Institute of Electronics
National Yang Ming Chiao Tung University
Hsinchu, Taiwan

AI Training Course Series

Neural Network Training Skills

Lecture 3

Architecture
Development &
Algorithm
Refinement for
Intelligent Computing

Presenter: Ren-Hong Yang
Advisor: Juinn-Dar Huang, Ph.D.

July 15, 2024

Outline (1/2)

- Introduction to Activation Functions
- Common Activation Functions
- Feature Normalization
- Batch Normalization Steps
- Other Normalization Methods
- Dropout and Dropblock

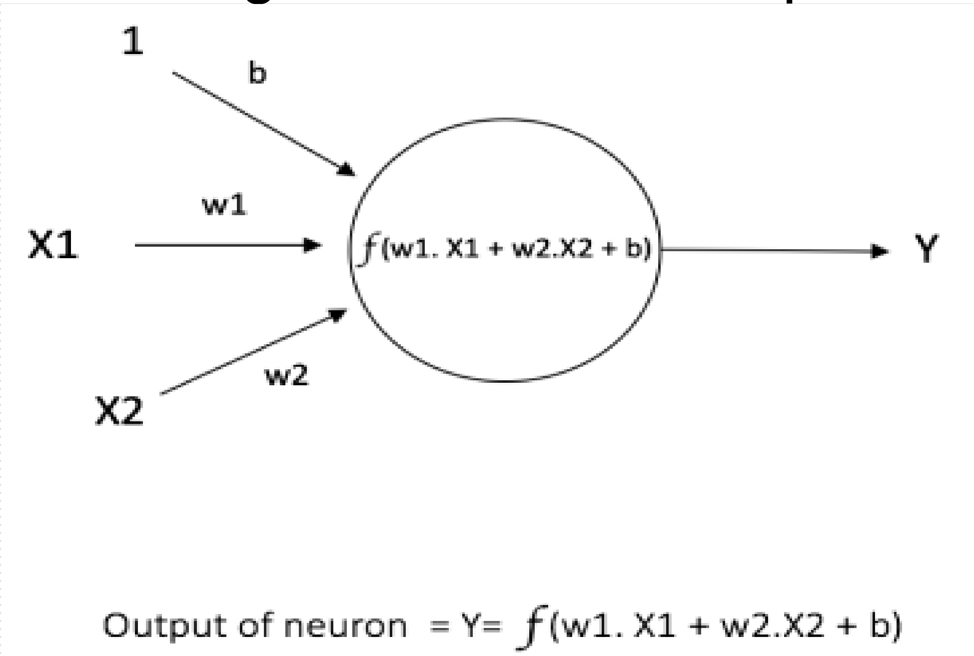
Outline (2/2)

- Introduction to Loss Functions
- Introduction to Momentum
- Introduction to Optimizers
- Common Optimizers
- Learning Rate Schedulers
- Data Augmentation
- References
- Homework

Introduction to Activation Functions

Neural Networks (1/3)

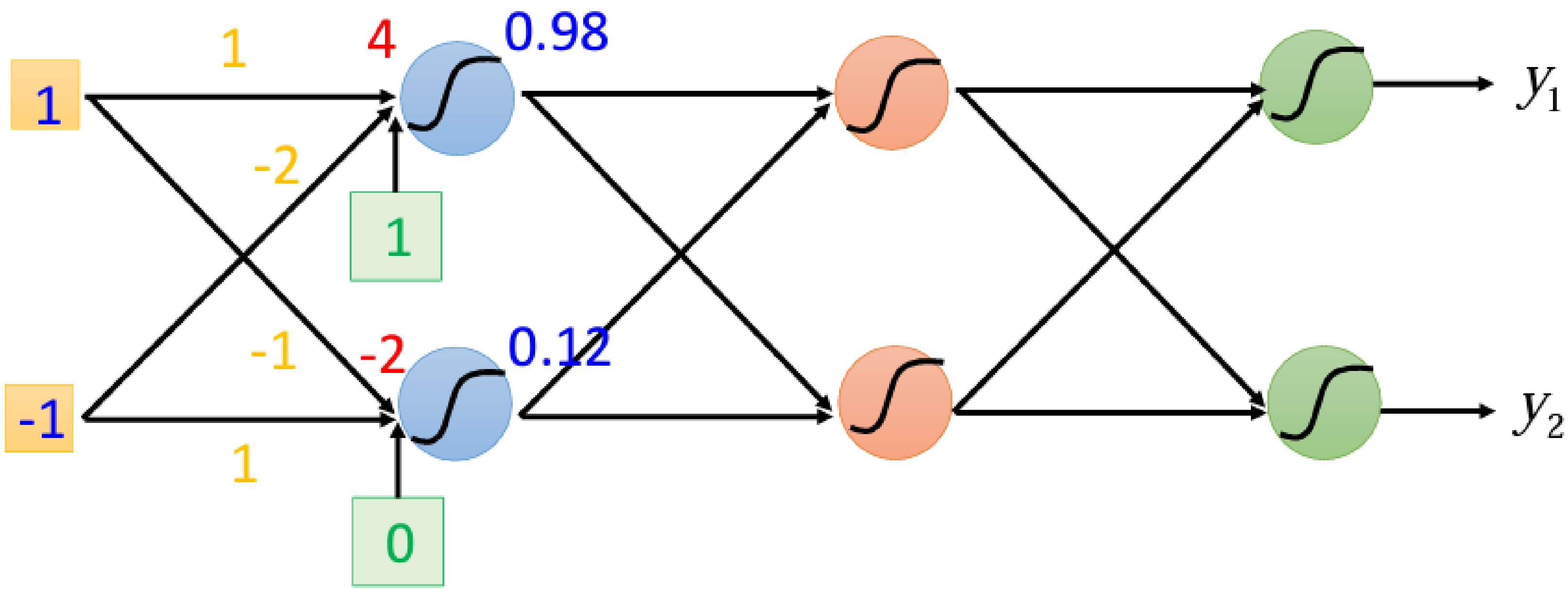
- Behavior of a neuron
  - calculate the "weighted sum" of its input and add a bias



  - the above equation is a linear equation
  - the output of the current layer is a linear combination of the output of the previous layer

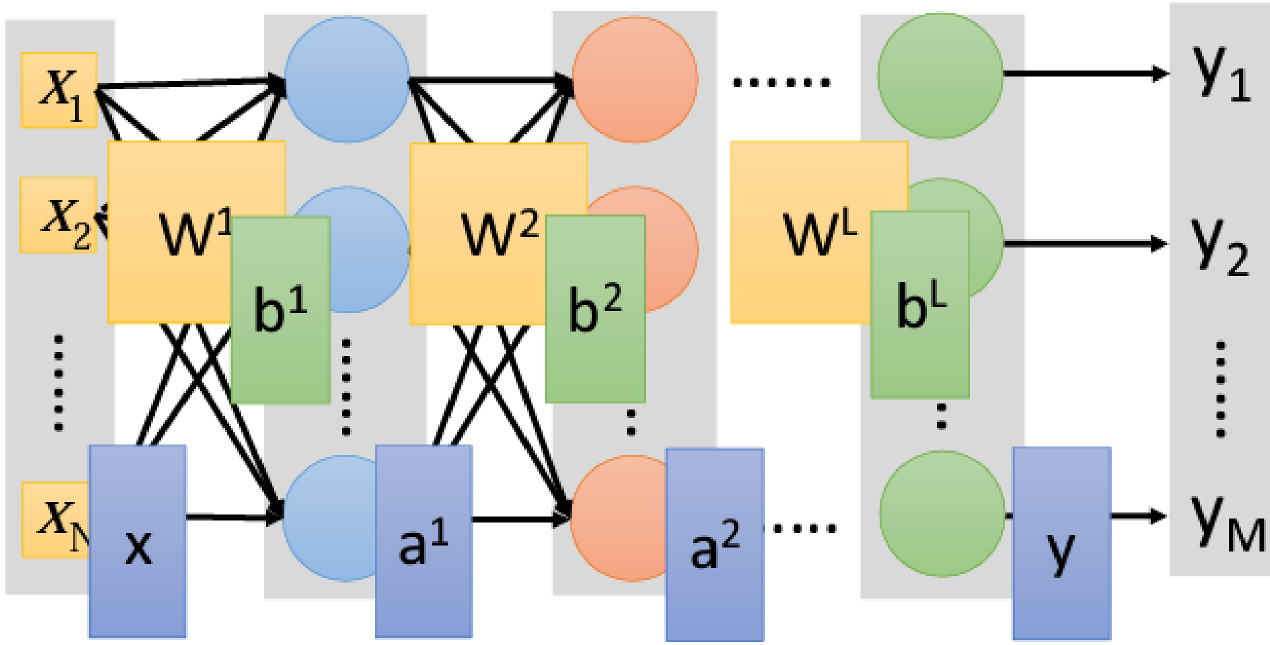
Neural Networks (2/3)

- Matrix operation



$$\sigma\left(\underbrace{\begin{bmatrix}1 & -2\\ -1 & 1\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix}+\begin{bmatrix}1\\ 0\end{bmatrix}}_{\begin{bmatrix}4\\ -2\end{bmatrix}}\right)=\begin{bmatrix}0.98\\ 0.12\end{bmatrix}$$

Neural Networks (3/3)



$$y = f(x)$$

Using parallel computing techniques to speed up matrix operation

$$= \sigma\left(W^L \cdots \overbrace{\sigma\left(W^2 \underbrace{\sigma\left(W^1 x + b^1\right)}_{a^1} + b^2\right)}^{a^2} \cdots + b^L\right)$$

Why Activation Functions?

- Most data sets are huge, discrete, and nonlinear
  – cannot always be represented using linear equations

- For an NN without nonlinear structure, it can be simplified as **a matrix multiplication and addition**
  – lose huge nonlinear features
  – hard to converge during NN training

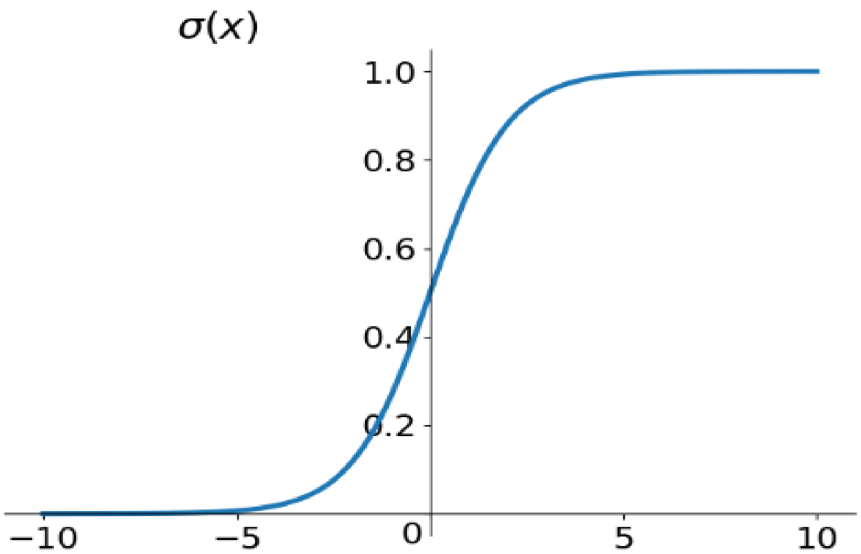
- Use activation functions to increase nonlinearity

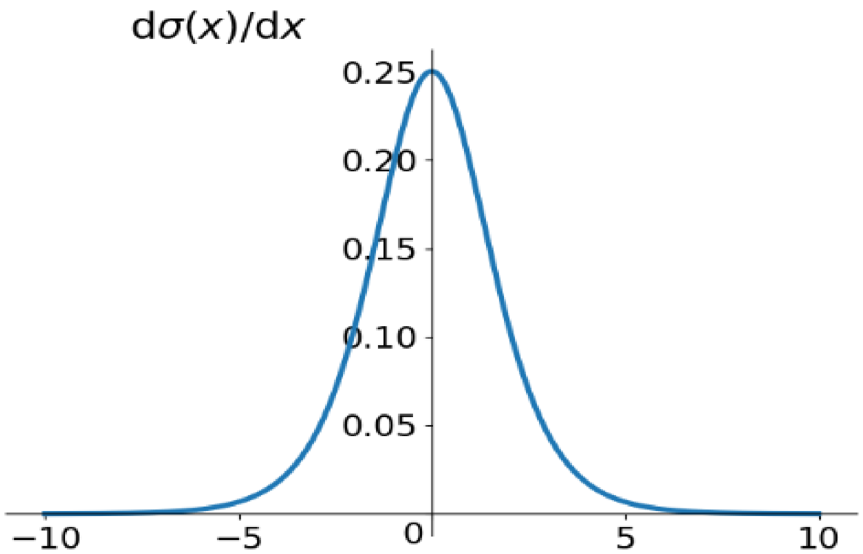
Common Activation Functions

Sigmoid Function (1/2)

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$d\sigma(x)/dx = \sigma(x)(1-\sigma(x))$$





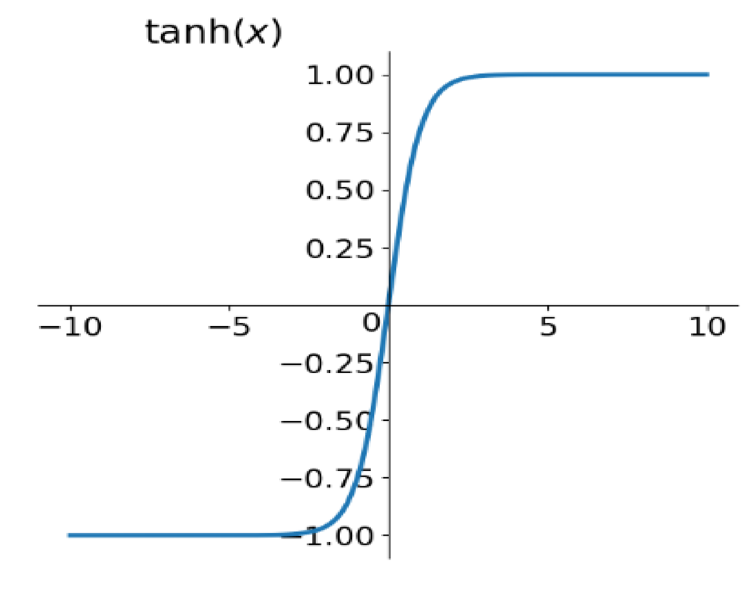
- Most commonly used in logistic regression
  - ranges from 0 to 1

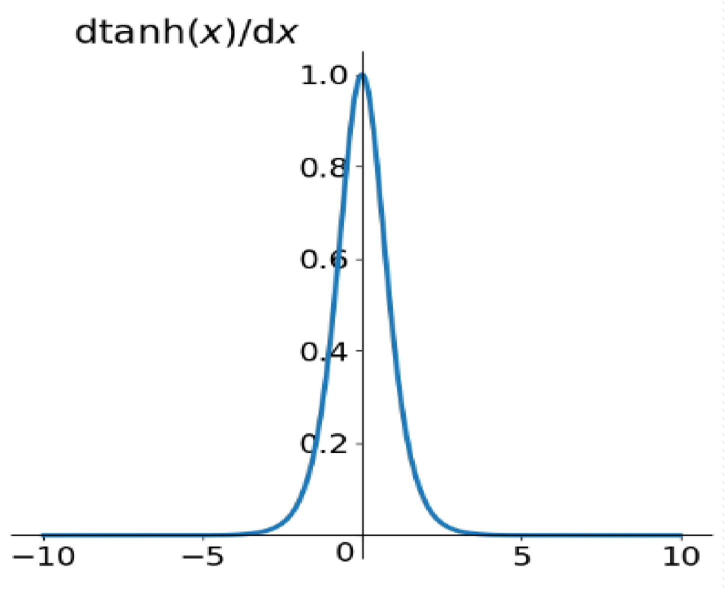
Sigmoid Function (2/2)

- Pros
  - continuous function, easy to find derivative
  - has upper and lower bound

- Cons
  - computationally complex due to exponential function and division operation
  - gradient vanishing
  - output is not zero-centered (always > 0)

Tanh Function (1/2)





$$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$

$$f'(x) = 1 - f(x)^2$$

- Variant of sigmoid function
  - tanh(x) = 2σ(2x) - 1

Tanh Function (2/2)

- Pros
  - alleviate zero-centered problem in sigmoid function

- Cons
  - still computationally complex
  - gradient vanishing as well

Softmax Function (1/2)

- $softmax(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}, for\ i = 1, 2, \dots, N$
- Most commonly used in multi-class classification
  - output value ranges from 0 to 1
  - sums to 1 for each input vector (probability distribution)

Input pixels, $\mathbf{x}$

Forward propagation $\longrightarrow$

Shape: (3, 32, 32)

Feedforward output, $\mathbf{y}_i$

cat	dog	horse
5	4	2
4	2	8
4	4	1

Shape: (3,)

Softmax function $\longrightarrow$

Softmax output, $\mathbf{S}(\mathbf{y}_i)$

cat	dog	horse
0.71	0.26	0.04
0.02	0.00	0.98
0.49	0.49	0.02

Shape: (3,)

Softmax Function (2/2)

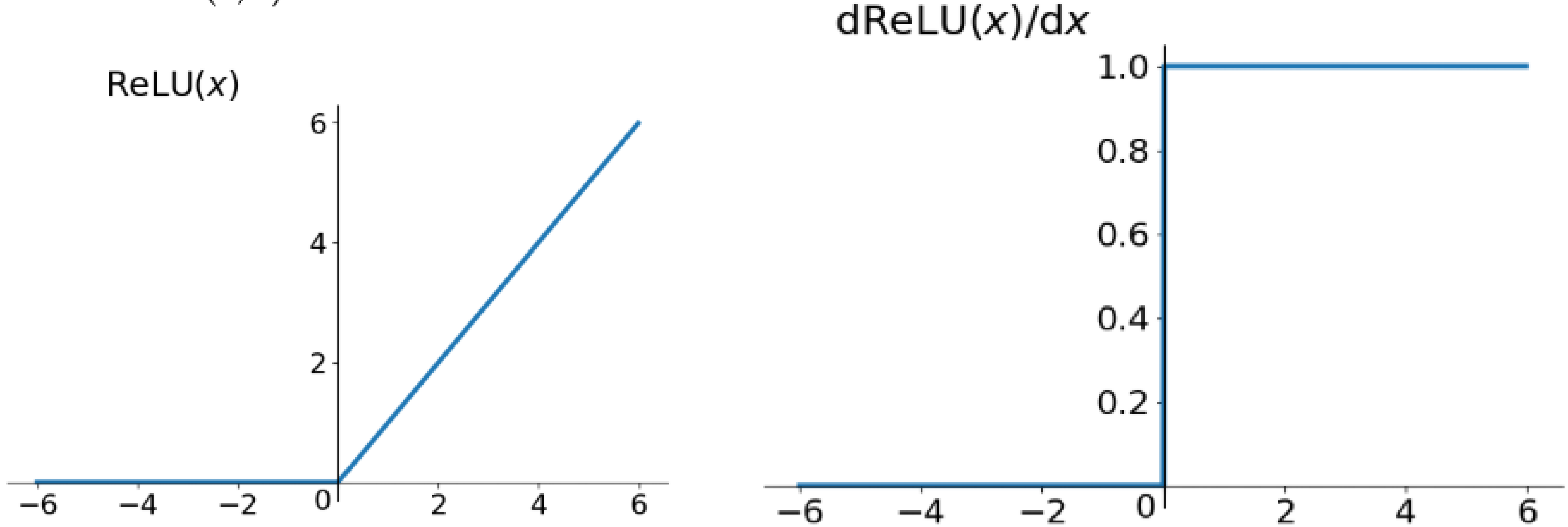
- Pros
  - continuous function, easy to find derivative
  - map output of data to probability distribution
  - used in the final layer of a classifier

- Cons
  - computationally complex

ReLU Function (1/2)

- ReLU (Rectified Linear Unit)

$$\mathrm{ReLU} = \max(0, x)$$



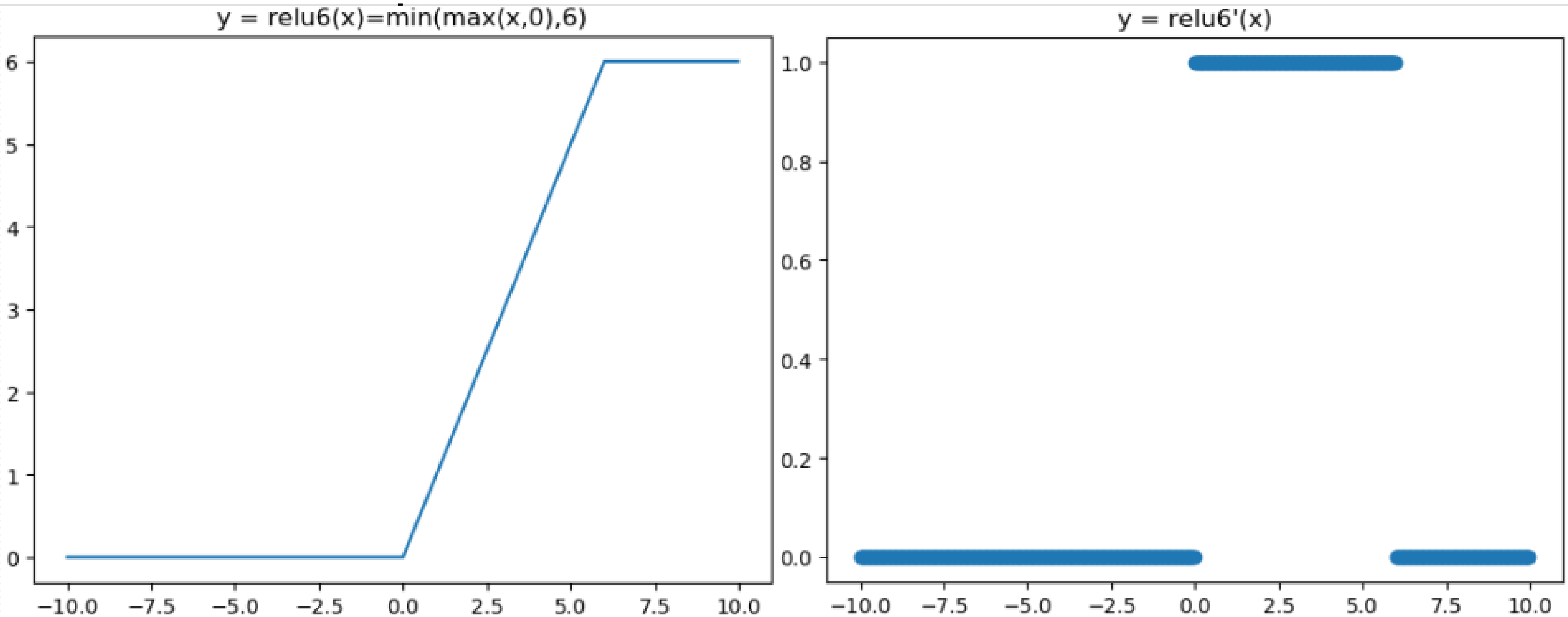
- Takes the maximum value of input
  - ranges from 0 to +∞

ReLU Function (2/2)

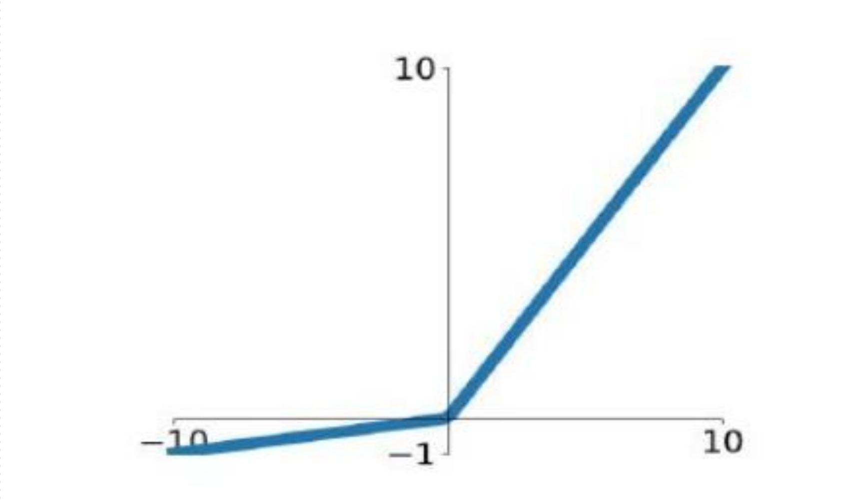
- Pros
  - high computation speed
  - alleviate gradient vanishing problem in positive interval
  - converge quickly when training
- Cons
  - dead ReLU problem (dying neuron issue)
    - output is killed in negative interval
    - gradient = 0 when input < 0

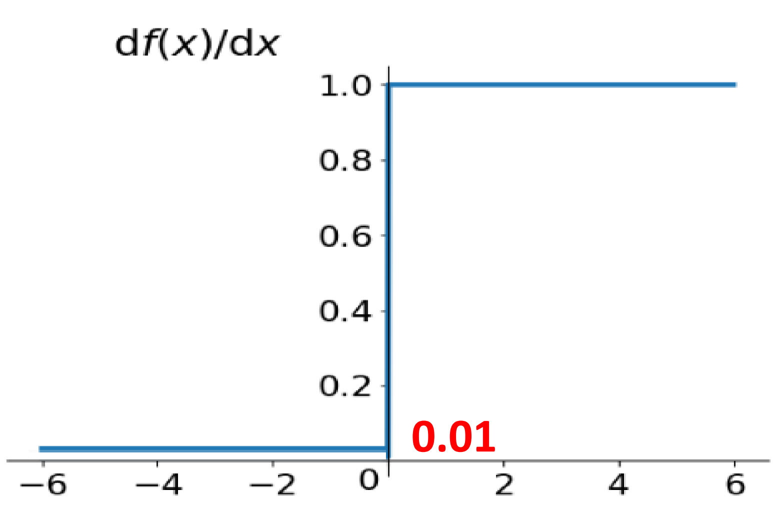
ReLU6 Function

- Like ReLU, but has maximum value 6
- Large range of output values may exceed the floating-point precision of the processor
  - increase robustness when used with low-precision



Leaky ReLU Function (1/2)





- $LeakyReLU(x) = \begin{cases} x & , x \geq 0 \\ 0.01x, & x < 0 \end{cases}$
  - 0.01 is a hyperparameter
- Leaky ReLU alleviates dead ReLU problem in ReLU
  - preserve small gradients in negative interval

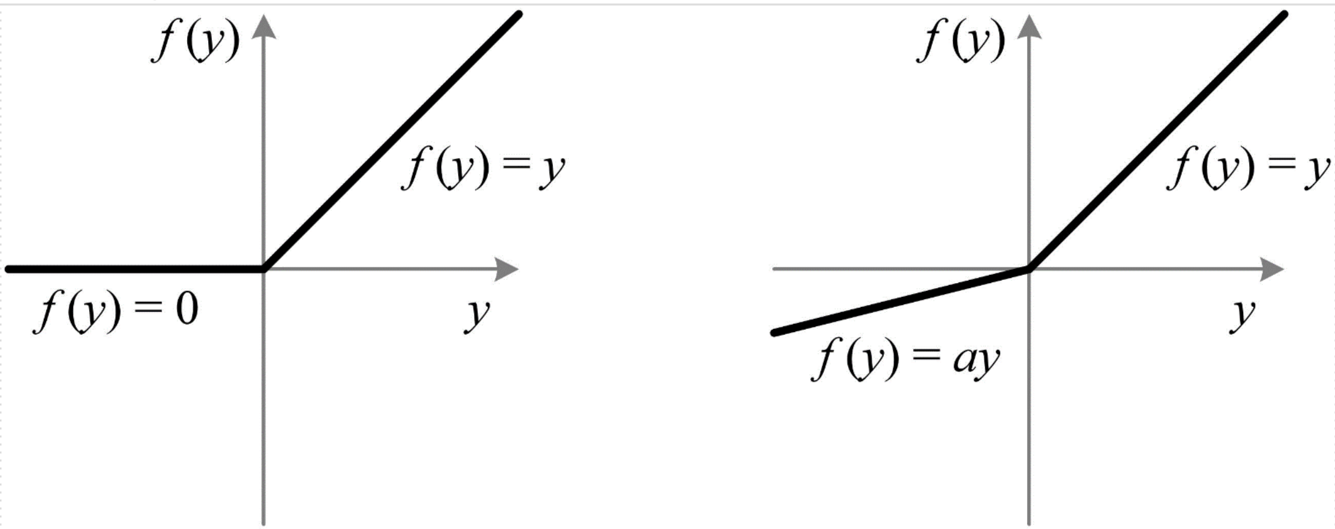
Leaky ReLU Function (2/2)

- Theoretically, Leaky ReLU has all advantages of ReLU
  - introduces similar nonlinearity as ReLU without dead ReLU problem

- Some papers indicate Leaky ReLU is not always better than ReLU

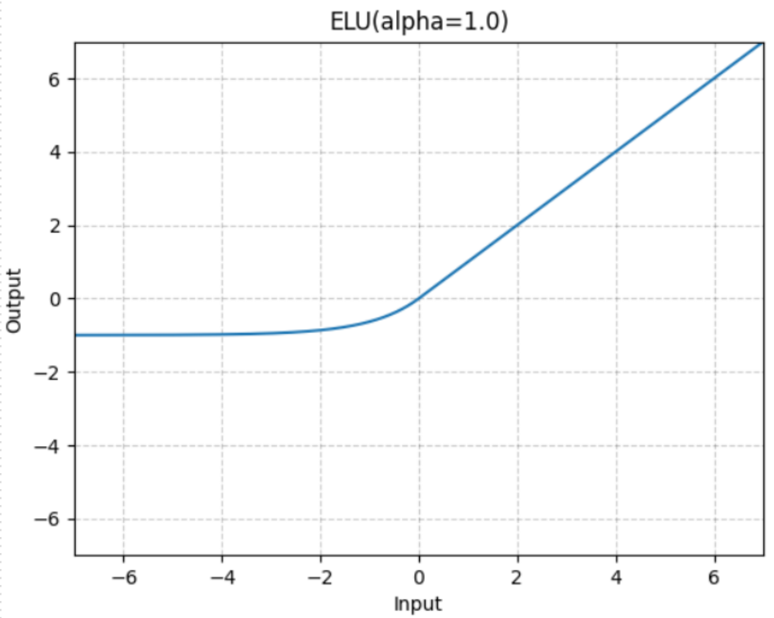
PReLU Function

- **Parametric** Rectified Linear Unit
- $PReLU(x) = \begin{cases} x \ \ , x \geq 0 \\ ax, x < 0 \end{cases}$
- $\boldsymbol{a}$ is a learnable parameter
- Different layers may require different types of nonlinearity



ELU Function

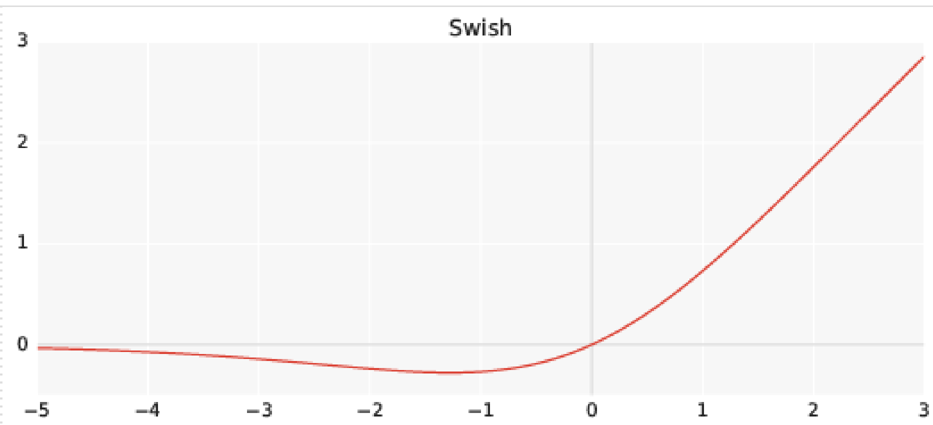
- **Exponential** Linear Unit
- $ELU(x) = \begin{cases} x & , x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$
  – α is a hyperparameter

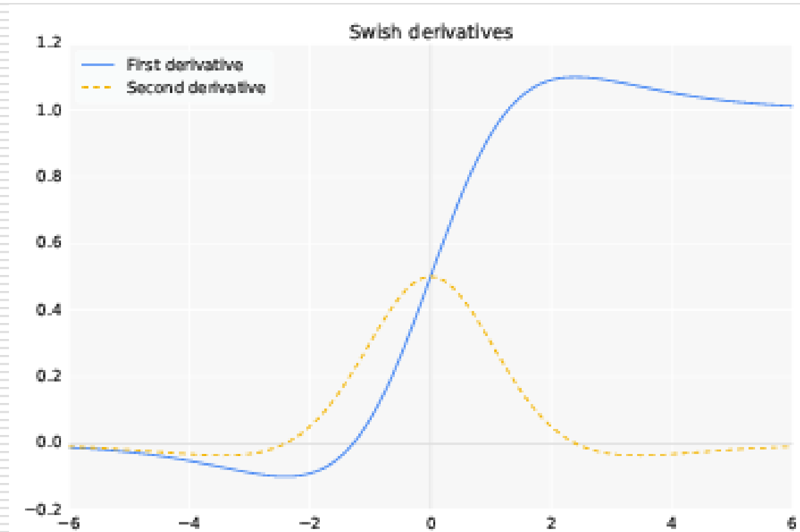


- Alleviate dead ReLU problem in ReLU
  – preserve small gradients in negative interval
  – nonlinear in negative interval

Swish (SiLU) Function

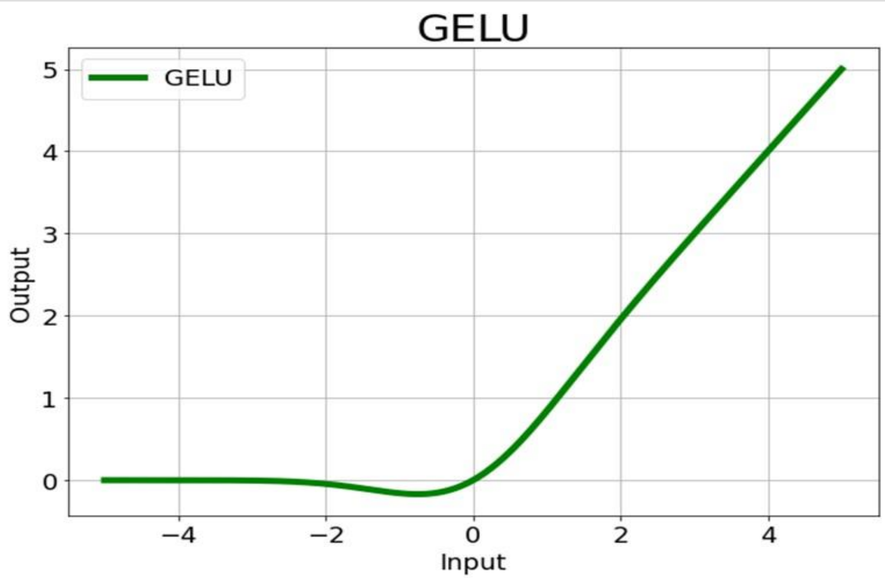
- **Sigmoid** Linear Unit (SiLU): a special case of Swish
- $Swish(x) = x \cdot sigmoid(\beta x)$
- $SiLU(x) = x \cdot sigmoid(x)$
- Alleviate dead ReLU problem in ReLU
  - preserve small gradients in negative interval
  - smooth and non-monotonic function





GELU (Gaussian Error Linear Unit)

- Provide well defined gradient for negative inputs
  - **Alleviate dying neuron issue**
- Widely used in various **Transformer-based** models
- Computationally expensive



original function:

$$GELU(x) = x \times CDF(x) = x \times \frac{1}{2}\left(1 + erf\left(\frac{x}{\sqrt{2}}\right)\right)$$

approximate function:

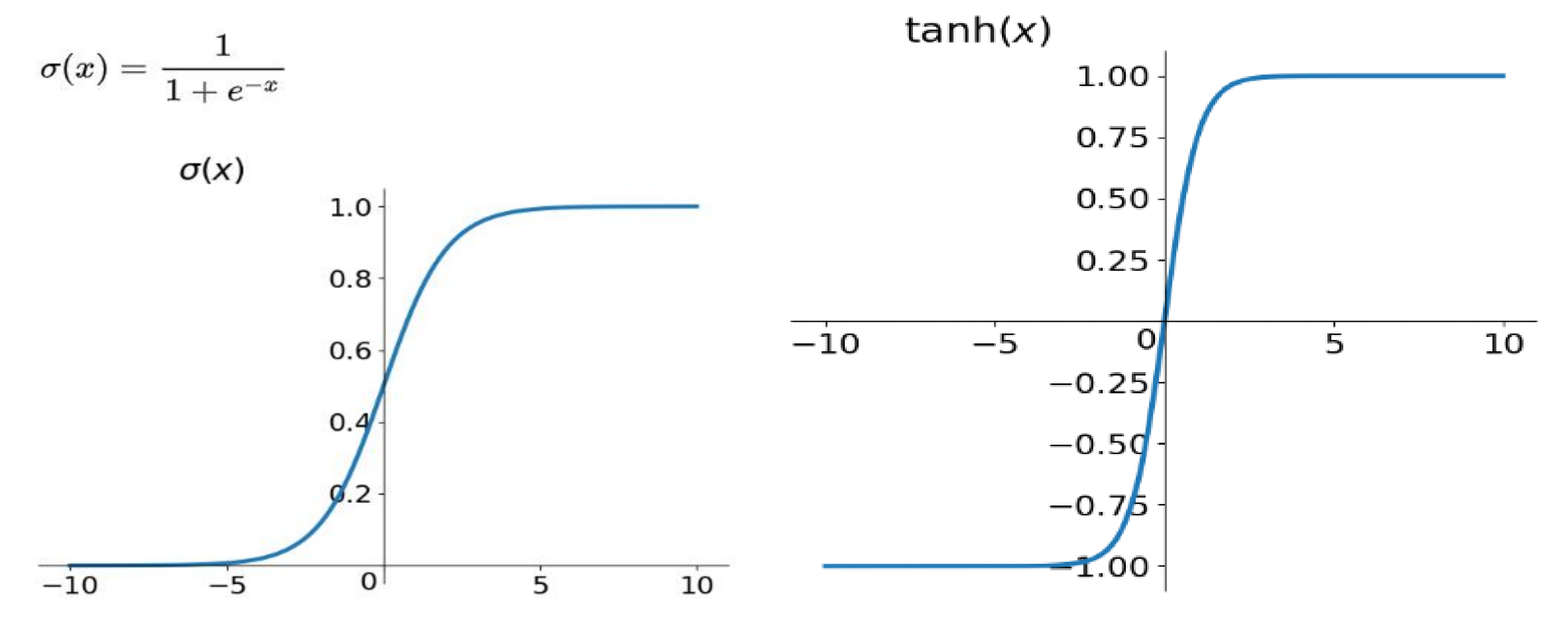
$$GELU_{\tanh}(x) = 0.5x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$$

Which One Is Popular?

- In most CNN applications, ReLU is most commonly used
  - high computation speed
  - converge quickly
  - preserve gradient
- In most Transformer applications, GELU is most commonly used

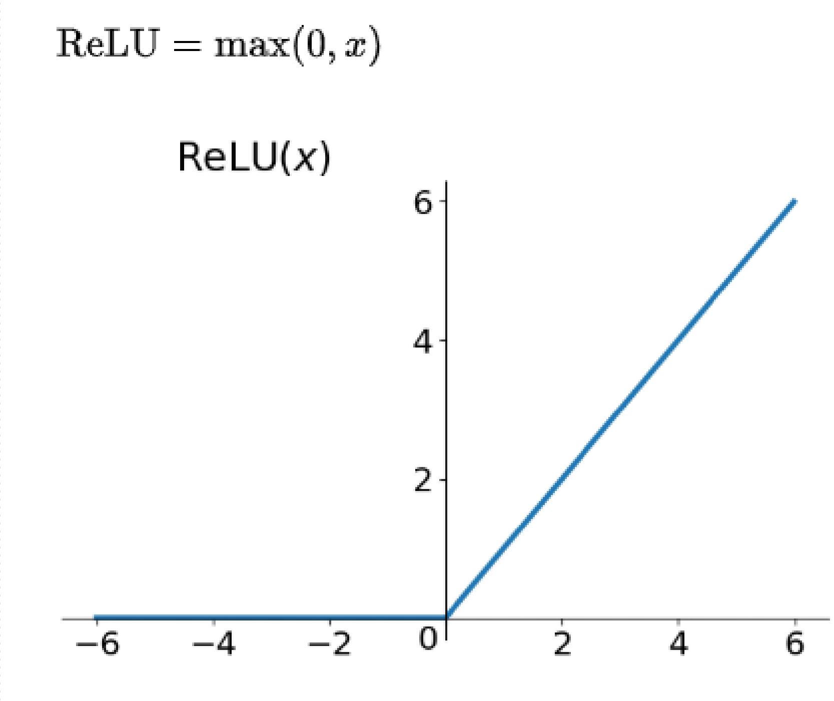
Activation Functions in PyTorch (1/4)

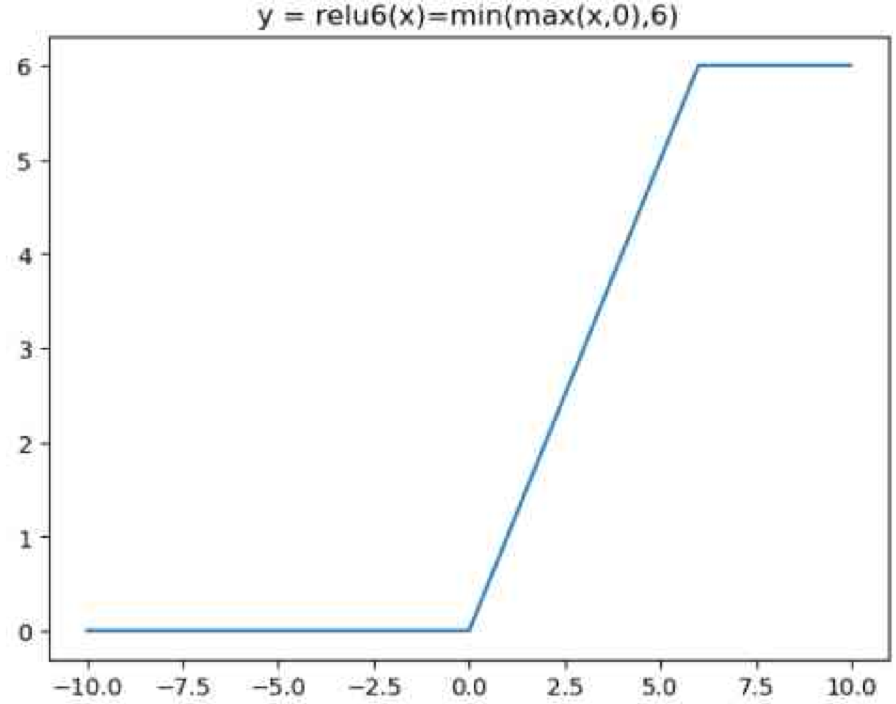
- nn.Sigmoid()
- nn.Tanh()
- nn.Softmax(dim=-1)
  - dim: every slice along dim will sum to 1



Activation Functions in PyTorch (2/4)

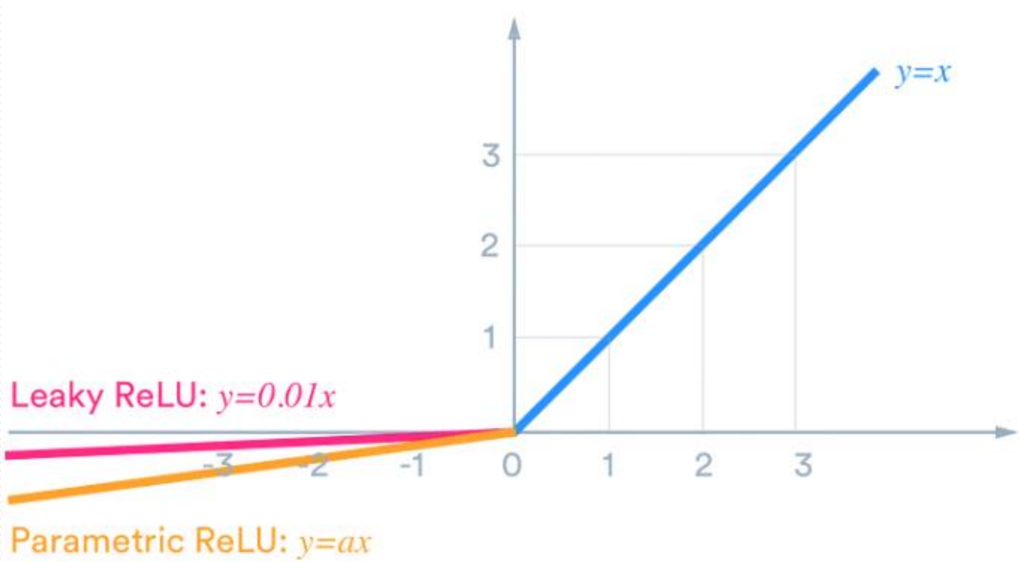
- nn.ReLU()
- nn.ReLU6()

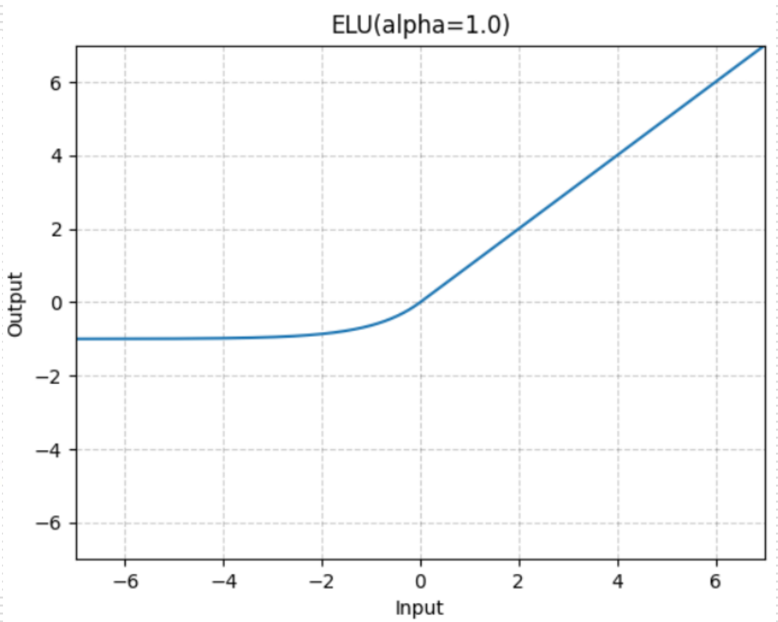




Activation Functions in PyTorch (3/4)

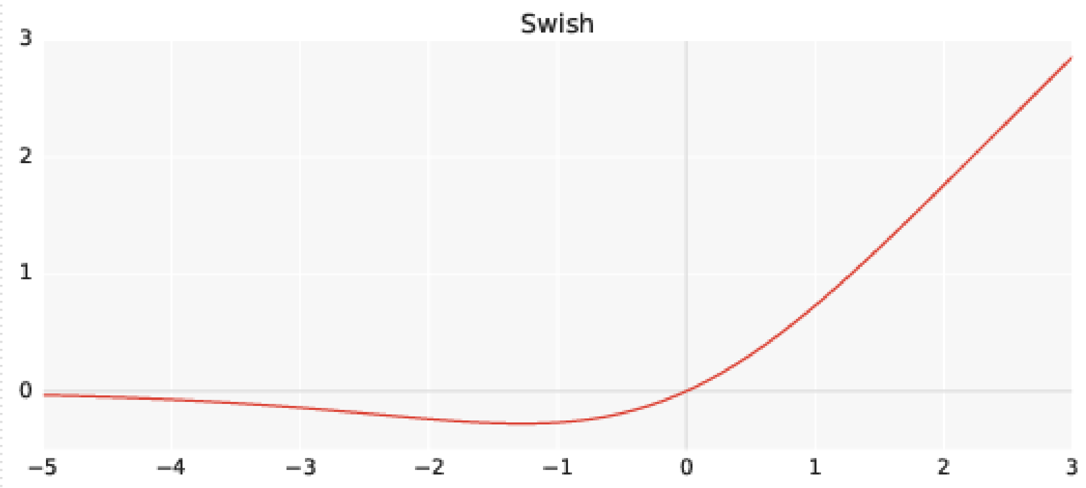
- nn.LeakyReLU (negative_slope=0.01)
- nn.PReLU(num_parameters=1, init=0.25)
  – num_parameters: number of a to learn
  – init: the initial value of a
- nn.ELU(alpha=1.0)



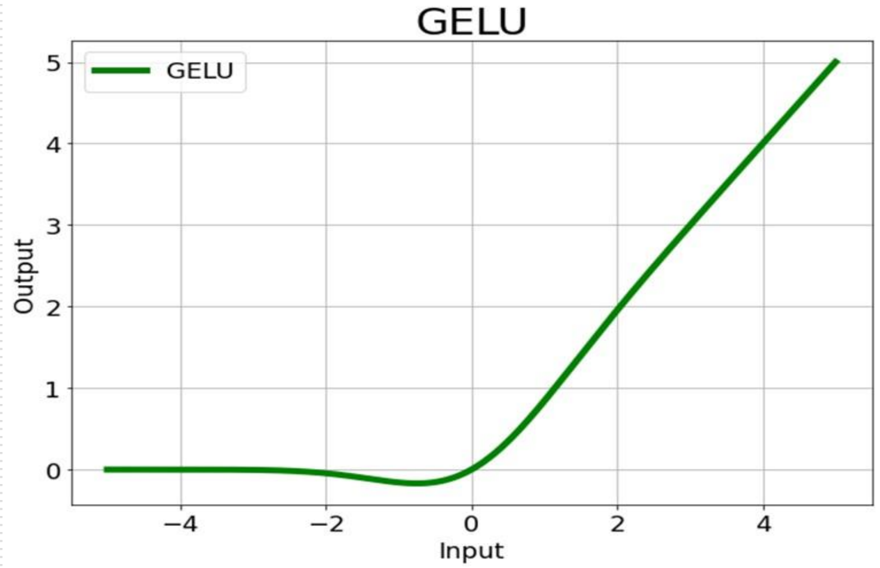


Activation Functions in PyTorch (4/4)

- nn.SiLU()

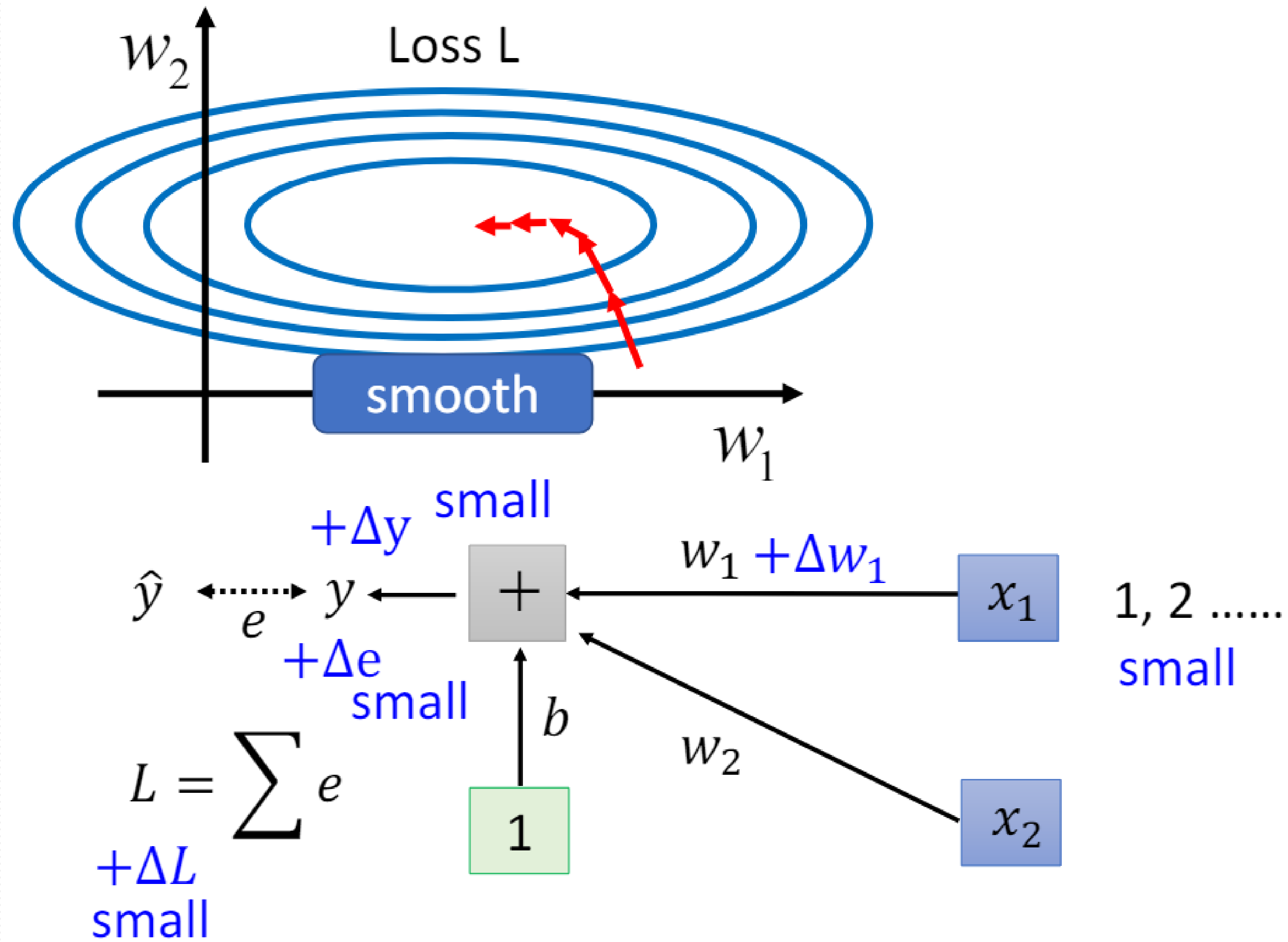


- nn.GELU()

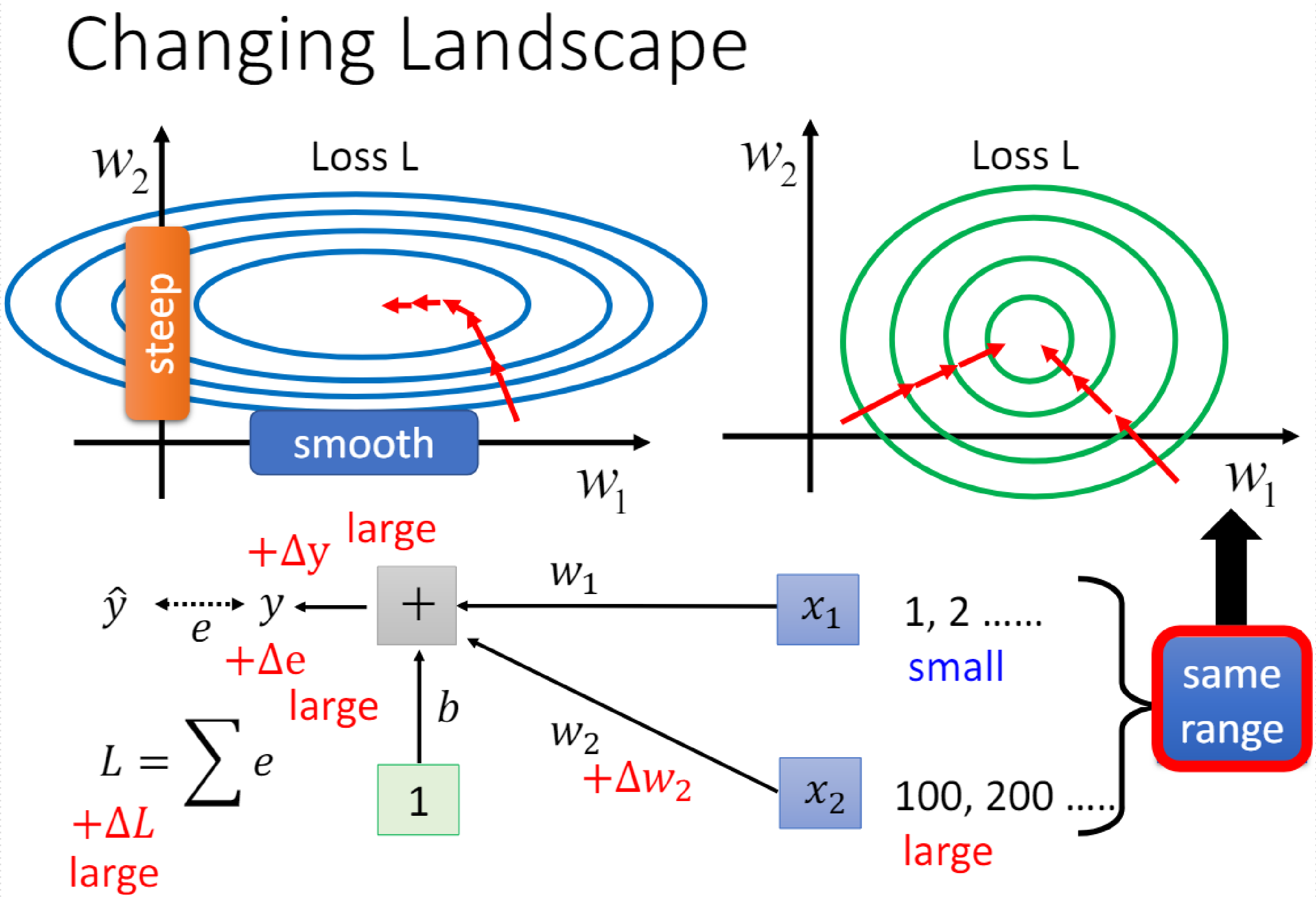


Introduction to Feature Normalization

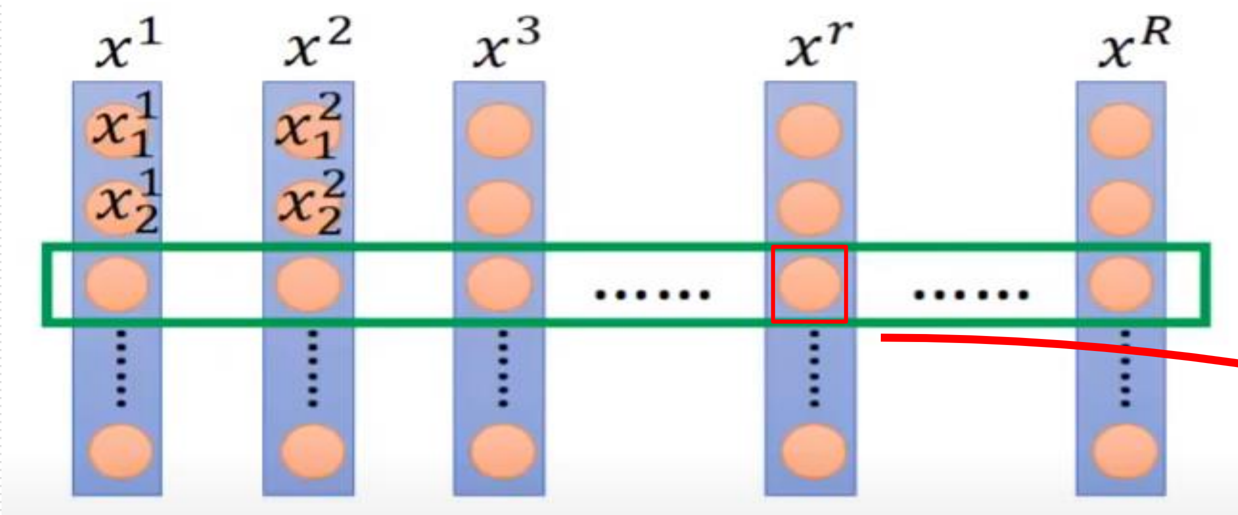
Changing Landscape



Changing Landscape



Feature Normalization (1/2)

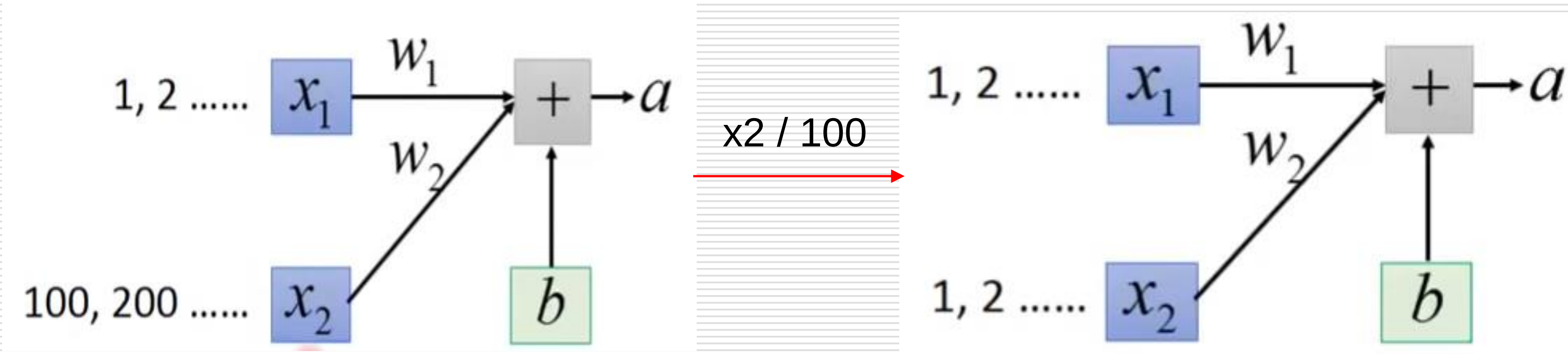


i_{th} *dimension*

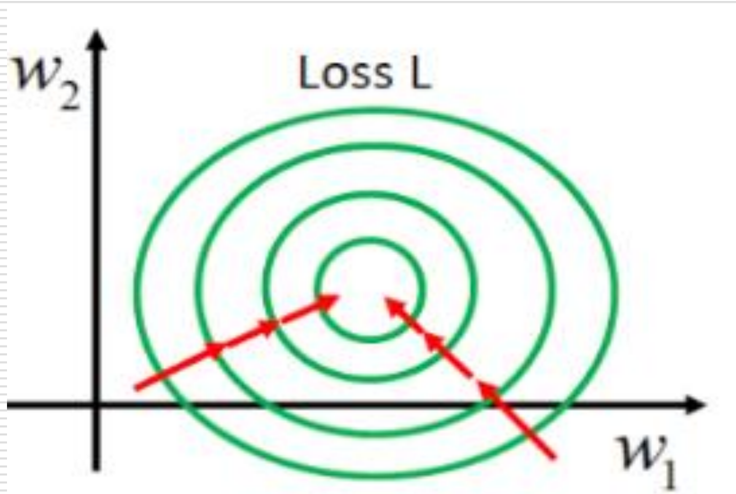
- For each dimension i
  - calculate mean(m_i) and standard deviation(σ_i)
- Normalize every element using m_i and σ_i

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

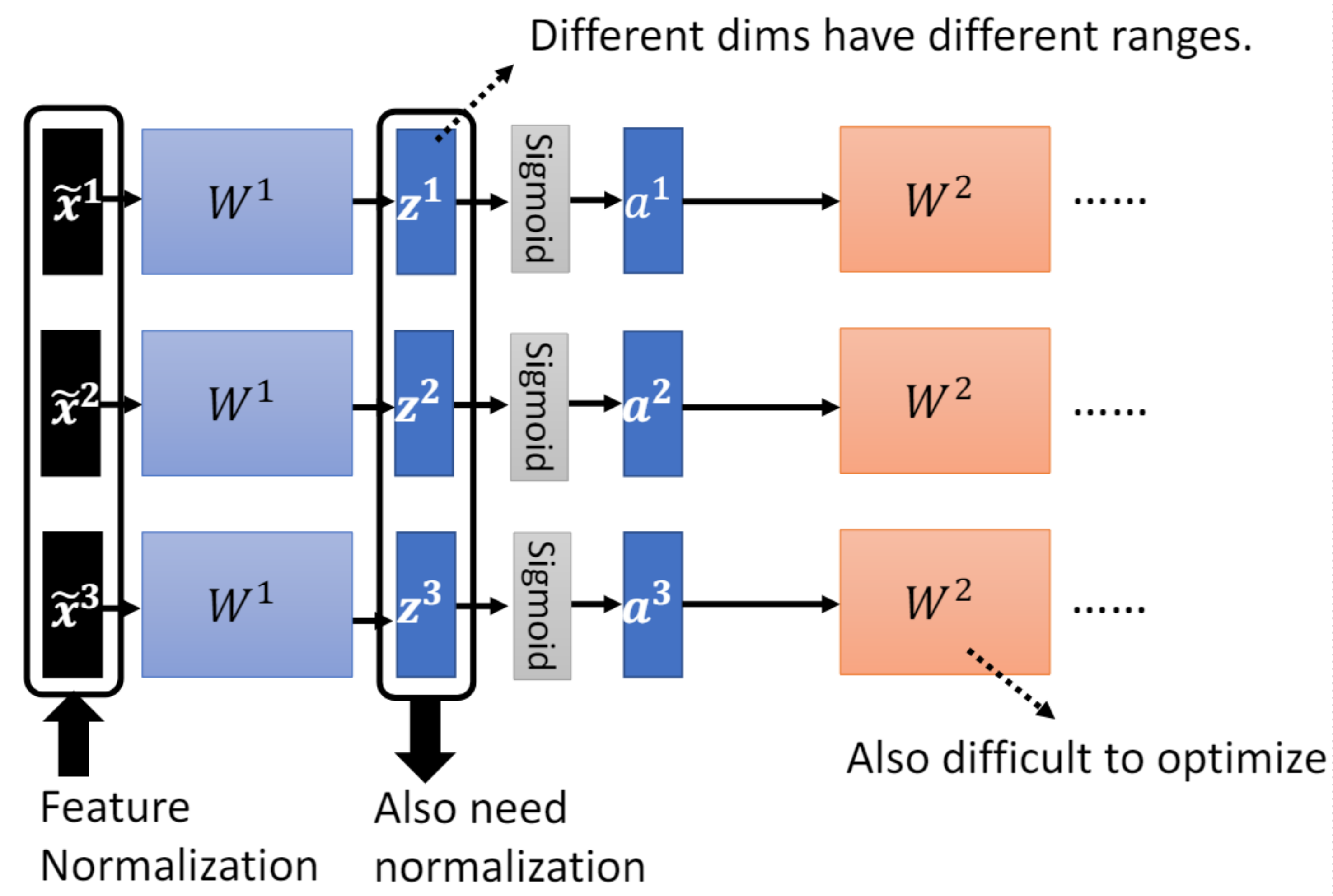
Feature Normalization (2/2)



- Makes x1 and x2 have rough same range
  - w1 and w2 will have same impact on Loss function



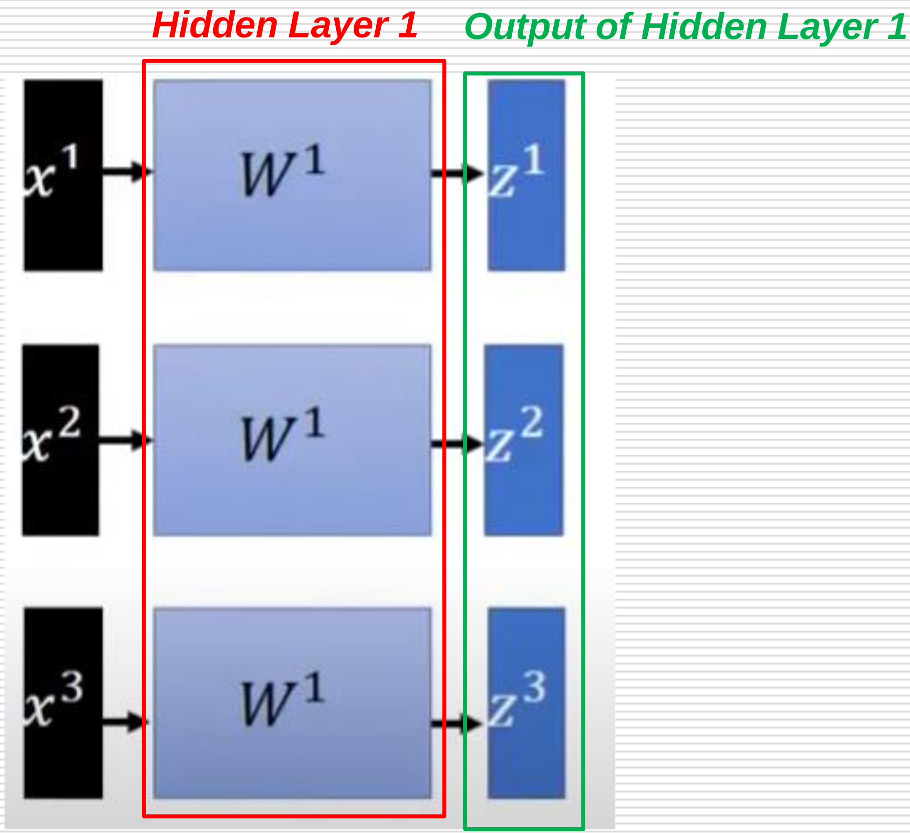
Considering Deep Learning



Batch Normalization Steps

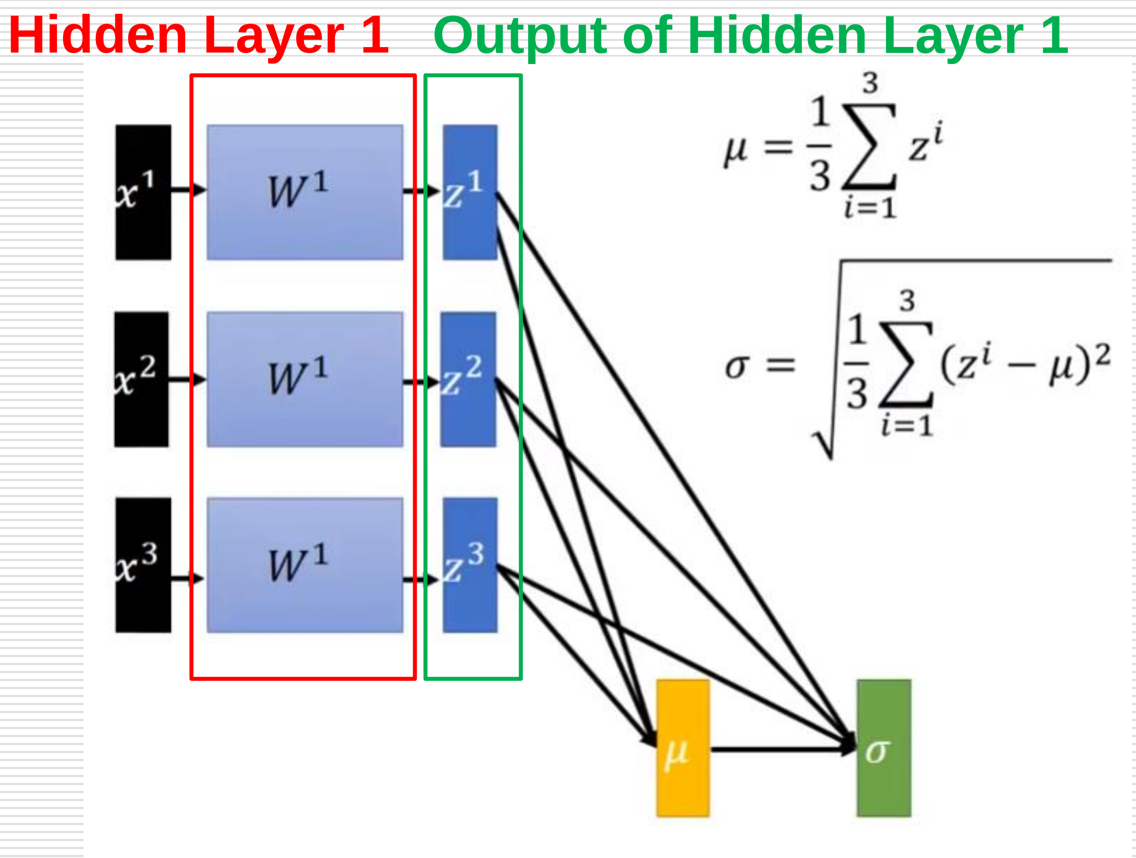
Batch and Batch Size

- In a training process, we train a batch (set) of data instead of a single data at once
  - train x^1, x^2, x^3 at the same time ➔ batch size =3



Batch Normalization (1/4)

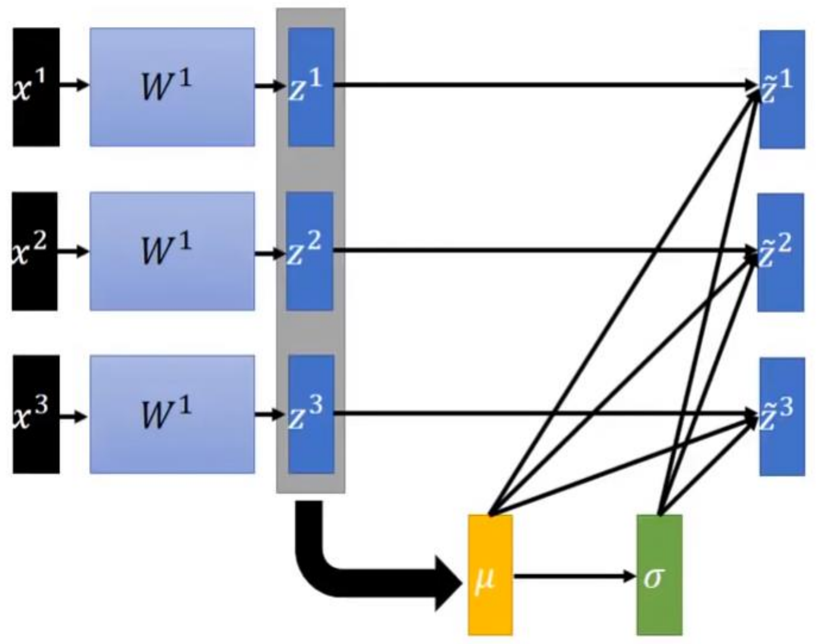
- Target: Normalize output of hidden layer 1 (z^1, z^2, z^3)
- Step 1: calculate mean (μ) and standard deviation (σ) in every batch (z^1, z^2, z^3)



Batch Normalization (2/4)

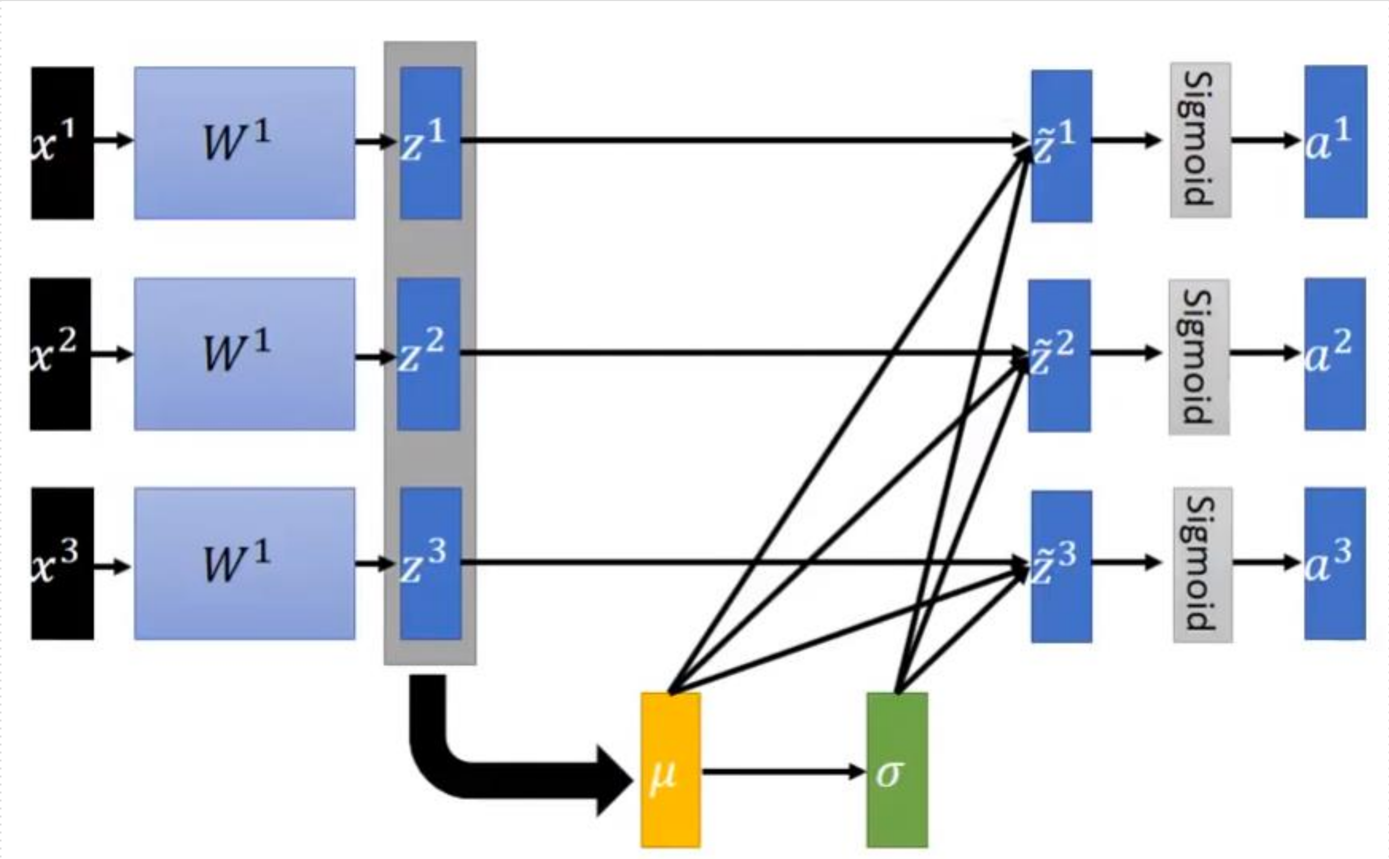
- Step 2: Normalize z^1, z^2, z^3 with mean, standard deviation
  - will get mean = 0 and deviation = 1 after Step 2

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



Batch Normalization (3/4)

- Step 3: Send normalized value to activation function and next layer



Batch Normalization (4/4)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1 \ldots m}\}$;
Parameters to be learned: γ, β

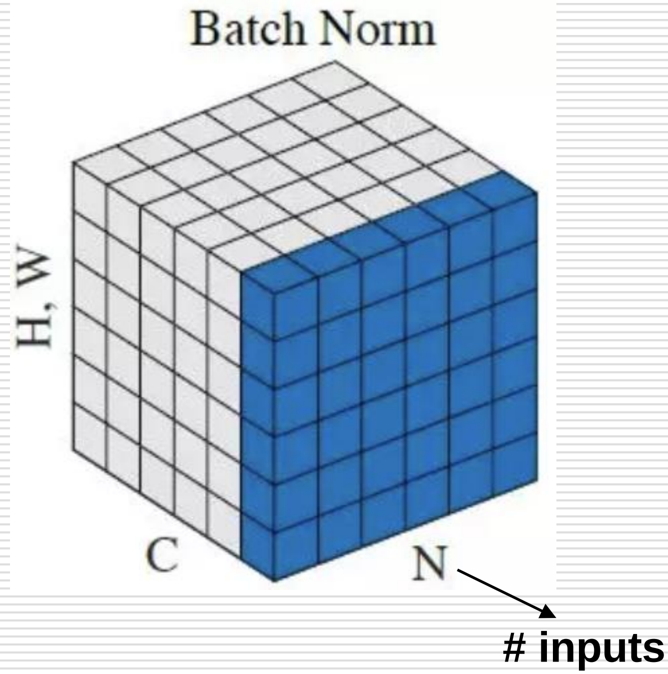
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



- When back propagation, impact of z^1, z^2, z^3 on mean and deviation will also be passed on
  - mean and deviation are also training parameters

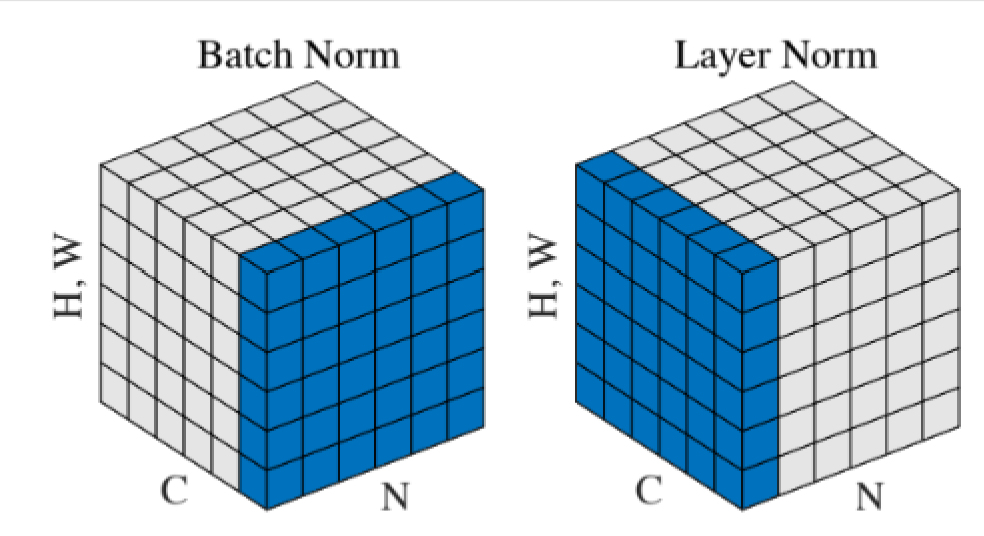
Benefits of Batch Normalization

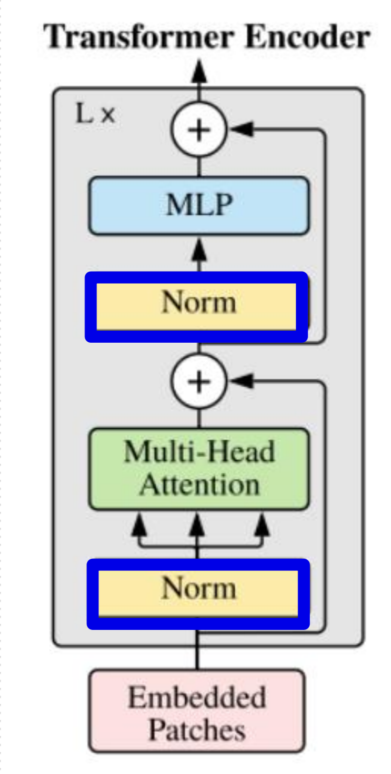
- Reduces training time, and makes very deep net trainable
- Less vanishing gradient
- Learning is less affected by initialization

Other Normalization Methods

Layer Norm (LN)

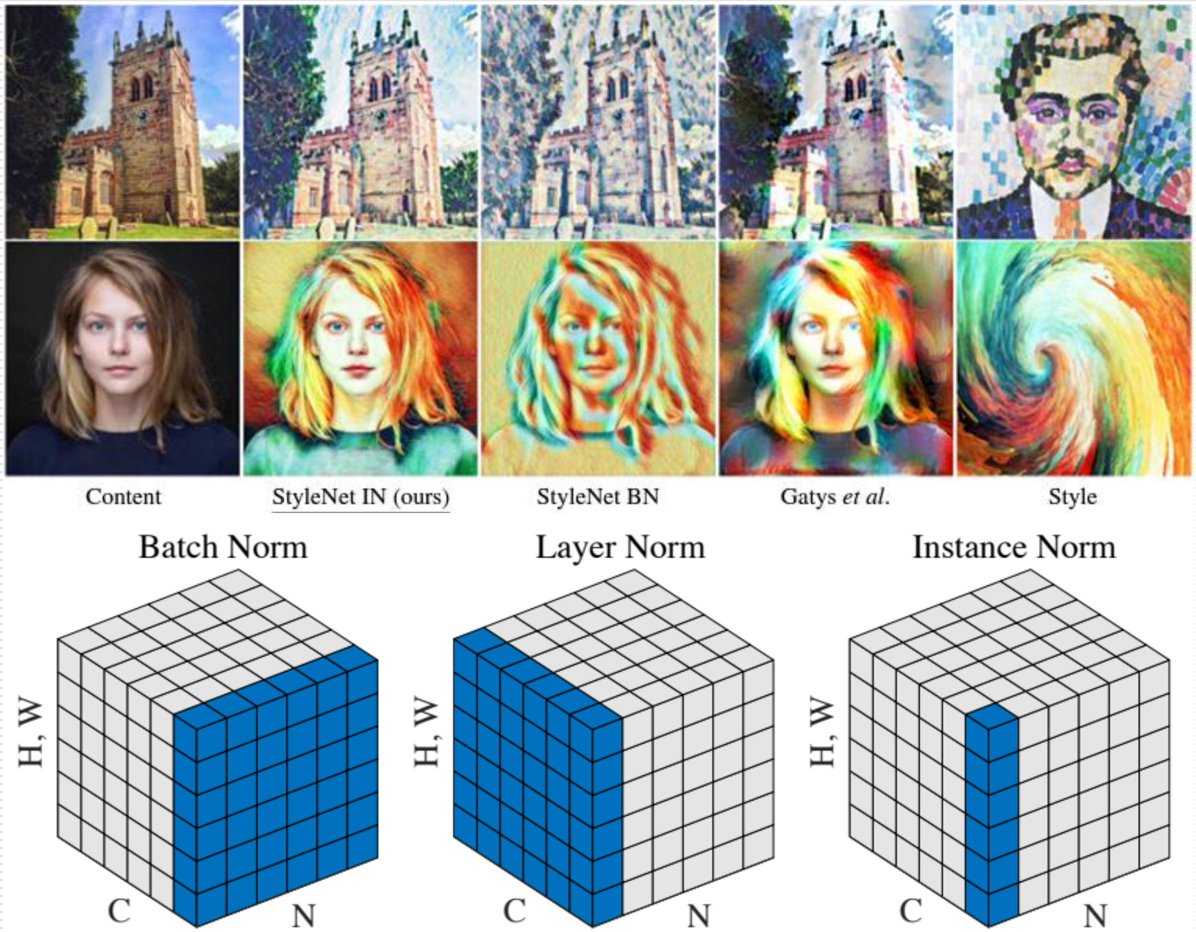
- Like Batch Norm, but normalize input feature map (channel) dimension instead of batch dimension
  - independent of batch size
  - widely used in Transformer-based models





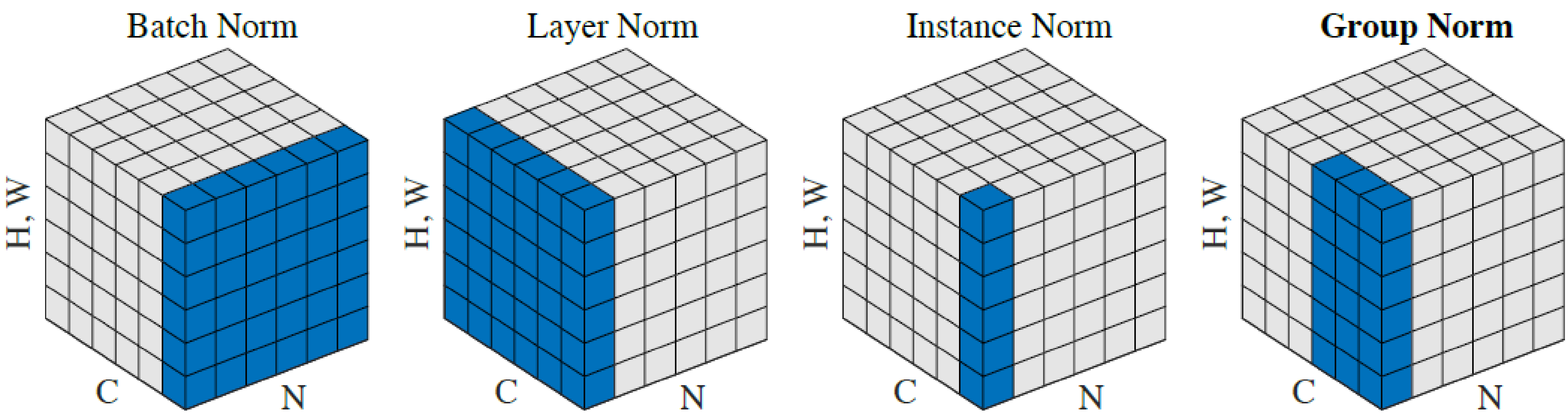
Instance Norm (IN)

- Normalize each individual content image
  - widely used in generator network for stylization task



Group Norm (GN)

- Separate the channels into groups, normalize each group of content images
  - between Instance Norm and Layer Norm
  - # of groups = 1 ➔ Layer Norm
  - # of groups = C ➔ Instance Norm



Normalization in PyTorch

- nn.BatchNorm1d(num_features)
- nn.BatchNorm2d(num_features)
- nn.BatchNorm3d(num_features)
- nn.LayerNorm(normalized_shape)
- nn.InstanceNorm1d(num_features)
- nn.InstanceNorm2d(num_features)
- nn.InstanceNorm3d(num_features)
- nn.GroupNorm(num_groups, num_channels)

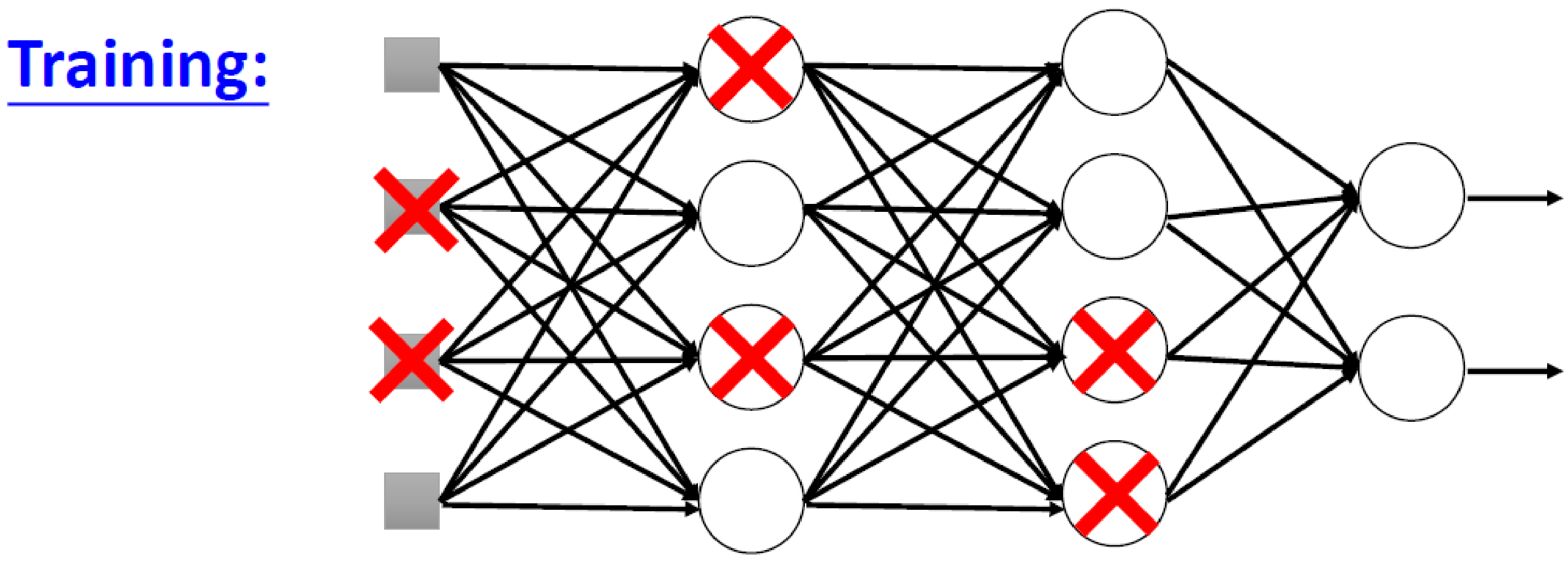
Dropout and Dropblock

Overfitting

- Think of it as over-training
  - caused by model that has too many parameters and is too complicated
  - feature learned by machine is close to training data
    - error becomes big when testing or validation

Dropout (1/5)

- Avoid overfitting in fully connected layer
  - discard hidden layer neurons every epoch with a certain probability when training
  - the discarded neurons won't transmit messages



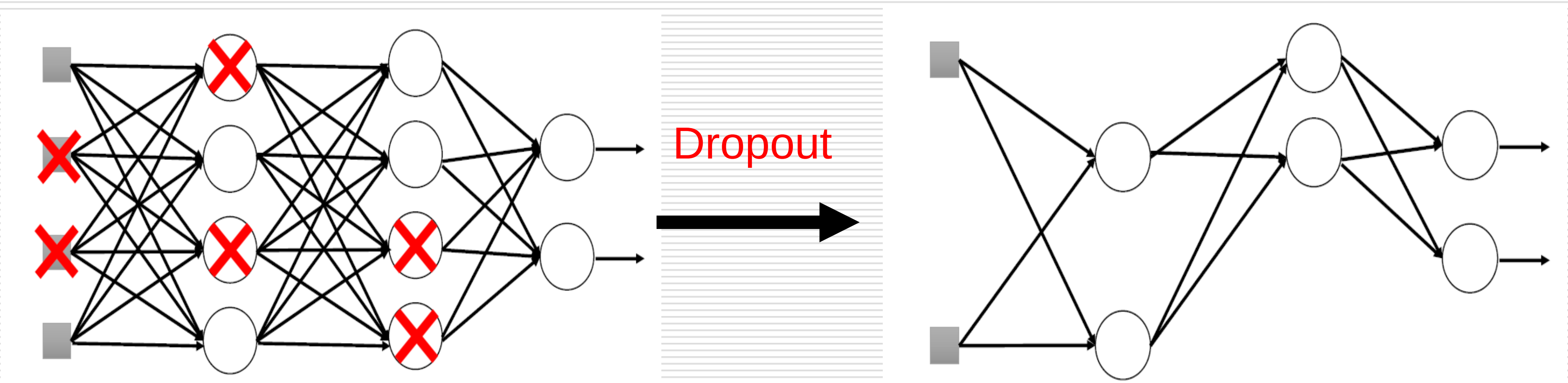
➢ **Each time before updating the parameters**

- Each neuron has p% to dropout

Dropout (2/5)

- Think of dropout as a way to reduce parameters
  - less parameters can avoid overfitting effectively
  - the whole model will become thinner
  - using the new network for training

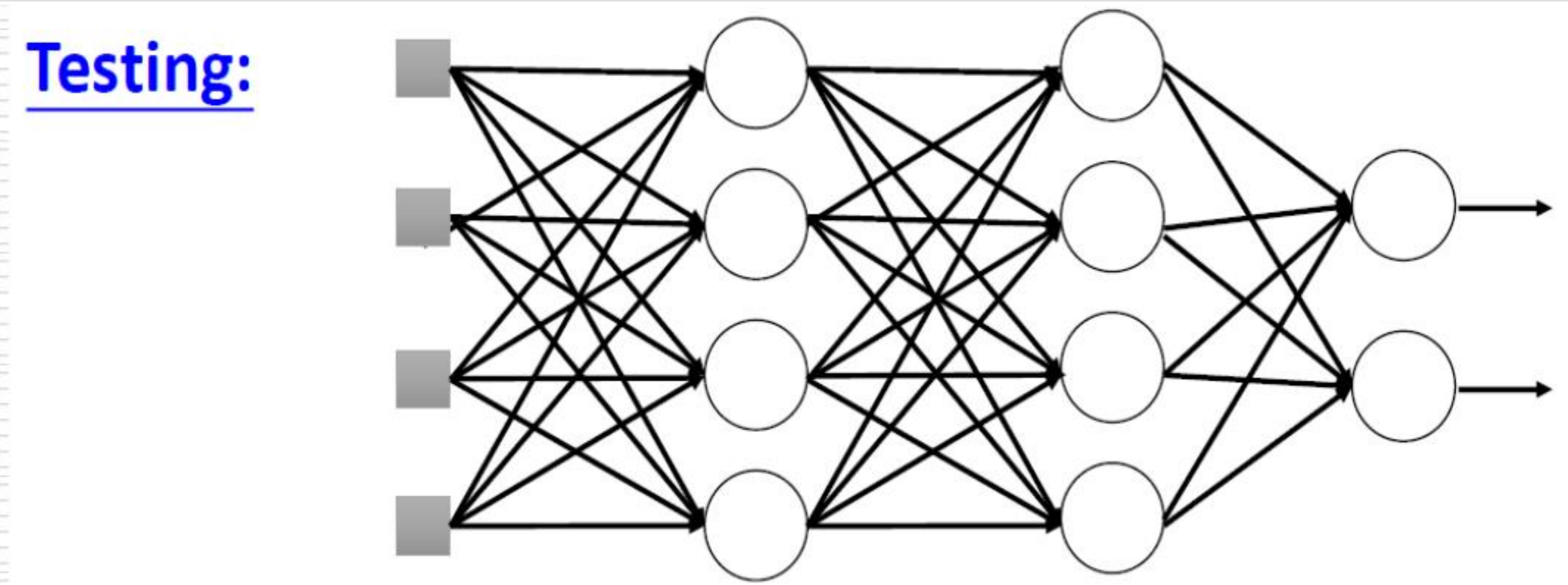
Thinner!!



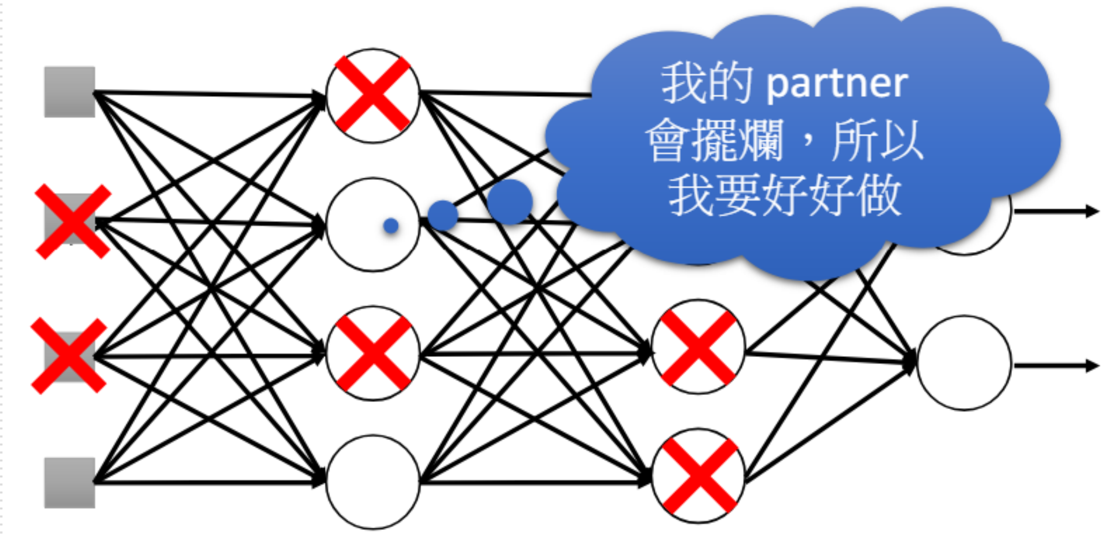
Dropout (3/5)

- In testing, use the non-dropout model to inference
  - if the dropout rate is p%, all weights times 1-p%
  - assume that the dropout rate is 50%

    if a weight w = 1 by training, set w = 0.5 for testing



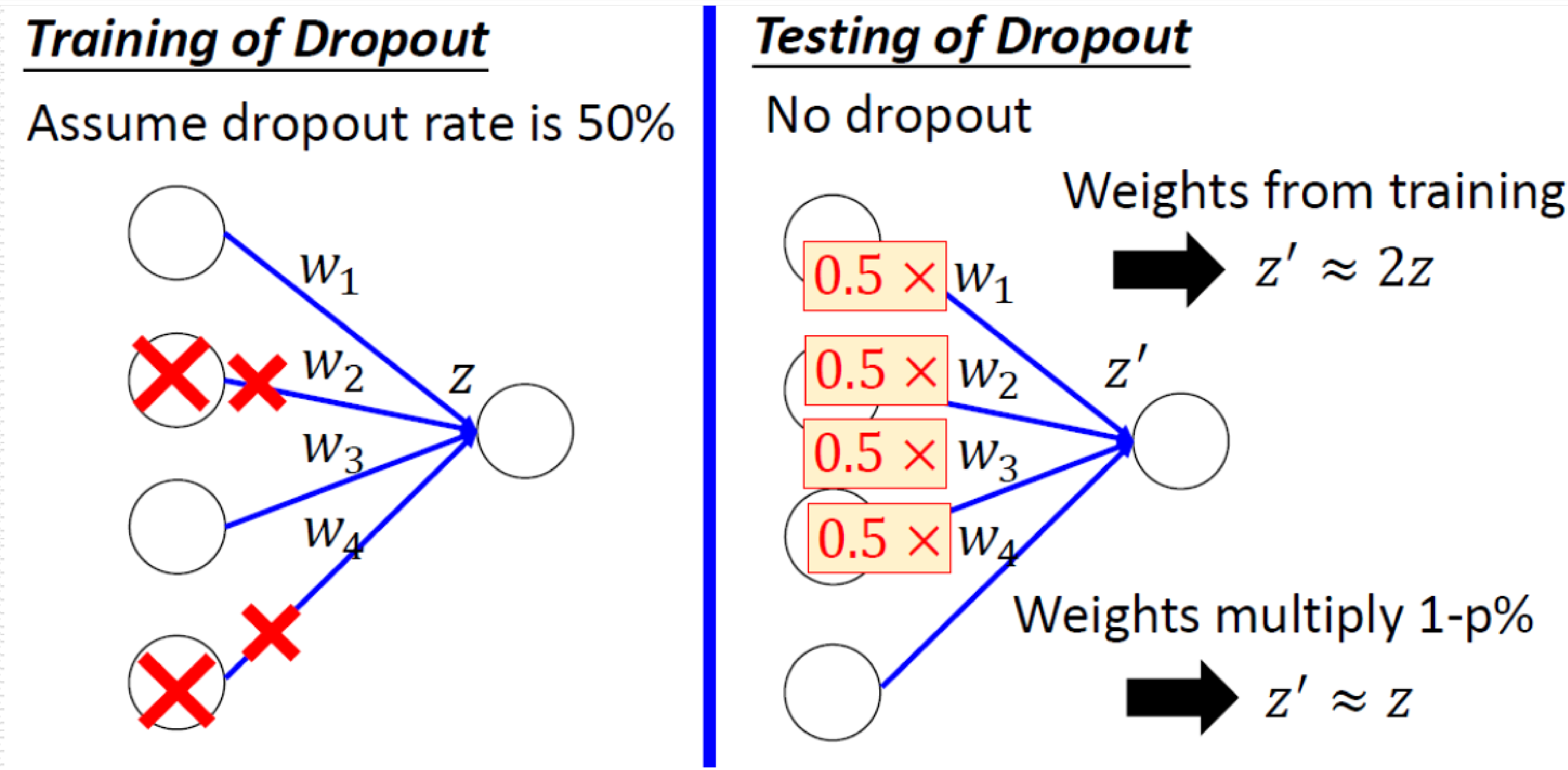
Dropout (4/5) – Intuitive Reason



- When teams up, if everyone expect the partner will do the work, nothing will be done finally
- However, if you know your partner will dropout, you will do better
- When testing, no one dropout actually, so obtaining good results eventually

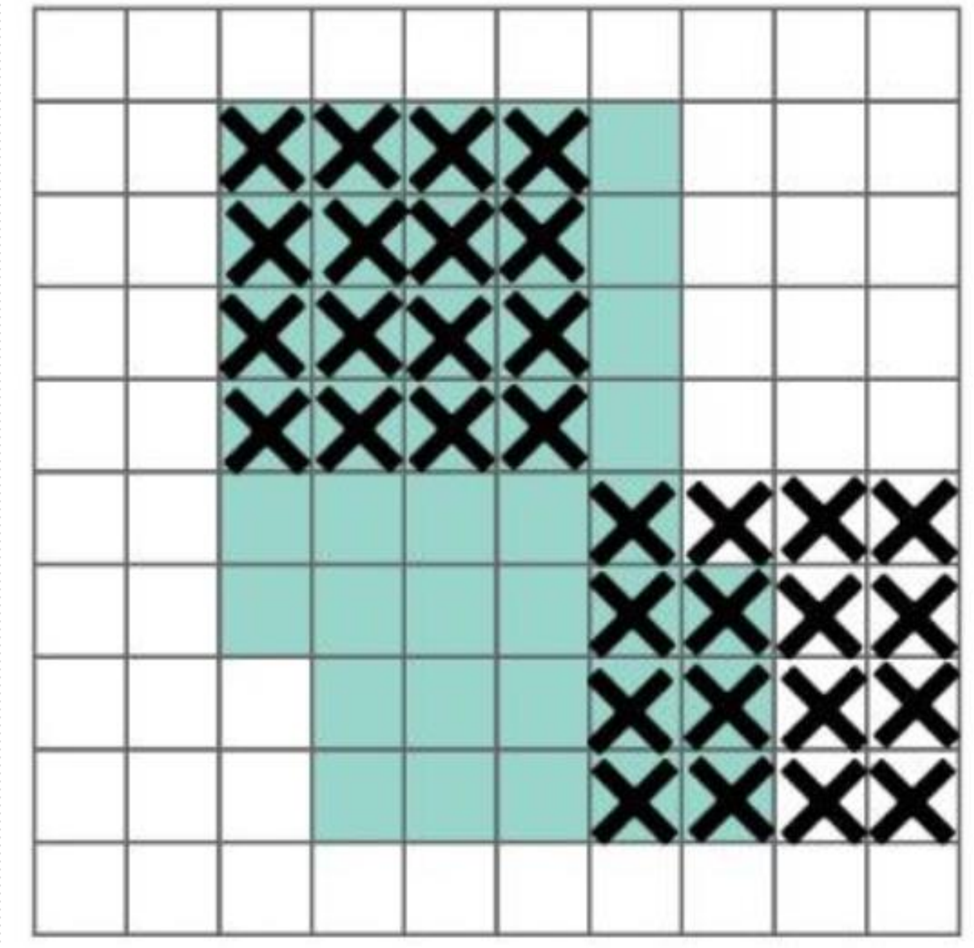
Dropout (5/5) – Intuitive Reason

- Why weights should multiply 1-p% when testing?
  - keeping the expected value of the output unchanged



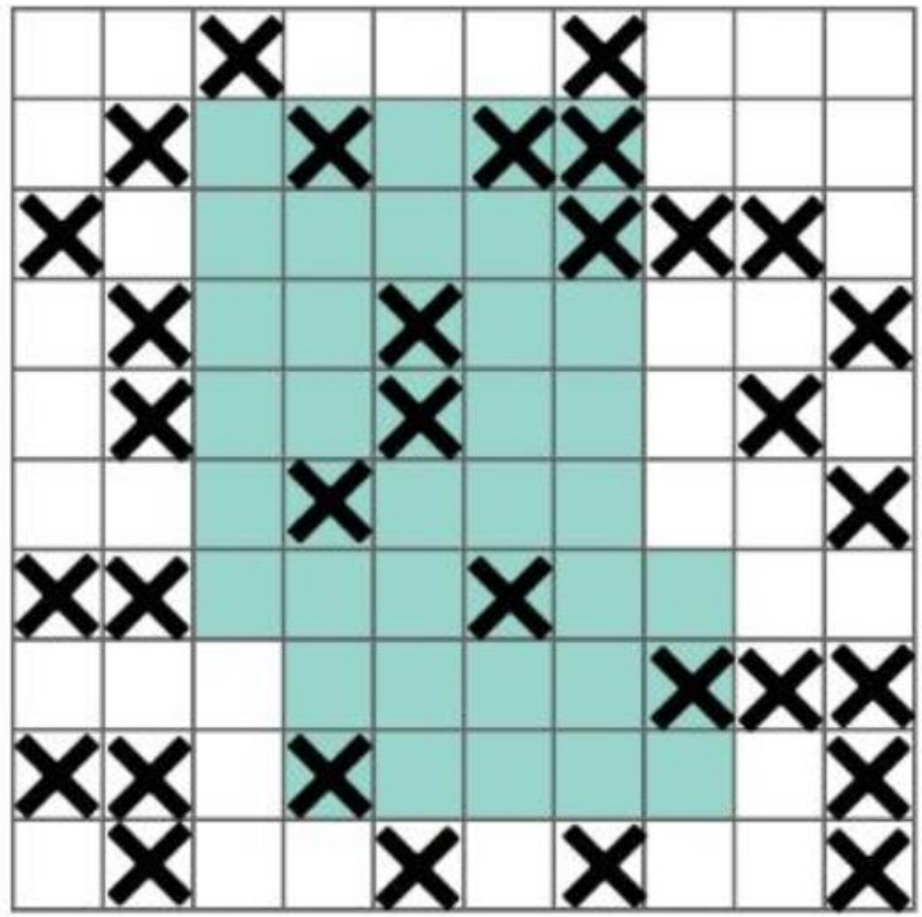
DropBlock (1/2)

- Similar to Dropout, but drops features in CNN kernel
  - randomly drop an element and its neighbors



DropBlock (2/2)

- What if we don't drop away adjacent elements?
  - after max or average pooling layer, effect of dropout may be canceled



Dropout and DropBlock in PyTorch

- Dropout: nn.Dropout(p=0.5)
  - p: dropout rate
  - any input shape is allowed

- DropBlock: torchvision.ops.drop_block2d(input, p=0.5, block_size=3)
  - input: input tensor with shape (N, C, H, W)
  - p: dropblock rate
  - block_size: size of the block to drop

Introduction to Loss Functions

MAE (Mean Absolute Error)

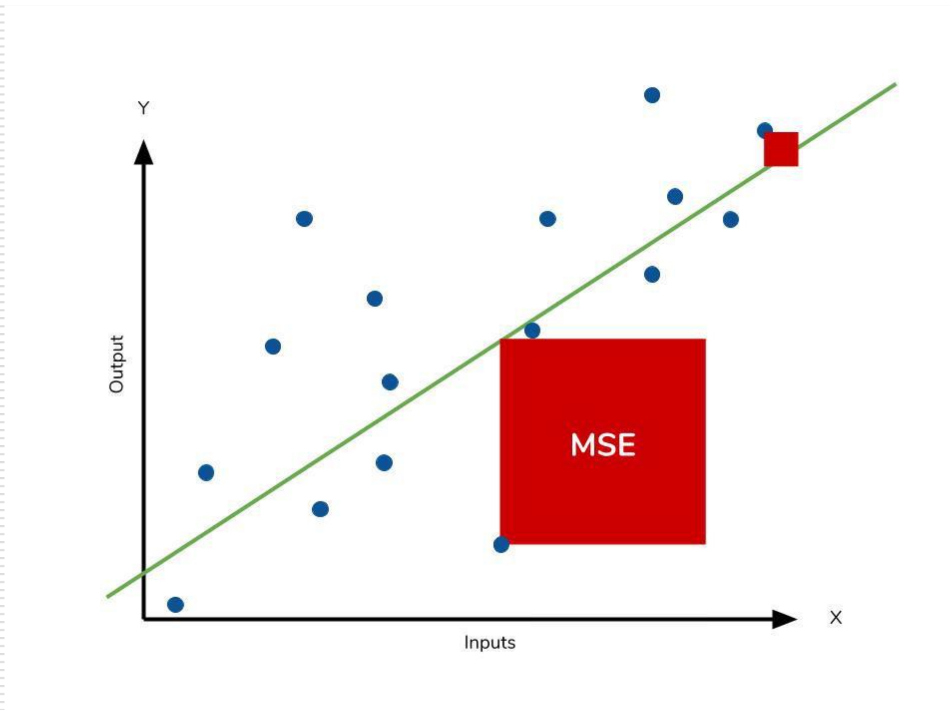
- Widely used in Regression
- L1 loss
- Greater interpretability

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i|$$

MSE (Mean Square Error)

- Widely used in Regression
- L2 loss
- Outlier cause huge errors
  - avoid occasional large errors more than MAE

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$



Cross Entropy (1/2)

- Measure the difference between two probability distributions
- Widely used in classification tasks
  - Target is a **class** instead of a **value**

$$H(p,q) = -\sum_{x} \underbrace{p(x)}_{\textit{target}} \log \underbrace{q(x)}_{\textit{prediction}}.$$

$$\underset{\textit{prediction}}{\begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}} \xleftrightarrow{\textit{CE}} \underset{\textit{target}}{\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}}$$

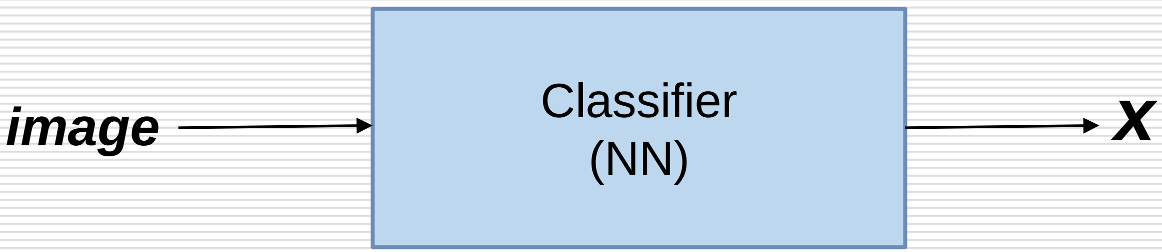
Cross Entropy (2/2)

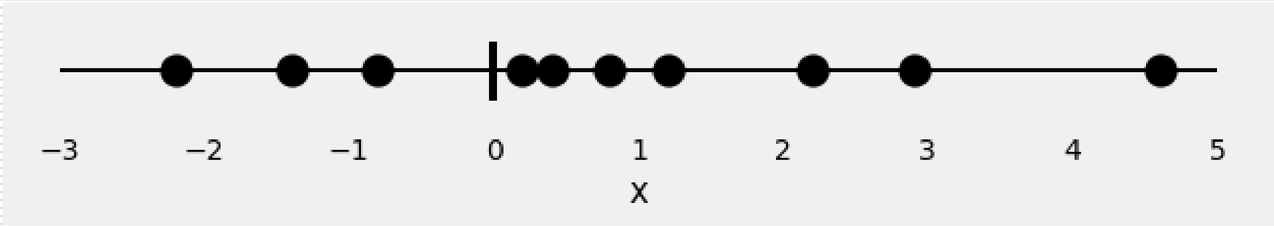
	Target (Label)	Model 1 (輸出)					
		機率輸出			實際One-hot encode		
		男生	女生	其他	男生	女生	其他
data 1	男生	0.4	0.3	0.3	1	0	0
data 2	女生	0.3	0.4	0.3	0	1	0
data 3	男生	0.5	0.2	0.3	1	0	0
data 4	其他	0.8	0.1	0.1	0	0	1
		模型1錯誤率: 1/4=0.25 Cross-entropy=6.966					

	Target (Label)	Model 2 (輸出)					
		機率輸出			實際One-hot encode		
		男生	女生	其他	男生	女生	其他
data 1	男生	0.7	0.1	0.2	1	0	0
data 2	女生	0.1	0.8	0.1	0	1	0
data 3	男生	0.9	0.1	0	1	0	0
data 4	其他	0.4	0.3	0.3	0	0	1
		模型1錯誤率: 1/4=0.25 Cross-entropy= 2.310					

Binary Cross-Entropy (BCE) Loss (1/3)

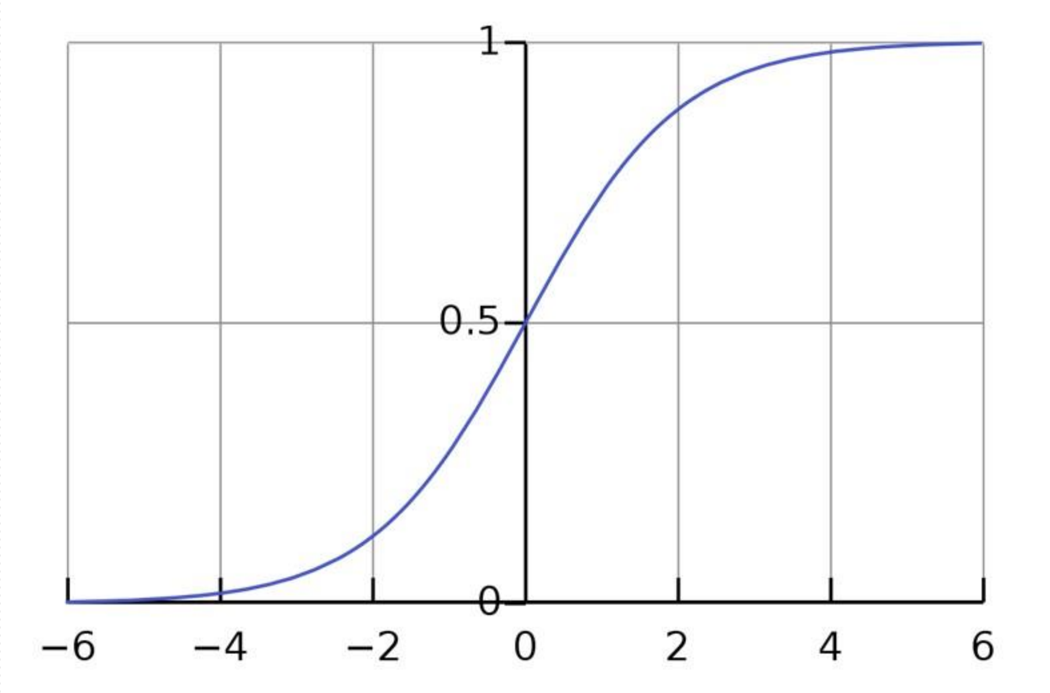
- Binary classification





Binary Cross-Entropy (BCE) Loss (2/3)

2.5		0.92
-3.5	$\xrightarrow{\text{sigmoid}}$	0.02
-0.5		0.38
x		***probability***



sigmoid

Binary Cross-Entropy (BCE) Loss (3/3)

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} \underbrace{y_i}_{\text{target}} \cdot log(\underbrace{p(y_i)}_{\text{prediction}}) + (1 - y_i) \cdot log(1 - p(y_i))$$

Data	Model 1	Model 2
Cat (label = 0)	0.8	0.3
Cat (label = 0)	0.6	0.2
Dog (label = 1)	0.3	0.6

Loss of Model 1 = -(log(1-0.8) + log(1-0.6) + log(0.3)) /3
= **0.53**

Loss of Model 2 = -(log(1-0.3) + log(1-0.2) + log(0.6)) /3
= **0.15 (better)**

Weighted Binary Cross-Entropy (WBCE)

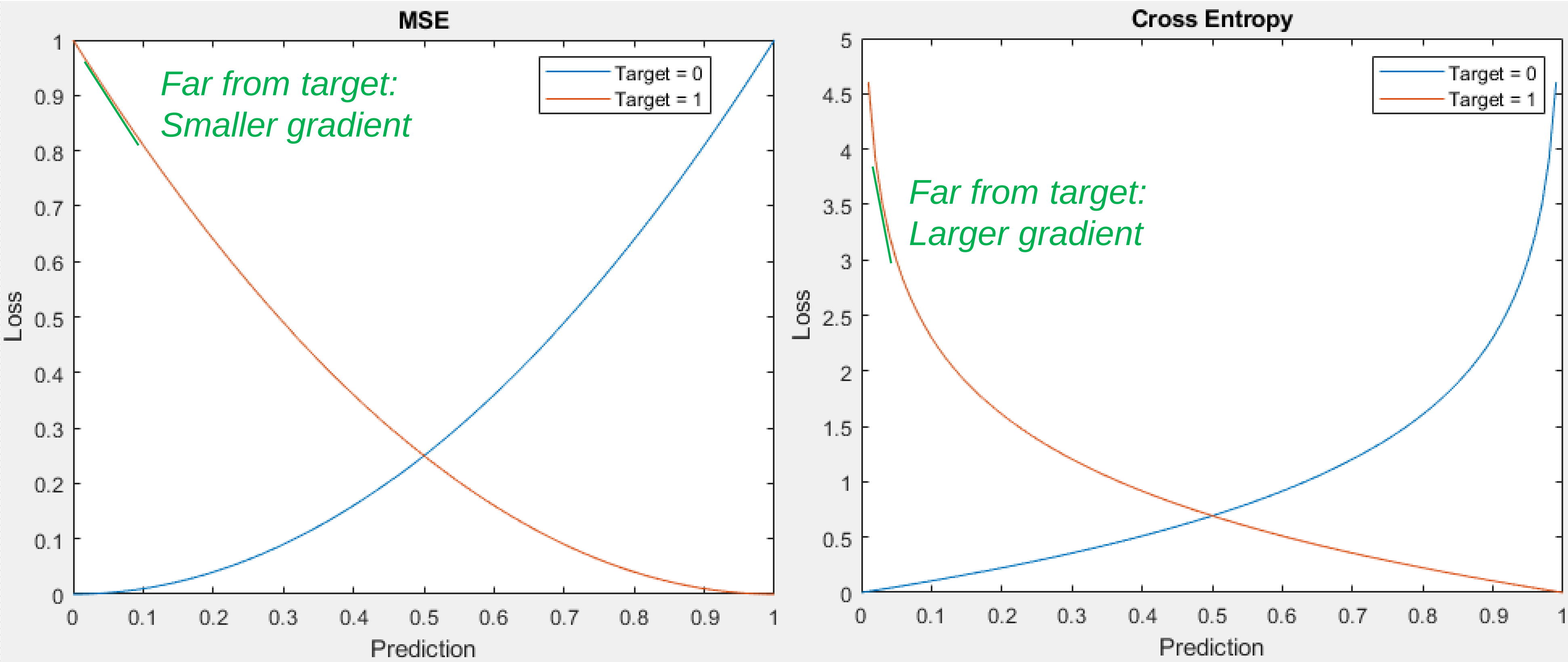
- Classify imbalanced datasets
  - 10% of training data are Dogs
  - 90% of training data are Cats
    - Classifier always outputs Cats ➔ 90% accuracy

- Set $\beta = 9$
  - weight the smaller class (dog) to a higher loss value

$$\mathrm{BCE}(\hat{p}, p) = -\big(p \cdot \log(p) + (1 - \hat{p}) \cdot \log(1 - p)\big)$$

$$\mathrm{WBCE}(\hat{p}, p) = -(\boxed{\beta} \cdot p \cdot \log(p) + (1 - \hat{p}) \cdot \log(1 - p))$$

MSE vs. Cross Entropy



- Cross Entropy punishes more than MSE when prediction is far away from target

Loss Functions in PyTorch

- nn.L1Loss()
- nn.MSELoss()
- nn.CrossEntropyLoss(weight=None)
  - weight: a manual rescaling weight given to each **class**
    - 1D Tensor, size = # of classes
- nn.BCELoss(weight=None)
  - weight: a manual rescaling weight given to the loss of each **batch** element
    - 1D Tensor, size = # of batches

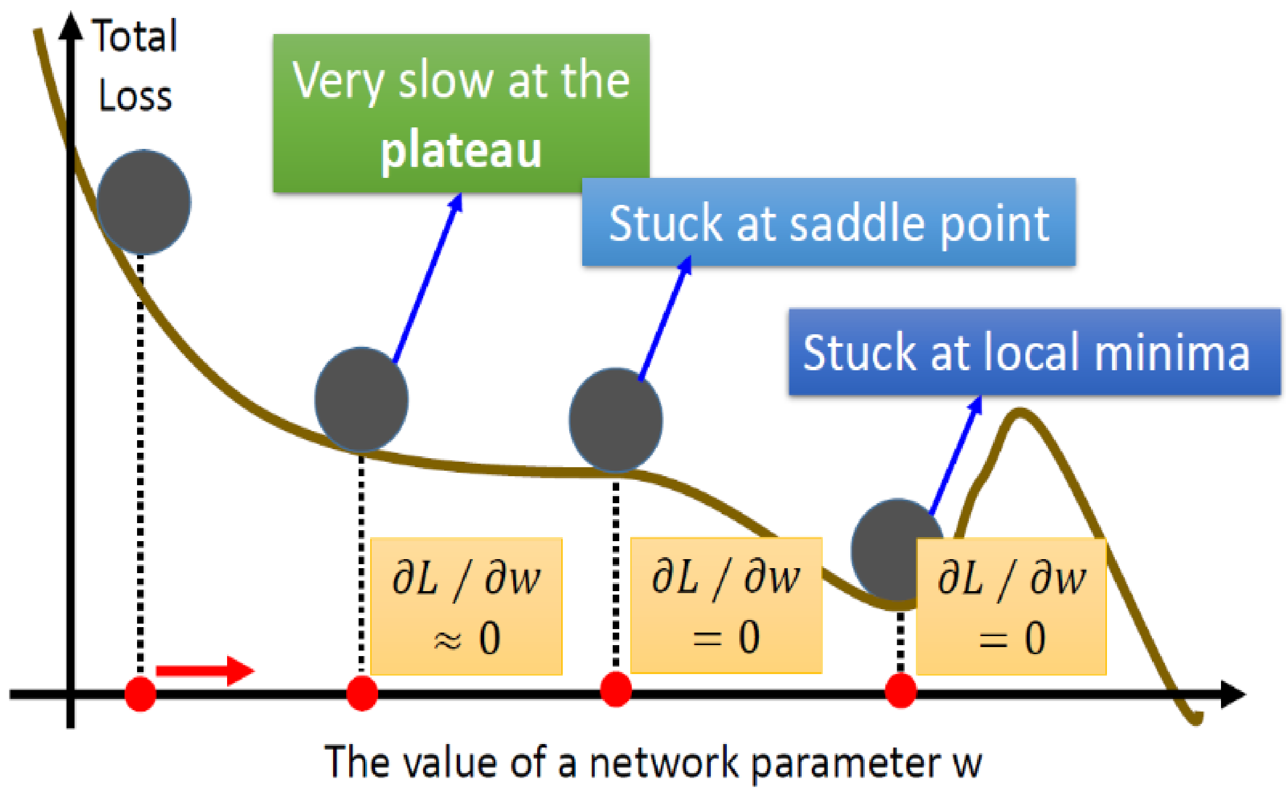
Introduction to Momentum

Limitations of Gradient Descent Method

- Stuck in local minimum
  - gradient = 0, but isn't global minimum (lowest point of loss)
  - can't get good accuracy

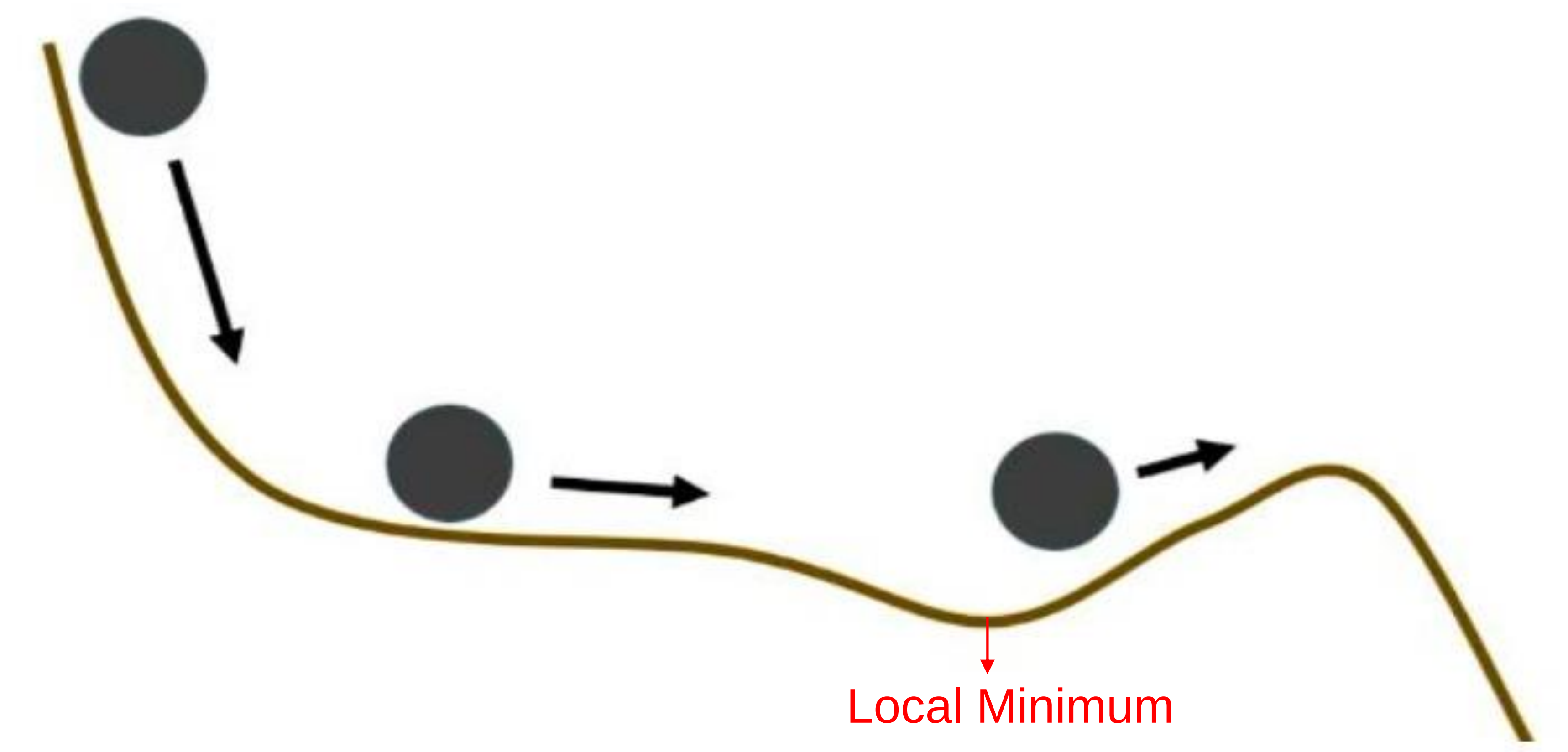
$$w_i^{t+1} \leftarrow w_i^t - \eta \cdot \frac{\partial L}{\partial w_i}$$

η: Learning Rate



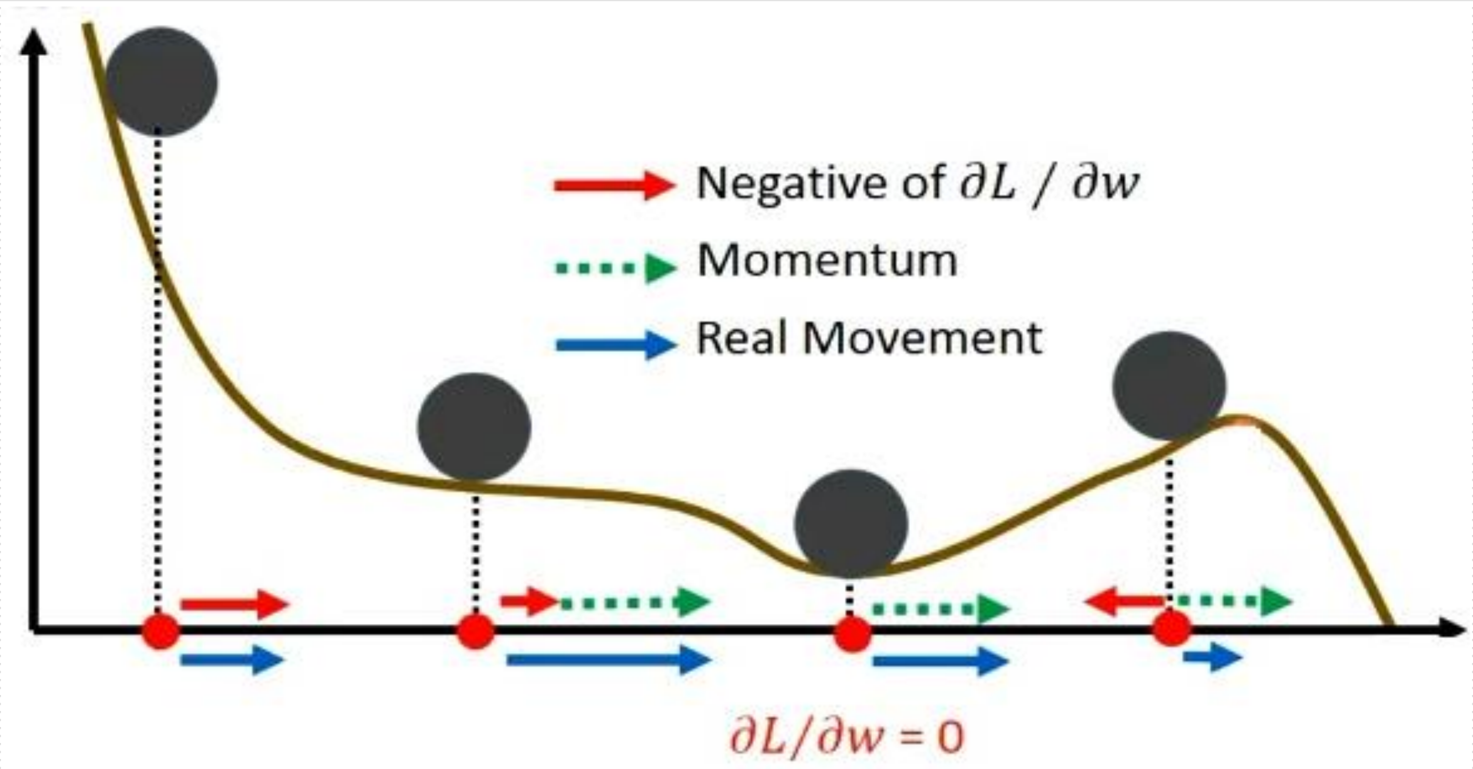
Momentum in Real World

- In real world, the ball may not stop at the local minimum
  - everything has inertia and momentum



Momentum in Gradient Descent (1/3)

- Introduce momentum into machine learning
  - even if gradient = 0 is encountered, it's still possible to continue to move forward
  - greater chance of finding the global minimum



Momentum in Gradient Descent (2/3)

- Each time the weight is updated, the previous result (inertial direction) is considered
  - if last gradient is in the same direction as this time, $|V_t|$ will become larger
  - the update gradient of the W parameter will become faster
  - actually the weighted sum of the previous gradients

$$V_t \leftarrow \beta V_{t-1} - \eta \frac{\partial L}{\partial w}$$

$$w \leftarrow w + V_t$$

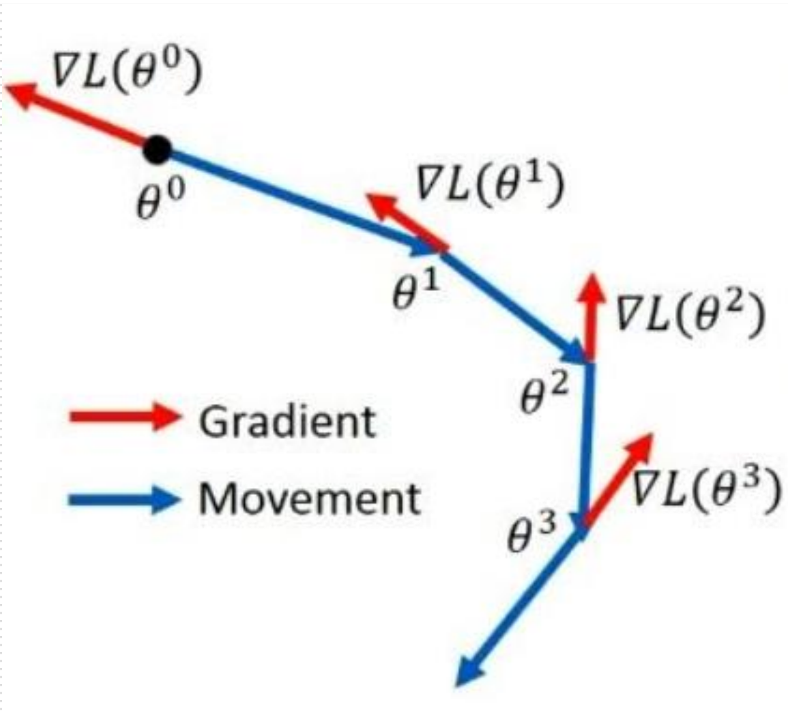
t: update times

β: momentum weight

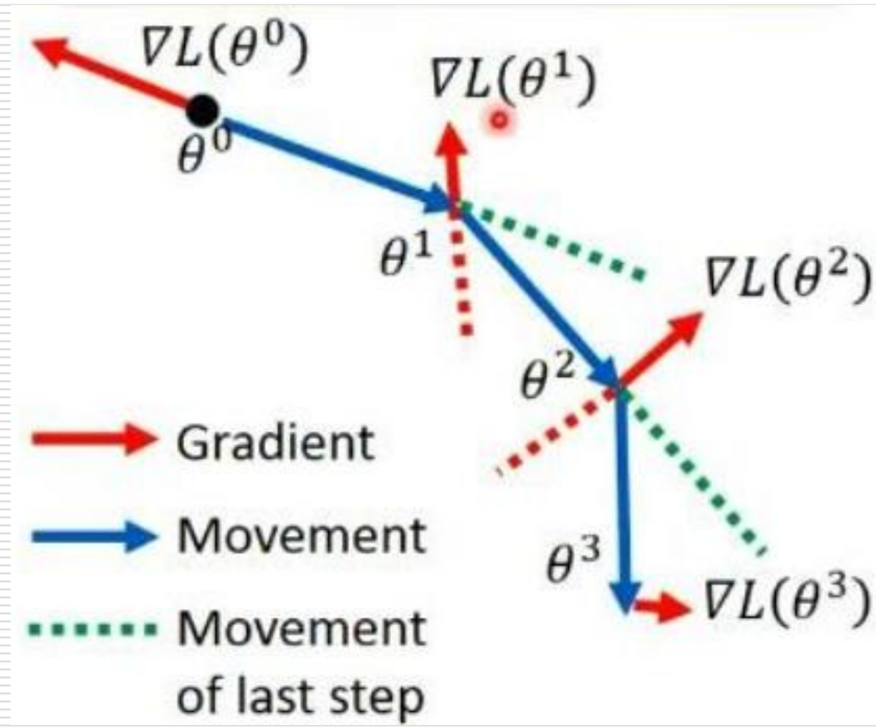
η: learning rate

V_t, V_{t-1}: momentum

Momentum in Gradient Descent (3/3)



**Common
Gradient Descent**



**Gradient Descent
with Momentum**

Introduction to Optimizers

Issue of Learning Rate (1/2)

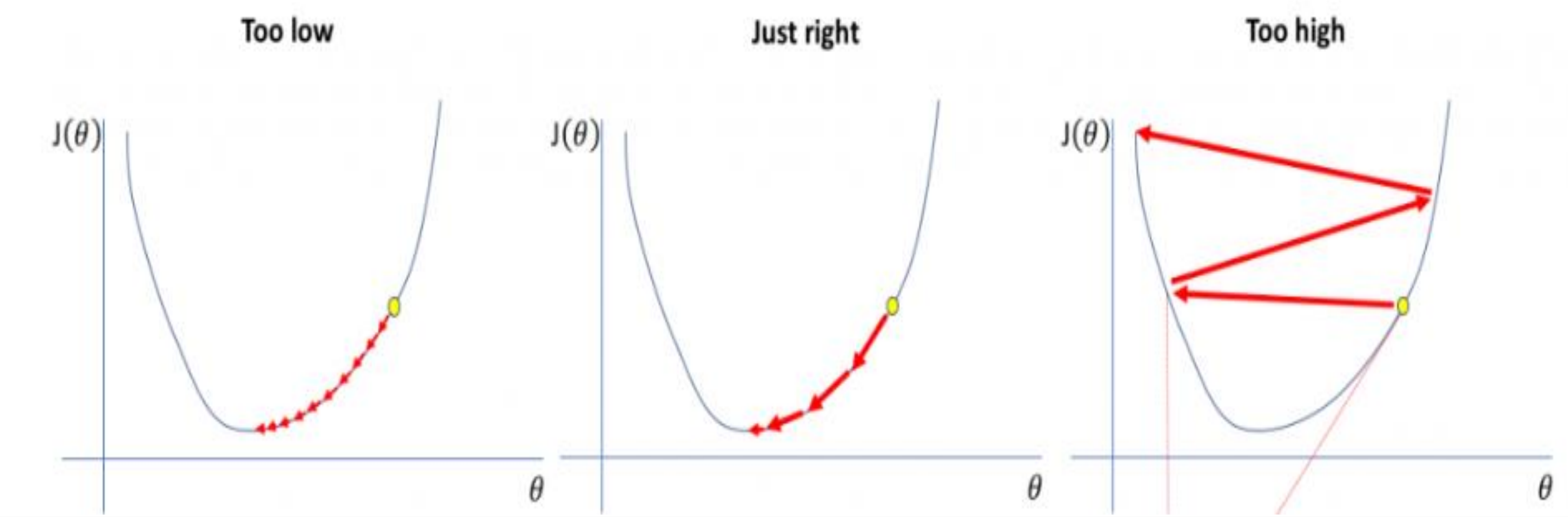
- In gradient descent method or momentum update, learning rate is fixed
  - small learning rate requires many updates before stable
  - large learning rate may be difficult to converge

$$w_i^{t+1} \leftarrow w_i^t - \boxed{\eta} \cdot \frac{\partial L}{\partial w_i}$$

$$V_t \leftarrow \beta V_{t-1} - \boxed{\eta} \frac{\partial L}{\partial w}$$

$$w \leftarrow w + V_t$$

Issue of Learning Rate (2/2)



requires many updates!

difficult to converge!

Dynamic Learning Rate (LR)

- Adjust the learning rate during training
  - at beginning, far from the destination, use larger LR
  - after several epochs, close to the destination, reduce LR

Weight Decay

- e.g., L2 regularization
- Smooth functions are preferred
  - output is less sensitive to input
  - has less influence on noisy input

$$y = b + \sum w_i x_i$$

$+w_i \Delta x_i$ $\qquad$ $+\Delta x_i$

- Larger λ, consider the training error less

$$w_i^{t+1} \longleftarrow w_i^t - \eta \cdot \frac{\partial L'}{\partial w_i} \cdot \text{ where } L'(\boldsymbol{W}) = L(\boldsymbol{W}) \boxed{+ \lambda \cdot \sum_{n=1}^{N} (w_n^t)^2}$$

$$\Longrightarrow w_i^{t+1} \longleftarrow w_i^t - \eta \cdot (\frac{\partial L}{\partial w_i} \boxed{+ 2\lambda w_i^t})$$

Smaller w's are better!

$$\Longrightarrow w_i^{t+1} \longleftarrow \underbrace{\boxed{(1 - 2\eta\lambda)}}_{< 1} w_i^t - \eta \cdot \frac{\partial L}{\partial w_i}$$

Approach to 0!

Common Optimizers

SGD Optimizer

- Stochastic Gradient Descent
- Randomly choose a mini-batch (batch size = 1) instead of a whole batch to update gradient for each iteration

$$g_{t,i} = \nabla_{w_i} L\left(w_i^{(t)}\right)$$ gradient of the i-th weight at t-th iteration

$$w_i^{(t+1)} = w_i^{(t)} - \eta g_{t,i}$$

Adagrad Optimizer (1/2)

- Adaptive gradient
  - during the training process, adjust LR from large to small

$$w_i^{(t+1)} = w_i^{(t)} - \eta \frac{1}{\sqrt{G_{t,ii} + \varepsilon}} g_{t,i}$$

$$G_{t,ii} = \boxed{\sum_{\tau=1}^{t} g_{\tau,i}^2}$$

Accumulate squared gradients
as increasing iterations

$$w_i^{(t+1)} = w_i^{(t)} - \eta \frac{1}{\sqrt{\sum_{\tau=1}^{t} \left(\nabla_{w_i} L \left(w_i^{(t)} \right) \right)^2 + \varepsilon}} g_{t,i}$$

ε : a term to avoid denominator = 0 (set to 1e-7 usually)

Adagrad Optimizer (2/2)

$$w^1 = w^0 - \frac{\eta^0}{\sqrt{G_0+\varepsilon}} g_0 \qquad G_0 = \sqrt{(g^0)^2}$$

$$w^2 = w^1 - \frac{\eta^1}{\sqrt{G_1+\varepsilon}} g_1 \qquad G_1 = \sqrt{(g^0)^2+(g^1)^2}$$

$$w^3 = w^2 - \frac{\eta^2}{\sqrt{G_2+\varepsilon}} g_2 \qquad G_2 = \sqrt{(g^0)^2+(g^1)^2+(g^2)^2}$$

RMSprop Optimizer (1/2)

- Root Mean Square Prop
  - learning rate decrease too fast for Adagrad
  - replace sum (too large) with average

$$w_i^{(t+1)} = w_i^{(t)} - \eta \frac{1}{\sqrt{\sum[g_i^2]_t + \varepsilon}} g_{t,i}$$

$$\sum[g_i^2]_t = \boxed{\alpha} \boxed{\sum[g_i^2]_{t-1}} + \boxed{(1-\alpha)} g_{t,i}^2$$

Average squared gradients
of the last (t-1) iterations

α : a hyperparameter for weighting the average of squared gradients of the last (t-1) iterations

RMSprop Optimizer (2/2)

$$\boldsymbol{\theta}_i^{\mathbf{1}} \leftarrow \boldsymbol{\theta}_i^0 - \frac{\eta}{\sigma_i^0} \boldsymbol{g}_i^{\mathbf{0}} \qquad \sigma_i^0 = \sqrt{\left(\boldsymbol{g}_i^{\mathbf{0}}\right)^2}$$

$$0 < \alpha < 1$$

$$\boldsymbol{\theta}_i^{\mathbf{2}} \leftarrow \boldsymbol{\theta}_i^1 - \frac{\eta}{\sigma_i^1} \boldsymbol{g}_i^{\mathbf{1}} \qquad \sigma_i^1 = \sqrt{\alpha\left(\sigma_i^0\right)^2 + (1-\alpha)\left(\boldsymbol{g}_i^{\mathbf{1}}\right)^2}$$

$$\boldsymbol{\theta}_i^3 \leftarrow \boldsymbol{\theta}_i^2 - \frac{\eta}{\sigma_i^2} \boldsymbol{g}_i^{\mathbf{2}} \qquad \sigma_i^2 = \sqrt{\alpha\left(\sigma_i^1\right)^2 + (1-\alpha)\left(\boldsymbol{g}_i^{\mathbf{2}}\right)^2}$$

$$\vdots$$

$$\boldsymbol{\theta}_i^{\boldsymbol{t+1}} \leftarrow \boldsymbol{\theta}_i^{\boldsymbol{t}} - \frac{\eta}{\sigma_i^t} \boldsymbol{g}_i^{\boldsymbol{t}} \qquad \sigma_i^t = \sqrt{\alpha\left(\sigma_i^{t-1}\right)^2 + (1-\alpha)\left(\boldsymbol{g}_i^{\boldsymbol{t}}\right)^2}$$

Adam Optimizer

- Adaptive Moment Estimation
  - Momentum + RMSprop + correction
  - $m_i^{(t)}$: moving average of gradients (the 1st moment)
  - $v_i^{(t)}$: moving average of squared gradients (the 2nd moment)
  - $\beta_1, \beta_2 \in [0, 1)$: a hyperparameter for exponential decay rates of these moving averages

$$m_i^{(t)} = \beta_1\, m_i^{(t-1)} + \ (1 - \beta_1)\, g_{t,i}$$

$$v_i^{(t)} = \beta_2\, v_i^{(t-1)} + \ (1 - \beta_2)\, g_{t,i}^2$$

$$\widehat{m}_i^{(t)} = \frac{m_i^{(t)}}{1 - \beta_1^t}\ ;\ \hat{v}_i^{(t)} = \frac{v_i^{(t)}}{1 - \beta_2^t}$$

$$w_i^{(t+1)} = w_i^{(t)} - \eta \frac{1}{\sqrt{\hat{v}_i^{(t)}} + \varepsilon} \widehat{m}_i^{(t)}$$

Optimizers in PyTorch (1/2)

- SGD (optionally with momentum or weight decay)
  - torch.optim.SGD(model.parameters(), lr=<required parameter>, momentum=0, weight_decay=0)

- Adagrad (optionally with weight decay)
  - torch.optim.Adagrad(model.parameters(), lr=0.01, weight_decay=0, eps=1e-10)

- RMSprop (optionally with momentum or weight decay)
  - torch.optim.RMSprop(model.parameters(), lr=0.01, alpha=0.99, eps=1e-08, weight_decay=0, momentum=0)

Optimizers in PyTorch (2/2)

- Adam (optionally with L2 regularization)
  - torch.optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0)

    β_1 β_2
- AdamW (Decoupled Weight Decay Regularization)
  - torch.optim.AdamW(model.parameters(), lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0.01)

$$w_i^{(t+1)} = w_i^{(t)} - \eta\left(\frac{1}{\sqrt{\hat{v}_i^{(t)}} + \varepsilon}\hat{m}_i^{(t)} + \lambda w_i^{(t)}\right)$$

- LAMB
  - torch_optimizer.Lamb(model.parameters(), lr=0.1, betas=(0.9, 0.999), eps=1e-08, weight_decay=0)

LAMB Optimizer

- Adam or AdamW don't work well for a large batch
- Layer-wise scaled by $\phi\left(\left\|w_i^{(t)}\right\|\right)$
- Layer-wise normalized to unit l_2-norm
- Scale the batch size of BERT pre-training to 64K without losing accuracy
- Reducing the BERT training time from 3 days to around 76 minutes

$$w_i^{(t+1)} = w_i^{(t)} - \eta \frac{\phi\left(\left\|w_i^{(t)}\right\|\right)}{\left\|r_i^{(t)} + \lambda w_i^{(t)}\right\|}\left(r_i^{(t)} + \lambda w_i^{(t)}\right)$$

$$r_i^{(t)} = \frac{1}{\sqrt{\hat{v}_i^{(t)}} + \varepsilon}\widehat{m}_i^{(t)} \quad ; \ \phi(z) = \min\{\max\{z, \gamma_l\}, \gamma_u\}$$

γ_l, γ_u: lower and upper bound for z

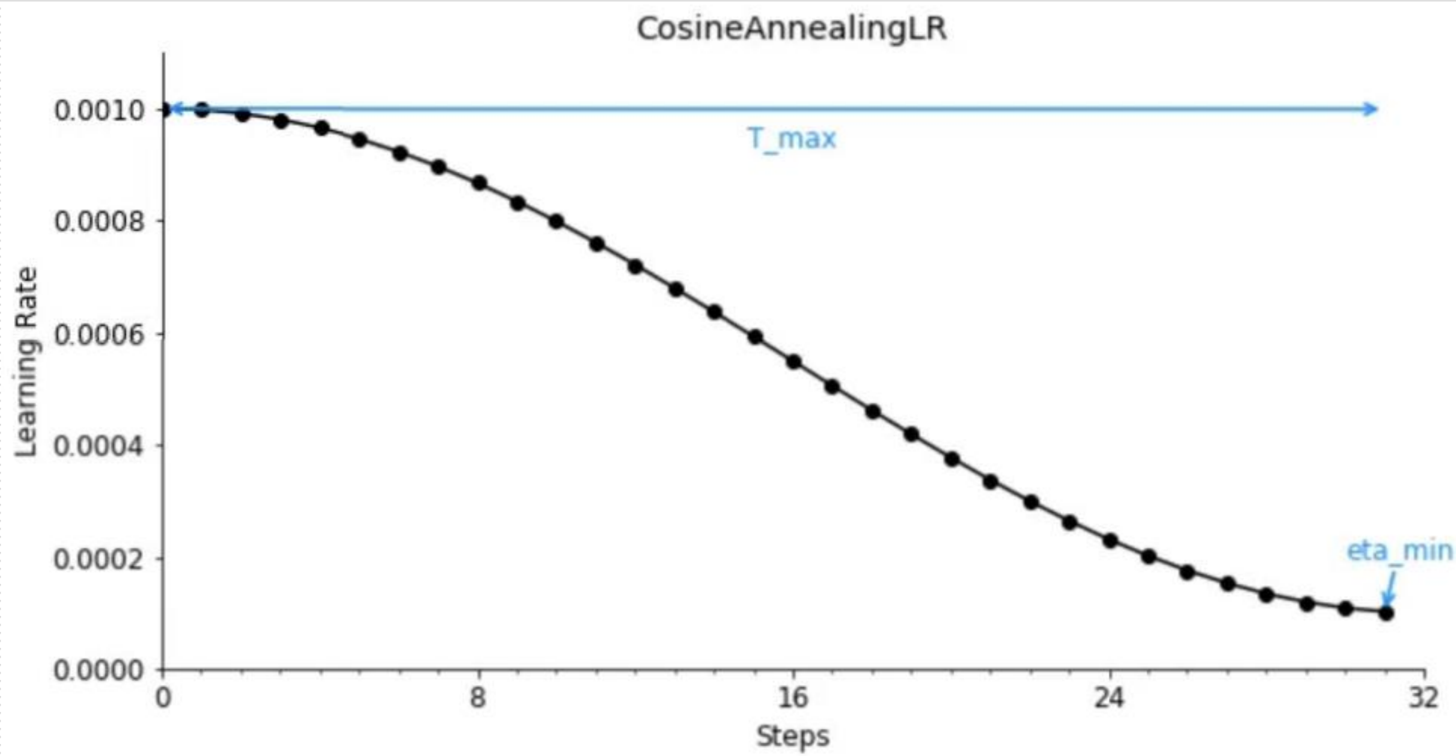
Learning Rate Schedulers

Learning Rate Scheduler

- Predefined framework that adjusts the learning rate between iterations as the training progresses
  - CosineAnnealingLR
  - CosineAnnealingWarmRestarts
  - CyclicLR
  - OneCycleLR
  - ReduceLROnPlateau

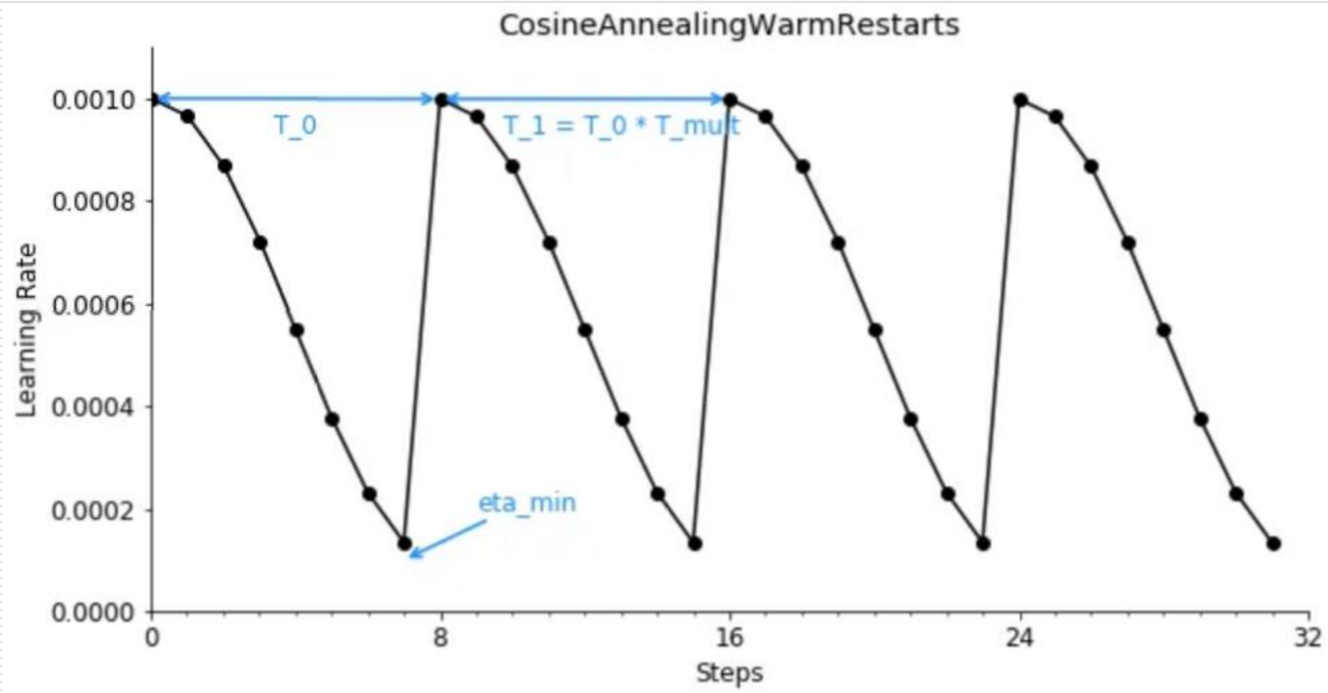
CosineAnnealingLR

- torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max = 32, eta_min = 1e-4)
  - T_max: maximum number of iterations (**epochs**)
  - eta_min: minimum learning rate



CosineAnnealingWarmRestarts

- torch.optim.lr_scheduler.CosineAnnealingWarmRestarts (optimizer, T_0 = 8, T_mult = 1, eta_min = 1e-4)
  - T_0: number of iterations for the first restart
  - T_mult: a factor increases Ti after a restart
  - eta_min: minimum learning rate



Warm Restart

- Increasing LR causes the model to diverge

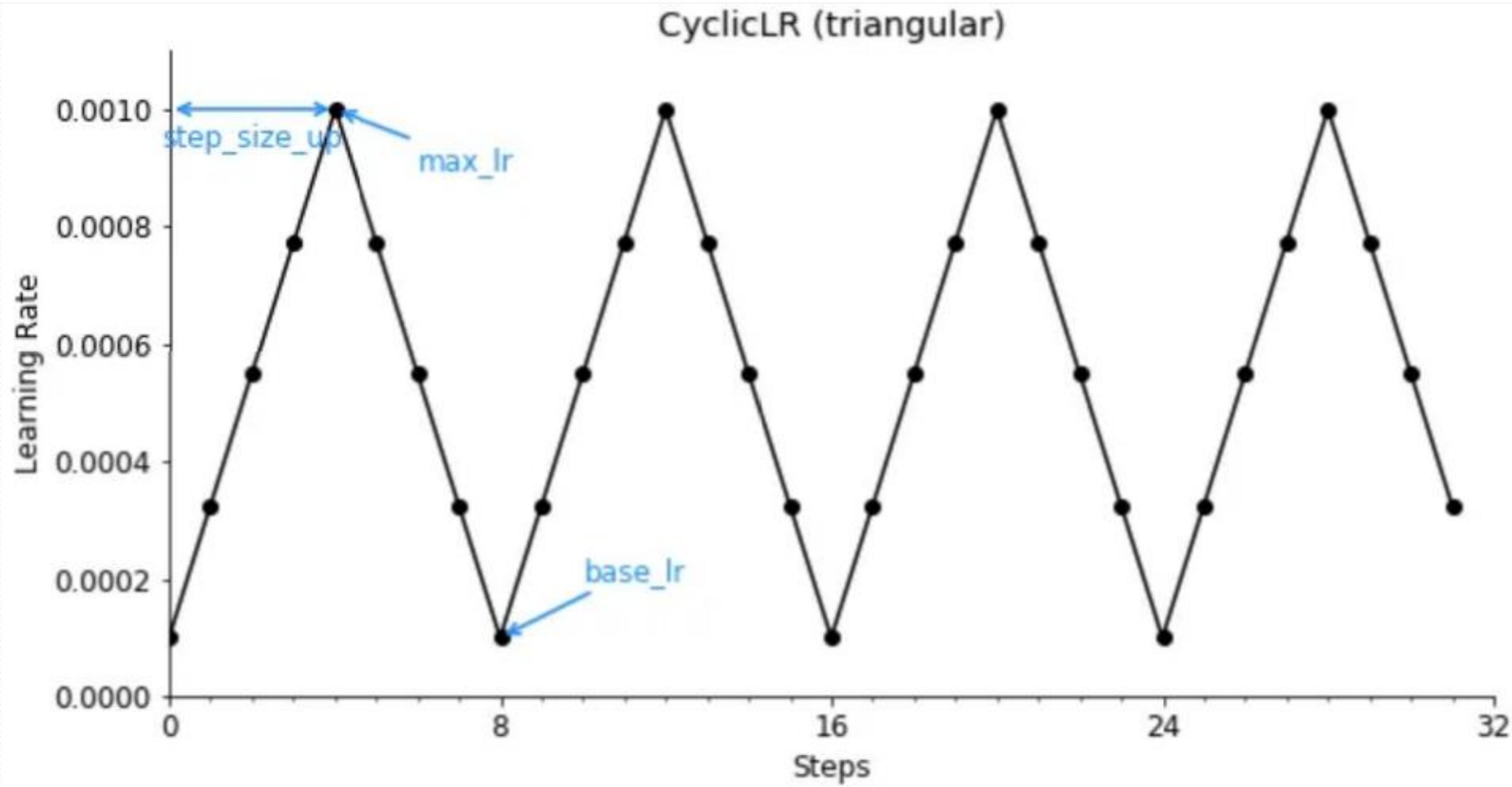
- Intentional divergence enables the model to escape local minimum and find an even better global minimum

CyclicLR (1/2)

- Use warm restart strategy
- Cyclical Learning Rate (CLR) Policy
- Change the learning rate after every batch
- torch.optim.lr_scheduler.CyclicLRCyclicLR(optimizer, base_lr = 0.0001, max_lr = 1e-3, step_size_up = 4, mode = "triangular")
  - base_lr: initial LR, the lower boundary in the cycle
  - max_lr: upper LR boundaries in the cycle
  - step_size_up: number of training iterations in the increasing half of a cycle
  - mode: "triangular“ or "triangular2“ or "exp_range"

CyclicLR (2/2)

- mode = "triangular"

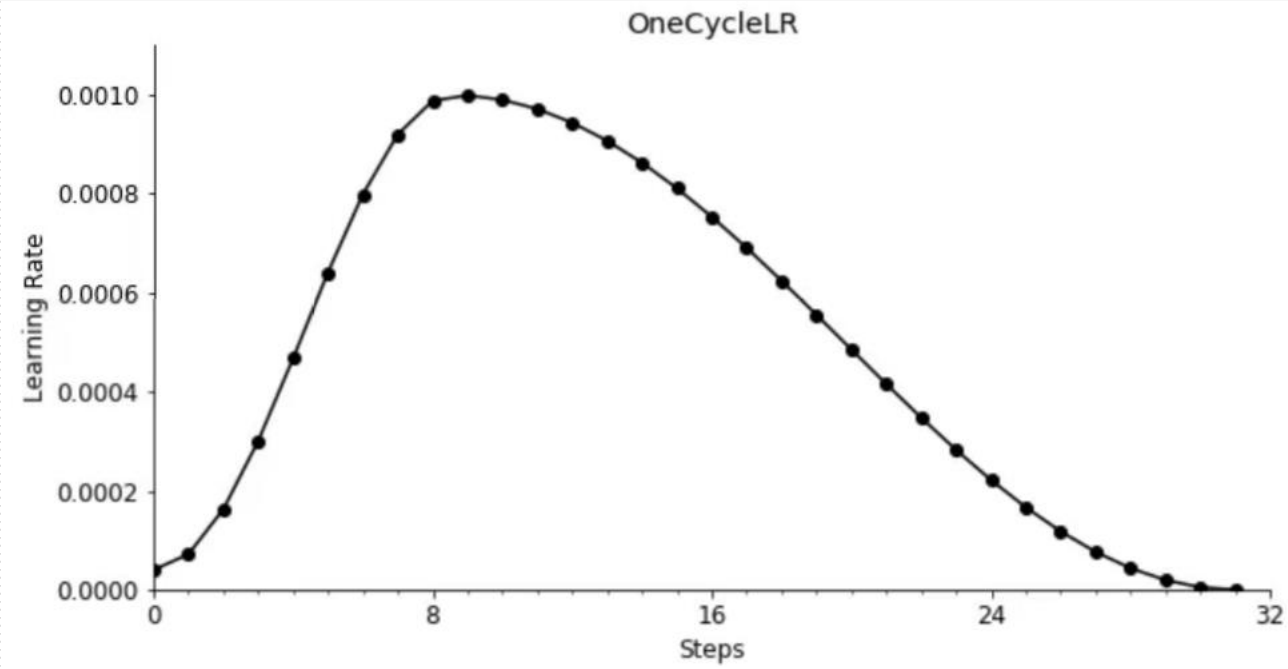


OneCycleLR (1/2)

- 1cycle learning rate policy
  - achieve super-convergence
- Change the learning rate after every batch
- Use warm-up learning rate strategy
  - data are new for the model at the beginning of training
  - smaller LR (less weight update) to gain more knowledge
  - prevent overfitting to the first data

OneCycleLR (2/2)

- torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr = 1e-3, steps_per_epoch = 8, epochs = 4)
  - max_lr: upper learning rate boundaries in the cycle
  - steps_per_epoch: number of steps per epoch to train for
  - epochs: number of epochs to train for



ReduceLROnPlateau

- Reduce learning rate when a metric has stopped improving for a 'patience' number of epochs
- Update LR independent of epochs
- torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=2)
  - mode='**min**': lr will be reduced when the quantity monitored has stopped **decreasing**
  - factor: factor by which the learning rate will be reduced new_lr = lr * factor
  - patience=2: ignore the first 2 epochs with no improvement, and will decrease the LR after the 3rd epoch if the loss still hasn't improved then

Data Augmentation

Insufficient Training Data

- Collecting large amounts of data is usually a big problem
  - some data is limited, e.g., medical images
  - if data is insufficient or the complexity is too low, it may also cause overfitting

Data Augmentation (1/5)

- Create new training materials under limited data
  - avoid overfitting
  - improve accuracy

- Common methods
  - geometric transformations
    - pad, resize, rotate, flip, crop
  - color space transformations
    - gray scale, change the brightness, contrast, saturation or hue
  - increase noise
  - Gaussian blur

Data Augmentation (2/5)

Resize

Rotate

Data Augmentation (3/5)

Flip

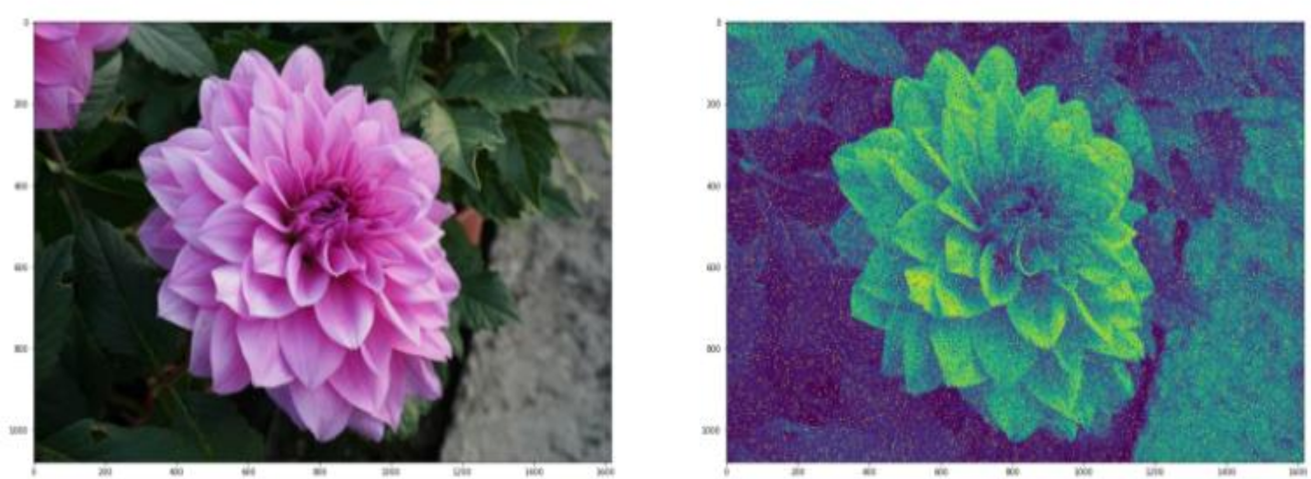
Crop

Data Augmentation (4/5)

Gray scale

Color Jitter: brightness, contrast, saturation or hue

Data Augmentation (5/5)

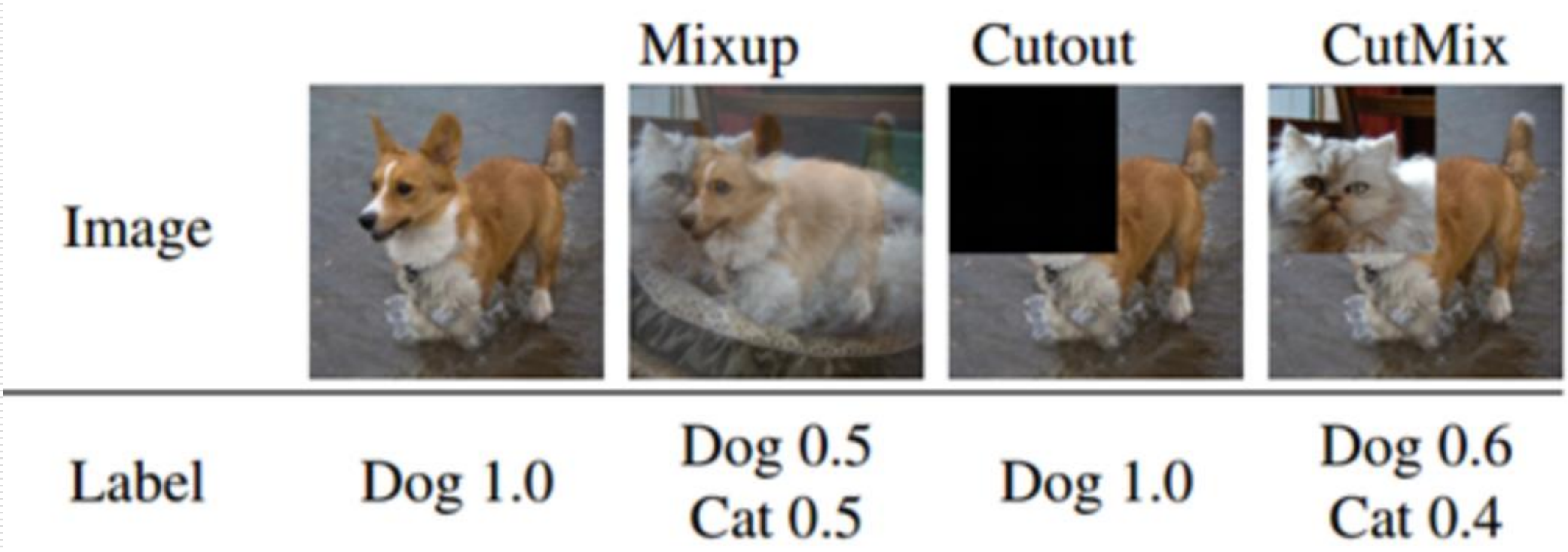


Noise

Gaussian Blur

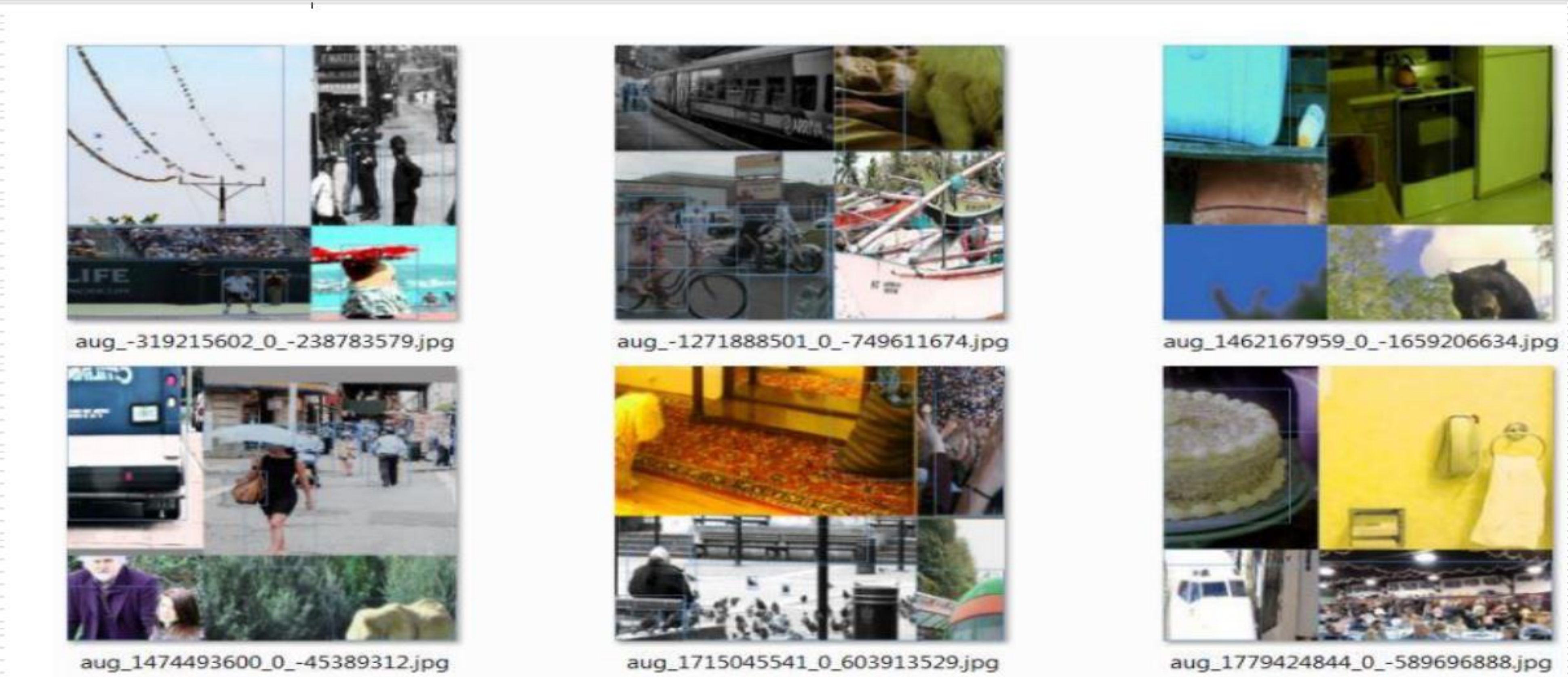
Advanced Augmentation (1/3)

- Fusion of different features in one training sample
  - **adjust the label** according to the mixing ratio
  - Mixup, Cutout, CutMix



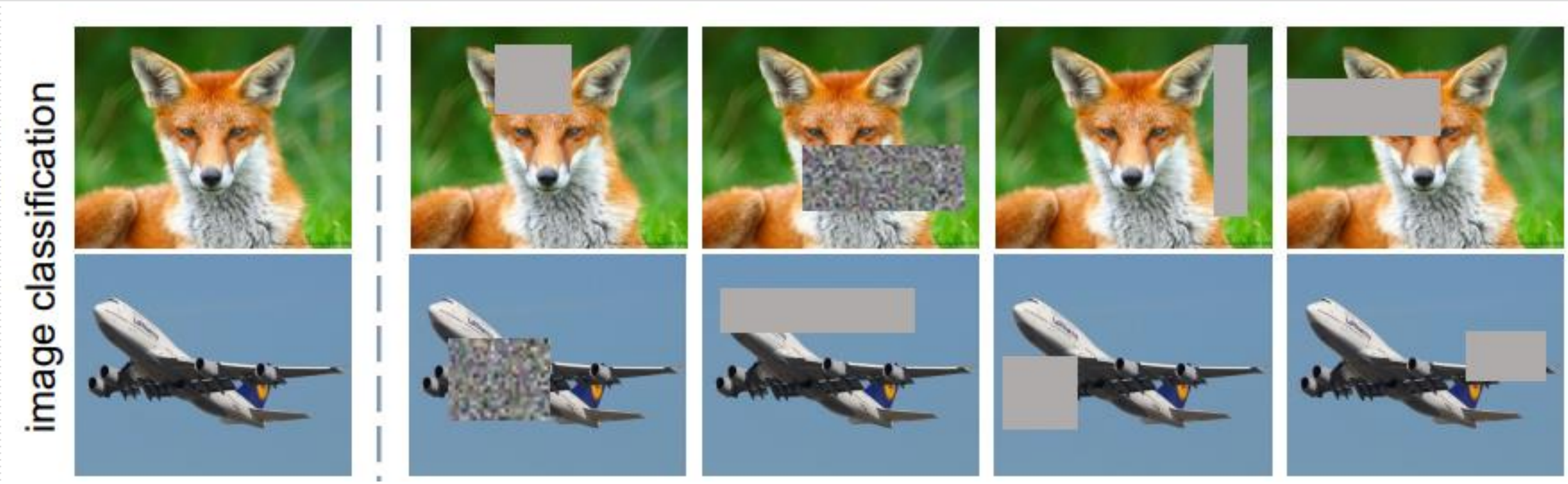
Advanced Augmentation (2/3)

- Fuse more pictures together for training
  - mosaic method



Advanced Augmentation (3/3)

- Random Erasing
  - randomly choose a rectangle region in the image and erase its pixels with random values or the ImageNet mean pixel value



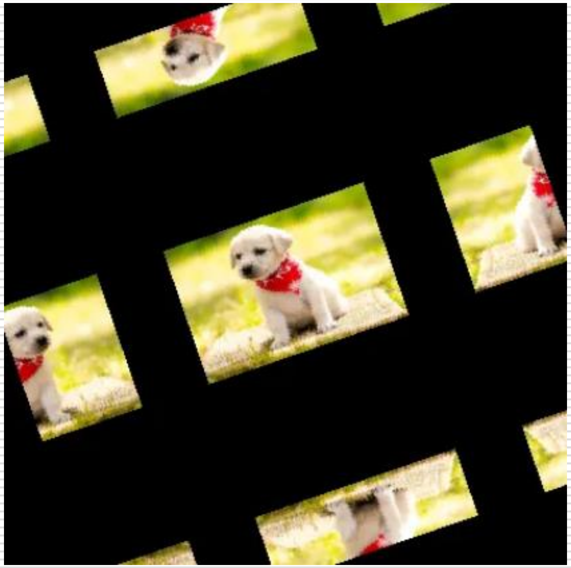
Data Augmentation – Example

- Combine different data augmentation methods
- Code

```
transform_set = [
                transforms.CenterCrop(200),
                transforms.Pad(100, padding_mode='symmetric'),
                transforms.RandomRotation(30),
                transforms.ColorJitter()
]
transform = transforms.Compose([
    transforms.Resize((100,150)),
    transforms.RandomApply(transform_set, p=0.5)
])

new_img = transform(img_pil)
new_img
```

- Result



References

References (1/3)

- 台大李宏毅老師機器學習課程
  - http://speech.ee.ntu.edu.tw/~tlkagk/courses_ML16.html
- 聊一聊深度學習的activation function
  - https://zhuanlan.zhihu.com/p/25110450
- Gradient descent with momentum
  - https://zhuanlan.zhihu.com/p/34240246
- Gradient descent optimization algorithms
  - https://reurl.cc/nz8Kkn
- Group Normalization
  - https://arxiv.org/abs/1803.08494

References (2/3)

- Regularization
  - https://hackmd.io/@allen108108/Bkp-RGfCE
- Adam: A Method for Stochastic Optimization
  - https://arxiv.org/abs/1412.6980
- Large Batch Optimization for Deep Learning: Training BERT in 76 minutes
  - https://arxiv.org/abs/1904.00962
- Learning Rate Scheduler
  - https://towardsdatascience.com/a-visual-guide-to-learning-rate-schedulers-in-pytorch-24bbb262c863

References (3/3)

- Warm-up strategy
  - https://chih-sheng-huang821.medium.com/%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92warm-up%E7%AD%96%E7%95%A5%E5%9C%A8%E5%B9%B9%E4%BB%80%E9%BA%BC-95d2b56a557f
- PyTorch documentation
  - https://pytorch.org/docs/stable/optim.html
  - https://pytorch.org/docs/stable/nn.html
  - https://pytorch.org/vision/main/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py

Homework

Homework (1/4)

- Download the example file from FB club
  - copy it to your colab
  - data set: FashionMNIST
  - current accuracy: 55.57% (10 epochs)
  - press “Run all” to run the code
- Target
  - use the training techniques learned today to further improve your accuracy
- Download your code which achieve the best accuracy as a .py file and a .ipynb file and submit it
- Write a report about how you improve the accuracy

Homework (2/4)

```
Test Loss: 0.304884

Test Accuracy of Class     0: 83.90% (839/1000)
Test Accuracy of Class     1: 98.20% (982/1000)
Test Accuracy of Class     2: 82.30% (823/1000)
Test Accuracy of Class     3: 90.90% (909/1000)
Test Accuracy of Class     4: 85.20% (852/1000)
Test Accuracy of Class     5: 96.70% (967/1000)
Test Accuracy of Class     6: 74.70% (747/1000)
Test Accuracy of Class     7: 97.00% (970/1000)
Test Accuracy of Class     8: 97.90% (979/1000)
Test Accuracy of Class     9: 96.00% (960/1000)

Test Accuracy (Overall): 90.28% (9028/10000)
```

- Specifications
  - Do not modify the number of epochs!
  - Do not modify valid_size!
  - Do not modify the base model architecture!
    - › you cannot omit or modify the 3 layers of nn.Linear or add new layers of conv, linear, etc.
    - › you can add normalization, activation function or dropout layer
  - You can try various methods to improve your accuracy, but you need to provide screenshot of your results in your report.
  - Write down your best overall test accuracy at the beginning of your report. (e.g., Best accuracy: 90.28%)
  - TA will run your code if needed!

Homework (3/4)

- Grading Policy
  - Overall Test Accuracy (60%)
    - Accuracy $\geq$ 80% (30%)
    - Accuracy $\geq$ 84% (30% + 20%)
    - Accuracy $\geq$ 89% (30% + 20% + 10%)
  - Report (40%)
    - try more training techniques and tune more hyperparameters !
  - Bonus (5%)
    - Accuracy $\geq$ 90% or an excellent report !

Homework (4/4)

- Deadline: 7/21 (Sun.) 23:59

- Submit the following files to rhyang.ee12@nycu.edu.tw
  - HW3_[帳號].py
    - e.g., HW3_M112rhyang.py
  - HW3_[帳號].ipynb
    - e.g., HW3_M112rhyang.ipynb
  - HW3_Report_[帳號].pdf
    - e.g., HW3_Report_M112rhyang.pdf
  - 信件主旨：中文名字_HW3
    - e.g., 楊荏宏_HW3

Thank you