

Practice: VAE and GAN

Hao Dong

Peking University

Practice: VAE + GAN

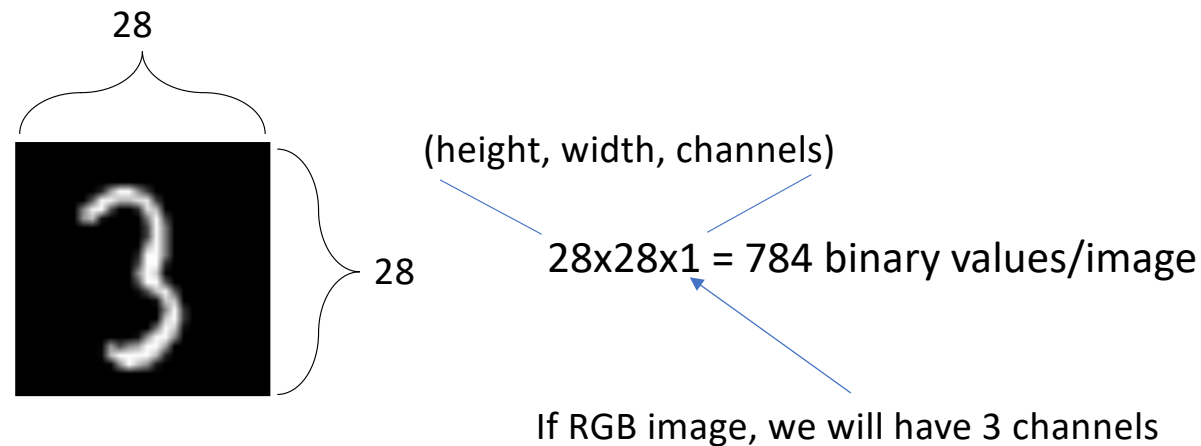
- Hello World: MNIST Classification
- Introduction of VAE
- VAE Architecture
- VAE Training
- VAE Interpolation
- Sampling
- Introduction of DCGAN
- DCGAN Architecture
- DCGAN Training
- DCGAN Interpolation

- Hello World: MNIST Classification
- Introduction of VAE
- VAE Architecture
- VAE Training
- VAE Interpolation
- Sampling
- Introduction of DCGAN
- DCGAN Architecture
- DCGAN Training
- DCGAN Interpolation

Hello World: MNIST Classification



MNIST dataset



- **Image X is a list of row vectors:**

```
>>> X_train, y_train, X_val, y_val, X_test, y_test = tl.files.load_mnist_dataset(shape=(-1, 784))
>>> print(X_train.shape)
... (50000, 784)
```

- **Image X is a list of images:**

```
>>> X_train, y_train, X_val, y_val, X_test, y_test = tl.files.load_mnist_dataset(shape=(-1, 28, 28, 1))
>>> print(X_train.shape)
... (50000, 28, 28, 1)
```

Hello World: MNIST Classification

- Simple Iteration

```
>>> X = np.asarray(['a','a'], ['b','b'], ['c','c'], ['d','d'], ['e','e'], ['f','f'])
>>> y = np.asarray([0,1,2,3,4,5])
>>> for batch in tl.iterate.minibatches(inputs=X, targets=y, batch_size=2, shuffle=False):
>>>     print(batch)
... (array(['a', 'a'], ['b', 'b']), array([0, 1]))
... (array(['c', 'c'], ['d', 'd']), array([2, 3]))
... (array(['e', 'e'], ['f', 'f']), array([4, 5]))
```

Hello World: MNIST Classification

- Dataset API

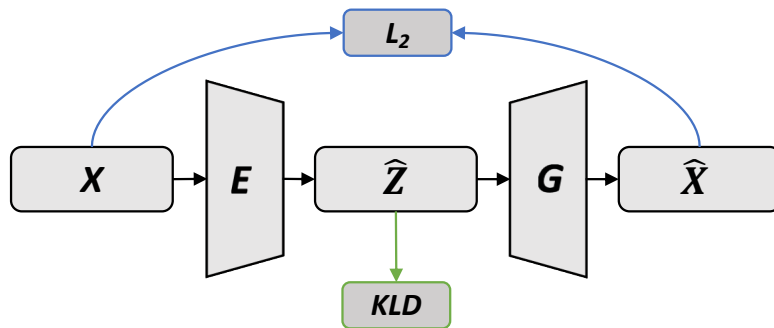
```
def get_mnist(batch_size):
    X_train, y_train, X_val, y_val, X_test, y_test = tf.keras.datasets.mnist.load_data()
    train_set = X_train
    length = len(train_set)

    def generator_train():
        for img in train_set:
            yield (img - 0.5) / 0.5 # a Tensor with values range in [-1, 1]

    train_ds = tf.data.Dataset.from_generator(generator_train, output_types=tf.float32)
    ds = train_ds.batch(batch_size)
    ds = ds.prefetch(buffer_size=2)
    return ds, length
```

- Hello World: MNIST Classification
- **Introduction of VAE**
 - VAE Architecture
 - VAE Training
 - VAE Interpolation
 - Sampling
- Introduction of DCGAN
 - DCGAN Architecture
 - DCGAN Training
 - DCGAN Interpolation

Introduction of VAE



- Two network architectures
- Two loss functions
- Reparameterization trick

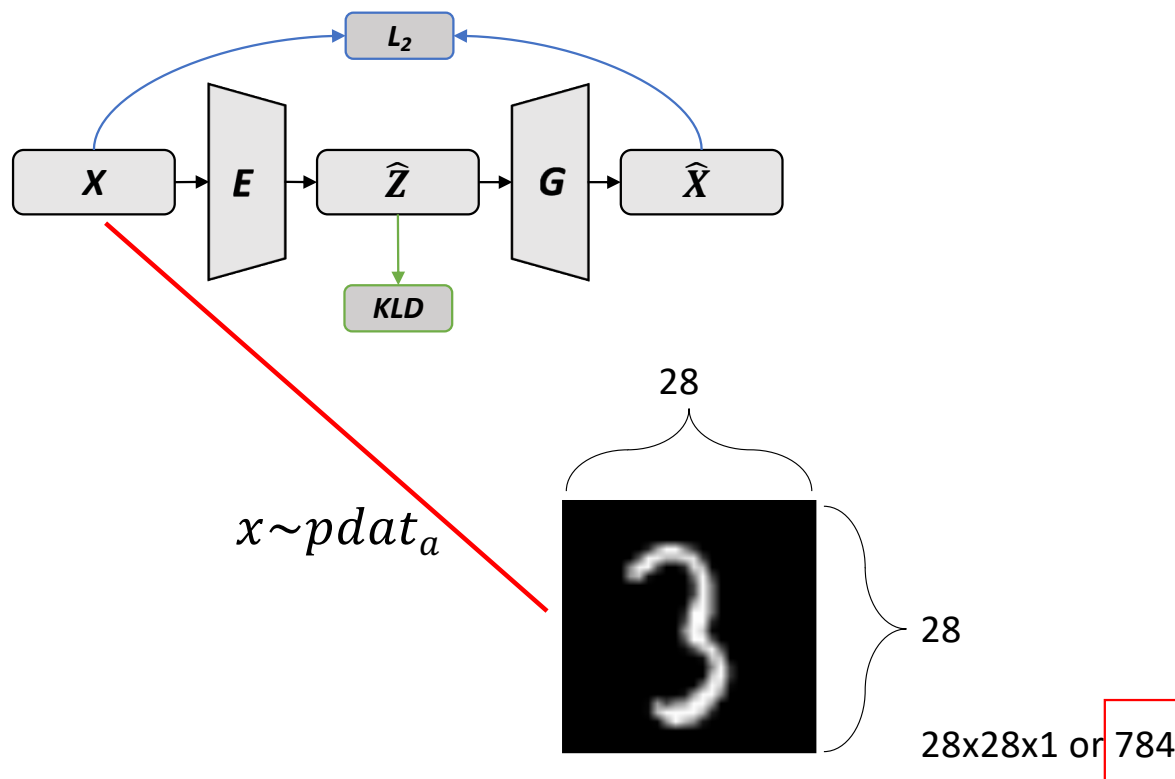
$$\tilde{\mathcal{L}}^B(\theta, \phi; \mathbf{x}^{(i)}) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + \frac{1}{L} \sum_{l=1}^L (\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}))$$

where $\mathbf{z}^{(i,l)} = g_\phi(\epsilon^{(i,l)}, \mathbf{x}^{(i)})$ and $\epsilon^{(l)} \sim p(\epsilon)$

- Hello World: MNIST Classification
- Introduction of VAE
- **VAE Architecture**
- VAE Training
- VAE Interpolation
- Sampling

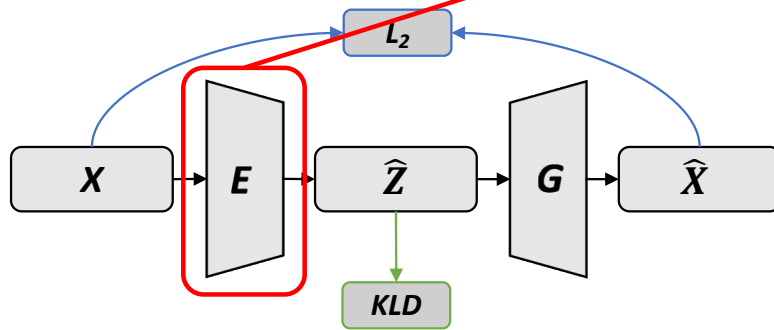
- Introduction of DCGAN
- DCGAN Architecture
- DCGAN Training
- DCGAN Interpolation

VAE Architecture



VAE Architecture

- Architecture of Encoder



```
def get_encoder(self, batch_size, origin_units, hidden_units, latent_units):
    init = tf.initializers.he_uniform()
    ni = Input((batch_size, origin_units))
    nn = Dense(hidden_units, act=tf.nn.relu, W_init=init, b_init=init)(ni)

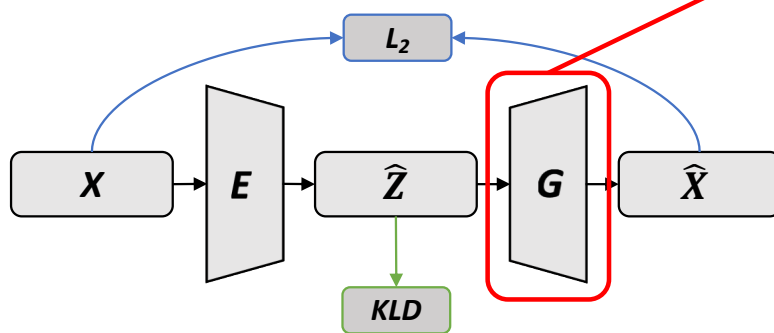
    mean = Dense(latent_units, W_init=init, b_init=init)(nn)
    log_sigma = Dense(latent_units, W_init=init, b_init=init)(nn)

    def sample(data):
        mean, log_sigma = data
        stddev = 0.5 * tf.exp(log_sigma)
        out = mean + stddev * tf.random.normal(mean.shape)
        return out

    z = Lambda(sample)([mean, log_sigma])
    return tf.models.Model(inputs=ni, outputs=[z, mean, log_sigma])
```

VAE Architecture

- Architecture of Decoder/Generator



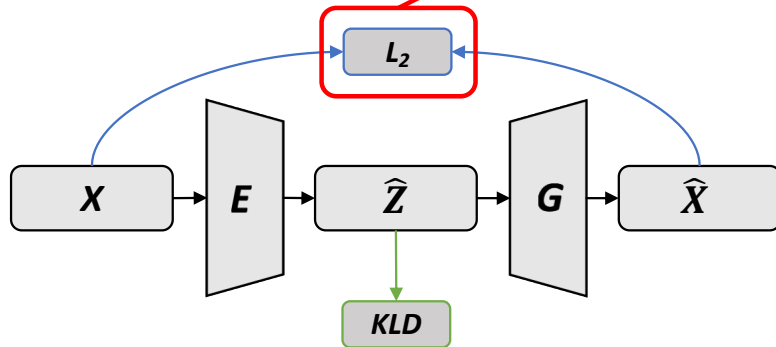
```
def get_decoder(self, batch_size, origin_units, hidden_units, latent_units):  
    init = tf.initializers.he_uniform()  
    ni = Input((batch_size, latent_units))  
    nn = Dense(hidden_units, act=tf.nn.relu, w_init=init, b_init=init)(ni)  
    no = Dense(origin_units, act=tf.nn.sigmoid, w_init=init, b_init=init)(nn)  
    return tl.models.Model(inputs=ni, outputs=no)
```

- Hello World: MNIST Classification
- Introduction of VAE
- VAE Architecture
- **VAE Training**
- VAE Interpolation
- Sampling

- Introduction of DCGAN
- DCGAN Architecture
- DCGAN Training
- DCGAN Interpolation

VAE Training

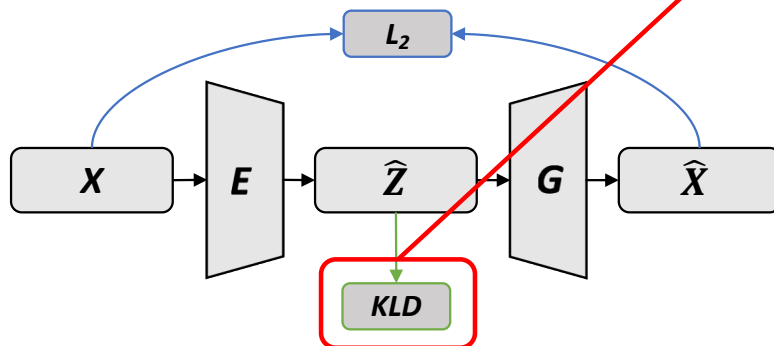
Reconstruction Loss



```
def recon_loss(x, y, k):  
    loss = k * tf.losses.binary_crossentropy(x, y)  
    loss = tf.reduce_sum(loss)  
    return loss
```

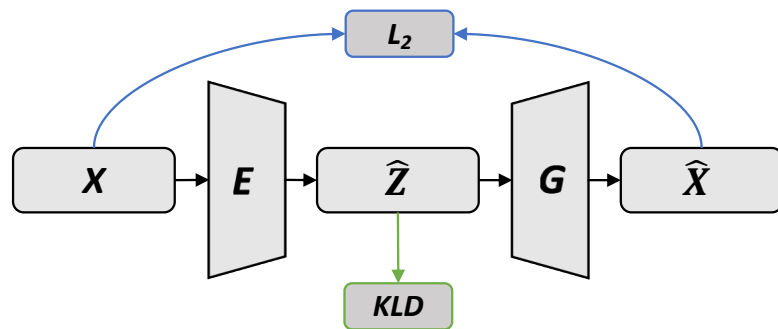
VAE Training

KL-Divergence loss



```
def KL_loss(mu, log_sigma):  
    loss = -0.5 * tf.reduce_sum(1 + log_sigma - tf.exp(log_sigma) - mu**2)  
    loss = tf.reduce_sum(loss)  
    return loss
```

VAE Training



Training Pipeline

```
for epoch in range(flags.n_epoch):
    for step, batch_images in enumerate(images):
        if batch_images.shape[0] != flags.batch_size:
            break
        step_time = time.time()
        last_batch = batch_images

        with tf.GradientTape(persistent=True) as tape:
            reconstr_img, n_mean, n_log_sigma = vae(batch_images)

            reconstr_loss = recon_loss(batch_images, reconstr_img, k)
            latent_loss = KL_loss(n_mean, n_log_sigma)

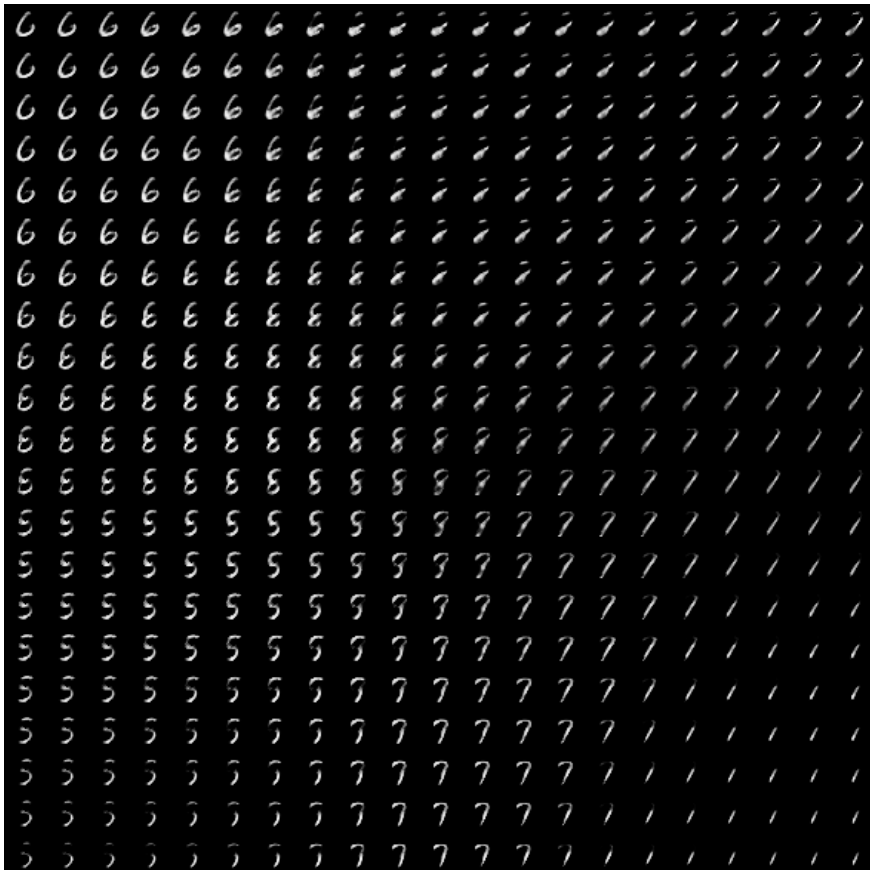
            loss = tf.add(reconstr_loss, latent_loss)

        grad = tape.gradient(loss, vae.trainable_weights)
        vae_optimizer.apply_gradients(zip(grad, vae.trainable_weights))
        del tape
```


- Hello World: MNIST Classification
- Introduction of VAE
- VAE Architecture
- VAE Training
- VAE Interpolation
- Sampling

- Introduction of DCGAN
- DCGAN Architecture
- DCGAN Training
- DCGAN Interpolation

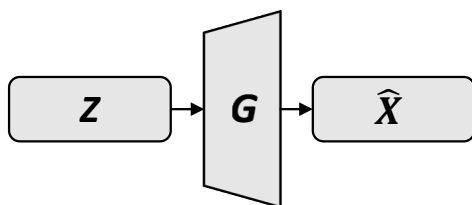
VAE Interpolation



```
if flags.z_dim == 2:  
    z = []  
    for x in range(21):  
        for y in range(21):  
            z.append([(x-10)/10., (y-10)/10.])  
    z = np.array(z).astype(np.float32)  
    vae.eval()  
    gen_result = vae.generate(z)  
    tl.visualize.save_images(gen_result.numpy().reshape([-1, 28, 28, 1]), [21, 21],  
                             '{}/feature_visualization.png'.format(flags.sample_dir))
```

- Hello World: MNIST Classification
- Introduction of VAE
- VAE Architecture
- VAE Training
- VAE Interpolation
- **Sampling**
- Introduction of DCGAN
- DCGAN Architecture
- DCGAN Training
- DCGAN Interpolation

Sampling

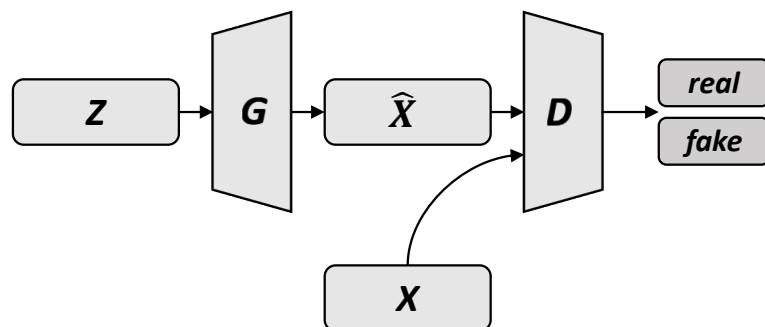


```
def generate(self, z_in):  
    assert z_in.shape[-1] == self.latent_units  
  
    return self.decoder(z_in)
```

```
vae.eval()  
tmp_result, tmp_mean, tmp_logvar = vae(last_batch) # get the reconstruction results  
z = np.random.normal(0.0, 1.0, (flags.sample_size, flags.z_dim)).astype(np.float32)  
gen_result = vae.generate(z) # get a randomly generated results  
vae.train()  
tl.visualize.save_images(tmp_result.numpy().reshape([-1, 28, 28, 1]), [num_tiles, num_tiles],  
                          '{}/train_{:02d}.png'.format(flags.sample_dir, epoch))  
tl.visualize.save_images(gen_result.numpy().reshape([-1, 28, 28, 1]), [num_tiles, num_tiles],  
                          '{}/generate_{:02d}.png'.format(flags.sample_dir, epoch))
```

- Hello World: MNIST Classification
- Introduction of VAE
- VAE Architecture
- VAE Training
- VAE Interpolation
- Sampling
- **Introduction of DCGAN**
- DCGAN Architecture
- DCGAN Training
- DCGAN Interpolation

Introduction of DCGAN



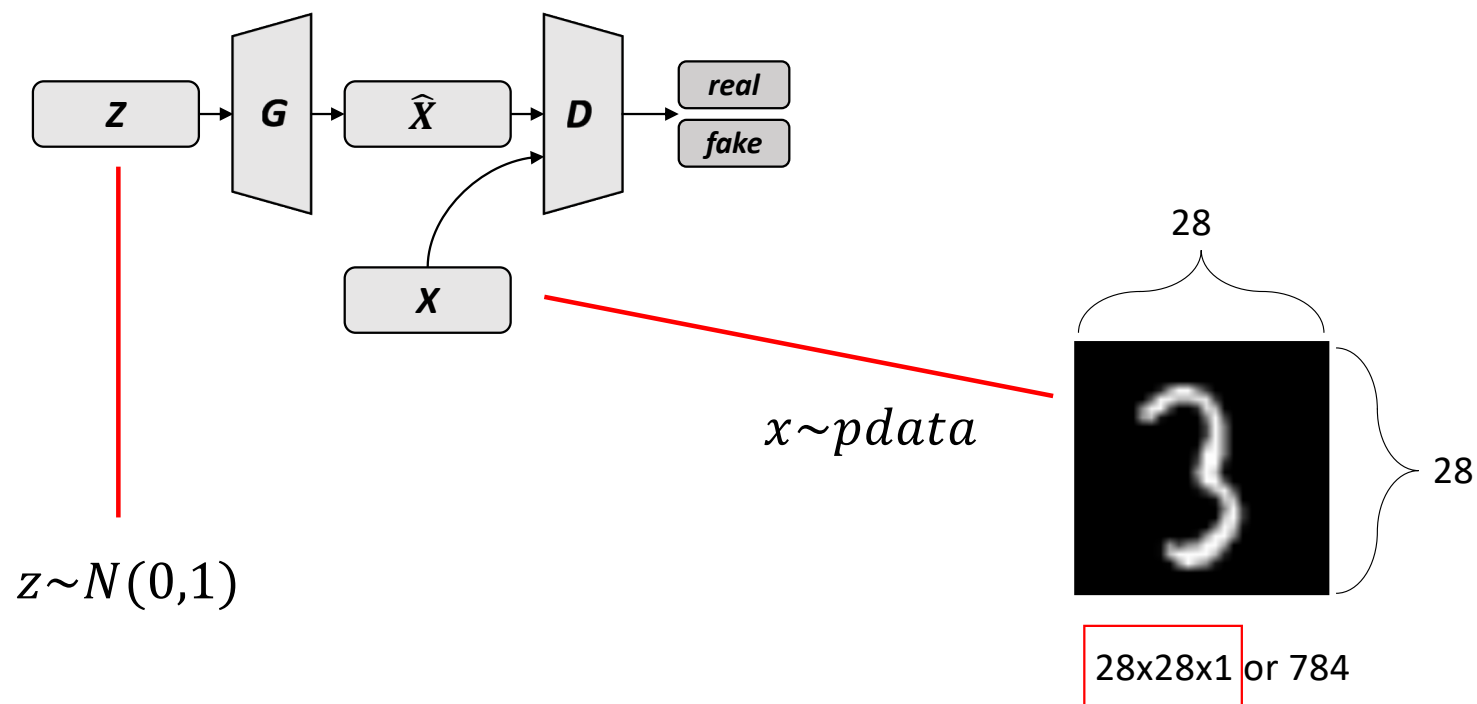
- Two network architectures
- Two loss functions

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

- Hello World: MNIST Classification
- Introduction of VAE
- VAE Architecture
- VAE Training
- VAE Interpolation
- Sampling

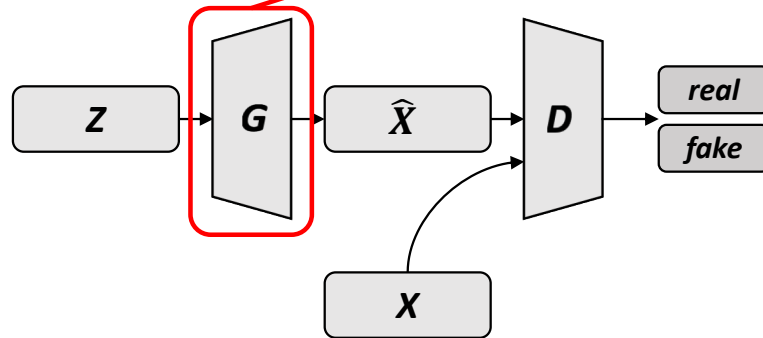
- Introduction of DCGAN
- **DCGAN Architecture**
- DCGAN Training
- DCGAN Interpolation

DCGAN Architecture



DCGAN Architecture

Architecture of generator

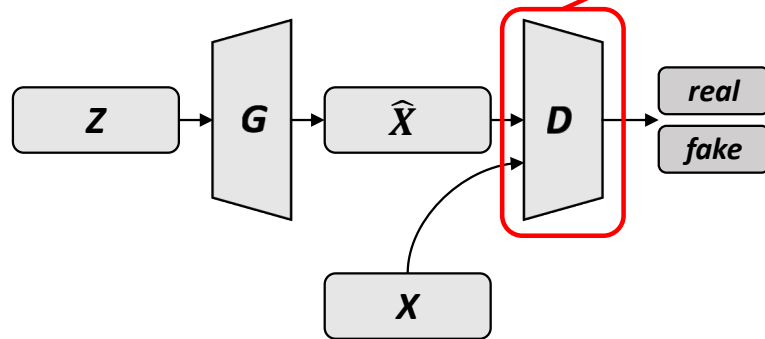


```
def get_generator(shape, gf_dim=64, o_size=32, o_channel=3):
    image_size = o_size
    s4 = image_size // 4
    lrelu = lambda x: tf.nn.leaky_relu(x, 0.2)

    ni = Input(shape)
    nn = Dense(n_units=(gf_dim * 4 * s4 * s4))(ni)
    nn = Reshape(shape=(-1, s4, s4, gf_dim * 4))(nn)
    nn = BatchNorm2d(decay=0.9, act=lrelu)(nn)
    nn = DeConv2d(gf_dim * 2, (5, 5), (1, 1))(nn)
    nn = BatchNorm2d(decay=0.9, act=lrelu)(nn)
    nn = DeConv2d(gf_dim, (5, 5), (2, 2))(nn)
    nn = BatchNorm2d(decay=0.9, act=lrelu)(nn)
    nn = DeConv2d(o_channel, (5, 5), (2, 2), act=tf.nn.tanh)(nn)

    return tf.nn.tanh(nn)
```

DCGAN Architecture



Architecture of discriminator

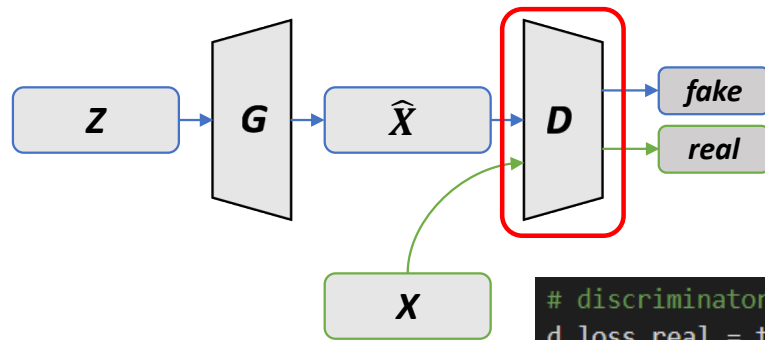
```
def get_discriminator(shape, df_dim=64):  
    lrelu = lambda x : tf.nn.leaky_relu(x, 0.2)  
  
    ni = Input(shape)  
    nn = Conv2d(df_dim, (5, 5), (2, 2), act=lrelu)(ni)  
    nn = Conv2d(df_dim * 2, (5, 5), (2, 2))(nn)  
    nn = BatchNorm2d(decay=0.9, act=lrelu)(nn)  
    nn = Conv2d(df_dim*4, (5, 5), (2, 2))(nn)  
    nn = BatchNorm2d(decay=0.9, act=lrelu)(nn)  
    nn = Flatten()(nn)  
    nn = Dense(n_units=1)(nn)  
  
    return tf.keras.Model(inputs=ni, outputs=nn, name='discriminator')
```

- Hello World: MNIST Classification
- Introduction of VAE
- VAE Architecture
- VAE Training
- VAE Interpolation
- Sampling

- Introduction of DCGAN
- DCGAN Architecture
- **DCGAN Training**
- DCGAN Interpolation

DCGAN Training

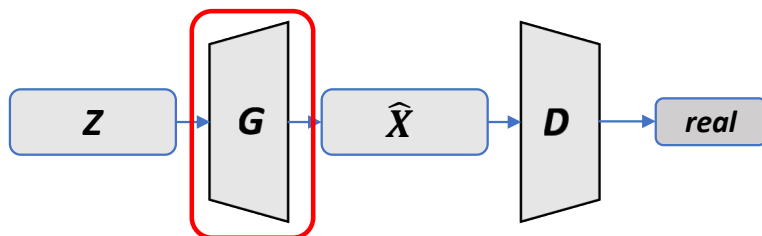
Loss of discriminator



```
# discriminator: real images are labelled as 1
d_loss_real = t1.cost.sigmoid_cross_entropy(d2_logits, tf.ones_like(d2_logits), name='dreal')
# discriminator: images from generator (fake) are labelled as 0
d_loss_fake = t1.cost.sigmoid_cross_entropy(d_logits, tf.zeros_like(d_logits), name='dfake')
# combined loss for updating discriminator
d_loss = d_loss_real + d_loss_fake
```

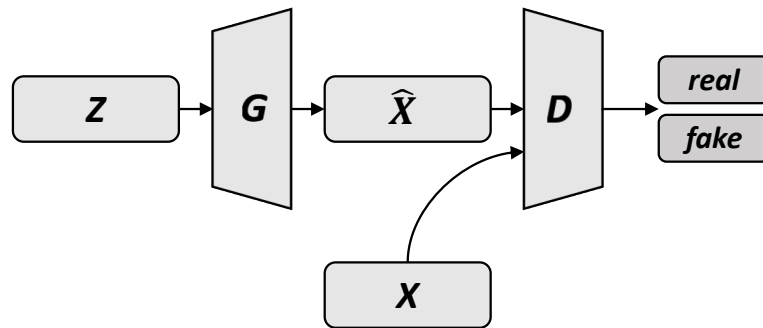
DCGAN Training

Loss of generator



```
# generator: try to fool discriminator to output 1  
g_loss = t1.cost.sigmoid_cross_entropy(d_logits, tf.ones_like(d_logits), name='gfake')
```

DCGAN Training



Training pipeline

```

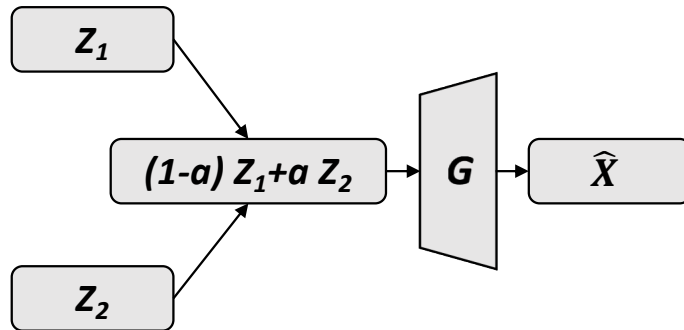
for epoch in range(flags.n_epoch):
    for step, batch_images in enumerate(images):
        if batch_images.shape[0] != flags.batch_size: # if the remaining data in this epoch < batch_size
            break
        step_time = time.time()
        with tf.GradientTape(persistent=True) as tape:
            z = np.random.normal(loc=0.0, scale=1.0, size=[flags.batch_size, flags.z_dim]).astype(np.float32)
            d_logits = D(G(z))
            d2_logits = D(batch_images)
            # discriminator: real images are labelled as 1
            d_loss_real = tl.cost.sigmoid_cross_entropy(d2_logits, tf.ones_like(d2_logits), name='dreal')
            # discriminator: images from generator (fake) are labelled as 0
            d_loss_fake = tl.cost.sigmoid_cross_entropy(d_logits, tf.zeros_like(d_logits), name='dfake')
            # combined loss for updating discriminator
            d_loss = d_loss_real + d_loss_fake
            # generator: try to fool discriminator to output 1
            g_loss = tl.cost.sigmoid_cross_entropy(d_logits, tf.ones_like(d_logits), name='gfake')

        grad = tape.gradient(g_loss, G.trainable_weights)
        g_optimizer.apply_gradients(zip(grad, G.trainable_weights))
        grad = tape.gradient(d_loss, D.trainable_weights)
        d_optimizer.apply_gradients(zip(grad, D.trainable_weights))
        del tape
  
```

- Hello World: MNIST Classification
- Introduction of VAE
- VAE Architecture
- VAE Training
- VAE Interpolation
- Sampling

- Introduction of DCGAN
- DCGAN Architecture
- DCGAN Training
- **DCGAN Interpolation**

DCGAN Interpolation



```

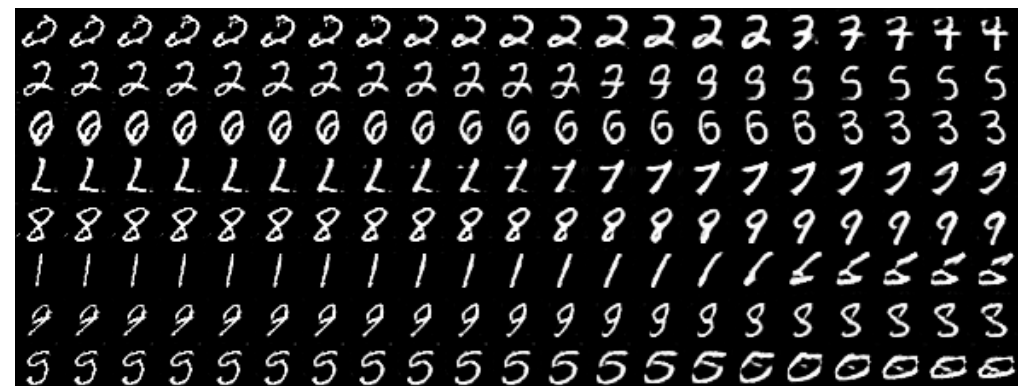
z1 = np.random.normal(loc=0.0, scale=1.0, size=[8, flags.z_dim]).astype(np.float32)
z2 = np.random.normal(loc=0.0, scale=1.0, size=[8, flags.z_dim]).astype(np.float32)

G = get_generator([None, flags.z_dim], gf_dim=64, o_size=flags.output_size, o_channel=flags.c_dim)
G.load_weights('{}G.npz'.format(flags.checkpoint_dir))
G.eval()

z = []
for i in range(8):
    for j in range(21):
        a = (j - 10) / 10.
        z.append(a*z1[i] + (1-a)*z2[i])
z = np.array(z)

gen_img = G(z)

tl.visualize.save_images(gen_img.numpy(), [8, 21],
                        '{}/interplation.png'.format(flags.sample_dir))
  
```



More

Improved GAN

LSGAN

WGAN

WGAN-GP

BiGAN

VAE-GAN

...

with MNIST

Pix2Pix
CycleGAN
SRGAN

...

with other datasets

Proposal Your Projects

Thanks