Deep Learning Seminar

5. Training Neural Network

Contents

Reference: lecture note (Fei-Fei Li)
lecture note (Andrew Ng)
모두를 위한 머신러닝 (Sung kim)

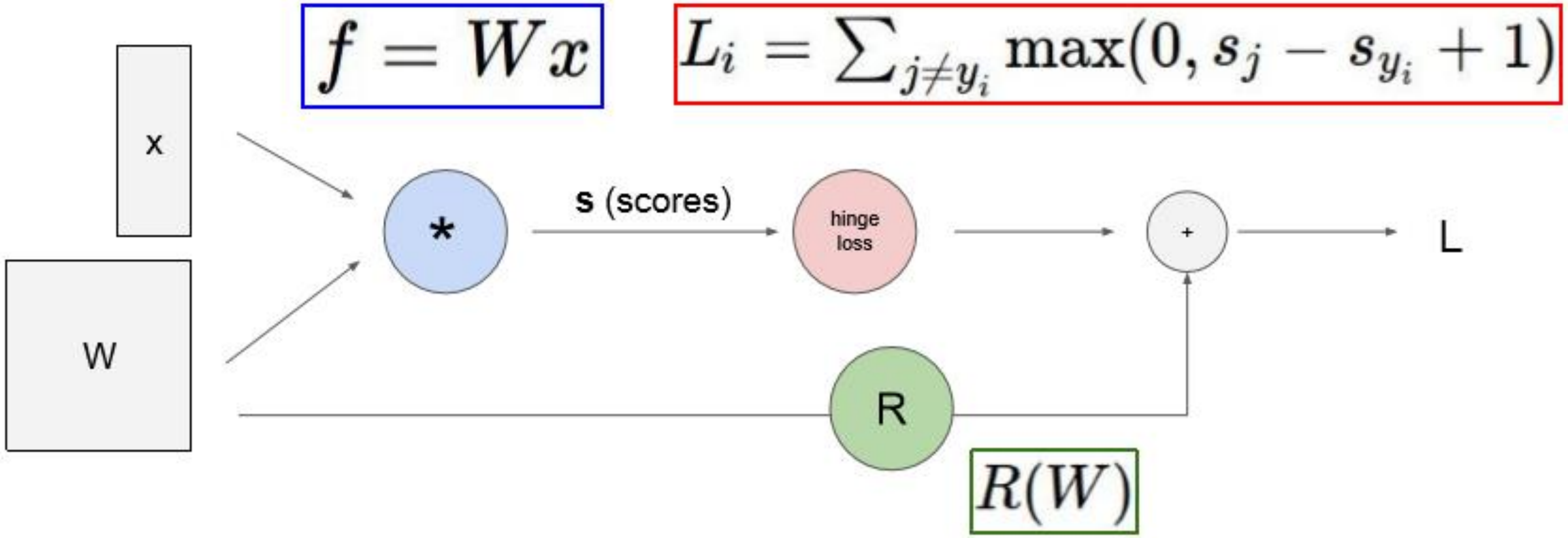
1. Review

1-1) Classification

1-2) Learning network

Classification

1. Linear Classifier

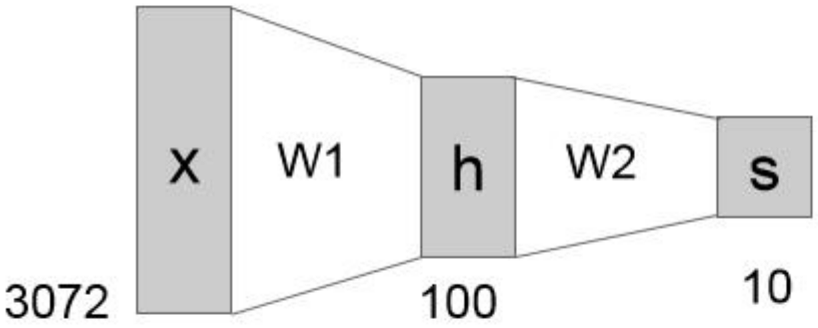


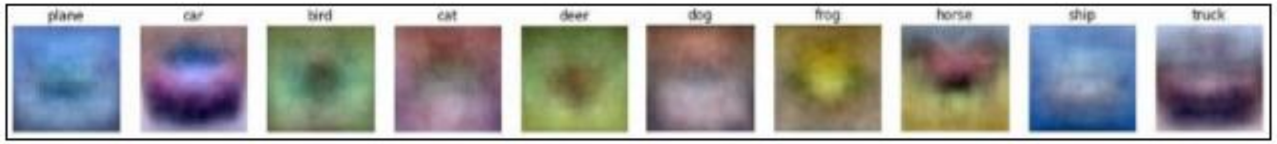
Classification

2. Fully Connected Network

Linear score function: $f = Wx$

2-layer Neural Network $f = W_2 \max(0, W_1 x)$





Classification

3. Convolutional Neural Network

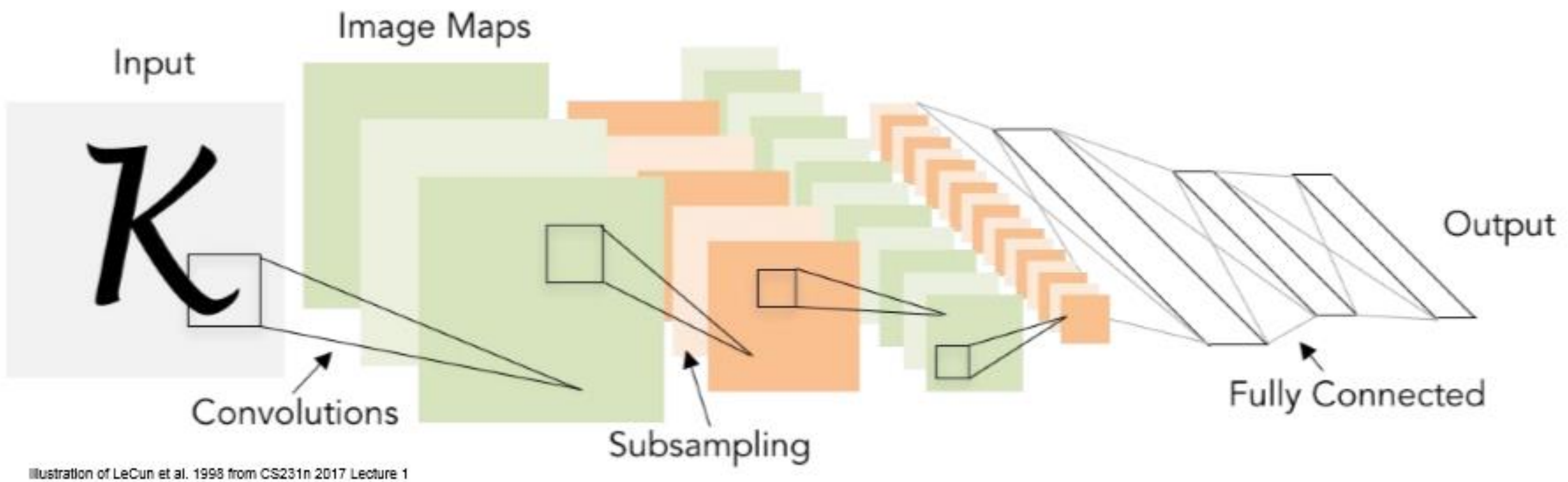
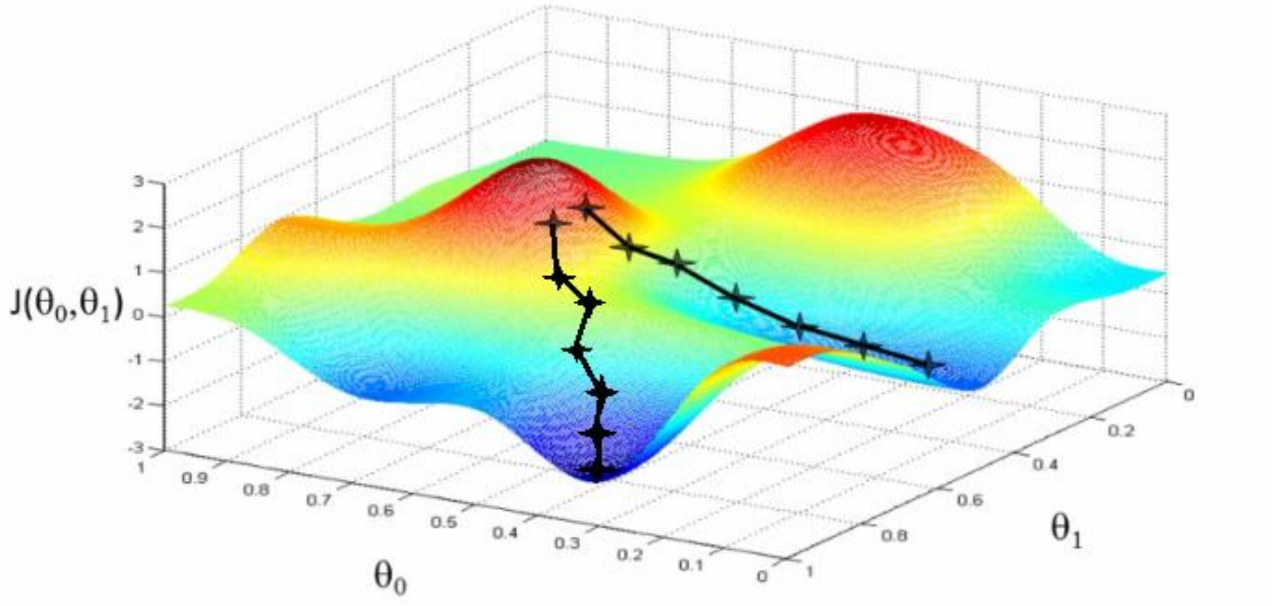
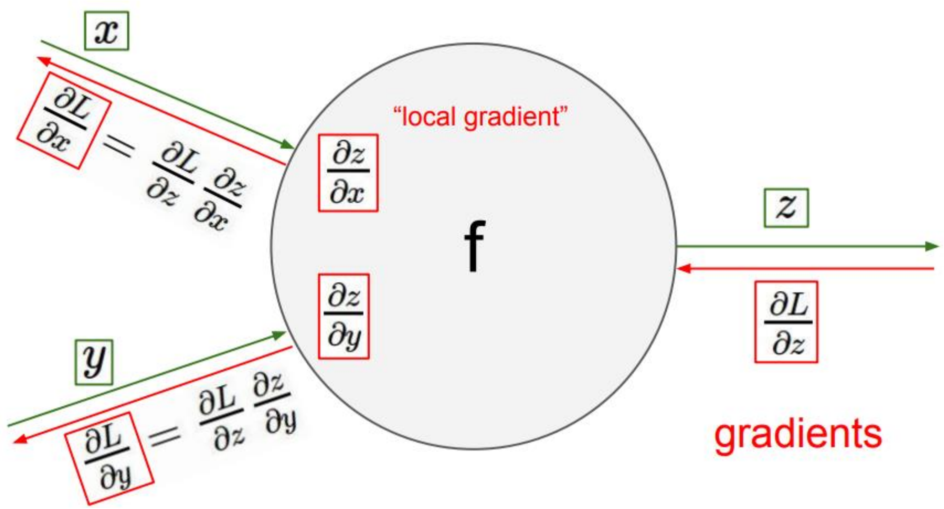


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Learning Network



Optimizer



Backpropagtaion

Learning Network

- Deep Learning Pipeline

1. Training Data Loading
2. Training Data Augmentation
3. Deep Neural Network Training with Training Data
4. Deep Neural Network Testing with Testing Data
5. Inference with verified Deep Neural Network

Learning Network

1. Training Data Loading
2. Training Data Augmentation
3. Deep Neural Network Training with Training Data

Loop:

1. **Sample** a batch of data
2. **Forward** prop it through the graph (network), get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient

2. Training neural network (part I)

2-1) Training overview

2-2) Activation functions

2-3) Data preprocessing

2-4) Weight initialization

2-5) Batch normalization

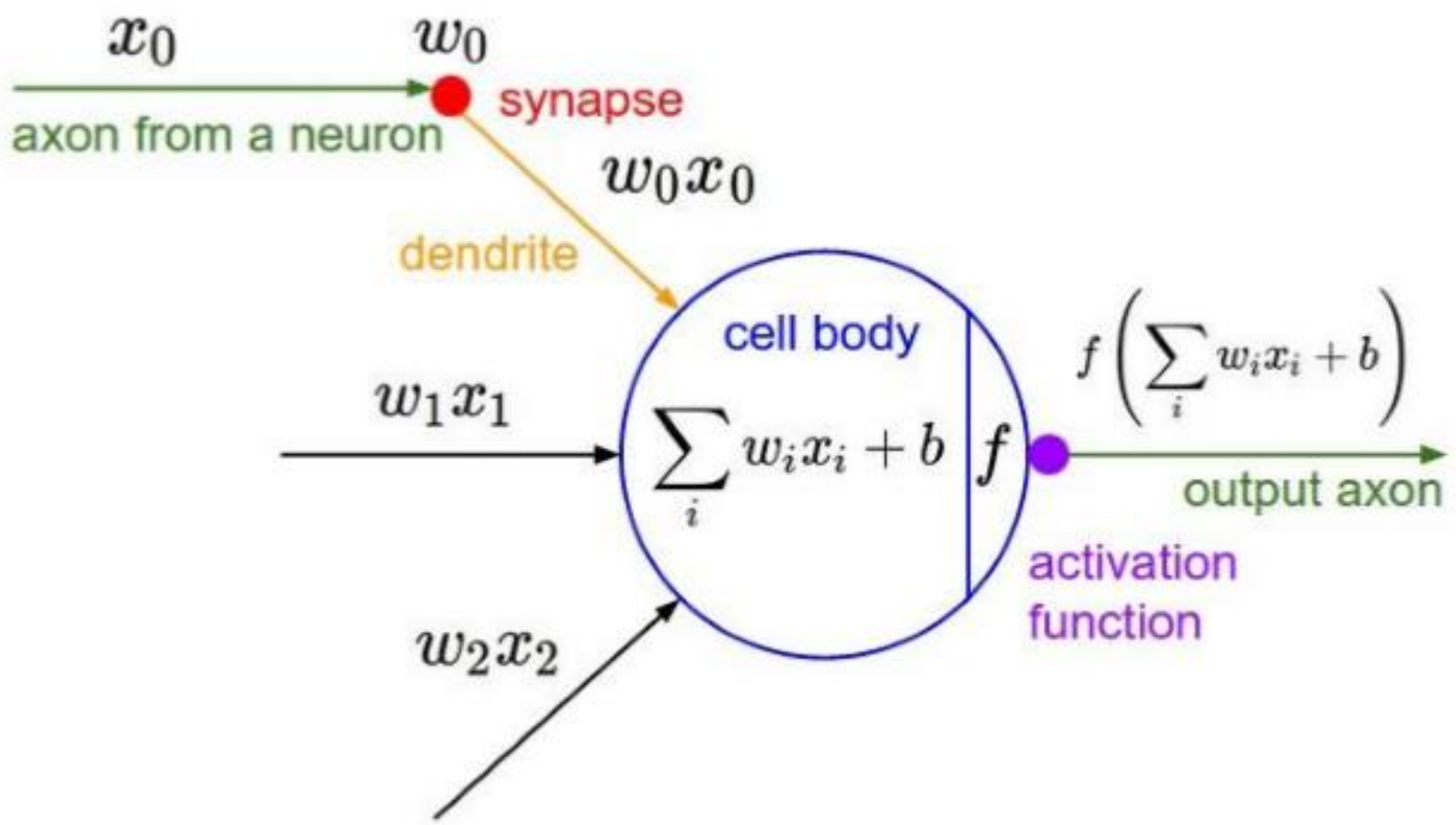
2-6) Babysitting the learning process

2-7) Hyperparameter optimization

Training overview

1. **One time setup**
   activation functions, preprocessing, weight initialization, regularization, gradient checking
2. **Training dynamics**
   babysitting the learning process,
   parameter updates, hyperparameter optimization
3. **Evaluation**
   model ensembles

Activation Functions



Activation Functions

- Limit of linear classification

Layer 1: $y_1 = w_1 x + b_1$

Layer 2: $y_2 = w_2 y_1 + b_2$

Layer 3: $y_3 = w_3 y_2 + b_3$

…

Layer n: $y_n = w_n y_{n-1} + b_n$

Activation Functions

- Limit of linear classification

Layer 1: $y_1 = w_1 x + b_1$

Layer 2: $y_2 = w_2 y_1 + b_2 \quad = \quad w_2(w_1 x + b_1) + b_2$

Layer 3: $y_3 = w_3 y_2 + b_3 \quad = \quad w_3(w_2(w_1 x + b_1) + b_2) + b_3$

…

Layer n: $y_n = w_n y_{n-1} + b_n \quad = \quad w_n(w_{n-1}(w_{n-2}(\ldots w_1 x + b_1) + \cdots) + b_n$

Activation Functions

- Limit of linear classification

Layer 1: $\mathrm{y}_1 = \mathrm{w}_1\mathrm{x} + \mathrm{b}_1$

Layer 2: $\mathrm{y}_2 = \mathrm{w}_2\mathrm{y}_1 + \mathrm{b}_2 \quad = \quad w_2(w_1x + b_1) + b_2 \quad = \quad c_2x + d_2$

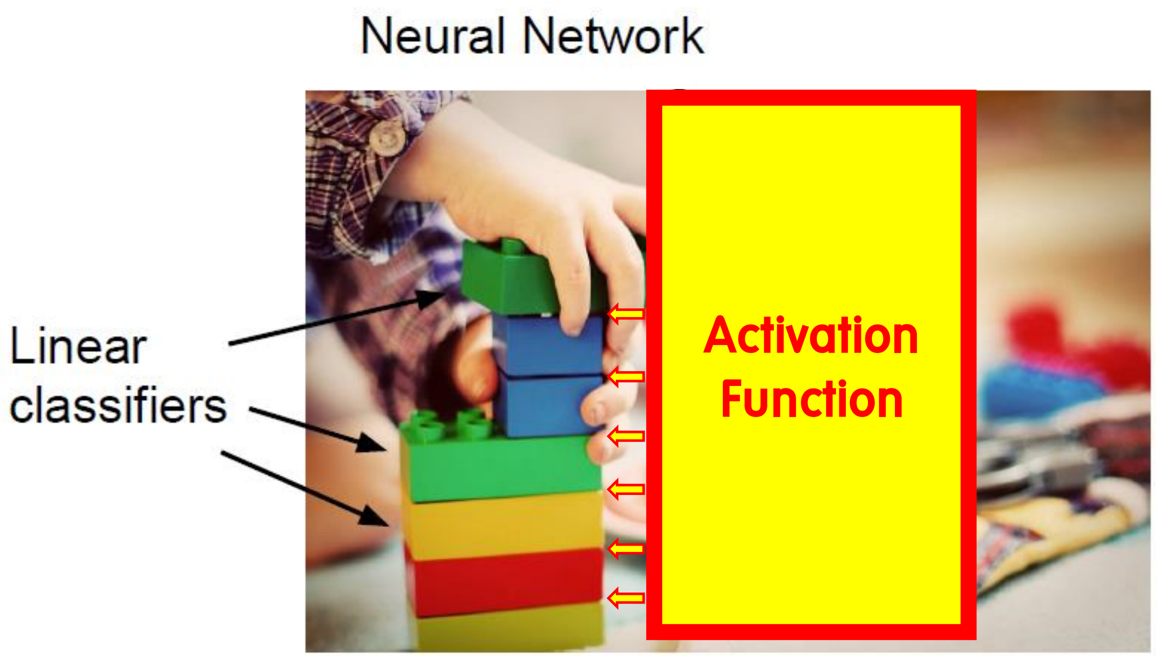
Layer 3: $\mathrm{y}_3 = \mathrm{w}_3\mathrm{y}_2 + \mathrm{b}_3 \quad = \quad w_3(w_2(w_1x + b_1) + b_2) + b_3 \quad = \quad c_3x + d_3$

...

Layer n: $\boxed{\mathrm{y}_\mathrm{n}} = \mathrm{w}_\mathrm{n}\mathrm{y}_{\mathrm{n}-1} + \mathrm{b}_\mathrm{n} \quad = \quad w_n(w_{n-1}(w_{n-2}(\dots w_1x + b_1) + \cdots) + b_n \quad = \quad \boxed{c_nx + d_n}$

Activation Functions

Generate **non-linear mappings** from inputs to outputs



Activation Functions

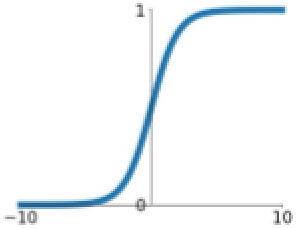
Layer 1:	$\mathrm{y}_1 = \mathrm{w}_1\mathrm{x} + \mathrm{b}_1$			
Act.	$\widehat{\mathrm{y}_1} = \ tanh(\mathrm{y}_1)$	$= \ \tanh(w_1 x + b_1)$		
Layer 2:	$\mathrm{y}_2 = \mathrm{w}_2\widehat{\mathrm{y}_1} + \mathrm{b}_2$	$= \ w_2(\tanh(w_1 x + b_1)) + b_2$	$\neq$	$c_2 x + d_2$

(Activation function: tanh)

Activation Functions

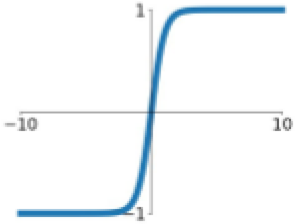
Sigmoid

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



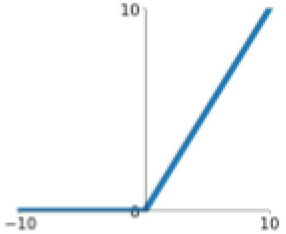
tanh

$\tanh(x)$



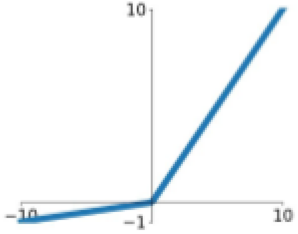
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

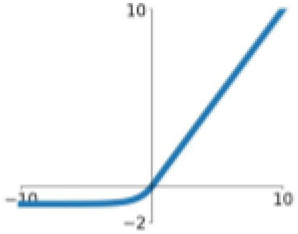


Maxout

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

ELU

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



Activation Functions

- Summary

1. Use ReLU. Be careful with your learning rates
2. Don' t use Sigmoid

Data Preprocessing

- RGB Image range: 0~255
- Network input range: 0~1 (Recommandable)

In practice,

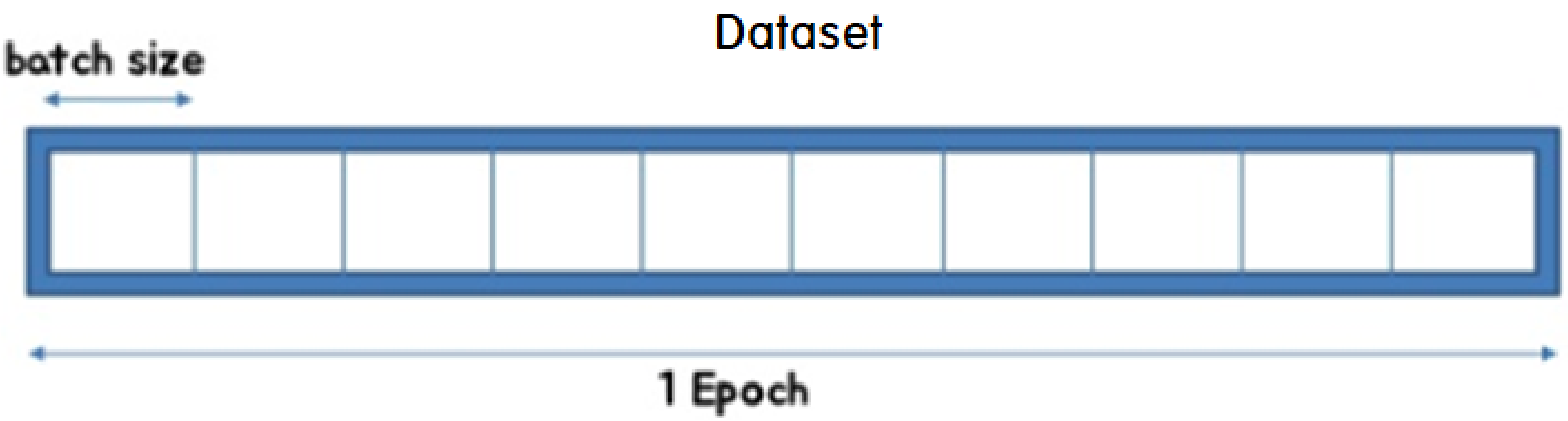
network_input = image_matrix / 255

Data Preprocessing

- Data augmentation

Batch Normalization

- Batch (=mini batch)



Batch Normalization

- Impact of batch size on error

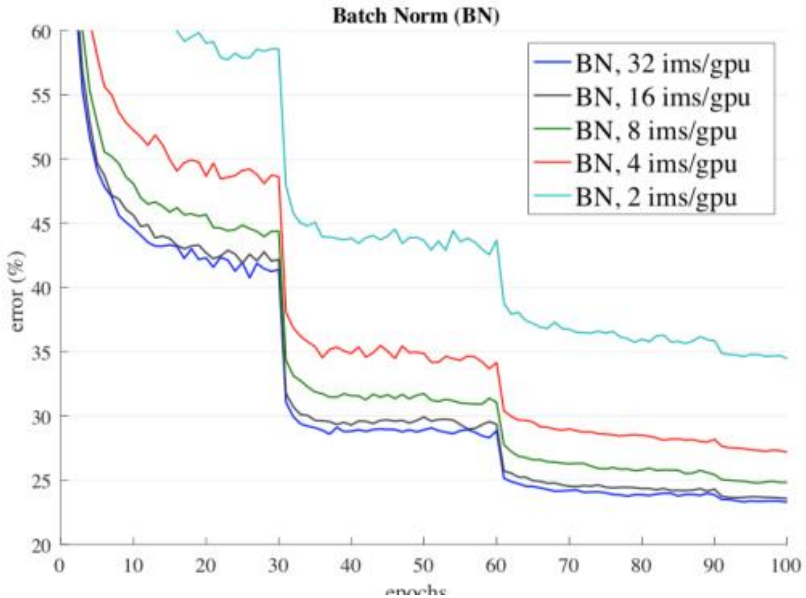
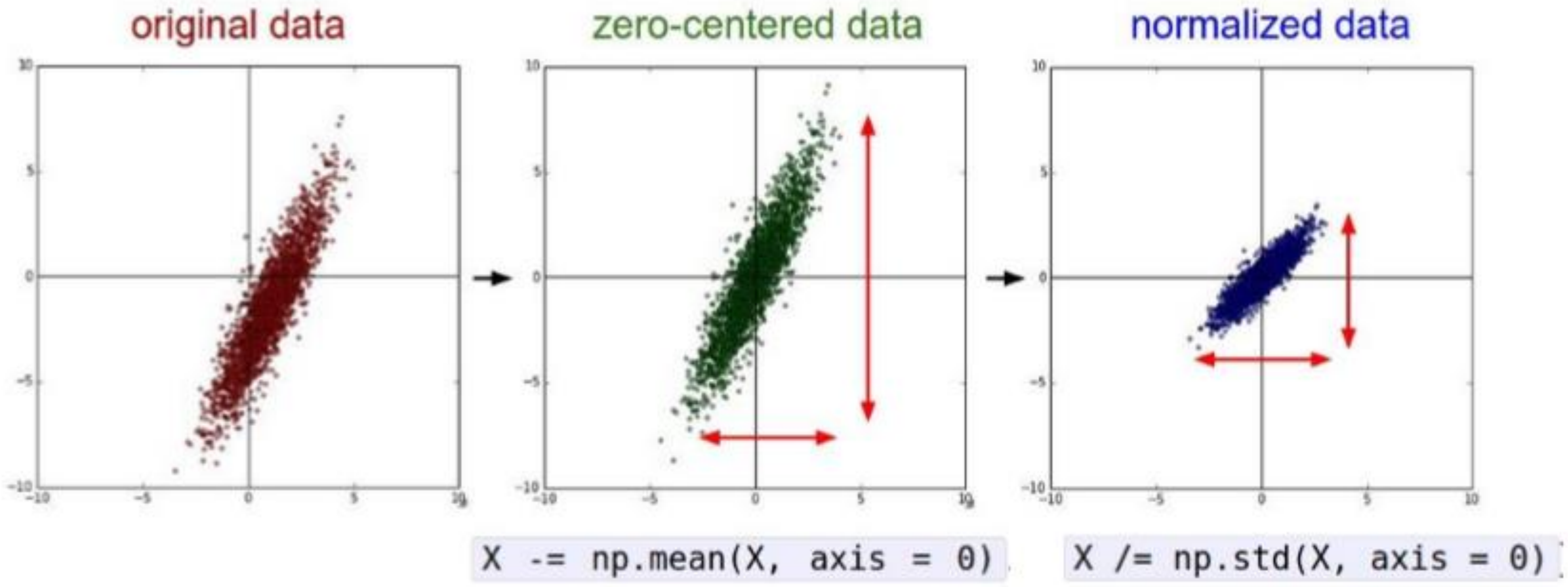


Figure 5. **Sensitivity to batch sizes**: ResNet-50's validation error of BN (left) and GN (right) trained with 32, 16, 8, 4, and 2 images/GPU

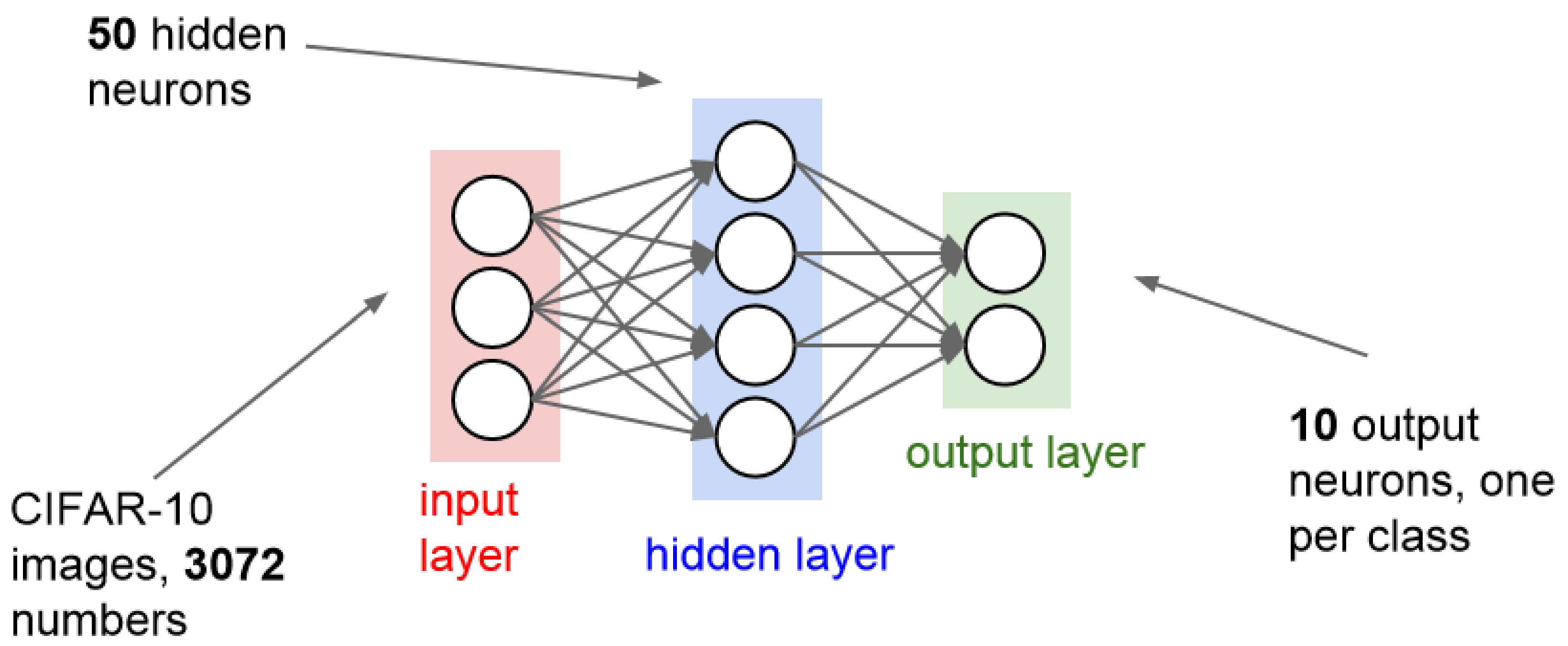
Babysitting the Learning Process

- Step1: Preprocessing the data (+ data augmentation)



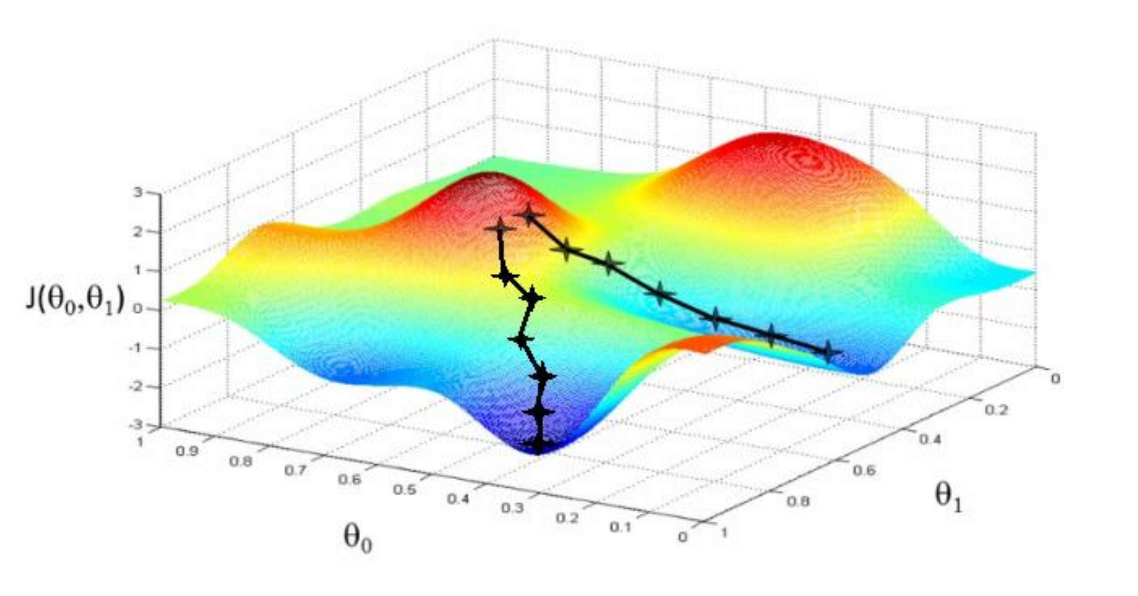
Babysitting the Learning Process

- Step2: Design the network architecture



Babysitting the Learning Process

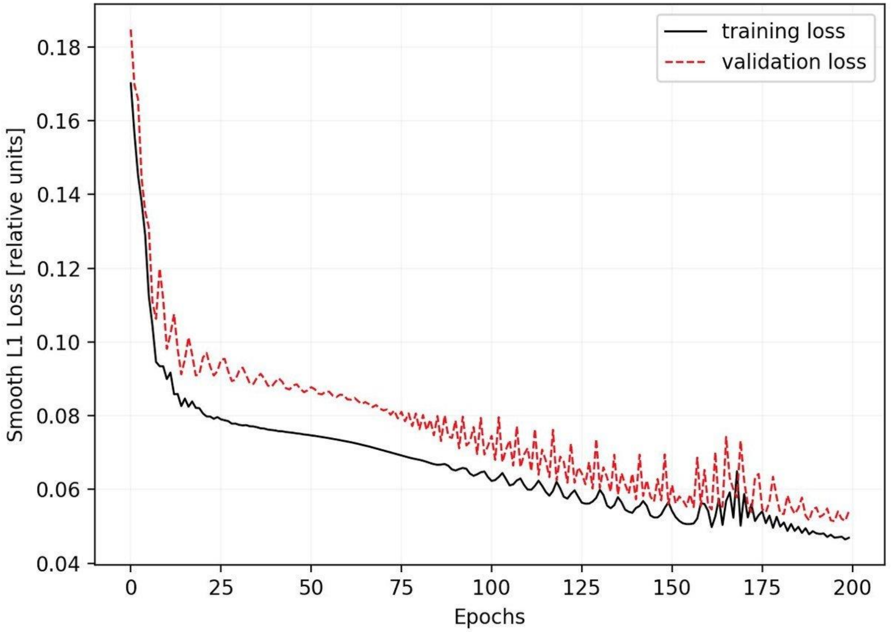
- Step3: Design loss function / Optimizer (lr)



Babysitting the Learning Process

- Step4: Train model and analyze results

Visualize train & validation loss

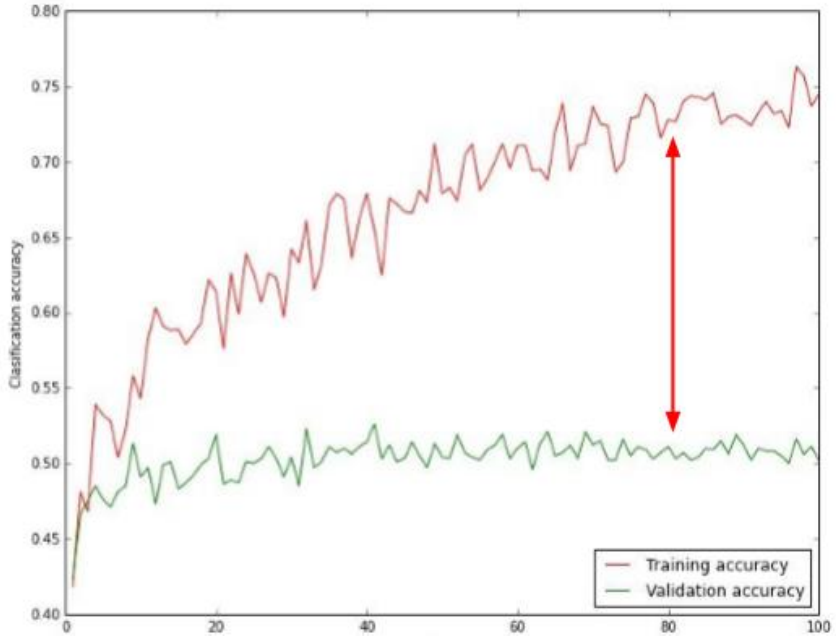



```
Finished epoch 1 / 200: cost 2.302603, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 2 / 200: cost 2.302258, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 3 / 200: cost 2.301849, train: 0.600000, val 0.600000, lr 1.000000e-03
```

Babysitting the Learning Process

- Step4: Train model and analyze results

Check overfitting



big gap = overfitting

=> increase regularization strength?

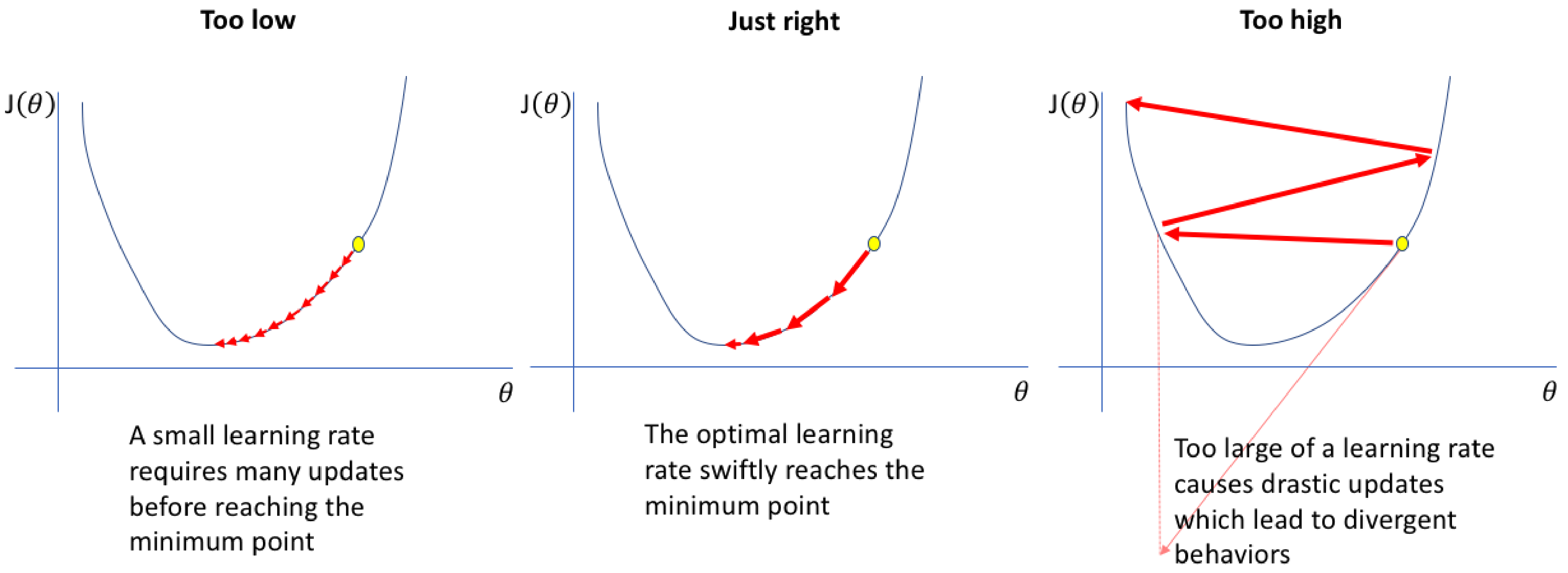
no gap

=> increase model capacity?

Babysitting the Learning Process

- Step4: Train model and analyze results

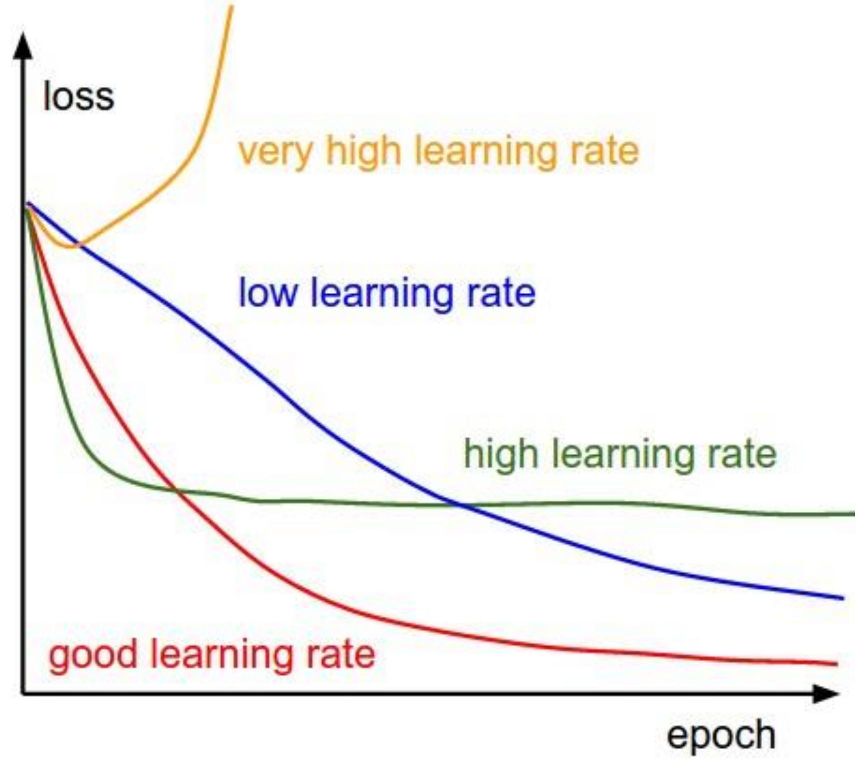
Check learning rate



Babysitting the Learning Process

- Step4: Train model and analyze results

Check learning rate



Hyperparameter Optimization

Choices about the algorithm that we set rather than learn

(≈ Heuristic Values)

ex)

- Initial learning rate
- learning rate decay
- batch size
- epoch
- number of layer
- convolutional kernel size
- pooling type
- activation function
- upsampling method,
- optimizer
- data augmentation parameters
- over-sampling ratio
- k-fold cross validation
- etc...

Summary (Training neural network)

• Activation func.	(Use ReLU)
• Data normalization	(Divide 255)
• Data augmentation	(Must do it)
• Weight Initialization	(Xavier init.)
• Batch Normalization	(Must do it)

- Babysitting the Learning process
- Hyperparameter optimization