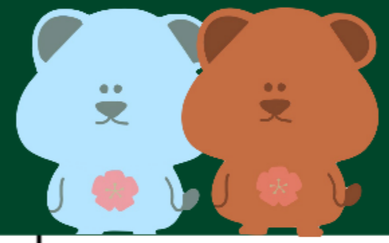


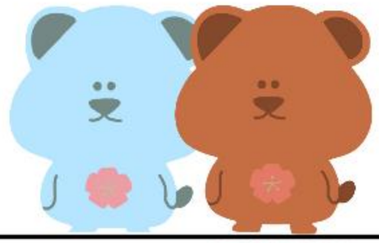
# Training Neural Networks, Part 1

최예은, 최지우

EWHA EURON

# Index





EWHA EURON

# review

EWHA EURON

# #01 CNN

## Convolutional Neural Networks

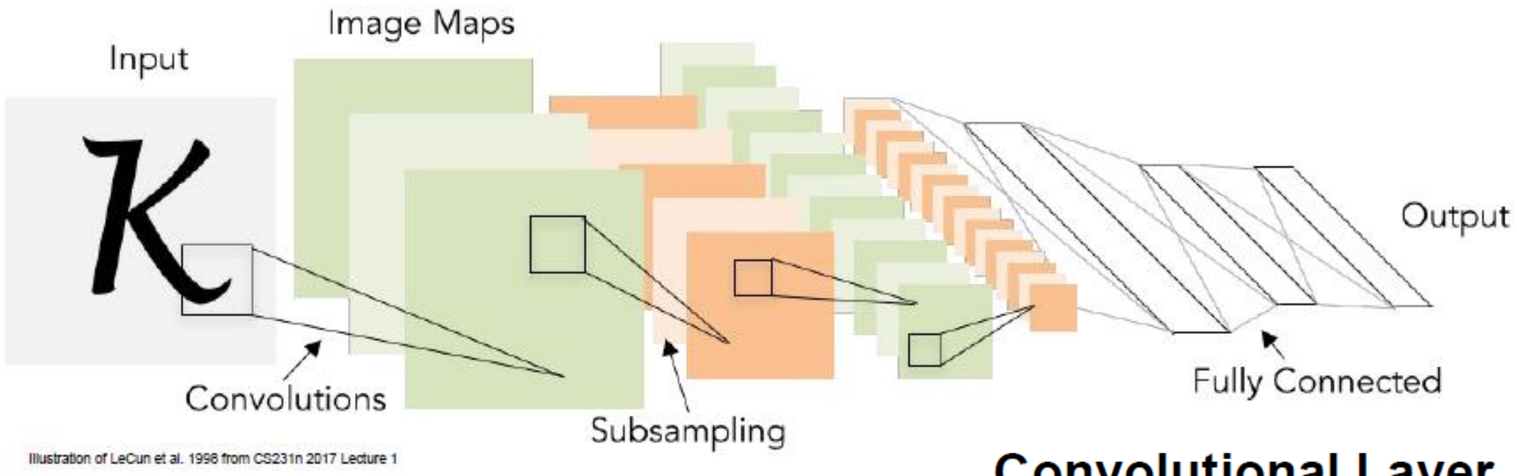
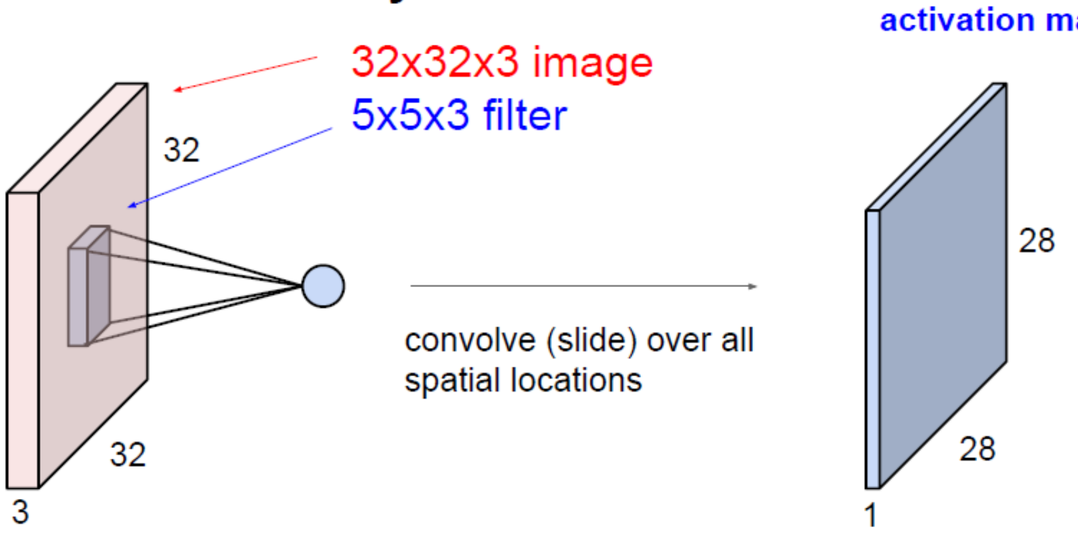


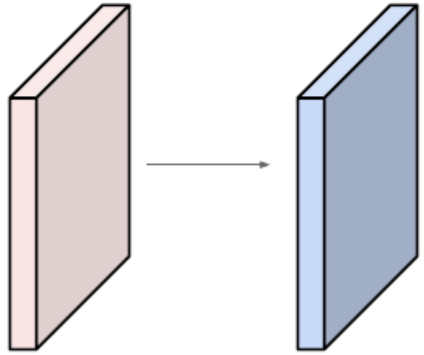

Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

## Convolutional Layer



# #01 CNN

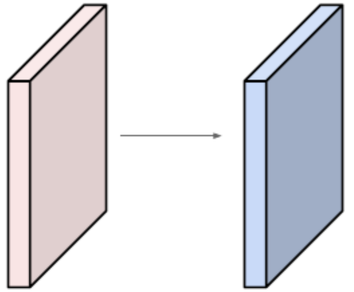
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
**32x32x10**

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

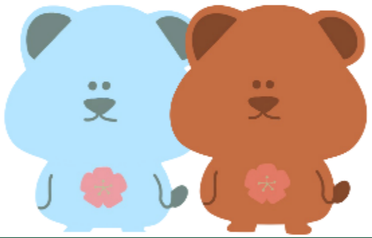
Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

# #01 CNN

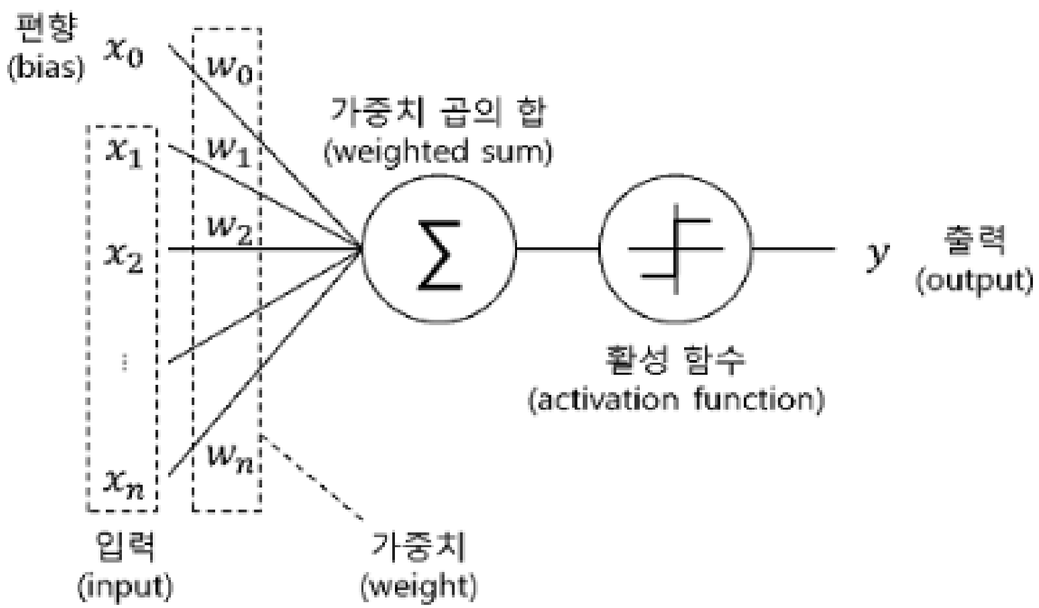
**Summary**. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$,
  - their spatial extent $F$,
  - the stride $S$,
  - the amount of zero padding $P$.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and $K$ biases.
- In the output volume, the $d$-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the $d$-th filter over the input volume with a stride of $S$, and then offset by $d$-th bias.

# Activation Functions



# #01 Activation Functions




신경망의 학습 절차

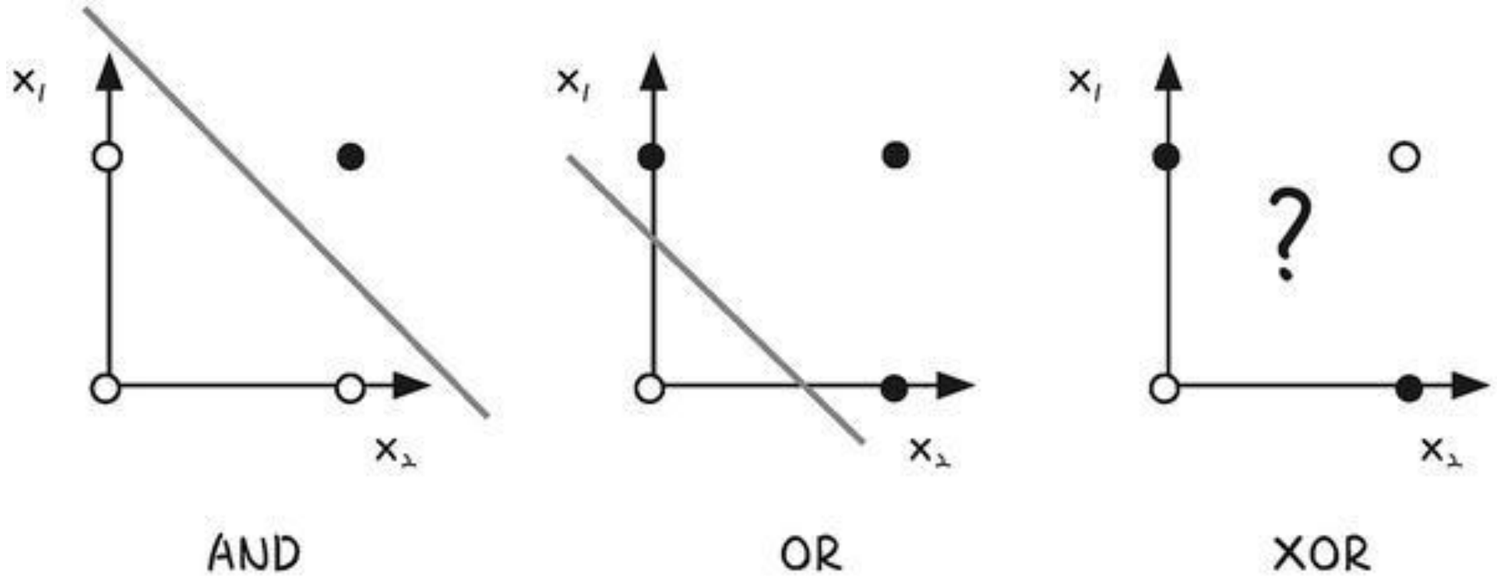
활성화 함수 :

노드의 입력값이 임계치를 넘어가면 활성화가 되고 넘지 않으면 비활성화하게끔 되어있다.

(스위치 같은 개념!)

# #01 Activation Functions

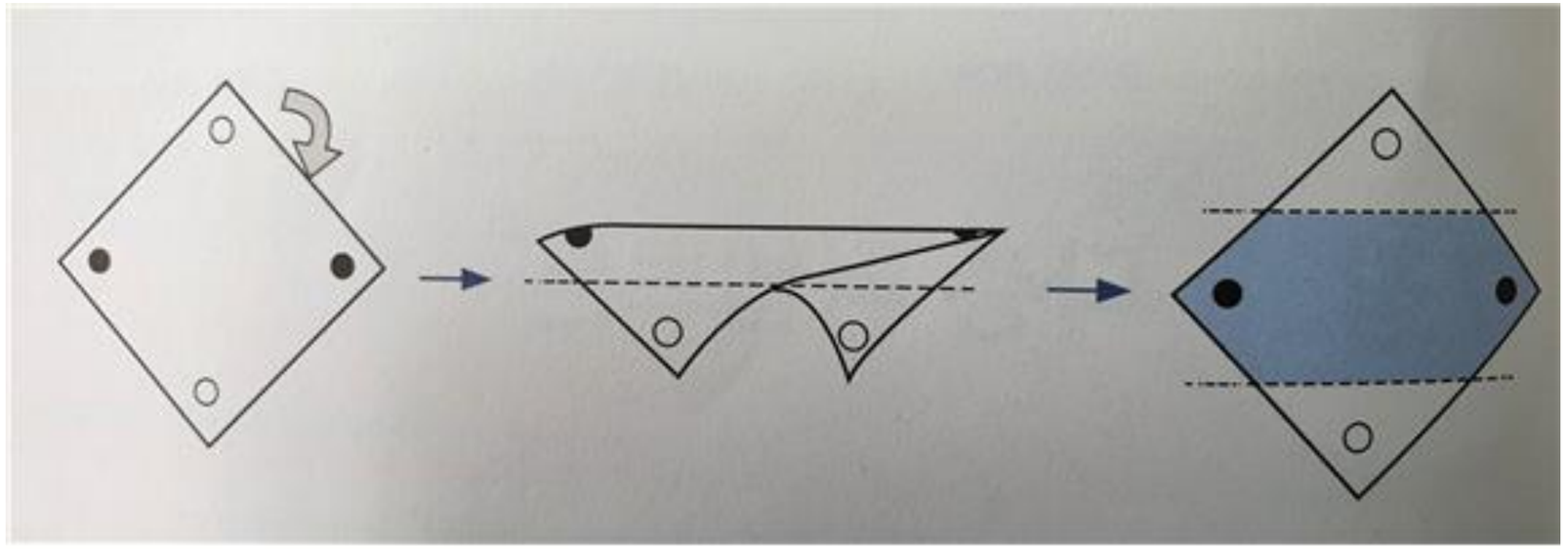
Q. 활성화 함수를 왜 사용하는가?




퍼셉트론의 한계

# #01 Activation Functions

Q. 활성화 함수를 왜 사용하는가?



다층 퍼셉트론

**즉, 다층 퍼셉트론에서 모델의 표현력을 증가시키기 위해 활성화 함수가 필요하다.**

# #01 Activation Functions

## Q. 활성화 함수를 왜 사용하는가?

**활성화 함수는 선형 함수를 비선형 함수로 출력하고 신호를 전달하는 역할을 함.**

**예를 들어, 활성화 함수를 h(x) = cx 선형 함수 라고 하자.**

**3층으로 구성된 신경망을 만들고 싶을 때, h(h(h(x))) = c • c • c • x 이다.**

**$C^3$ • x 인데 초기의 h(x) 에 a = $c^3$ 넣은 것과 같고 선형 함수 이다.**

**→ 선형 함수의 퍼셉트론 즉, 1층 구조로 Layers 쌓은 것과 같다.**

# #01 Activation Functions

## Q. 활성화 함수를 왜 사용하는가?

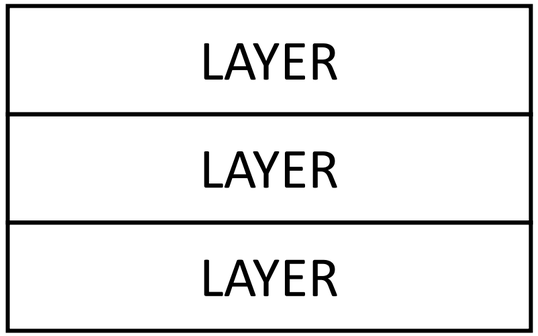
**활성화 함수는 선형 함수를 비선형 함수로 출력하고 신호를 전달하는 역할을 함.**

**예를 들어, 활성화 함수를 h(x) = cx 선형 함수 라고 하자.**

**3층으로 구성된 신경망을 만들고 싶을 때, h(h(h(x))) = c • c • c • x 이다.**

**$C^3$ • x 인데 초기의 h(x) 에 a = $c^3$ 넣은 것과 같고 선형 함수 이다.**

**→ 선형 함수의 퍼셉트론 즉, 1층 구조로 Layers 쌓은 것과 같다.**



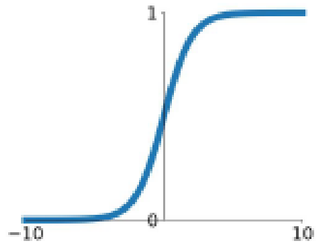

**다층 퍼셉트론**


**단층 퍼셉트론**

# #01 Activation Functions

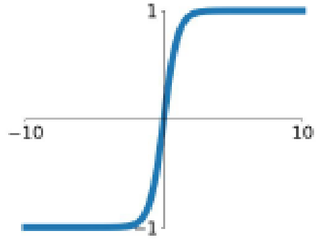
**Sigmoid**

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



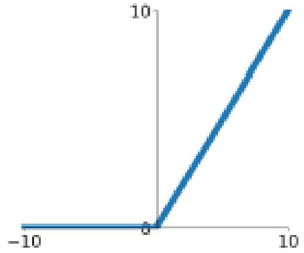

**tanh**

$$\tanh(x)$$



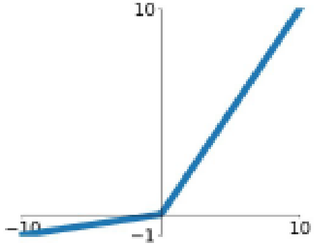

**ReLU**

$$\max(0, x)$$




**Leaky ReLU**

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

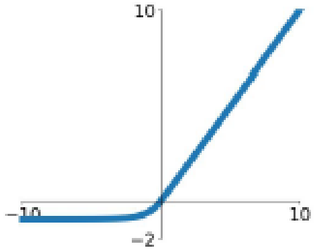



**Maxout**

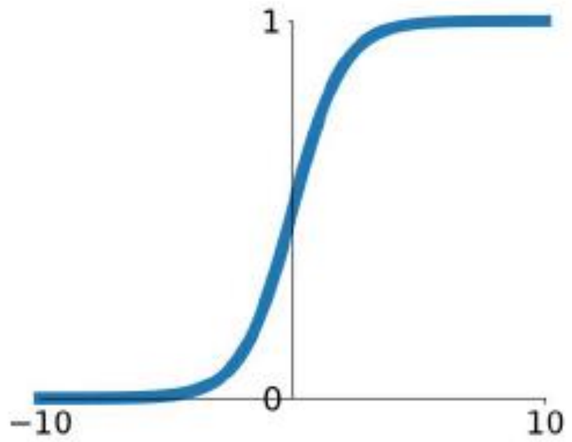
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

**ELU**

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



# #01 Activation Functions




**Sigmoid**

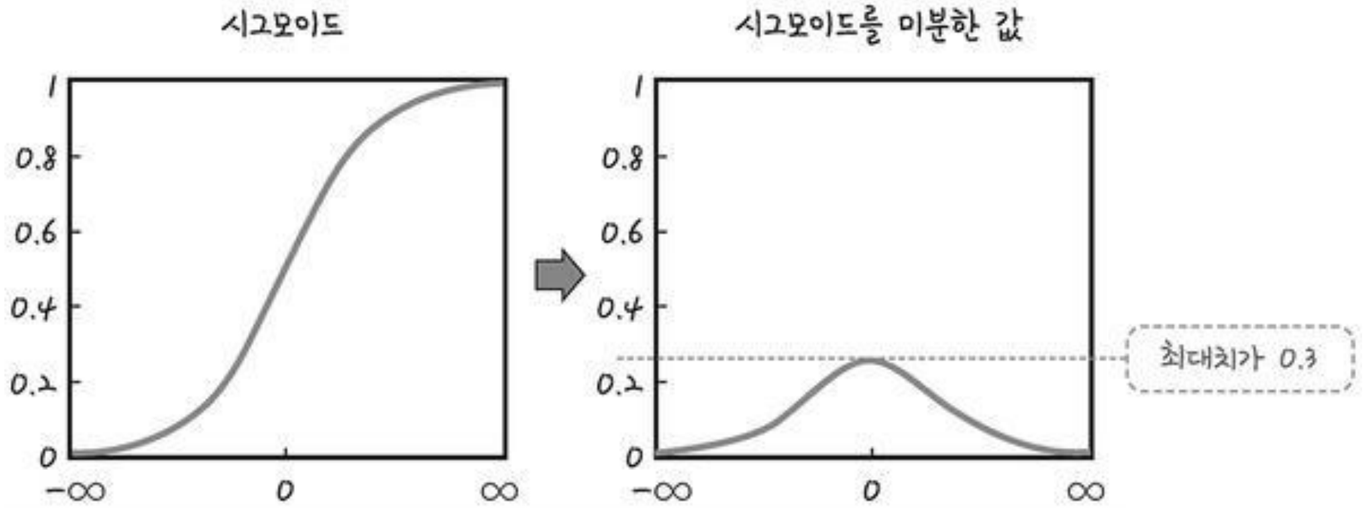
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive

# #01 Activation Functions

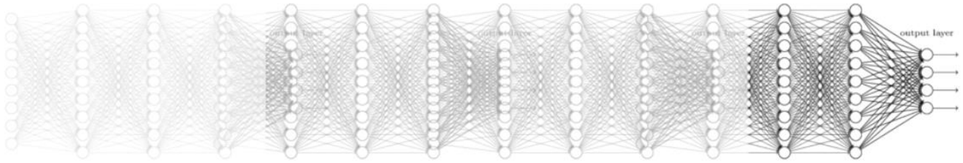
## 시그모이드 함수의 문제점. (Vanishing gradient)




Vanishing gradient (NN winter2: 1986-2006)

→ **시그모이드의 미분값의 최대치가 0.3 이다. 1보다 작으므로 역전파를 진행할 때 값이 0에 가까워지고 가중치를 수정하기가 어려워짐.**

**(Vanishing gradient 발생)**

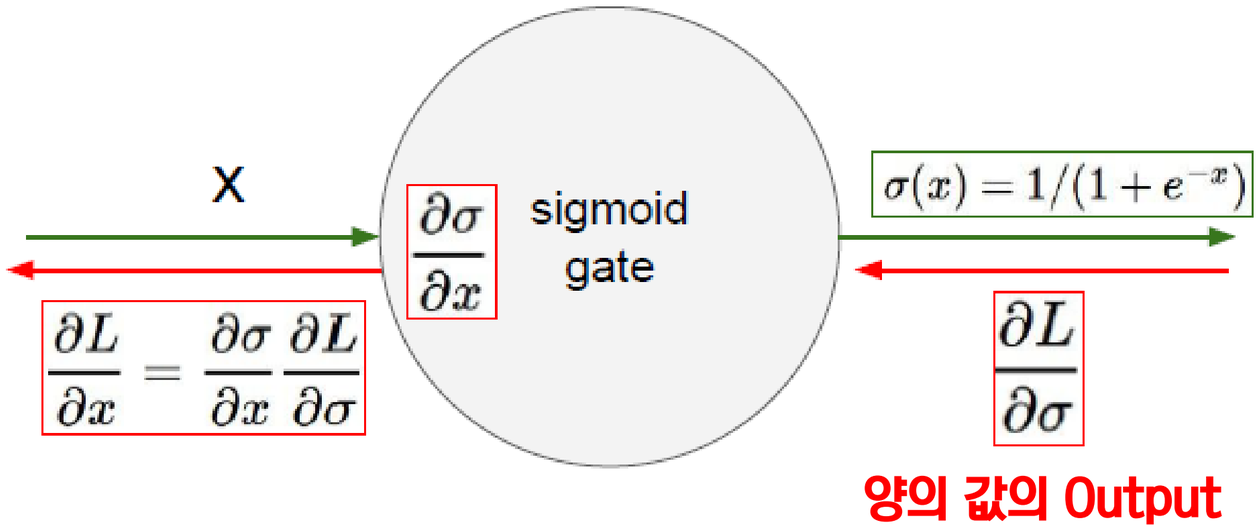



EWHA EURON

# #01 Activation Functions

## 시그모이드 함수의 문제점. (Not zero-centered)

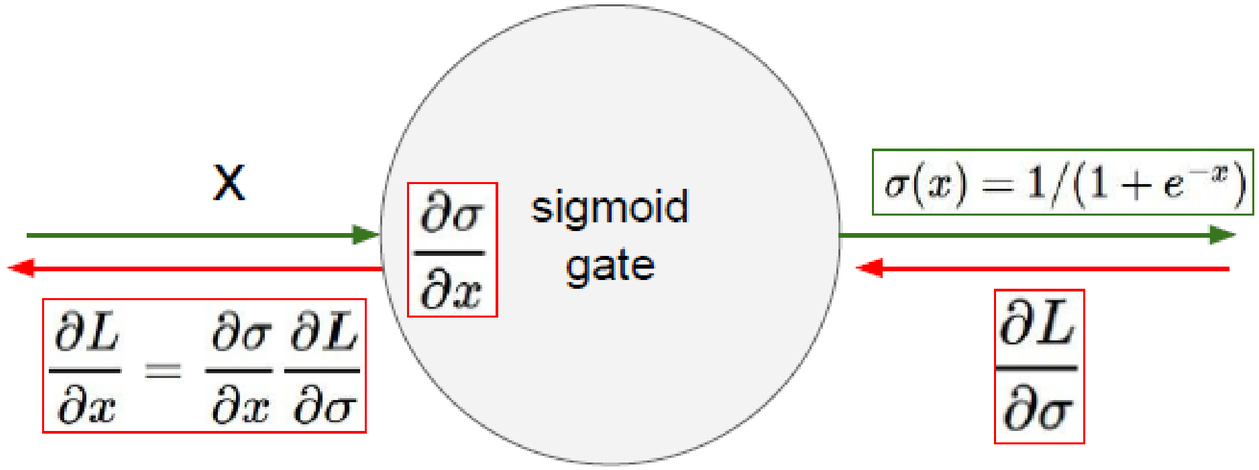
**Sigmoid 는 [0,1] 범위의 output 을 출력한다.**



# #01 Activation Functions

## 시그모이드 함수의 문제점. (Not zero-centered)

**Sigmoid 는 [0,1] 범위의 output 을 출력한다.**
**역전파를 진행할 때, W의 gradient 는 모두 양수 또는 음수의 값을 갖게 된다.**

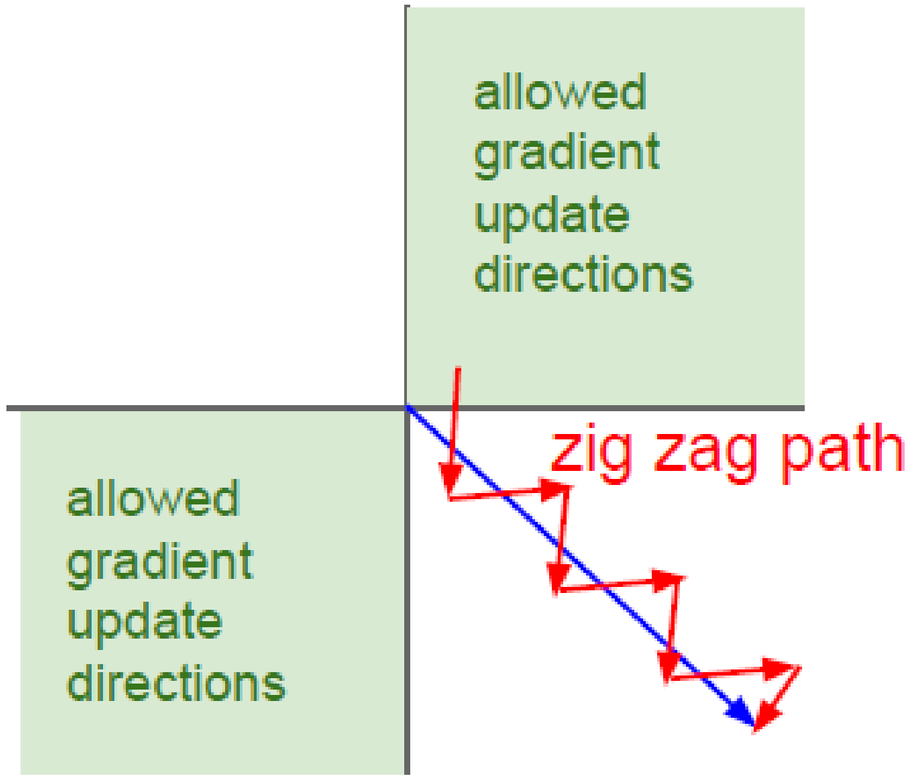



W의 grad 는 da/dx 에 의존 (upstream grad)

양의 값의 Output

# #01 Activation Functions

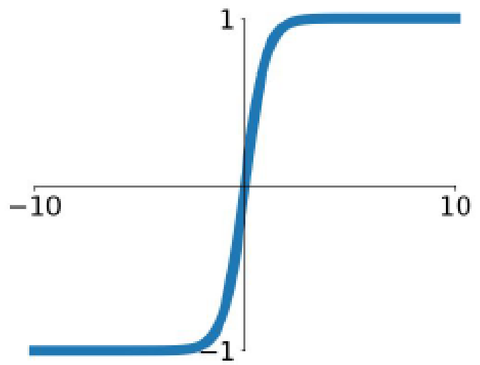
시그모이드 함수의 문제점. (Not zero-centered)




**모두 양수 또는 음수이므로 zig zag 방향으로 가기 때문에 수렴이 매우 느려진다.**

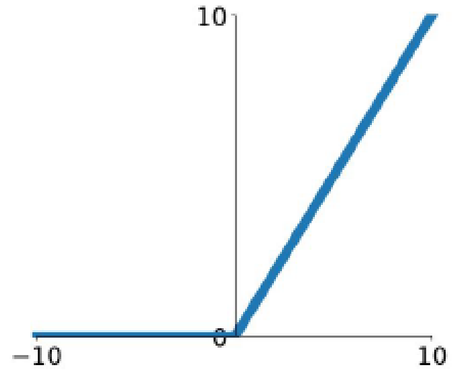
**Not zero centered 는 수렴이 느릴 수 밖에 없다.**

# #01 Activation Functions




**tanh(x)**

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(



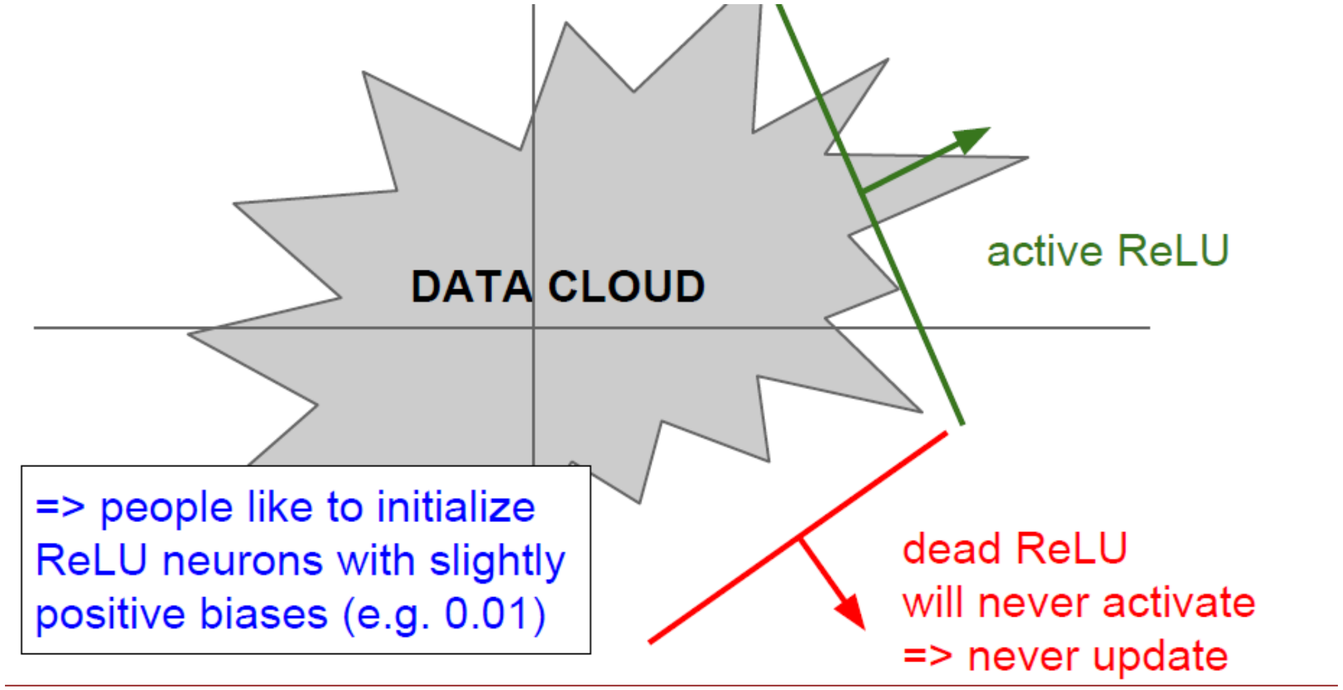

**ReLU**
(Rectified Linear Unit)

- 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

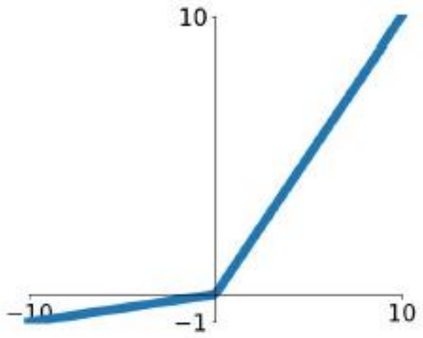
# #01 Activation Functions

## ReLU 함수의 문제점.

1. Not zero-centered
2. Dead ReLU



# #01 Activation Functions

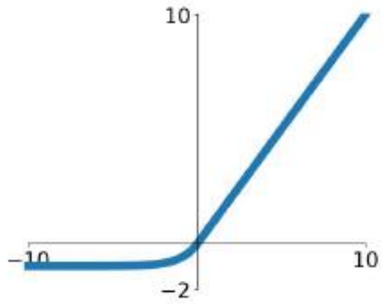



**Leaky ReLU**

$$f(x) = \max(0.01x, x)$$

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

**Exponential Linear Units (ELU)**




$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

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

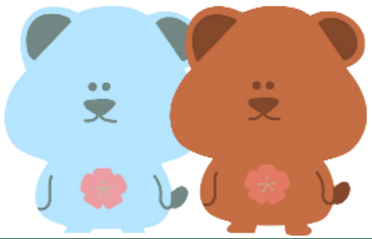
- Computation requires exp()

# #01 Activation Functions

**TLDR: In practice:**

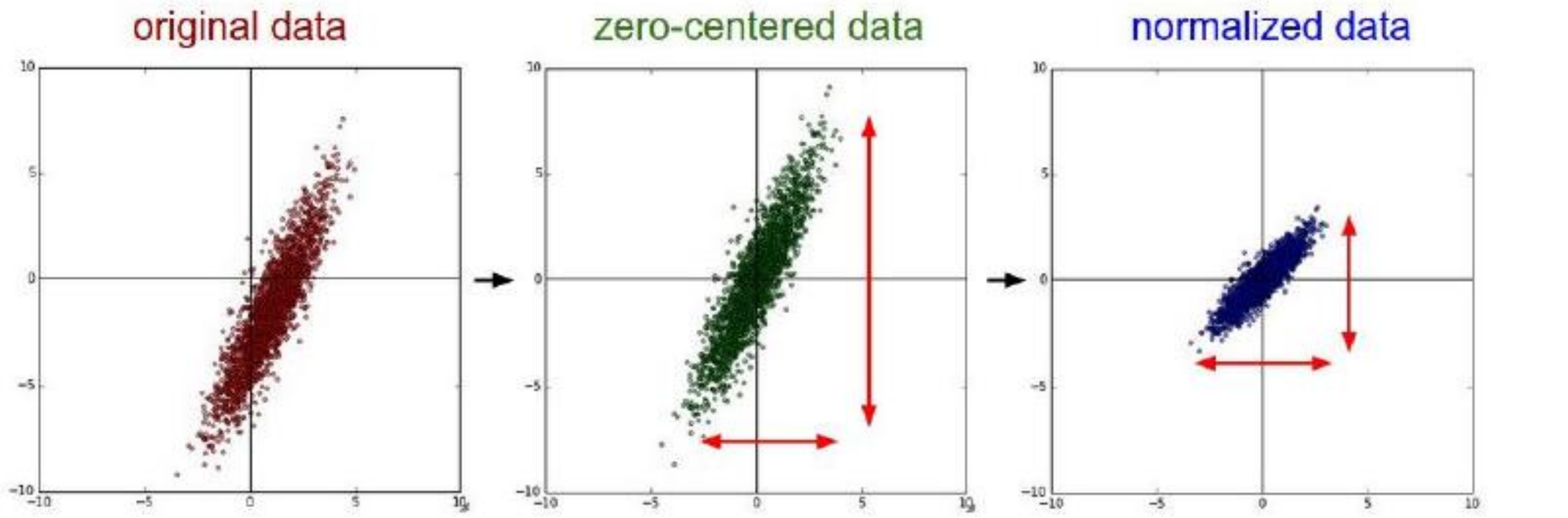
- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh but don’t expect much
- Don’t use sigmoid

# Data preprocessing, Weight Initialization



EWHA EURON

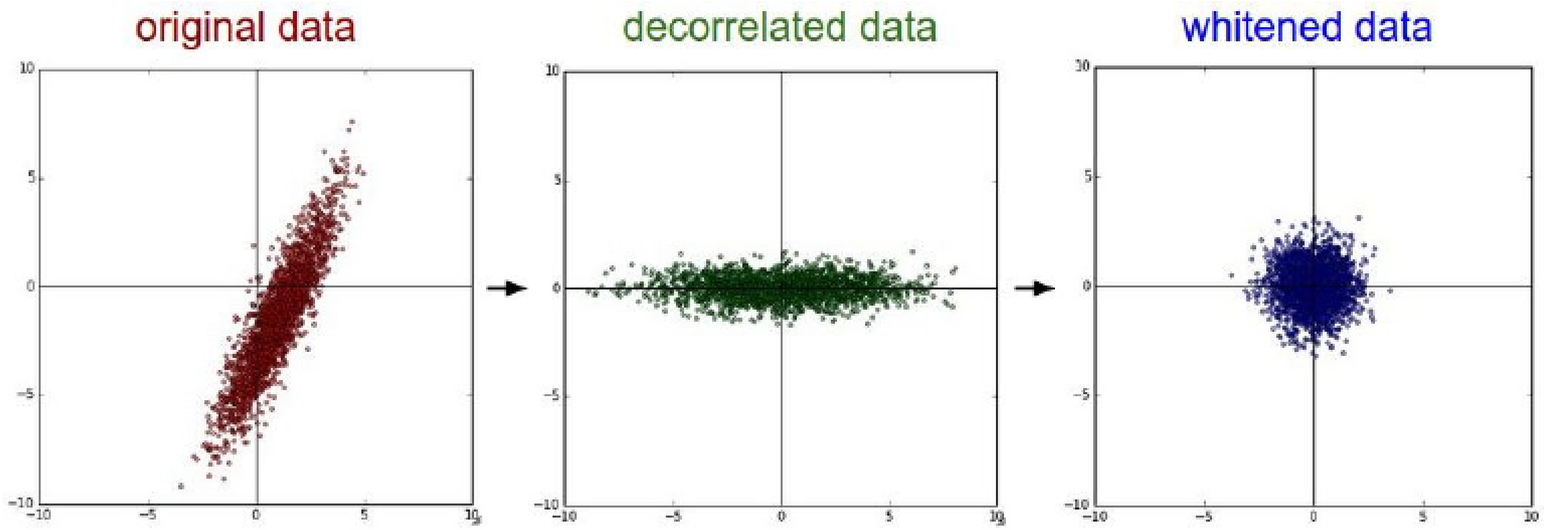
# #02 Data preprocessing



# #02 Data preprocessing

## Step 1: Preprocess the data

In practice, you may also see **PCA** and **Whitening** of the data



# #02 Data preprocessing

**TLDR: In practice for Images:** center only

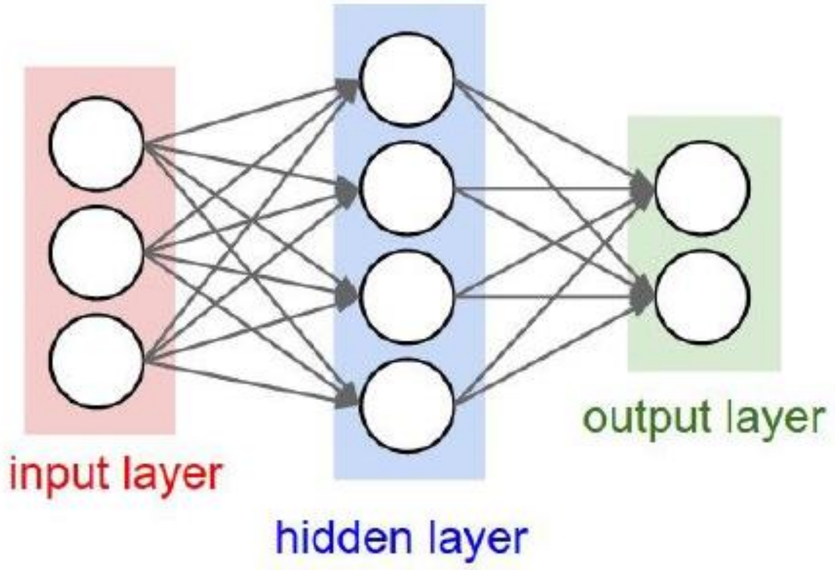
e.g. consider CIFAR-10 example with [32,32,3] images

- Subtract the mean image (e.g. AlexNet)
  (mean image = [32,32,3] array)
- Subtract per-channel mean (e.g. VGGNet)
  (mean along each channel = 3 numbers)

Not common to normalize variance, to do PCA or whitening

# #03 Weight Initialization

- Q: what happens when W=0 init is used?



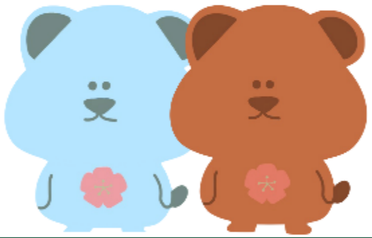

**가중치가 0이라서 모든 뉴런은 모두 다 같은 연산을 할 것이고,**
**gradient 가 동일하게 될 것이다.**

# #03 Weight Initialization

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

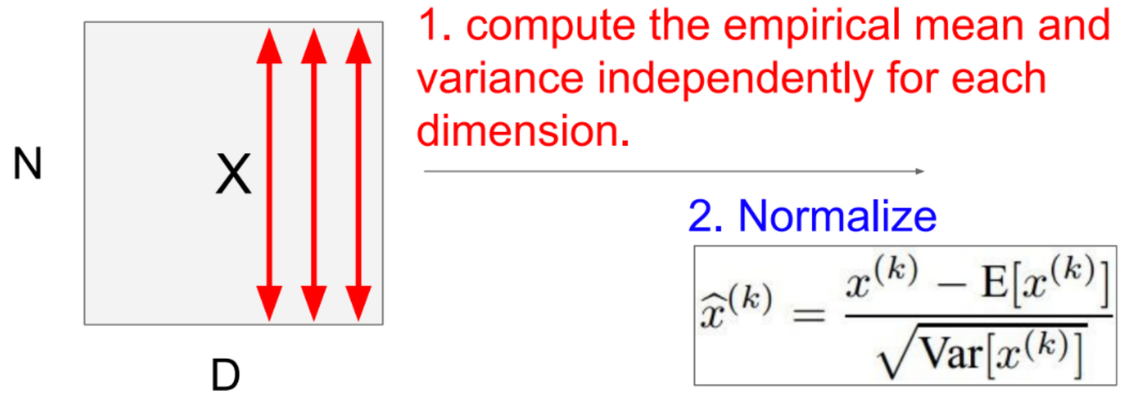
"Xavier initialization"
[Glorot et al., 2010]

# Batch Normalization



# #Batch Normalization

- Batch 별로 평균과 분산을 각각 구해 정규화
- 레이어의 입력이 unit gaussiam이 되도록 강제하는 것 -> saturation방지
- Gradient Vanishing 이 일어나지 않도록 하는 방법 중 하나
  - Activation function의 변화, Careful Initialization, Small learning rate 등으로 해결했지만, 이런 간접적인 방법보다 training하는 과정 자체를 안정화해서 학습 속도를 가속시킬 근본적인 방법임



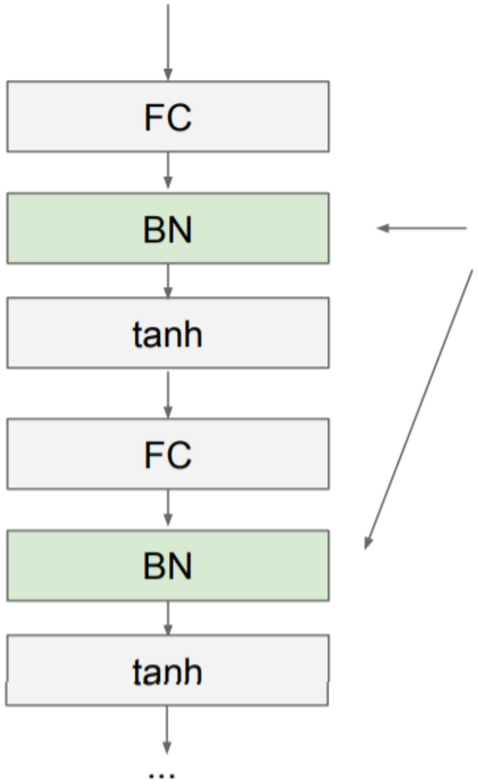

1) N: Batch 당 N개의 학습 데이터
   D: 데이터의 차원

2) 각 차원별(feature element 별로) 평균을 각각 구함

3) Batch 내에 이걸 전부 계산해 normalize

# #Batch Normalization

- 연산은 FC나 Conv layer 직후에 넣어줌
- 깊은 네트워크에서 각 레이어의 w가 지속적으로 곱해져 발생한 bad scaling effect를 normalization은 상쇄시킴




Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

# #Batch Normalization

Normalize:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

- BN은 입력값을 not saturated 한 영역에 있게 만든다.
- 하지만, 무조건 saturation을 막기 보단, saturation의 조절을 학습할 수 있다면 더 효율적인 결과를 얻음
  - ✓ 감마는 scaling, 베타는 shift의 효과
  - ✓ 감마값과 베타값을 학습을 통해 찾음
- 감마값에 분산값을, 베타값에 평균값을 넣으면 unit gaussian 이전의 원래 상태에 비슷하게 복구 가능

# #Batch Normalization

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma$, $\beta$
**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

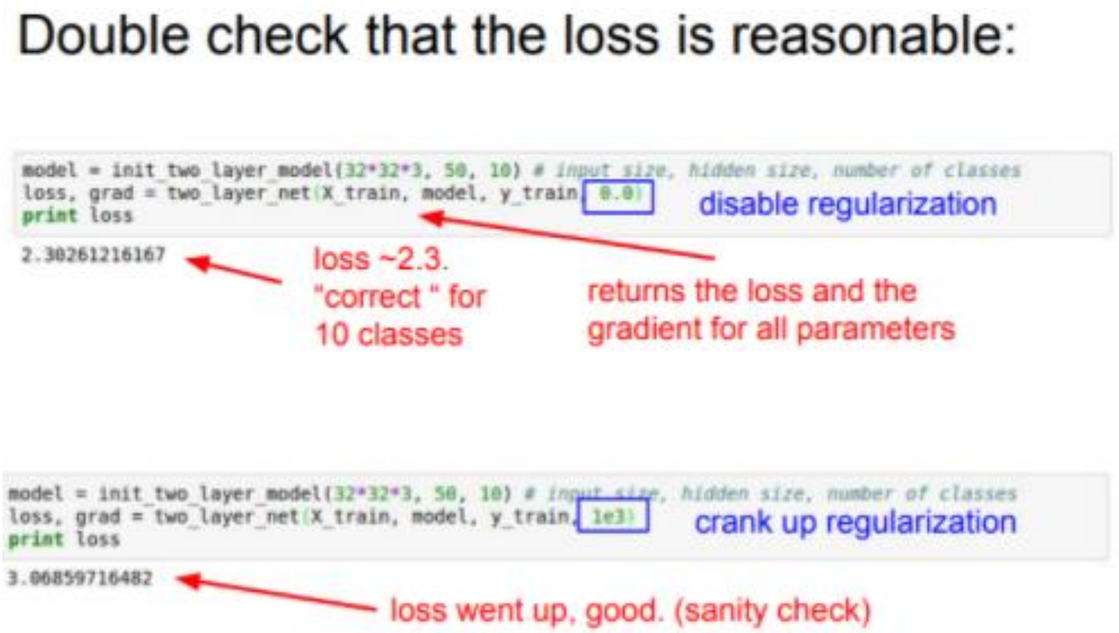
- BN 사용시, initialization에 대한 의존도가 낮아진다.
  - ✓ 다양한 초기화 시도 가능
  - ✓ 학습률 더 높일 수 있음
- Regularization의 역할도 수행한다.
  - ✓ 배치마다 평균, 분산 구하고 normalize
  - ✓ 각 레이어는 배치 안의 데이터에 영향을 받는다.
- Batch마다 입력값이 다르기에 랜덤한 값을 꺼내주는 dropout 필요성 감소됨

# Babysitting the Learning process

# #Babysitting the Learning process

<트레이닝을 모니터링 하는 방법>

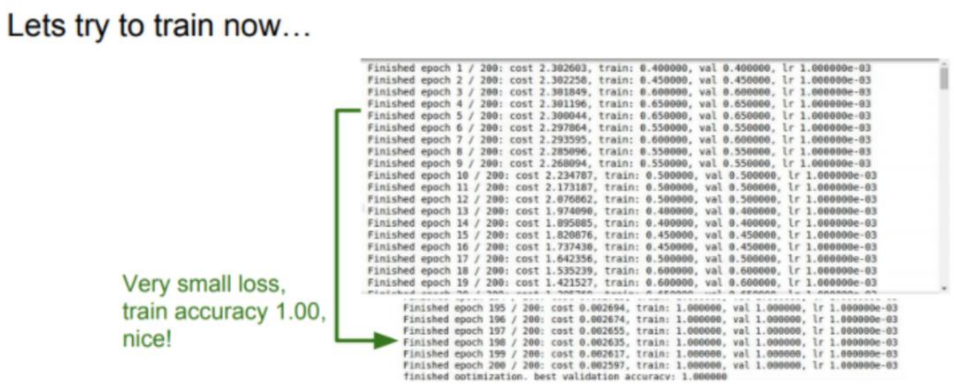
1. 데이터 전처리
2. 히든 레이어나 개수나 뉴런의 개수 등 기본 구조에 관한 아키텍쳐 정하기
3. Loss가 잘 나오는지 확인




- Softmax 함수, 규제값을 0으로 설정 => -log1/c 값으로 2.3
- 규제값 올렸을 때, loss가 어떻게 변하는지 sanity check

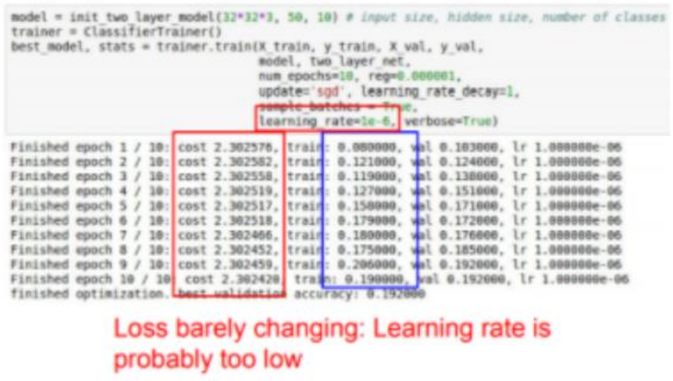
# #Babysitting the Learning process

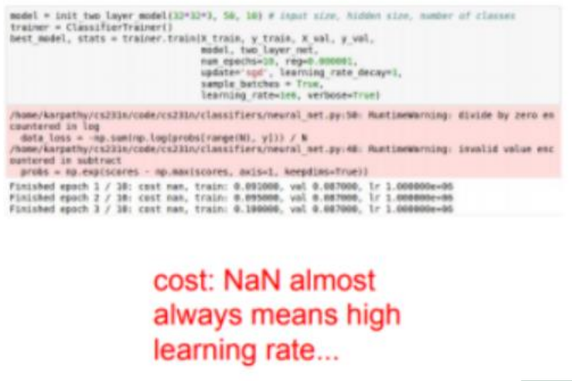
4. 실제로 훈련




- 매우 적은 데이터셋을 넣었을 때, 과적합 되면 모델이 정상적으로 작동중임

5. 규제값과 학습률 찾기 => 결과 확인 반복

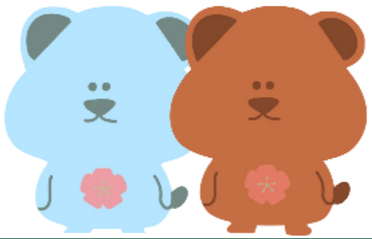



- 작은 regularization값을 넣고, 학습률을 낮게/높게 설정해보고 결과 확인 반복

=> 반복으로 적절한 학습률을 찾아감

# Hyperparameter Optimization



EWHA EURON

# #Hyperparameter Optimization

< 하이퍼 파라미터 최적화 >

1. Hyperparameter 값을 우선 설정
2. 범위 내에서 파라미터 값을 무작위로 추출(Random search)
3. Validation set을 이용하여 평가하는 Cross-validation 진행
4. 여러 번 반복 중 정확도를 체크하면 hyperparameter값의 범위를 좁힘

Now run finer search...

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)
```

adjust range

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-4, 0)
      lr = 10**uniform(-3, -4)
```

```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

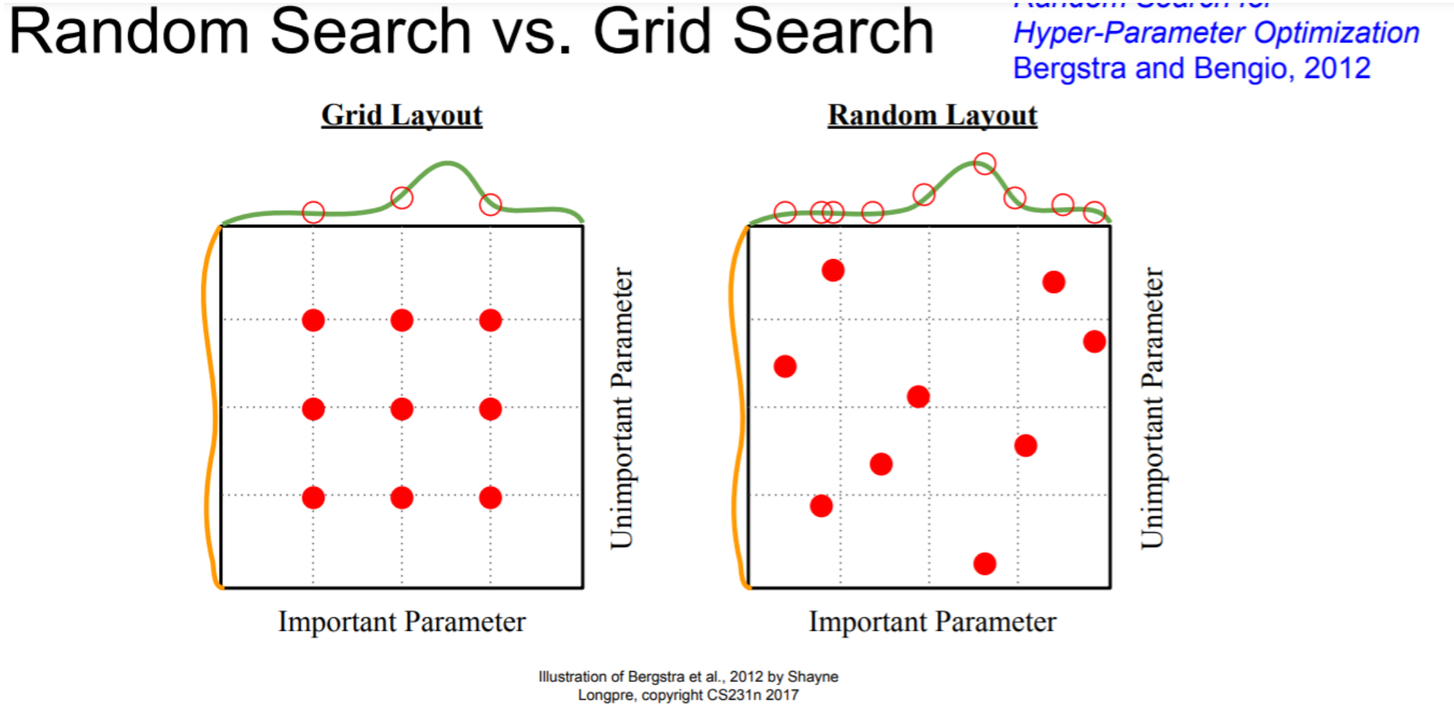
**53%** - relatively good for a 2-layer neural net with 50 hidden neurons.

But this best cross-validation result is worrying. Why?

- Hyperparameter값을 일일이 반복을 통해 찾는 과정 보단 search 방법들을 사용하는 것이 더욱 효과적이다.

EWHA EURON

# #Hyperparameter Optimization




## Grid Search

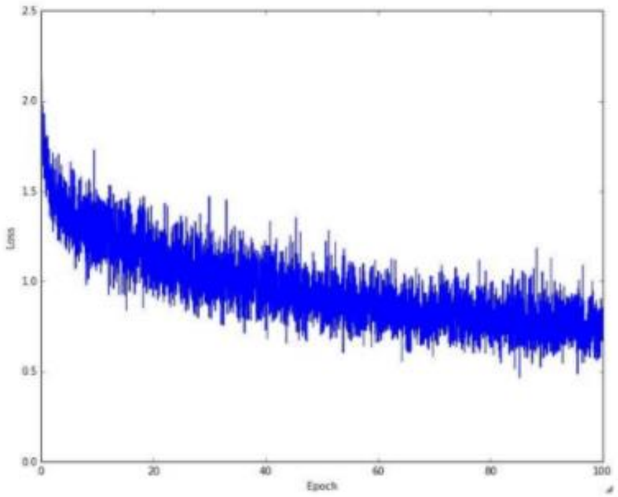
하이퍼파라미터 값의 범위를 지정해 일정한 간격을 두고 하이퍼파라미터 값을 지정, 각 값들에 대해 측정한 성능을 비교하여 가장 높은 성능을 보인 값 채택

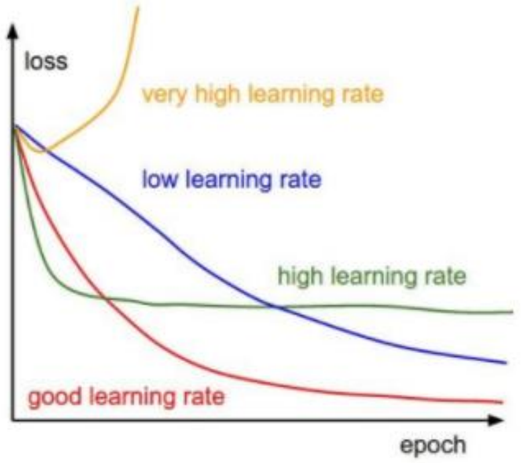
## Random Search

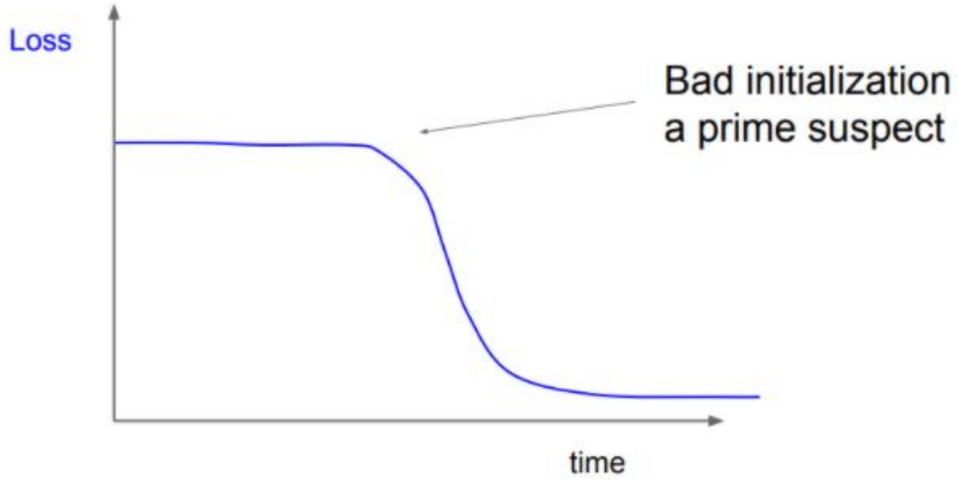
하이퍼파라미터 값의 범위를 지정해 범위내의 하이퍼파라미터 값들을 랜덤 샘플링하여 값을 지정, 불필요한 연산 수행 줄여 더 빠르게 찾을 수 있음. 더 많이 사용됨.

# #Hyperparameter Optimization

Monitor and visualize the loss curve



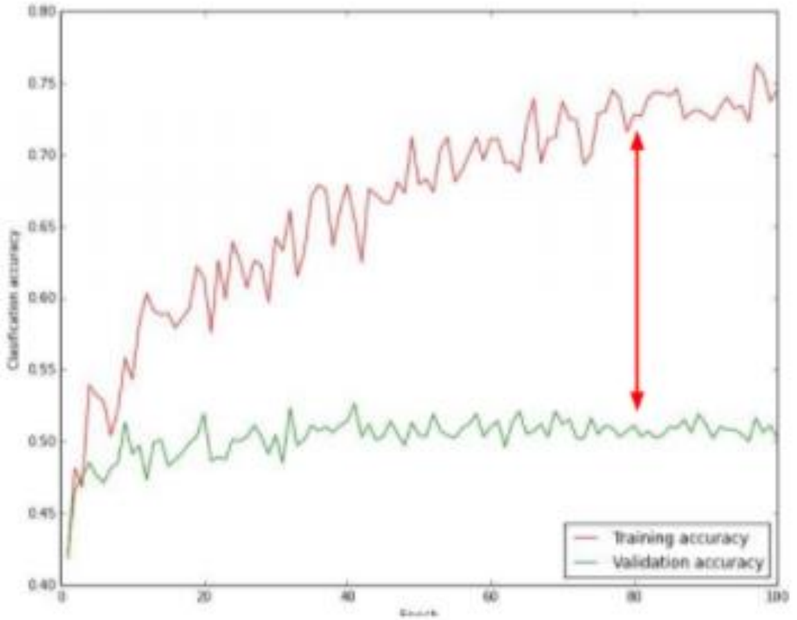

## Loss Curve

모니터링시 하이퍼 파라미터가 적합한지 아닌지 평가
학습률의 경우, 빨간색 선이 가장 좋은 형태

초기에 평평한 모양의 로스 커브가 나올 시 , 초기화가 잘못되었을 가능성이 크다.

# #Hyperparameter Optimization

Monitor and visualize the accuracy:

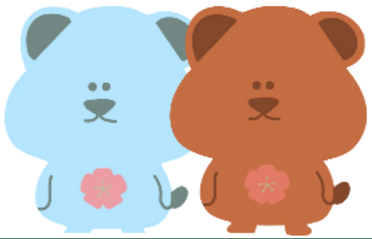



big gap = overfitting
=> increase regularization strength?

no gap
=> increase model capacity?

- 트레이닝 accuracy와 검증 accuracy의 갭이 클 경우 과적합 상태이니 규제값 강도를 올려봐야 한다.
- 반대로 갭이 없을 땐, model capacity를 늘려야 한다.

# Summary



EWHA EURON

# #Summary

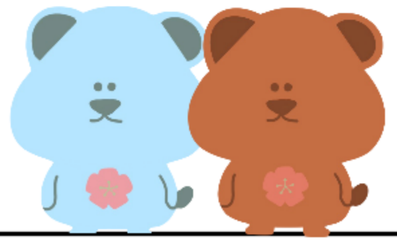
## Summary

TLDRs

We looked in detail at:

- Activation Functions (use ReLU)
- Data Preprocessing (images: subtract mean)
- Weight Initialization (use Xavier init)
- Batch Normalization (use)
- Babysitting the Learning process
- Hyperparameter Optimization
  (random sample hyperparams, in log space when appropriate)

# THANK YOU



EWHA EURON