Deep Into Deep

김성찬

07 Training Neural Networks

Contents

1. Training Neural Networks

One-time Setup

- Architecture, Activation functions, Preprocessing, Weights initialization, Regularization etc.

Training dynamics

- Babysitting the learning processing, Parameter updates, Hyperparameter optimization

Evaluation

- Model ensembles

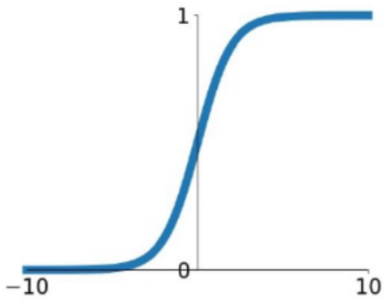
1. Training Neural Networks

- Activation function
- Data preprocessing
- Weight initialization
- Batch normalization
- Babysitting the learning processing
- Hyperparameter optimization

2. Activation function

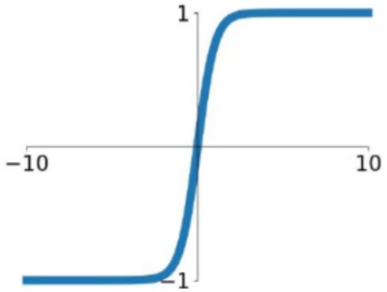
Sigmoid

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



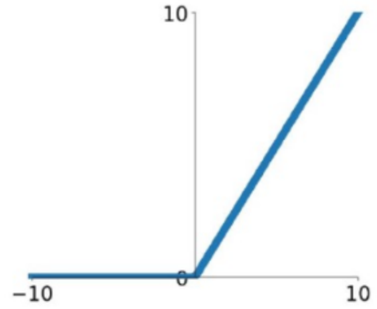
tanh

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



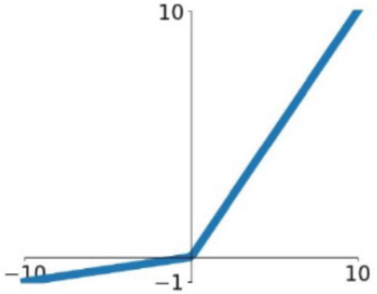
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

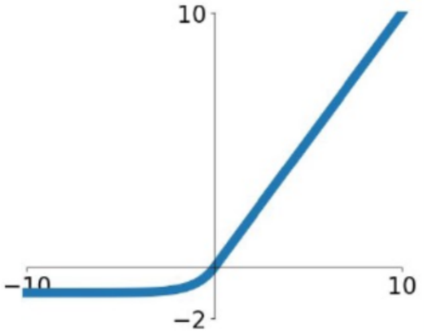


Maxout

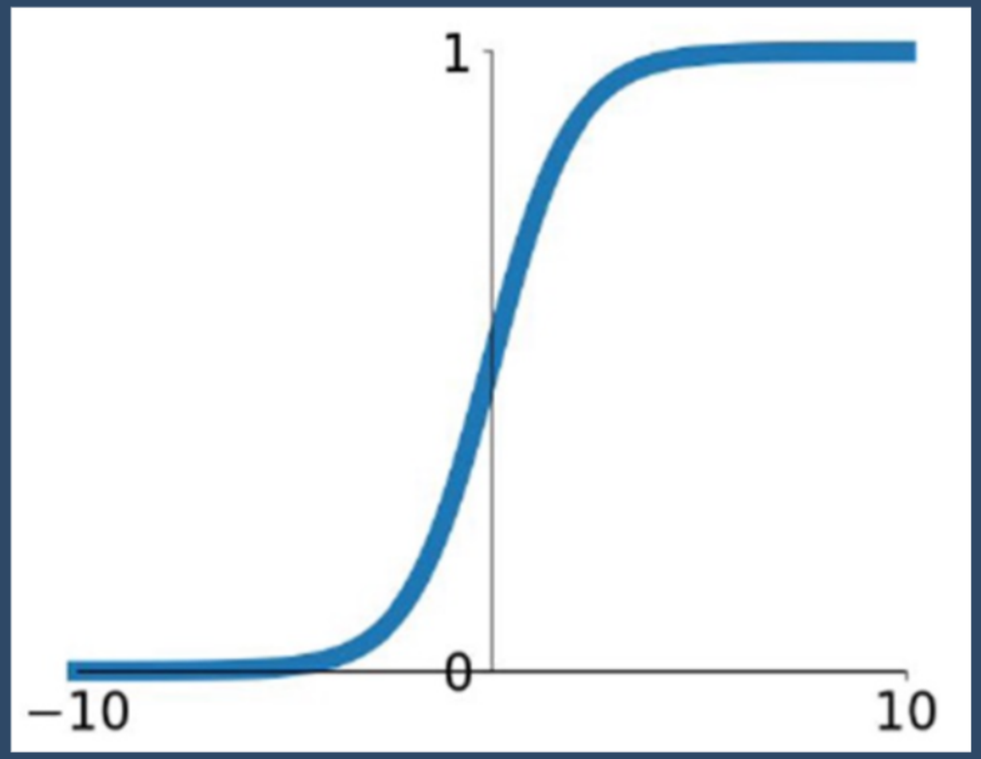
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^{x} - 1) & x < 0 \end{cases}$$



2. Activation function: Sigmoid Function



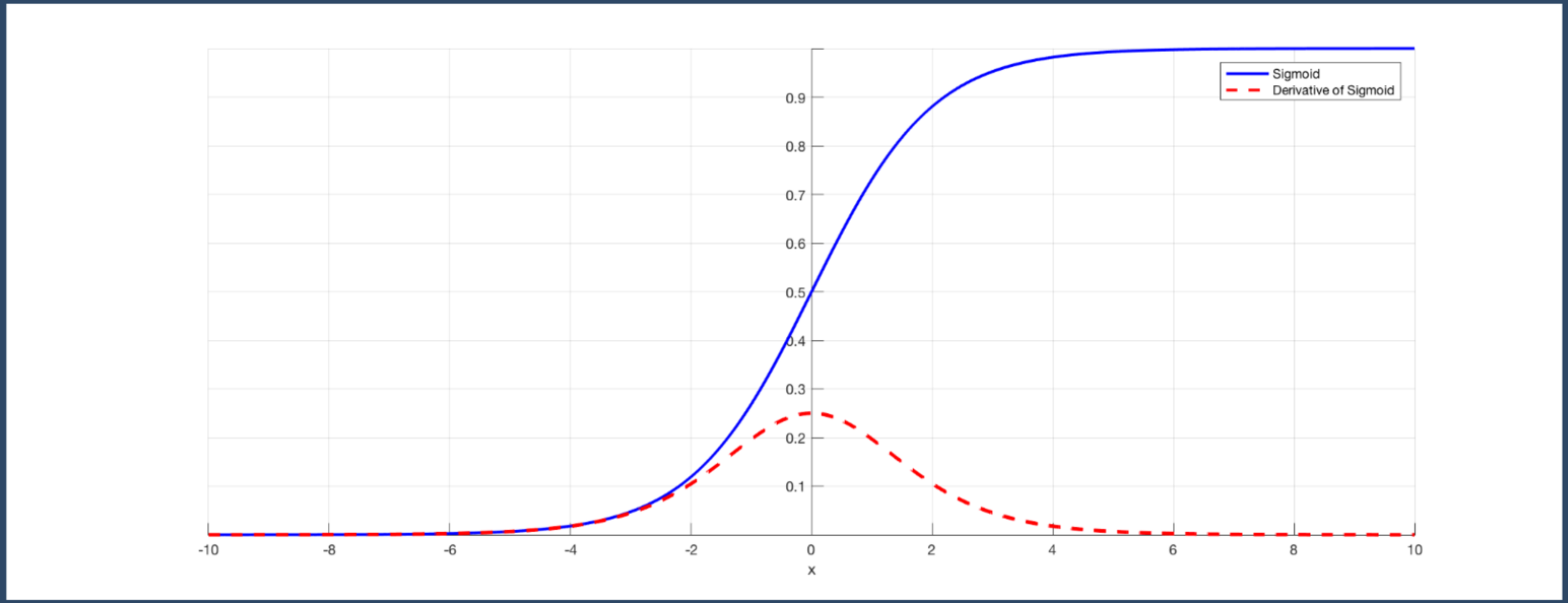
Sigmoid Function

- 숫자를 0과 1 사이의 값으로 변환한다.
- 뉴런의 활동 전위를 잘 해석한 활성화 함수로 유명하다.

3가지 문제점
- Gradient vanishing problem
- Sigmoid outputs are not zero-centered.
- exp() is a bit compute expensive.

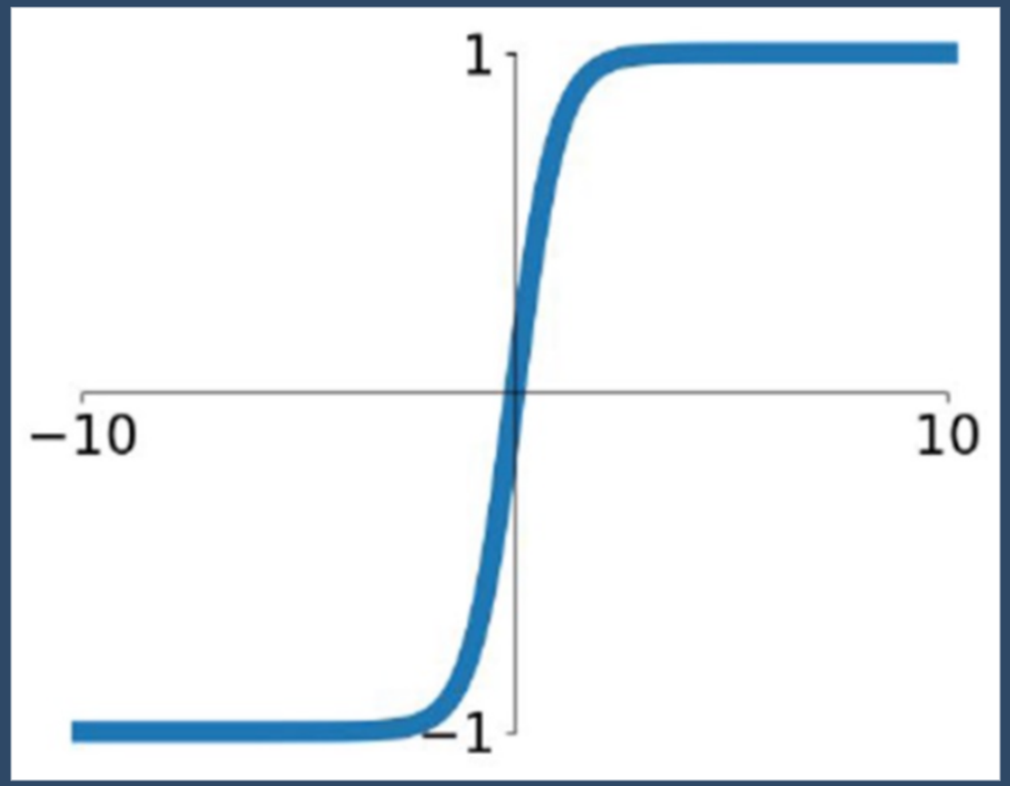
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

2. Activation function: Sigmoid Function



$$\sigma'(x) = \sigma(x)\{1 - \sigma(x)\}$$

2. Activation function: Tanh Function



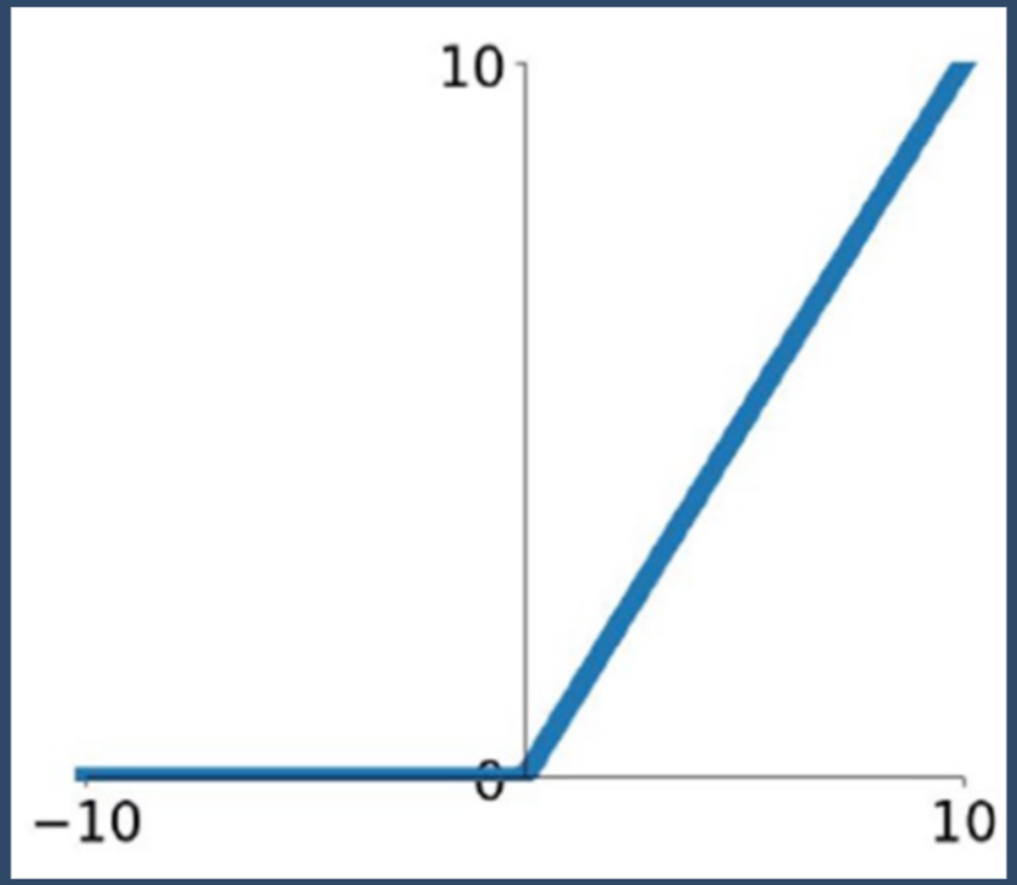
Tanh Function

- 숫자를 -1과 1 사이의 값으로 변환한다.
- Tanh outputs are zero-centered.

2가지 문제점
- still kills gradients when saturated
- exp() is a bit compute expensive

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

2. Activation function: Rectified Linear Unit (ReLU) Function

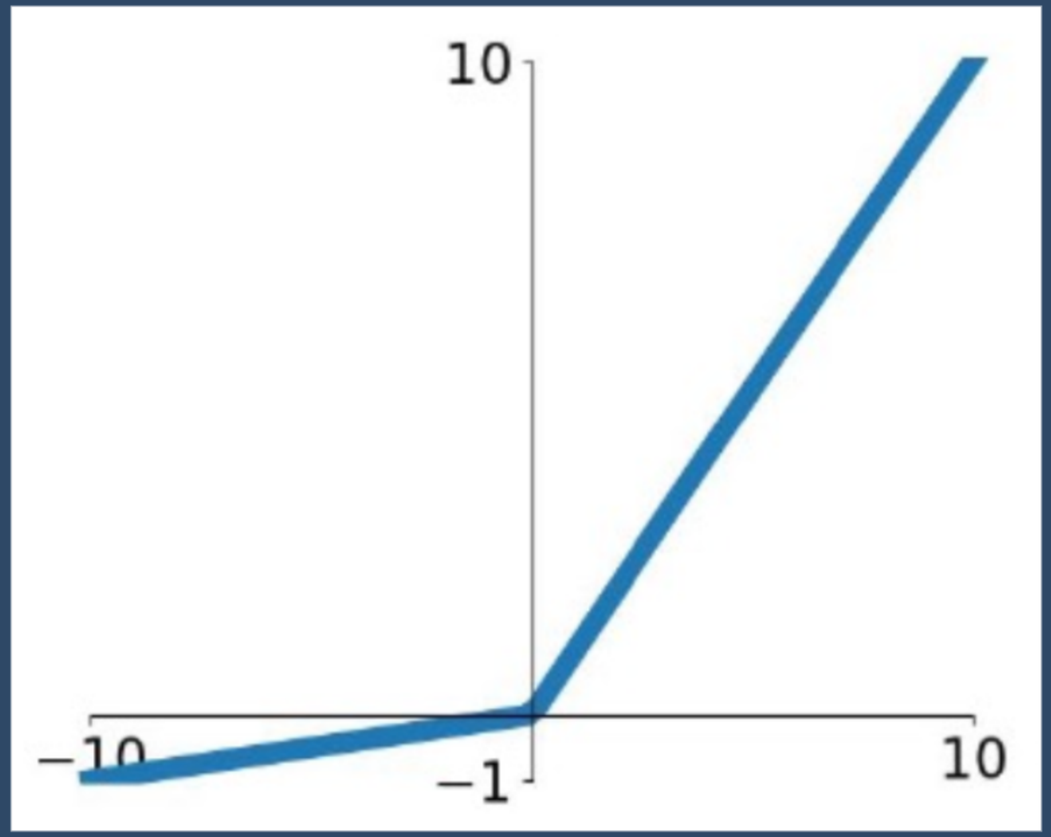


ReLU Function

- Does not saturate (in + region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

2가지 문제점
- Not zero-centered output
- An annoyance: what is the gradient when x < 0?

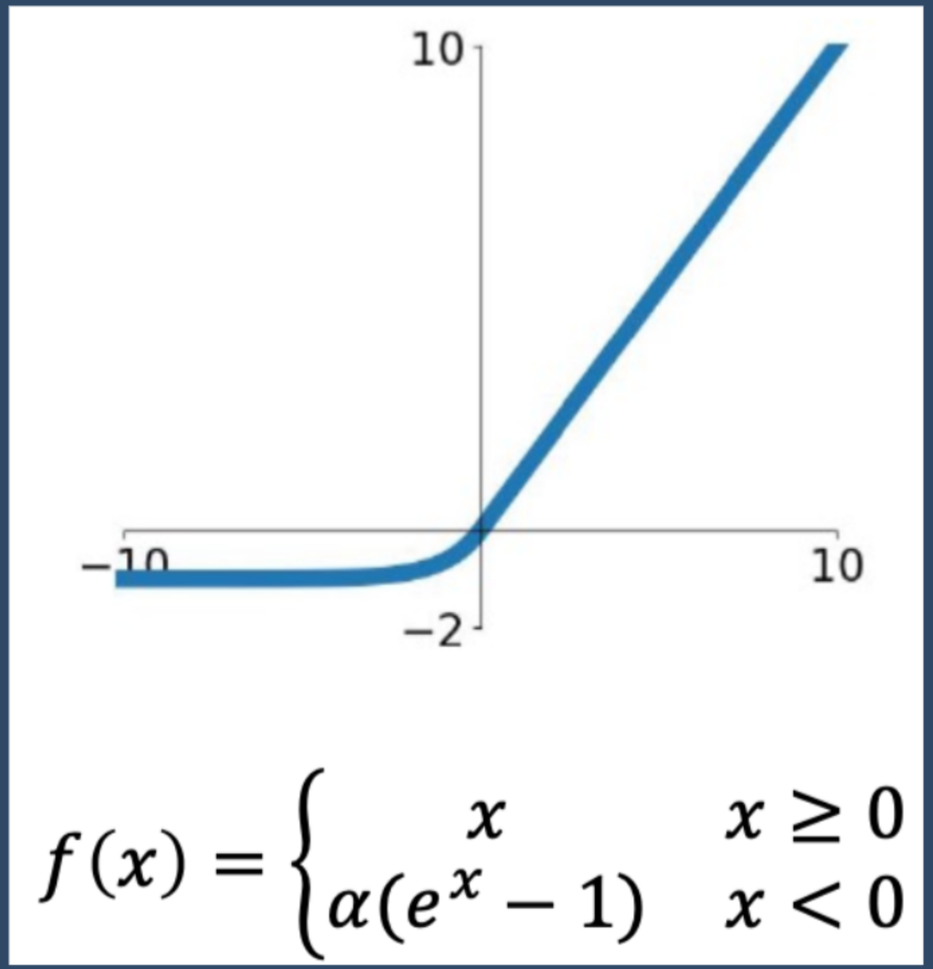
2. Activation function: Leaky ReLU Function



Leaky ReLU Function
- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- will not "die"

$$f(x) = \max(\alpha x, x)$$

2. Activation function: Exponential Linear Units (ELU) Function



ELU Function
- All benefits of ReLU
- Closer to zero mean outputs
- Negative saturation regime compared with Leaky RELU adds some robustness to noise

1가지 문제점
- Computation requires exp()

2. Activation function: Maxout Function

$$\max(W_1^T x + b_1, W_2^T x + b_2)$$

Does not have the basic form of dot product -> nonlinearity

- Generalizes ReLU and Leaky ReLU
- Linear Regime! Does not saturate! Does not die!

- It doubles the number of parameters/neuron

2. Activation function

Practice in Activation Functions

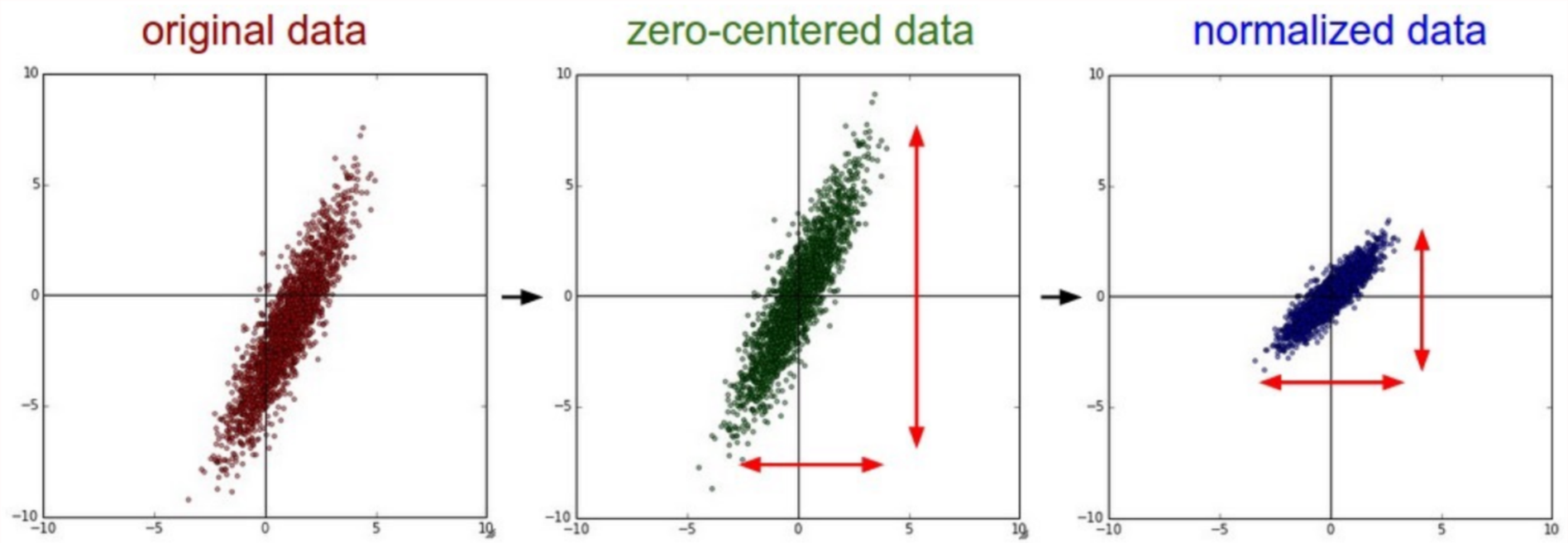
- Use ReLU. Becareful with your learning rates.

- Try out Leaky ReLU / Maxout / ELU.
  to squeeze out some marginal gains

- Don't use sigmoid or tanh.

3. Data preprocessing

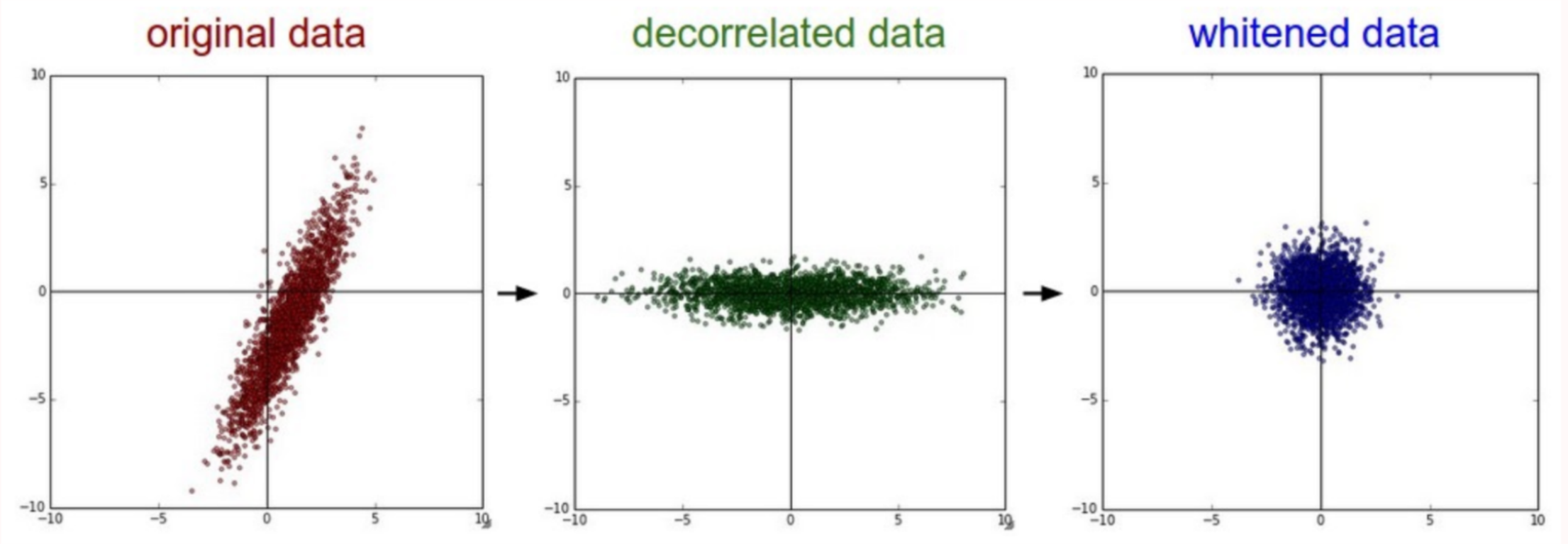
$$x \leftarrow (x - \mu_x) / \sqrt{\sigma_x^2 + \epsilon}$$



Before normalization:
Classification loss very sensitive to changes in weight matrix; hard to optimize.

After normalization:
less sensitive to small changes in weights; easier to optimize.

3. Data preprocessing



decorrelated data: data has diagonal covariance matrix.

whitened data: covariance matrix is the identity matrix.

4. Weight initialization: Random initialization

First idea: Small random numbers; Gaussian with zero mean and 1e-2 standard deviation.

-> All activations become zero.

1.0 standard deviation instead of 1e-2 standard deviation.

-> Almost all neurons completely saturated, either -1 and 1.
Gradients will be all zero -> no learning

4. Weight initialization: Xavier initialization

Reasonable initialization

입력 벡터와 선형 변환에 따른 출력 벡터의 분산이 같도록 한다.

분산은 입력 벡터의 크기.

4. Weight initialization: Xavier initialization

$$Var(s) = Var(\sum_{i}^{n} w_i x_i)$$

$$= \sum_{i}^{n} Var(w_i x_i)$$

$$= \sum_{i}^{n} [\{E(w_i)\}^2 Var(x_i) + \{E(x_i)\}^2 Var(w_i) + Var(x_i)Var(w_i)]$$

$$= \sum_{i}^{n} Var(x_i)Var(w_i)$$

$$= nVar(w)Var(x)$$

$$Var(XY) = \{E(X)\}^2 Var(Y) + \{E(Y)\}^2 Var(X) + Var(X)Var(Y)$$

4. Weight initialization: MSRA initialization

하지만 ReLU를 사용하면 Xavier initialization의 가정이 깨진다.

따라서 분산을 입력 벡터의 크기 / 2를 사용하는 MSRA initialization을 사용한다.

5. Batch normalization

D개의 차원을 가지는 N개의 샘플에 대한 텐서가 존재하면, 각각의 차원에 대하여 평균과 표준편차를 계산한다.

D개의 평균과 표준편차를 통해 모든 값을 normalization해준다.

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$y = \gamma \hat{x} + \beta$$

Test-Time에서는 학습에서 계산했던 평균과 표준편차를 사용한다.

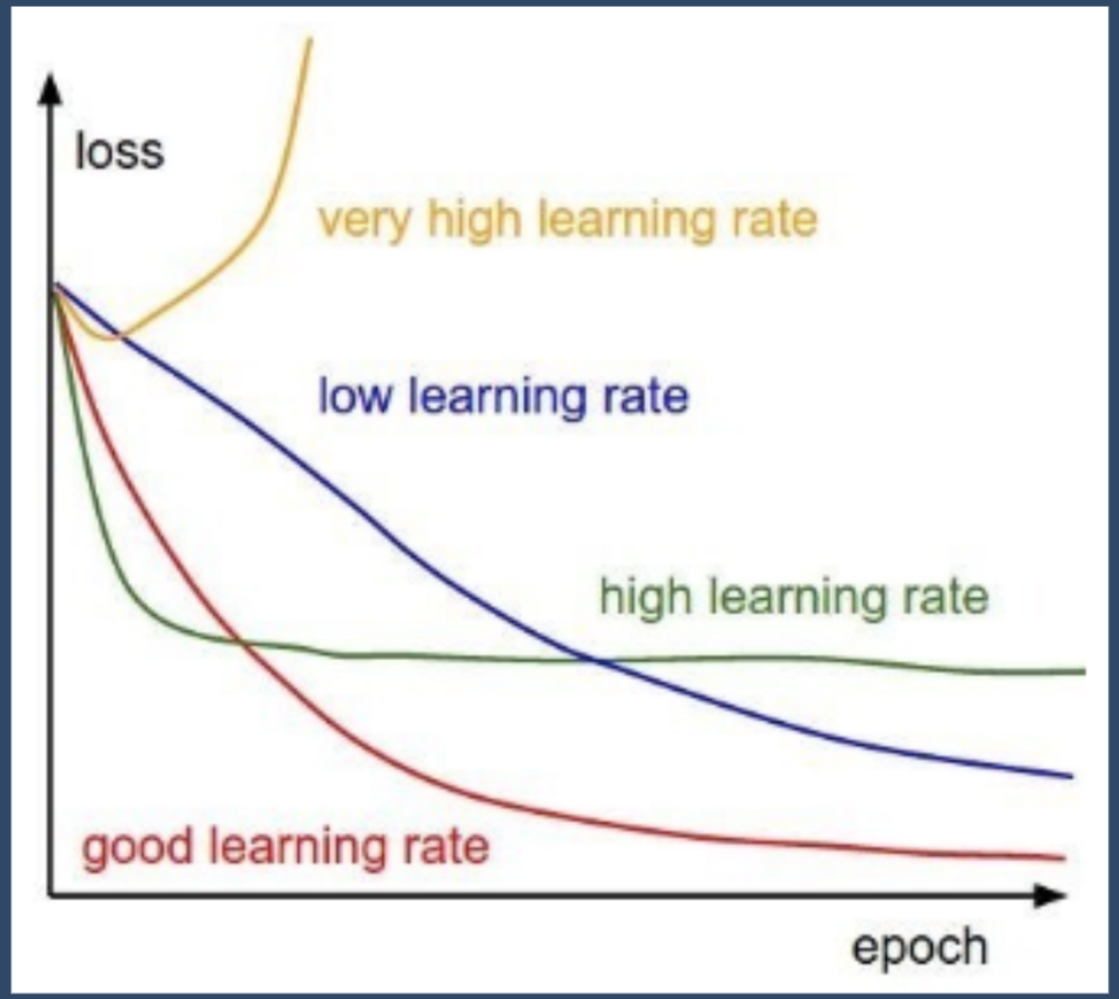
5. Batch normalization

Fully Connected 또는 Convolutional layer와 활성화 함수 사이에서 사용한다.

- Makes deep networks much easier to train
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!

문제점
- Behaves differently during training and testing

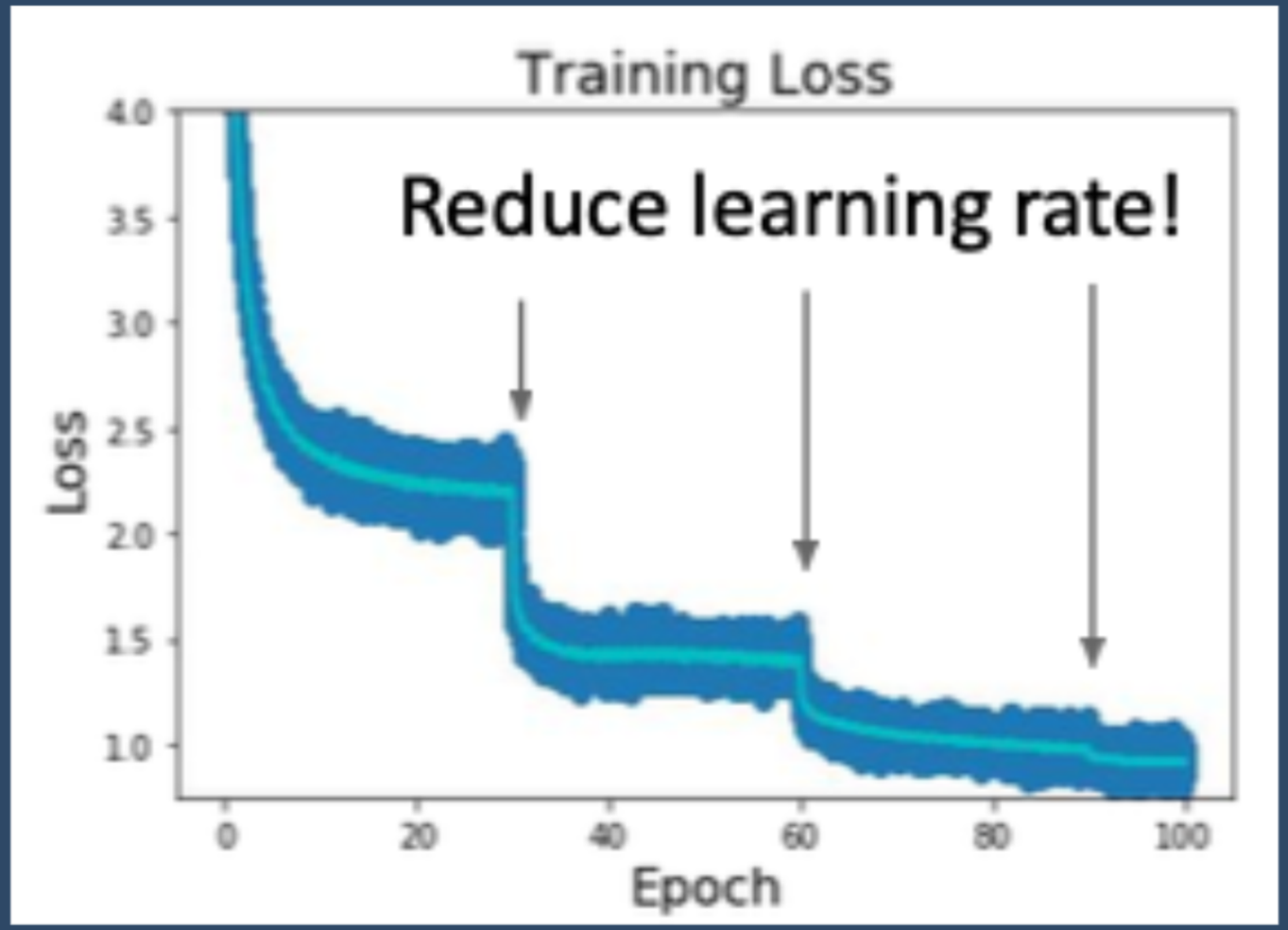
6. Babysitting the learning processing: Learning Rate Schedules



Which one of these learning rates is best to use?

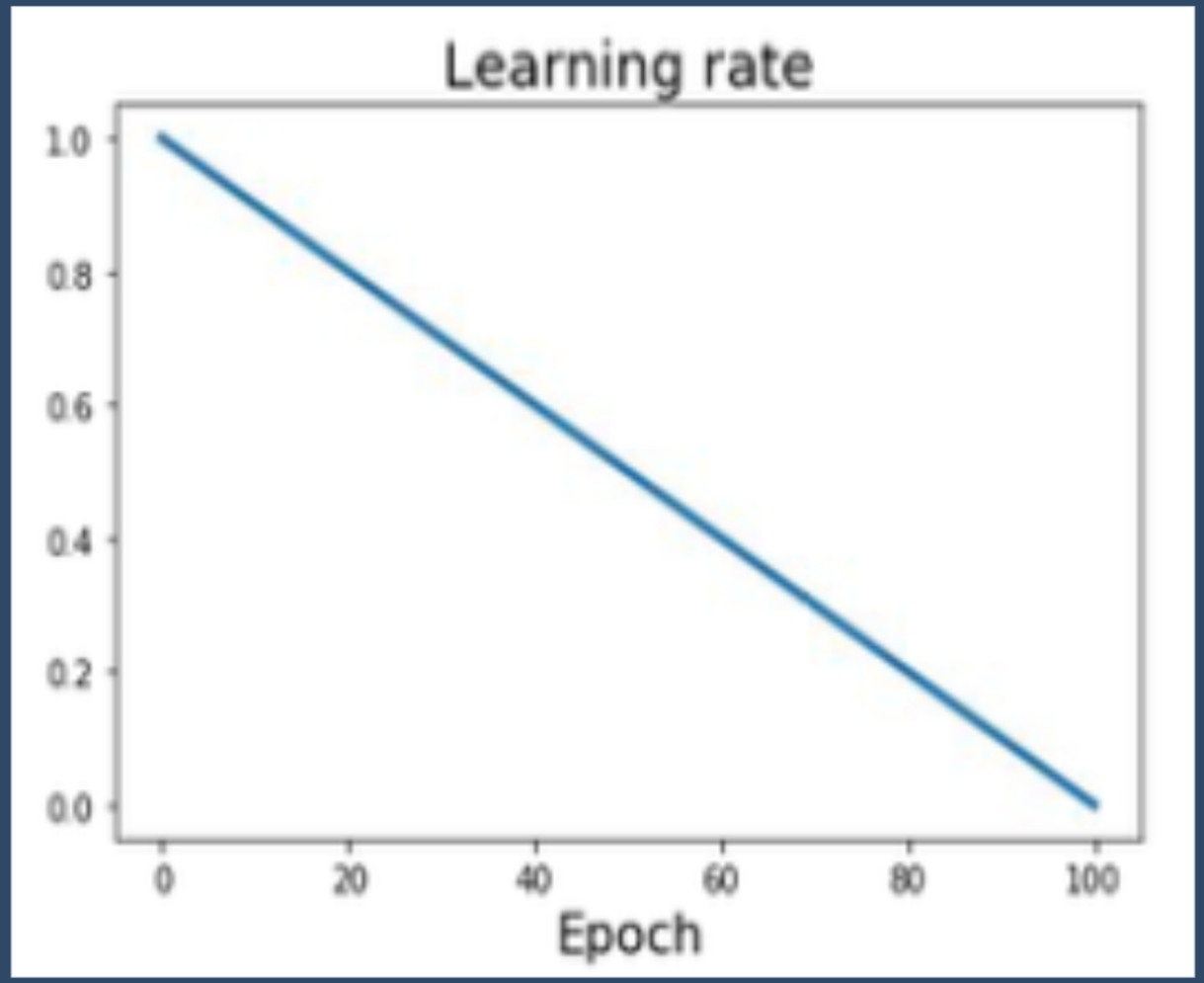
All of them!

6. Babysitting the learning processing: Learning Rate Schedules



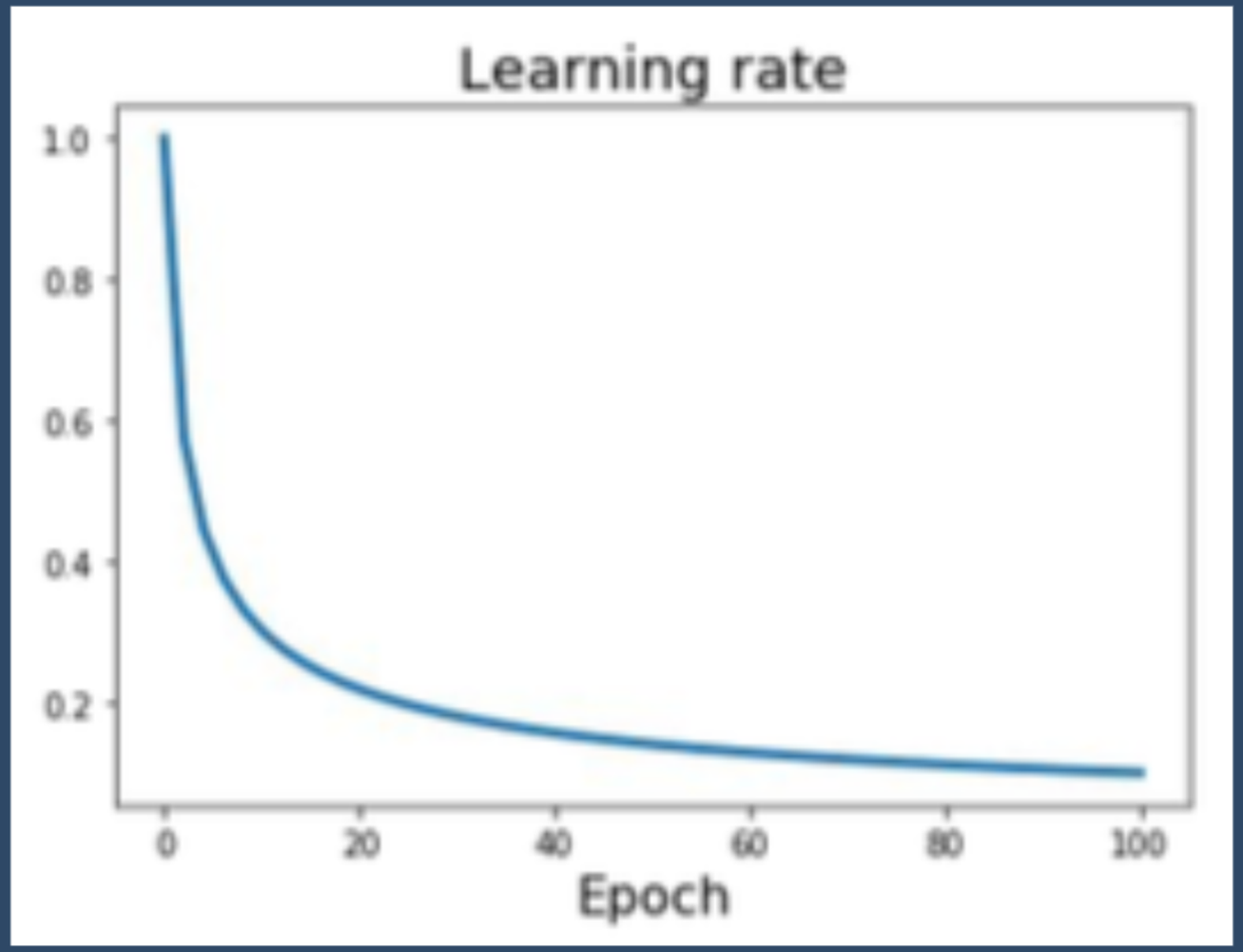
Step:
정해진 포인트에서
learning rate를 감소시킨다.

6. Babysitting the learning processing: Learning Rate Schedules



Linear:
learning rate를 선형적으로
조금씩 계속 감소시킨다.

6. Babysitting the learning processing: Learning Rate Schedules



Inverse Sqrt:

$$\alpha_+ = \alpha/\sqrt{t}$$

7. Hyperparameter optimization

Hyperparameters
- Network architecture
- Learning rate, its decay schedule (decay constant), update type (SGD, Momentum, AdaGrad, RMSProp)
- Regularization strength (L2 penalty, dropout strength)

Some tips and tricks
- Single validation set rather than multiple folds.
- Search on log-scale for learning rate and regularization strength.
- Random search is preferred to grid search.
- Coarse-to-fine search strategy

Thank you for listening

Deep Into Deep