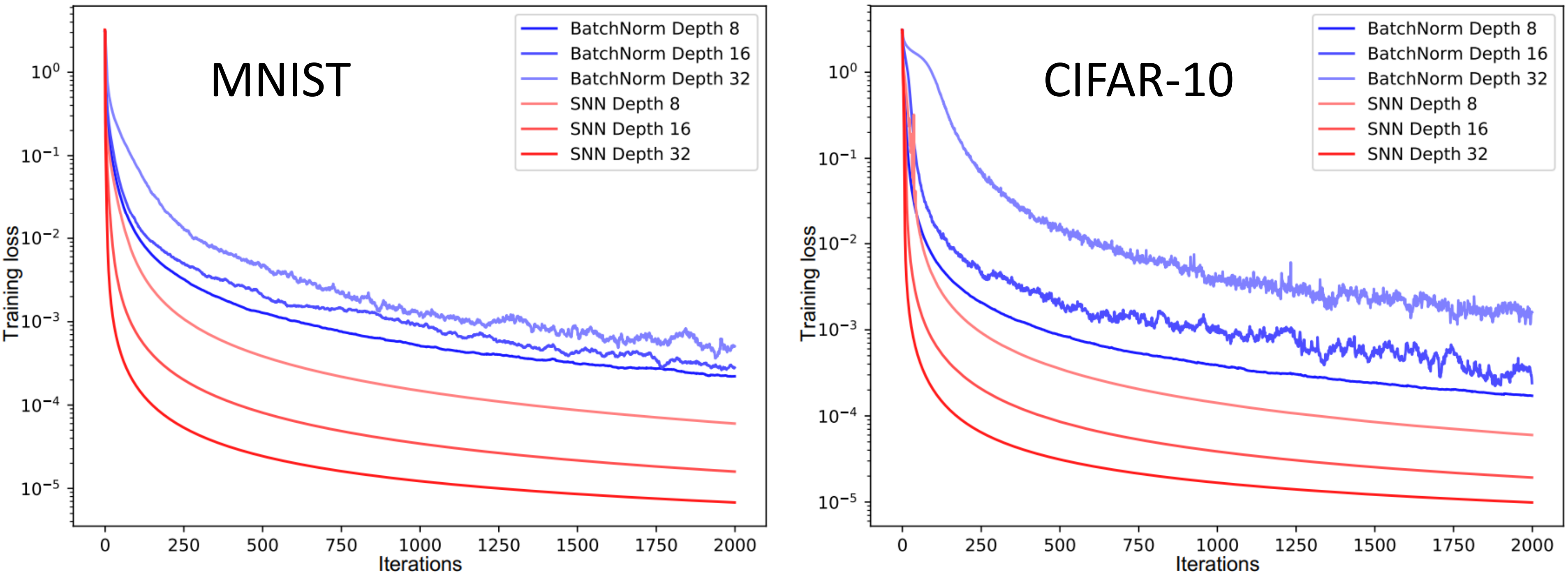


93 頁的證明

Source of joke: https://zhuanlan.zhihu.com/p/27336839

SELU is actually more general.

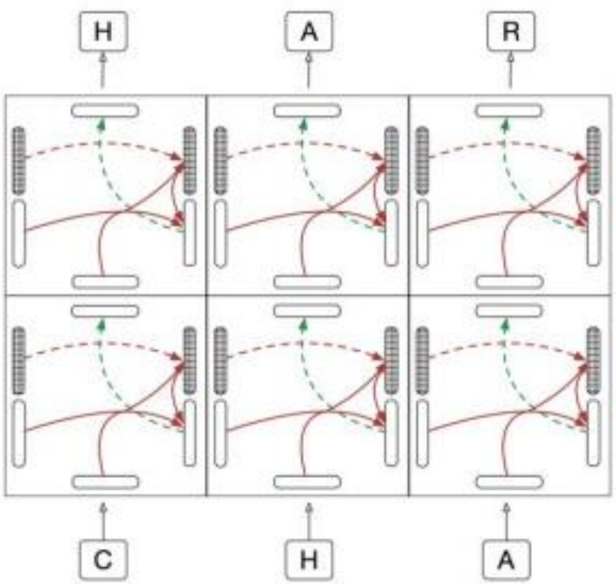


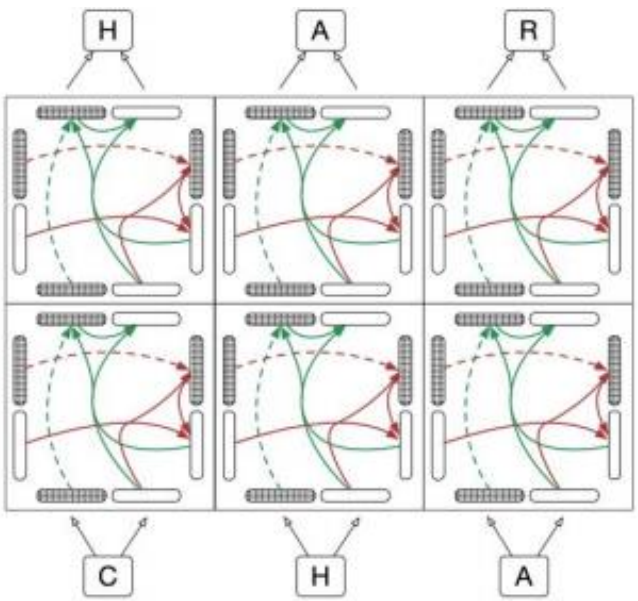


FNN method comparison			ML method comparison		
Method	avg. rank diff.	p-value	Method	avg. rank diff.	p-value
SNN	-0.756		SNN	-6.7	
MSRAinit	-0.240*	2.7e-02	SVM	-6.4	5.8e-01
LayerNorm	-0.198*	1.5e-02	RandomForest	-5.9	2.1e-01
Highway	0.021*	1.9e-03	MSRAinit	-5.4*	4.5e-03
ResNet	0.273*	5.4e-04	LayerNorm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

Demo

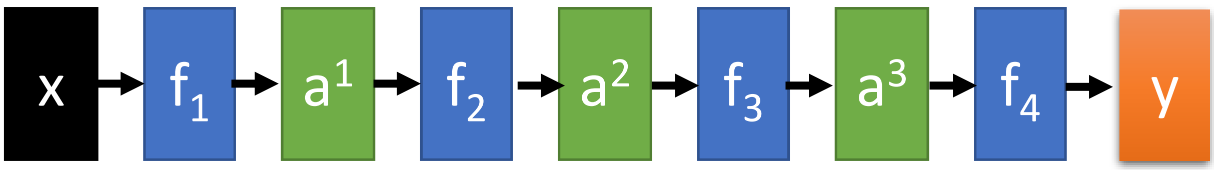
Highway Network & Grid LSTM





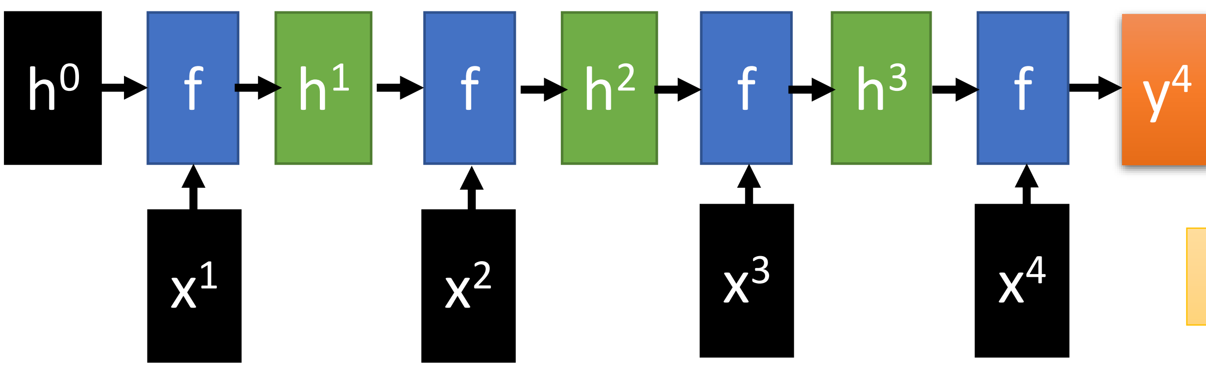
Feedforward v.s. Recurrent

1. Feedforward network does not have input at each step
2. Feedforward network has different parameters for each layer



$$a^t = f_l(a^{t-1}) = \sigma(W^t a^{t-1} + b^t)$$

t is layer



t is time step

$$h^t = f(h^{t-1}, x^t) = \sigma\left(W^h h^{t-1} + W^i x^t + b^i\right)$$

Applying gated structure in feedforward network

GRU → Highway Network

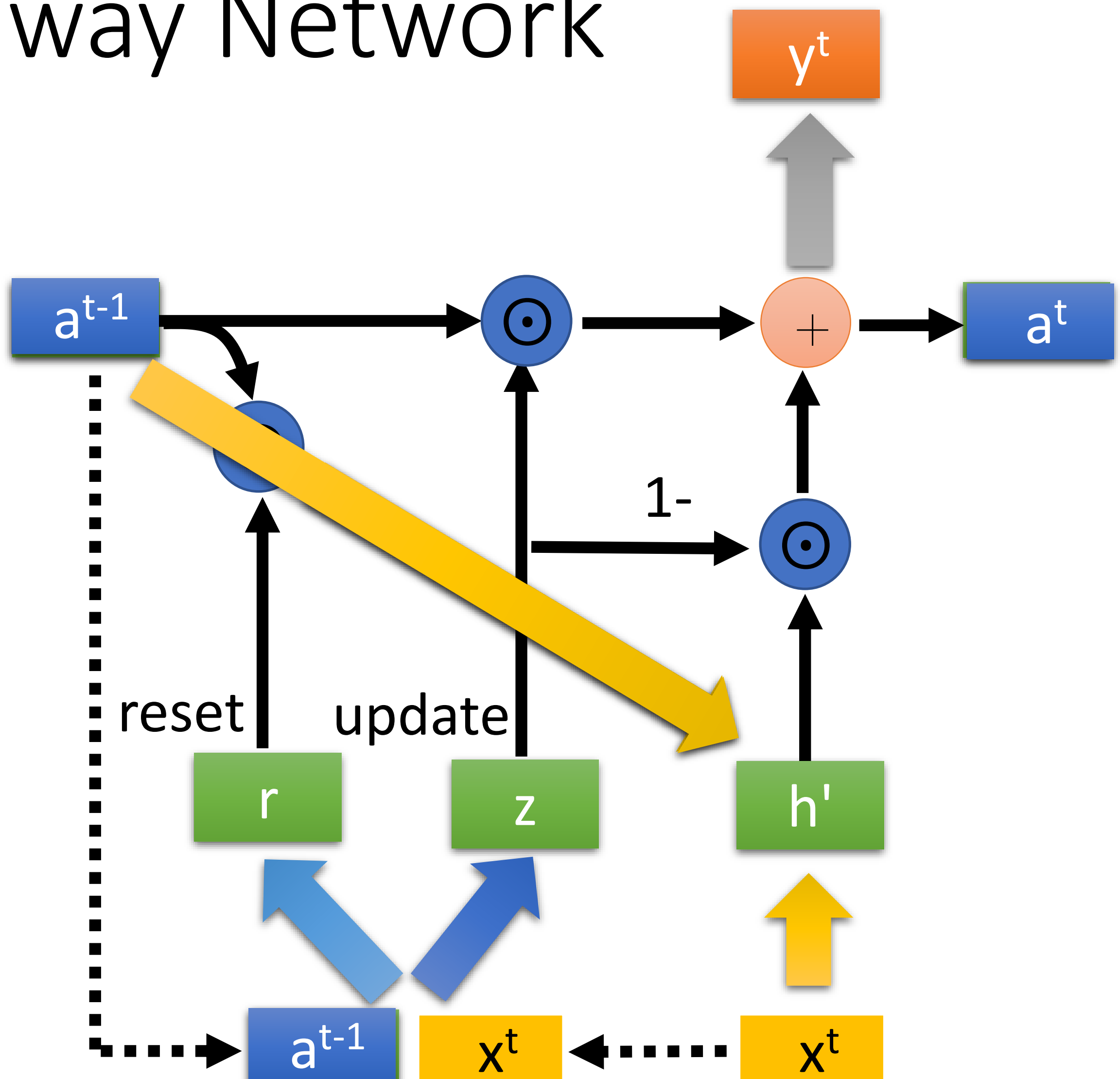
No input x^t at each step

No output y^t at each step

a^{t-1} is the output of the (t-1)-th layer

a^t is the output of the t-th layer

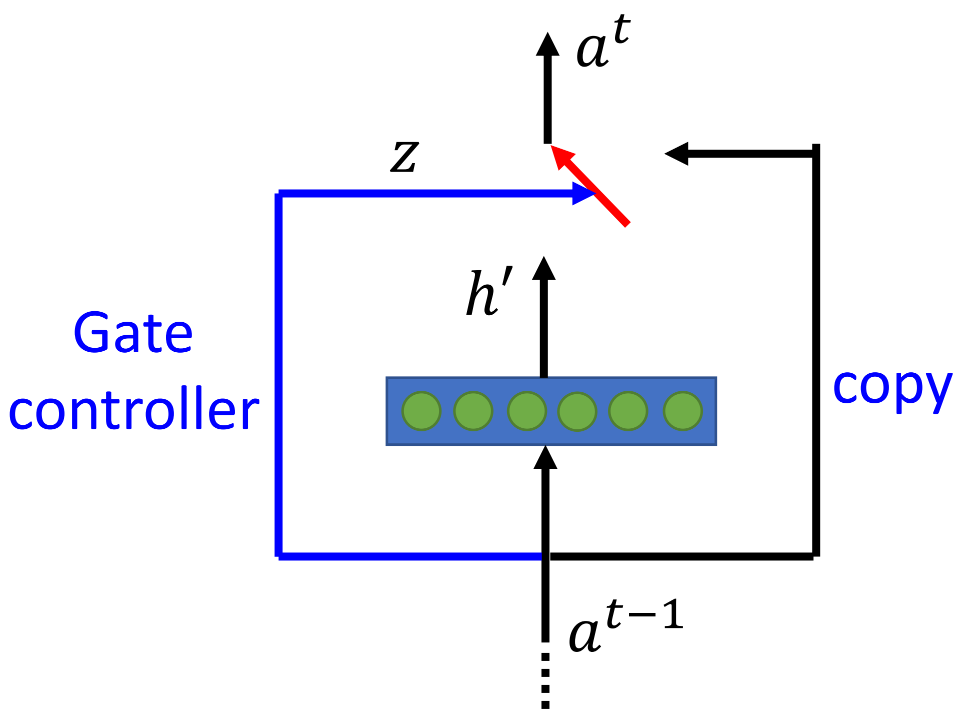
No reset gate



Highway Network

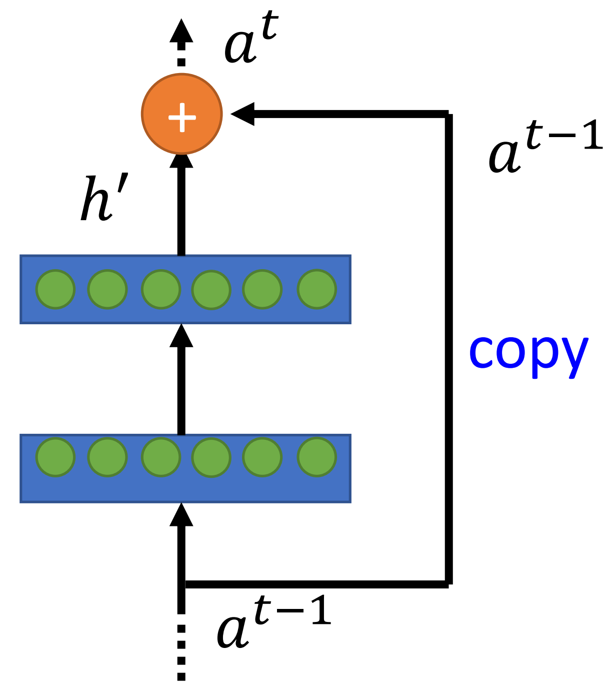
$$h' = \sigma(Wa^{t-1})$$
$$z = \sigma(W'a^{t-1})$$
$$a^t = z \odot a^{t-1} + (1-z) \odot h$$

- **Highway Network**

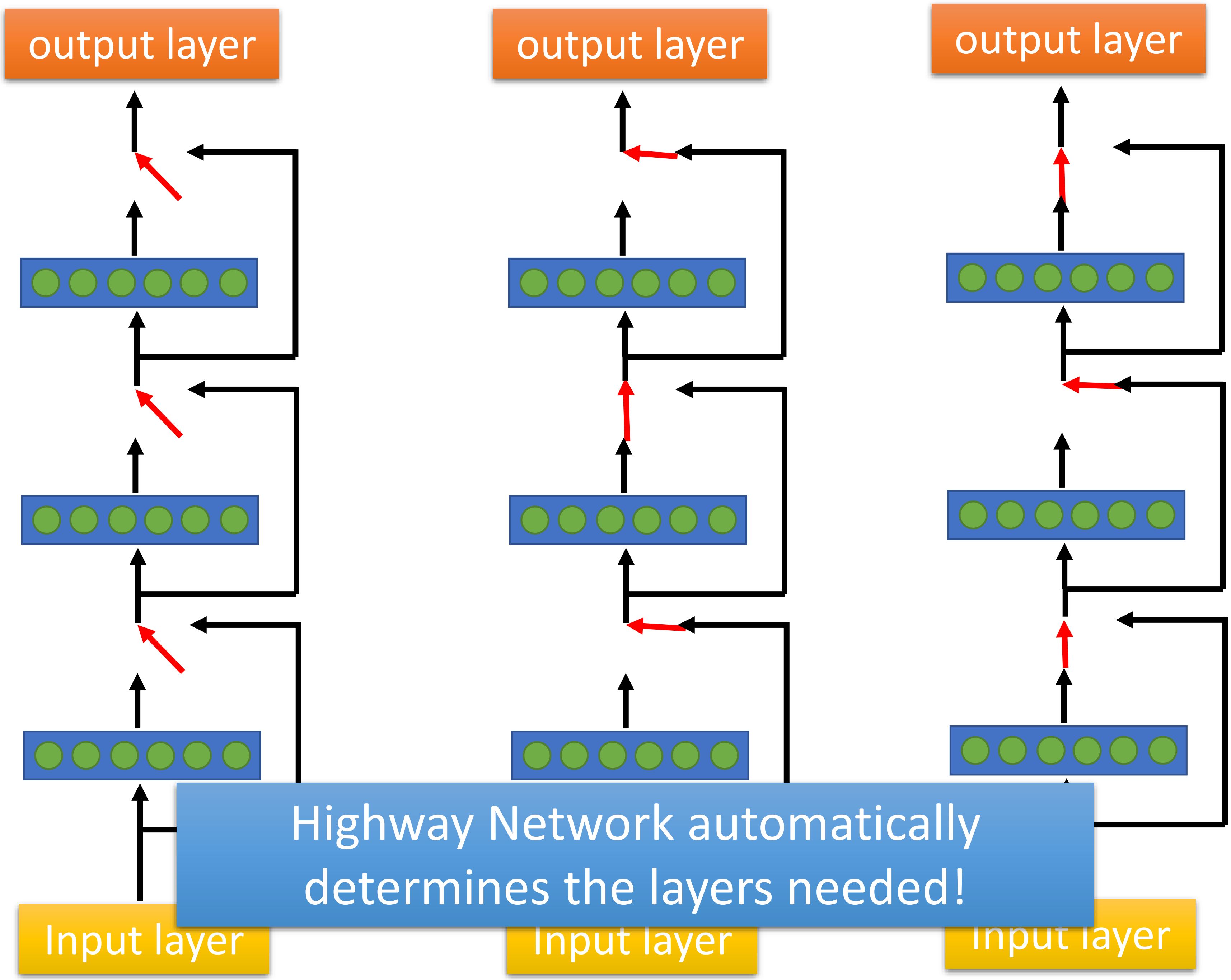


Training Very Deep Networks https://arxiv.org/pdf/1507.06228v2.pdf

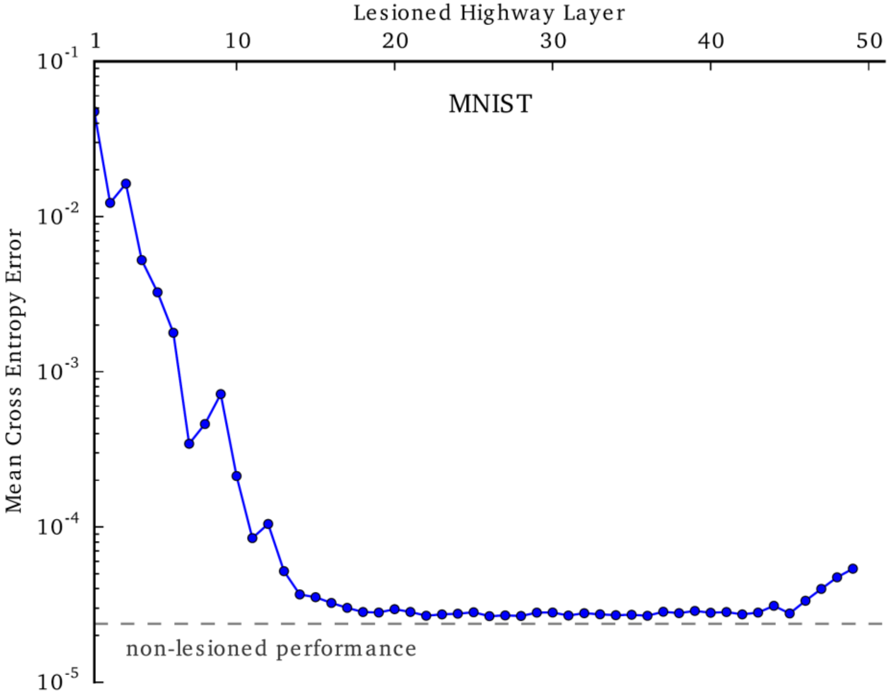
- **Residual Network**

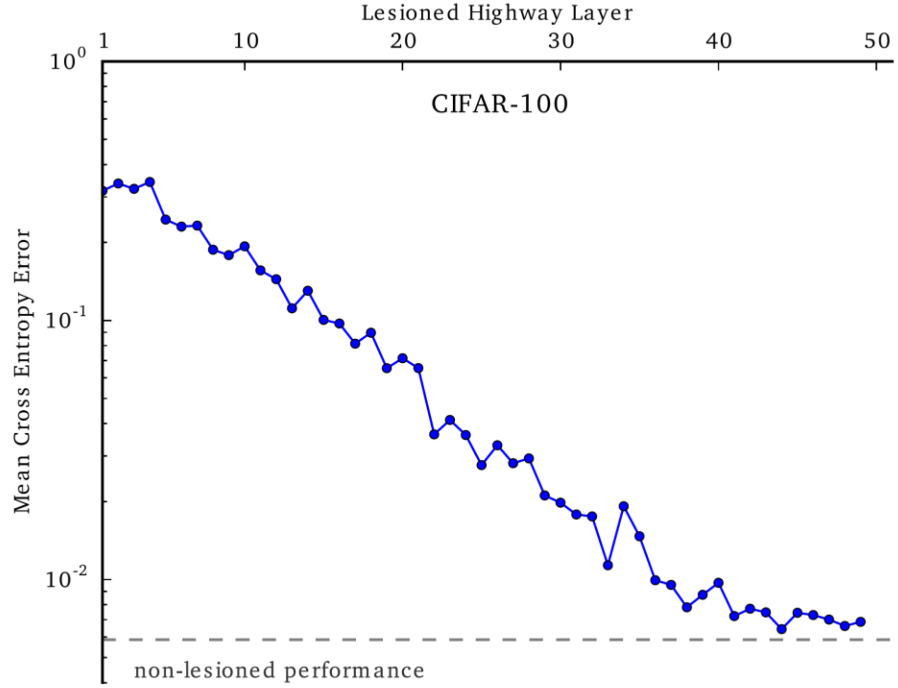


Deep Residual Learning for Image Recognition http://arxiv.org/abs/1512.03385

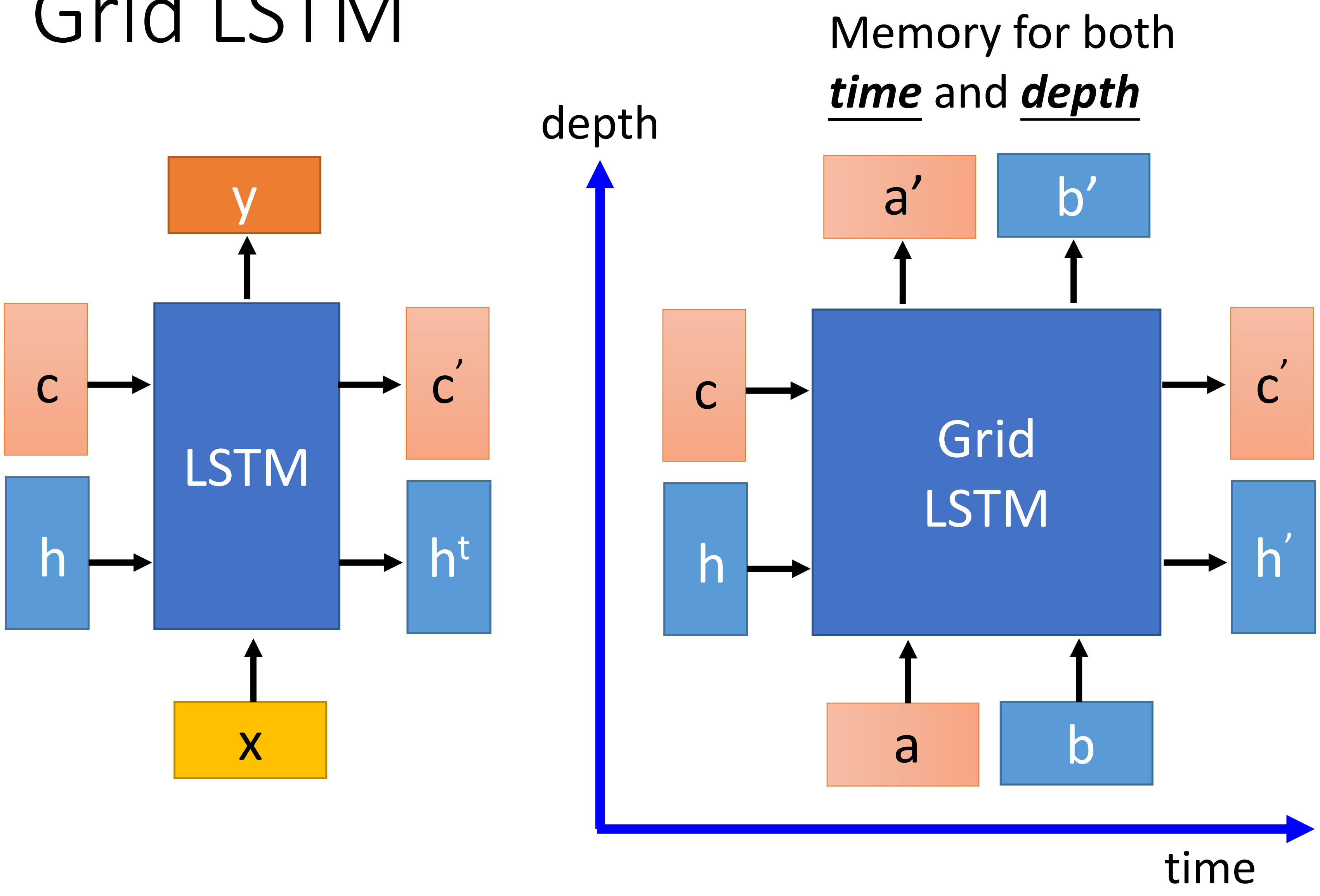

output layer
output layer
output layer
Highway Network automatically determines the layers needed!
Input layer
Input layer
Input layer

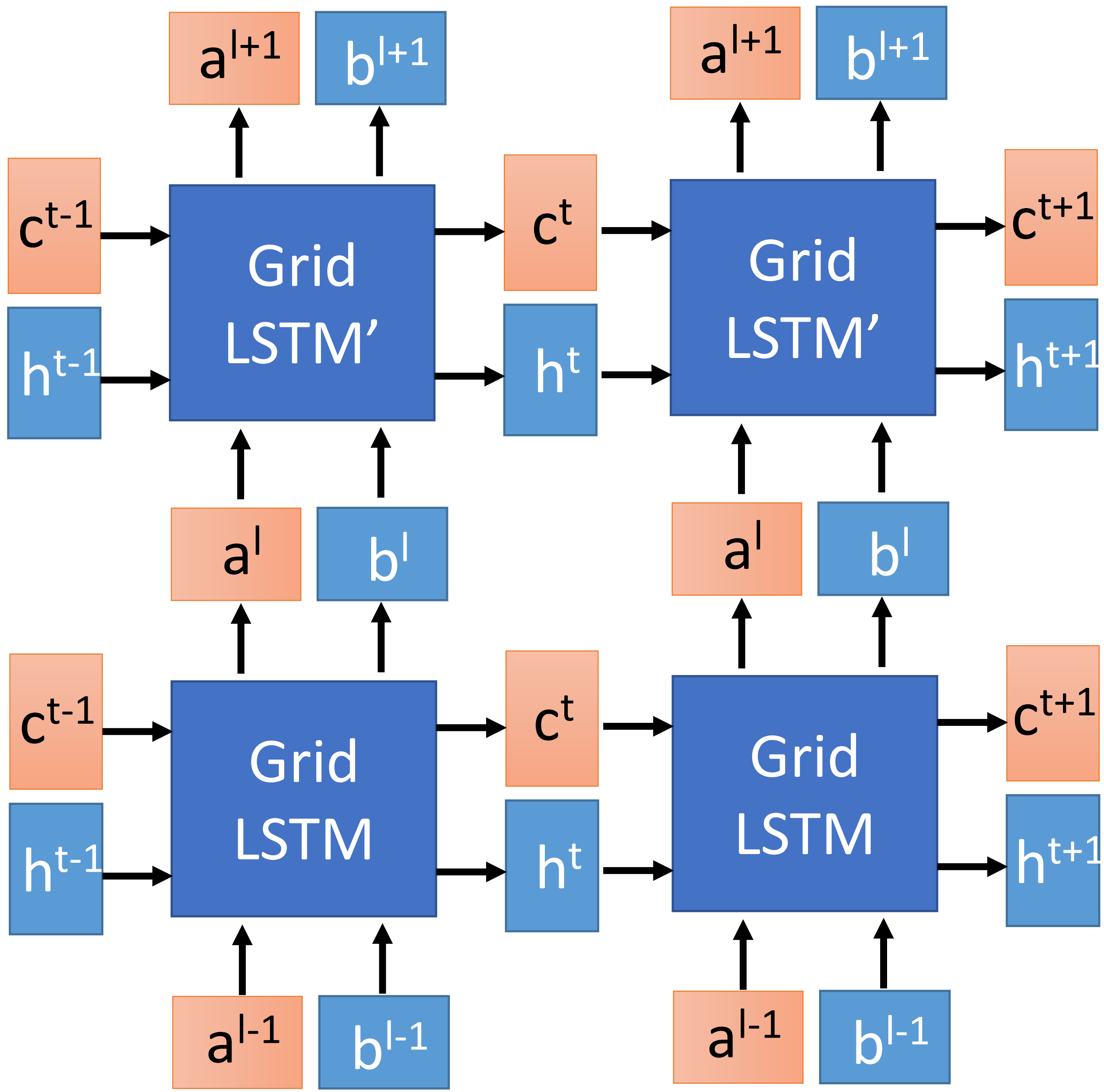
Highway Network





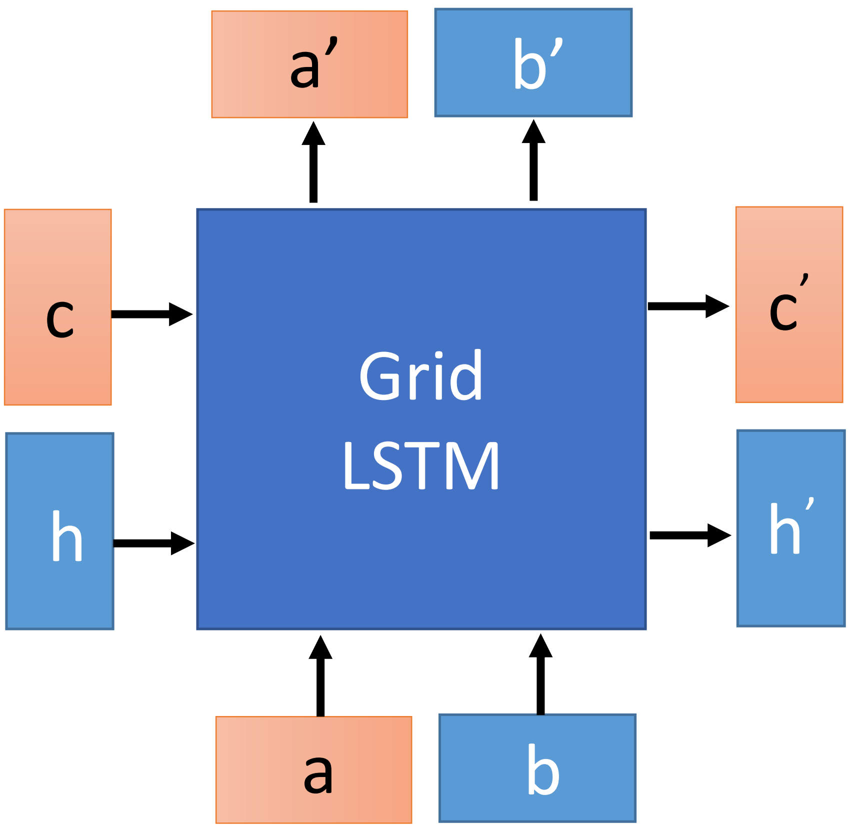
Grid LSTM

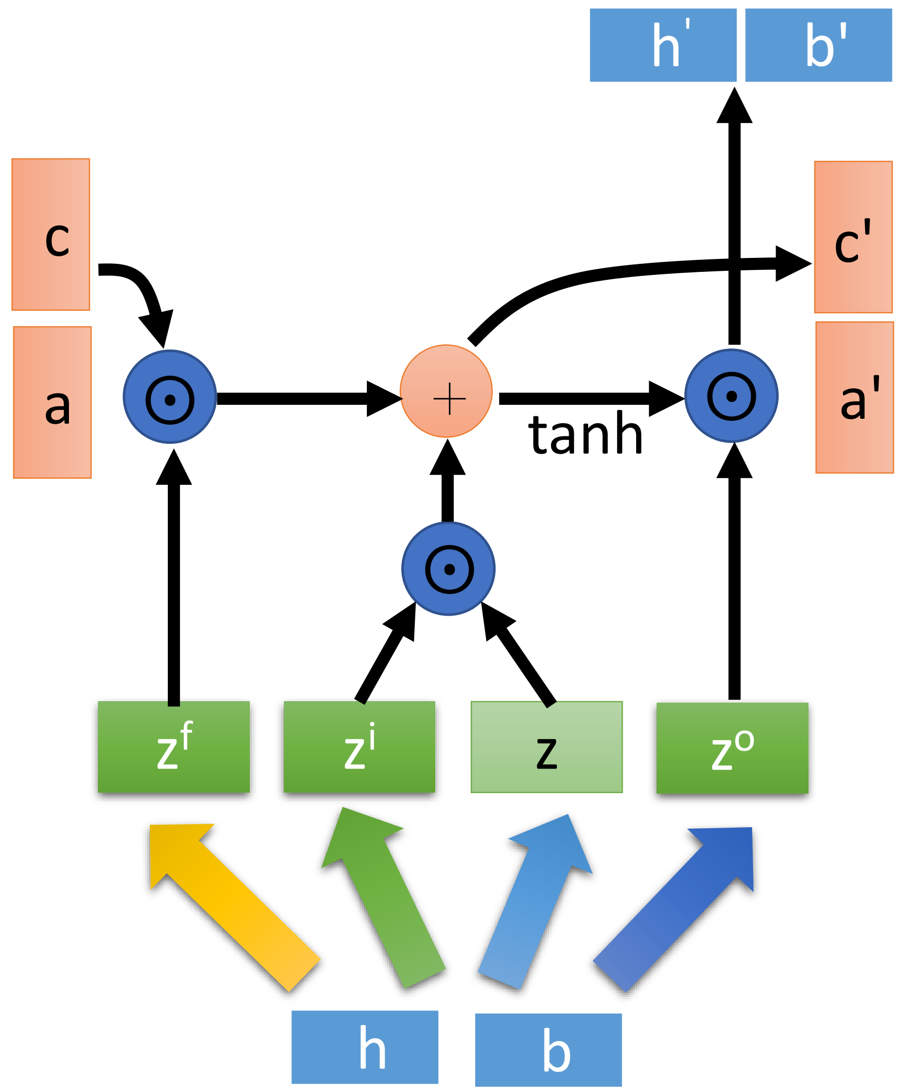

y
c
h
LSTM
c'
h^t
x
Memory for both
time and depth
depth
a'
b'
c
h
Grid
LSTM
c'
h'
a
b
time



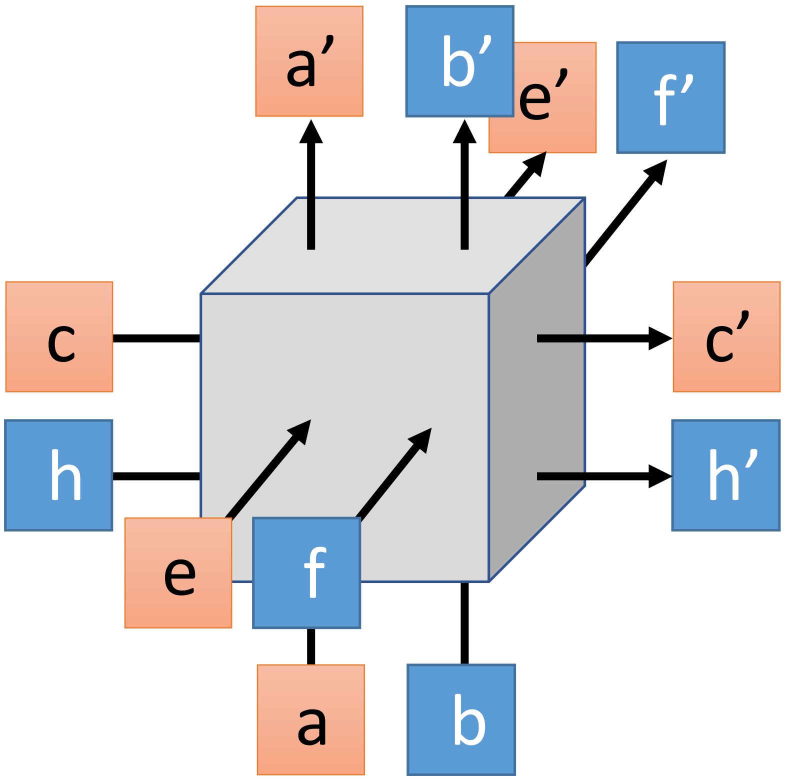
a^{l+1}
b^{l+1}
c^{t-1}
h^{t-1}
Grid LSTM'
c^t
h^t
c^{t+1}
h^{t+1}
a^l
b^l
Grid LSTM
a^{l-1}
b^{l-1}

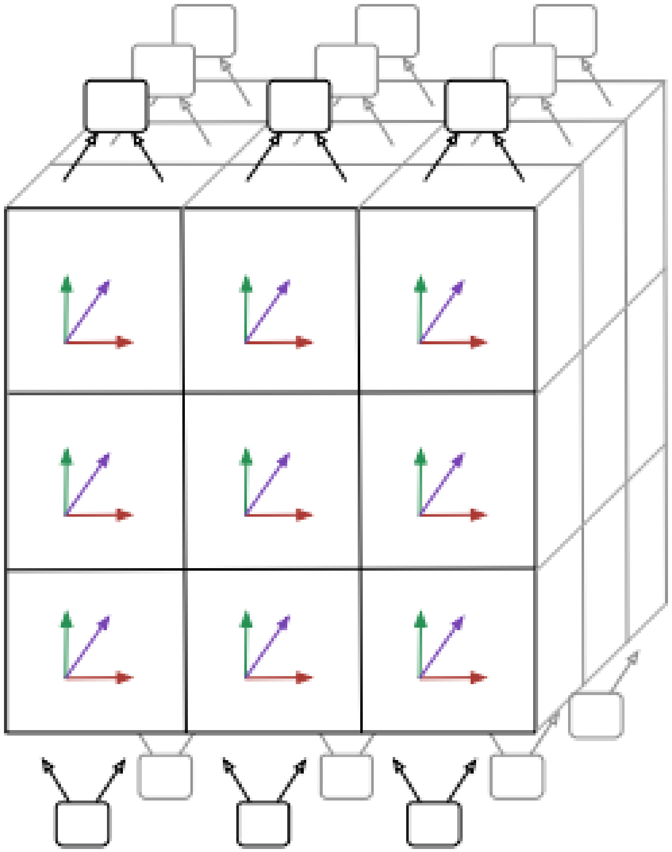
Grid LSTM





3D Grid LSTM





3D Grid LSTM

3 x 3 images

- Images are composed of pixels