Practical Machine Learning

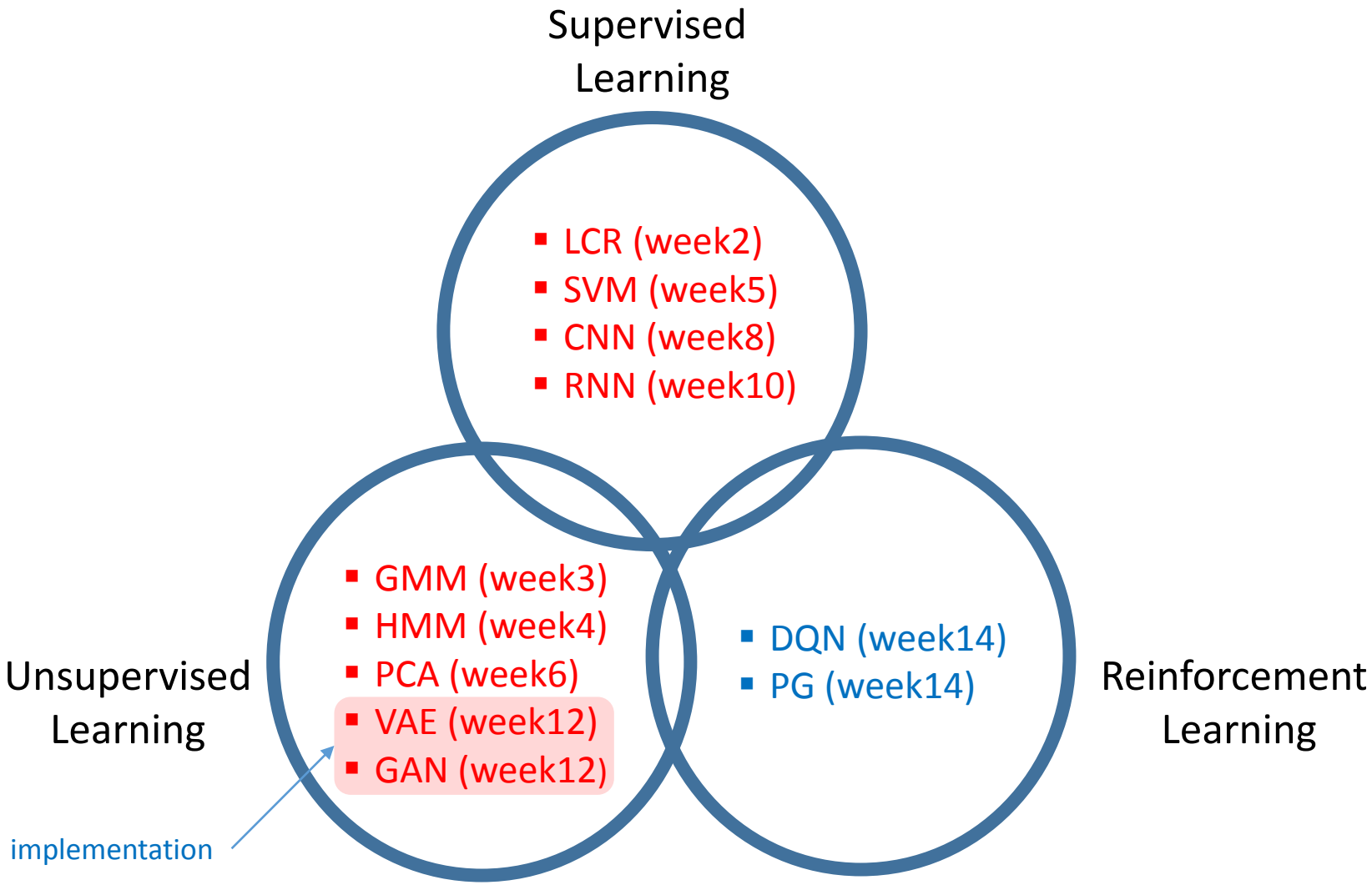
Lecture 13

Tensorflow –VAE/DCGAN implementation

Dr. Suyong Eum

OSAKA UNIVERSITY

Where we are

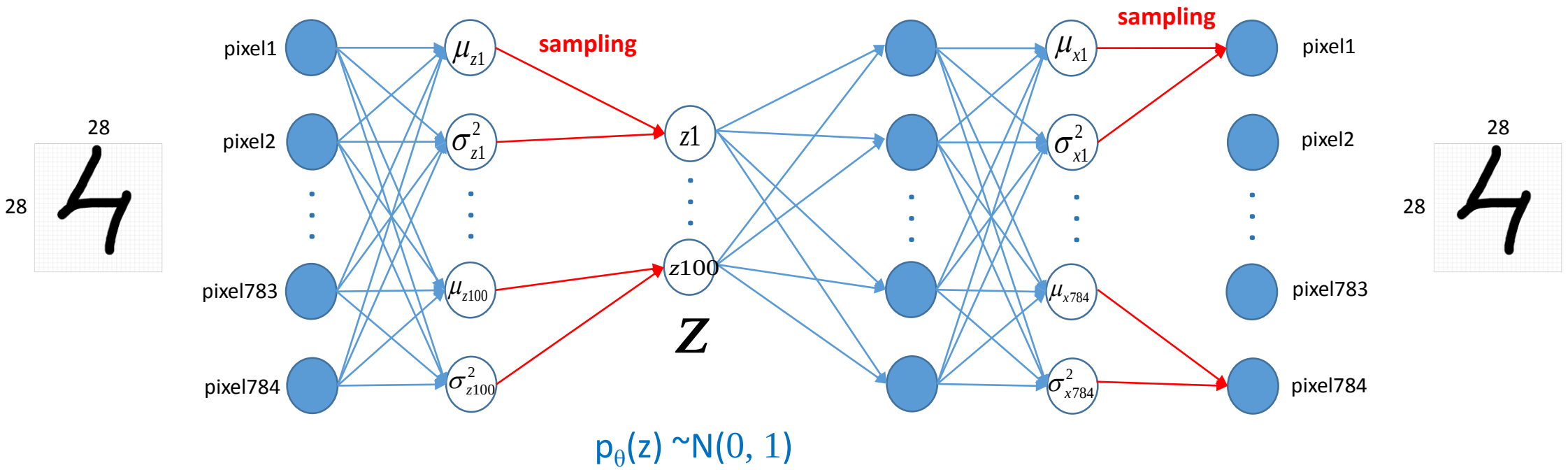

Supervised Learning
LCR (week2)
SVM (week5)
CNN (week8)
RNN (week10)
Unsupervised Learning
GMM (week3)
HMM (week4)
PCA (week6)
VAE (week12)
GAN (week12)
Reinforcement Learning
DQN (week14)
PG (week14)
implementation

You are going to learn

- ❑ Variational Auto Encoder (VAE) implementation
- ❑ Deep Convolutional GAN (DCGAN) implementation

Variational AutoEncoder (VAE)

Whole structure of VAE implementation



$p_\theta(z) \sim N(0, 1)$

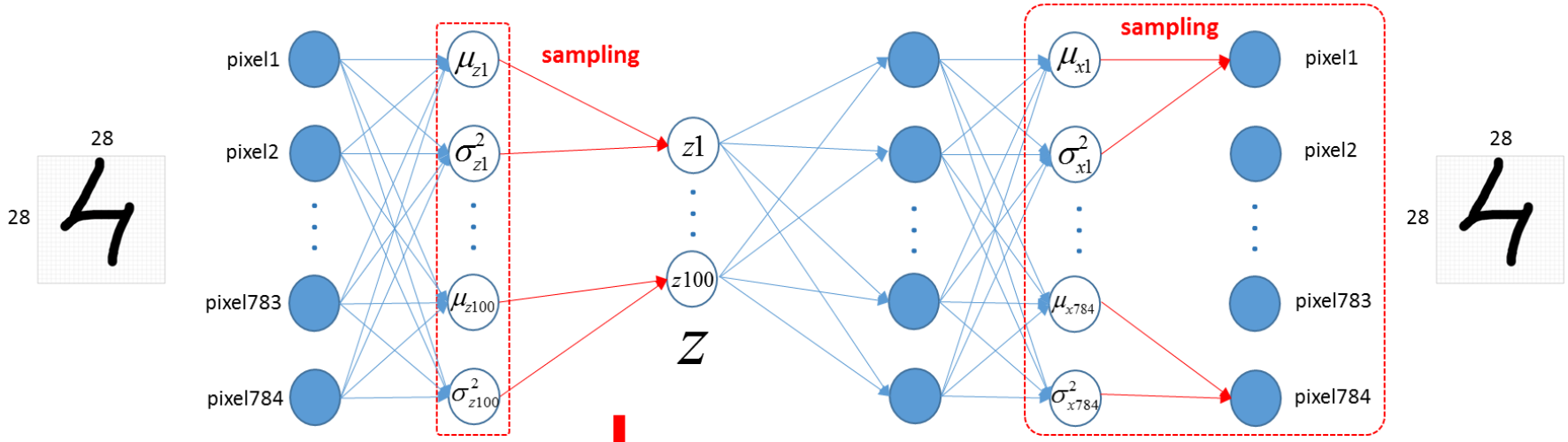
$q_\varphi(z|x) \sim N(\mu, \sigma^2)$

$p_\theta(x|z) \sim N(\mu, \sigma^2)$ or Bernoulli (p)

$$\{w, b\} \in \varphi$$

$$\{w, b\} \in \theta$$

Function blocks for the implementation



$\{w, b\} \in \varphi$

$\{w, b\} \in \theta$

$q_\varphi(z|x) \sim N(\mu, \sigma^2)$

$p_\theta(x|z) \sim N(\mu, \sigma^2)$ or Bernoulli (p)

1) Data loading

Data Batch
28x28x64

2) Encoder function

function q(X)
return μ, σ

3) Sampling function: re-parameterization

function z_sample(μ, σ)
return Z

4) Decoder function

function p(Z)
return X

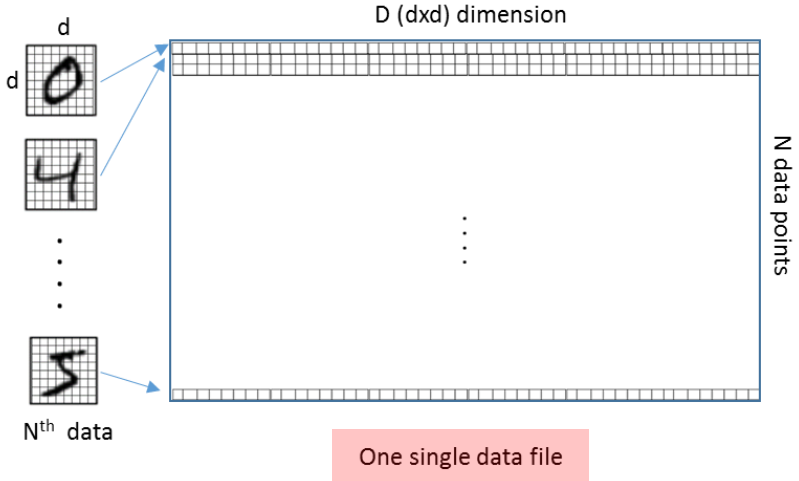
5) Loss function

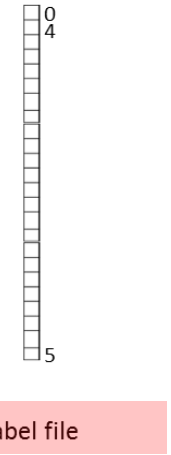
function L(X)
return error

Back-propagating the error to tune the parameters φ and θ

1) Data loading

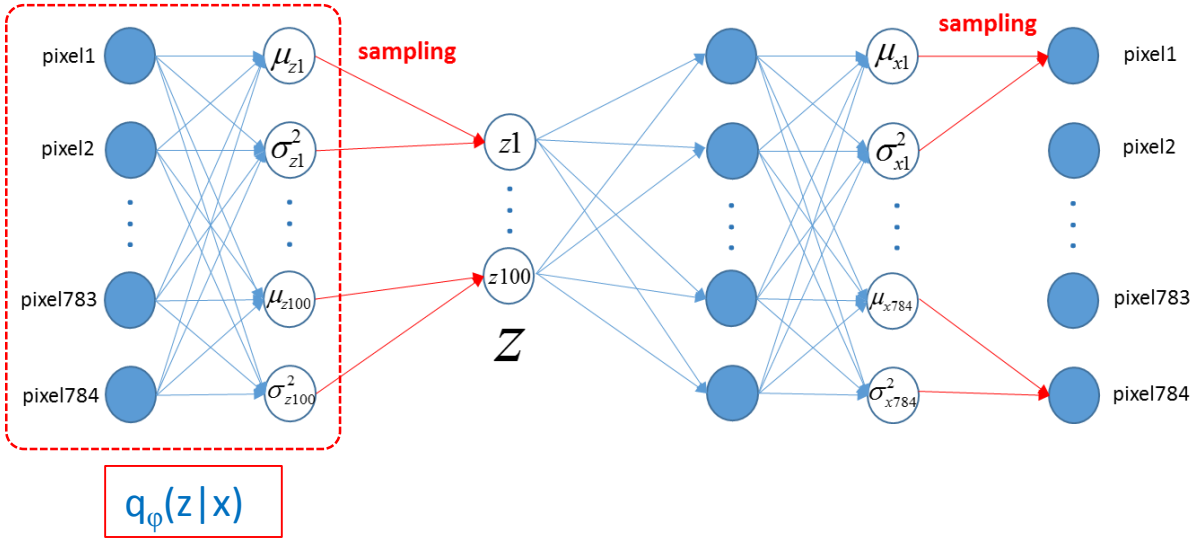
- MNIST data set in tensorflow (or directly from http://yann.lecun.com/exdb/mnist/)
  - Training data
    - One single file (45M) which includes 60,000 hand digit images for training,
    - One single file (59K) which includes corresponding labels.
  - Testing data
    - One single file (7.5M) which includes 10,000 hand digit images for testing,
    - One single file (9.8K) which includes corresponding labels.

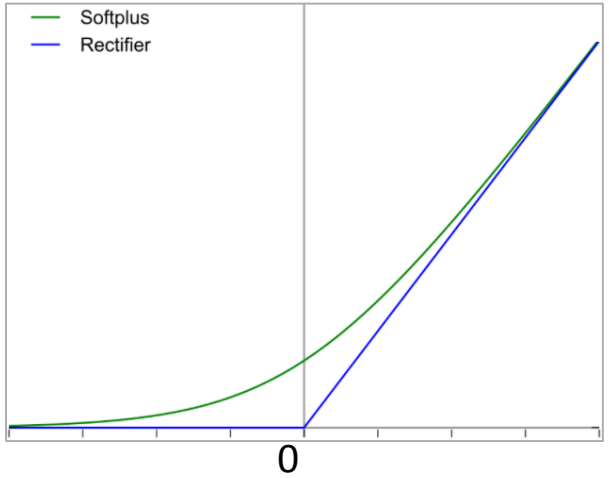


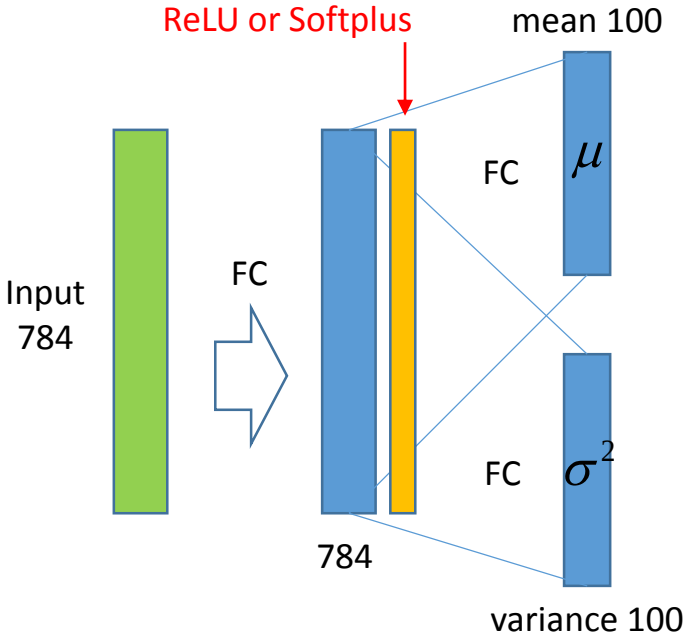



```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
x, y = mnist.train.next_batch(batch_size)
```

2) Encoder function

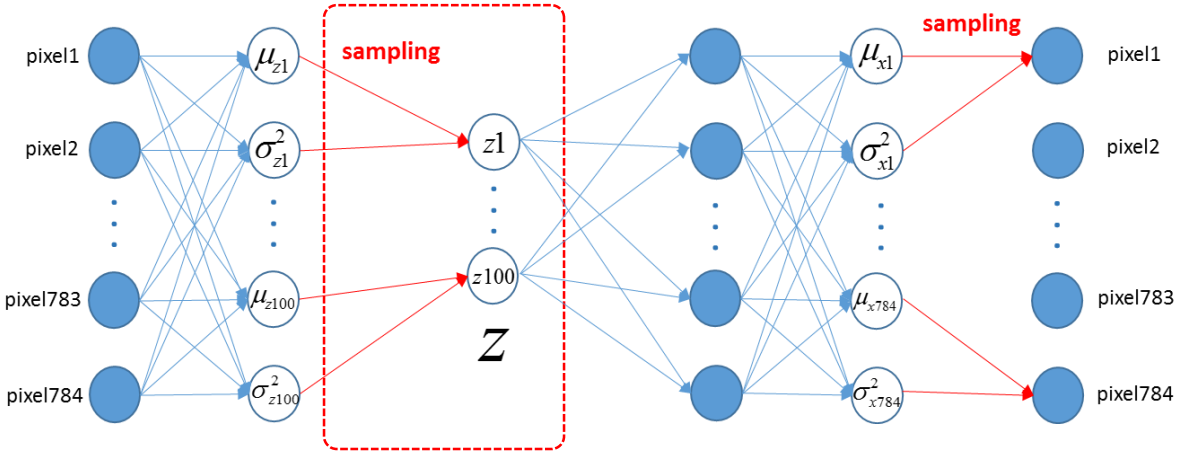




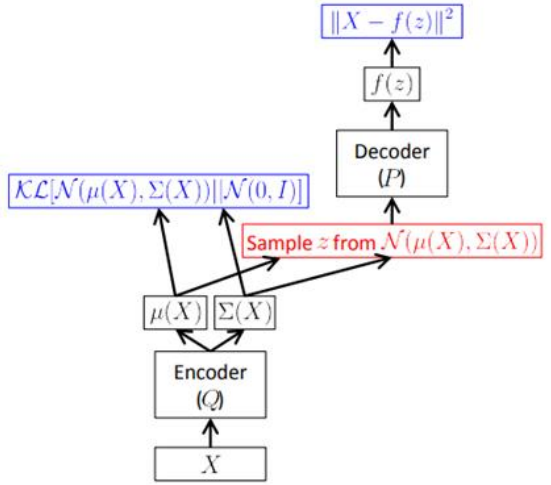


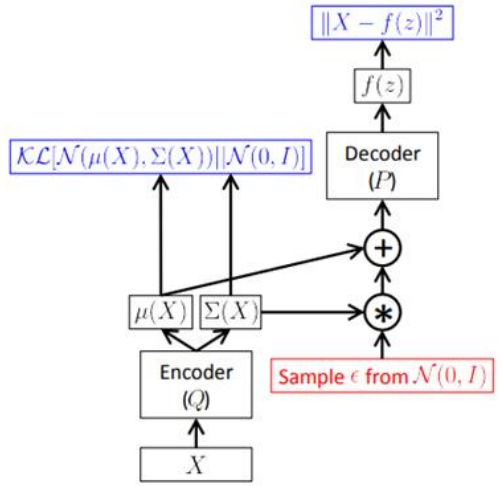
- Encoder $q_\varphi(z|x)$
  - Input: an image data (X): batch files
  - Fully connected (FC) hidden layer
  - ReLU or Softplus activation function
    - tf.nn.relu https://github.com/wiseodd/generative-models
    - tf.nn.softplus https://jmetzen.github.io/2015-11-27/vae.html
  - Output: means and variances of Gaussian distributions for latent variables “Z”

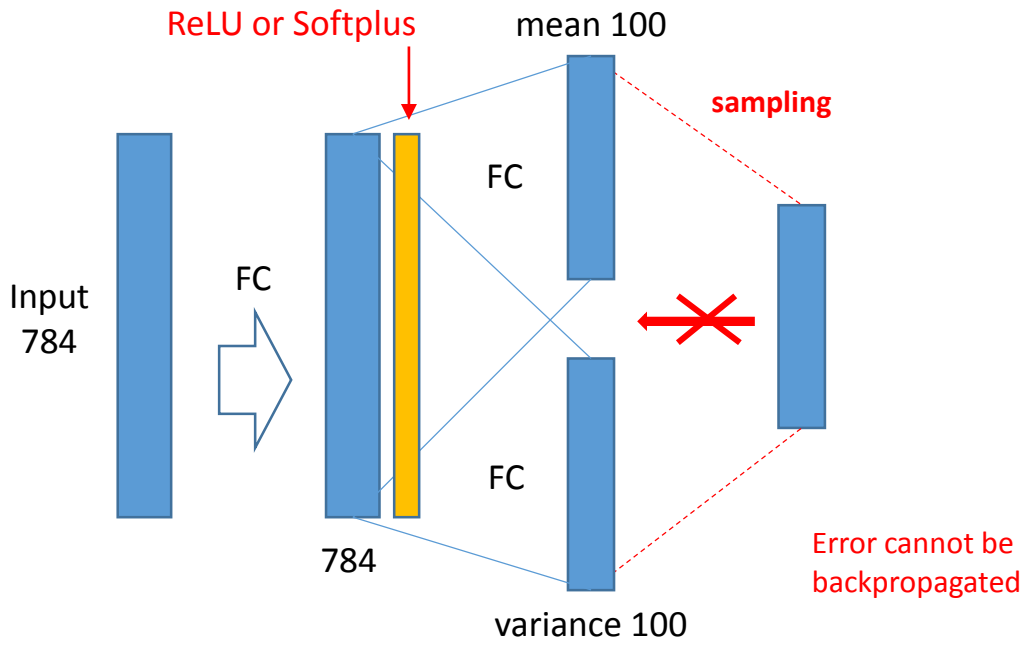
3) Sampling function: re-parameterization

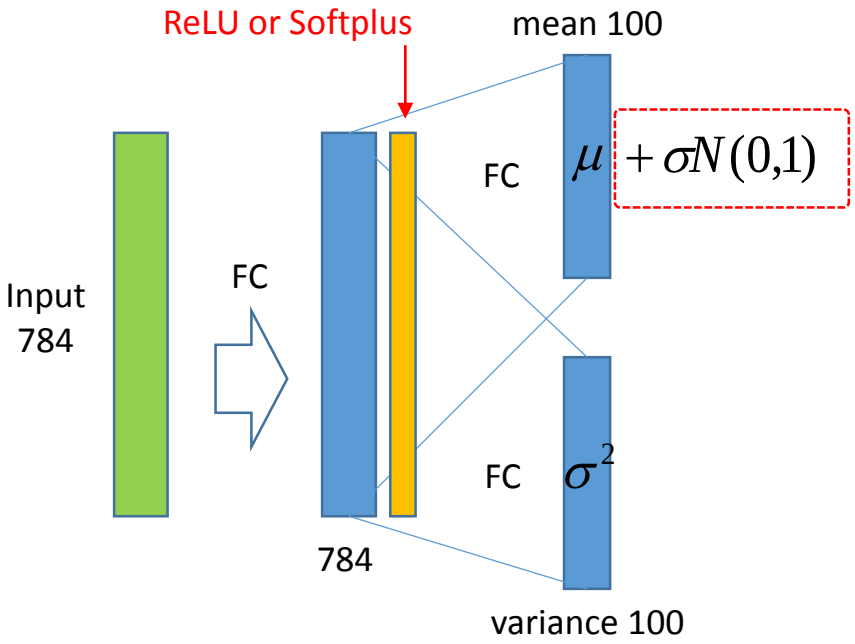


$q_\varphi(z|x)$

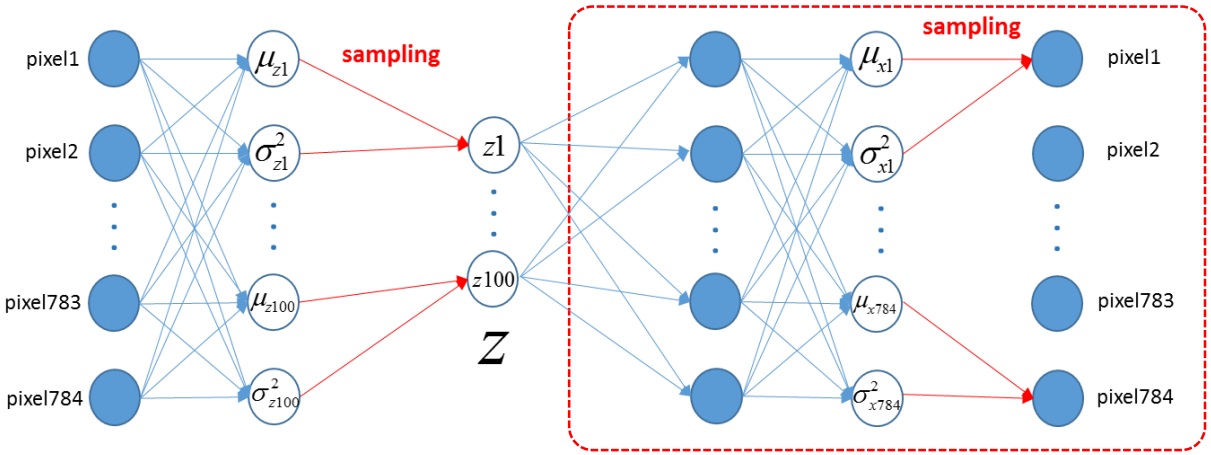




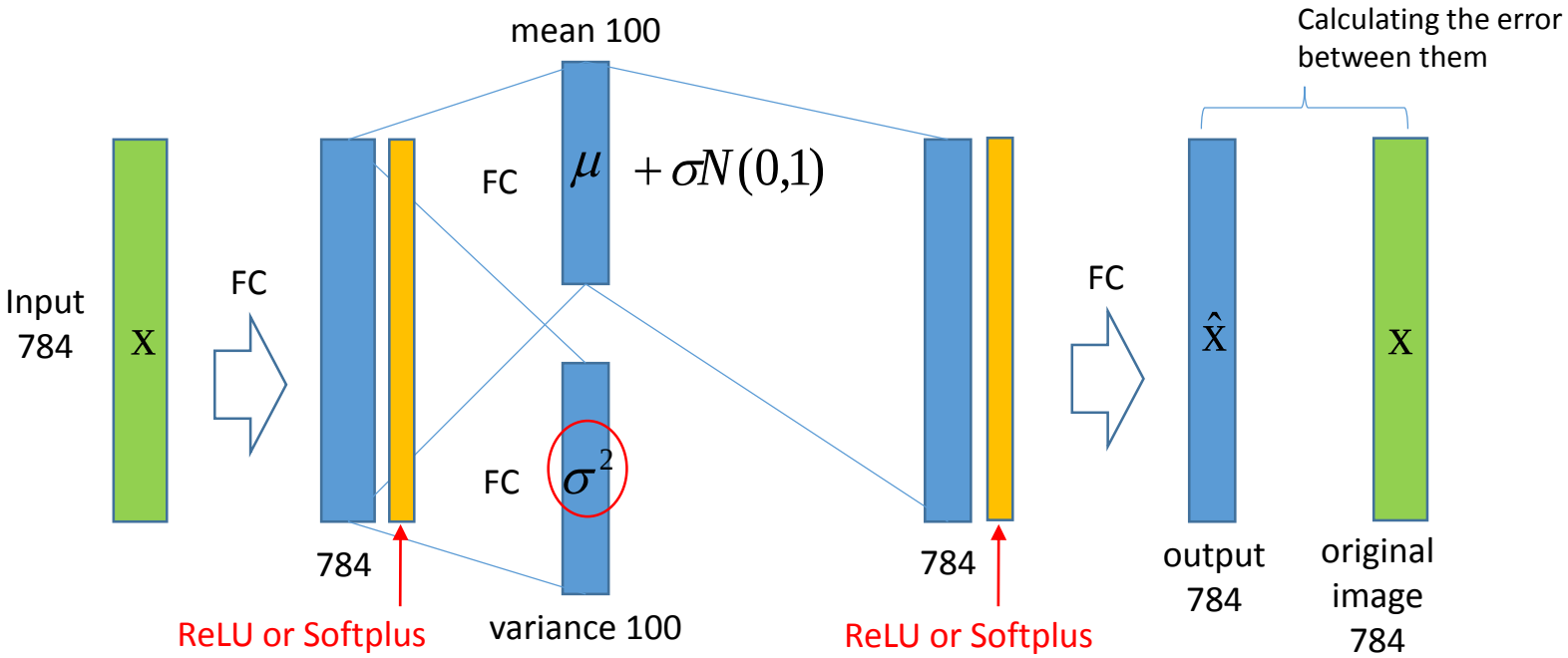




4) Decoder function



$q_\varphi(z|x)$



5) Loss function

- We maximized the lower bound of the likelihood function below to estimate the parameters; θ and φ (weights of the neural networks).
- In tensorflow, an optimization algorithm is implemented to minimize the loss, and so we change the optimization problem accordingly

$$\max_{\varphi,\theta}\left(-D_{KL}(q_\varphi(z\mid \mathrm{x}^{(\mathrm{i})})\,\|\,p_\theta(z))+E_{q_\varphi(z|\mathrm{x}^{(\mathrm{i})})}(\log p_\theta(\mathrm{x}^{(\mathrm{i})}\mid z))\right)$$

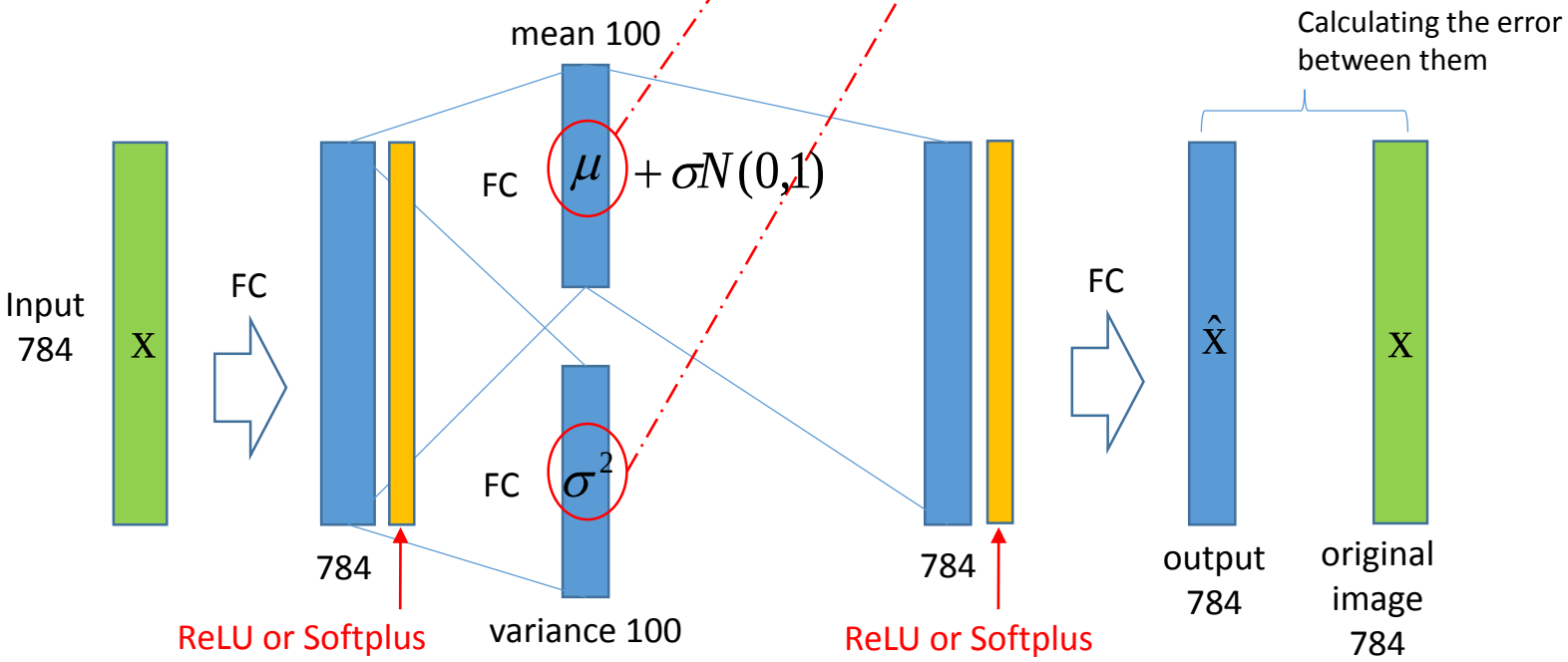
$$\min_{\varphi,\theta}\left(D_{KL}(q_\varphi(z\mid \mathrm{x}^{(\mathrm{i})})\,\|\,p_\theta(z))-E_{q_\varphi(z|\mathrm{x}^{(\mathrm{i})})}(\log p_\theta(\mathrm{x}^{(\mathrm{i})}\mid z))\right)$$

5) Loss function: Latent loss

$$\log p_\theta(\mathrm{x}^{(i)}) \geq -D_{KL}(q_\varphi(z \mid \mathrm{x}^{(i)}) \,\|\, p_\theta(z)) + E_{q_\varphi(z|\mathrm{x}^{(i)})}(\log p_\theta(\mathrm{x}^{(i)} \mid z))$$

$$= \frac{1}{2}\sum_{j=1}^{J}\left(1+\log((\sigma_{z_j}^{(i)})^2) - (\mu_{z_j}^{(i)})^2 - (\sigma_{z_j}^{(i)})^2\right)$$

- Latent loss



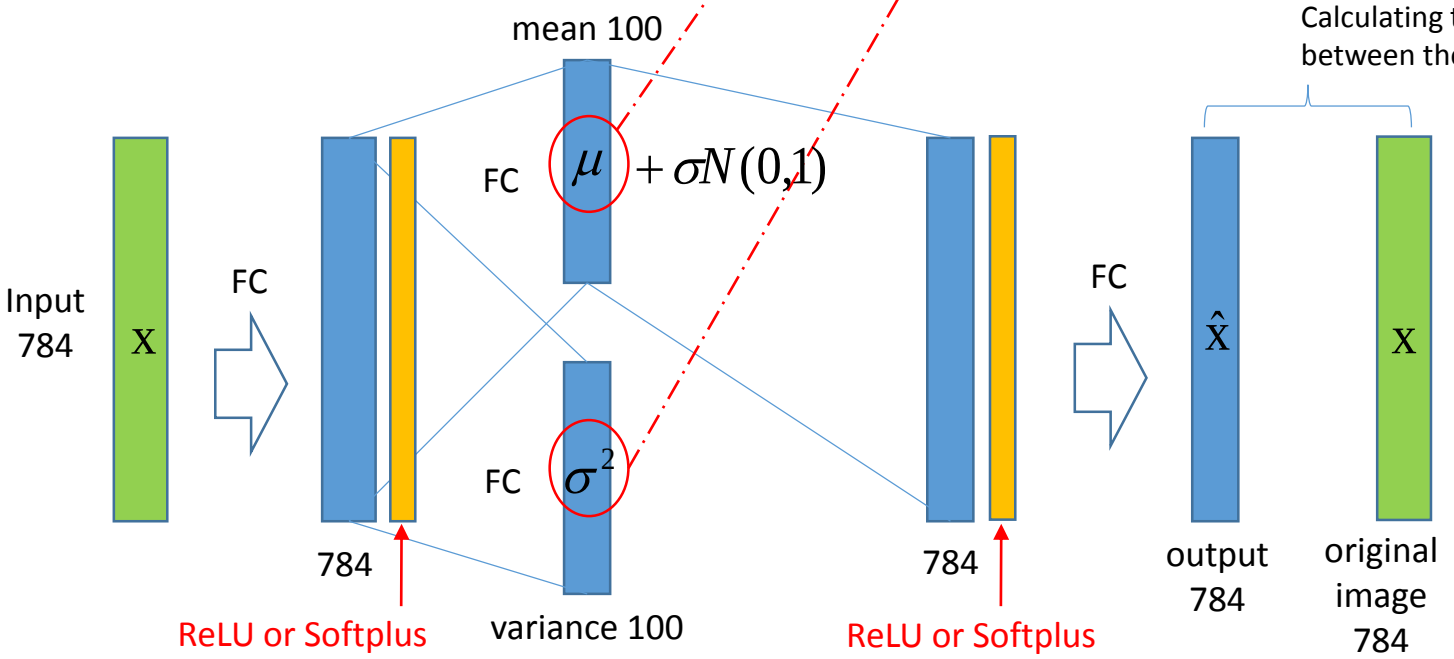
5) Loss function: reconstruction loss

$$\log p_\theta(\mathrm{x}^{(i)}) \geq -D_{KL}(q_\varphi(z \mid \mathrm{x}^{(i)}) \,\|\, p_\theta(z)) + E_{q_\varphi(z|\mathrm{x}^{(i)})}(\log p_\theta(\mathrm{x}^{(i)} \mid z))$$

$$= \frac{1}{2}\sum_{j=1}^{J}\left(1+\log((\sigma_{z_j}^{(i)})^2) - (\mu_{z_j}^{(i)})^2 - (\sigma_{z_j}^{(i)})^2\right)$$

1) Gaussian
2) Bernoulli

- Latent loss
- Reconstruction loss



Calculating the error between them

$$= x^{(i)}\hat{x}^{(i)} + (1-x^{(i)})(1-\hat{x}^{(i)})$$

x

0.	0.	0.	0.	0.	0.	0.
0.10980393	0.49803925	0.61176473	0.31764707	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.	0.03137255	0.00784314	0.07843138	0.43529415	0.46274513	
0.84313732	0.90980399	0.96862751	0.85490203	0.52941179	0.	0.
0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	0.	0.	0.	0.

$\hat{x}$

8.29576671e-01	2.25610152e-01	2.02522390e-02	4.82337356e-01
5.16538285e-02	2.06836089e-01	3.81448448e-01	1.49948254e-01
9.86250281e-01	1.46263391e-01	2.86624637e-02	9.65128660e-01
7.27081716e-01	9.45814967e-01	1.02548182e-01	1.05315238e-01
8.23803186e-01	3.49941873e-03	7.28377581e-01	9.78516757e-01
6.44292772e-01	7.11309165e-03	8.89515042e-01	7.42009878e-02
9.61810350e-01	8.40429783e-01	8.40326011e-01	6.09001637e-01
6.46354258e-01	7.32857466e-01	1.23425394e-01	6.06033020e-02

5) Loss function: reconstruction loss

$$\log p_\theta(x^{(i)}) \geq \underbrace{-D_{KL}(q_\varphi(z|x^{(i)}) \| p_\theta(z))}_{\text{Latent loss}} + \underbrace{E_{q_\varphi(z|x^{(i)})}(\log p_\theta(x^{(i)}|z))}_{\text{Reconstruction loss}}$$

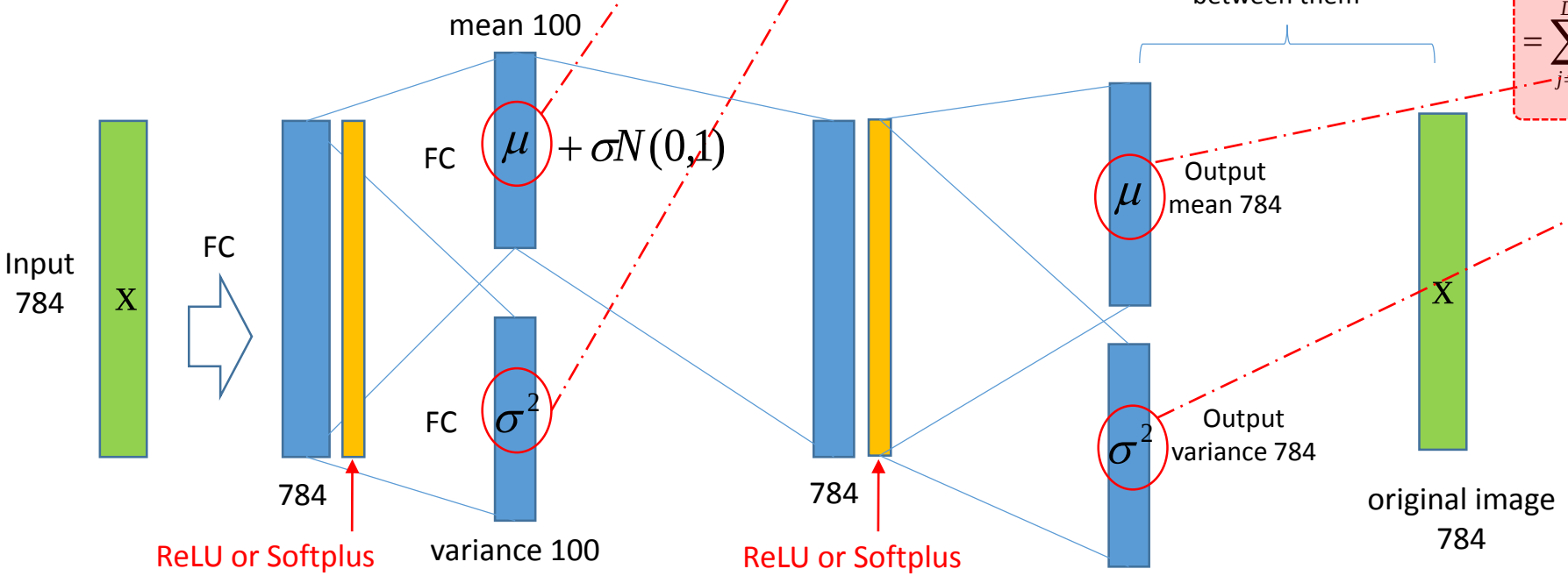
$$= \frac{1}{2}\sum_{j=1}^{J}\left(1+\log((\sigma_{z_j}^{(i)})^2) - (\mu_{z_j}^{(i)})^2 - (\sigma_{z_j}^{(i)})^2\right)$$

- Latent loss

1) Gaussian
2) Bernoulli

- Reconstruction loss

$$= \sum_{j=1}^{D}\left(\frac{1}{2}\log((\sigma_{x_j}^{(i)})^2) + \frac{(x_j^{(i)} - \mu_{x_j})^2}{2\sigma_{x_j}^2}\right)$$



5) Loss function: latent loss + reconstruction loss

D_KL(q(z|x) || P(z)): latent loss

```
latent_loss = -0.5 * tf.reduce_sum(- 1 – z_var +  z_mu**2 + tf.exp(z_var), 1)
```

$$= -\frac{1}{2}\sum_{j=1}^{J}\left(1+\log((\sigma_{z_j}^{(i)})^2)-(\mu_{z_j}^{(i)})^2-(\sigma_{z_j}^{(i)})^2\right)$$

log(p|z): reconstruction loss

- Gaussian: https://github.com/oduerr/dl_tutorial/blob/master/tensorflow/vae/vae_demo-2D.ipynb

```
recon_loss = tf.reduce_sum(0.5 * x_var + (tf.square(x - x_mu) / (2.0 * tf.exp(x_var))), 1)
```

$$= \sum_{j=1}^{D}\left(\frac{1}{2}\log((\sigma_{x_j}^{(i)})^2)+\frac{\left(x_j^{(i)}-\mu_{x_j}\right)^2}{2\sigma_{x_j}^2}\right)$$

- Bernoulli: https://jmetzen.github.io/2015-11-27/vae.html

```
recon_loss = tf.reduce_sum(-x * tf.log(1e-10 + x_mu) - (1 - x) * tf.log(1e-10 + 1 - x_mu), 1)
```

$$= x^{(i)}p^{(i)}+(1-x^{(i)})(1-p^{(i)})$$

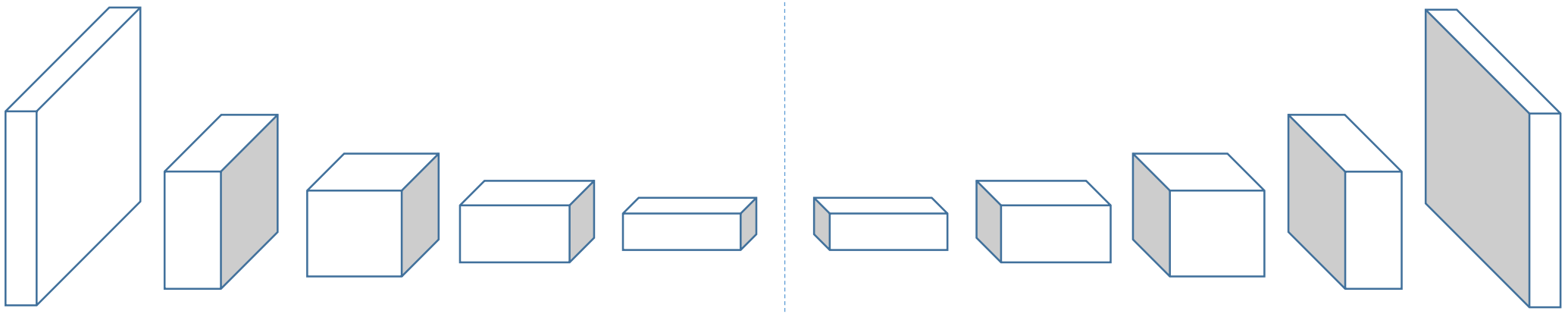
- Cross entropy: (vanilla_vae) https://github.com/wiseodd/generative-models

```
recon_loss = tf.reduce_sum(tf.nn.sigmoid_cross_entropy_with_logits(logits=logits, labels=x, 1))
```

Deep Convolutional GAN (DCGAN)

Whole structure of DCGAN implementation

- Deep Convolutional GAN (DCGAN)
  - https://arxiv.org/abs/1511.06434



Discriminator Network

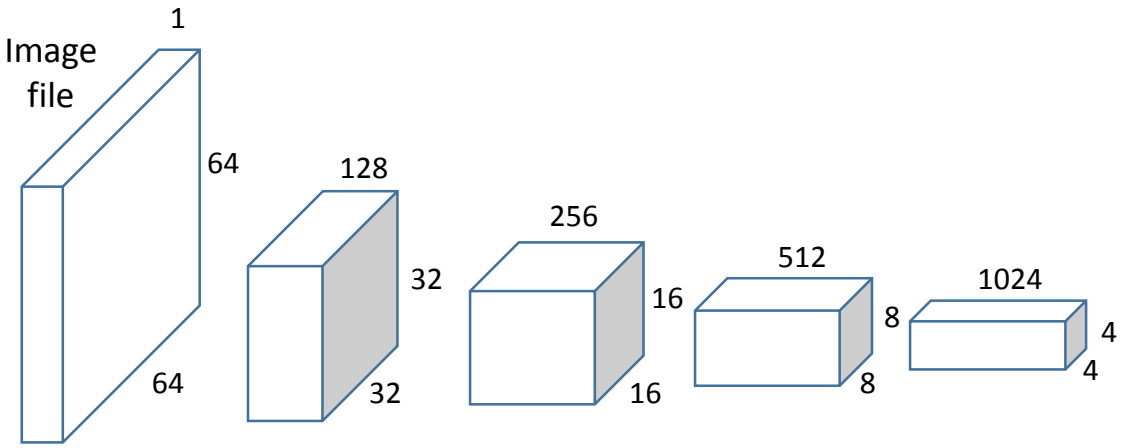
Generator Network

- Convolutional network
- Discriminator network:
  - Binary classification problem (fake or real)

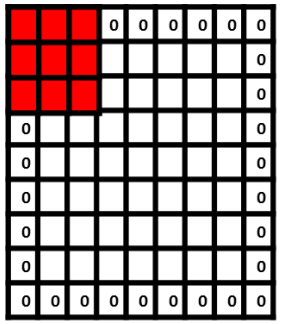
- De-convolutional network
- Generator network:
  - Generating a fake image close to real

Discriminator network

- All convolutional layers without pooling

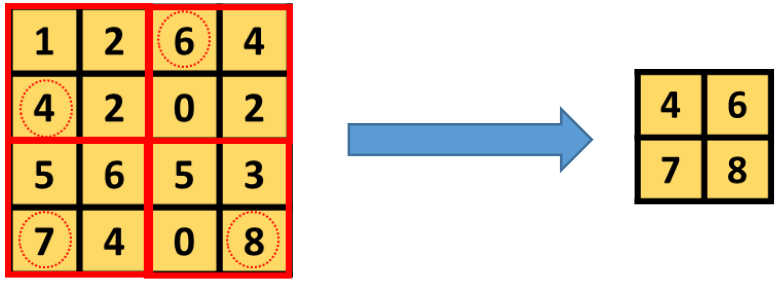


Convolutional layers without pooling



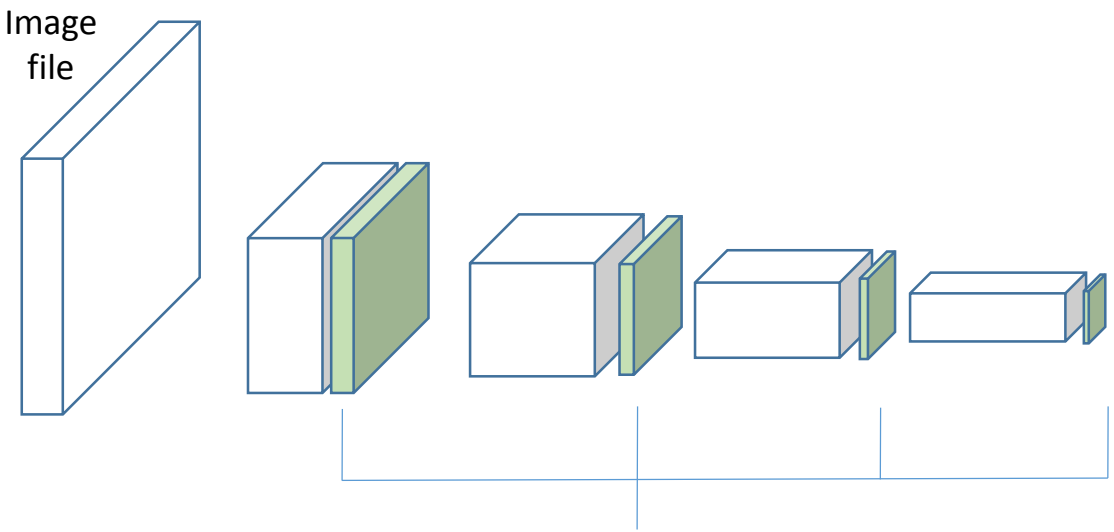
Convolution layer

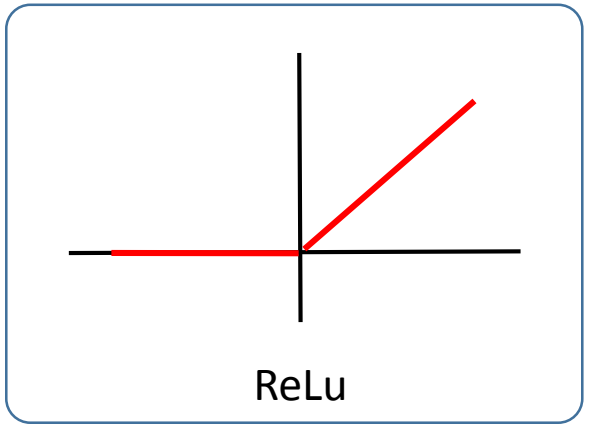
1. Pooling layer ?
- Reduction of the dimension
- Operation over each activation map independently (e.g., max or average pooling)

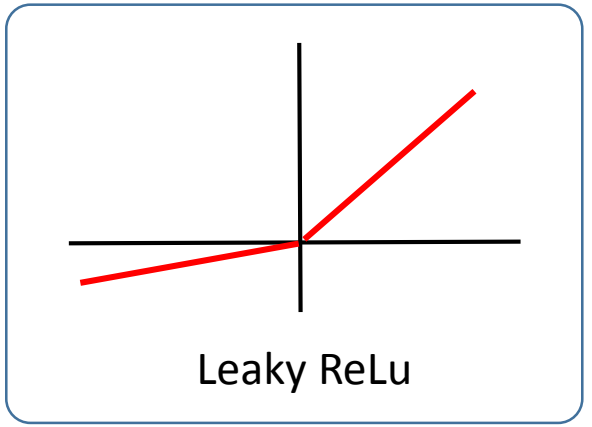


Discriminator network

- All convolutional layers without pooling
- Leaky ReLu activation function



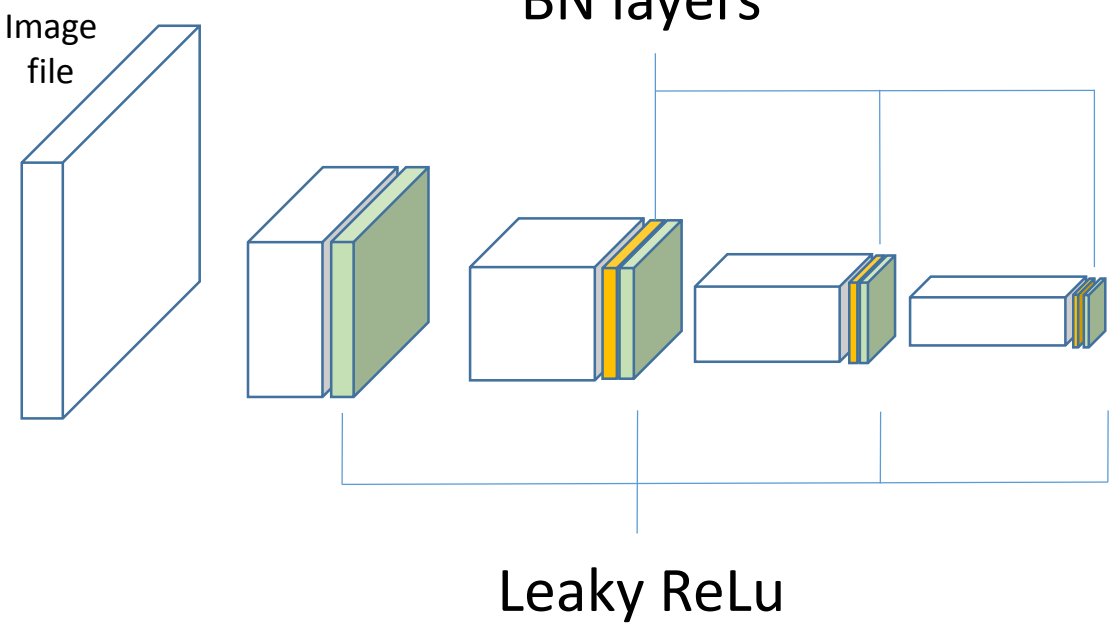




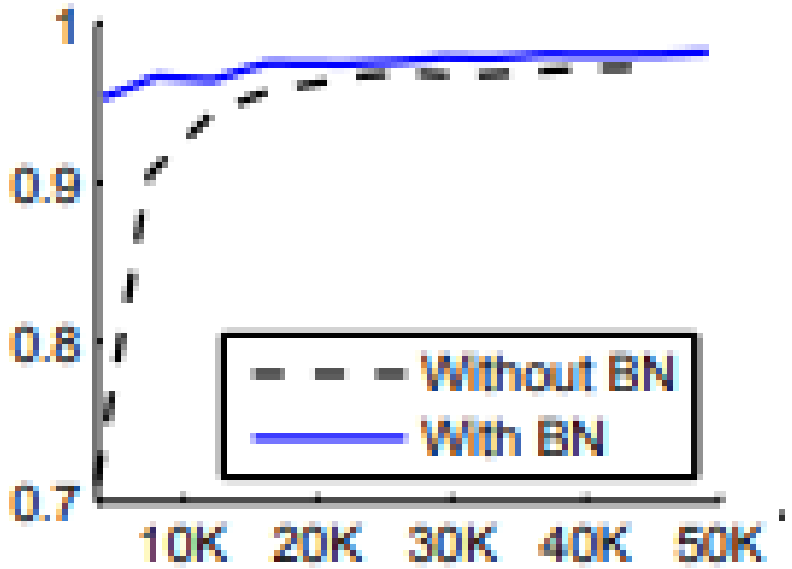
Discriminator network: Batch normalization

- ❑ All convolutional layers without pooling
- ❑ Leaky ReLu activation function
- ❑ Batch Normalization (BN) layers

tf.layers.batch_normalization



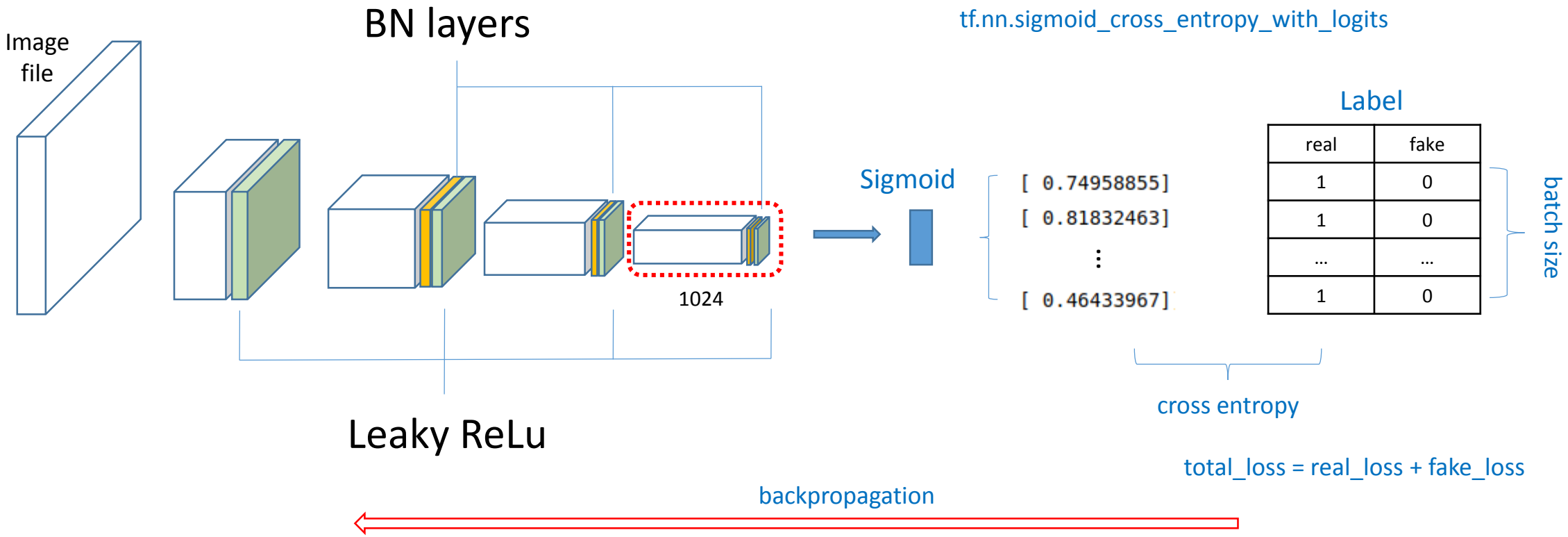
- ✓ Reduce learning time by normalizing input data (batch) to activation function
- ✓ Being considered as a standard part of DNN architectures



https://arxiv.org/pdf/1502.03167.pdf

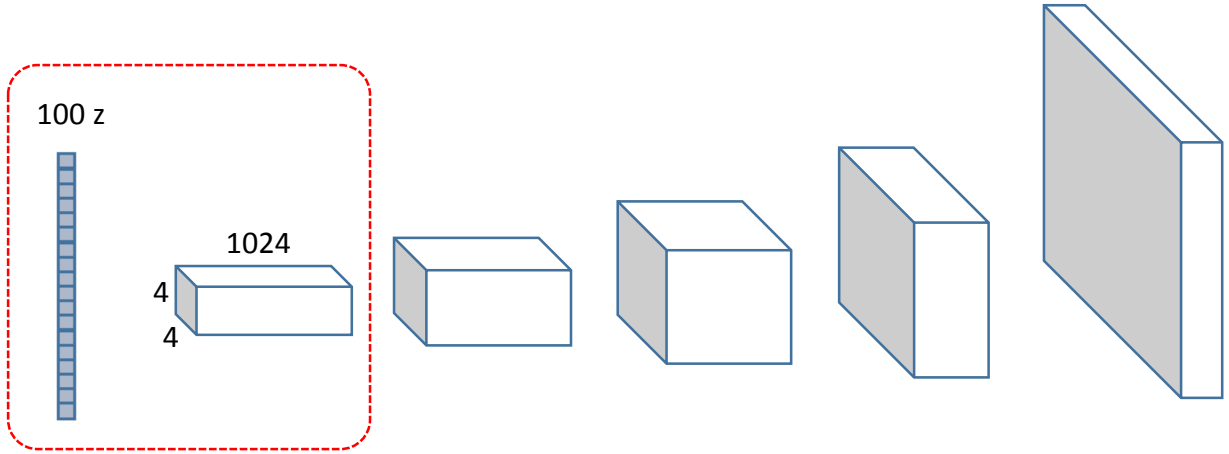
Discriminator network

- All convolutional layers without pooling
- Leaky ReLu activation function
- Batch Normalization (BN) layers



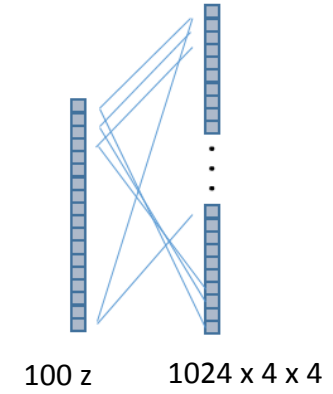
Generator network

- Fully connected (matrix multiplication)

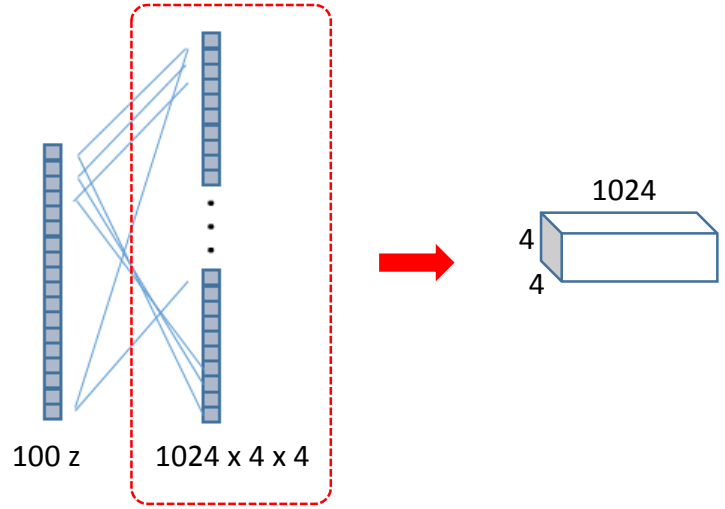



```
# 1st hidden layer
conv1 = tf.layers.conv2d_transpose(z, 1024, [4, 4], strides=(1, 1), padding='valid')
lrelu1 = lrelu(tf.layers.batch_normalization(conv1, training=isTrain), 0.2)
```

1) Create matrix multiplication (FC)

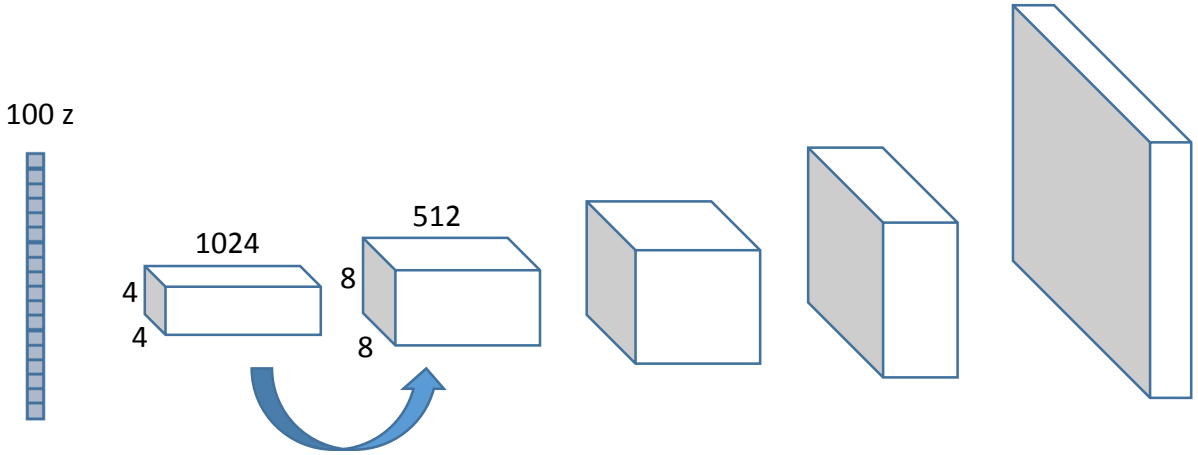


2) Reshape the flattened



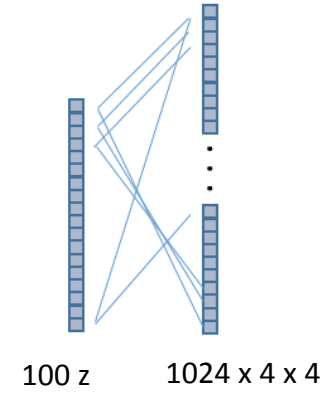
Generator network

- Fully connected (matrix multiplication)
- Deconvolution: transposed convolution

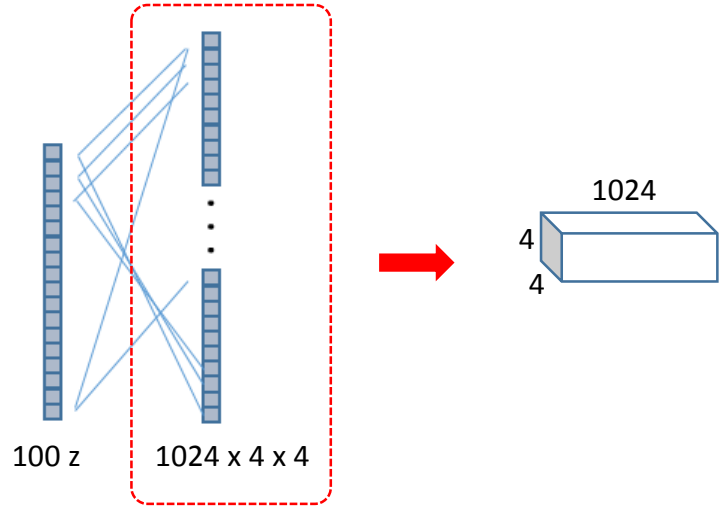



```
# 2nd hidden layer
conv2 = tf.layers.conv2d_transpose(lrelu1, 512, [4, 4], strides=(2, 2), padding='same')
lrelu2 = lrelu(tf.layers.batch_normalization(conv2, training=isTrain), 0.2)
```

1) Create matrix multiplication (FC)

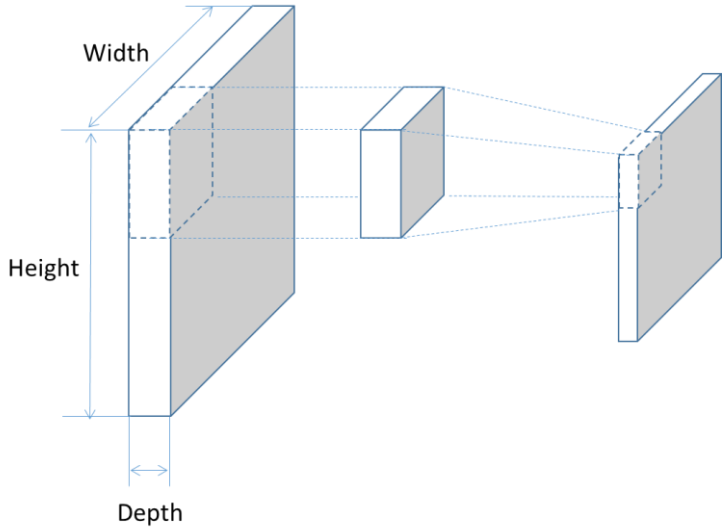


2) Reshape the flattened

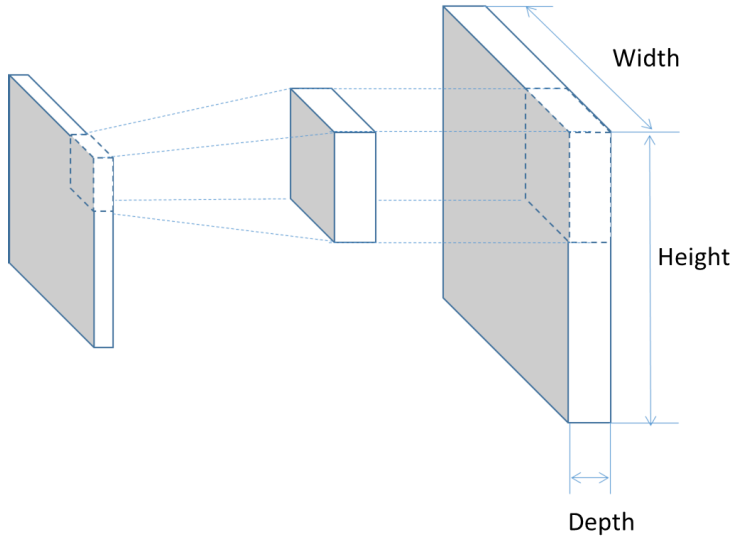


Generator network – transposed convolution

- "Deconvolution" is a misleading name and should be called "transposed convolutional layer".
- Transposed convolution works by swapping the forward and backward passes of a convolution.
- In Tensorflow, it is implemented in "conv2d_transpose()".

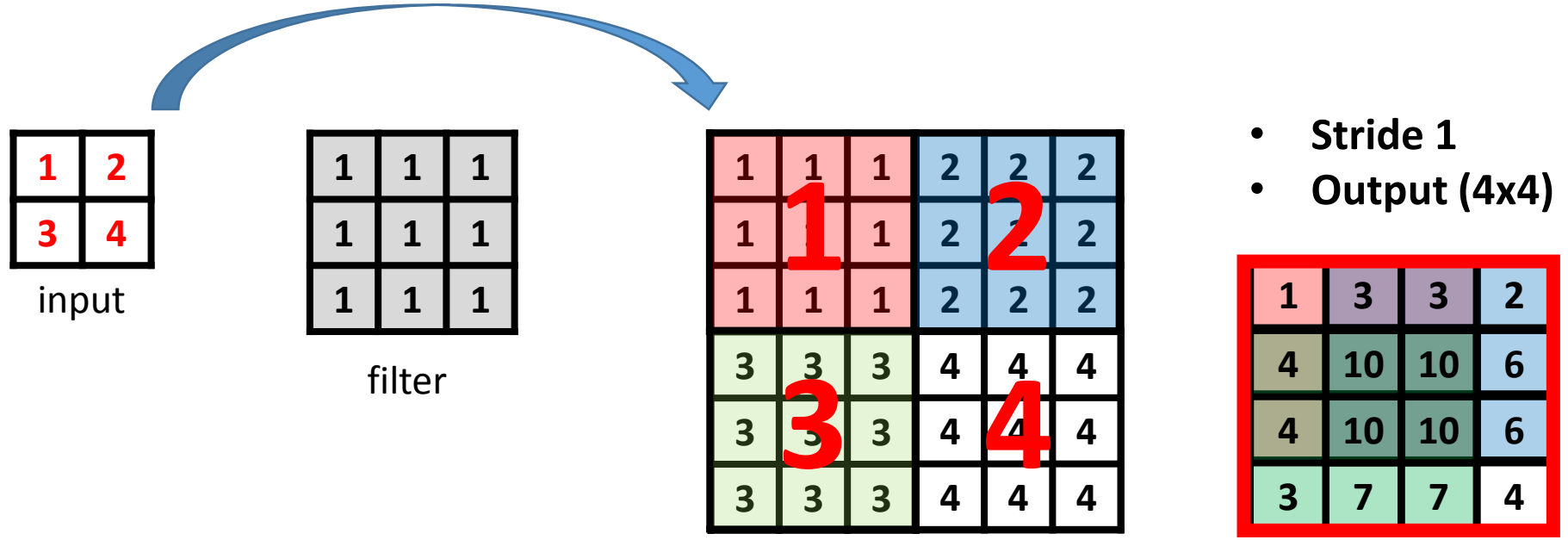


Convolution: down sample



Deconvolution: up sample

Generator network – transposed convolution: example (1)

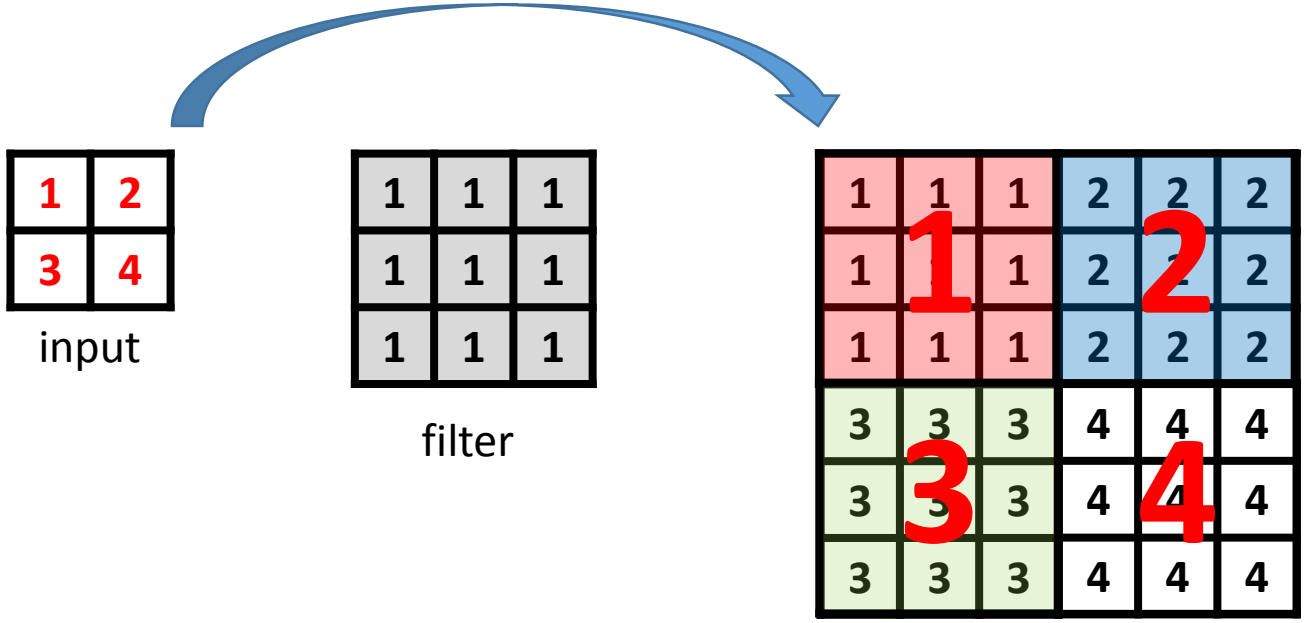


- **Stride 1**
- **Output (4x4)**

```
# input [1, 2, 2, 1] (batch_size, height, width, channels)      # stride [batch_size, height, width, channels]
input = tf.constant(np.array([[
    [[1], [2]],
    [[3], [4]]
]]), tf.float32)

# filter [3, 3, 1, 1] (height, width, input_channels, output_channels)
filter = tf.constant(np.array([
    [[1]], [[1]], [[1]]],
    [[1]], [[1]], [[1]]],
    [[1]], [[1]], [[1]]]
]), tf.float32)

conv = tf.nn.conv2d_transpose(input, filter, output_shape=(1, 4, 4, 1), strides=[1, 1, 1, 1], padding='VALID')

with tf.Session() as session:
    print (session.run(conv))
```

VALID padding
- Filter stays inside the output to produce the input.

Generator network – transposed convolution: example (2)



- **Stride 2**
- **Output (4x4)**

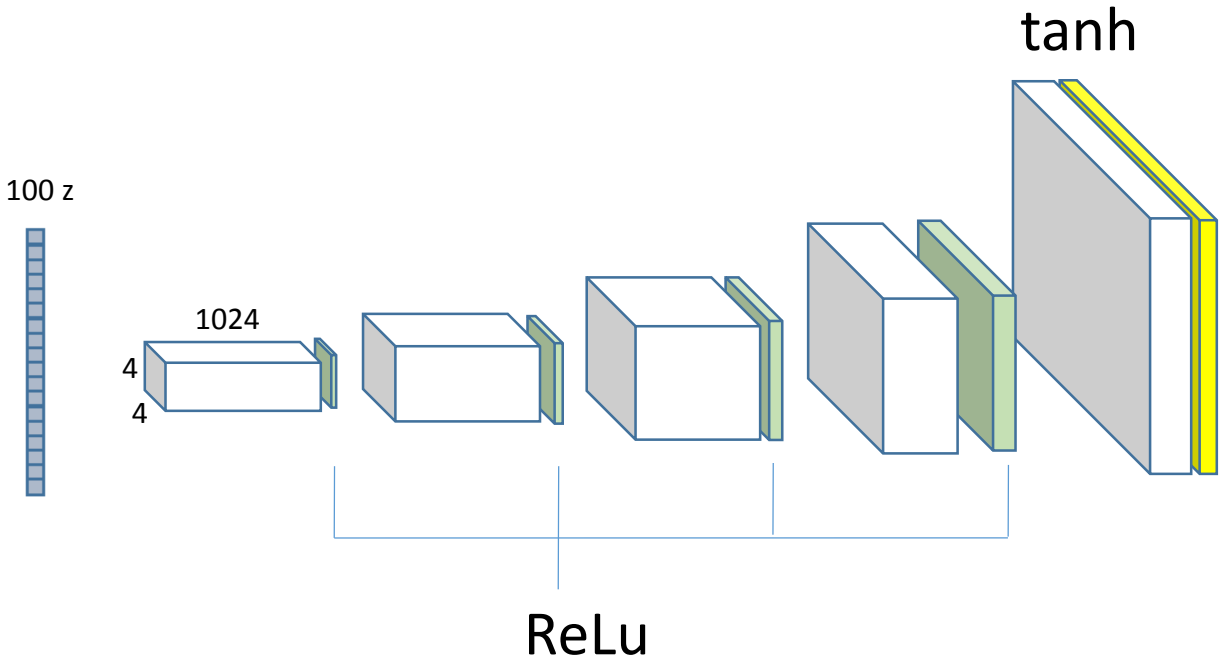
1	1	3	2	2
1	1	3	2	2
4	4	10	6	6
3	3	7	4	4
3	3	7	4	4

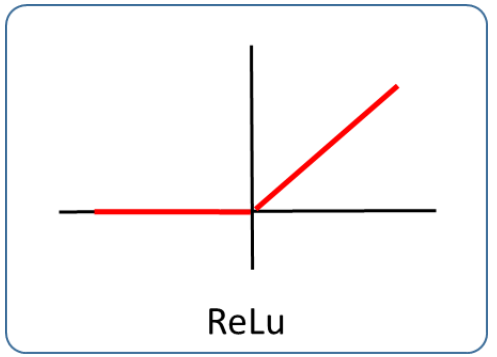
```
# input [1, 2, 2, 1] (batch_size, height, width, channels)
input = tf.constant(np.array([[
    [[1], [2]],
    [[3], [4]]
]]), tf.float32)

# filter [3, 3, 1, 1] (height, width, input_channels, output_channels)
filter = tf.constant(np.array([
    [[[1]], [[1]], [[1]]],
    [[[1]], [[1]], [[1]]],
    [[[1]], [[1]], [[1]]]
]), tf.float32)

conv = tf.nn.conv2d_transpose(input, filter, output_shape=(1, 4, 4, 1), strides=[1, 2, 2, 1], padding='SAME')

with tf.Session() as session:
    print (session.run(conv))
```

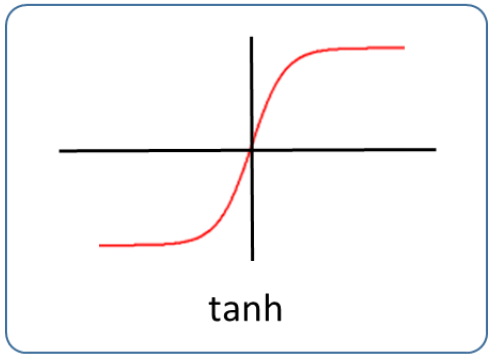
SAME padding
- Filter slips outside the output to produce the input.

Generator network

- Fully connected (matrix multiplication)
- Deconvolution: transposed convolution
- ReLu activation function in each layer except the last: “tanh” activation.

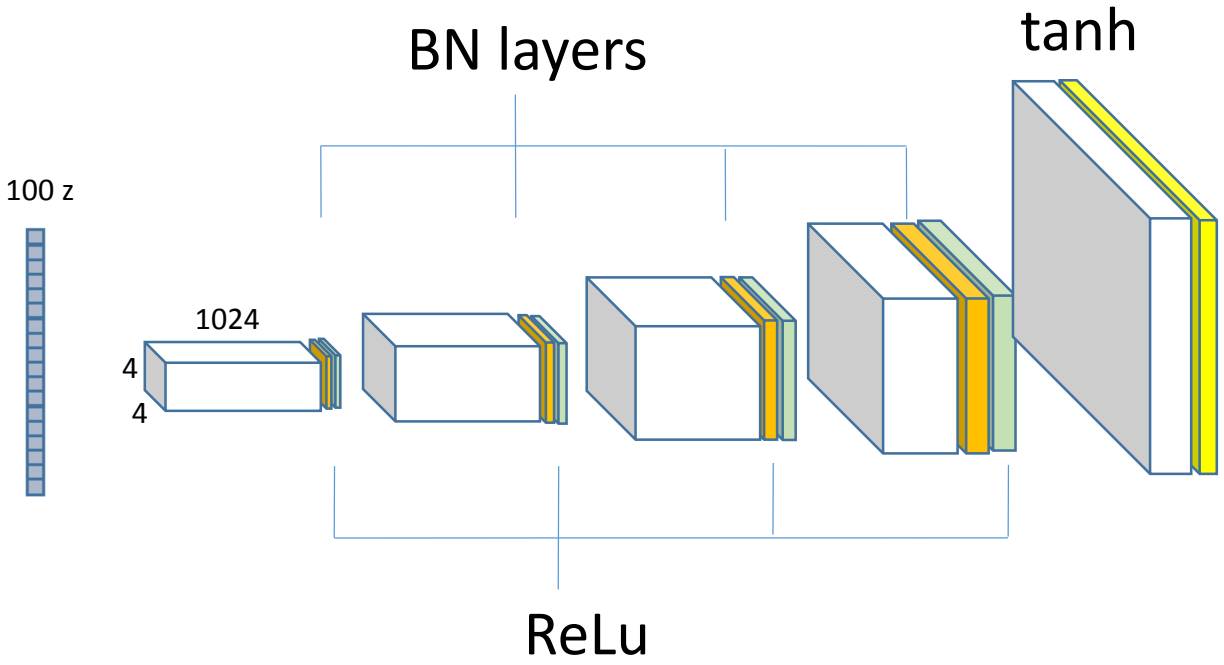


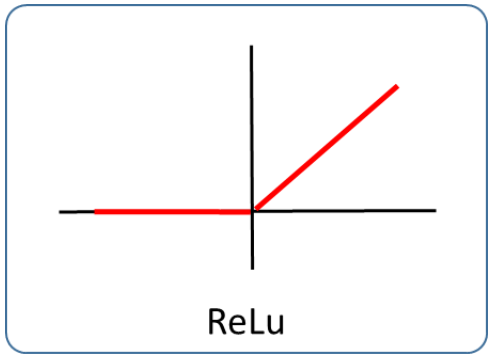


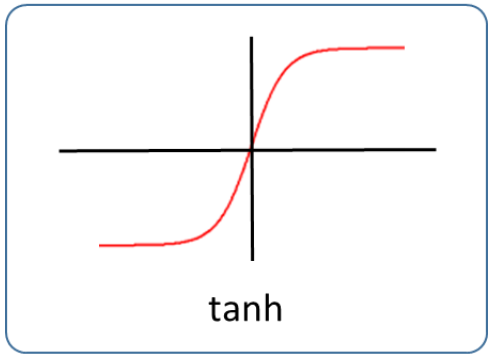


Generator network

- Fully connected (matrix multiplication)
- Deconvolution: transposed convolution
- ReLu activation function in each layer except the last: “tanh” activation
- Batch Normalization (BN) layers

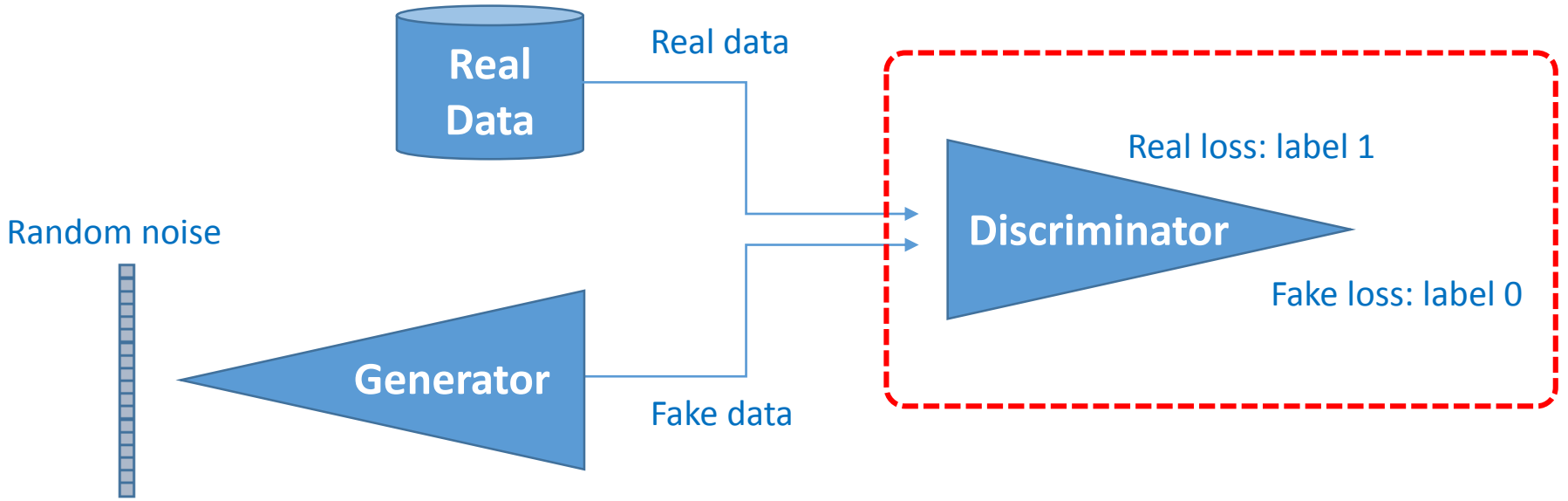






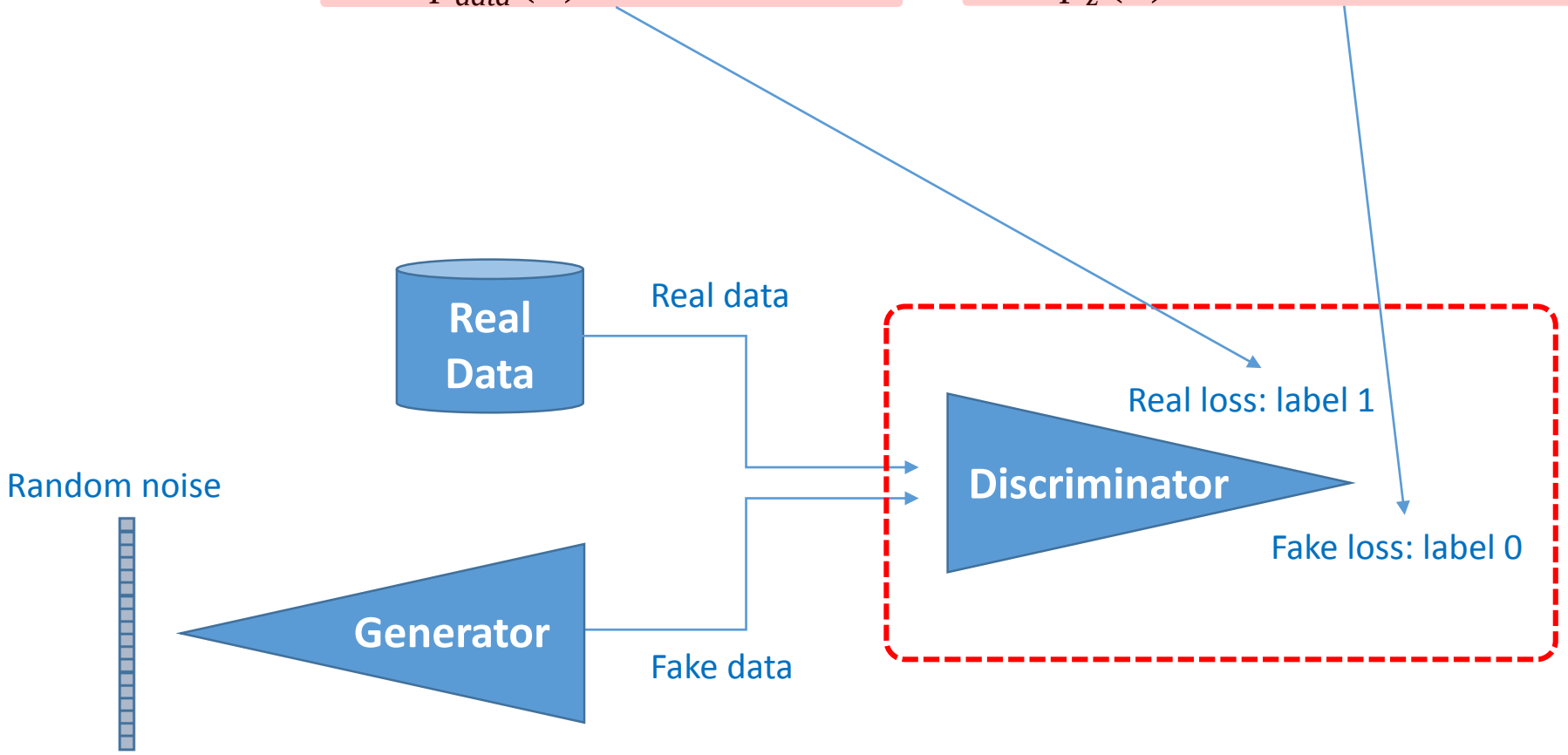
Operation of DCGAN: 1) training the discriminator

$$\max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1-D(G(z)))]$$



Operation of DCGAN: 1) training the discriminator

$$\max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

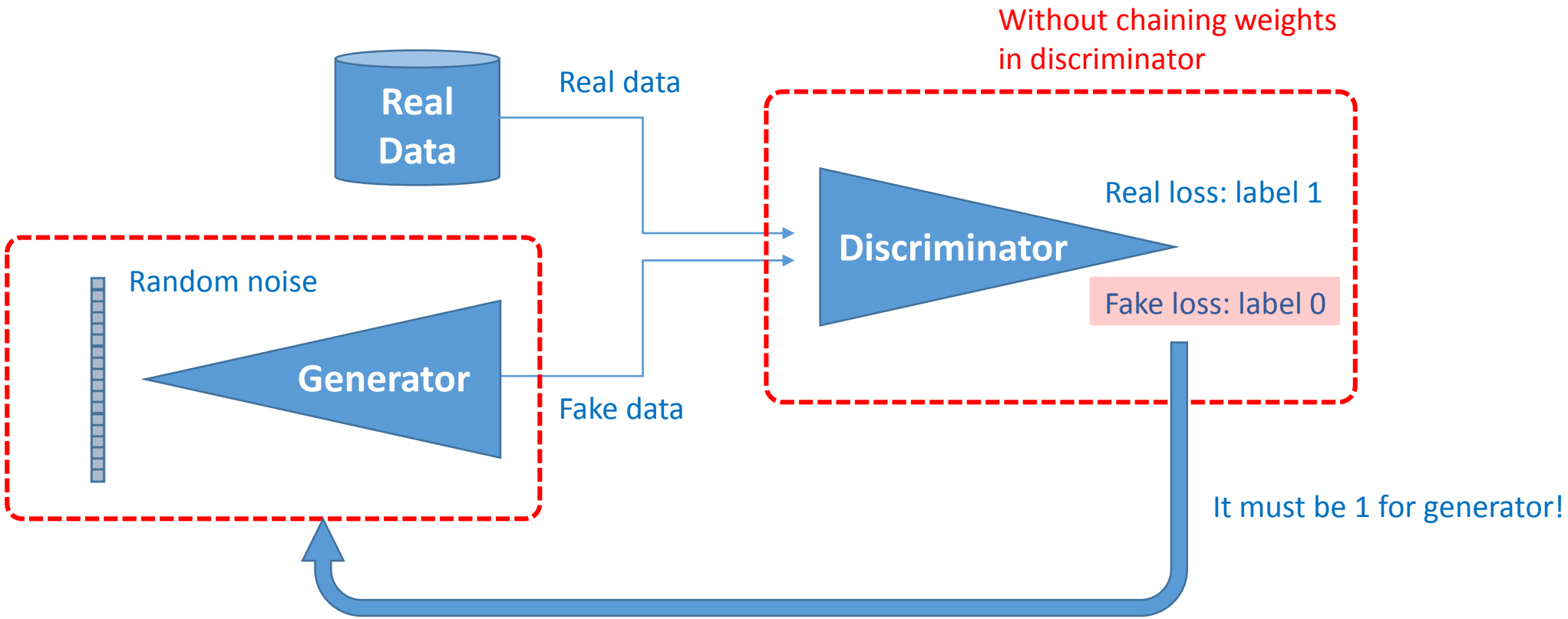


Operation of DCGAN: 2) training the generator

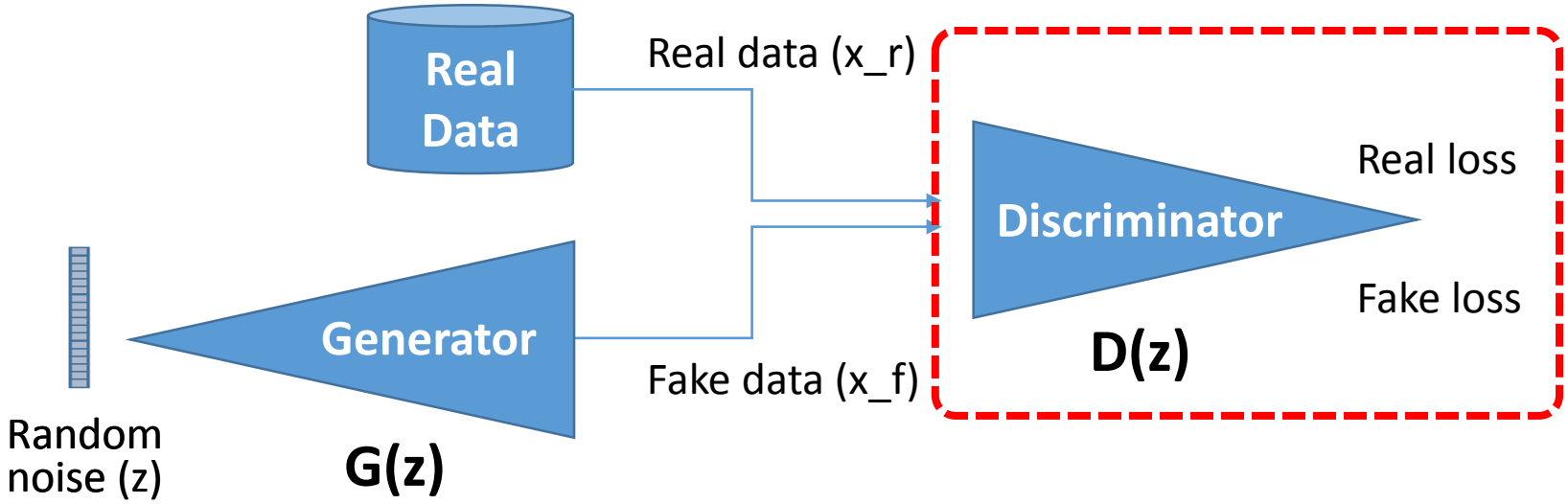
$$\min_G V(D,G) = E_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

$$\max_G V(D,G) = E_{z\sim p_z(z)}[\log(D(G(z)))]$$

- Same but the below is easy to training and so is used in practice.



Training the DCGAN: training discriminator



Data batch

x_r

Discriminator function (2)

function D(x_r)
return x_r_logits

Real_Loss function (3)

function L (logits=real, label=real)
return real_loss

+

Loss for D(x)

Generator function (1)

function G(z)
return x_f

Discriminator function (2)

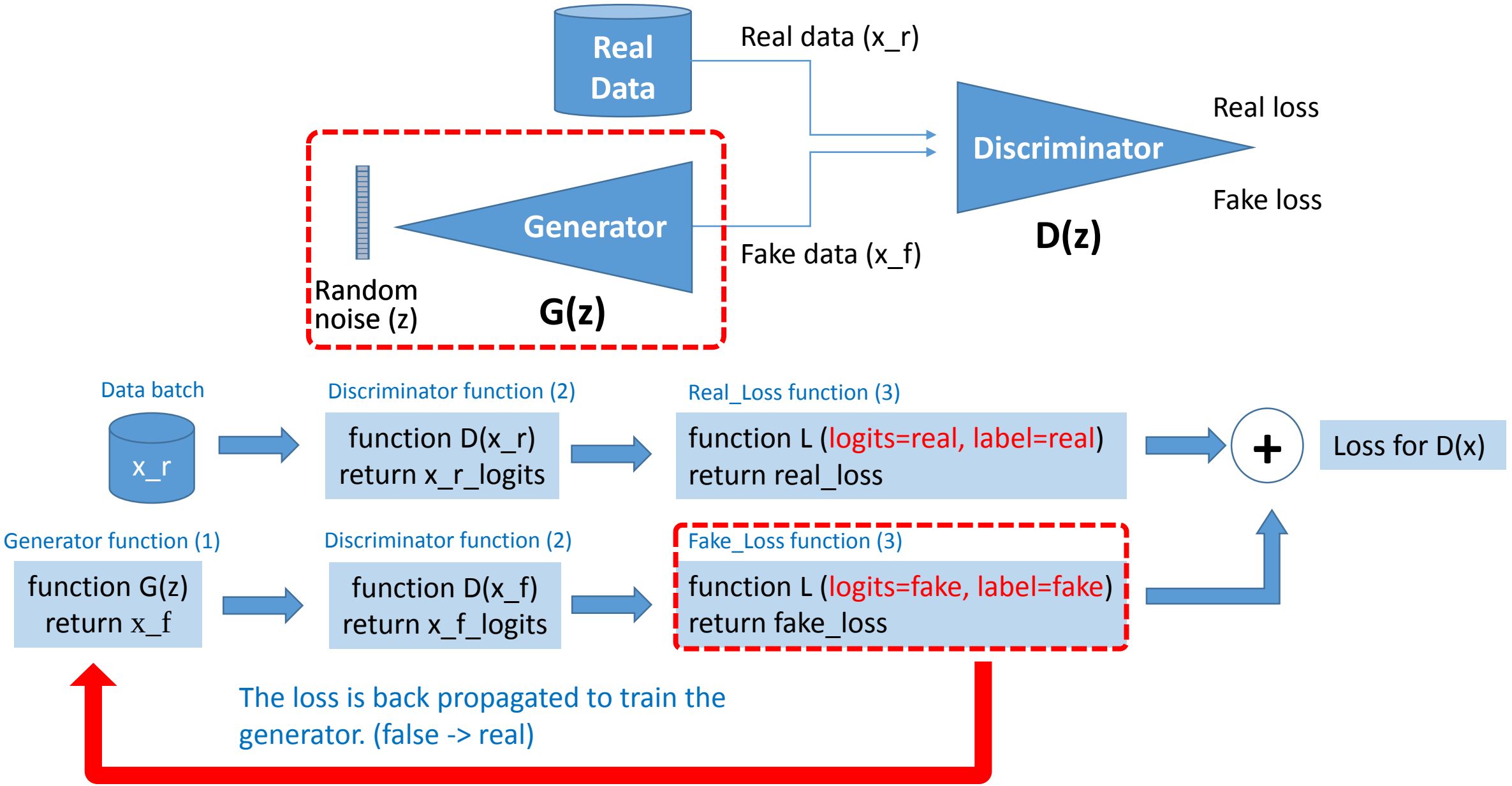
function D(x_f)
return x_f_logits

Fake_Loss function (3)

function L (logits=fake, label=fake)
return fake_loss

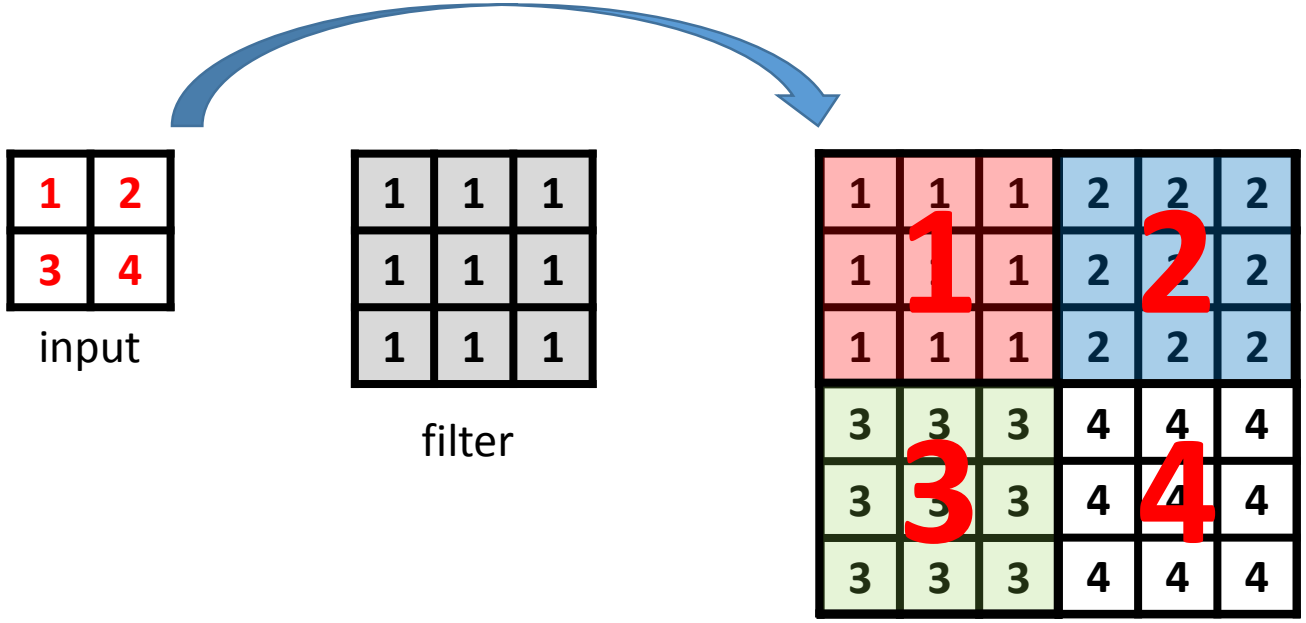
The loss is back propagated to train the discriminator.
(real -> real, false -> false)

Training the DCGAN: training generator



Backup Slides

Generator network – transposed convolution: example (2)

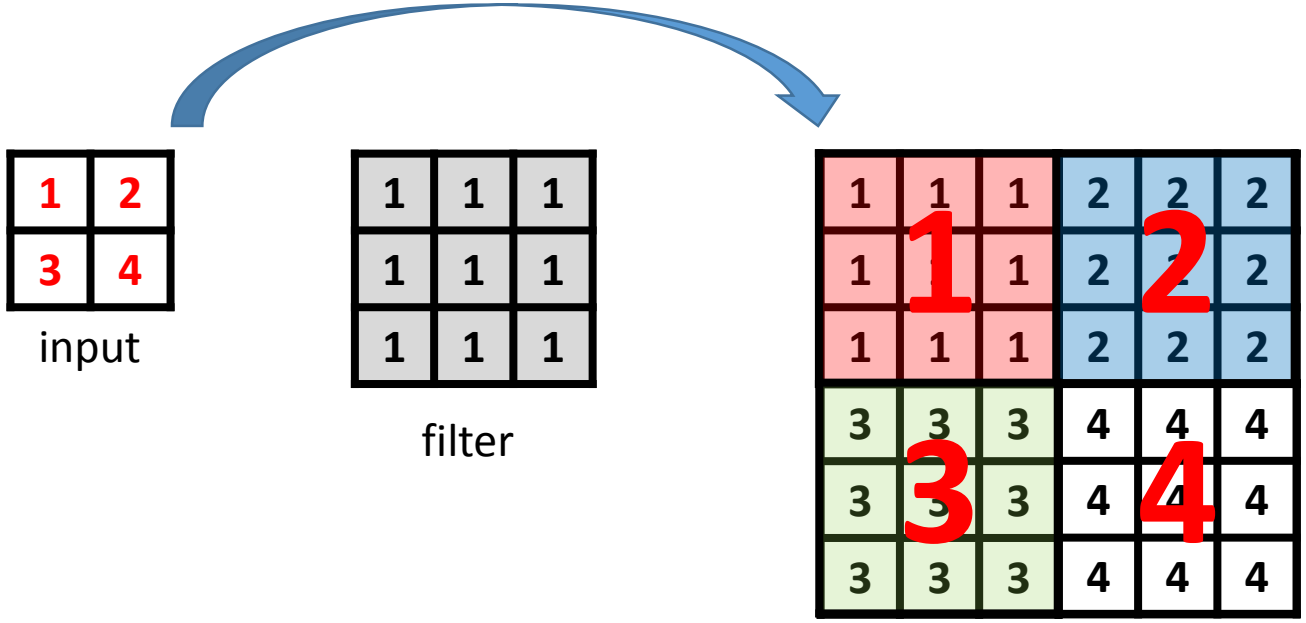


- **Stride 2**
- **Output (5x5)**

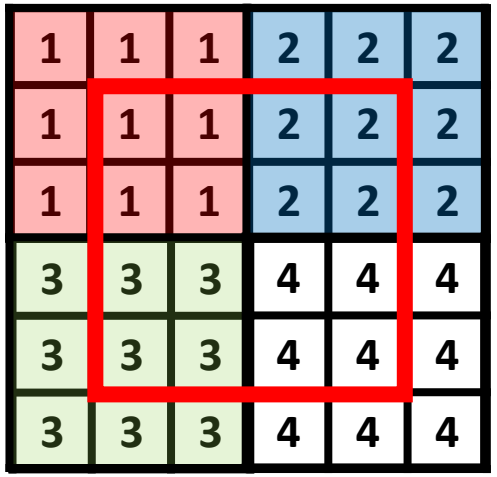
1	1	3	2	2
1	1	3	2	2
4	4	10	6	6
3	3	7	4	4
3	3	7	4	4

```
# input [1, 2, 2, 1] (batch_size, height, width, channels)
input = tf.constant(np.array([[
    [[1], [2]],
    [[3], [4]]
]]), tf.float32)

# filter [3, 3, 1, 1] (height, width, input_channels, output_channels)
filter = tf.constant(np.array([
    [[1]], [[1]], [[1]]],
    [[1]], [[1]], [[1]]],
    [[1]], [[1]], [[1]]]
]), tf.float32)

conv = tf.nn.conv2d_transpose(input, filter, output_shape=(1, 5, 5, 1), strides=[1, 2, 2, 1], padding='VALID')

with tf.Session() as session:
    print (session.run(conv))
```

Generator network – transposed convolution: example (3)

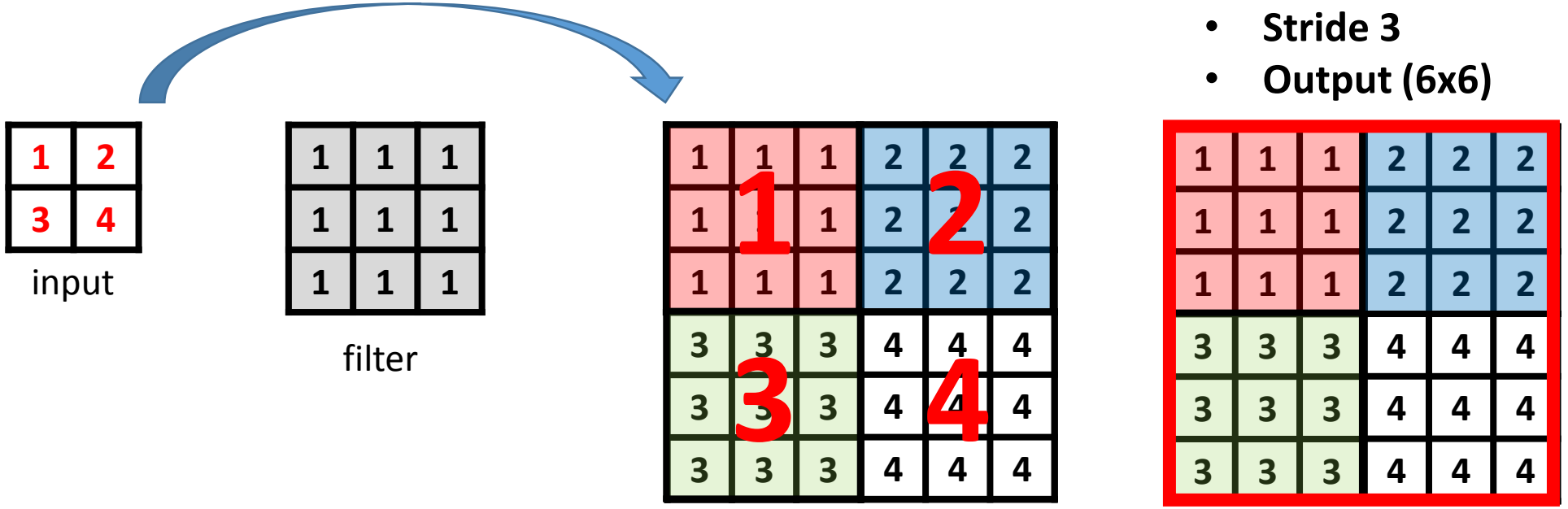


- **Stride 3**
- **Output (4x4)**



```
# input [1, 2, 2, 1] (batch_size, height, width, channels)
input = tf.constant(np.array([[
    [[1], [2]],
    [[3], [4]]
]]), tf.float32)

# filter [3, 3, 1, 1] (height, width, input_channels, output_channels)
filter = tf.constant(np.array([
    [[1]], [[1]], [[1]]],
    [[1]], [[1]], [[1]]],
    [[1]], [[1]], [[1]]]
]), tf.float32)

conv = tf.nn.conv2d_transpose(input, filter, output_shape=(1, 4, 4, 1), strides=[1, 3, 3, 1], padding='SAME')

with tf.Session() as session:
    print (session.run(conv))
```

Generator network – transposed convolution: example (4)



- **Stride 3**
- **Output (6x6)**

```
# input [1, 2, 2, 1] (batch_size, height, width, channels)
input = tf.constant(np.array([[
    [[1], [2]],
    [[3], [4]]
]]), tf.float32)

# filter [3, 3, 1, 1] (height, width, input_channels, output_channels)
filter = tf.constant(np.array([
    [[1]], [[1]], [[1]]],
    [[1]], [[1]], [[1]]],
    [[1]], [[1]], [[1]]]
]), tf.float32)

conv = tf.nn.conv2d_transpose(input, filter, output_shape=(1, 6, 6, 1), strides=[1, 3, 3, 1], padding='VALID')

with tf.Session() as session:
    print (session.run(conv))
```