GAN 수식 없이 쉽게 이해 해보기

- Generative Adversarial Networks -

모두의연구소

박은수 Research Director

진행할 내용들

- GAN 누구 ?
- GAN 어떻게 ? (GAN 코드 설명)

내가 사는 세상

실제 내 모습

가짜 내 모습

나는 누구인가 ...

실제 내 모습

나는 누구인가 ...

나는 어디서 왔는가 ...

실제 내 모습

내가 사는 세상

나는 어디서 왔는가 ...

실제 내 모습

내가 사는 세상

B

A

나는 어디서 왔는가 ...

이 화살표는
무엇으로 모델링 해볼까?

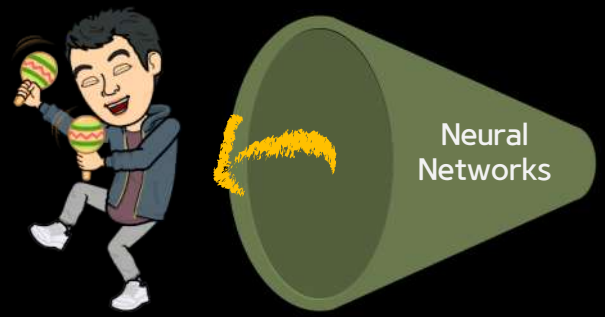
실제 내 모습

내가 사는 세상

B

A

뉴럴 네트워크 !!



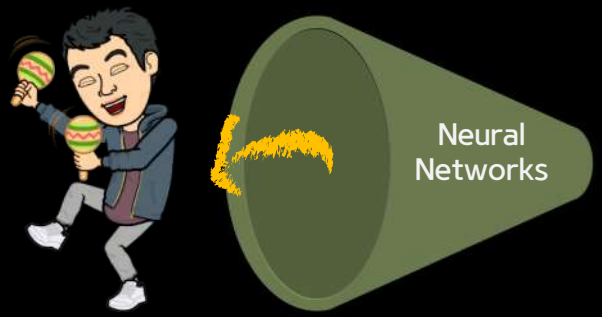
실제 내 모습

B

내가 사는 세상

A

내가 사는 세상은 ...



실제 내 모습

B

내가 사는 세상은
어떻게 생겼을까 ?

내가 사는 세상

A

이 숫자는 28x28 공간에
살고 있는가 ...

총 784 (28*28) 차원

28

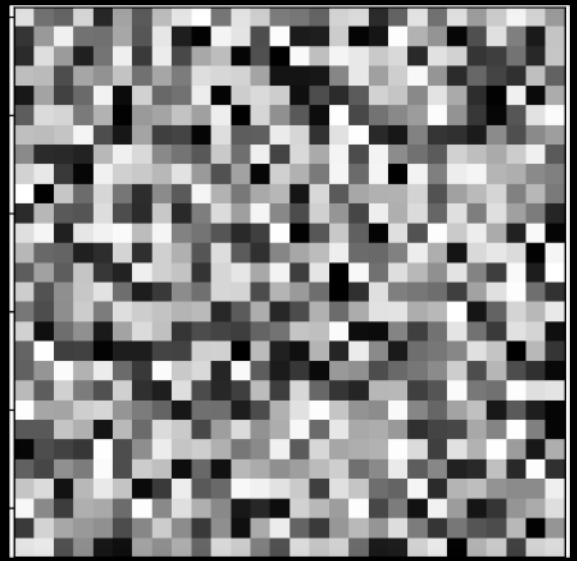
28

```
In [25]: image =  (np.random.rand(28,28)*255).astype(int)
```

4가 나올 때까지 해보자!!

계속 하다 보면 나올 수 있어 ....

이 공간 안에 살고 있잖아~

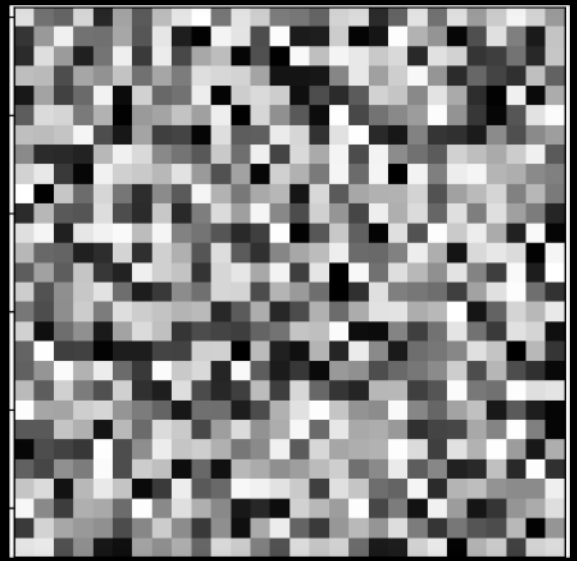


```
In [25]: image =  (np.random.rand(28,28)*255).astype(int)
```

4가 나올
때까지 해보자!!

계속 하다 보면 나올 수 있어 ....

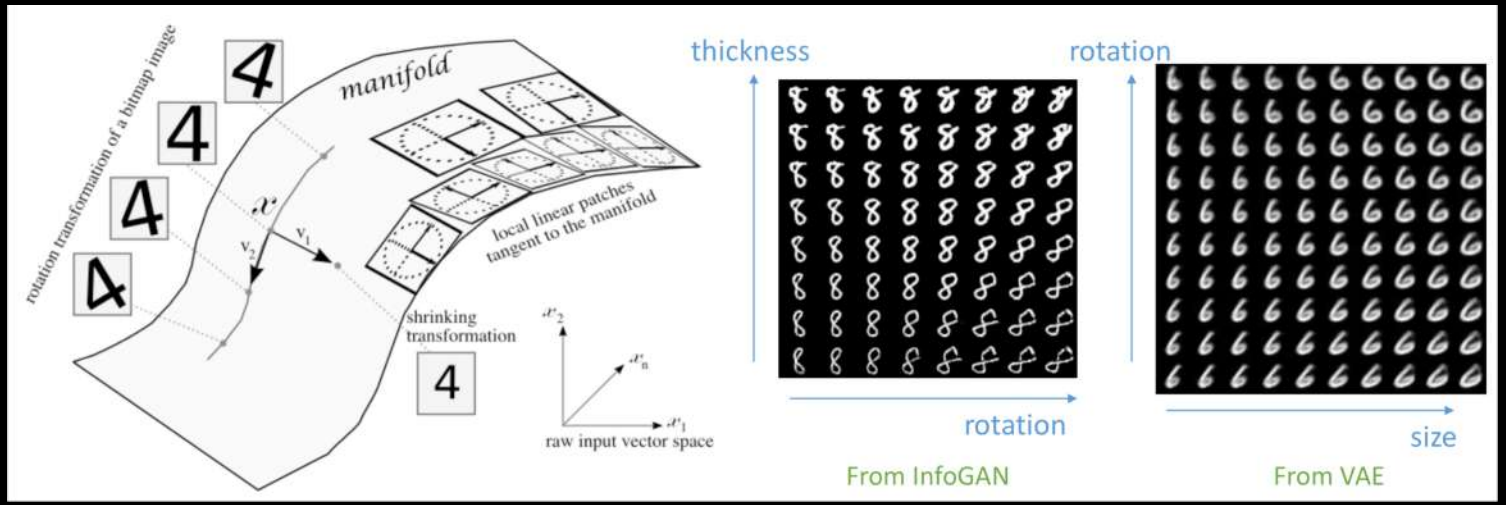
이 공간 안에 살고 있잖아~



살려줘 ...
이러다 죽겠어!!

숫자 4는 우리 눈에는 28x28로 보이지만,
어쩌면 이 보다 더 낮은 차원에 살고 있는 걸지도 몰라

한 축은 label, 한 축은 회전, 한 축은 두께, 한 축은 ..



숫자 4는 우리 눈에는 28x28로 보이지만,
어쩌면 이 보다 더 낮은 차원에 살고 있는 걸지도 몰라

한 축은 label, 한 축은 회전, 한 축은 두께, 한 축은 ..

4	숫자 레이블
5	회전
3	두께

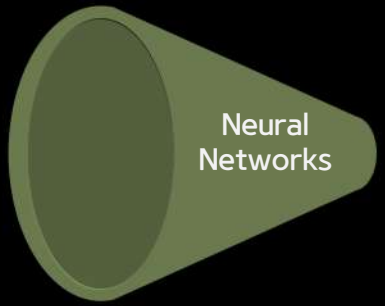
숫자 4는 우리 눈에는 28x28로 보이지만,
어쩌면 이 보다 더 낮은 차원에 살고 있는 걸지도 몰라

한 축은 label, 한 축은 회전, 한 축은 두께, 한 축은 ..

8	숫자 레이블
30	회전
2	두께

숫자 4는 우리 눈에는 28x28로 보이지만,
어쩌면 이 보다 더 낮은 차원에 살고 있는 걸지도 몰라

한 축은 label, 한 축은 회전, 한 축은 두께, 한 축은 ..



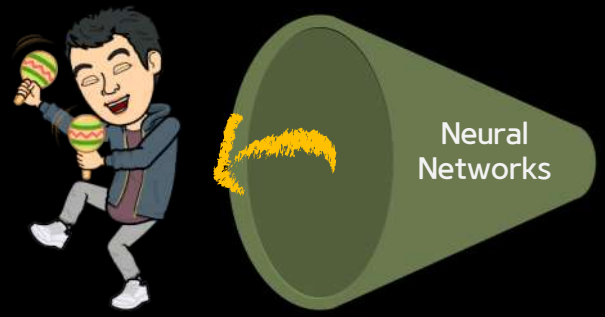
8
30
2

숫자 레이블

회전

두께

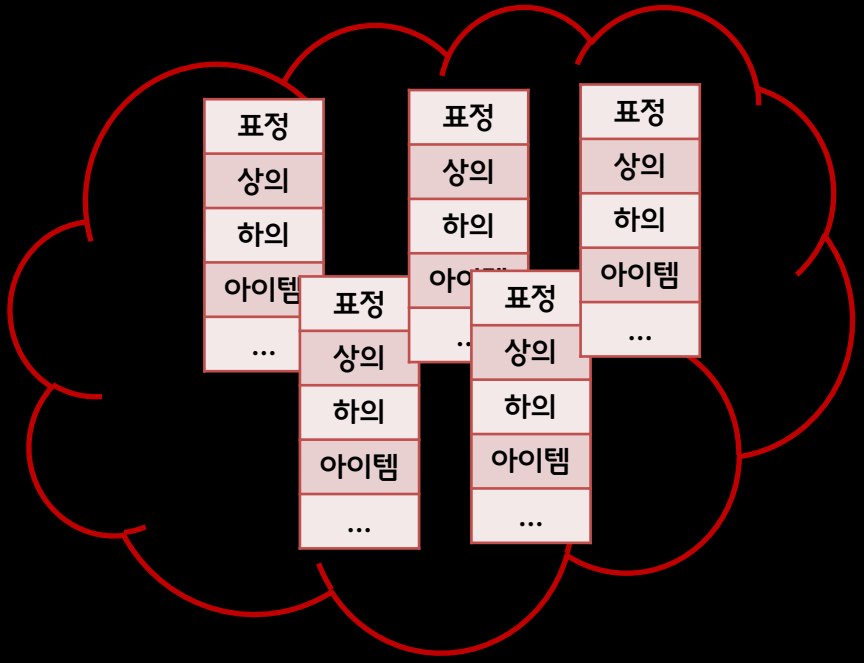
내가 사는 세상도 어쩌면 ...



Neural Networks

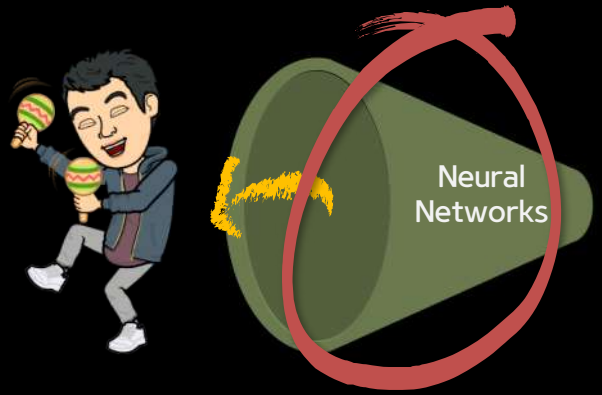
실제 내 모습

B



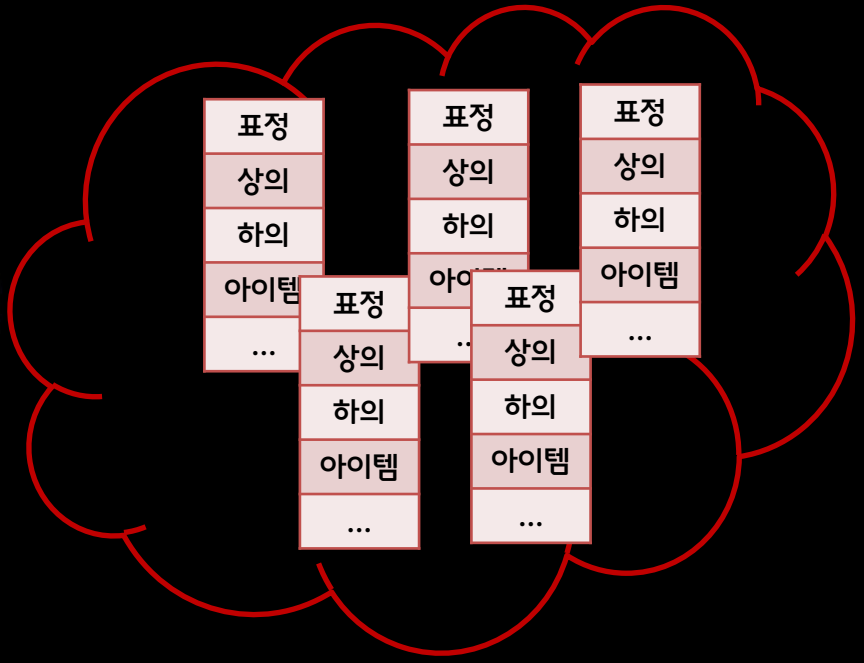
A

How ?


Neural
Networks

실제 내 모습

B


표정
상의
하의
아이템
…
표정
상의
하의
아이템
…
표정
상의
하의
아이템
…
표정
상의
하의
아이템
…

A

이런걸 하는 방법은 다행 하지만 .. 오늘 배울 것은 ...

GAN 누구?

Generative Adversarial Networks

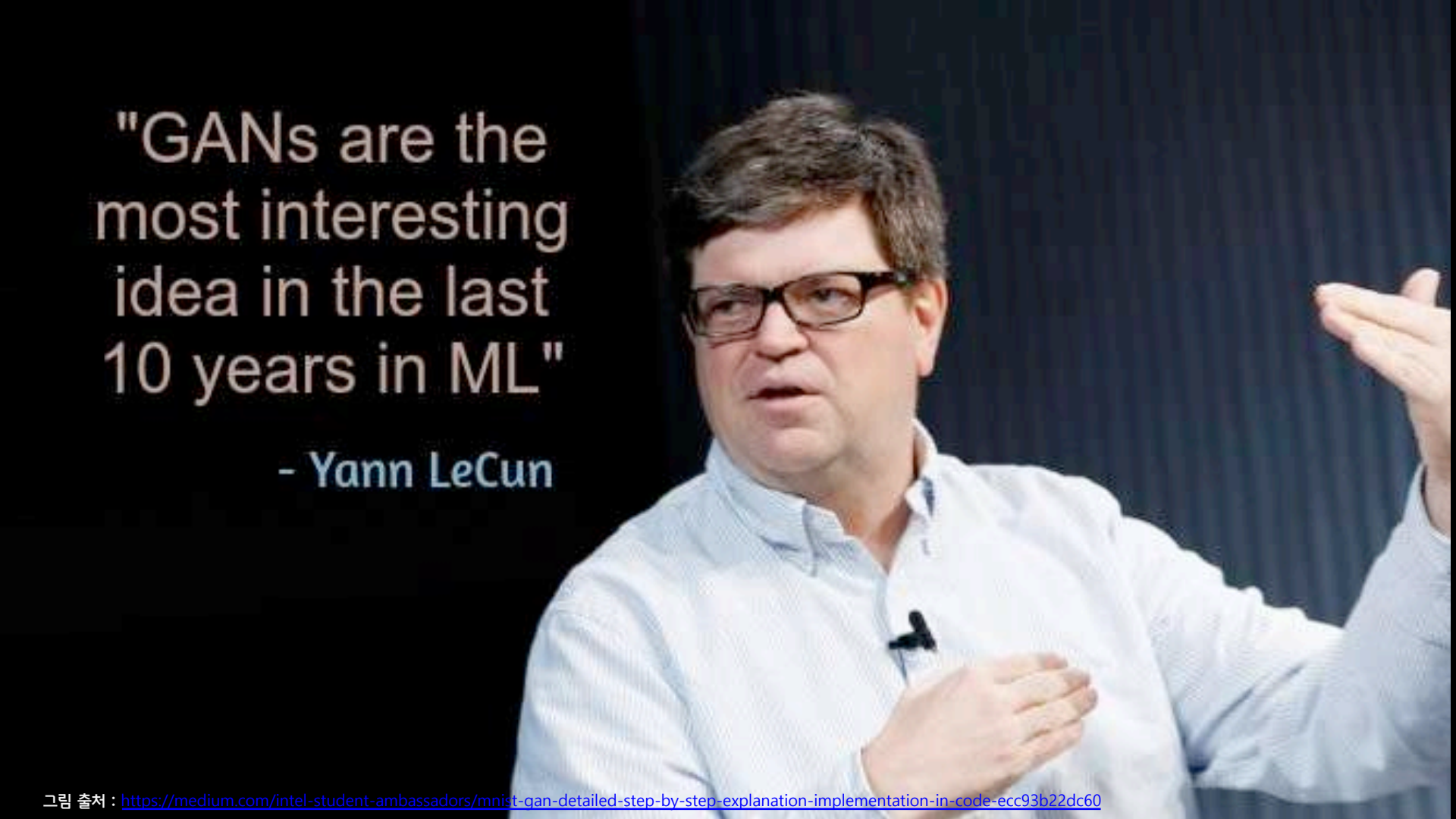


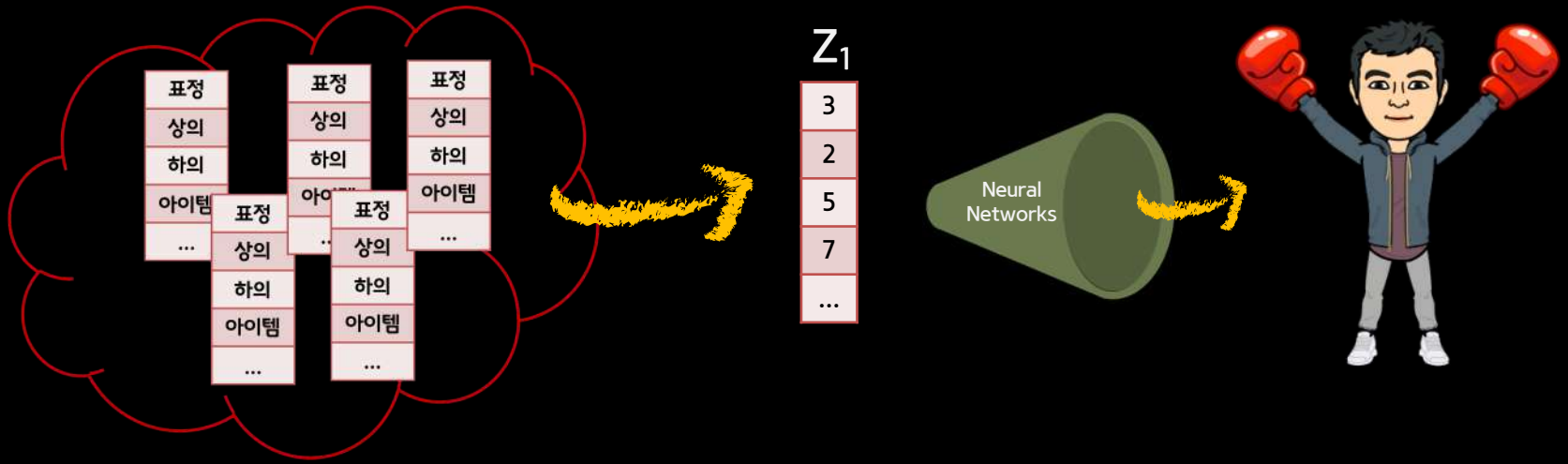
그림 출처 : https://medium.com/intel-student-ambassadors/mnist-gan-detailed-step-by-step-explanation-implementation-in-code-ecc93b22dc60

다양한 나의 모습 속 나의 모습

나의 다양한 공간

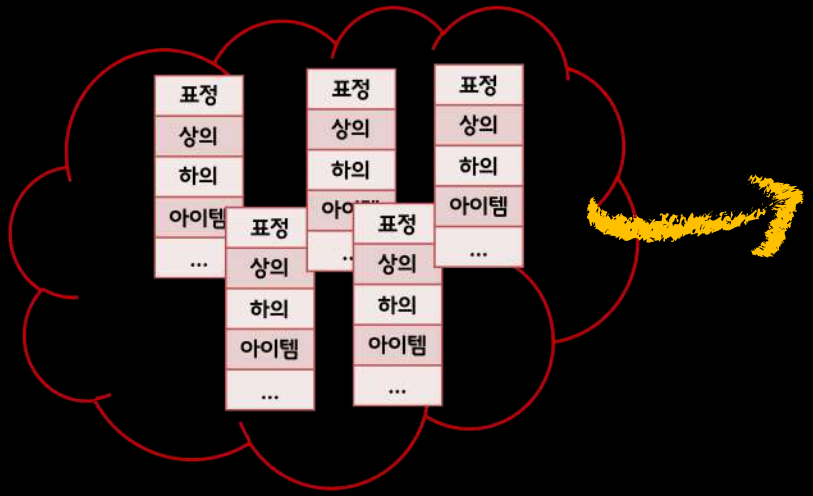
세상으로 나온 나

우린 더 단순한 공간에 있을지도 ...



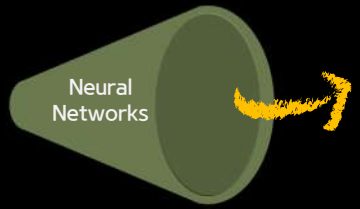
우린 더 단순한 공간에 있을지도 ...

완성 ??



z_1

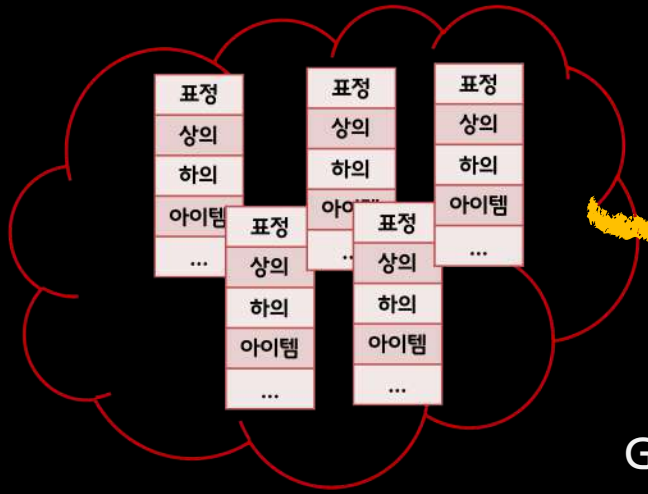
3
2
5
7
...



그럴리가 !!

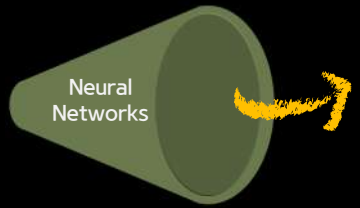
나의 다양한 공간

세상으로 나온 나



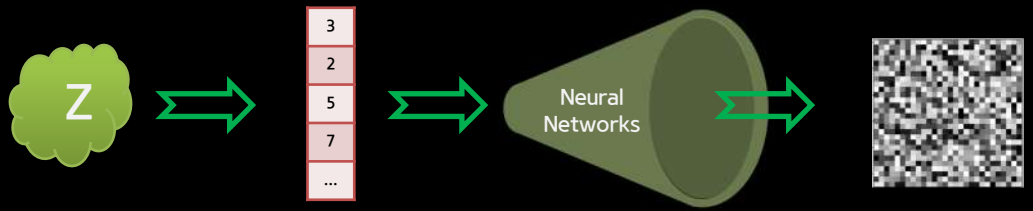
Z_1

3
2
5
7
...



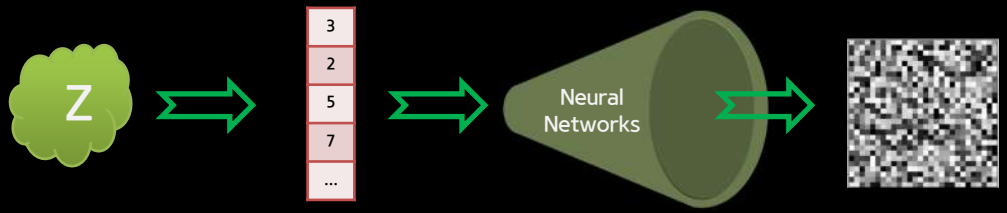
GAN으로 해결해 봅시다

Generative Adversarial Networks



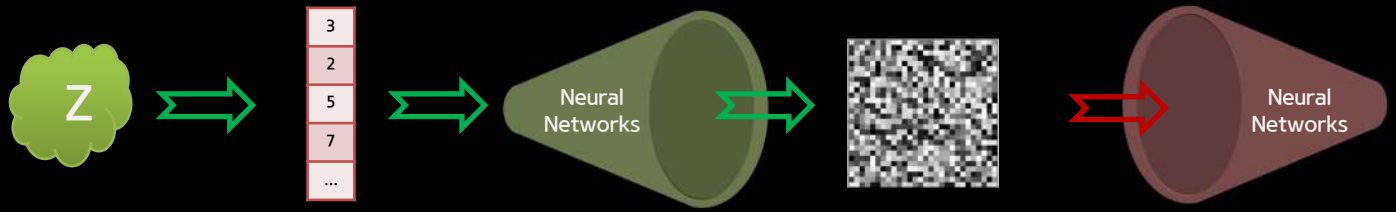
Generator

Generative Adversarial Networks



Generator

Generative Adversarial Networks



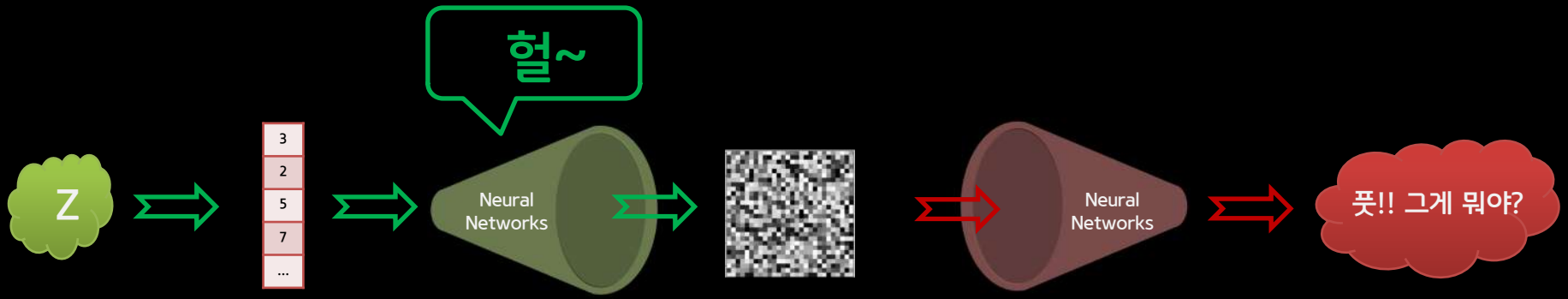
Generator

Generative Adversarial Networks

Generator

Discriminator

Generative Adversarial Networks

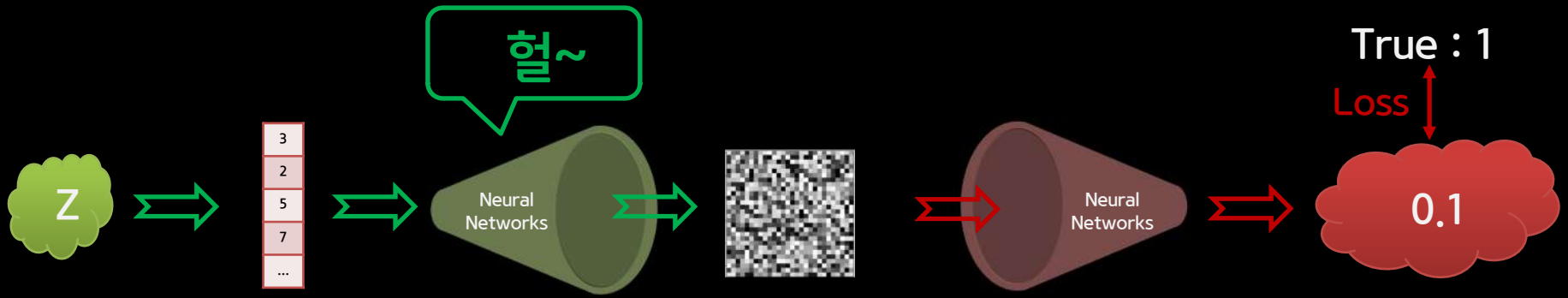


Generator

Discriminator

헐 ~ : Generator 학습

Generative Adversarial Networks

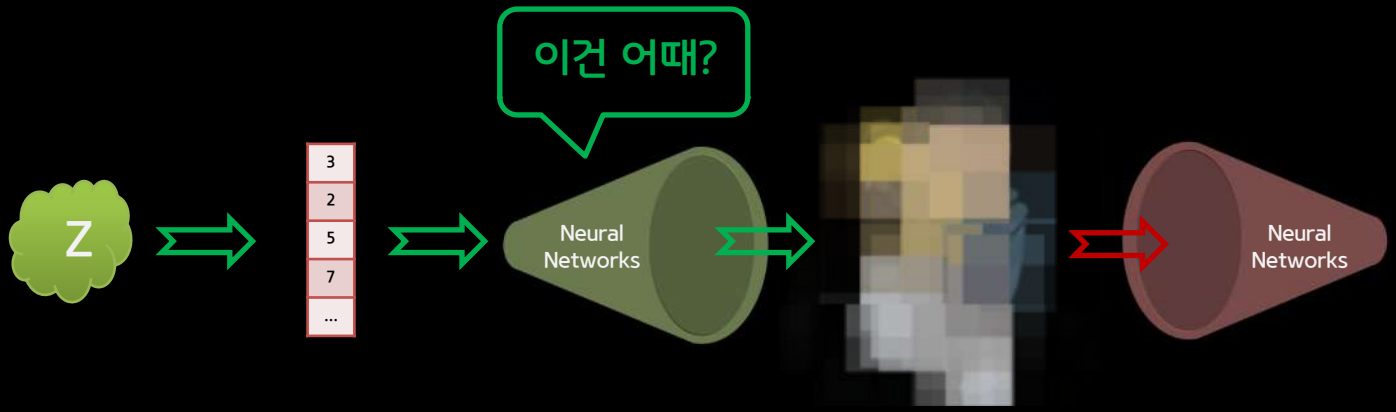


Generator

Discriminator

헐 ~ : Generator 학습

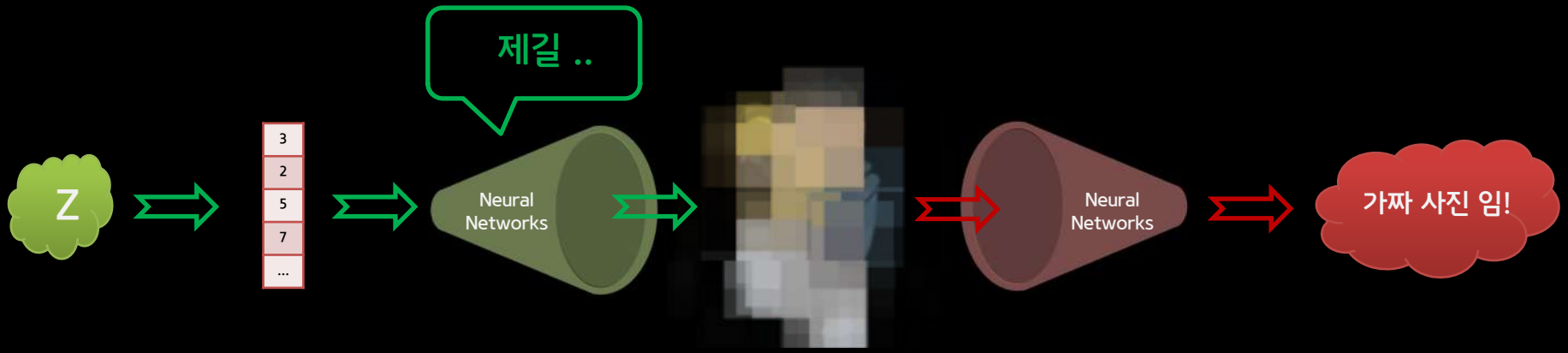
Generative Adversarial Networks



Generator

Discriminator

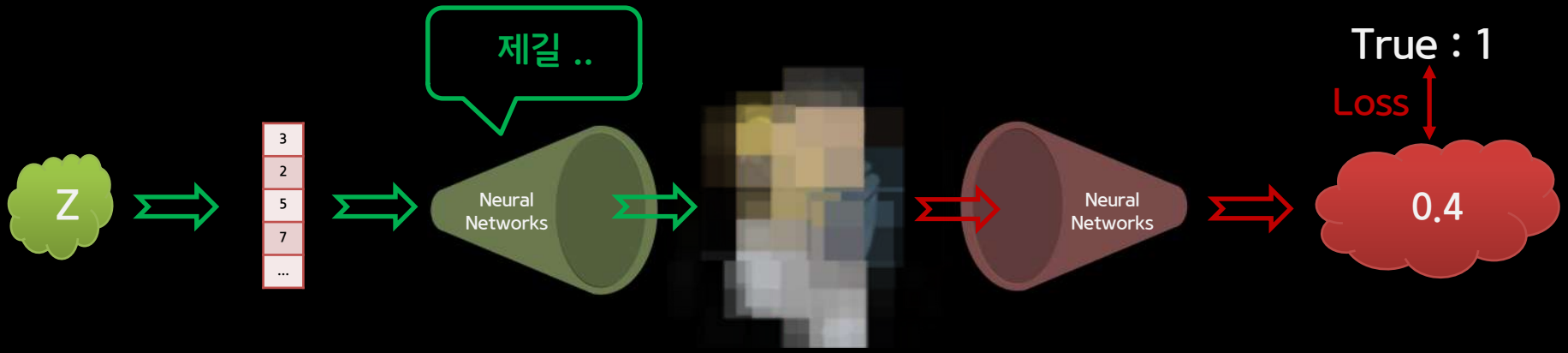
Generative Adversarial Networks



Generator Discriminator

제길 ~ : Generator 학습

Generative Adversarial Networks

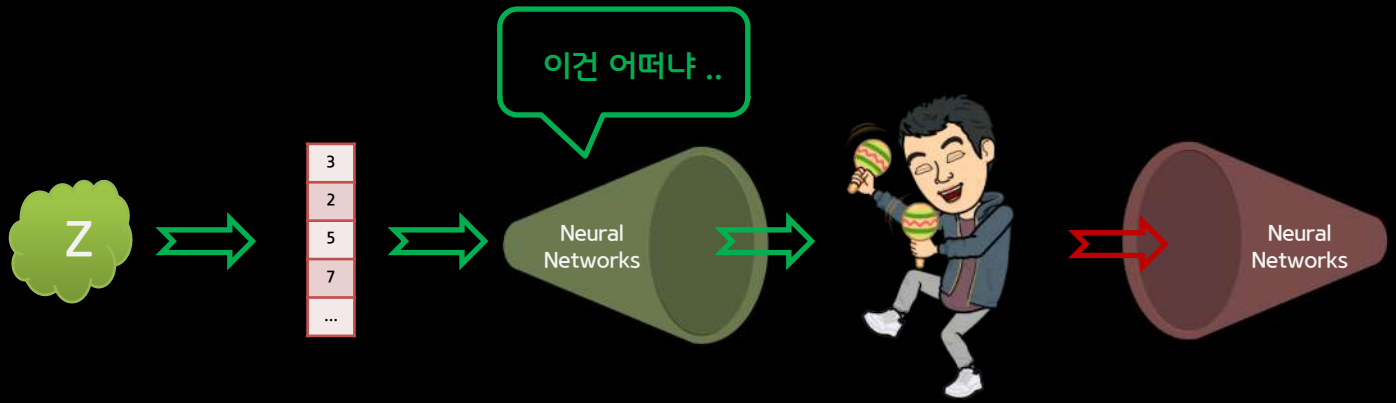


Generator

Discriminator

제길 ~ : Generator 학습

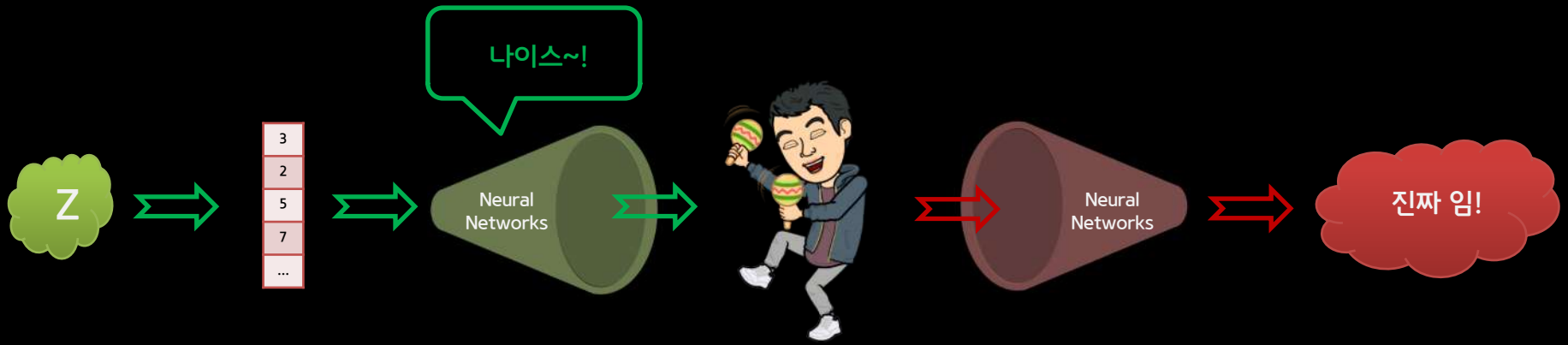
Generative Adversarial Networks



Generator

Discriminator

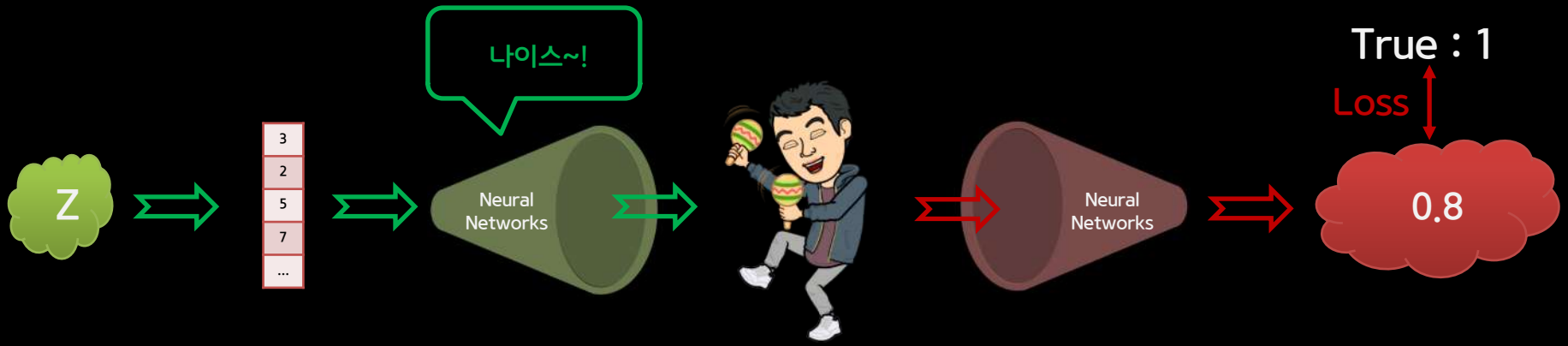
Generative Adversarial Networks



Generator

Discriminator

Generative Adversarial Networks

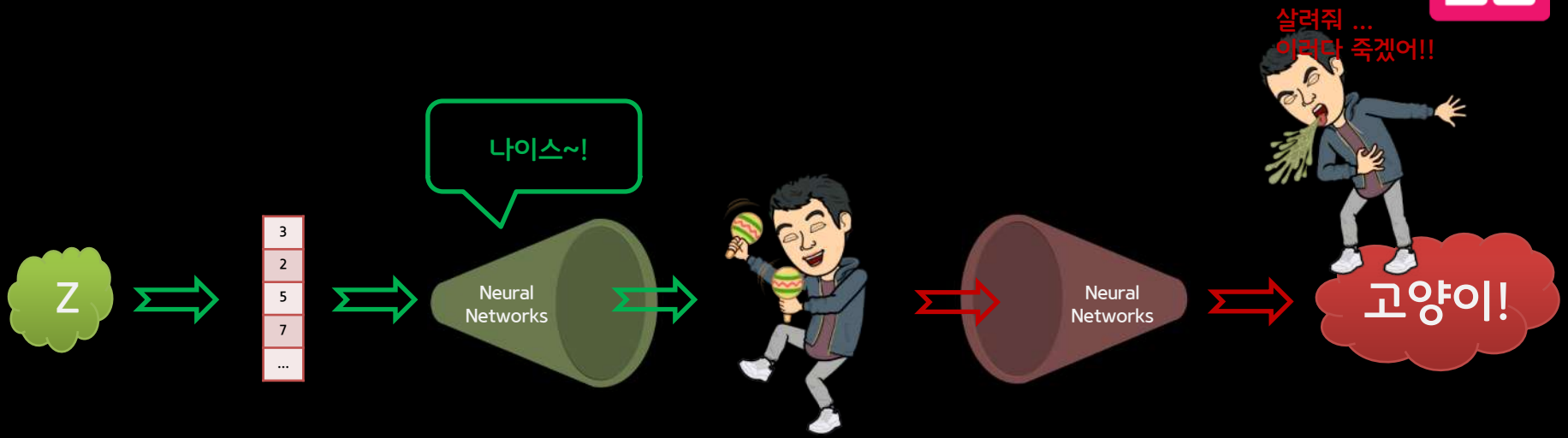


Generator

Discriminator

나이스~! : Generator 학습

그러나 실제로는 !!

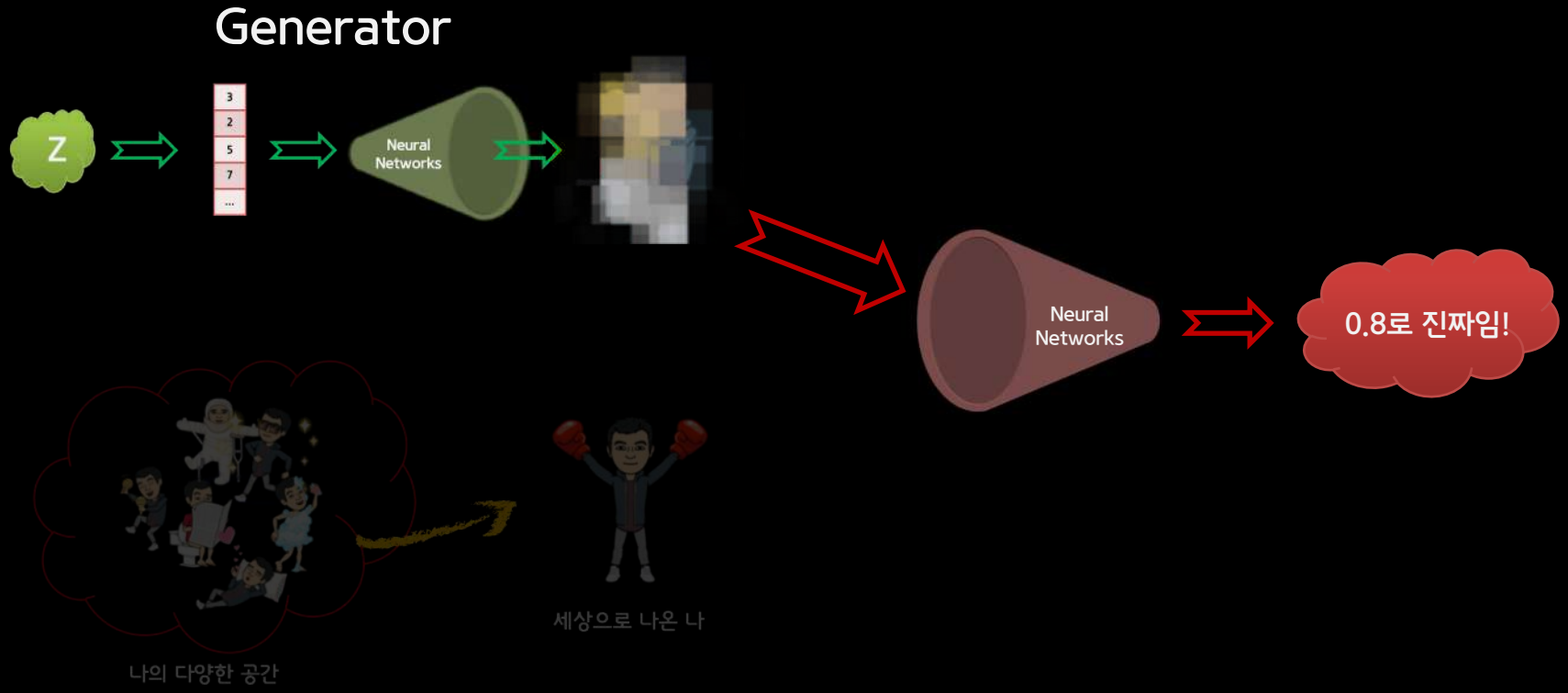


Generator

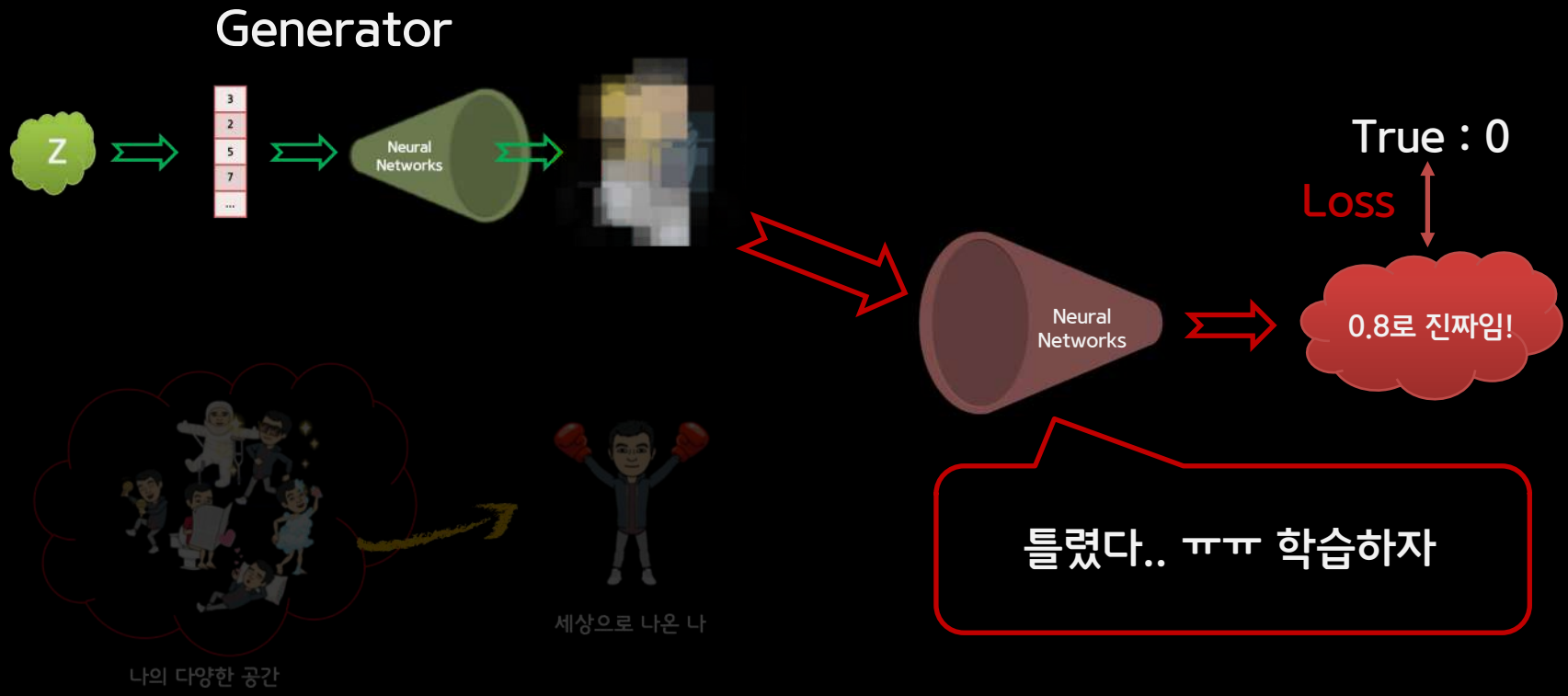
Discriminator

Discriminator는 똑똑 하지가 않습니다 ...

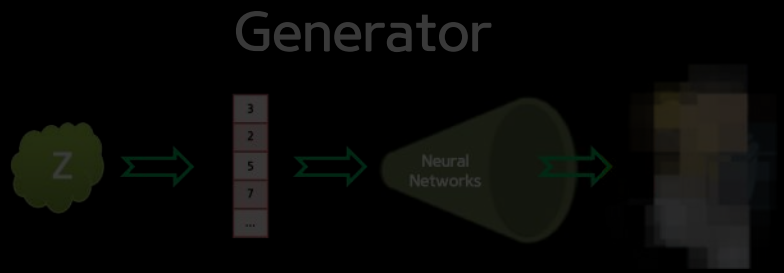
똑똑한 Discriminator를 위한 학습

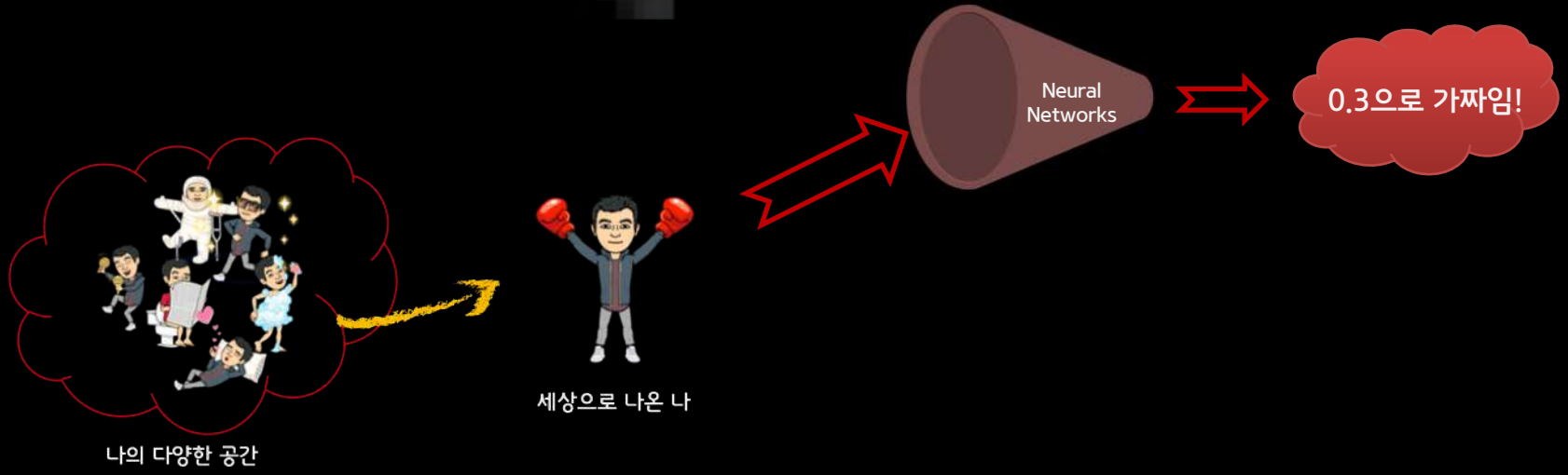


똑똑한 Discriminator를 위한 학습

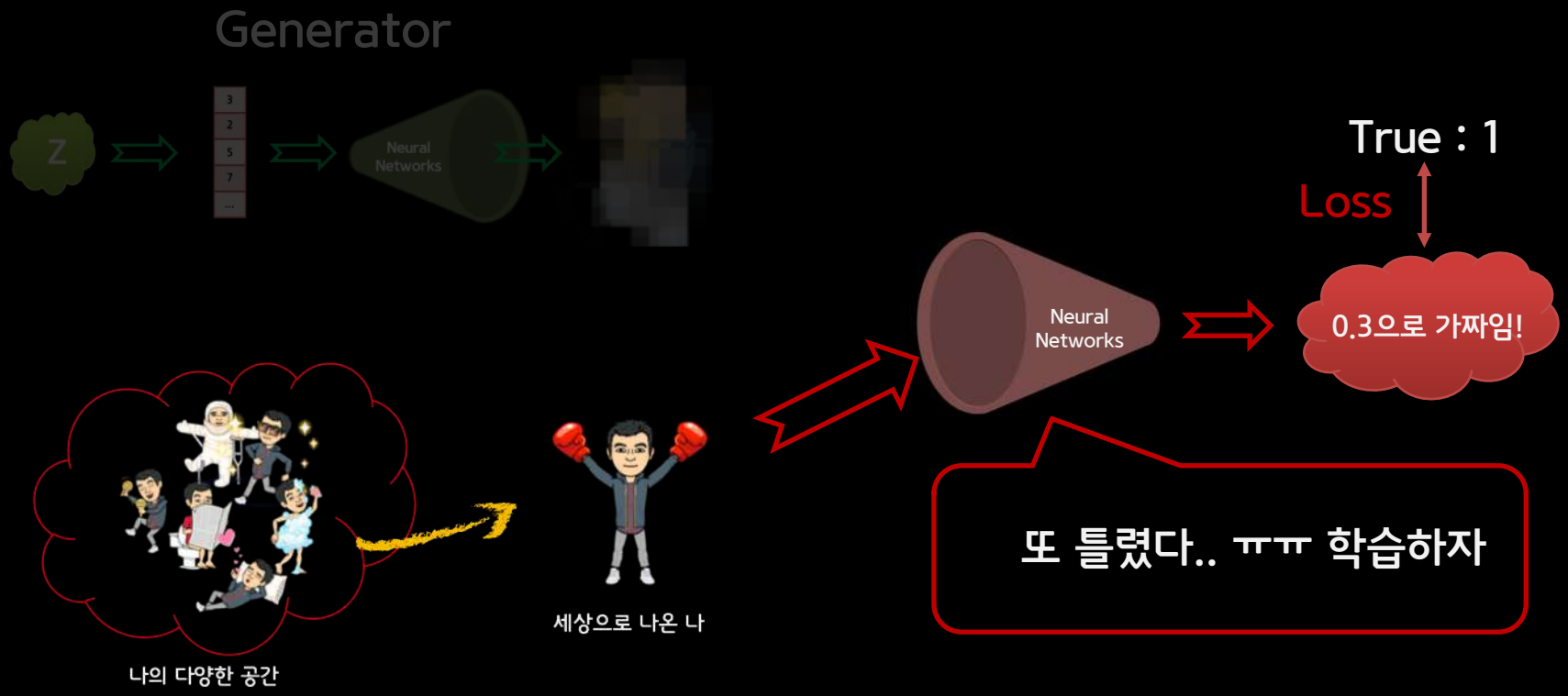


똑똑한 Discriminator를 위한 학습

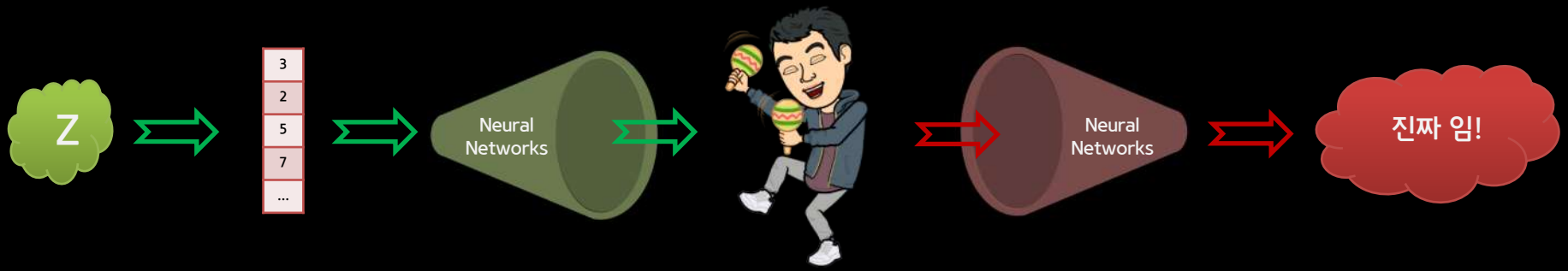




똑똑한 Discriminator를 위한 학습



Generative Adversarial Networks



Generator

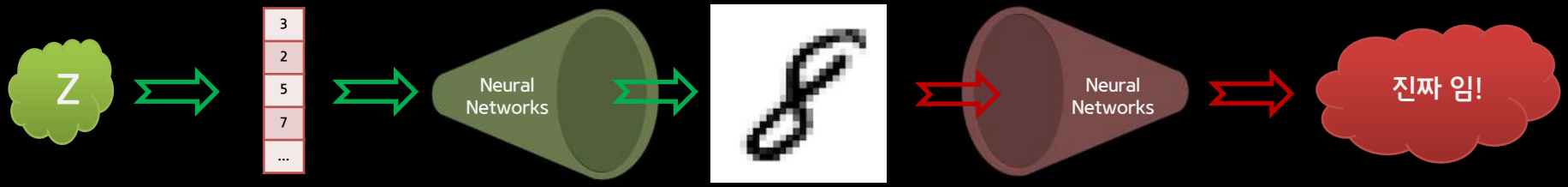
Discriminator

GAN 어떻게 ? (코드 설명)

https://www.tensorflow.org/beta/tutorials/generative/dcgan

추가로 새로 보이는 것들에 대한 이론까지 ...

구현할 GAN



Generator

Discriminator

Dataset 준비

데이터셋 로딩 및 준비

생성자와 감별자를 훈련하기위해 MNIST 데이터셋을 사용할것입니다. 생성자는 손글씨 숫자 데이터를 닮은 숫자들을 생성할 것입니다.

```
[7]  (train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
```

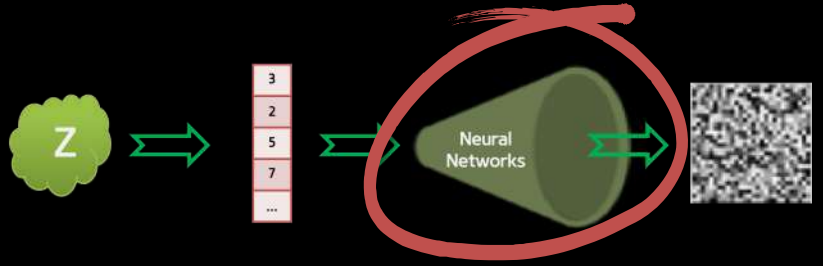
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

```
[8]  train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
     train_images = (train_images - 127.5) / 127.5 # 이미지를 [-1, 1]로 정규화합니다.
```

```
[9]  BUFFER_SIZE = 60000
     BATCH_SIZE = 256
```

```
[10] # 데이터 배치를 만들고 섞습니다.
     train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

Generator 만들기

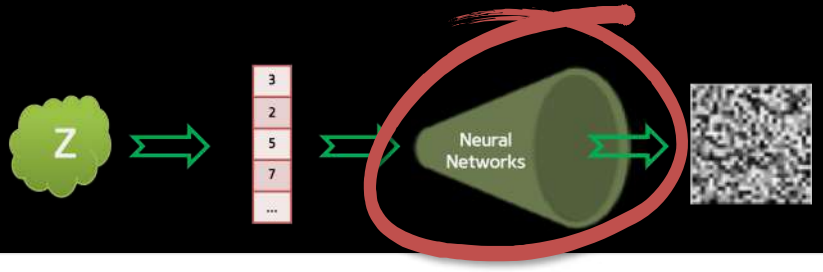


생성자

생성자는 시드값 (seed; 랜덤한 잡음)으로부터 이미지를 생성하기 위해, `tf.keras.layers.Conv2DTranspose` (업샘플링) 층을 이용합니다. 처음 `Dense`층은 이 시드값을 인풋으로 받습니다. 그 다음 원하는 사이즈 28x28x1의 이미지가 나오도록 업샘플링을 여러번 합니다. tanh를 사용하는 마지막 층을 제외한 나머지 각 층마다 활성함수로 `tf.keras.layers.LeakyReLU`을 사용하고 있음을 주목합시다.

```
[ ]  def make_generator_model():
         model = tf.keras.Sequential()
         model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
         model.add(layers.BatchNormalization())
         model.add(layers.LeakyReLU())

         model.add(layers.Reshape((7, 7, 256)))
         assert model.output_shape == (None, 7, 7, 256) # 주목: 배치사이즈로 None이 주어집니다.

         model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
         assert model.output_shape == (None, 7, 7, 128)
         model.add(layers.BatchNormalization())
         model.add(layers.LeakyReLU())

         model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
         assert model.output_shape == (None, 14, 14, 64)
         model.add(layers.BatchNormalization())
         model.add(layers.LeakyReLU())

         model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
         assert model.output_shape == (None, 28, 28, 1)

         return model
```

Generator 만들기



생성자

생성자는 시드값 (seed; 랜덤한 잡음)으로부터 이미지를 생성하기 위해, `tf.keras.layers.Conv2DTranspose` (업샘플링) 층을 이용합니다. 처음 `Dense`층은 이 시드값을 인풋으로 받습니다. 그 다음 원하는 사이즈 28x28x1의 이미지가 나오도록 업샘플링을 여러번 합니다. tanh를 사용하는 마지막 층을 제외한 나머지 각 층마다 활성함수로 `tf.keras.layers.LeakyReLU`을 사용하고 있음을 주목합시다.

```
[ ]  def make_generator_model():
         model = tf.keras.Sequential()
         model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
         model.add(layers.BatchNormalization())
         model.add(layers.LeakyReLU())

         model.add(layers.Reshape((7, 7, 256)))
         assert model.output_shape == (None, 7, 7, 256) # 주목: 배치사이즈로 None이 주어집니다.

         model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
         assert model.output_shape == (None, 7, 7, 128)
         model.add(layers.BatchNormalization())
         model.add(layers.LeakyReLU())

         model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
         assert model.output_shape == (None, 14, 14, 64)
         model.add(layers.BatchNormalization())
         model.add(layers.LeakyReLU())

         model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
         assert model.output_shape == (None, 28, 28, 1)

         return model
```

안 배운 것들

- Batch Normalization
- Leaky RELU
- Transpose Convolution

가중치의 초깃값

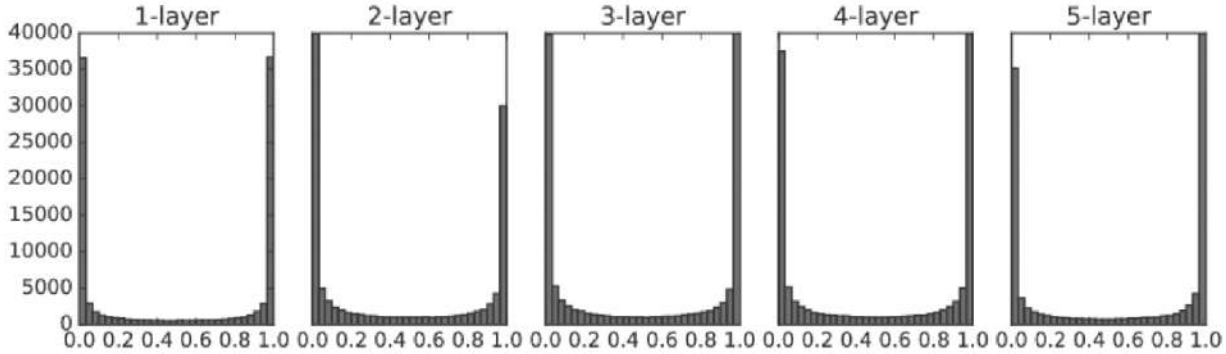
모두의연구소

- 은닉층의 활성화 값 분포
  - 초깃값의 변화에 따라 은닉층의 활성화 값 분포가 어떻게 변하는지 확인해 보고자 합니다
  - 실험내용
    - 5개의 층, 각 층의 뉴런은 100개
    - 입력 데이터로 1000개의 데이터를 무작위로 생성
    - 활성화 함수로 시그모이드
    - 각 층의 활성화 함수 값을 activations변수에 저장

가중치의 초깃값

모두의연구소

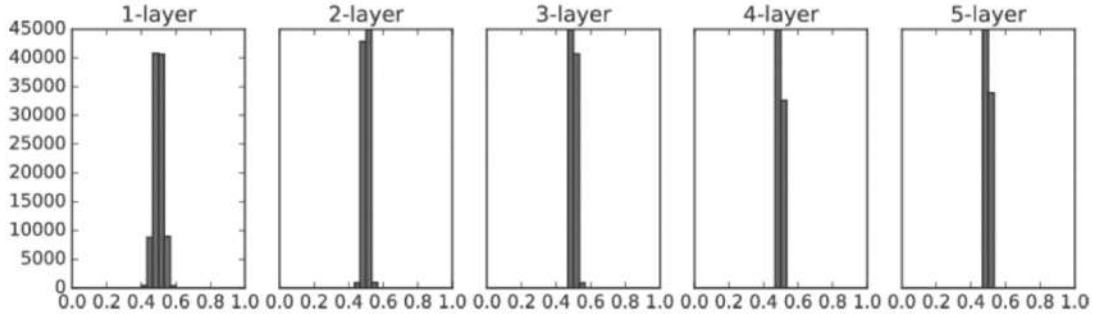
- 은닉층의 활성화 값 분포
  - 가중치를 표준편차가 1인 정규분포로 초기화 할때의 각 층의 활성화값 분포



- 값이 0과 1에 치우져 분포되어 있습니다
- 이 경우 시그모이드의 미분은 0에 가까워집니다
- 역전파 시 점점 그 값이 사라 집니다 (gradient vanishing)
- 층을 깊게 하면 더 심각해 질 것임

가중치의 초깃값

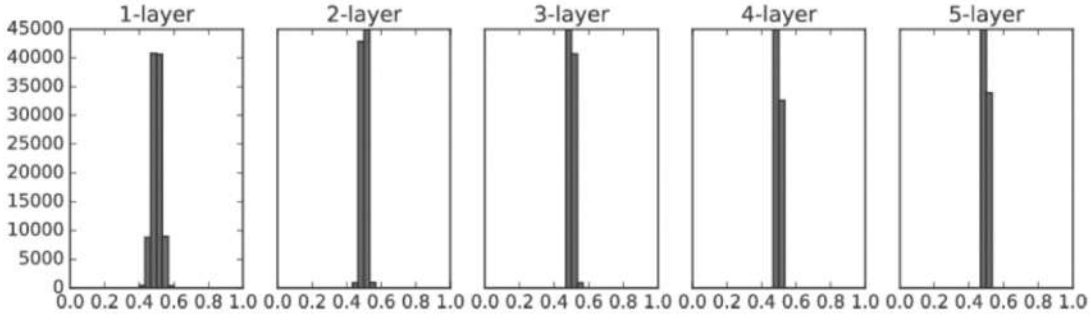
- 은닉층의 활성화 값 분포
  - 가중치를 표준편차가 0.01인 정규분포로 초기화 할때의 각 층의 활성화값 분포



  - 0.5 부근에 집중 됨. 기울기 소실이 발생하지 않음
  - 활성화 값이 치우쳐 있다는 것은 표현력 관점에서 문제가 있는 것
    - 다수의 뉴런이 거의 같은 값을 출력하니 뉴런을 여러 개 둔 의미가 없어짐. 100개가 거의 같은 값을 출력하니 1개짜리와 별반 다를바 없음
    - **활성화 값이 치우치면 표현력이 제한되어 있는 것임**

가중치의 초깃값

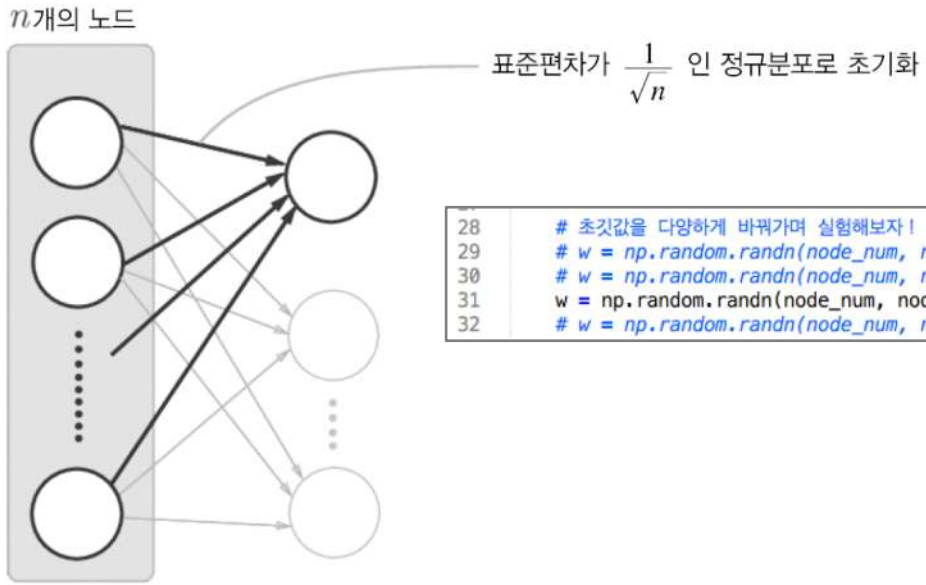
- 은닉층의 활성화 값 분포
  - 가중치를 표준편차가 0.01인 정규분포로 초기화 할때의 각 층의 활성화값 분포



- **WARNING.** 각 층의 활성화 값은 적당히 고루 분포되어야 합니다. 층과 층 사이에 적당하게 다양한 데이터가 흐르게 해야 신경망 학습이 효율적으로 이뤄지기 때문입니다. 반대로 치우친 데이터가 흐르면 기울기 소실이나 표현력 제한 문제에 빠져 학습이 잘 이뤄지지 않는 경우가 생깁니다.

가중치의 초깃값

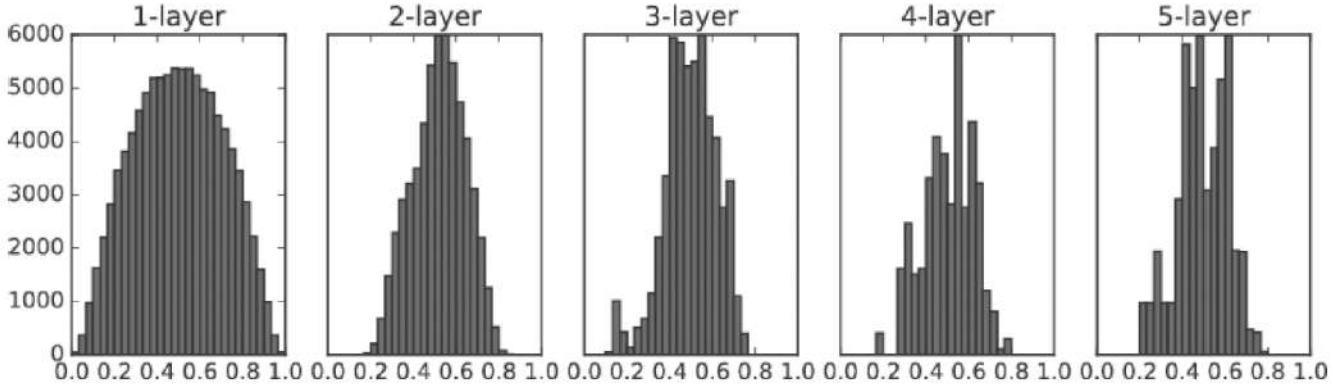
- 은닉층의 활성화 값 분포
  - Xavier 초깃값
    - 초깃값의 표준편차가 $\frac{1}{\sqrt{n}}$인 분포를 사용




```
28      # 초깃값을 다양하게 바꿔가며 실험해보자!
29      # w = np.random.randn(node_num, node_num) * 1
30      # w = np.random.randn(node_num, node_num) * 0.01
31      w = np.random.randn(node_num, node_num) * np.sqrt(1.0 / node_num)
32      # w = np.random.randn(node_num, node_num) * np.sqrt(2.0 / node_num)
```

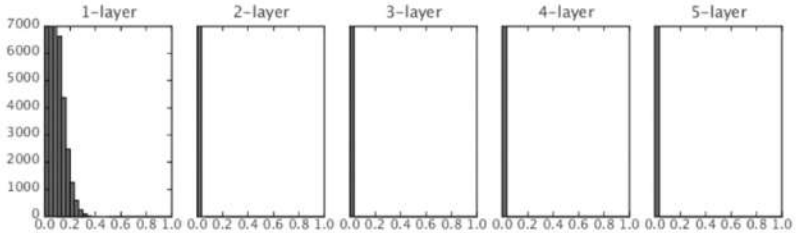
가중치의 초깃값

- 은닉층의 활성화 값 분포
  - Xavier 초깃값
    - 초깃값의 표준편차가 $\frac{1}{\sqrt{n}}$인 분포를 사용

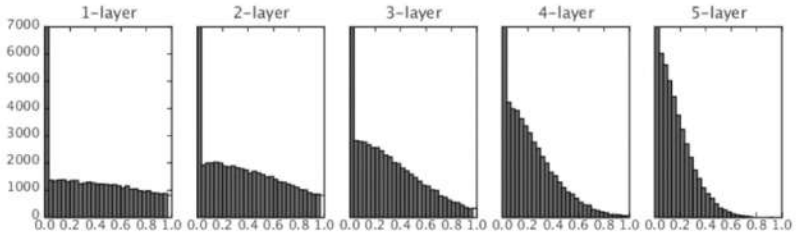


가중치의 초깃값

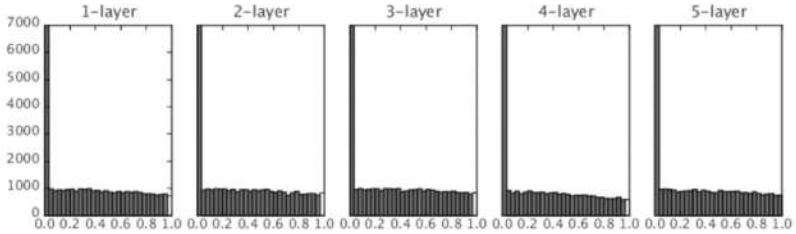
- 시그모이드 대신 ReLU를 사용한다면
  - He 초깃값 (Kaming He) : 표준편차가 $\frac{2}{\sqrt{n}}$인 분포 사용



표준편차가 0.01인 정규분포를 가중치 초깃값으로 사용한 경우



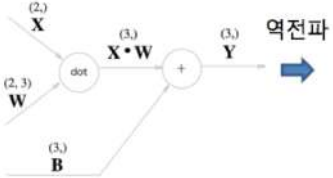
Xavier 초깃값을 사용한 경우



He 초깃값을 사용한 경우

가중치의 초깃값 (ReLU)

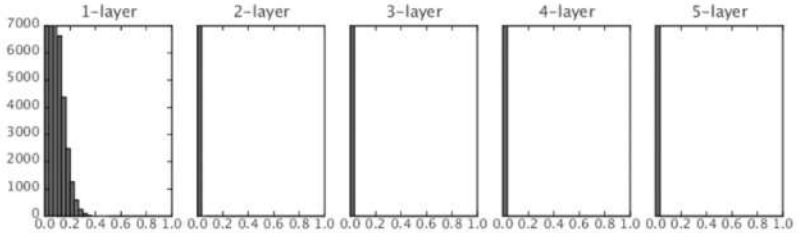
- 신경망에 작은 데이터가 흐른다는 것 -> 역전파 시 가중치의 기울기 역시 작아짐을 의미



역전파

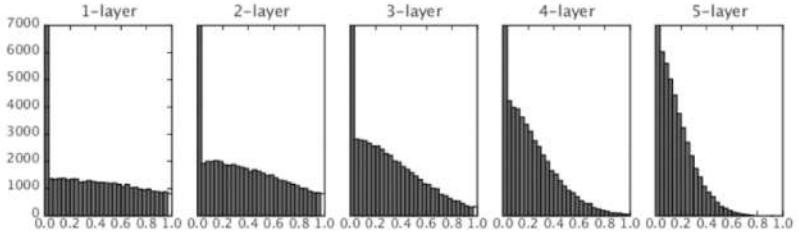
$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^{\mathrm{T}} \cdot \frac{\partial L}{\partial \mathbf{Y}}$$



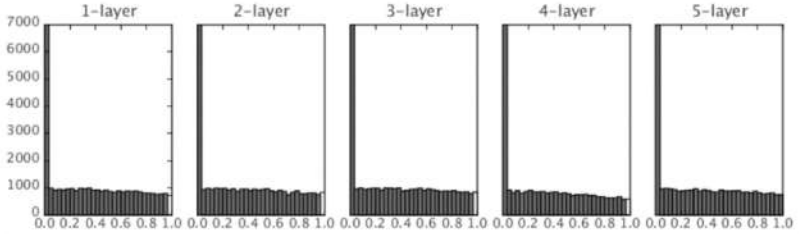
표준편차가 0.01인 정규분포를 가중치 초깃값으로 사용한 경우

- 깊어질 수록 0에 쏠림 증가



Xavier 초깃값을 사용한 경우

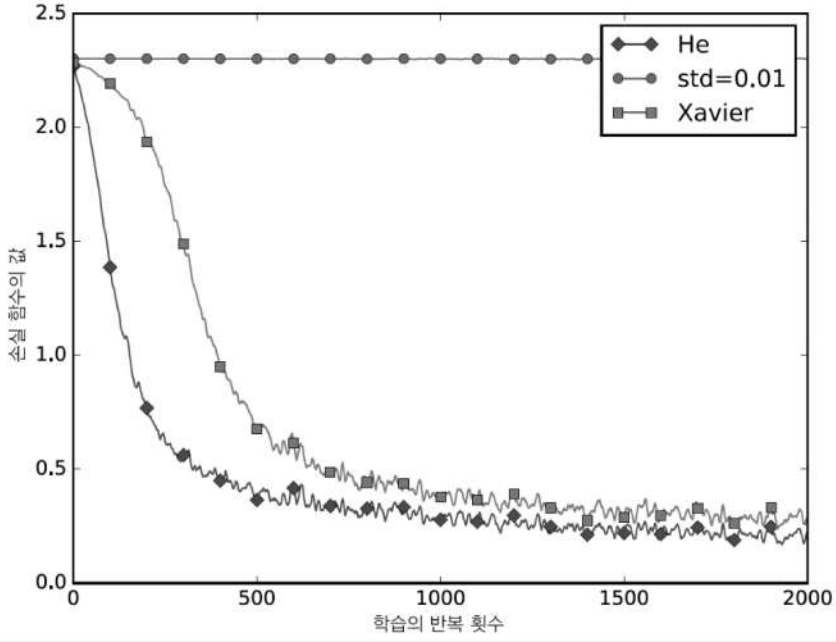
- 쓸만함



He 초깃값을 사용한 경우

가중치의 초깃값

- MNIST 데이터셋으로 본 가중치 초깃값 비교
  - 5개층, 각 뉴런 100, 활성화 함수 ReLU



초깃값이 매우 중요하군요.
그렇지만 불편하네요~

배치 정규화

배치 정규화 (Batch Normalization)

모두의연구소

Batch Normalization

[Ioffe and Szegedy, 2015]

"you want unit gaussian activations? just make them so."

consider a batch of activations at some layer.
To make each dimension unit gaussian, apply:

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

this is a vanilla differentiable function...

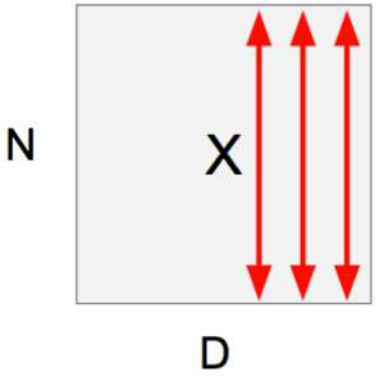
배치 정규화 (Batch Normalization)

모두의연구소

Batch Normalization

[Ioffe and Szegedy, 2015]

"you want unit gaussian activations?
just make them so."



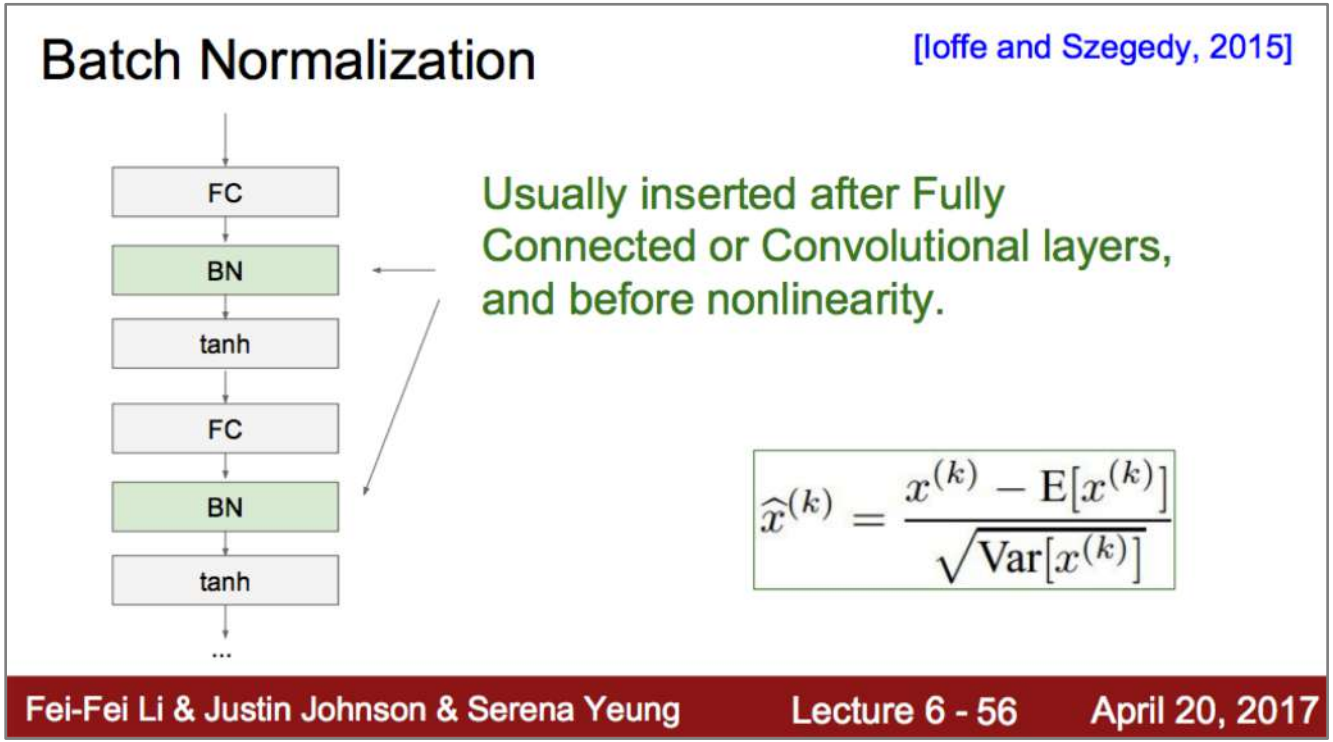
1. compute the empirical mean and variance independently for each dimension.

2. Normalize

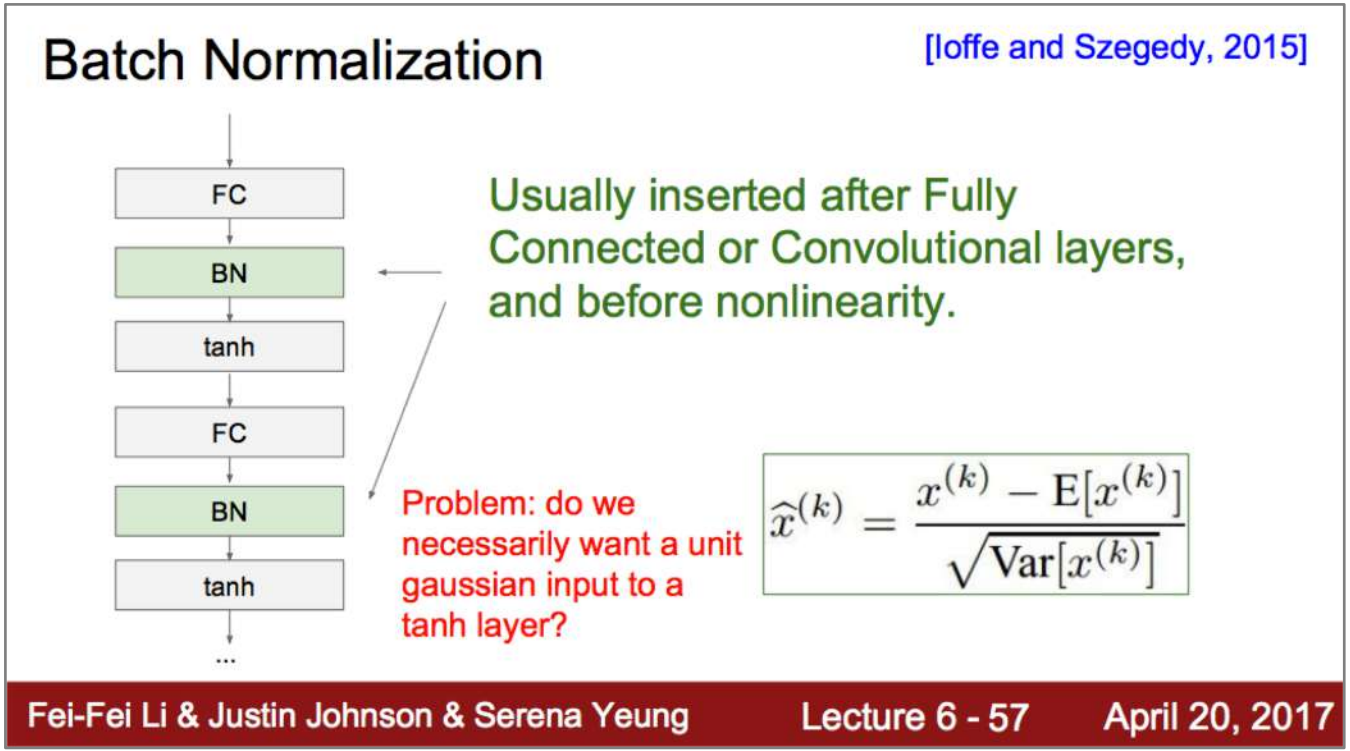
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 55 April 20, 2017

배치 정규화 (Batch Normalization)



배치 정규화 (Batch Normalization)



배치 정규화 (Batch Normalization)

모두의연구소

Batch Normalization

[Ioffe and Szegedy, 2015]

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.

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 6 - 58 April 20, 2017

배치 정규화 (Batch Normalization)

모두의연구소

Batch Normalization

[Ioffe and Szegedy, 2015]

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

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

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

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

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

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

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

배치 정규화 (Batch Normalization)

모두의연구소

Batch Normalization

[Ioffe and Szegedy, 2015]

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

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

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

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

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

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

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

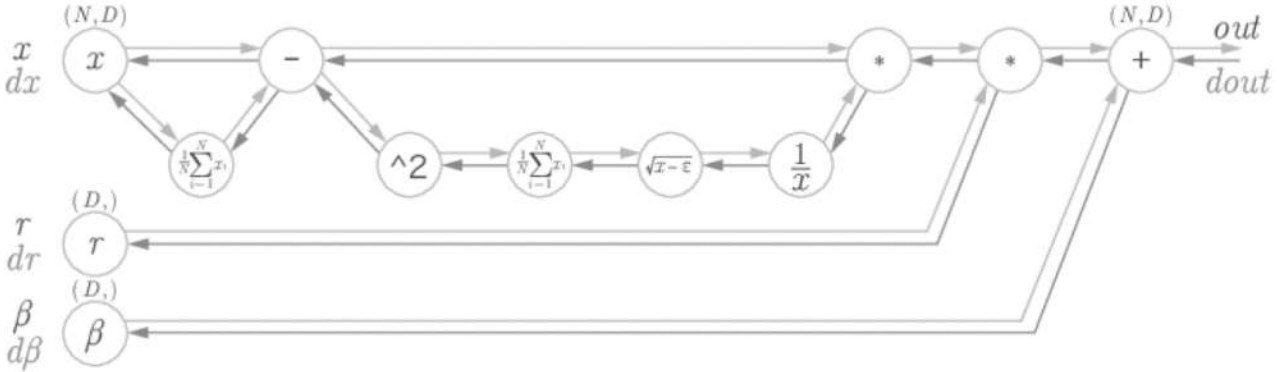
(e.g. can be estimated during training with running averages)

배치 정규화 (Batch Normalization)

모두의연구소

- 학습 시 미니배치를 단위로 데이터 분포가 평균이 0, 분산이 1이되도록 정규화 수행

배치 정규화 계산 그래프

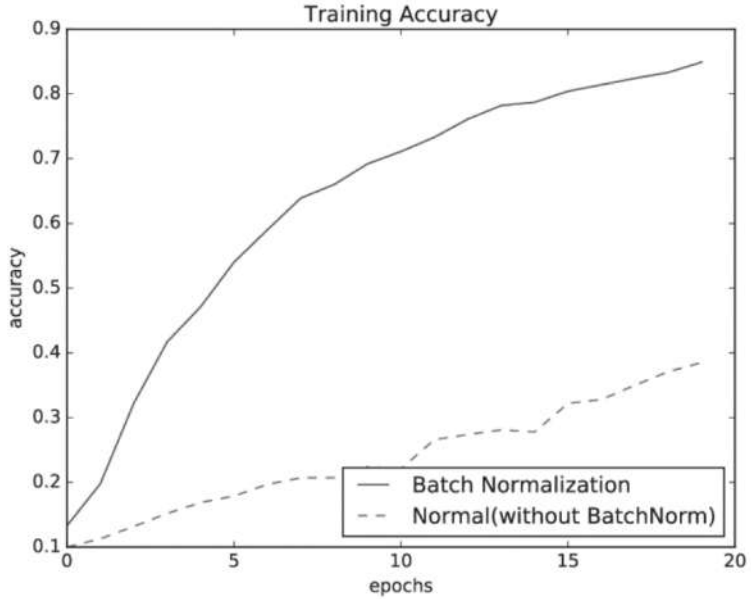


배치 정규화 (Batch Normalization)

모두의연구소

- 학습 시 미니배치를 단위로 데이터 분포가 평균이 0, 분산이 1이되도록 정규화 수행

  배치 정규화 효과 : 배치 정규화가 학습 속도를 높인다

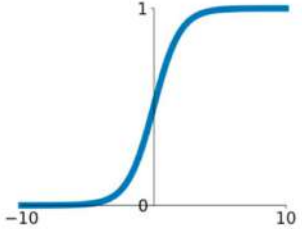


- Batch Normalization
- Leaky RELU
- Transpose Convolution

Activation Functions

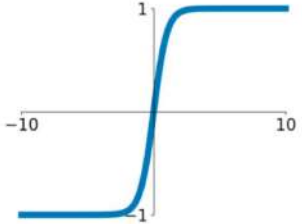
Sigmoid

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



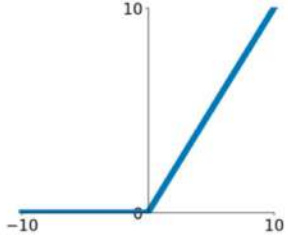
tanh

$$\tanh(x)$$



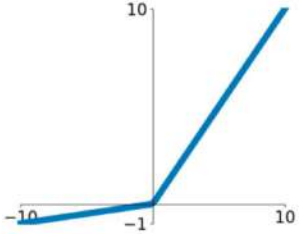
ReLU

$$\max(0, x)$$



Leaky ReLU

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

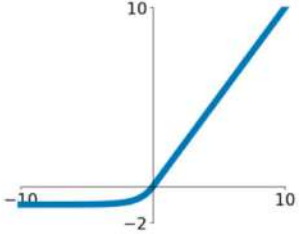


Maxout

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

ELU

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

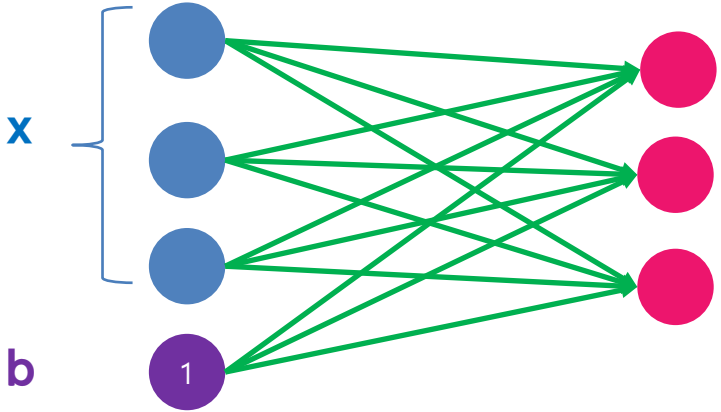


돌아보기 ..

모두의연구소

(3x4) (4x1)

W_1X^T



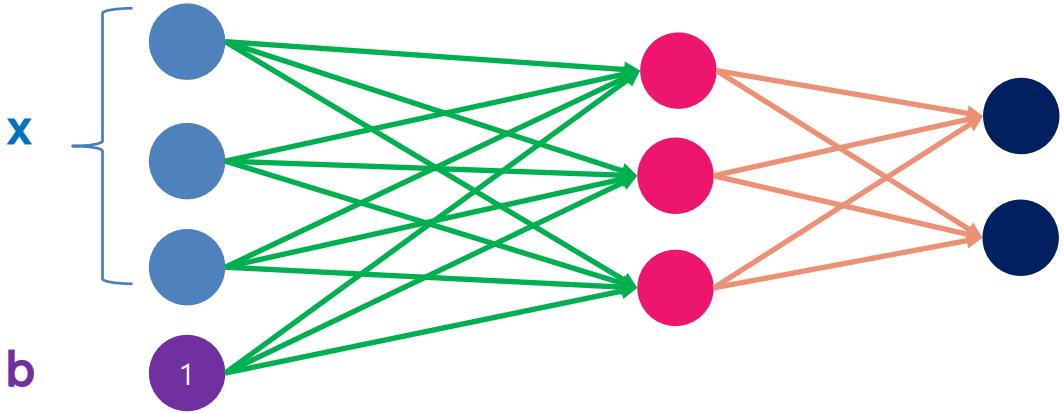
돌아보기 ..

모두의연구소

(2x3) (3x4) (4x1)

W_2 W_1X^T

레이어추가



돌아보기 ..

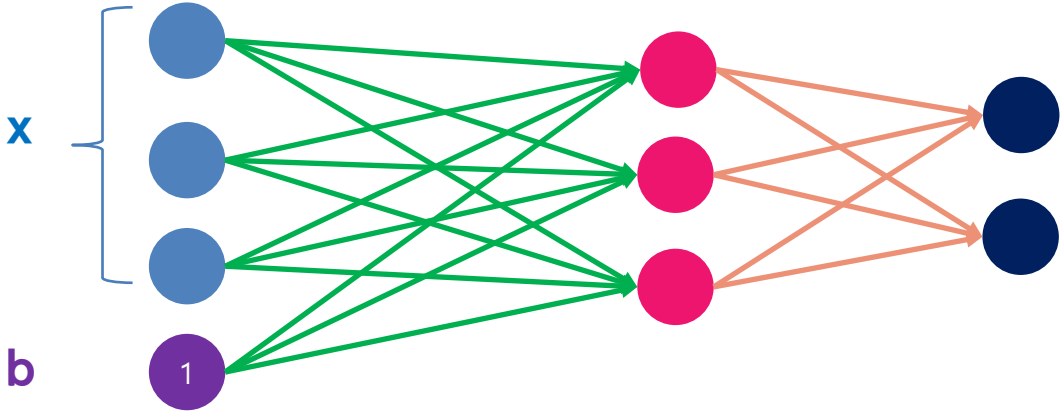
모두의연구소

$$(2\times3)\ (3\times4) \quad (2\times4)$$
$$W_2\ W_1 = W$$

$$\left[\begin{matrix}(2\times3)\ (3\times4)\\ W_2\ W_1\end{matrix}\right]\begin{matrix}(4\times1)\\ X^T\end{matrix}$$

핫 ~

- 두개의 레이어가 하나로 표현
- 레이어를 쌓는 효과가 없어짐

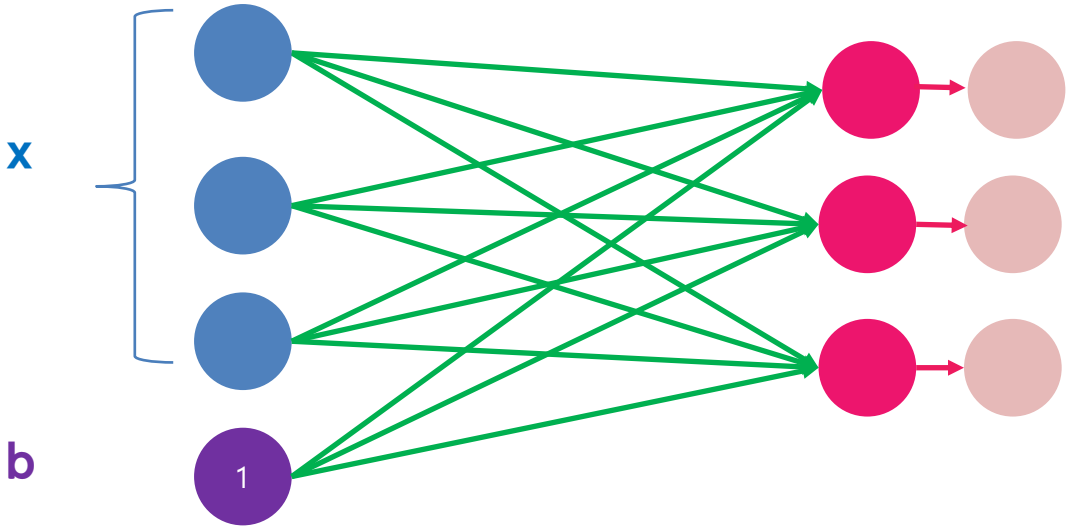


비선형 연산이 필요

돌아보기 ..

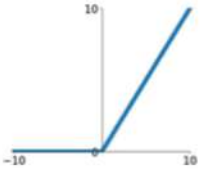
모두의연구소

$$\max(0, W_1 X^T)$$



ReLU

$$\max(0, x)$$

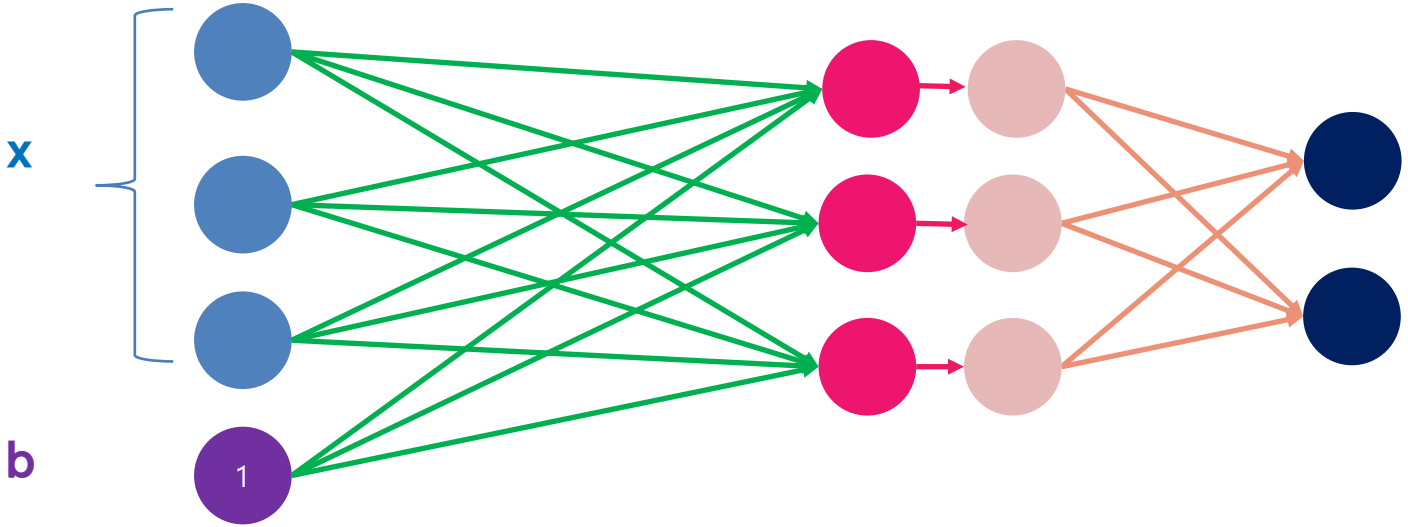


돌아보기 ..

모두의연구소

레이어 추가

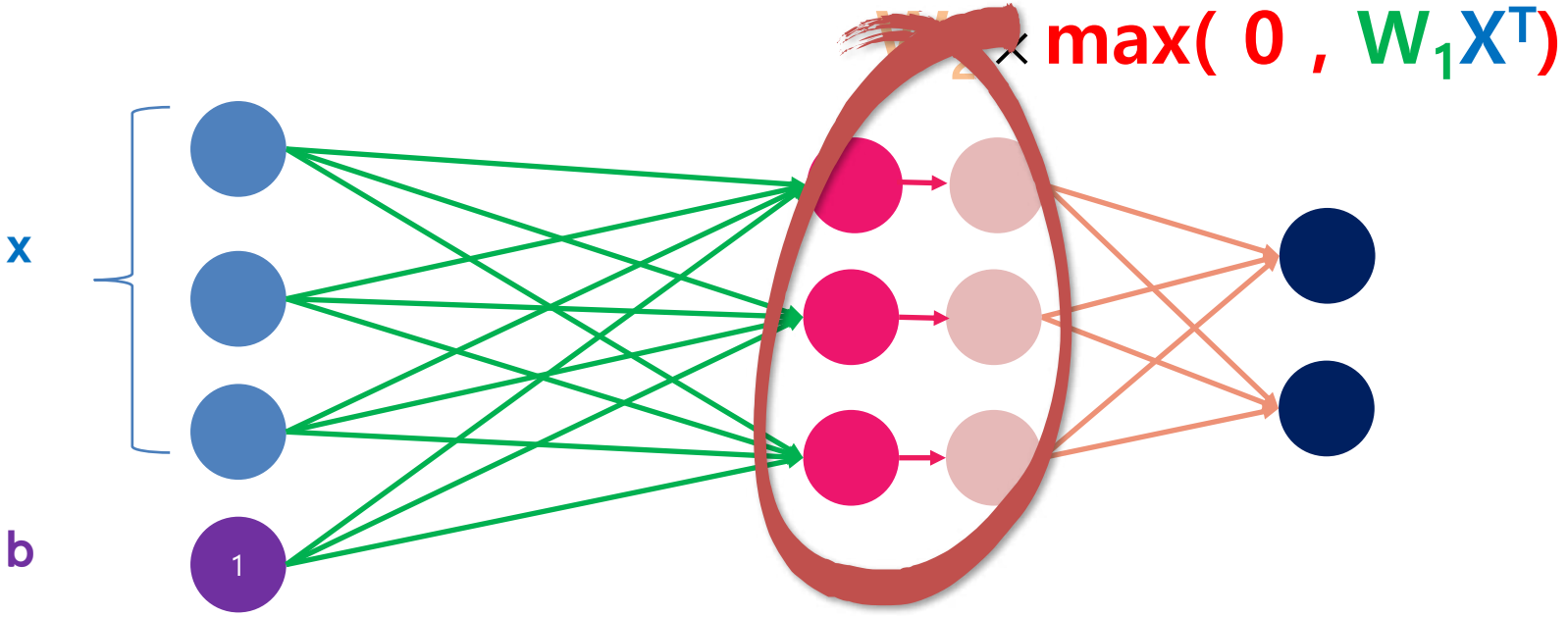
$$W_2 \times \max(0, W_1 X^T)$$



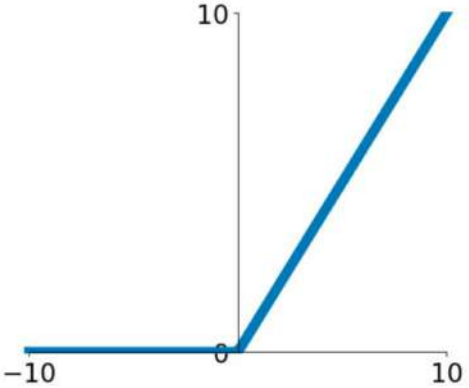
돌아보기 ..

모두의연구소

레이어 추가



Activation Functions



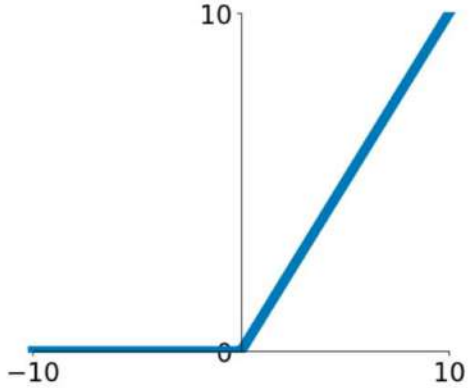
ReLU
(Rectified Linear Unit)

- Computes **f(x) = max(0,x)**

- 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

[Krizhevsky et al., 2012]

Activation Functions



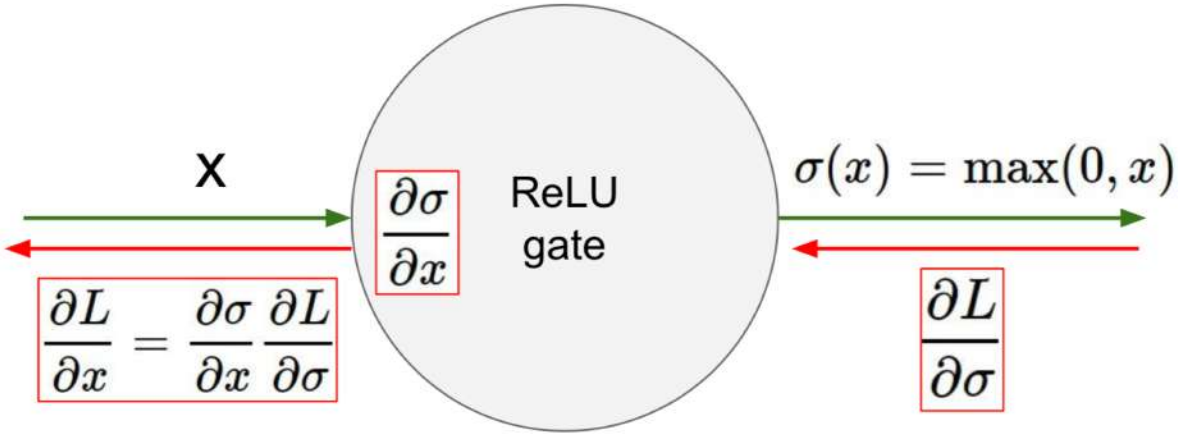
ReLU
(Rectified Linear Unit)

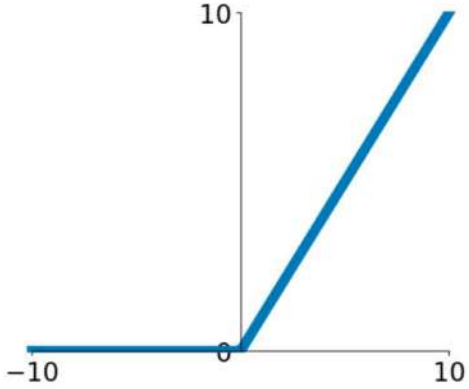
- Computes **f(x) = max(0,x)**

- 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

- Not zero-centered output
- An annoyance:

hint: what is the gradient when x < 0?

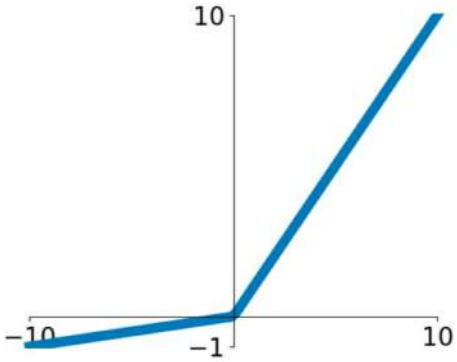




What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

Activation Functions

[Mass et al., 2013]
[He et al., 2015]



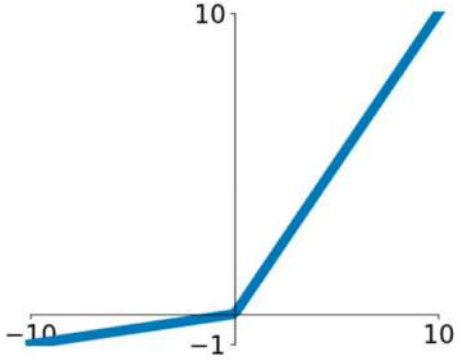
- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not "die".**

Leaky ReLU

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

Activation Functions

[Mass et al., 2013]
[He et al., 2015]



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

Leaky ReLU

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

Parametric Rectifier (PReLU)

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

backprop into \alpha (parameter)

- Batch Normalization
- Leaky RELU
- Transpose Convolution

So far: Image Classification

This image is CC0 public domain

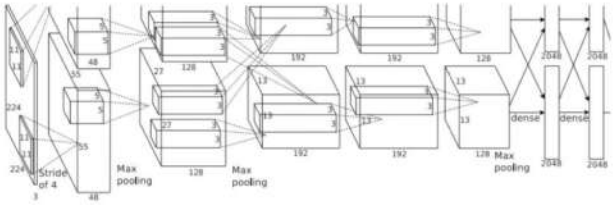


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

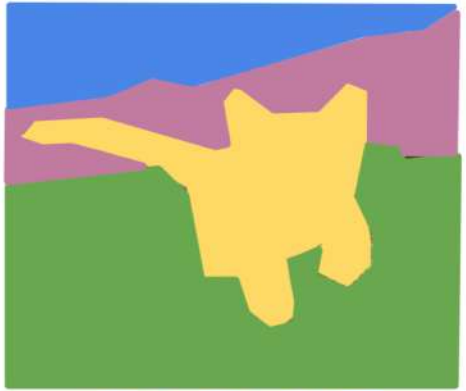
Vector:
4096

Fully-Connected:
4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Other Computer Vision Tasks

Semantic Segmentation



GRASS, CAT, TREE, SKY

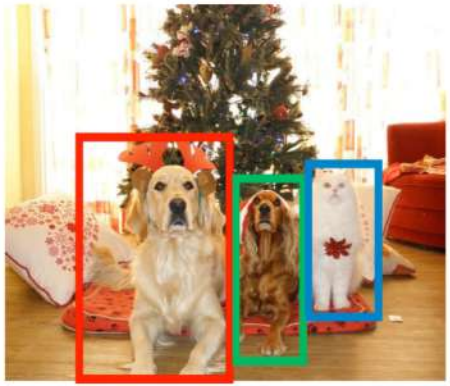
No objects, just pixels

Classification + Localization

CAT

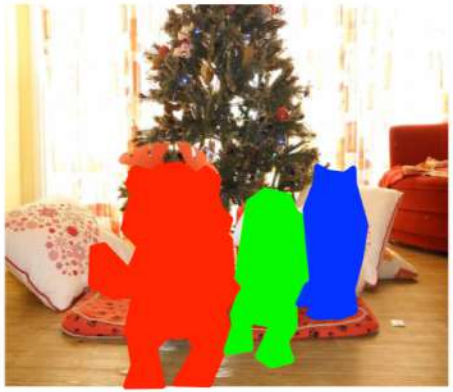
Single Object

Object Detection



DOG, DOG, CAT

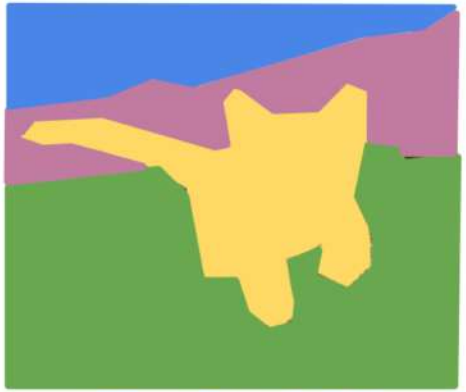
Instance Segmentation



DOG, DOG, CAT

Multiple Object

Semantic Segmentation

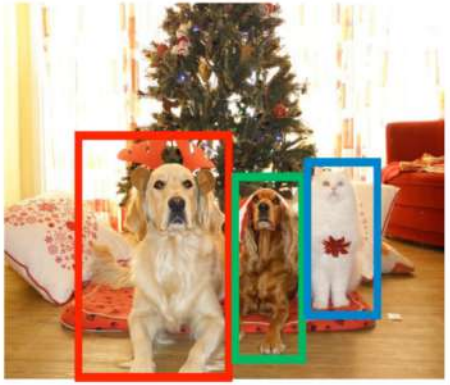


GRASS, CAT, TREE, SKY

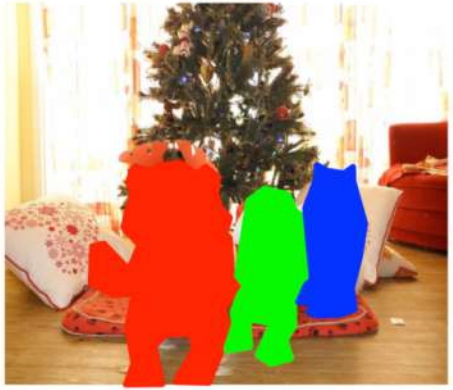
No objects, just pixels

CAT

Single Object



DOG, DOG, CAT



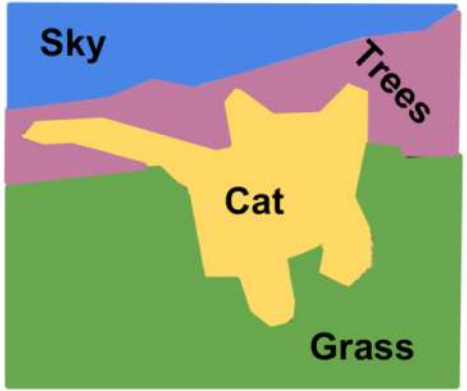
DOG, DOG, CAT

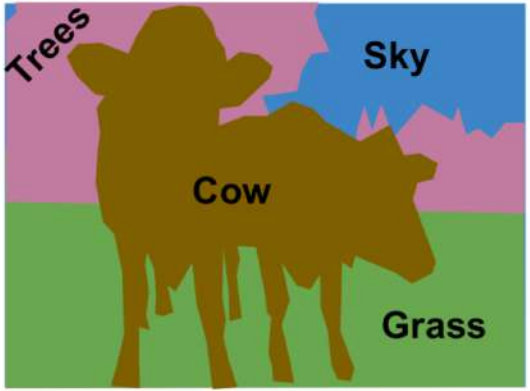
Multiple Object

Semantic Segmentation

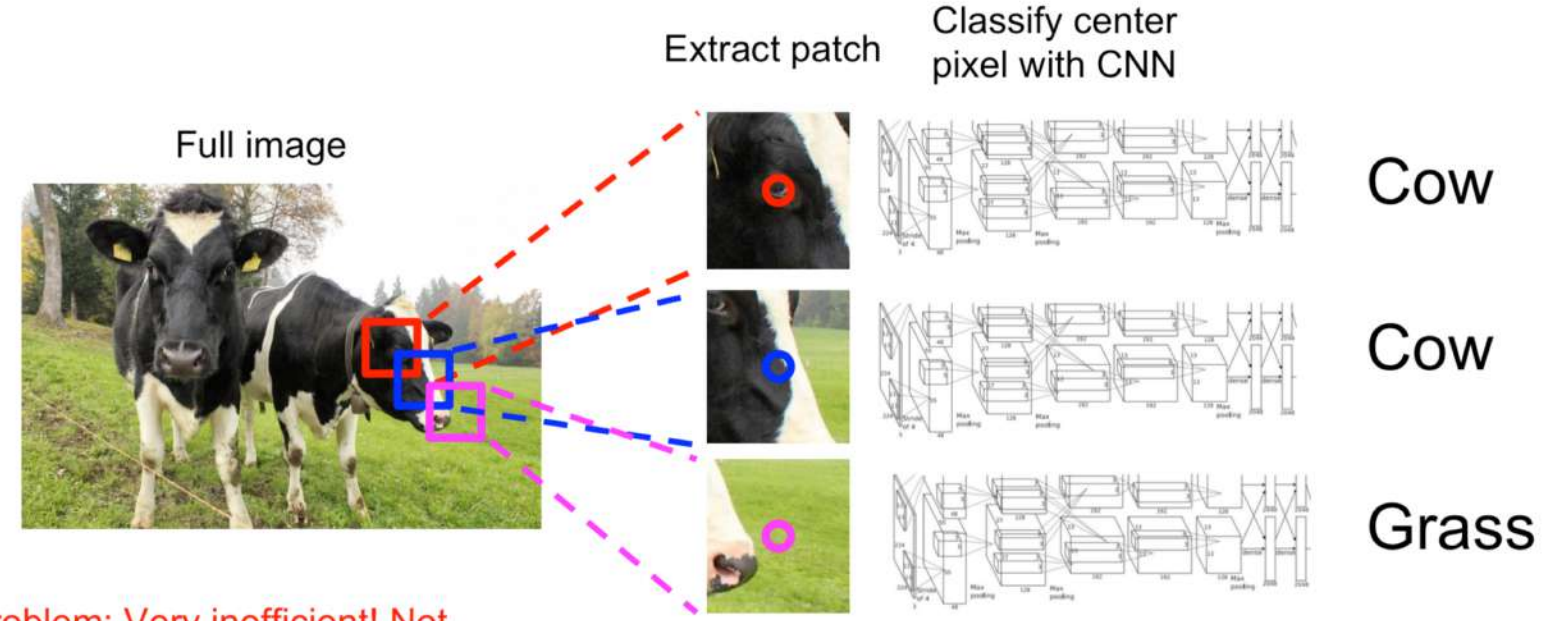
Label each pixel in the image with a category label

Don't differentiate instances, only care about pixels





Semantic Segmentation Idea: Sliding Window



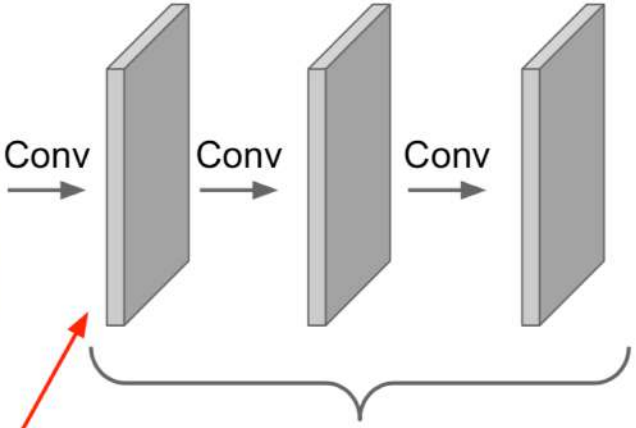
Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

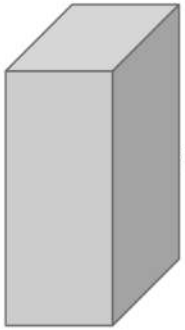
Semantic Segmentation Idea: Fully Convolutional

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!

Input:
3 x H x W



Convolutions:
D x H x W



Scores:
C x H x W

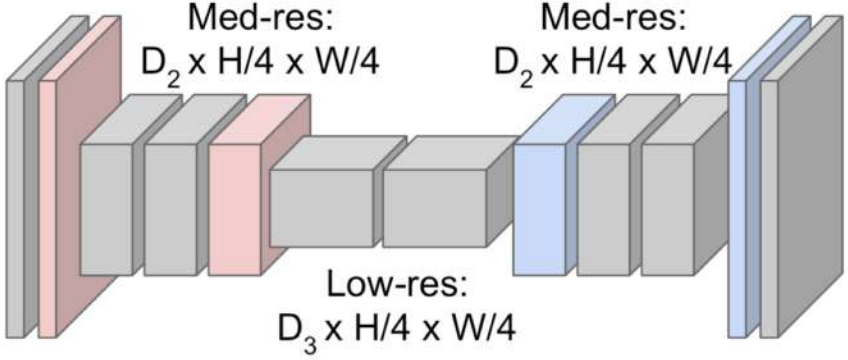
Predictions:
H x W

Problem: convolutions at original image resolution will be very expensive ...

Semantic Segmentation Idea: Fully Convolutional

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2

Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

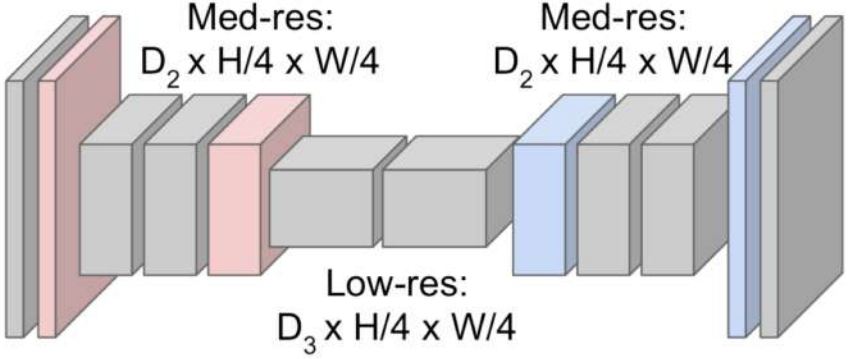
Semantic Segmentation Idea: Fully Convolutional

Downsampling:
Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling:
???

Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

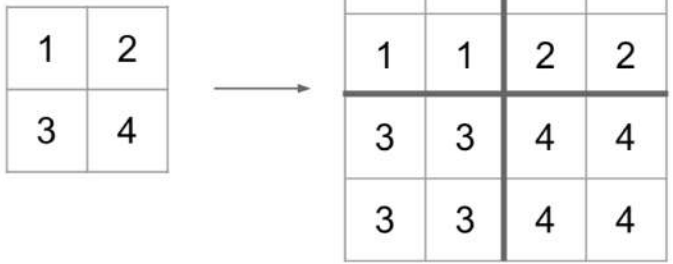
High-res:
D_1 x H/2 x W/2

Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

In-Network upsampling: “Unpooling”

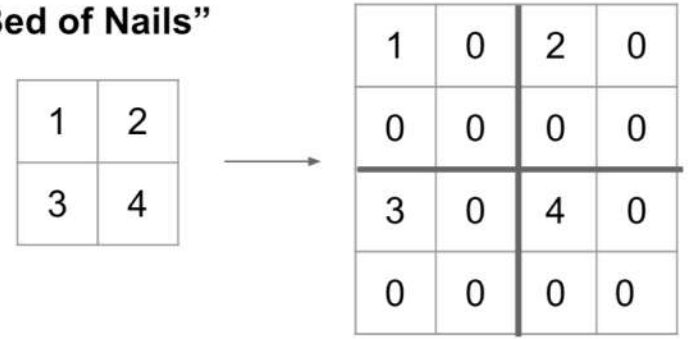
Nearest Neighbor



Input: 2 x 2

Output: 4 x 4

“Bed of Nails”



Input: 2 x 2

Output: 4 x 4

In-Network upsampling: “Max Unpooling”

Max Pooling
Remember which element was max!

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

Input: 4 x 4

5	6
7	8

Output: 2 x 2

Rest of the network

Max Unpooling
Use positions from pooling layer

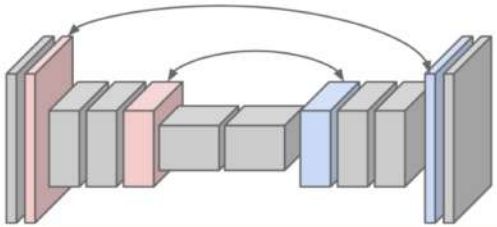
1	2
3	4

Input: 2 x 2

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

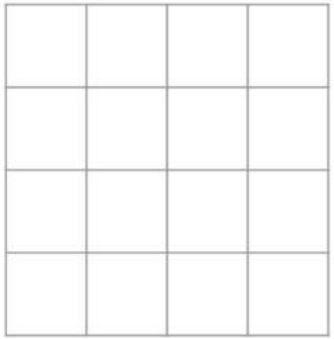
Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers

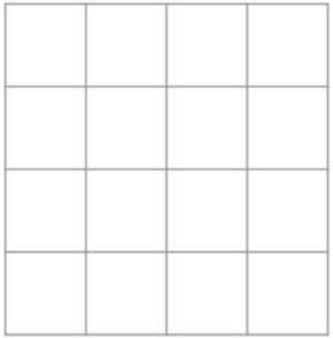


Learnable Upsampling: Transpose Convolution

Recall: Typical 3 x 3 convolution, stride 1 pad 1



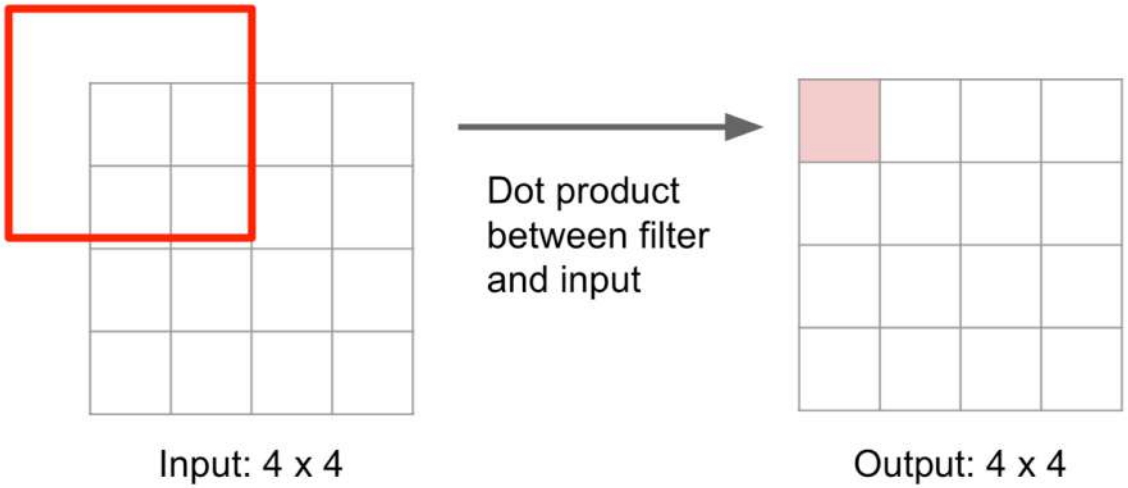
Input: 4 x 4



Output: 4 x 4

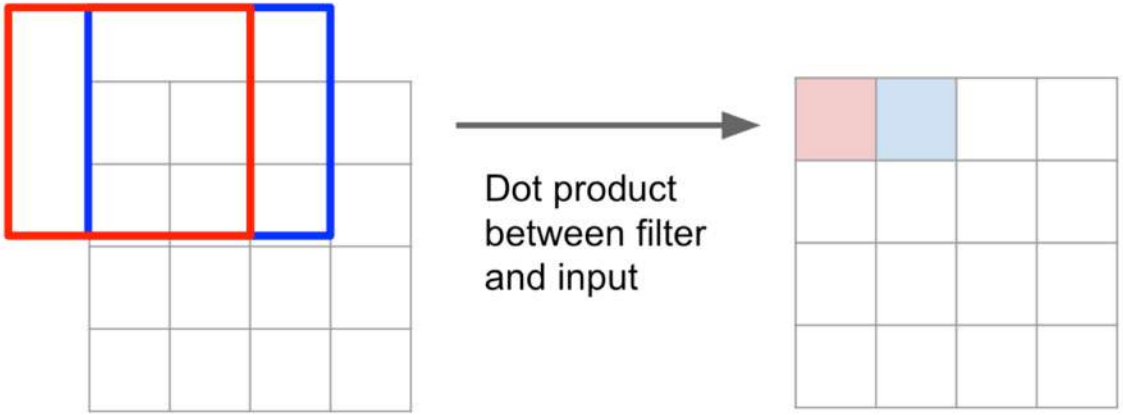
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 1 pad 1



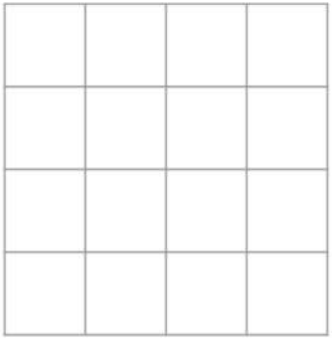
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 1 pad 1

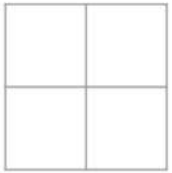


Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



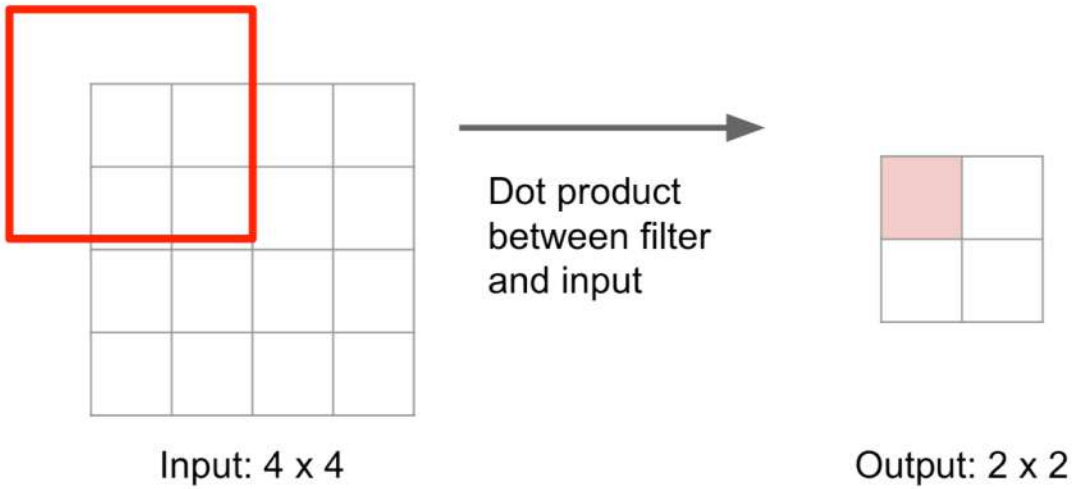
Input: 4 x 4



Output: 2 x 2

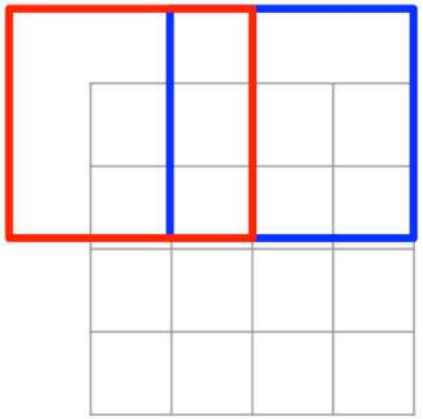
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



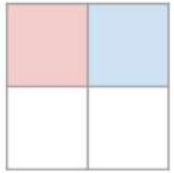
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



Input: 4 x 4

Dot product between filter and input



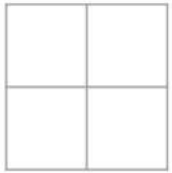
Output: 2 x 2

Filter moves 2 pixels in the input for every one pixel in the output

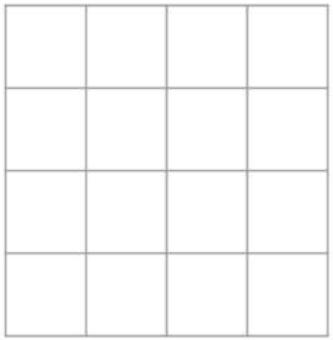
Stride gives ratio between movement in input and output

Learnable Upsampling: Transpose Convolution

3 x 3 **transpose** convolution, stride 2 pad 1



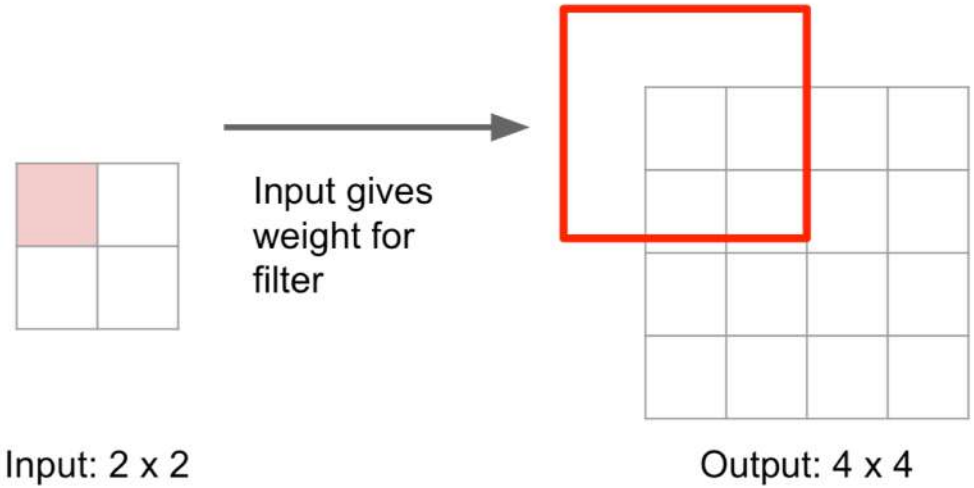
Input: 2 x 2



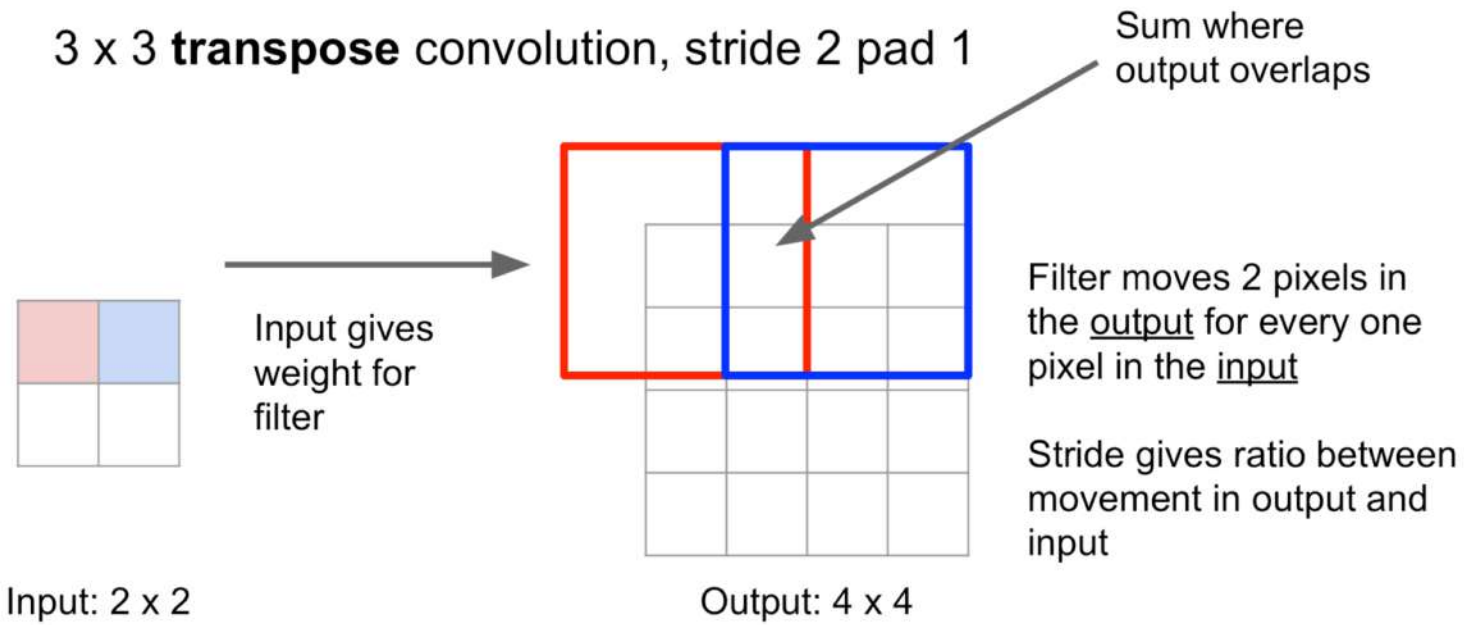
Output: 4 x 4

Learnable Upsampling: Transpose Convolution

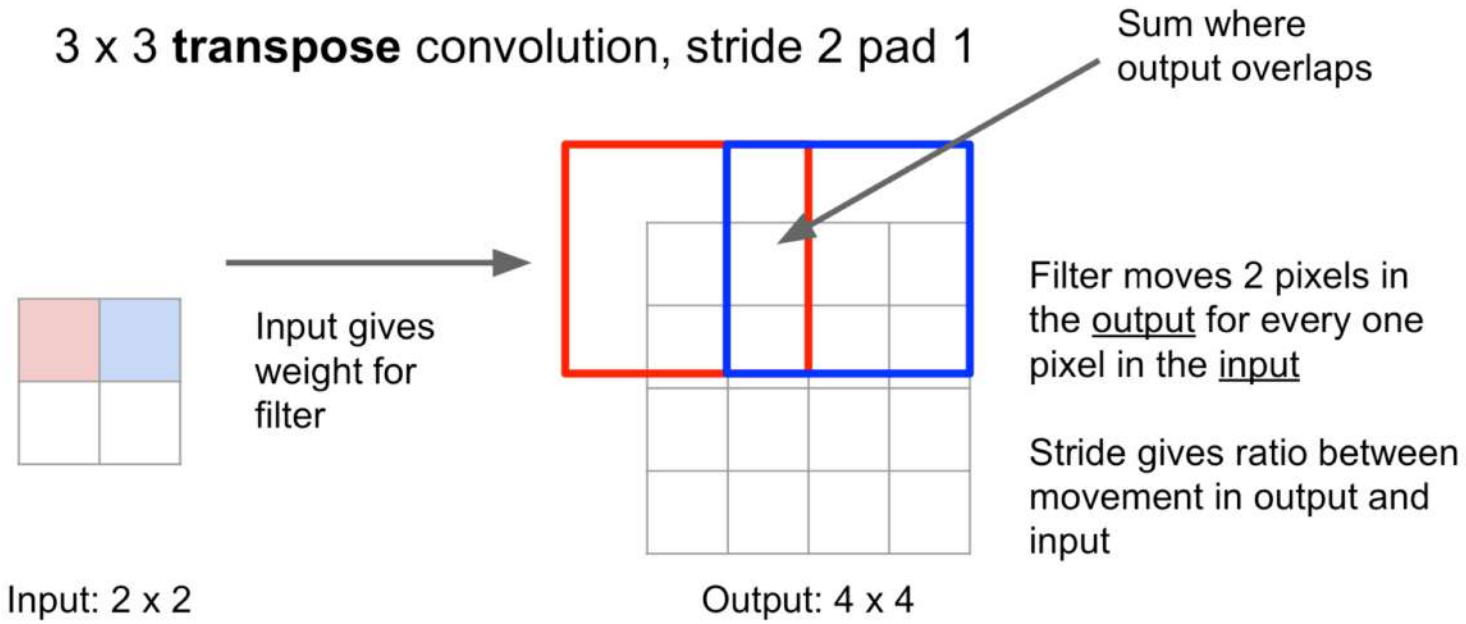
3 x 3 **transpose** convolution, stride 2 pad 1



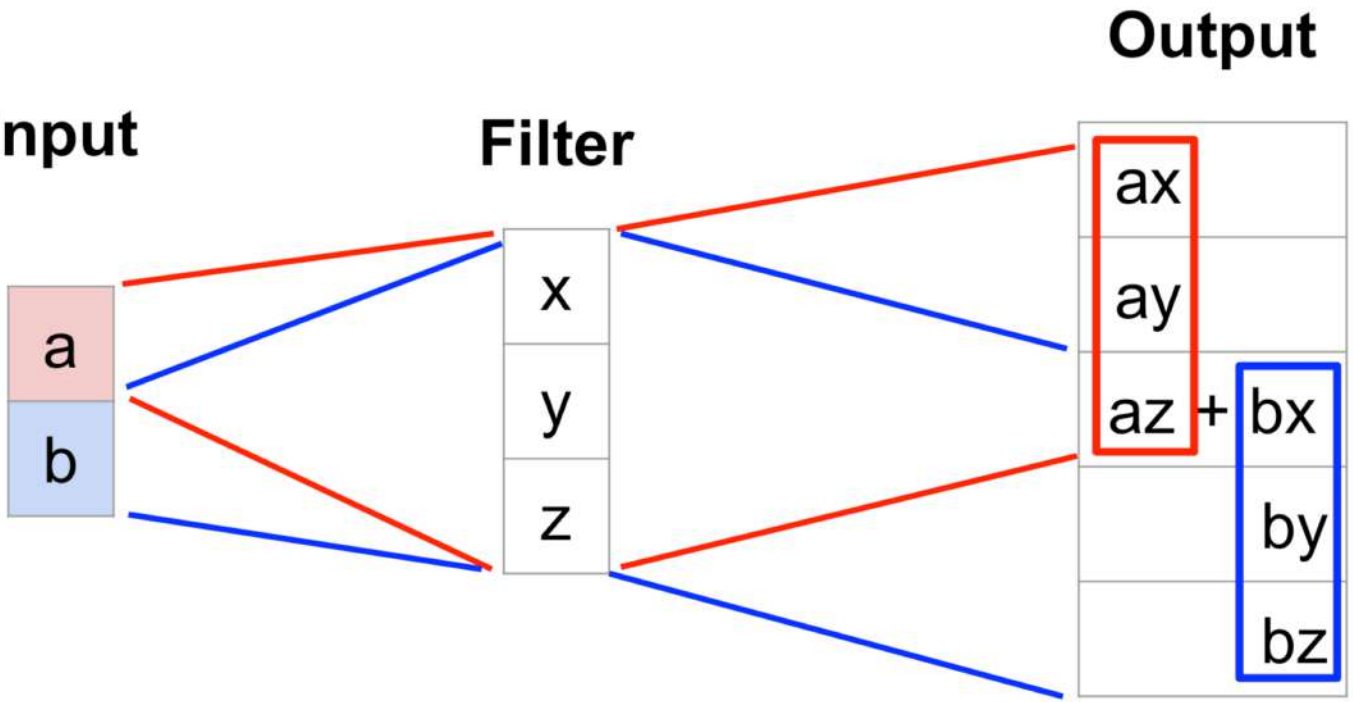
Learnable Upsampling: Transpose Convolution



Learnable Upsampling: Transpose Convolution



Transpose Convolution: 1D Example



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Need to crop one pixel from output to make output exactly 2x input

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & x & y & x & 0 & 0 \\ 0 & 0 & x & y & x & 0 \\ 0 & 0 & 0 & x & y & x \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & x & y & x & 0 & 0 \\ 0 & 0 & x & y & x & 0 \\ 0 & 0 & 0 & x & y & x \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

When stride=1, convolution transpose is just a regular convolution (with different padding rules)

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & 0 & x & y & x & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=2, padding=1

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=2, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

When stride>1, convolution transpose is no longer a normal convolution!

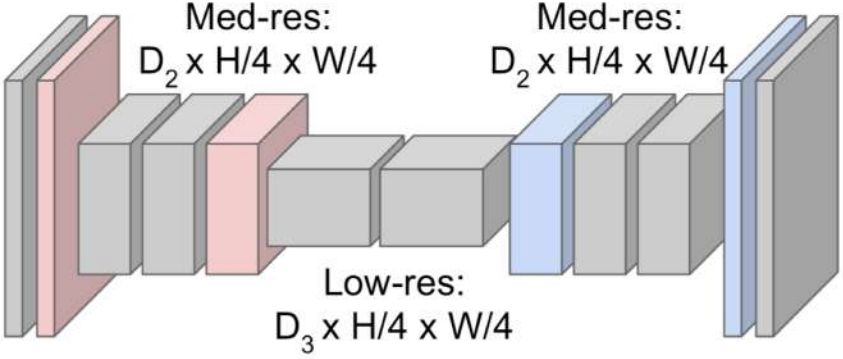
Semantic Segmentation Idea: Fully Convolutional

Downsampling:
Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling:
Unpooling or strided transpose convolution

Input:
3 x H x W



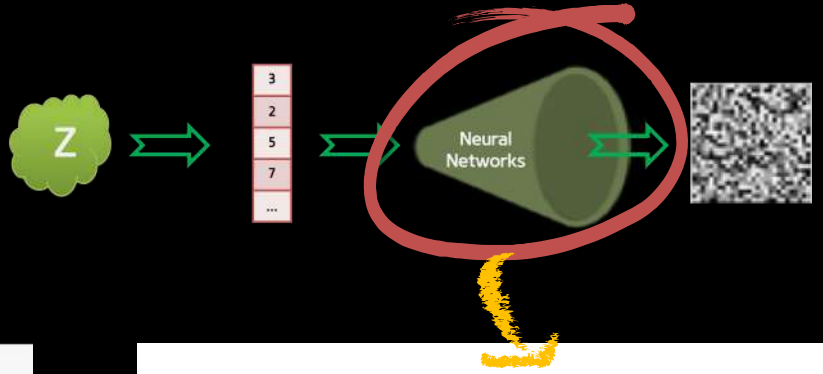
High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2

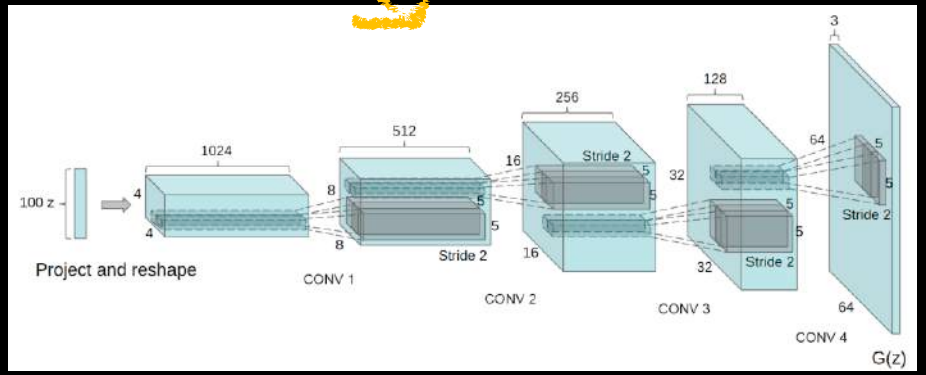
Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Generator 만들기

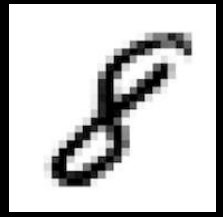


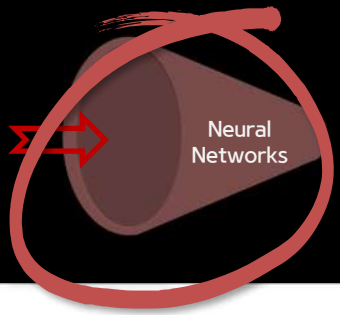

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256) # 주목: 배치사이즈로 Nor

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padd
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), paddi
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), paddin
    assert model.output_shape == (None, 28, 28, 1)

    return model
```



코드랑 dimension 이 다릅니다.

Discriminator



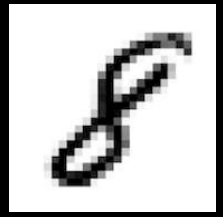


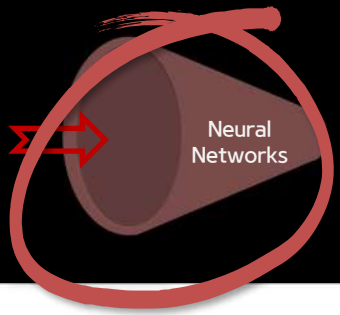
감별자

감별자는 합성곱 신경망(Convolutional Neural Network, CNN) 기반의 이미지 분류기입니다.

```
[ ]  def make_discriminator_model():
         model = tf.keras.Sequential()
         model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                                 input_shape=[28, 28, 1]))
         model.add(layers.LeakyReLU())
         model.add(layers.Dropout(0.3))

         model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
         model.add(layers.LeakyReLU())
         model.add(layers.Dropout(0.3))

         model.add(layers.Flatten())
         model.add(layers.Dense(1))

         return model
```

Discriminator





감별자

감별자는 합성곱 신경망(Convolutional Neural Network, CNN) 기반의 이미지 분류기입니다.

```
[ ]  def make_discriminator_model():
         model = tf.keras.Sequential()
         model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                                 input_shape=[28, 28, 1]))
         model.add(layers.LeakyReLU())
         model.add(layers.Dropout(0.3))

         model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
         model.add(layers.LeakyReLU())
         model.add(layers.Dropout(0.3))

         model.add(layers.Flatten())
         model.add(layers.Dense(1))

         return model
```

안 배운 것

바른 학습을 위해

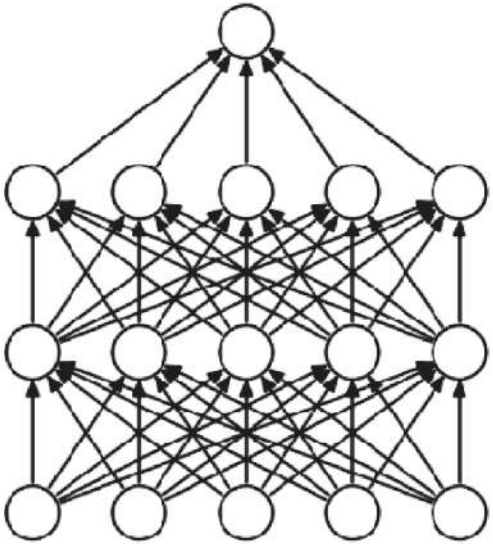
모두의연구소

- 오버피팅 억제 : **드롭아웃** (Dropout)
  - 훈련때 은닉층의 뉴런을 무작위로 골라 삭제하면서 학습하는 방법
  - 방법론
    - 1) 훈련때 데이터를 흘릴 때마다 삭제할 뉴런을 무작위로 선택하고
    - 2) 시험때는 모든 뉴런에 신호를 전달
      - 단, 시험 때는 각 뉴런의 출력에 훈련 때 삭제한 비율을 곱하여 출력

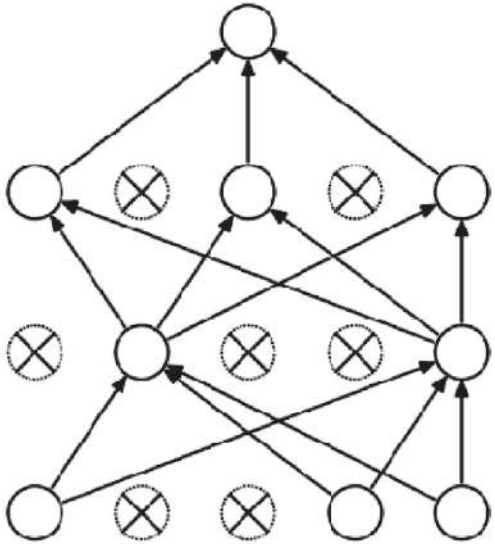
바른 학습을 위해

모두의연구소

- 오버피팅 억제 : **드롭아웃** (Dropout)



(a) 일반 신경망

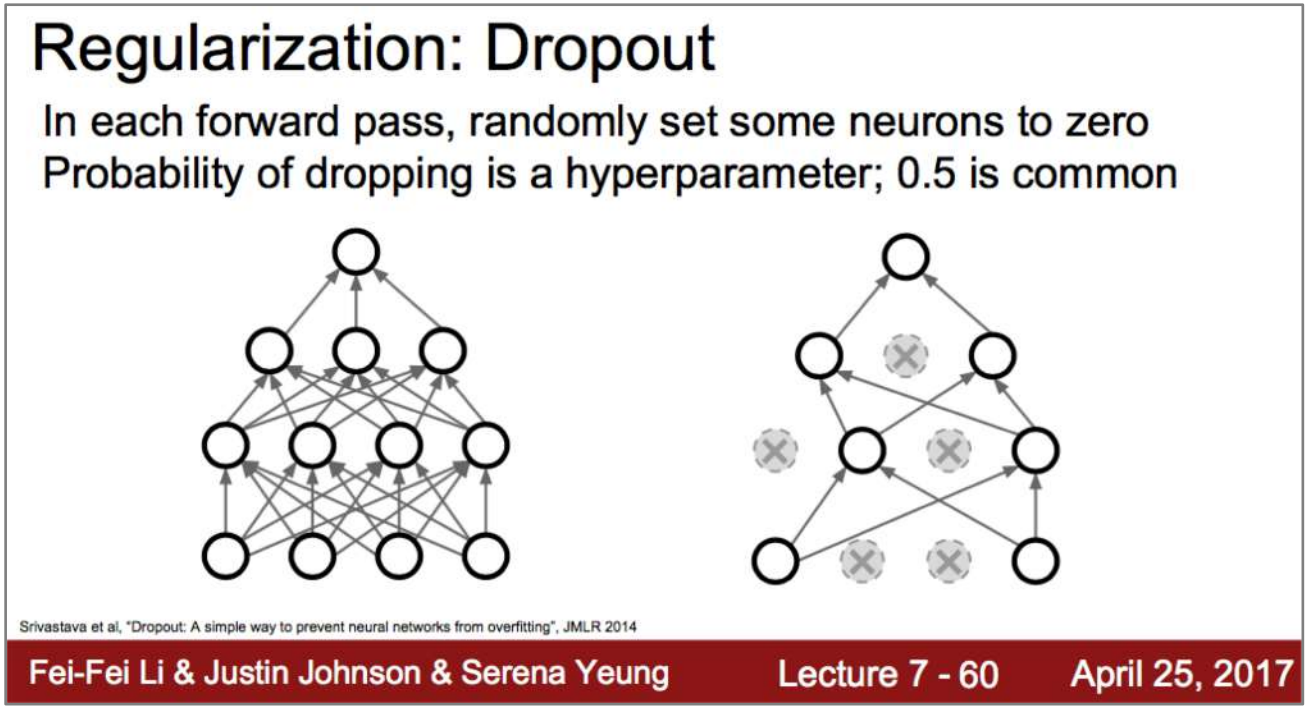


(b) 드롭아웃을 적용한 신경망

바른 학습을 위해

모두의연구소

- 오버피팅 억제 : **드롭아웃** (Dropout)



바른 학습을 위해

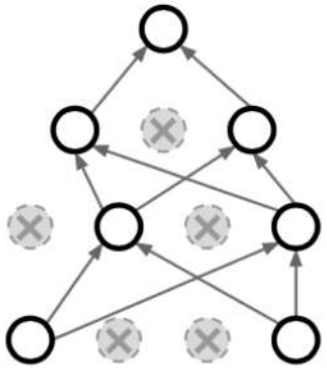
모두의연구소

- 오버피팅 억제 : **드롭아웃** (Dropout)

Regularization: Dropout

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

Example forward pass with a 3-layer network using dropout

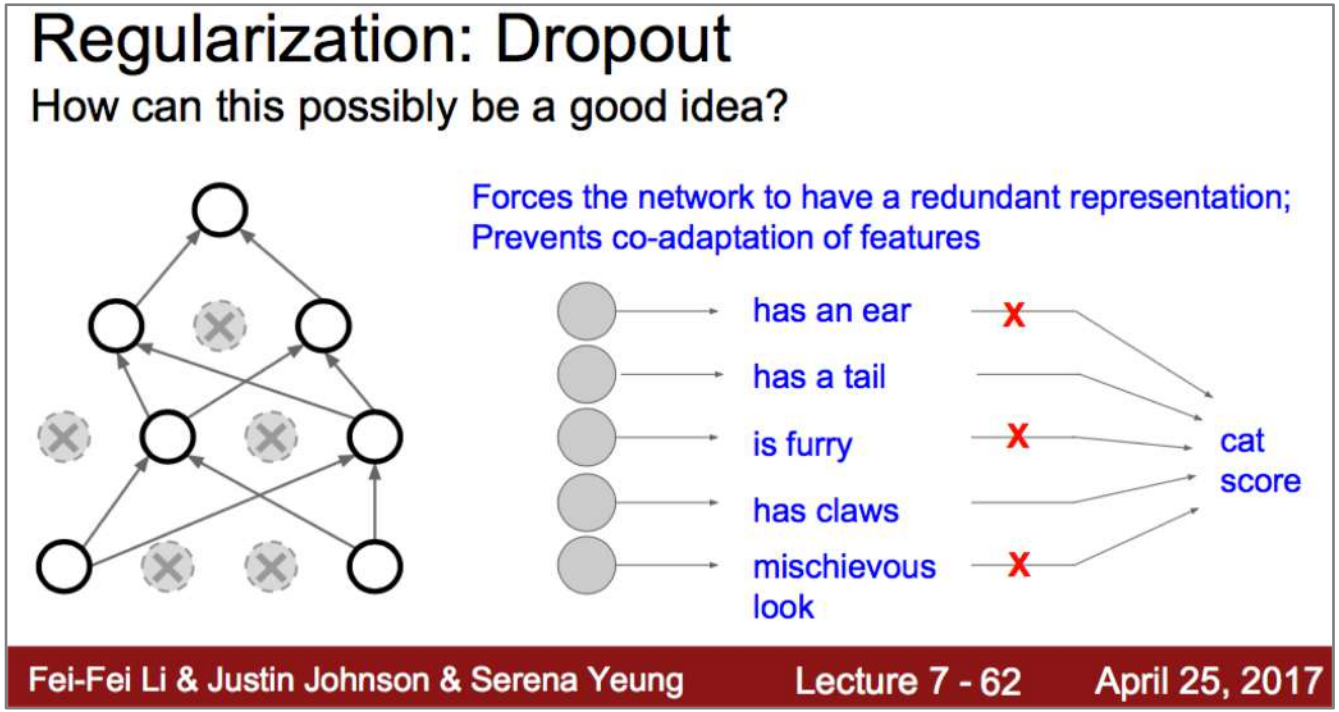


Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 7 - 61 April 25, 2017

바른 학습을 위해

모두의연구소

- 오버피팅 억제 : **드롭아웃** (Dropout)



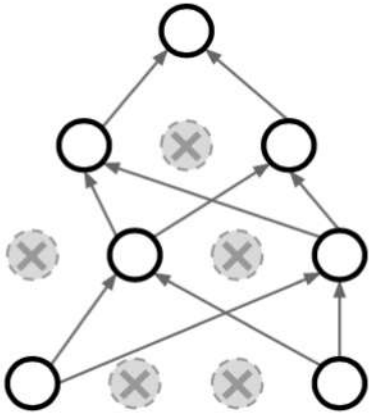
바른 학습을 위해

모두의연구소

- 오버피팅 억제 : **드롭아웃** (Dropout)

Regularization: Dropout

How can this possibly be a good idea?



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

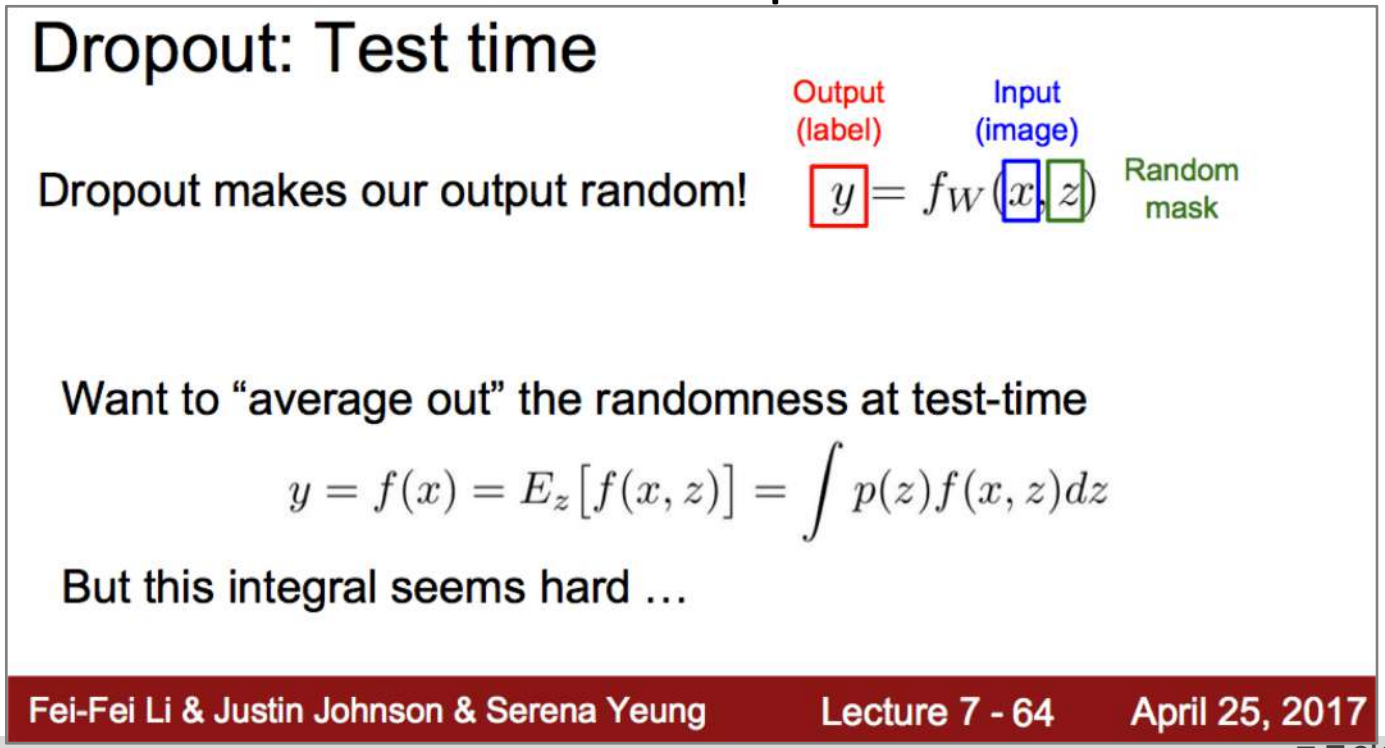
An FC layer with 4096 units has
$2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 7 - 63 April 25, 2017

바른 학습을 위해

모두의연구소

- 오버피팅 억제 : **드롭아웃** (Dropout)



바른 학습을 위해

모두의연구소

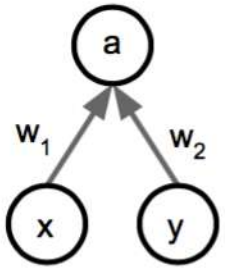
- 오버피팅 억제 : **드롭아웃** (Dropout)

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z[f(x,z)] = \int p(z) f(x,z) dz$$

Consider a single neuron.



Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 7 - 65 April 25, 2017

바른 학습을 위해

모두의연구소

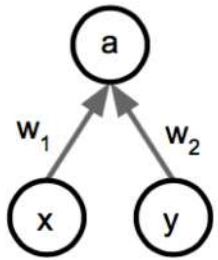
- 오버피팅 억제 : **드롭아웃** (Dropout)

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z[f(x,z)] = \int p(z) f(x,z) dz$$

Consider a single neuron.



At test time we have: $E[a] = w_1 x + w_2 y$

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 7 - 66 April 25, 2017

바른 학습을 위해

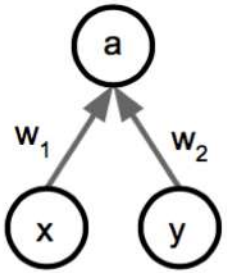
모두의연구소

- 오버피팅 억제 : **드롭아웃** (Dropout)

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$



Consider a single neuron.

At test time we have: $E[a] = w_1 x + w_2 y$

During training we have:

$$\begin{aligned} E[a] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 7 - 67 April 25, 2017

바른 학습을 위해

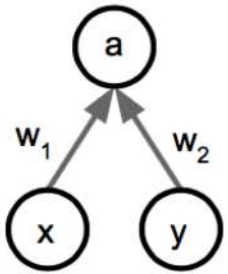
모두의연구소

- 오버피팅 억제 : **드롭아웃** (Dropout)

Dropout: Test time

Want to approximate the integral

$$y = f(x) = E_z[f(x,z)] = \int p(z) f(x,z) dz$$



Consider a single neuron.

At test time we have: $E[a] = w_1 x + w_2 y$

During training we have:

$$E[a] = \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) = \frac{1}{2}(w_1 x + w_2 y)$$

At test time, **multiply** by dropout probability

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 7 - 68 April 25, 2017

바른 학습을 위해

모두의연구소

- 오버피팅 억제 : **드롭아웃** (Dropout)

Dropout: Test time

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

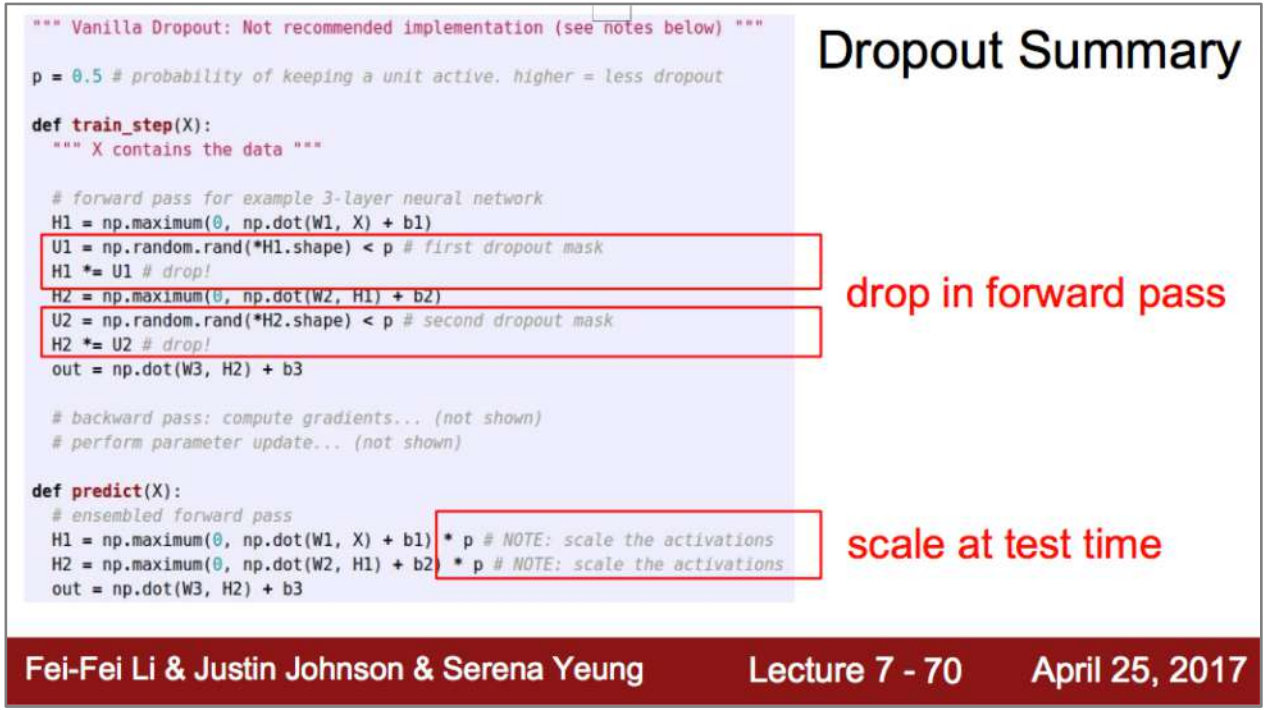
At test time all neurons are active always
=> We must scale the activations so that for each neuron:
output at test time = expected output at training time

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 7 - 69 April 25, 2017

바른 학습을 위해

모두의연구소

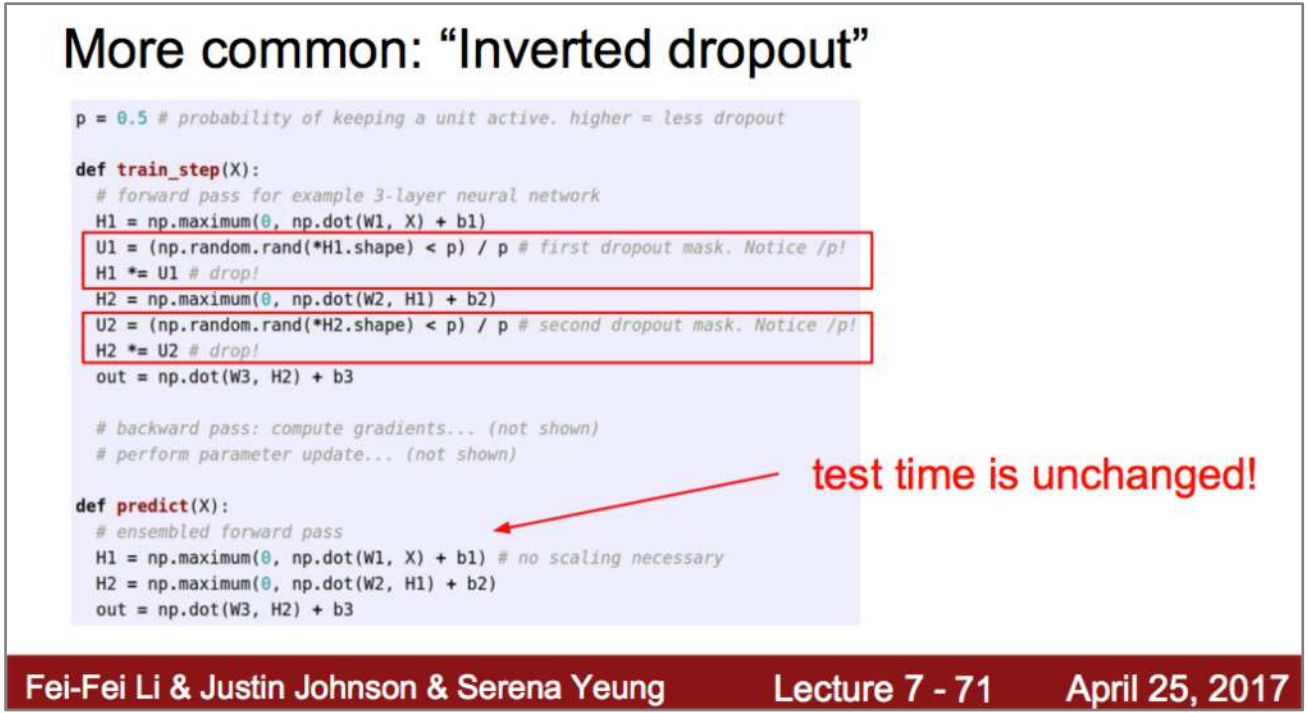
- 오버피팅 억제 : **드롭아웃** (Dropout)



바른 학습을 위해

모두의연구소

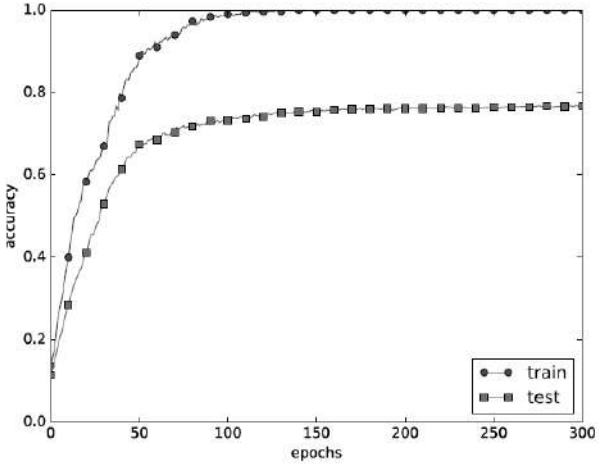
- 오버피팅 억제 : **드롭아웃** (Dropout)



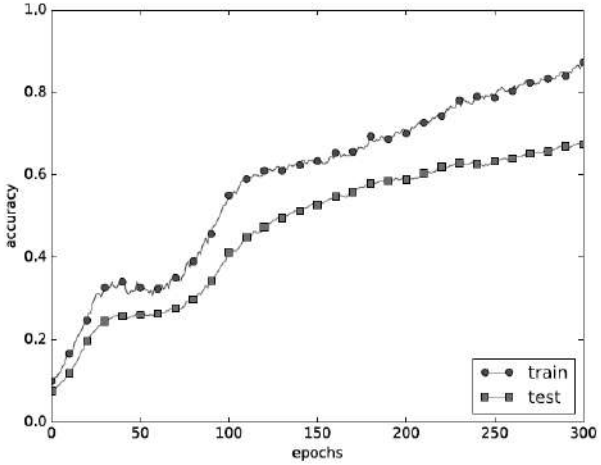
바른 학습을 위해

모두의연구소

- 오버피팅 억제 : **드롭아웃** (Dropout)

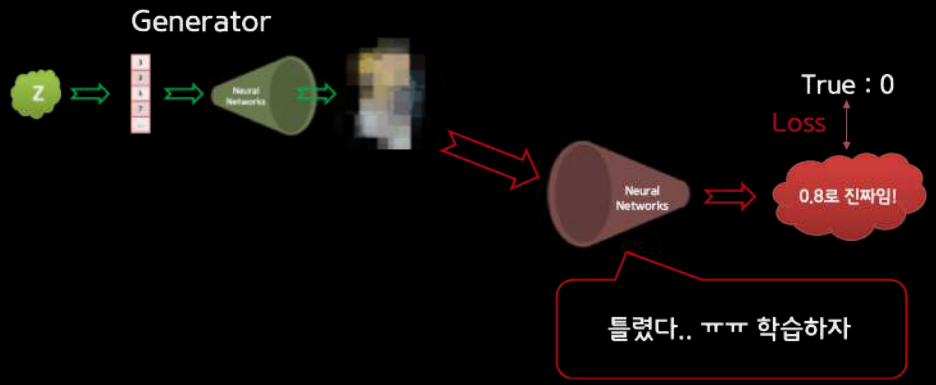


드랍아웃 미적용



드랍아웃 적용

Discriminator Loss



감별자 손실함수

이 메서드는 감별자가 가짜 이미지에서 얼마나 진짜 이미지를 잘 판별하는지 수치화합니다. 진짜 이미지에 대한 감별자의 예측과 1로 이루어진 행렬을 비교하고, 가짜 (생성된) 이미지에 대한 감별자의 예측과 0으로 이루어진 행렬을 비교합니다.

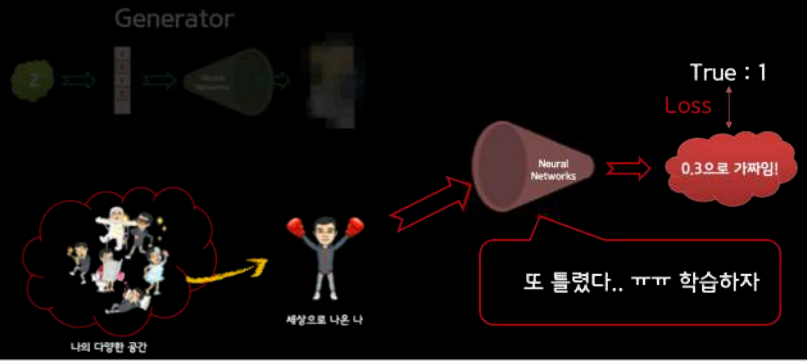
```
[ ]  def discriminator_loss(real_output, fake_output):
         real_loss = cross_entropy(tf.ones_like(real_output), real_output)
         fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
         total_loss = real_loss + fake_loss
         return total_loss
```

생성자 손실함수

생성자의 손실함수는 감별자를 얼마나 잘 속였는지에 대해 수치화를 합니다. 직관적으로 생성자가 원활히 수행되고 있다면, 감별자는 가짜 이미지를 진짜 (또는 1)로 분류를 할 것입니다. 여기서 우리는 생성된 이미지에 대한 감별자의 결정을 1로 이루어진 행렬과 비교를 할 것입니다.

```
[ ]  def generator_loss(fake_output):
         return cross_entropy(tf.ones_like(fake_output), fake_output)
```

Discriminator Loss



감별자 손실함수

이 메서드는 감별자가 가짜 이미지에서 얼마나 진짜 이미지를 잘 판별하는지 수치화합니다. 진짜 이미지에 대한 감별자의 예측과 1로 이루어진 행렬을 비교하고, 가짜 (생성된) 이미지에 대한 감별자의 예측과 0으로 이루어진 행렬을 비교합니다.

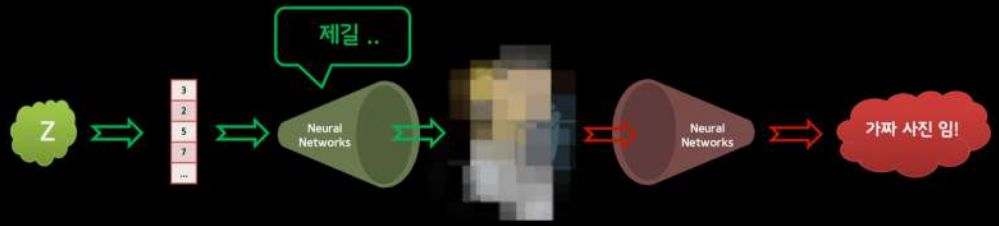
```
[ ]  def discriminator_loss(real_output, fake_output):
         real_loss = cross_entropy(tf.ones_like(real_output), real_output)
         fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
         total_loss = real_loss + fake_loss
         return total_loss
```

생성자 손실함수

생성자의 손실함수는 감별자를 얼마나 잘 속였는지에 대해 수치화를 합니다. 직관적으로 생성자가 원활히 수행되고 있다면, 감별자는 가짜 이미지를 진짜 (또는 1)로 분류를 할 것입니다. 여기서 우리는 생성된 이미지에 대한 감별자의 결정을 1로 이루어진 행렬과 비교를 할 것입니다.

```
[ ]  def generator_loss(fake_output):
         return cross_entropy(tf.ones_like(fake_output), fake_output)
```

Generator Loss



감별자 손실함수

이 메서드는 감별자가 가짜 이미지에서 얼마나 진짜 이미지를 잘 판별하는지 수치화합니다. 진짜 이미지에 대한 감별자의 예측과 1로 이루어진 행렬을 비교하고, 가짜 (생성된) 이미지에 대한 감별자의 예측과 0으로 이루어진 행렬을 비교합니다.

```
[ ]  def discriminator_loss(real_output, fake_output):
         real_loss = cross_entropy(tf.ones_like(real_output), real_output)
         fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
         total_loss = real_loss + fake_loss
         return total_loss
```

정답이 1(True)이 되게 해야 함

생성자 손실함수

생성자의 손실함수는 감별자를 얼마나 잘 속였는지에 대해 수치화를 합니다. 직관적으로 생성자가 원활히 수행되고 있다면, 감별자는 가짜 이미지를 진짜 (또는 1)로 분류를 할 것입니다. 여기서 우리는 생성된 이미지에 대한 감별자의 결정을 1로 이루어진 행렬과 비교를 할 것입니다.

```
[ ]  def generator_loss(fake_output):
         return cross_entropy(tf.ones_like(fake_output), fake_output)
```

기타 학습을 위한 작업 들

감별자와 생성자는 따로 훈련되기 때문에, 감별자와 생성자의 옵티마이저는 다릅니다.

```
[ ]  generator_optimizer = tf.keras.optimizers.Adam(1e-4)
     discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

체크포인트 저장

이 노트북은 오랫동안 진행되는 훈련이 방해되는 경우에 유용하게 쓰일 수 있는 모델의 저장방법과 복구방법을 보여줍니다.

```
[ ]  checkpoint_dir = './training_checkpoints'
     checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
     checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                      discriminator_optimizer=discriminator_optimizer,
                                      generator=generator,
                                      discriminator=discriminator)
```

기타 학습을 위한 작업 들

훈련 루프 정의하기

```
[ ]  EPOCHS = 50
     noise_dim = 100
     num_examples_to_generate = 16

     # 이 시드를 시간이 지나도 재활용하겠습니다.
     # (GIF 애니메이션에서 진전 내용을 시각화하는데 쉽기 때문입니다.)
     seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

훈련 루프는 생성자가 입력으로 랜덤시드를 받는 것으로부터 시작됩니다. 그 시드값을 사용하여 이미지를 생성합니다. 감별자를 사용하여 (훈련 세트에서 갖고온) 진짜 이미지와 (생성자가 생성해낸) 가짜이미지를 분류합니다. 각 모델의 손실을 계산하고, 그래디언트 (gradients)를 사용해 생성자와 감별자를 업데이트합니다.

```
[ ]  # `tf.function`이 어떻게 사용되는지 주목해 주세요.
     # 이 데코레이터는 함수를 "컴파일"합니다.
     @tf.function
     def train_step(images):
         noise = tf.random.normal([BATCH_SIZE, noise_dim])

         with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
           generated_images = generator(noise, training=True)

           real_output = discriminator(images, training=True)
           fake_output = discriminator(generated_images, training=True)

           gen_loss = generator_loss(fake_output)
           disc_loss = discriminator_loss(real_output, fake_output)

         gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
         gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

         generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
         discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variabl
```

다음 주 :

- pix2pix
- Cycle GAN
- Text generation

박 은 수 Research Director

E-mail : es.park@modulabs.co.kr