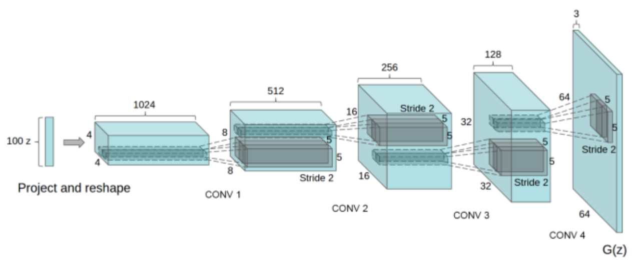


Deep Convolutional Generative Adversarial Networks[1]

CSE 728 Week 03 Presentation

Lectured by Prof. David Doermann

Presented by: Xinguo Zhu

Outline

- Review of GAN
- DCGAN Introduction
- Convolutional/Transposed Layer
- Batch Normalization
- Dropout
- Activation
- DCGAN Revisit
- Demo

GAN

Background

Purpose of GAN:

- Learns P(X), the distribution of training data
- Generate samples from P(X)
- Lots of GAN variations for more specified goal

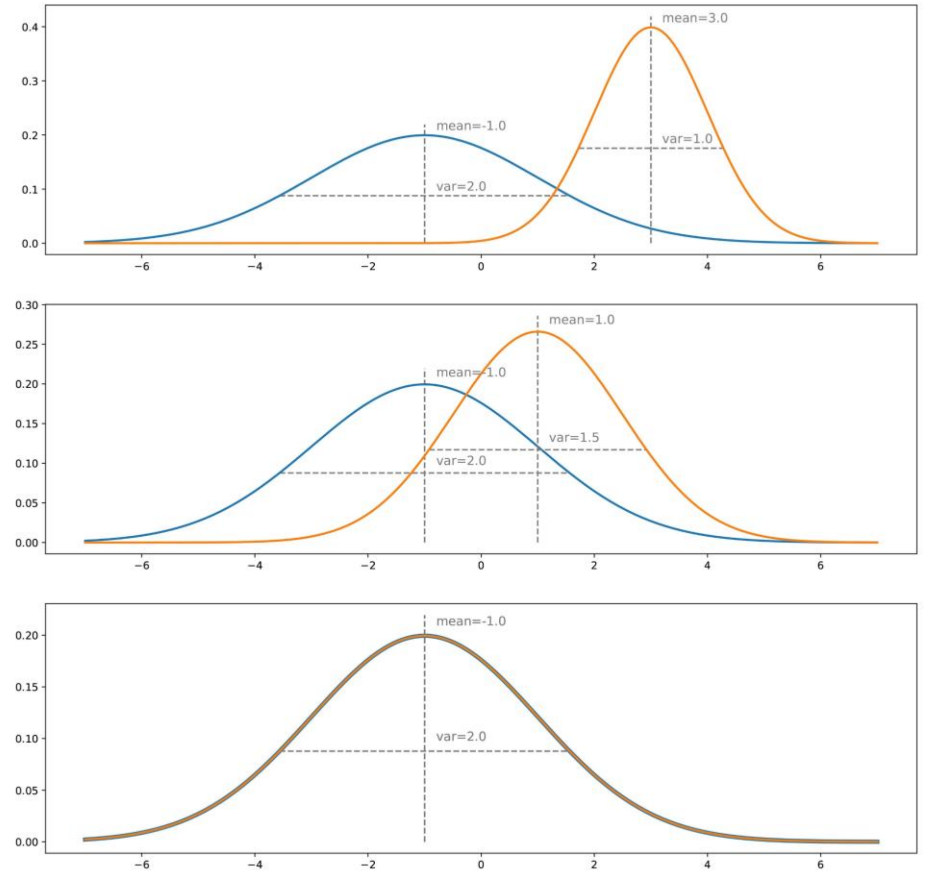


Illustration of GAN training

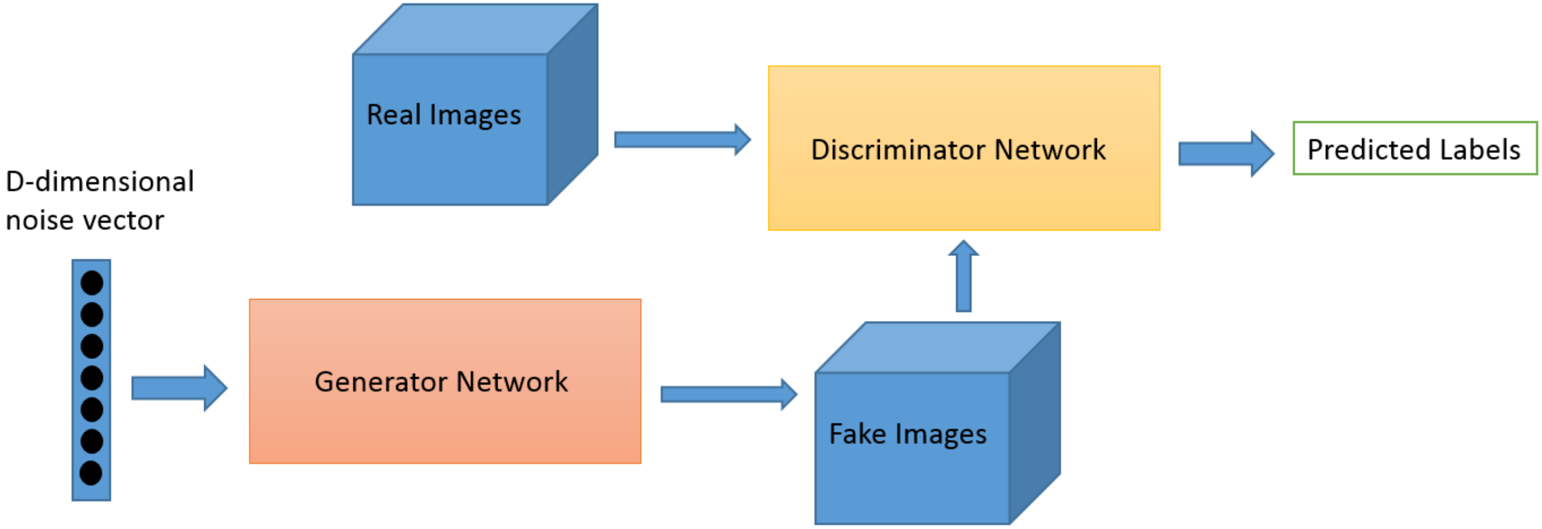
Image from: Joseph Rocca

Min-Max Game

A GAN is defined by the following min-max game

$$\min_G \max_D V(D, G) = \mathbb{E}_X \log D(X) + \mathbb{E}_Z \log(1 - D(G(Z)))$$

- D wants $D(X) = 1$ and $D(G(Z)) = 0$
- G wants $D(G(Z)) = 1$



GAN Architecture

Image from: https://skymind.ai/wiki/generative-adversarial-network-gan

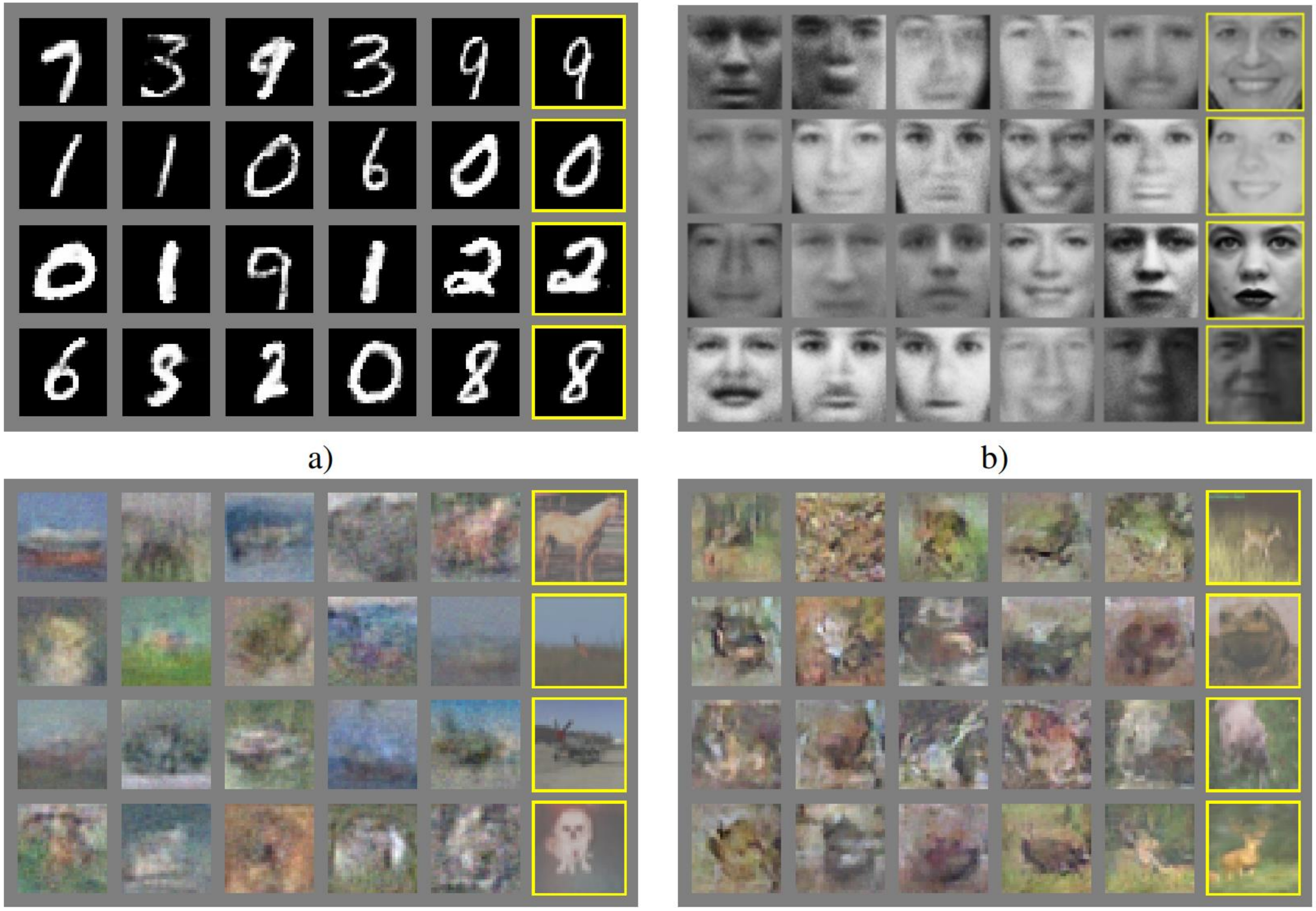
Challenges

Challenges of GAN:

- Generated images are blurry
- Results are noisy and incomprehensible
- Difficult to train
  - Non-convergence
  - Oscillation
  - Mode collapse
  - Gradient Vanish
- No good objective metrics for evaluating

Image from: Joseph Rocca

Blurry Results



a) b) c) d)

Image from: Goodfellow et al., 2014

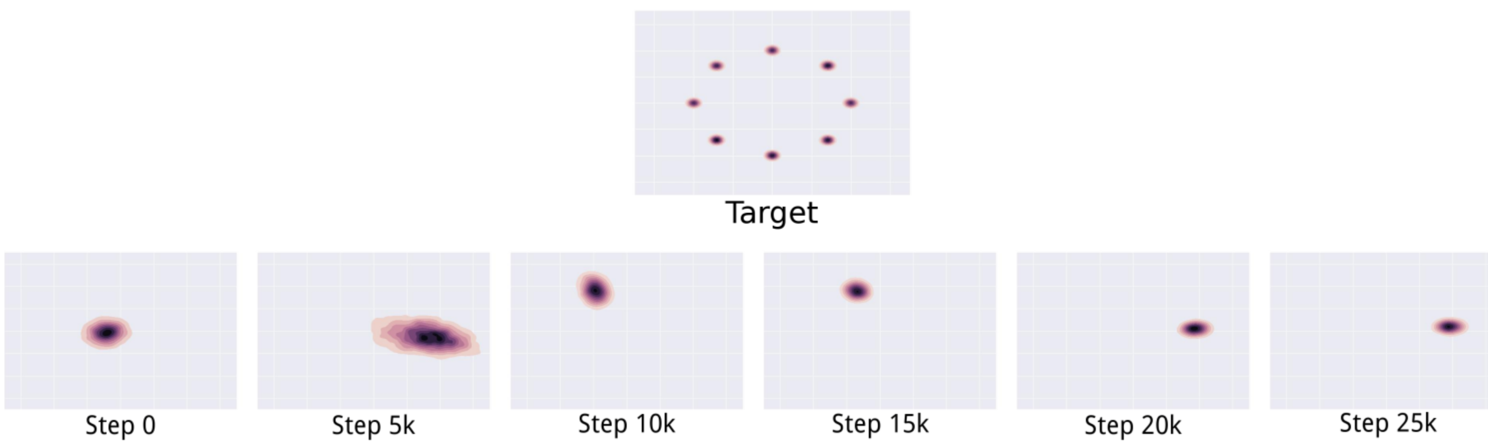
Incomprehensible Results

Image from: Goodfellow et al., 2016

Mode Collapse

$$\min_G \max_D V(G, D) \neq \max_D \min_G V(G, D)$$

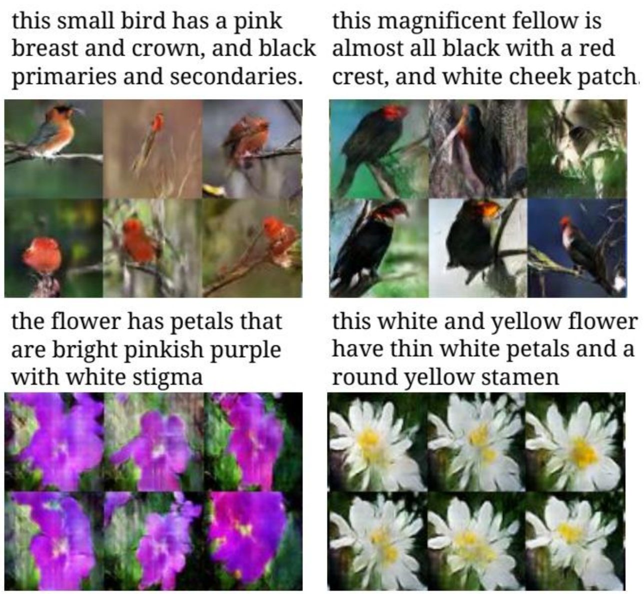
- D in inner loop: convergence to correct distribution
- G in inner loop: place all mass on most likely point



Slide rom: Goodfellow et al., 2016

Mode Collapse

Mode collapse causes low output diversity

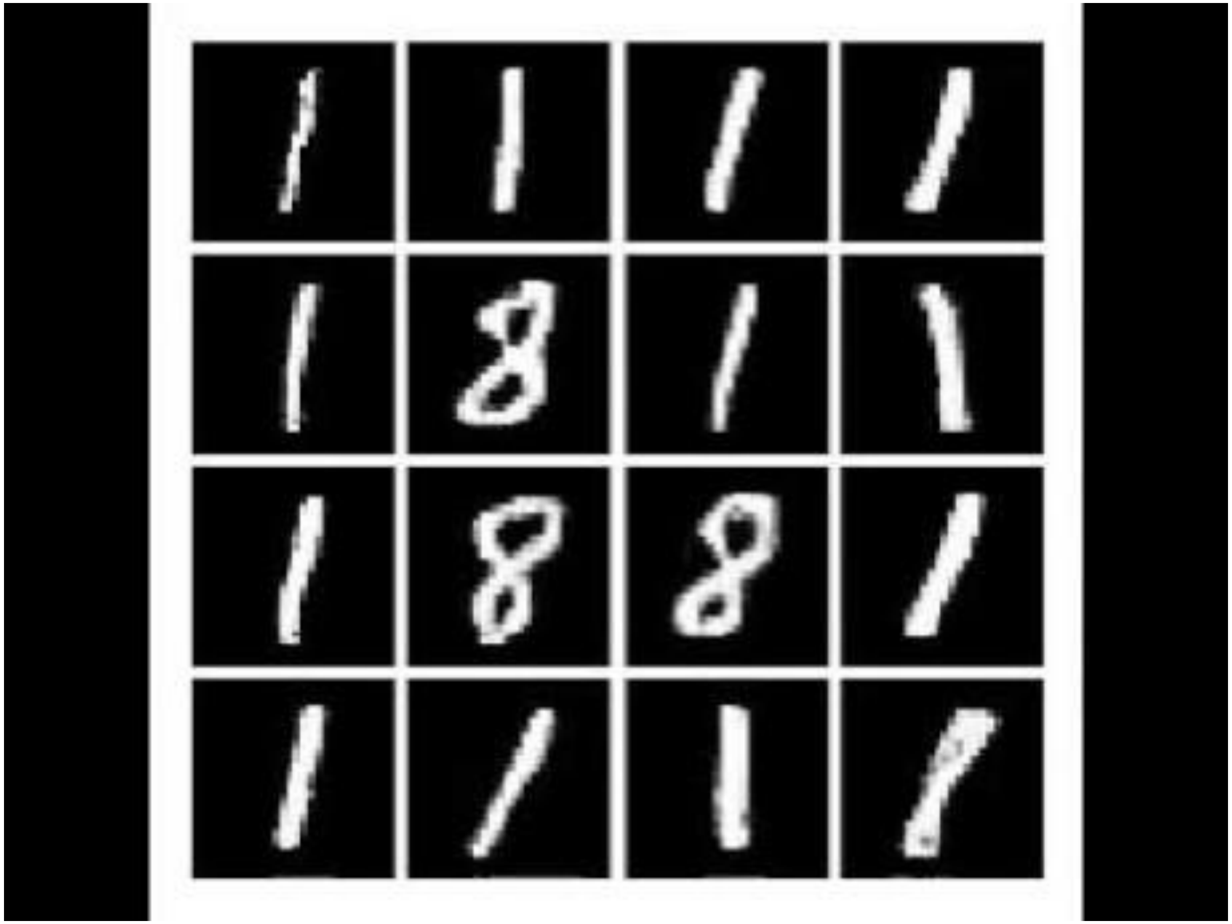


(Reed et al 2016)

(Reed et al, submitted to ICLR 2017)

Slide from: Goodfellow et al., 2016

Mode Collapse



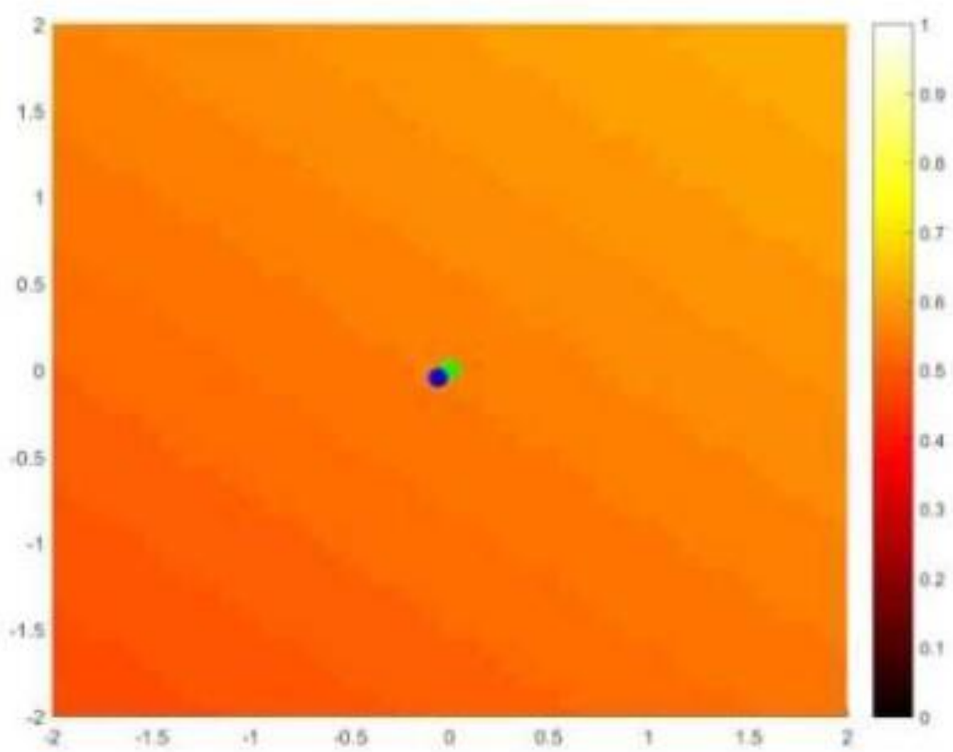
https://www.youtube.com/watch?v=ktxhiKhWoEE

Ref: link

Oscillation

“Oscillation”: can train for a very long time, generating very many different categories of samples, without clearly generating better samples.

-- [NIPS 2016 Tutorial: Generative Adversarial Networks](), 2016.



https://www.youtube.com/watch?v=ebMei6bYeWw

Gradient Vanishment

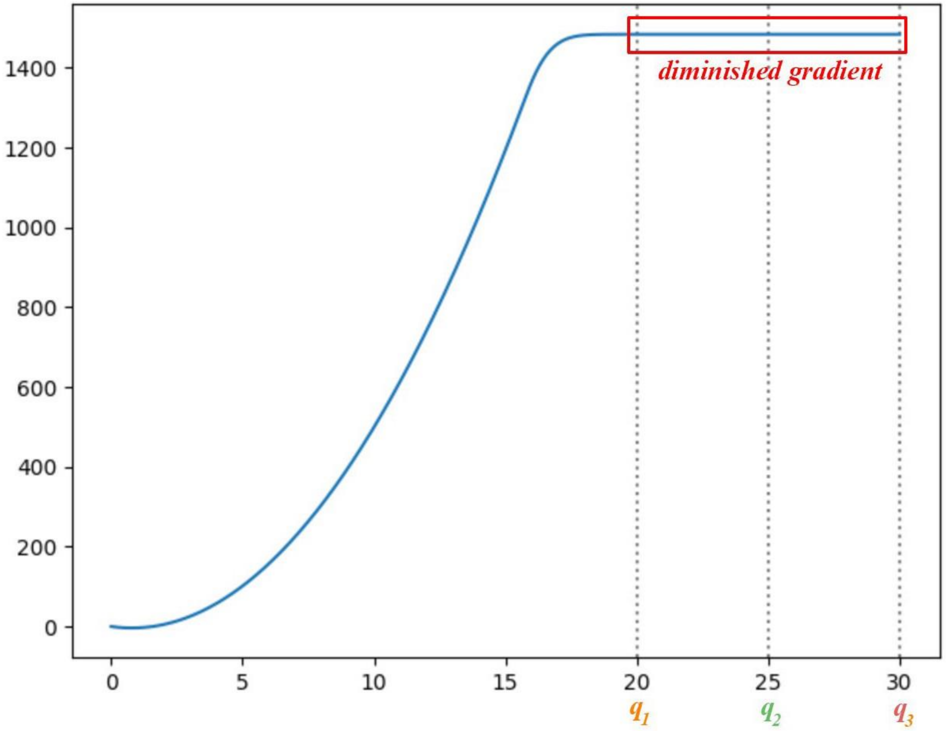


Image: Jonathan Hui

Outline

- Review of GAN
- DCGAN Introduction
- Convolutional/Transposed Layer
- Batch Normalization
- Dropout
- Activation
- DCGAN Revisit
- Demo

DCGAN Intro

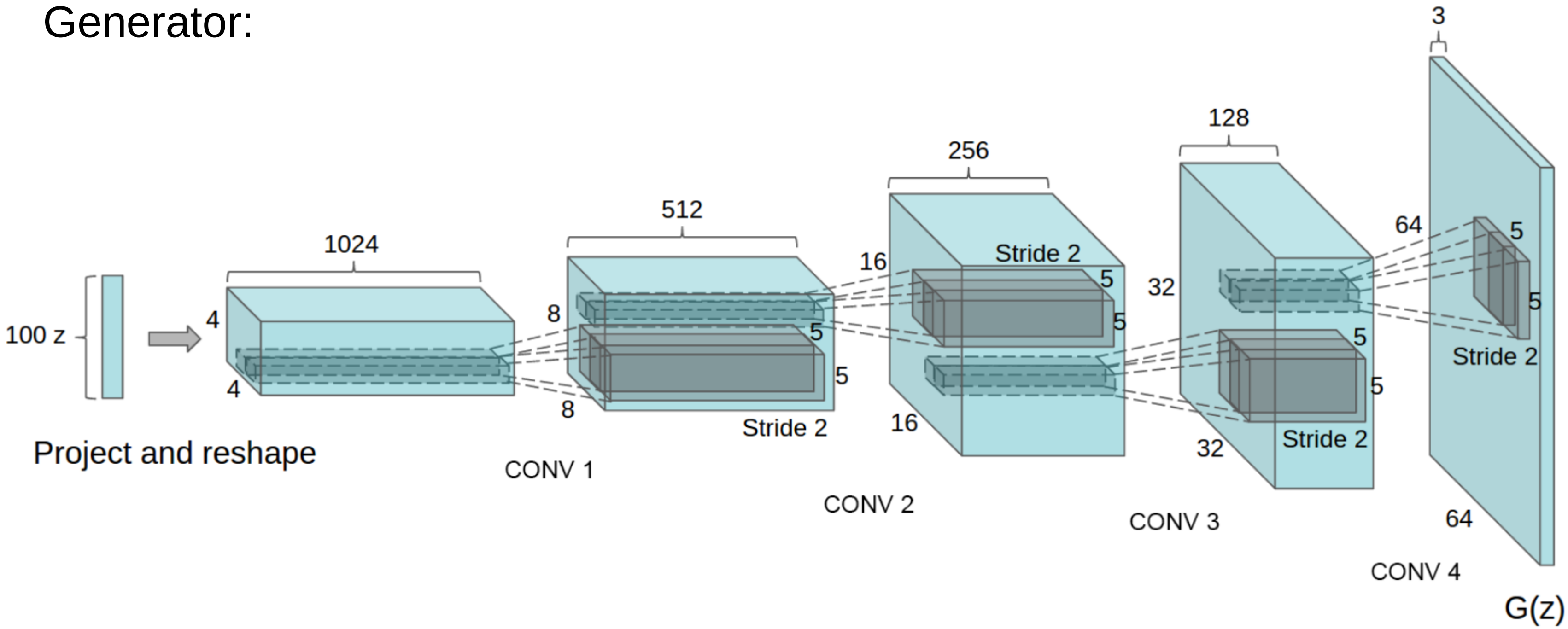
Various techniques have been employed in DCGAN for stable GAN training and higher resolution and deeper generative model:

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers
- Use Adam optimizer (Kingma & Ba, 2014) with tuned hyperparameters
- Use dropout

Ref: Radford et al., 2016

DCGAN Architecture

Generator:



# of Kernel	512	256	128	3
Kernel Size	5x5	5x5	5x5	5x5
Stride	2	2	2	2

Image from: Radford et al., 2016

DCGAN Architecture

Discriminator:

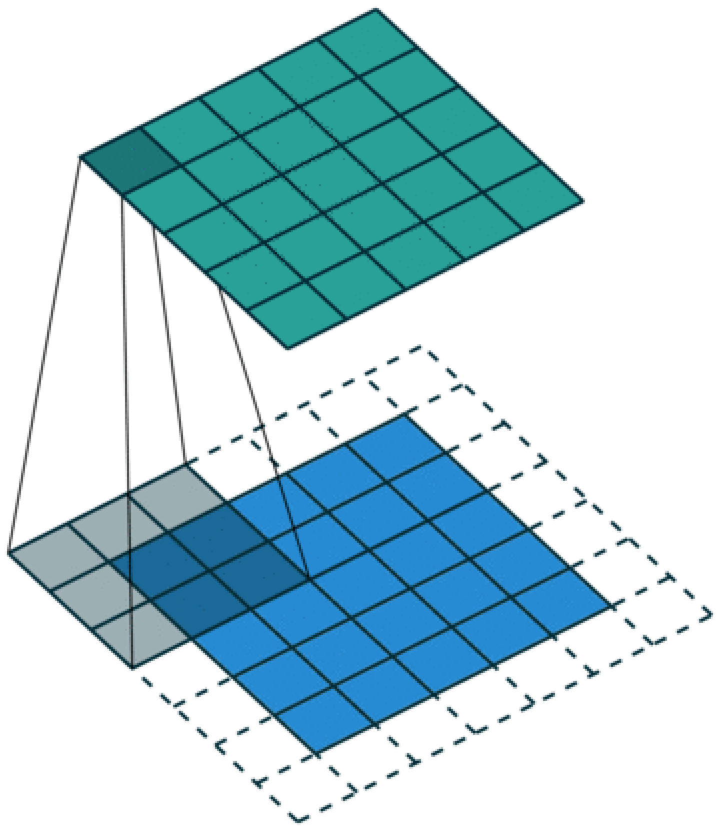
Image from: https://gluon.mxnet.io/chapter14_generative-adversarial-networks/dcgan.html

Outline

- Review of GAN
- DCGAN Introduction
- Convolutional/Transposed Layer
- Batch Normalization
- Dropout
- Activation
- DCGAN Revisit
- Demo

Convolutional Layer

DCGAN



$$Output = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

- i – Size of image
- p – Padding
- k – Size of kernel
- s – Stride

SAME padding: 5x5x1 image is padded with 0s to create a 5x5x1 image
Ref: link

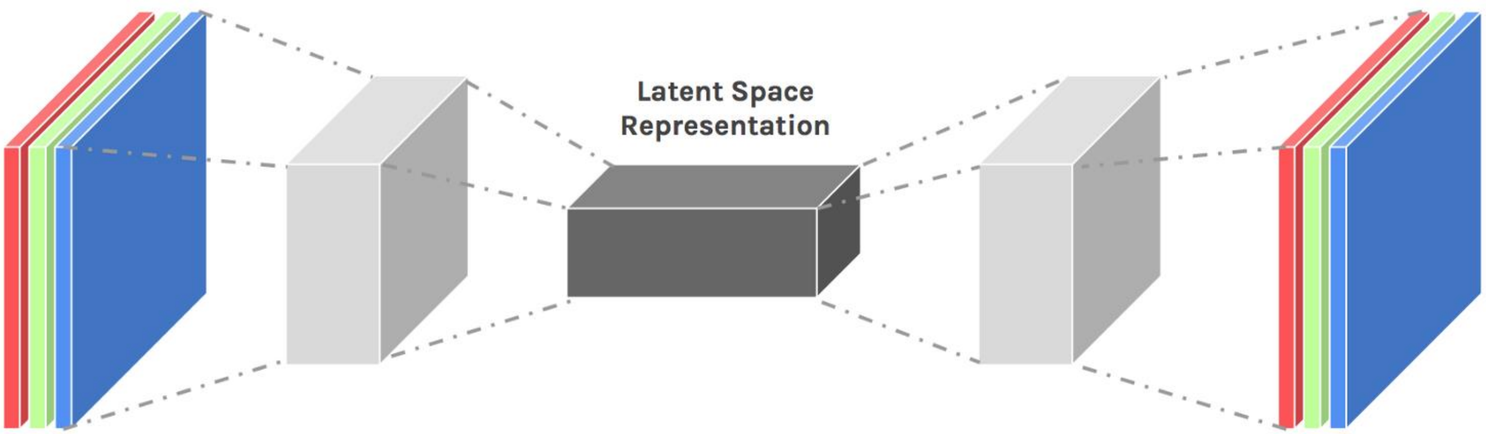
Transposed Conv Layer

What is fractional-strided convolutions /
transposed convolution /
deconvolution?

Transposed Conv Layer

DCGAN

Why do we need transposed convolutions?



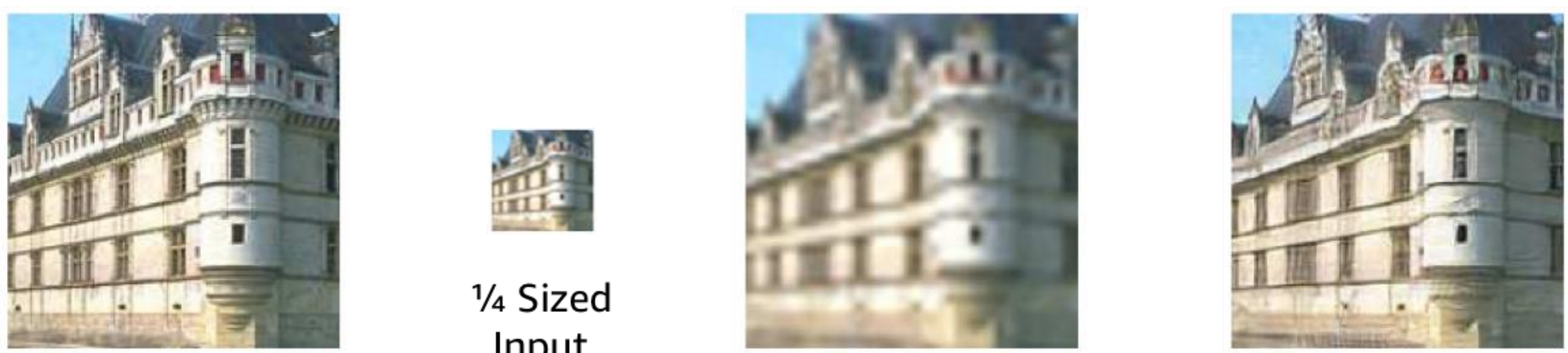
Architecture of Auto Encoder
image from: [link]

Transposed Conv Layer

DCGAN

Traditional upsampling approaches:

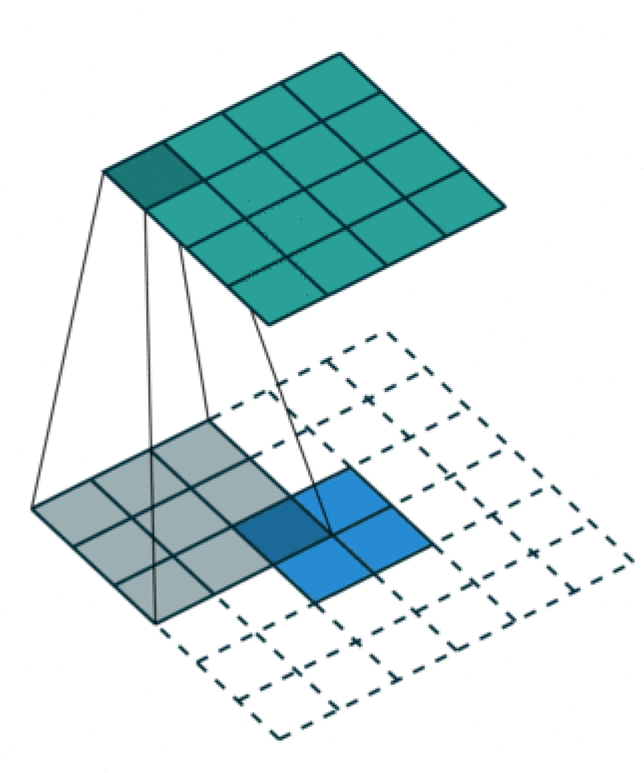
- Nearest neighbor interpolation
- Bi-linear interpolation
- Bi-cubic interpolation



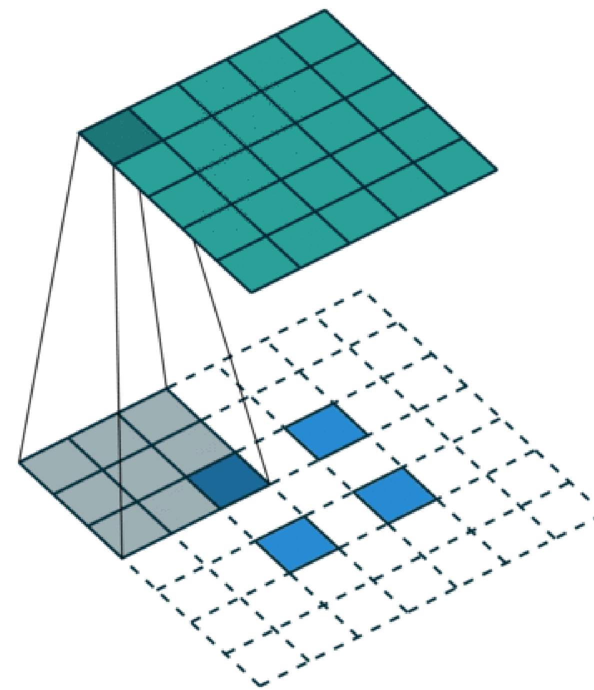
Bicubic vs Transposed Convolution in Upsampling
image from: link

Transposed Conv Layer

DCGAN



Up-sampling a 2 x 2 input to
a 4 x 4 output



Up-sampling a 2 x 2 input to
a 5 x 5 output.

Images adopted from this [link]

V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv:1603.07285 [cs, stat]*, Mar. 2016.

Transposed Conv Layer

DCGAN

$$Output = (i-1) * s - 2p + k$$

OR

$$Output = \begin{cases} i * s & \text{'same' padding} \\ (i-1) * s + k & \text{'valid' padding} \end{cases}$$

- i $- Size\ of\ image$
- p $- Padding$
- k $- Size\ of\ kernel$
- s $- Stride$

```
self.sample = keras.layers.Conv2DTranspose(filters = 512,
                                           kernel_size = 3,
                                           strides = 1,
                                           padding='valid')
```

Ref: [link]()

Outline

- Review of GAN
- DCGAN Introduction
- Convolutional/Transposed Layer
- Batch Normalization
- Dropout
- Activation
- DCGAN Revisit
- Demo

Batch Normalization

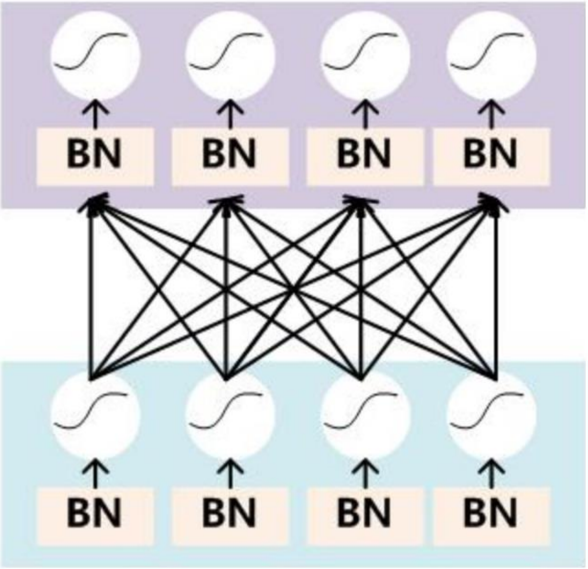
DCGAN

- What is batch normalization?

Batch Normalization

DCGAN

- What is batch normalization?

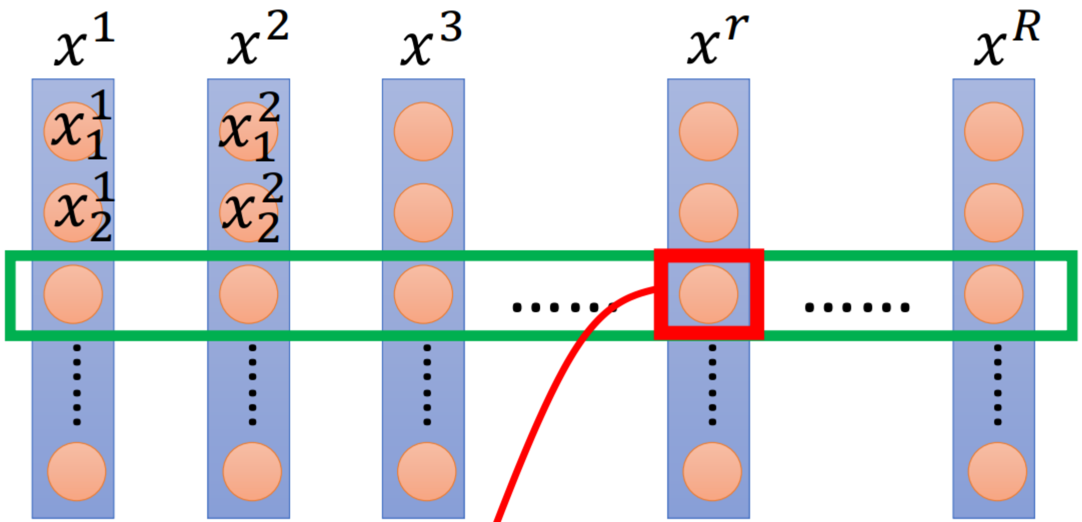


Batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

Image from: [Link]

Batch Normalization

DCGAN



For each dimension i:

mean: m_i

standard deviation: σ_i

$$x_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$$

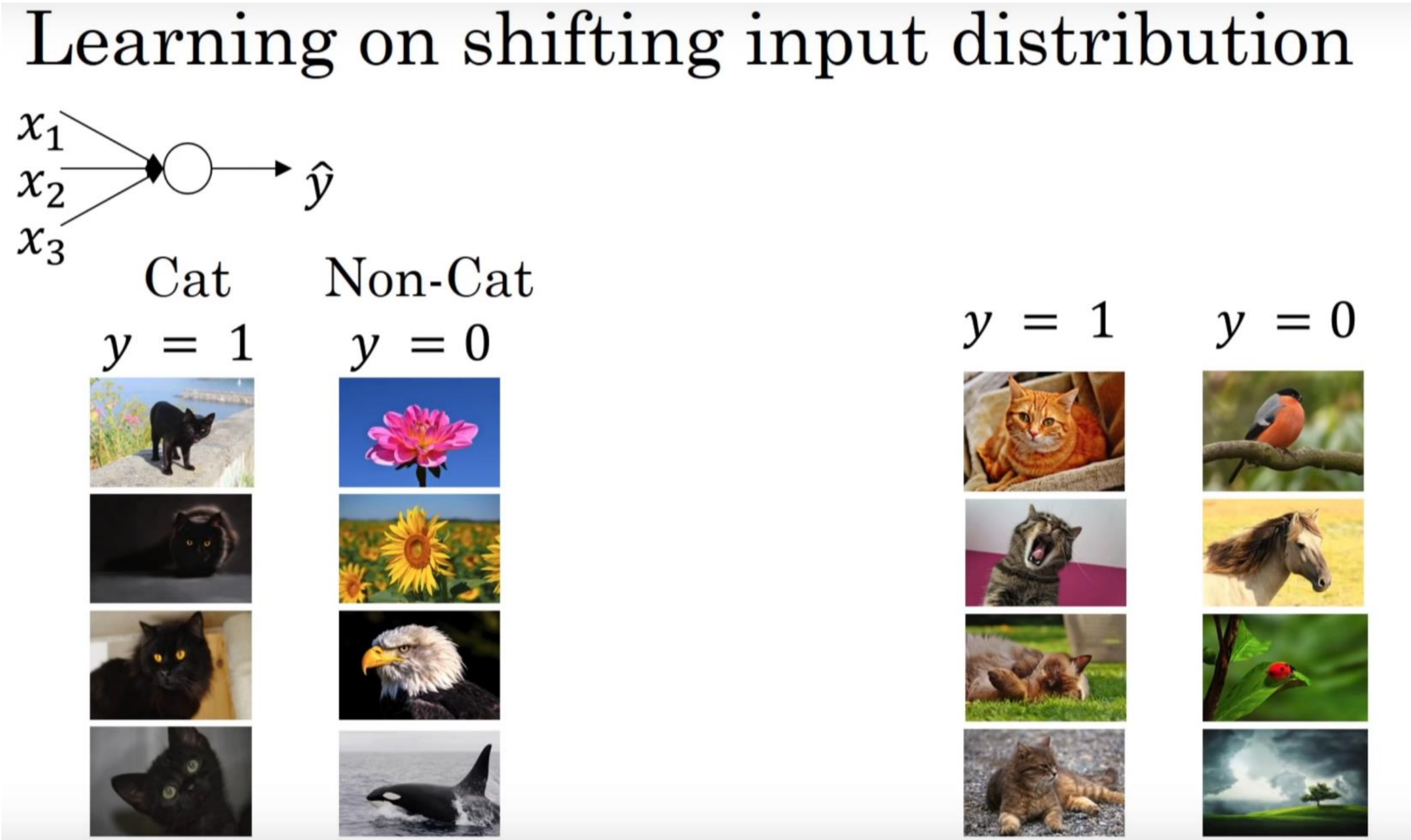
The means of all dimensions are 0, and the variances are all 1

Image from: [Link]()

Batch Normalization

- Why do we need batch normalization?

Batch Normalization



[Ref: Andrew NG]

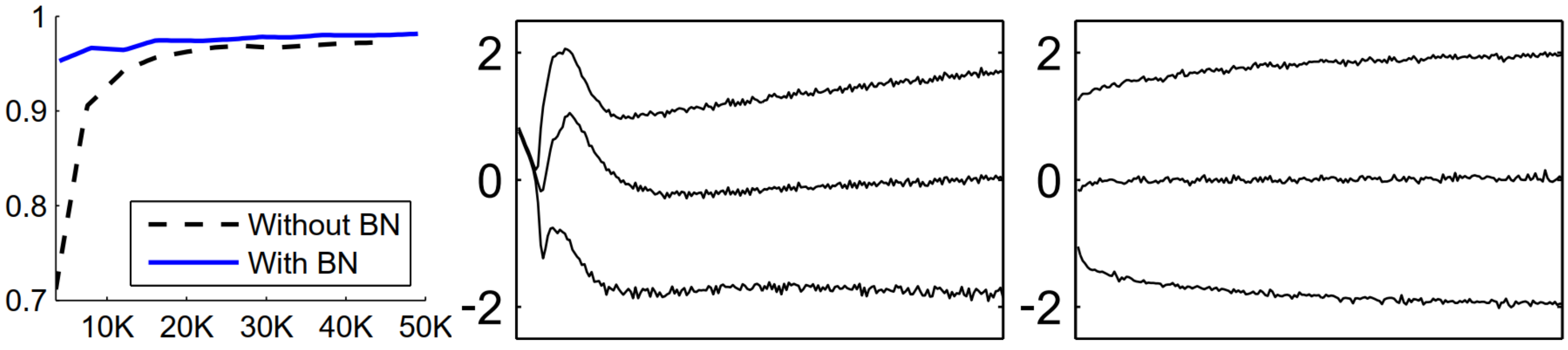
Batch Normalization

- Reduces the amount by what the hidden unit values shift around (covariance shift)
- Allows each layer of a network to learn by itself a little bit more independently of other layers
- Allows for higher learning rate
- Reduces overfitting
- Stabilizes and Speeds up training
- Helps gradient flow in deeper models

Ref: [link]

Batch Normalization

DCGAN



(a) (b) Without BN (c) With BN

Image from: Ioffee et al., 2016

Batch Normalization

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

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

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Image from: Ioffee et al., 2016

Batch Normalization

- Shall we apply batch normalization to all layers?
- Directly applying batchnorm to all layers however, resulted in sample oscillation and model instability. This was avoided by not applying batchnorm to the generator output layer and the discriminator input layer.

Ref: Ioffee et al., 2016

Outline

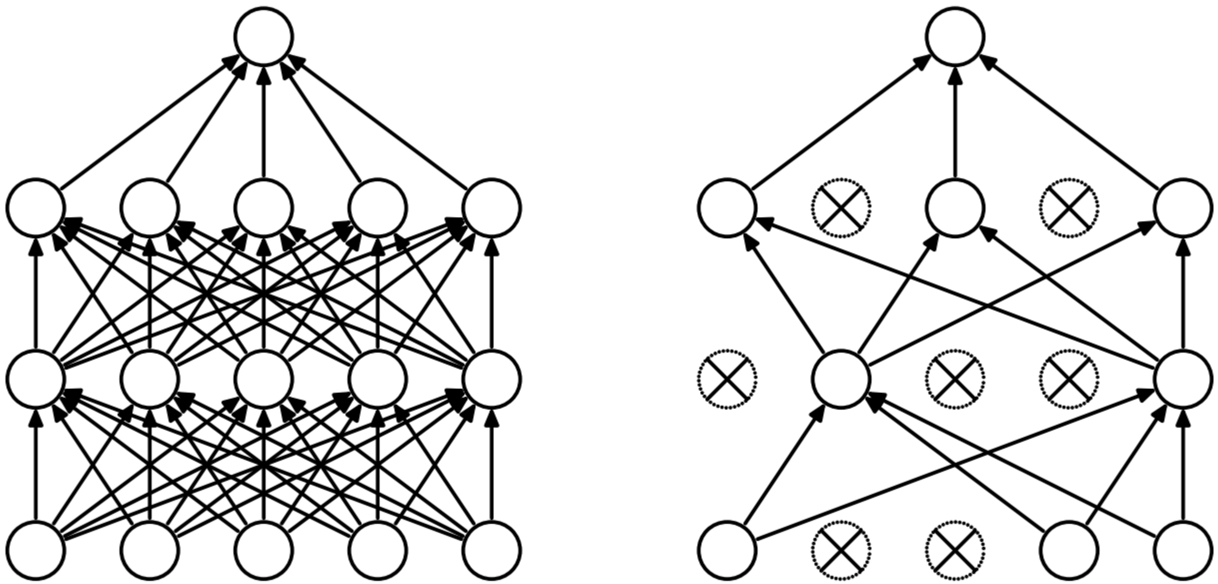
- Review of GAN
- DCGAN Introduction
- Convolutional/Transposed Layer
- Batch Normalization
- **Dropout**
- Activation
- DCGAN Revisit
- Demo

Dropout

- What is dropout?

Dropout

- What is dropout?



(a) Standard Neural Net (b) After applying dropout.

Ref: Srivastava et al., 2014

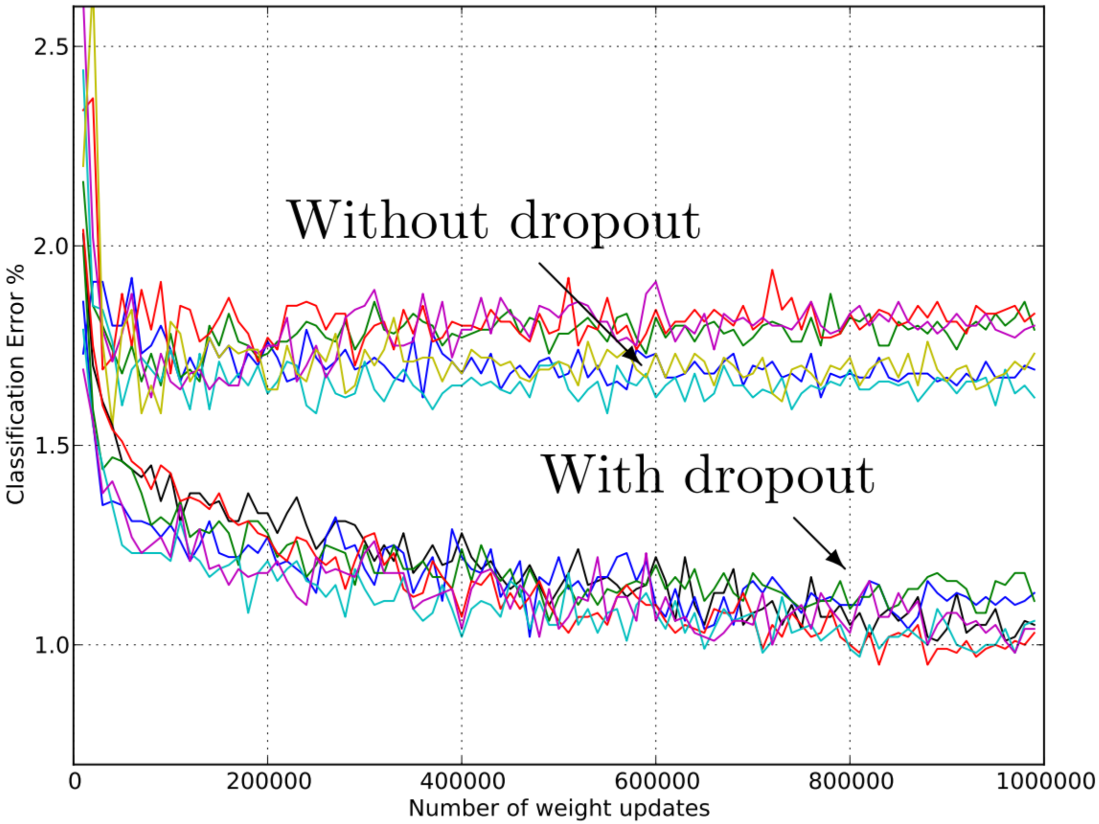
Dropout

- Why do we need dropout?

Dropout

- Why do we need dropout?

- to prevent overfitting



Ref: Srivastava et al., 2014

Dropout

Co-Adaption in Neural Network

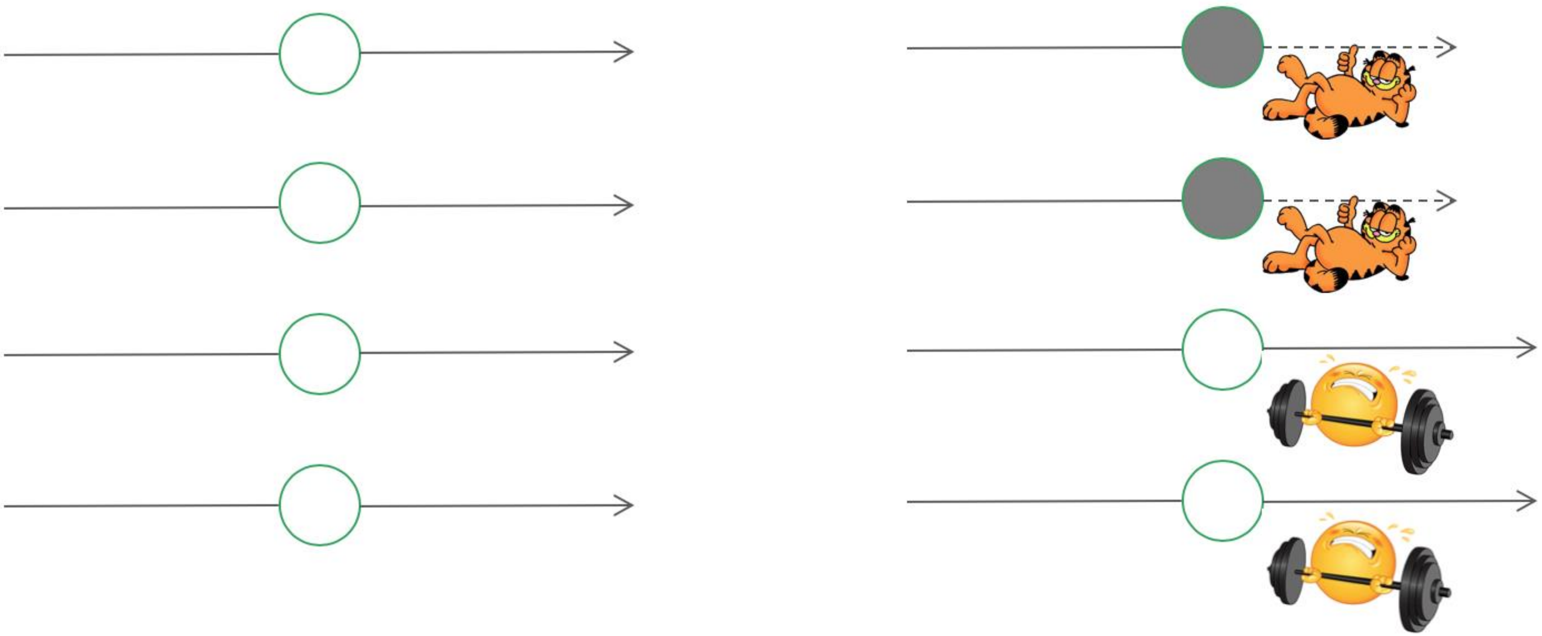


Image from: [link]

Dropout

- **Training Phase:**
- For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction, *p*, of nodes (and corresponding activations).

- **Testing Phase:**
- Use all activations, but reduce them by a factor *p* (to account for the missing activations during training).

Ref: [link]

Dropout

DCGAN

- Implementation

```python
class Downsample(keras.Model):

    def __init__(self, filters, size):
        super(Downsample, self).__init__()
        self.conv1 = keras.layers.Conv2D(filters, (size, size),
                                        strides=2,padding='same',  use_bias=False)
        self.batchnorm = keras.layers.BatchNormalization()

    def call(self, x, training):
        x = self.conv1(x)
        x = self.batchnorm(x, training=training)
        x = tf.nn.leaky_relu(x)
        return x
```

Outline

- Review of GAN
- DCGAN Introduction
- Convolutional/Transposed Layer
- Batch Normalization
- Dropout
- Activation
- DCGAN Revisit
- Demo

Activation Function

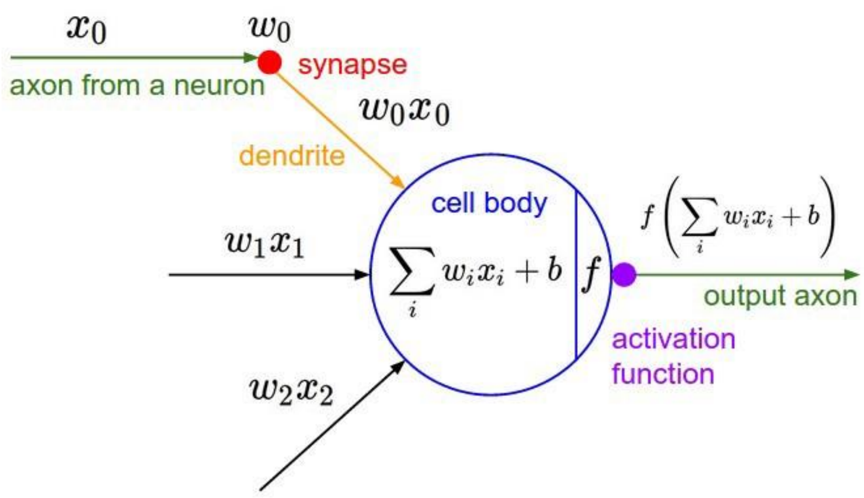


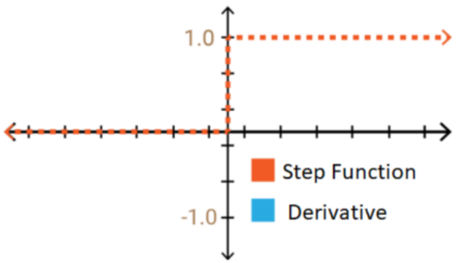
Image from: link

Activation Function

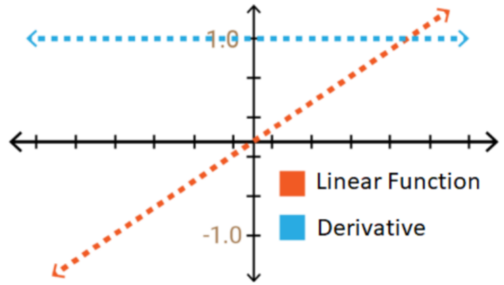
- Step Function
- Linear Function
- Sigmoid Function
- Hyperbolic Tangent Function
- ReLU (Rectified Linear Unit) Function
- Leaky-ReLU Function
- Softmax Function

Activation Function

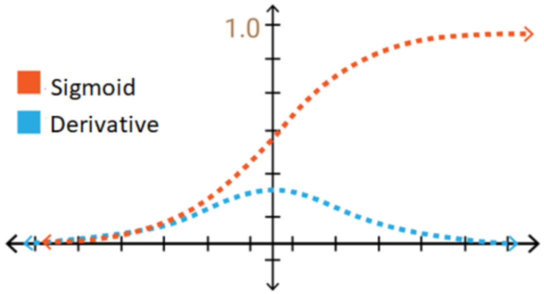
DCGAN



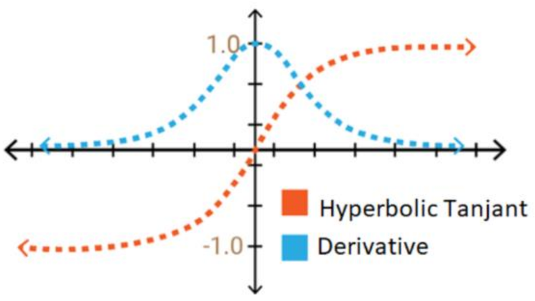
Step Function and Derivative



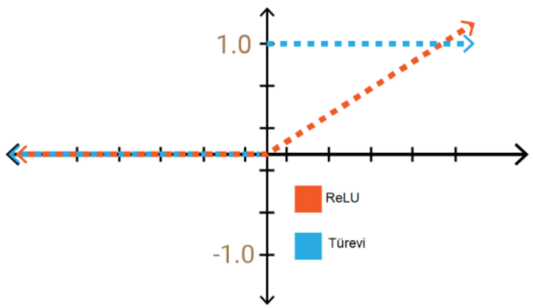
Linear Function and Derivative



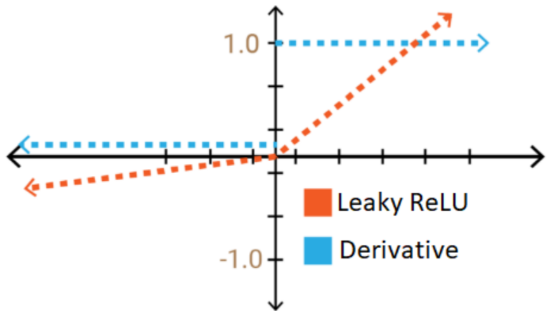
Sigmoid Function and Derivative



Hyperbolic Tangent and Derivative



ReLU Function and Derivative



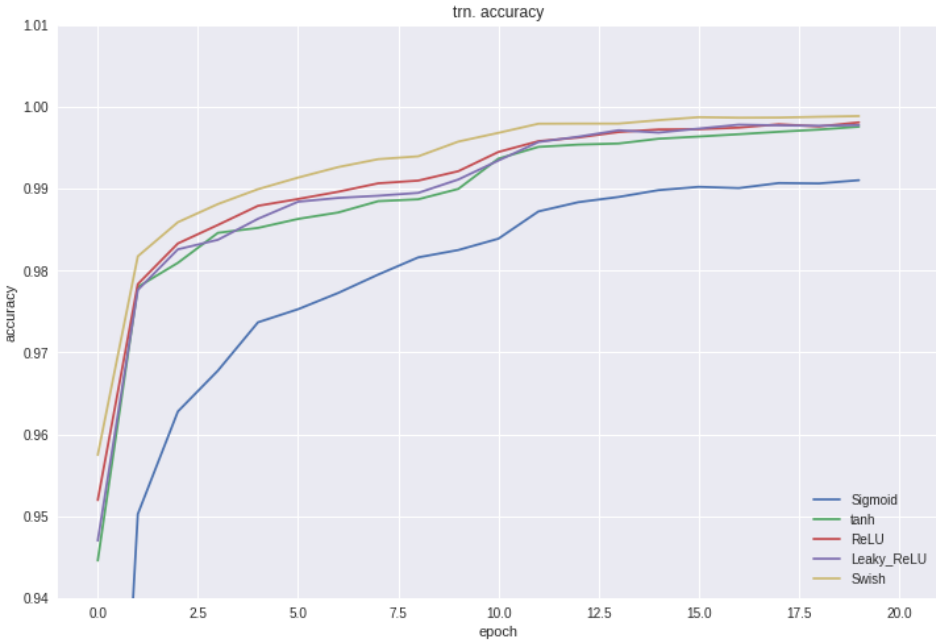
Leaky ReLU Function and Derivative

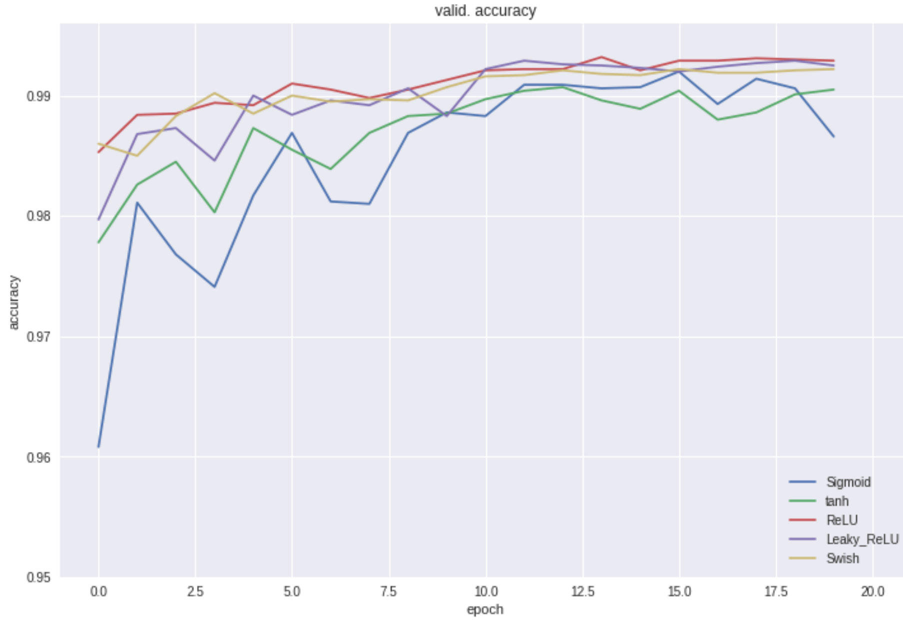
Image from: [link]

Activation Function

DCGAN

Comparison of activation functions for the Convolutional Neural Network Model on the classic MNIST dataset.





Reference: [link]

Activation Function

DCGAN

ACTIVATION FUNCTION	EQUATION	RANGE
Linear Function	$f(x) = x$	$(-\infty, \infty)$
Step Function	$f(x) = \begin{cases} 0 \ for \ x < 0 \\ 1 \ for \ x \geq 0 \end{cases}$	$\{0, 1\}$
Sigmoid Function	$f(x) = \sigma(x) = \dfrac{1}{1 + e^{-x}}$	$(0, 1)$
Hyperbolic Tanjant Function	$f(x) = \tanh(x) = \dfrac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$(-1, 1)$
ReLU	$f(x) = \begin{cases} 0 \ for \ x < 0 \\ x \ for \ x \geq 0 \end{cases}$	$[0, \infty)$
Leaky ReLU	$f(x) = \begin{cases} 0.01 \ for \ x < 0 \\ x \ for \ x \geq 0 \end{cases}$	$(-\infty, \infty)$

Image from: link

Outline

- Review of GAN
- DCGAN Introduction
- Convolutional/Transposed Layer
- Batch Normalization
- Dropout
- Activation
- DCGAN Revisit
- Demo

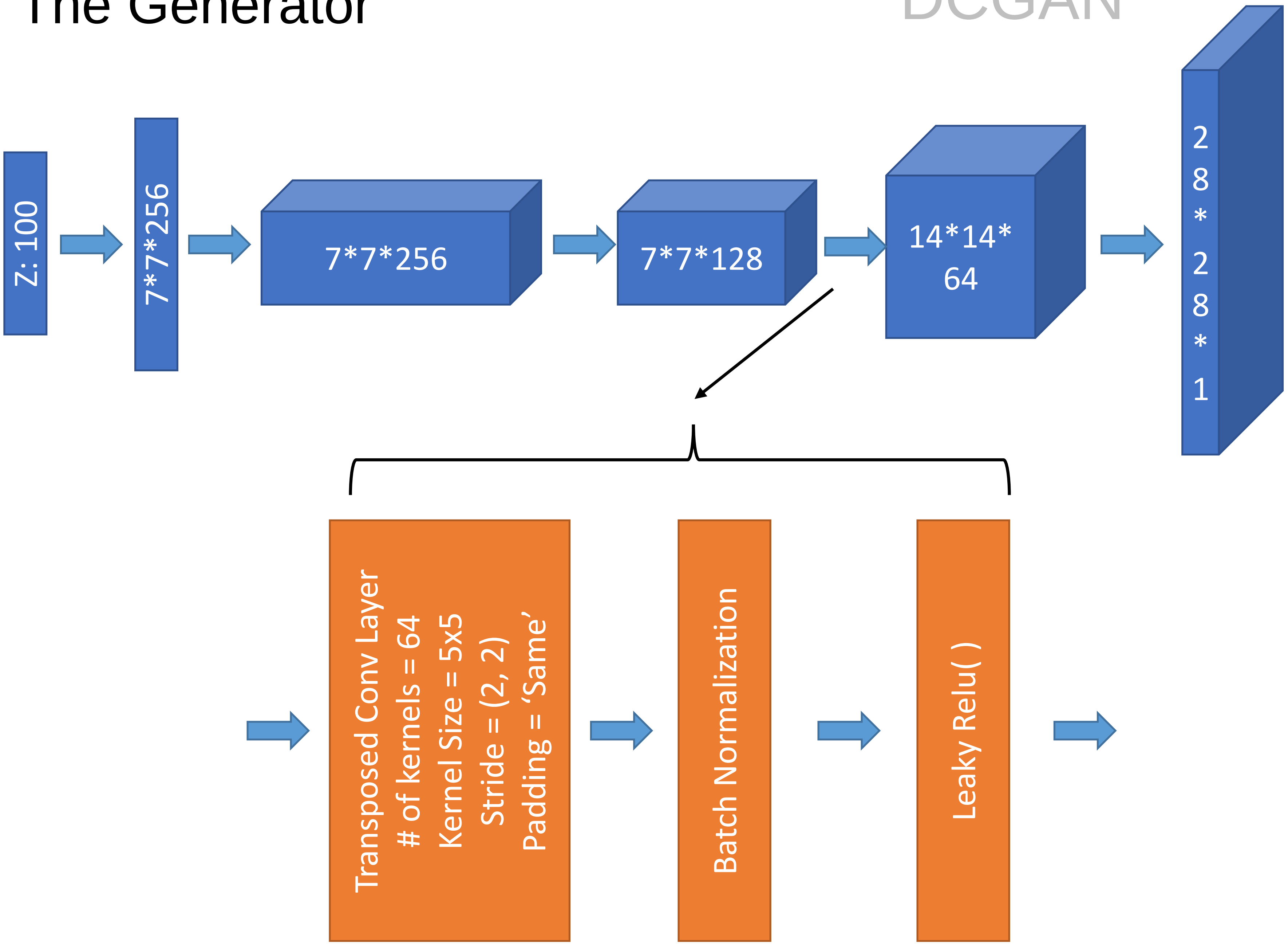
Revisit

Techniques used in DCGAN:

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers
- Use Adam optimizer (Kingma & Ba, 2014) with tuned hyperparameters
- Use dropout

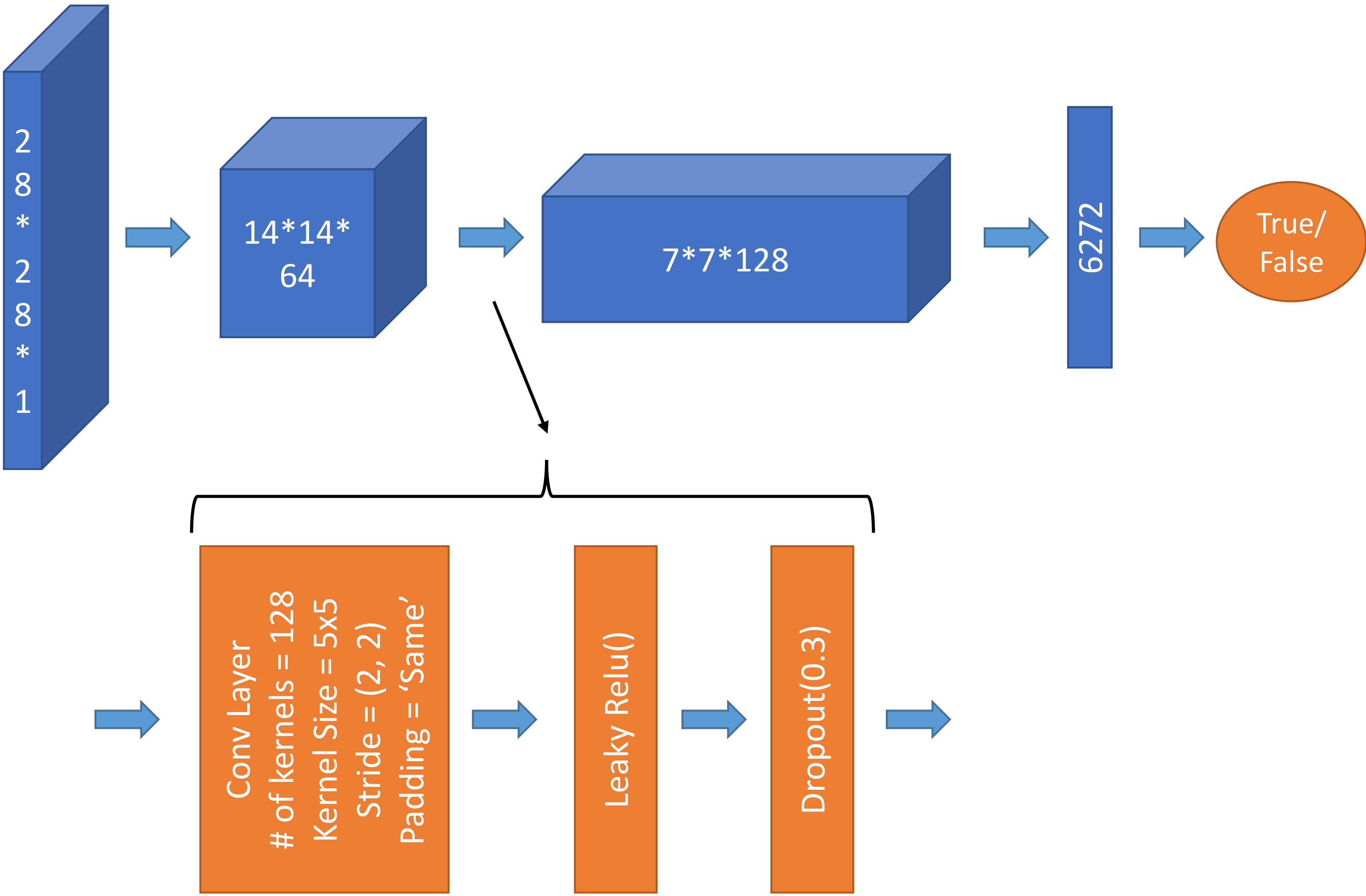
The Generator

DCGAN



The Discriminator

DCGAN



Outline

- Review of GAN
- DCGAN Introduction
- Convolutional/Transposed Layer
- Batch Normalization
- Dropout
- Activation
- DCGAN Revisit
- Demo

Code Demo

DCGAN

https://colab.research.google.com/drive/1Bb3xliBeoMo2peIM-5doehnb71tXo1K-

Reference

- [1] A. Radford, L. Metz, and S. Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” *arXiv:1511.06434 [cs]*, Nov. 2015.

- [2] I. Goodfellow, “NIPS 2016 Tutorial: Generative Adversarial Networks,” p. 57.

- [3] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *arXiv:1603.07285 [cs, stat]*, Mar. 2016.

- [4]S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” *arXiv:1502.03167 [cs]*, Feb. 2015.

- [5]N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” p. 30.