Deep Learning Seminar

6. Training Neural Network

HA SEUNG HYUN

Contents

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

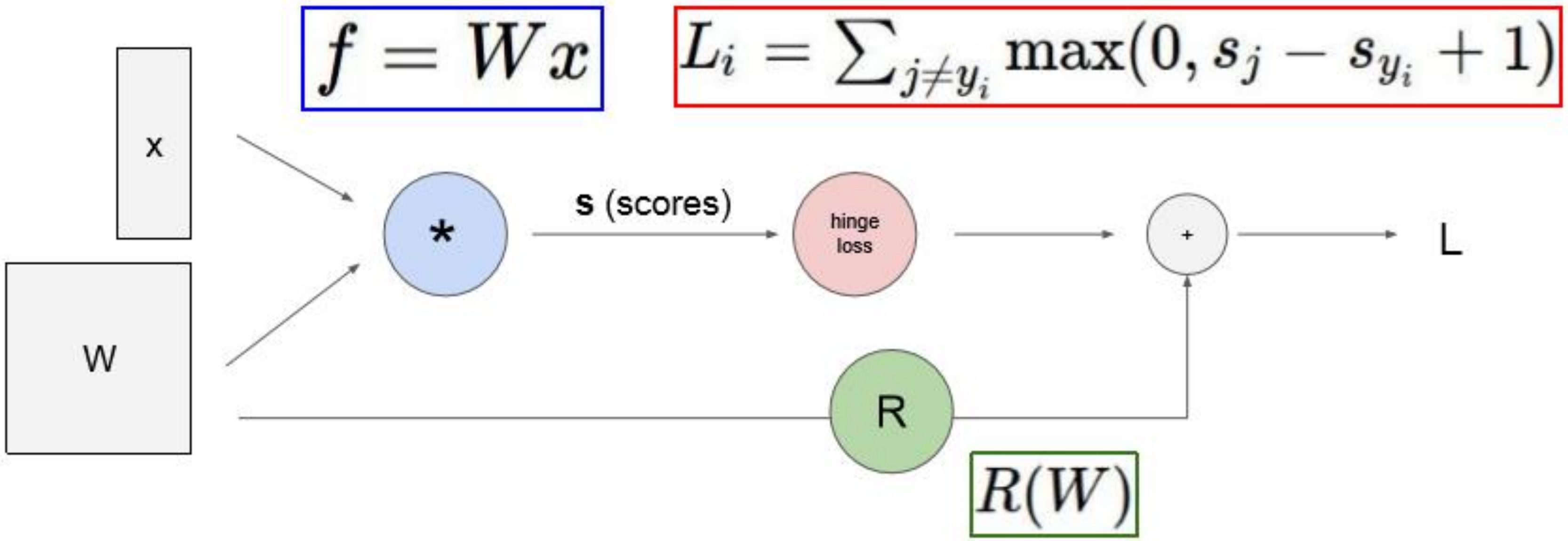
1. Review

1-1) Classification

1-2) Learning network

Classification

1. Linear Classifier



Classification

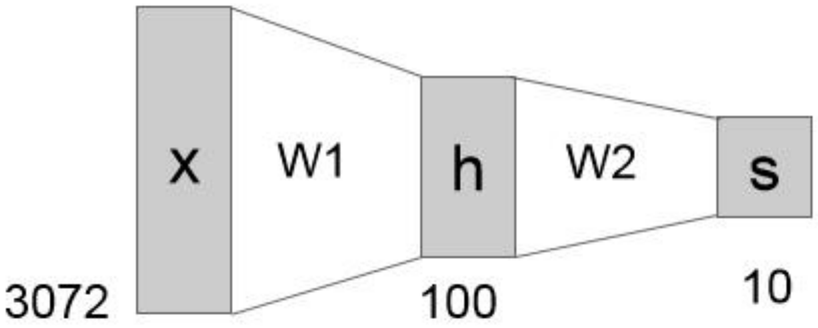
단점
1. 공간 정보를 잃어버린다.
2. computation cost ↑, 연산량이 많기 때문에

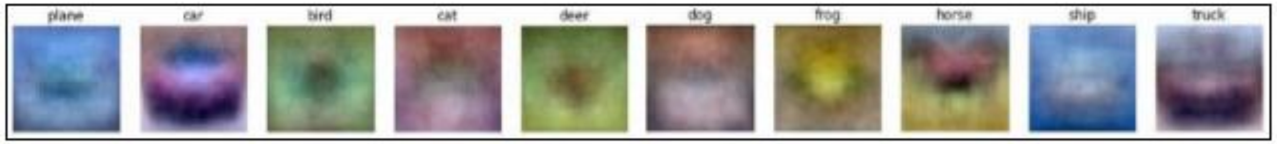
2. Fully Connected Network

FCL 를 여러 개 쌓은 것

Linear score function: $f = Wx$

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





Classification

filter라는 조그만한 창문이 이미지를 훑고 지나간다
1. 공간 정보 유지
2. 작은 부분을 보기 때문에 computation cost ↓

3. Convolutional Neural Network

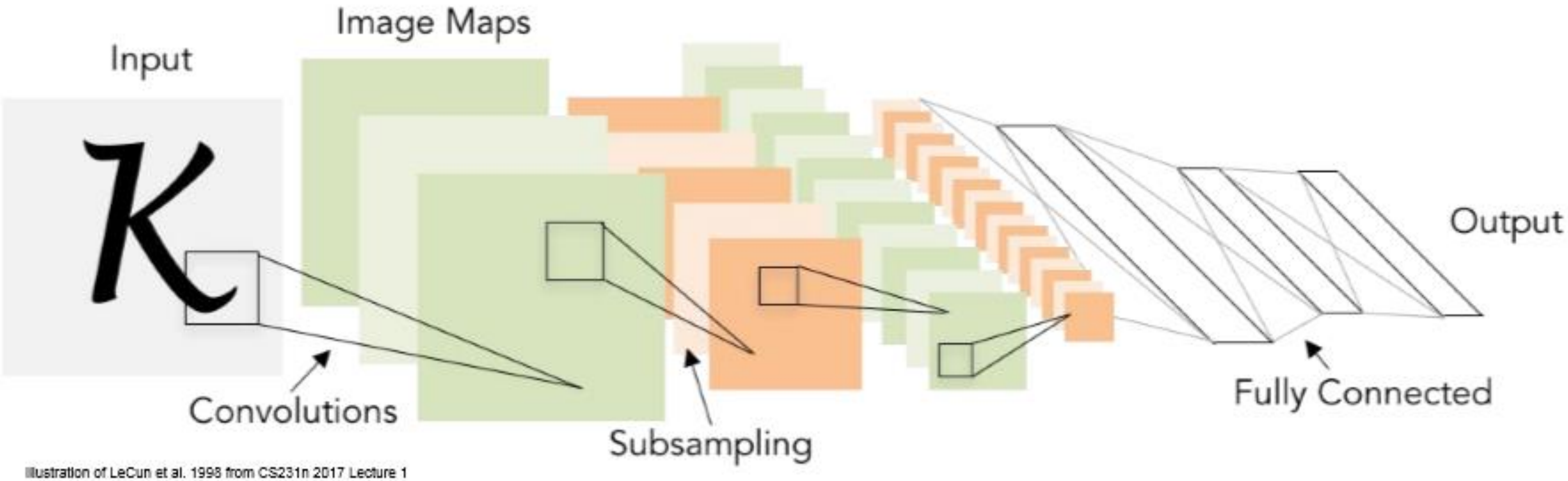
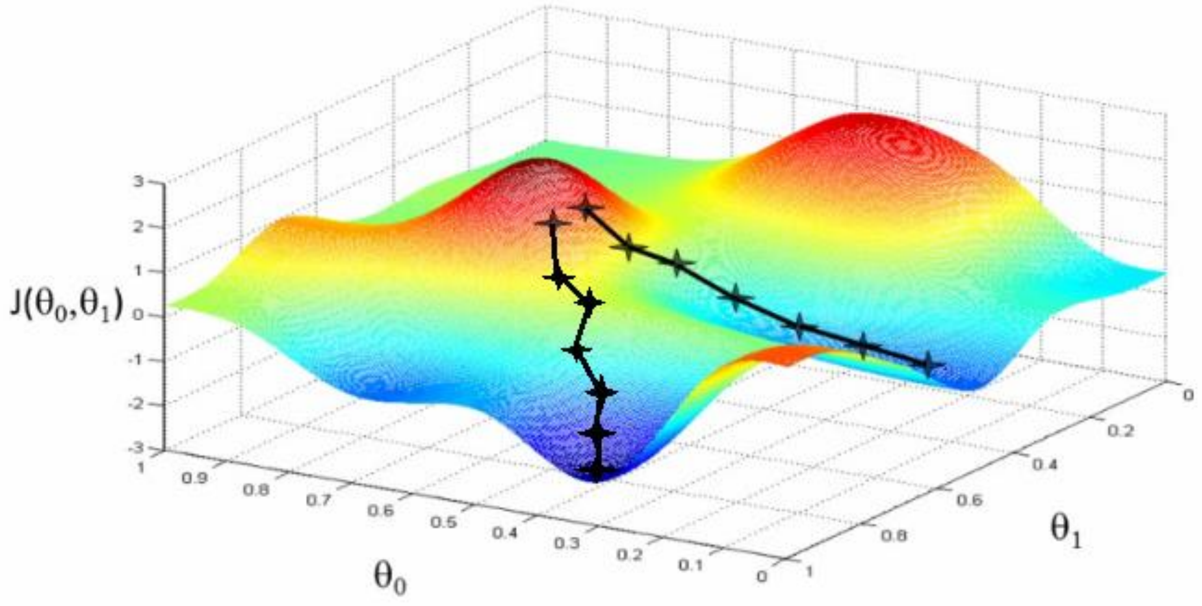


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

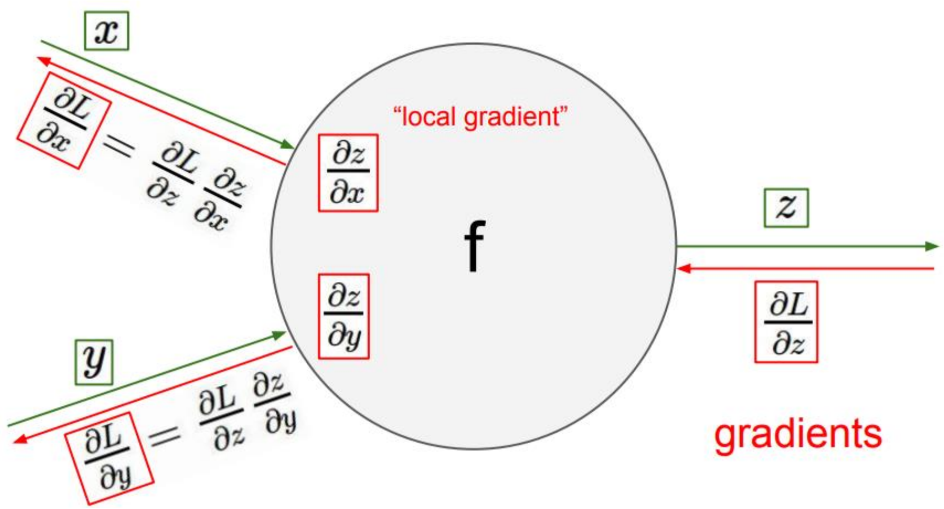
Learning Network

하산 하는 방법 (학습 방향)
Back Propagation
  loss.backward()
  model.step()

SGD
S : batch_size
- batch_size마다 loss값이 나온다.



Optimizer



Backpropagtaion

Learning Network

이 부분을 반복한다는 것은
Epoch
Batch_size
와 관련이 있다.

- Deep Learning Pipeline

반복

1. Training Data Loading　1_1. 물리적 저장. ToTensor()
   1_2. Loader 할때 100, 64. 즉 batch size 별로 로더하게 만들고 데이터 섞어준다.

2. Training Data Augmentation　1_2.를 하면서 같이 일어남. transforms

3. Deep Neural Network Training with Training Data and Validation Data

ex) 문제집을 100권 푼다라고 하면 문제집 10권씩 풀 때마다 모의고사(validation test data) 보게 한다.

단. validation 은 학습 대상이 아니기 때문에 loss값만 얻고 backward 하지 않는다.

4. Deep Neural Network Testing with Testing Data

5. Inference with verified Deep Neural Network

test → loss 0 ↓
acc 100 ↑

상용화 직접 마지막 실용 테스트로써 정답(label)없이 test

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

어떻게 학습 시키는 것이 가장 효과적인 방법인가?

2. Training neural network (part I)

"목표 : Accuracy 높여야한다."

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

동일한 Network라고 해도 어떻게 학습하냐에 따라서 성능 처리 차이가 심하다

Training overview

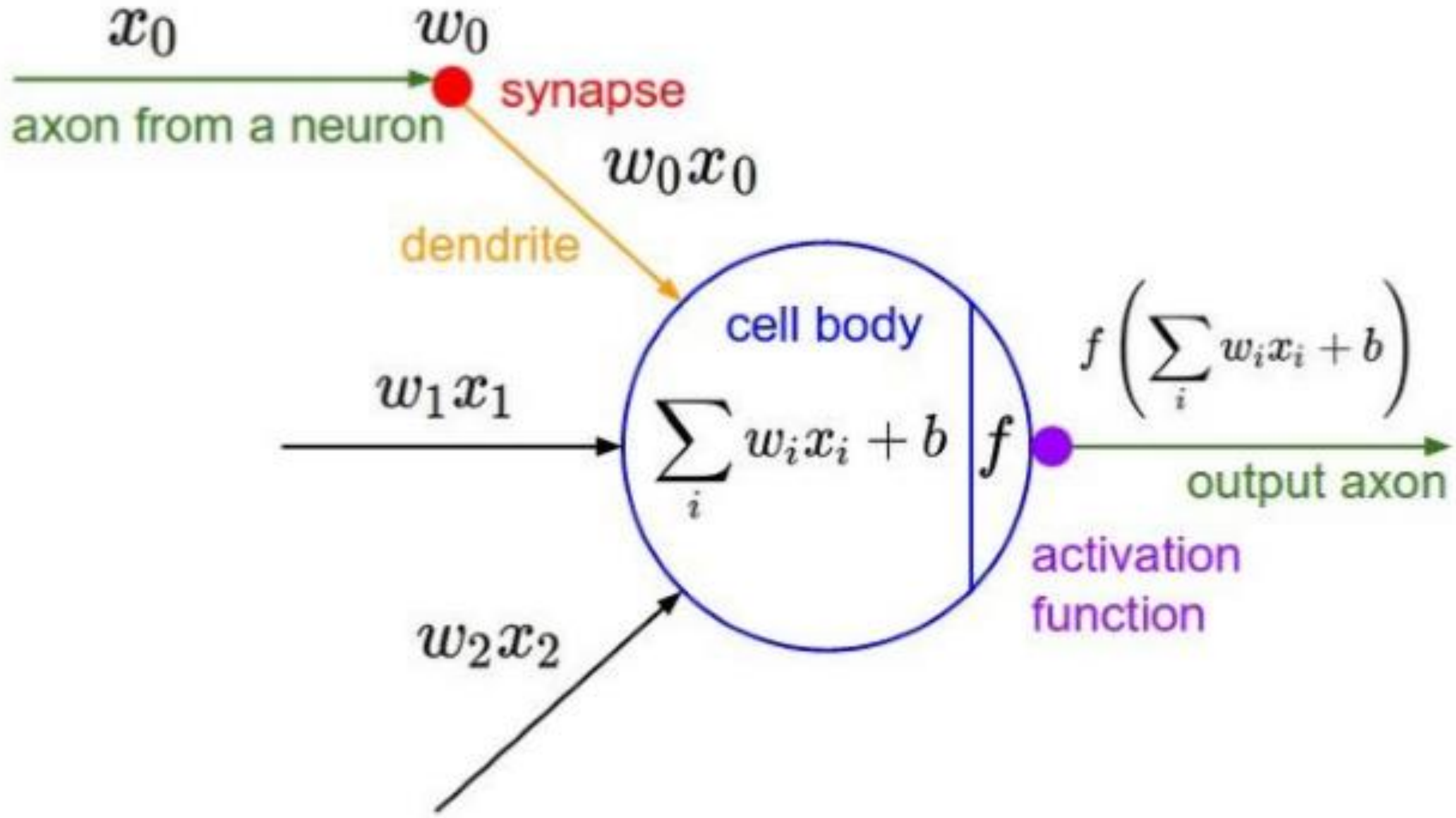
ReLu()를 사용하라!!
sigmoid 는 성능이 안좋다!!

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

N.N → Activation Function

Activation Functions

1. 특정치 이상의 값만 뽑기 위해서
2. 선형은 아무리 쌓아도 선형이기 때문에 비선형을 추가하여 다양성 부여



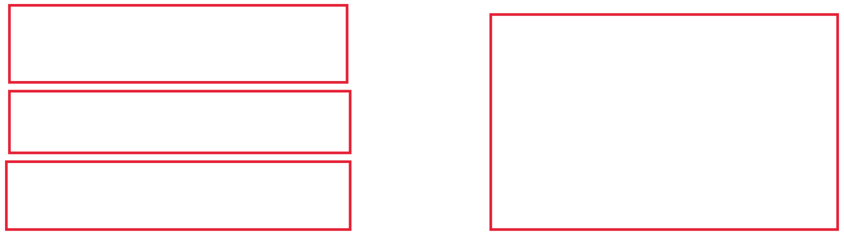
Activation Functions

- 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: $y_3 = w_3 y_2 + b_3$



선형으로만 구성되면 결국 두 가지는 같은 것

...

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

아무리 많은 linear를 쌓아도 결국에는 linear
→ Activation function으로 해결

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

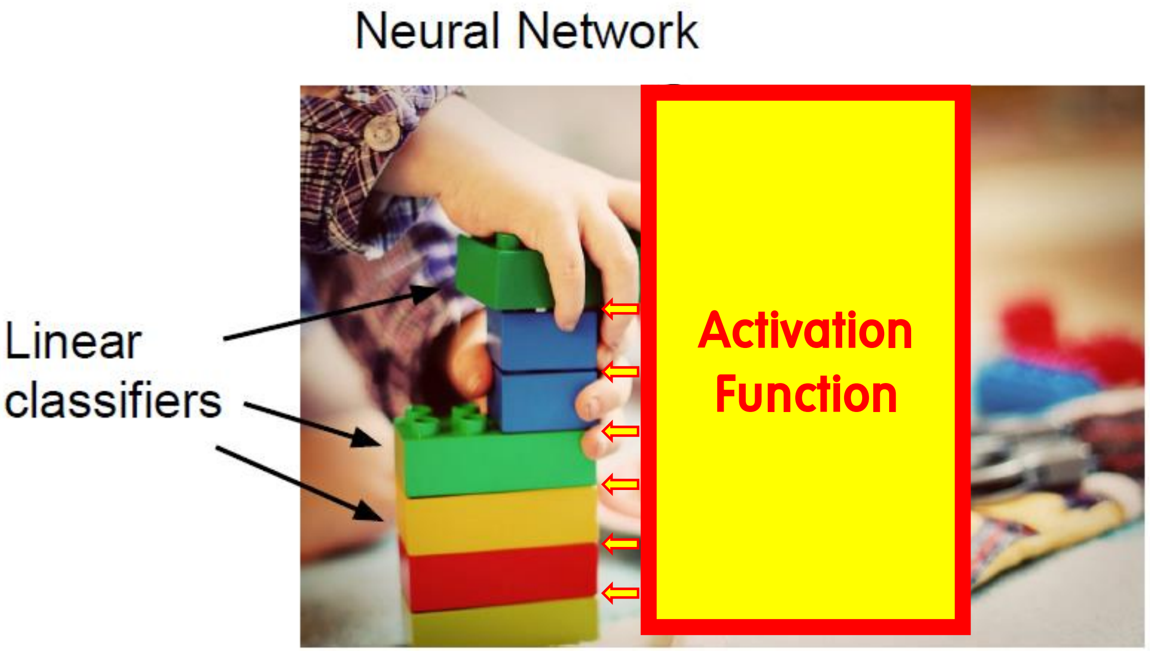
Layer 3: $\mathrm{y}_3 = \mathrm{w}_3\mathrm{y}_2 + \mathrm{b}_3 \quad = w_3(w_2(w_1x + b_1) + b_2) + b_3 \quad = c_3x + d_3$

...

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

Activation Functions

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



Activation Functions

Layer 1: $y_1 = w_1 x + b_1$

ReLu()

Act. $\widehat{y_1} = tanh(y_1) = \tanh(w_1 x + b_1)$

Layer 2: $y_2 = w_2 \widehat{y_1} + b_2 = w_2(\tanh(w_1 x + b_1)) + b_2 \neq c_2 x + d_2$

(Activation function: tanh)

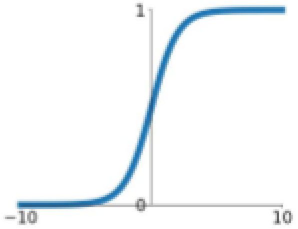
Activation Functions

비선형. 직선이 아니다 == 꺾은 직성

노란색의 경우 acc 가 90%~95%인 경우
97% 98% 으로 올려야 할때 다 써본다!!

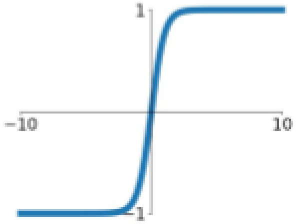
Sigmoid

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



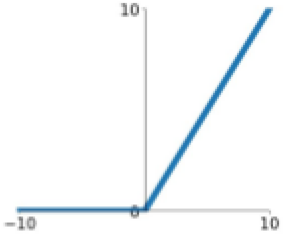
tanh

$\tanh(x)$



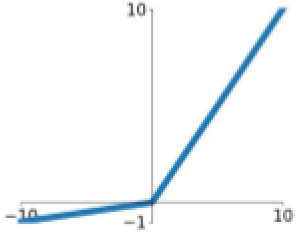
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

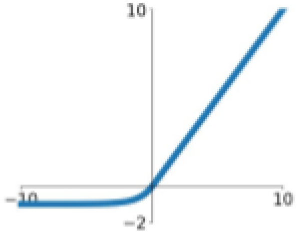


Maxout

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

ELU

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



Activation Functions

★★★
코드에서 sigmoid를 만나면 이건 activation으로 사용된 것이 아니라
분류 모델에서 binary 문제 해결에 있어서 사용되는데
예측 값이 하나일 때 softmax 함수 대신에 sigmoid 함수를 사용한다!!

- Summary

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

반드시 Sigmoid를 써야할 경우가 있다. (값 0~1)

Activation 함수
거의 ReLu() 사용.
But, Sigmoid()를 사용해야 하는 경우가 있다.

먼저,
X → ㅁ → ㅁ → 30
7
9

3개 중에 하나를 예측
softmax
내부로직 ① + ② e^30 / (e^30 + e^7 + e^9)
= 70%

X→ㅁ→ㅁ→3 하나만 예측값?
의미는?
binary문제!!
ex) 암예측 문제이면 - 10 이 나왔을 때 암이 아니다
3일 땐 암이다, 그러면 0일땐?? 애매하다!!
→ 예측값을 판단할 기준이 필요!! softmax 필요없다.
그러므로 양수, 음수로 기준을 두면 0도 양수이므로 암으로 판단

Data Preprocessing

→ weight 값이 곱해지는데 이 값 그대로 사용하면 값 변경이 크다!!
overfitting이 나타난다.
high frequency 해진다. noise 값에 취약해진다.

그러므로 /255 작업(scaling)을 통해 값을 0~1사이로 만들어 방지!!
transforms의 ToTensor()이 이 작업을 함.

- RGB Image range: 0~255
- Network input range: 0~1

(Recommandable)

In practice,

network_input = image_matrix / 255

Data Preprocessing

- Data augmentation

1~3단계 중 1단계에서 거의 모든 작업이 이뤄진다. 70~80%

Deep Learning

1. PreProcessing 2. Deep Learning 3. Post Processing

① Data Cleansing : 깨끗한 데이터를 위한 노가다 검수단계
(데이터 검수)
② 모든 값 Scaling /255
③ Data Augmentation → transforms.compose([...
(문제를 다양하게 → 성능↑)
④ 데이터 베율을 맞춰줌ㅁ
(데이터 비율 균일하게 - 각 label 갯수 같게)
⑤ 데이터 로딩할 때마다 섞어줌.

간트 차트 (일정 조율)

2.1 - - - - - - - - - - - - - - - - - - - 2.15 Preprocessing
2.10 - - - - - - - - - - - - - - - - 2.20 Deep Learning
2.18 - - - - - - - - - - - - - - - 2.28 Post Processing

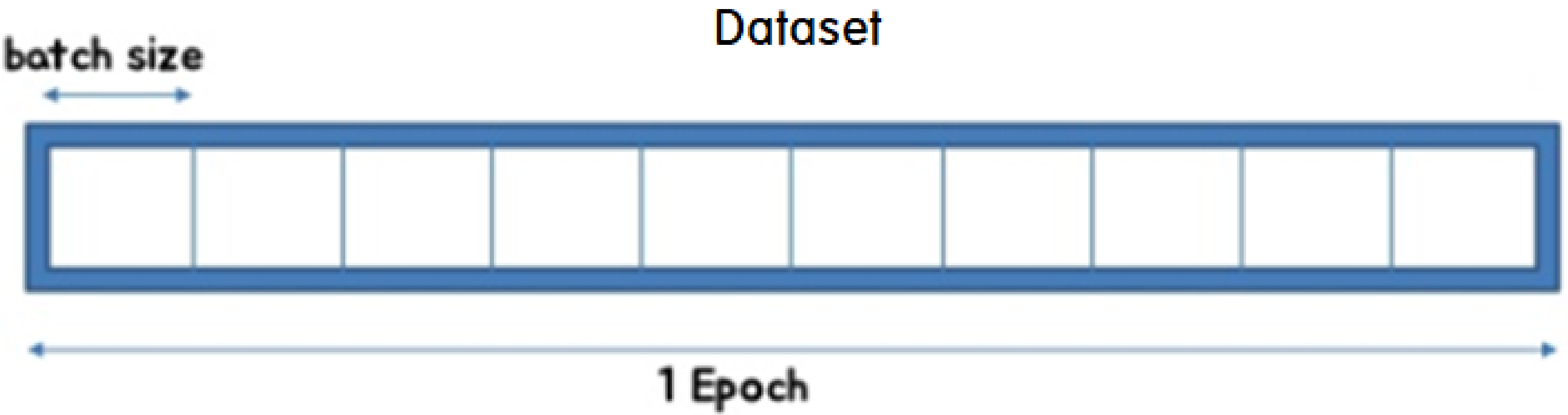
2021 02 01 3시 50분 부터 강의 다시 들어야할듯

Batch Normalization

ex) 7만장 : 6만장 | 1만장
batch size 100   1Epoch 600번
이미지 100개씩 로딩하니깐 각 100개마다 loss가 나온다.
첫번째 loss가 나온다는 것은 backward가 진행하고 다음 100개는 업데이트된 W로 학습하여 loss 추출. 이러한 작업을 600번 반복

- Batch (=mini batch)

Batch Normalization은 모든 layer마다 넣어야한다. 업계에서 별명이 슈퍼맨. 그 정도로 넣으면 성능이 좋아짐. sacling과는 차이가 있다.



흔히 Normalization은
① x를 평균만큼 빼고(중간값을 옮기고)
② 분산으로 나눈다(급변하는 데이터를 다루기 쉬운 안정적인 값. stable한 값. 으로 바꿔주는 것.)
Batch Normalization은 이러한 작업을 batch size 만큼 한다는 것.
네트워크 → 레어이 하나 → 값의 변경이 커짐 → stable → 레어이 하나 → 값의 변경이 커짐 → stable →...반복
값이 차분하지 않으면 학습이 잘 안된다.

batch normalization.png

Batch Normalization

모든 레이어마다 들어간다.
레이어마다 입력값의 variance가 커지는 것(Distribution)
→ internal covariate shift
→ Batch Normalization 이용

Data가 퍼져있다 (=분산↑)

Data가 임팩트 있게 모여있다 (=분산↓)
분산값이 낮을 수록 Data는 Dense하게 된다.
이렇게 하는 것 또한 Scaling이지만 normalization은 축도 같이 이동

- Impact of batch size on error

(x - mean)/ std
2, 100
이 둘 중 어느 값의 평균이 noise(stable)하냐
batch size는 높을 수록 stable함.
여기서는 16이상부터 안정적이므로 16이상부터
batch size를 설정하라.

그럼 batch 200이면 좋지 않을까? 좋다!!
그런데 현실적으론 computation 성능이 딸려서
불가능...돈 많으면 해라!!
그래서 최소 범위를 정해둔 것이고
100을 추천하는 것

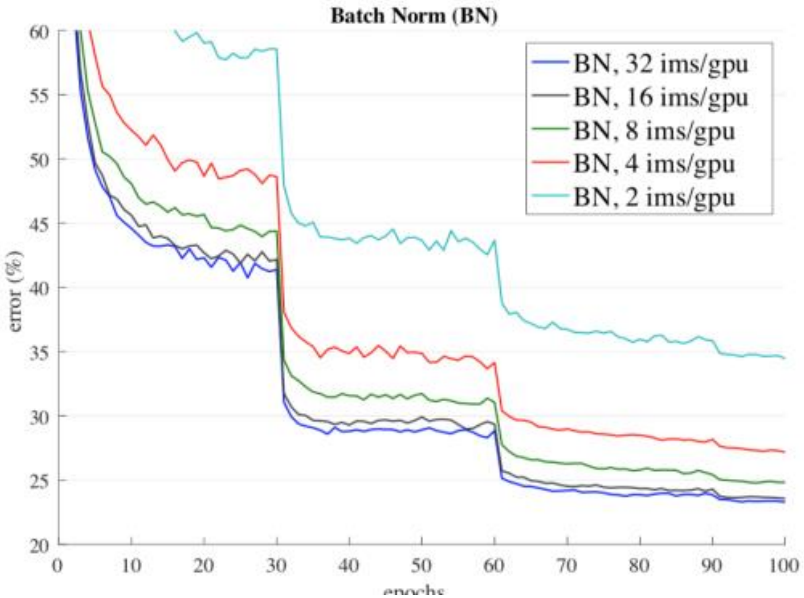
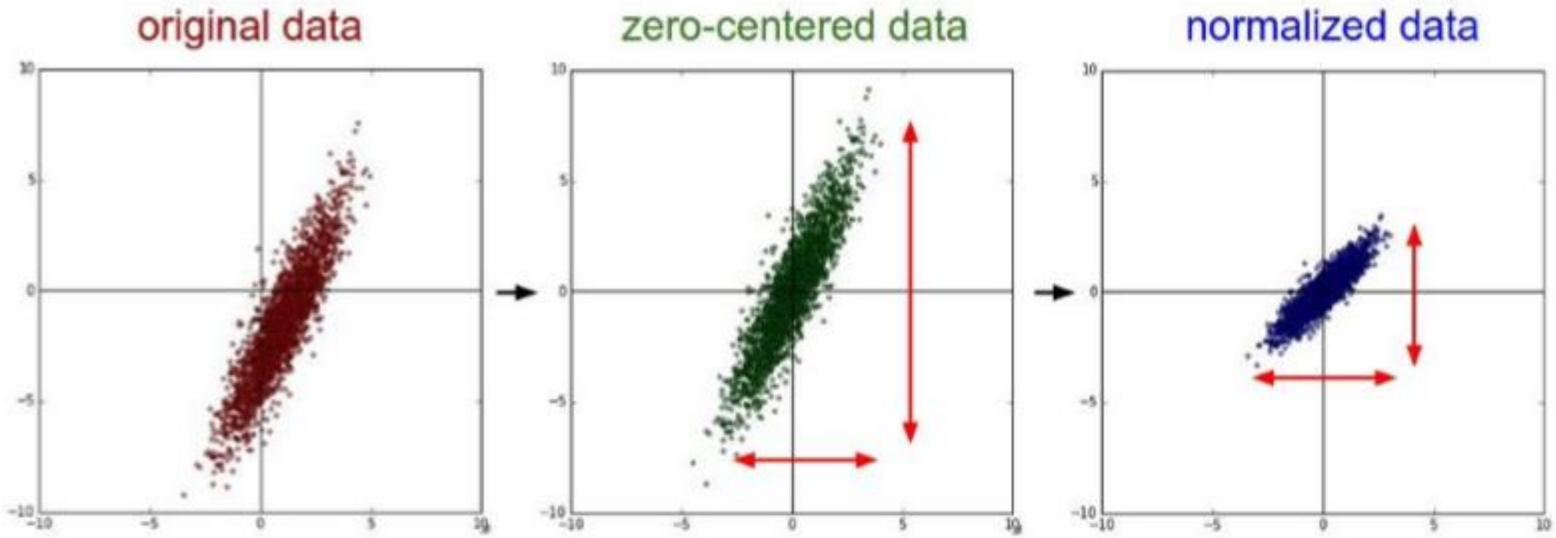


Figure 5. **Sensitivity to batch sizes**: ResNet-50's validation error of BN (left) and GN (right) trained with 32, 16, 8, 4, and 2 images/GPU

Batch size를 얼마로 하는가에 따른 성능 측정 모델

Babysitting the Learning Process

- Step1: Preprocessing the data (+ data augmentation)

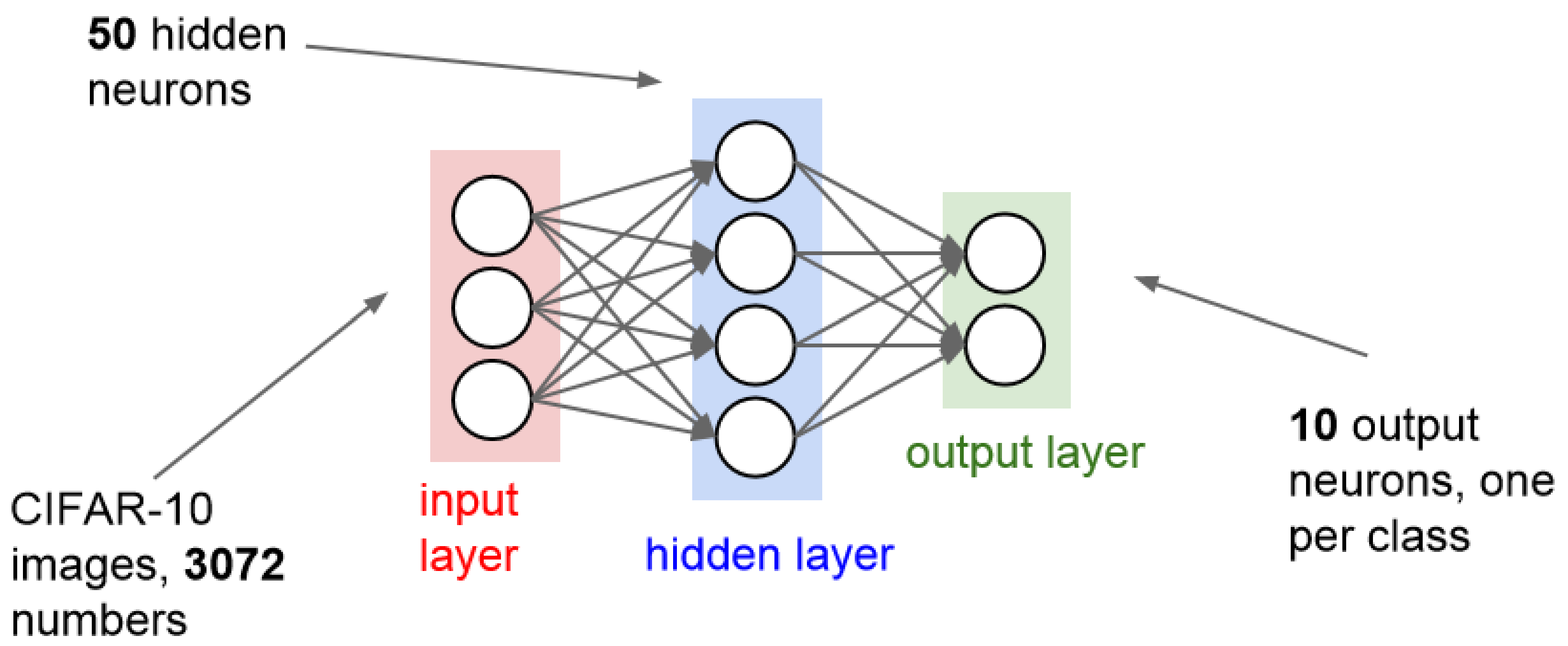



```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

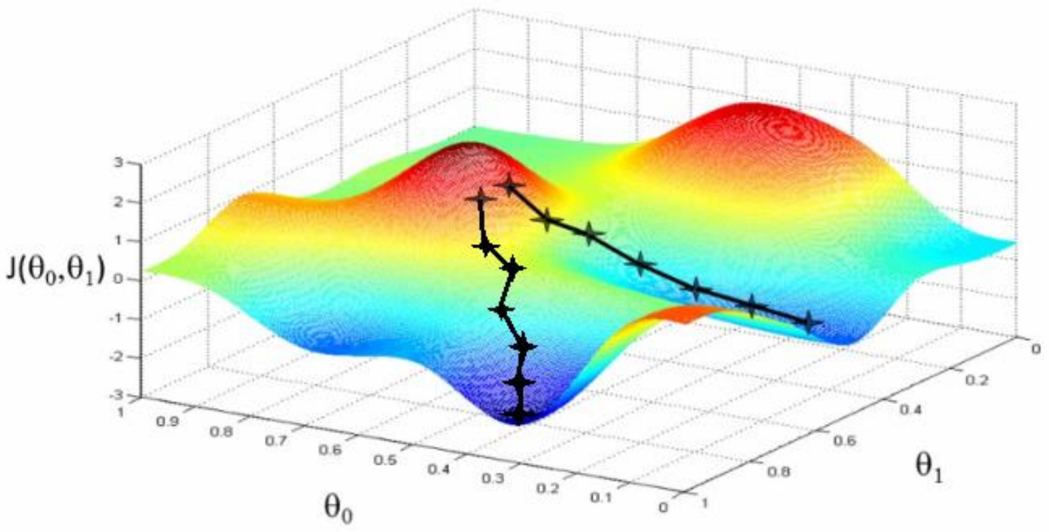
Babysitting the Learning Process

- Step2: Design the network architecture



Babysitting the Learning Process

- Step3: Design loss function / Optimizer (lr)

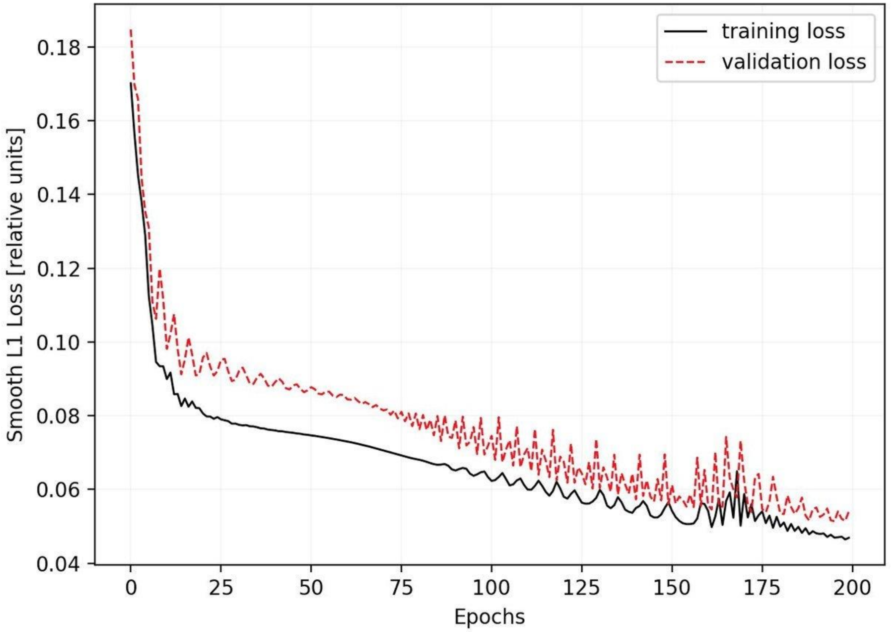


Babysitting the Learning Process

- Step4: Train model and analyze results

Visualize train & validation loss

이런 경우 시간을 둬 가면서 loss가 어떻게
나오는지 끝!!까지 봐야하는 경우



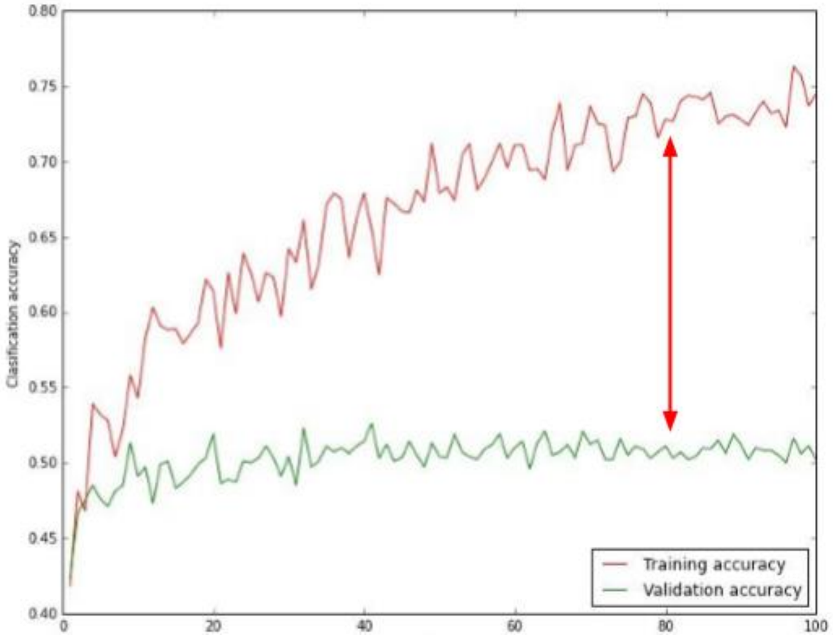

```
Finished epoch 1 / 200: cost 2.302603, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 2 / 200: cost 2.302258, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 3 / 200: cost 2.301849, train: 0.600000, val 0.600000, lr 1.000000e-03
```

Babysitting the Learning Process

- Step4: Train model and analyze results

이런 경우 대표적으로 건들 수 있는 값이
Learning Rate

Check overfitting

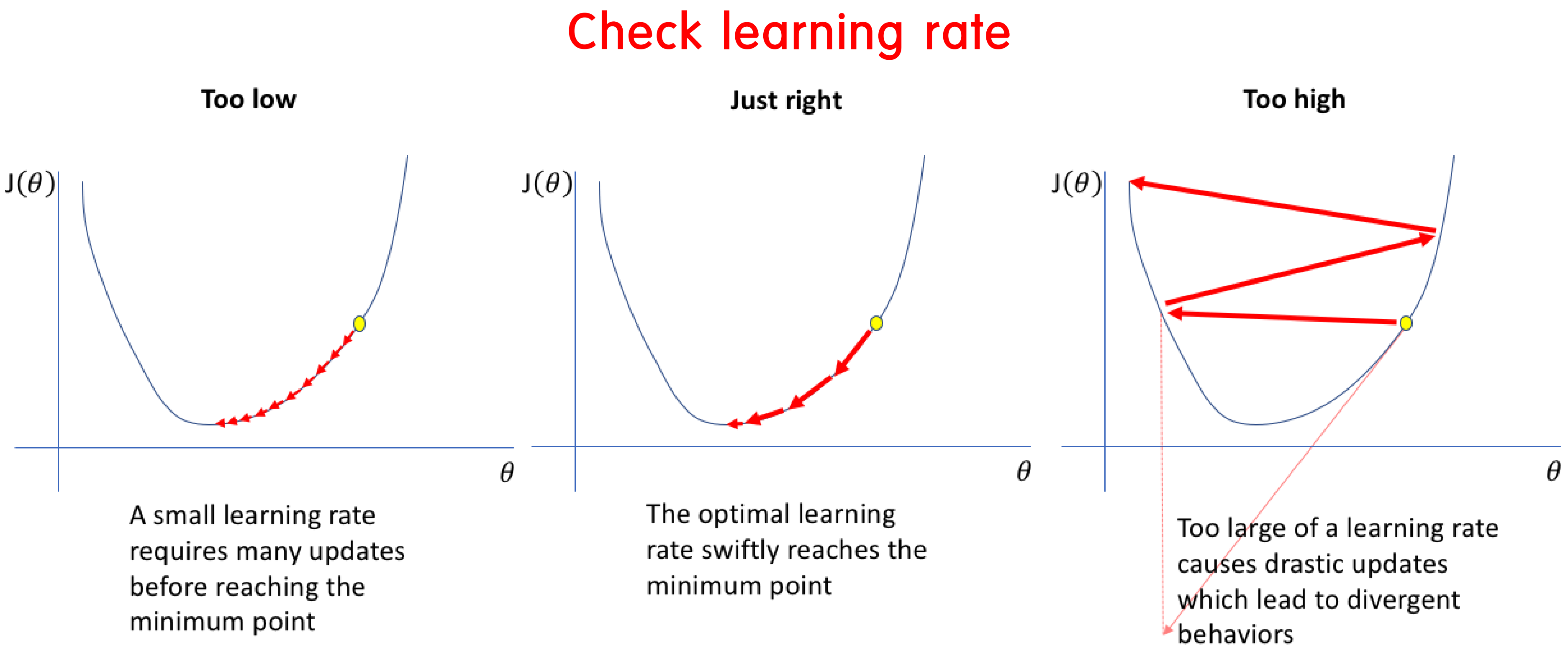


big gap = overfitting
=> increase regularization strength?

no gap
=> increase model capacity?

Babysitting the Learning Process

- Step4: Train model and analyze results



Babysitting the Learning Process

- Step4: Train model and analyze results

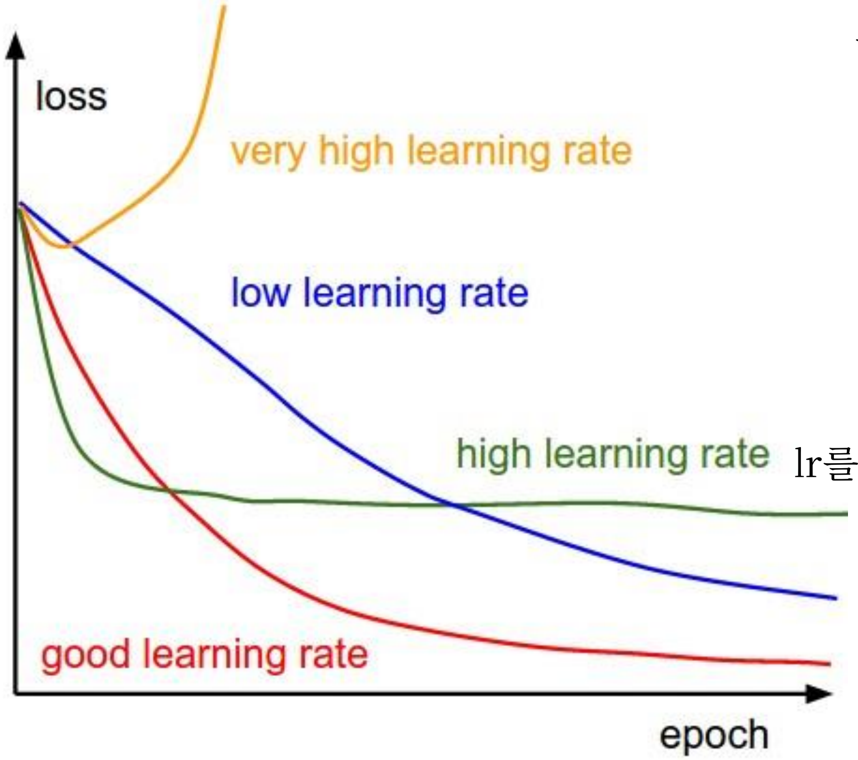
Epoch이 지날수록 Loss가 줄어드는 파형

경험이 중요하다는 것은 수억개의 데이터를 교차
검증하는 것은 불가능.
그래서 경험에 의해 어떤 값이면 acc가 높겠다 이
런 감이 생김. 딱 집어서 돌려야 하거든...

가이드가 따로 X
문제에 따라서 다양함.
경험이 중요...

딥러닝 개발자

Check learning rate



lr를 조오금 줄인다

Hyperparameter Optimization

Choices about the algorithm that we set rather than learn
(≈ Heuristic Values)

ex)

- Initial learning rate 추천하는 값 있음
- learning rate decay
- batch size 16↑ 32↑
- epoch 기본 100
- number of layer
- convolutional kernel size 3x3 대세
- pooling type
- activation function
- upsampling method, max pooling의 과거 이름
- optimizer Adam이 성능이 좋음.
- data augmentation parameters
- over-sampling ratio
- k-fold cross validation
- etc...

추천하는 값이 있다는 것은 GPU 성능이 부족할 뿐더러 그 안에서 조오금씩 바꿔가며 Test

Summary (Training neural network)

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

- Babysitting the Learning process
- Hyperparameter optimization