13 - Generating images

Ludwig Krippahl

Generating images

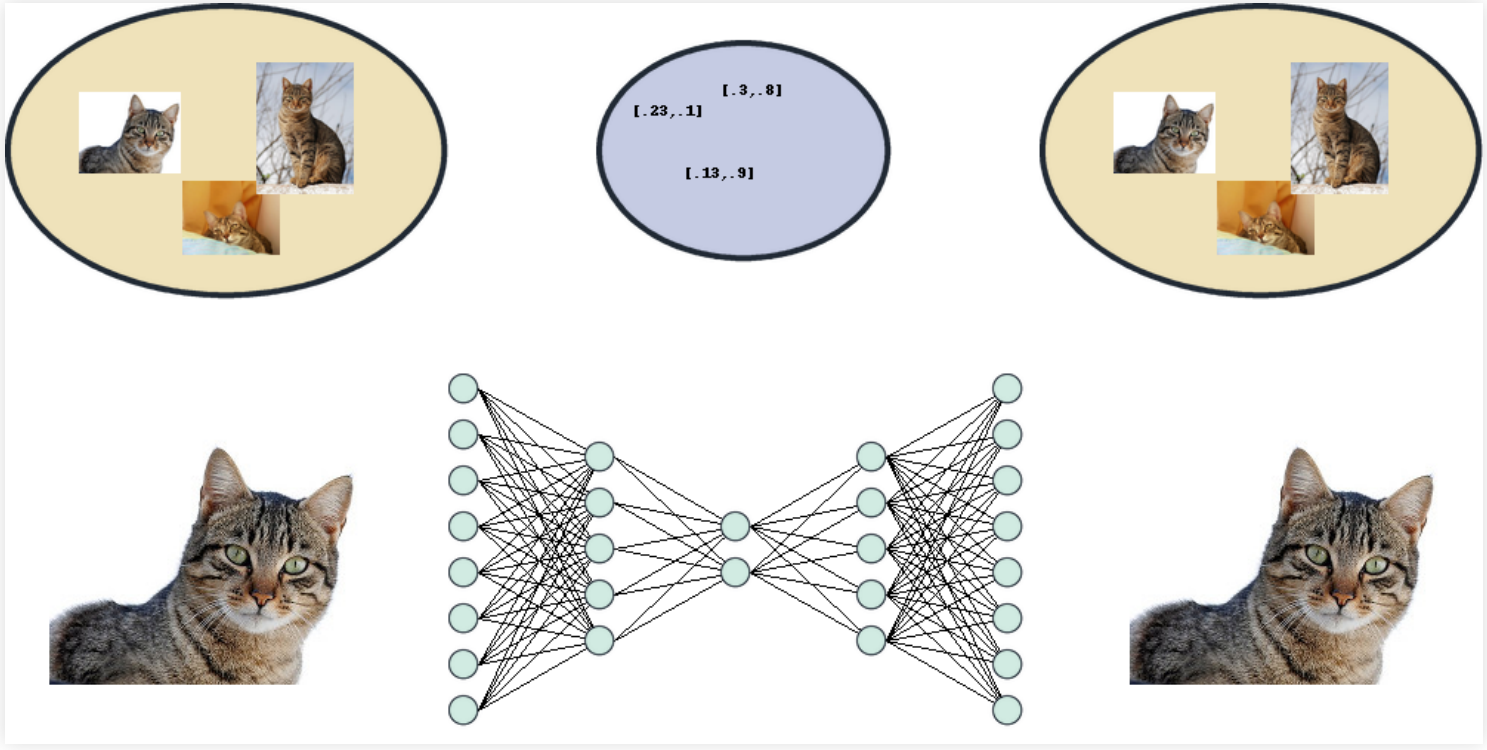
Summary

- Variational Autoencoder
- Generative Adversarial Networks

Variational Autoencoder

Autoencoder

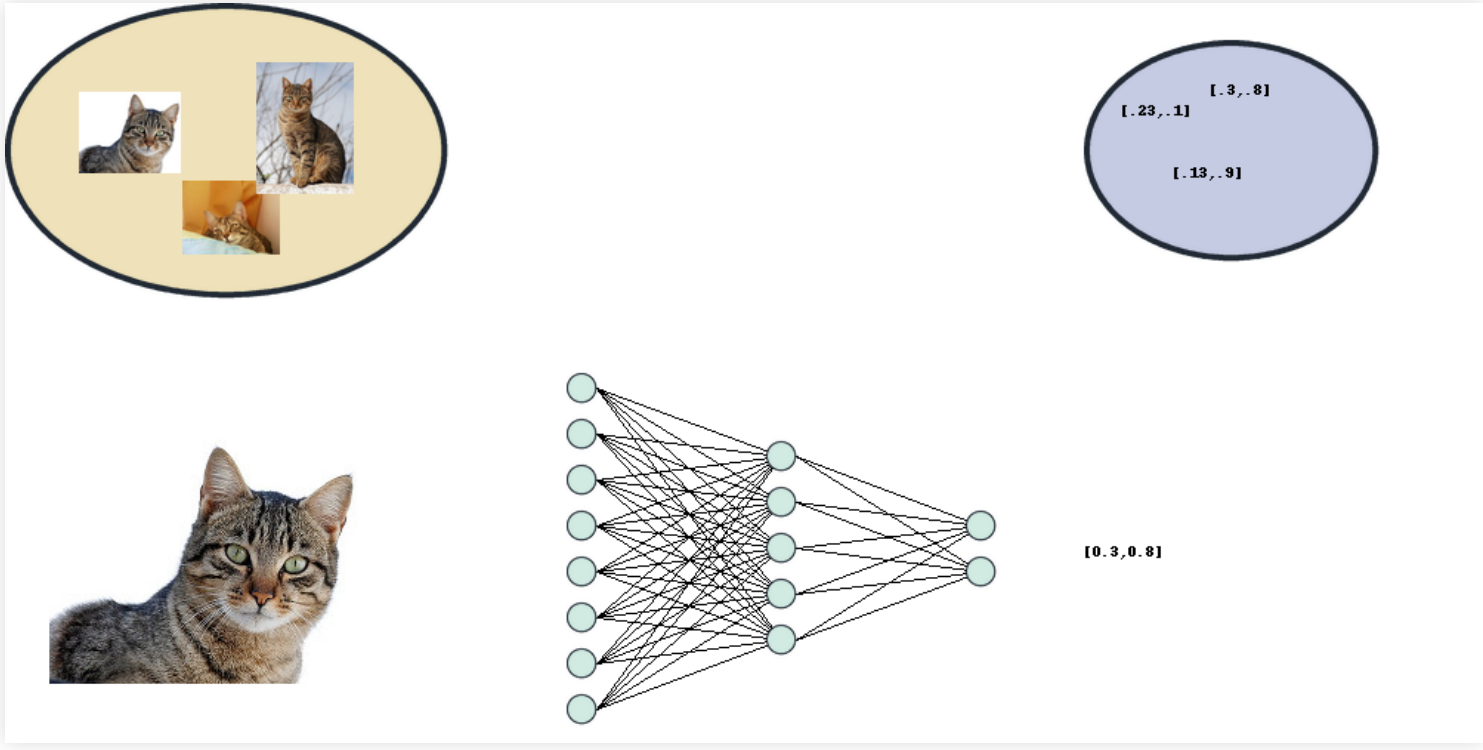
- Can we use autoencoders to generate new examples?



Cat images: Joaquim Alves Gaspar CC-SA

Autoencoder

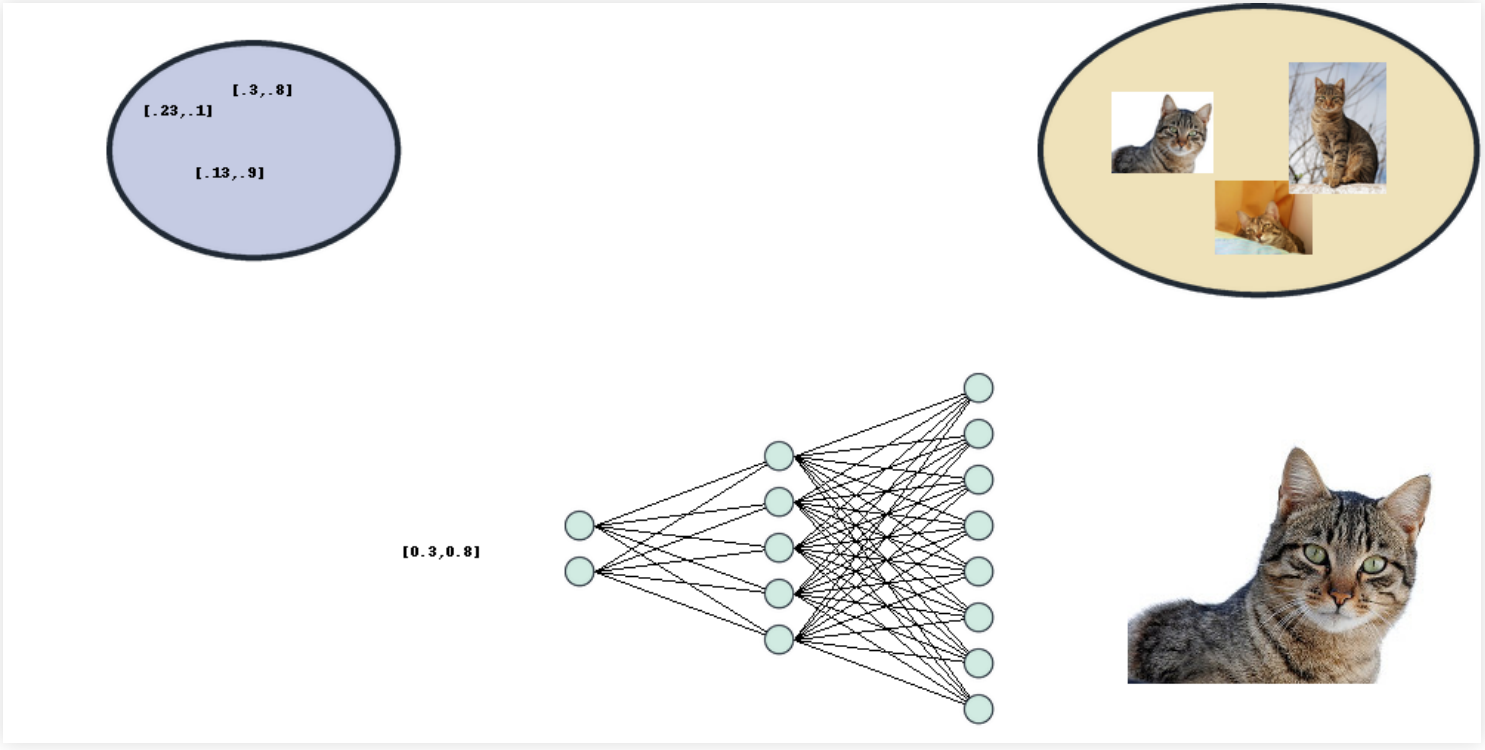
- Autoencoders create a latent representation from the data



Cat images: Joaquim Alves Gaspar CC-SA

Autoencoder

- And then decode to recreate the data from this representation



Cat images: Joaquim Alves Gaspar CC-SA

Autoencoder

- Can we use the decoder to generate new examples?

Discriminative vs Generative

- A discriminative model tries to approximate a function $p(y \mid x)$

- E.g. Logistic regression or softmax ANN predict the probability of each class given the features

- A generative model approximates $p(x, y)$ and then finds $p(y \mid x)$:

$$p(x, y) = p(y \mid x)p(x)$$

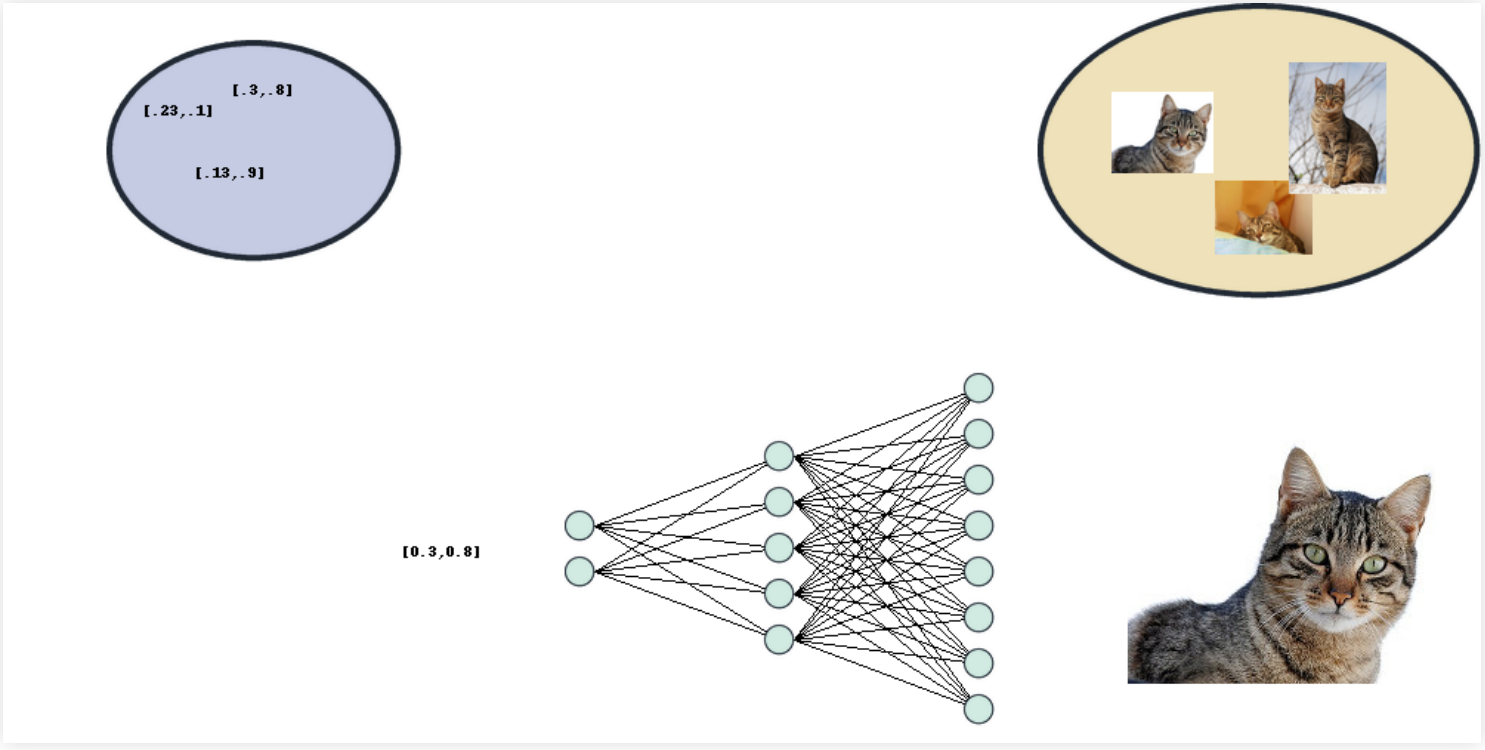
- This is generative because, knowing $p(x, y)$, we can sample from the distribution

With autoencoders

- We decode from z, so we need to find its distribution in order to generate examples from $p(z, y) = p(y \mid h)p(z)$

Generating Data

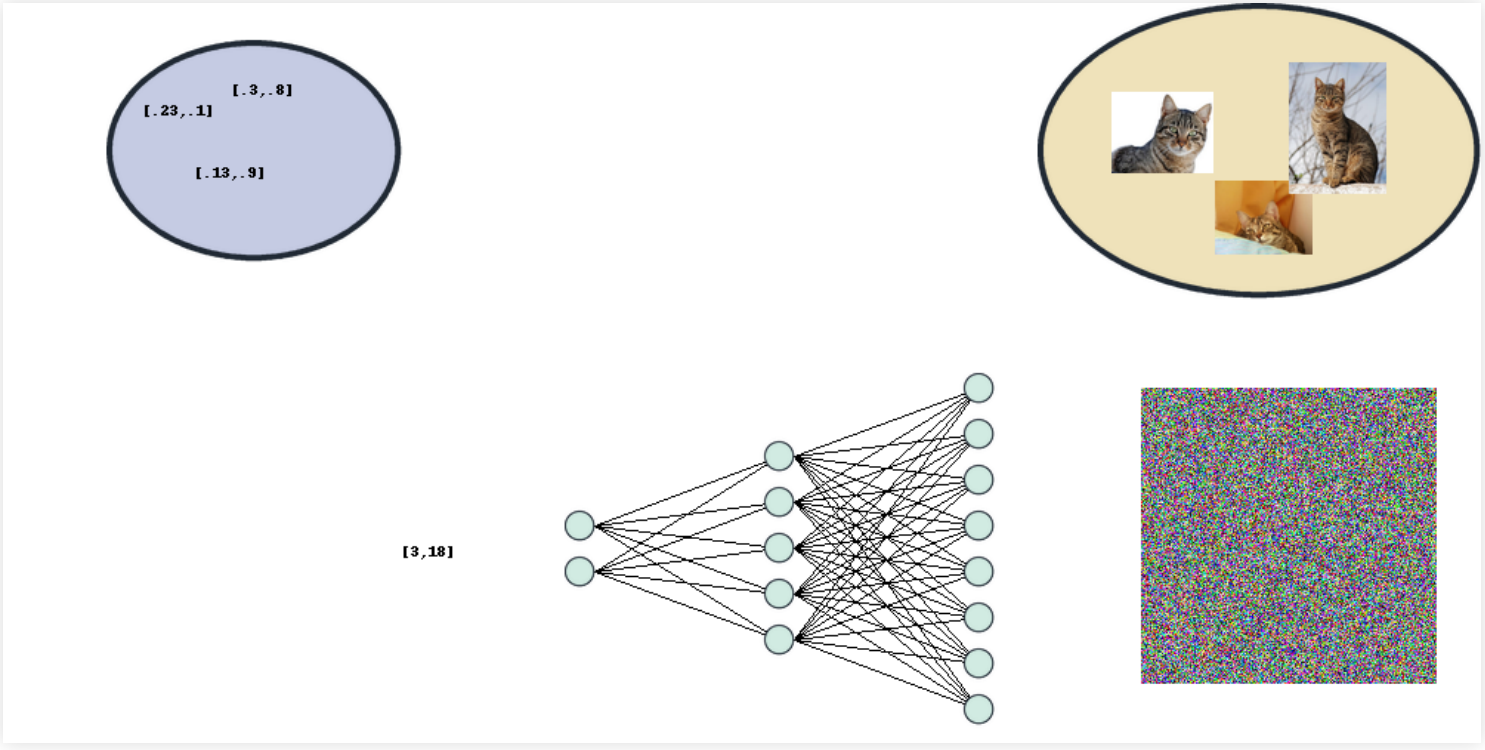
- Intuition: we need to sample the right part of the latent space



Cat images: Joaquim Alves Gaspar CC-SA

Generating Data

- Intuition: if outside the right region, the result is garbage



Cat images: Joaquim Alves Gaspar CC-SA

Generating Data

- The decoder is modelling a conditional probability $p_{decoder}(x \mid z)$
  - where z is given by the encoder part of the autoencoder
- We can think of encoder and decoder as modelling conditional probabilities

$$p_{encoder}(z \mid x) \qquad p_{decoder}(x \mid z)$$

- We can use the decoder to get $p(z, y) = p(y \mid h)p(z)$
- But we need $p(z)$. How do we get $p(z)$?

Variational Autoencoder

- Can we use autoencoders to generate new examples?

- Yes, if we know the distribution of the latent space

Variational Autoencoders

- Train the autoencoder to encode into a given distribution

- E.g. mixture of independent Gaussians (multivariate Gaussian)

$$p(z \mid x) = N(z|\mu(x), \Sigma(x)) \qquad \Sigma(x) = \mathrm{diag}(\sigma_1^2, \sigma_2^2, \dots)$$

- This way we learn the distibution for generating examples of each type

Variational Autoencoders

- The loss function:

$$l(\theta;\phi;x,z) = -E_{q_\phi(z|x)}\left[\log p_\theta(x \mid z)\right] + D_{KL}\left(q_\phi(z \mid x) \,||\, p(z)\right)$$

$$D_{KL} = -\frac{1}{2}\left(\sum_i (log\ \sigma_i^2 + 1) + \sum_i log\ \sigma_i^2 + \sum_i \mu_i^2\right)$$

- for $p(z \mid x) = N(z|\mu(x), \Sigma(x))$ $\quad\quad$ $\Sigma(x) = \mathrm{diag}(\sigma_1^2, \sigma_2^2, \dots)$

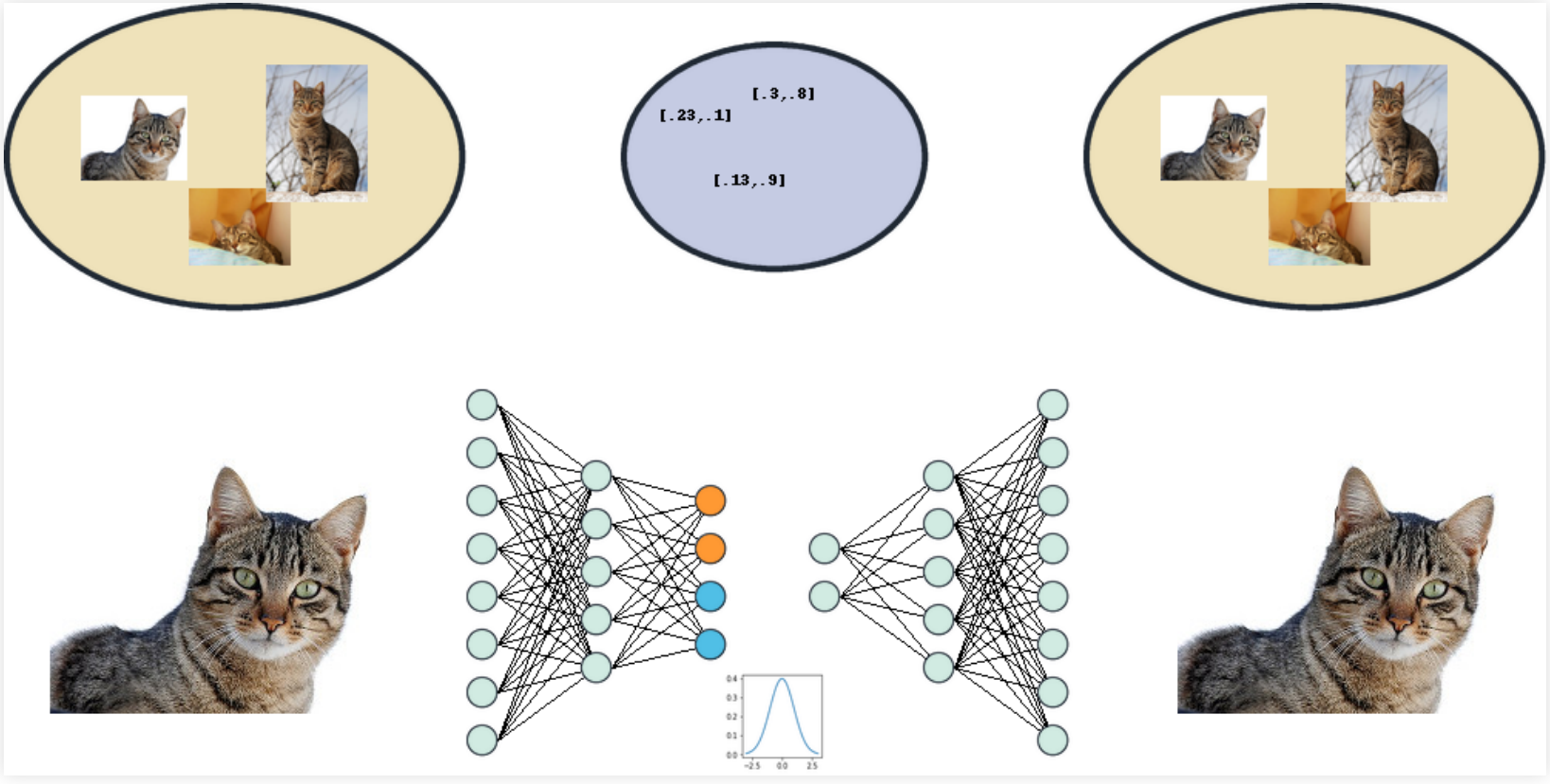
Variational Autoencoder:

- The encoder outputs parameters for probability distributions
- This way we know what the distributions are
- The decoder minimizes reconstruction error from samples from these distributions
- A regularization term minimizes divergence between latent $q_\phi(z \mid x)$ and a target $p(z)$

Variational Autoencoders

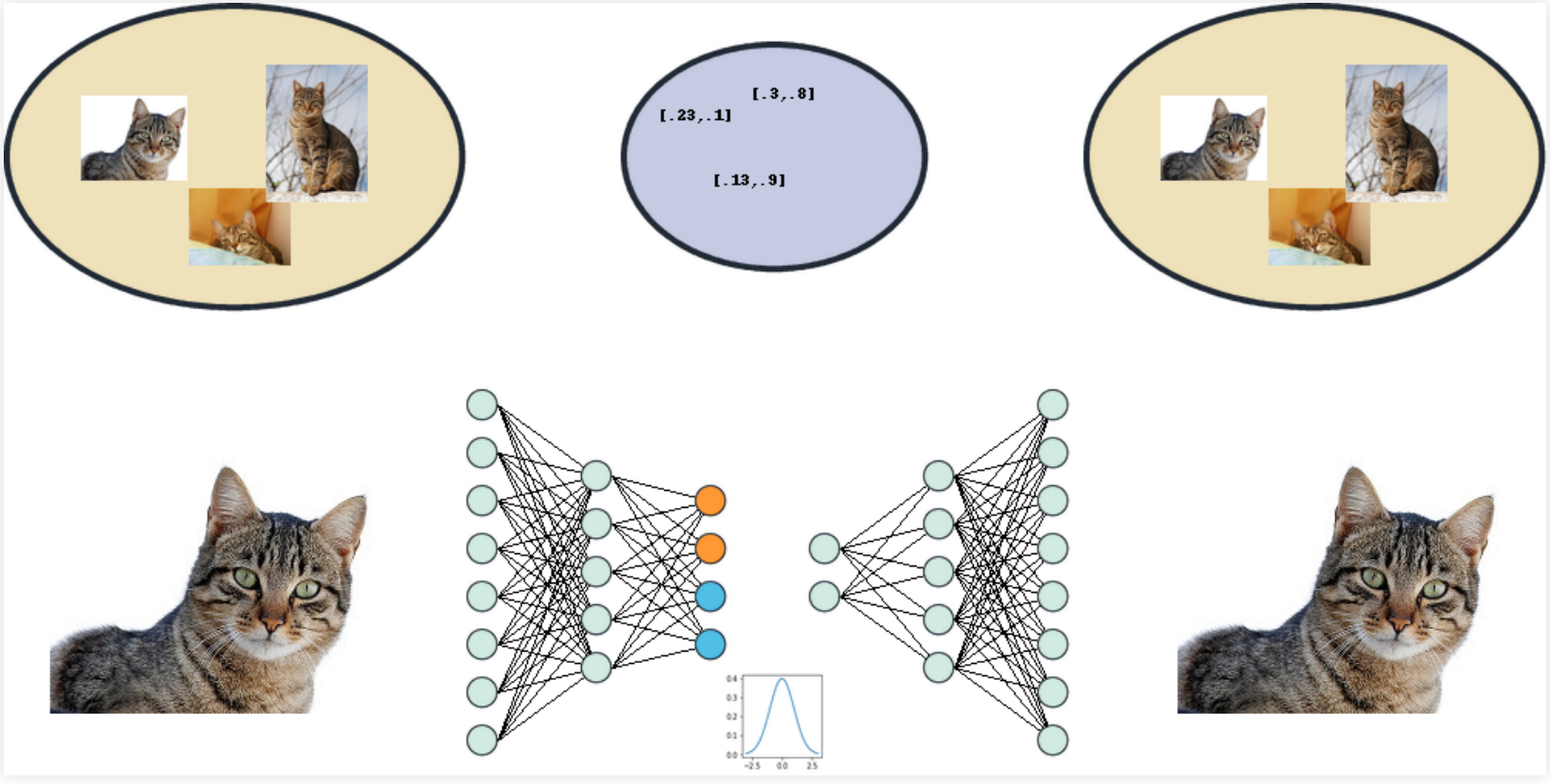
- Train the encoder to learn parameters for a given distribution

- E.g. Mean and standard deviation of independent Gaussians



Variational Autoencoders

- The decoder inputs random values drawn from those distributions
  - E.g. Mean and standard deviation of independent Gaussians



Variational Autoencoders

- How do we backpropagate through random sampling?
- Reparametrize: z is deterministic apart from a normally distributed error

$$h = \mu + \sigma \odot \epsilon \qquad \epsilon \sim \mathcal{N}(0,1)$$

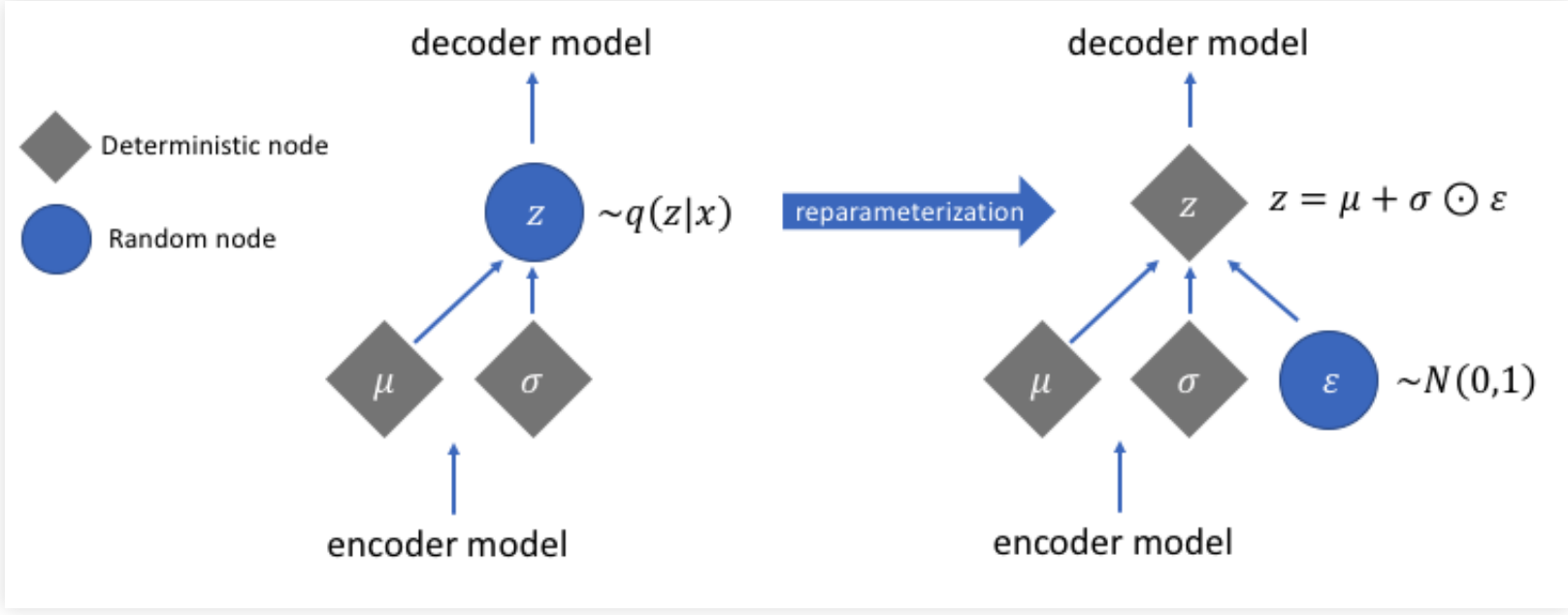


Image: Jeremy Jordan, Variational autoencoders.

Variational Autoencoders

- VAE can learn to disentangle meaningful attributes

- (This is a more sophisticated example)

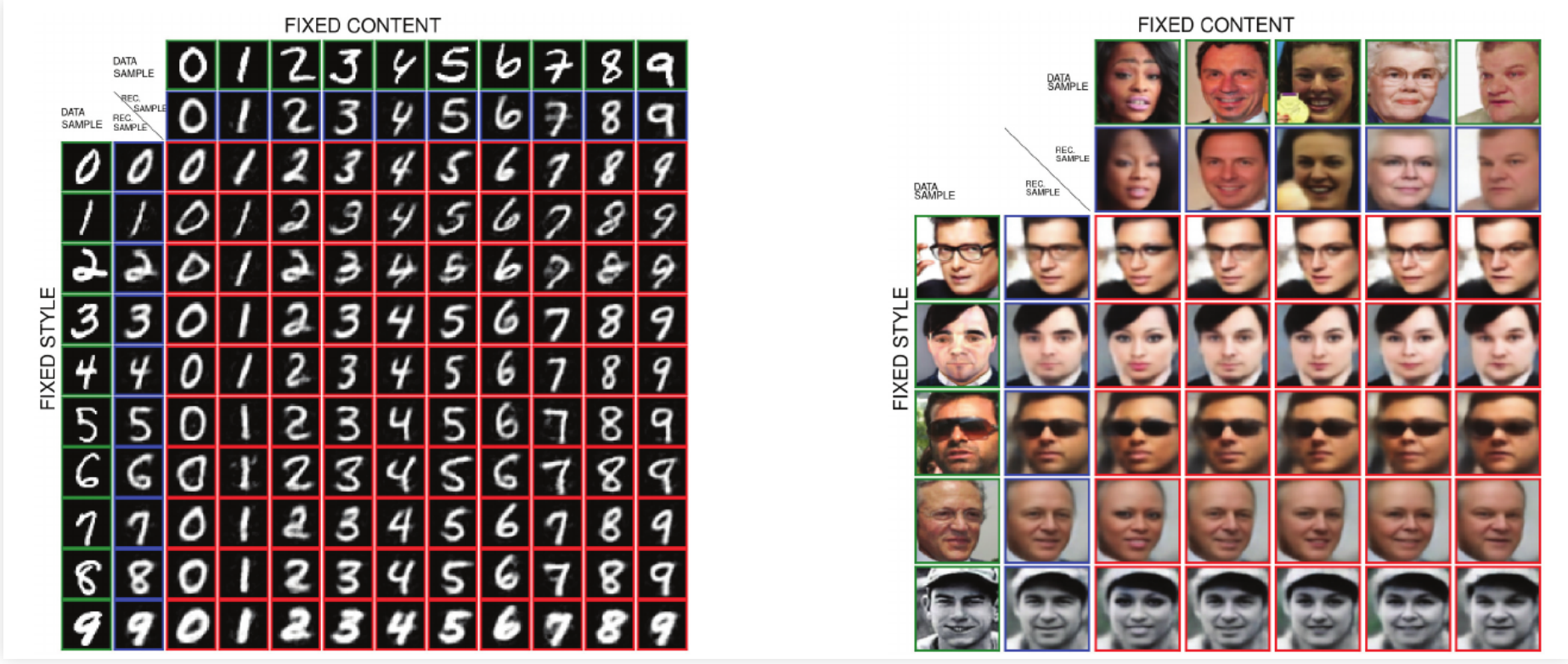


Image: Bouchacourt et. al., Multi-level variational autoencoder, 2018

Implementing a simple VAE

VAE implementation

VAE using the MNIST data set

(Adapted from: https://keras.io/examples/generative/vae/)

- We will create 3 models:
  - Encoder
  - Sampler
  - Decoder
- We will use a mix of convolution and dense layers
- Use 2 values for the latent space of the decoder
  - The encoder will output 4 values, variance and mean for each distribution
- We will use this regularization:

$$D_{KL} = -\frac{1}{2}\left(\sum_i (log\ \sigma_i^2 + 1) + \sum_i log\ \sigma_i^2 + \sum_i \mu_i^2\right)$$

VAE implementation

- Imports and constants

```
from tensorflow.keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import Adam,SGD
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Reshape,Input,BatchNormalization,Conv2D, \
                                    MaxPooling2D,Activation, Flatten, Dropout,
                                    Dense, UpSampling2D
from tensorflow.keras.losses import binary_crossentropy
from tensorflow.keras import backend as K

ORIGINAL_DIM = 28*28
LATENT_DIM = 2
```

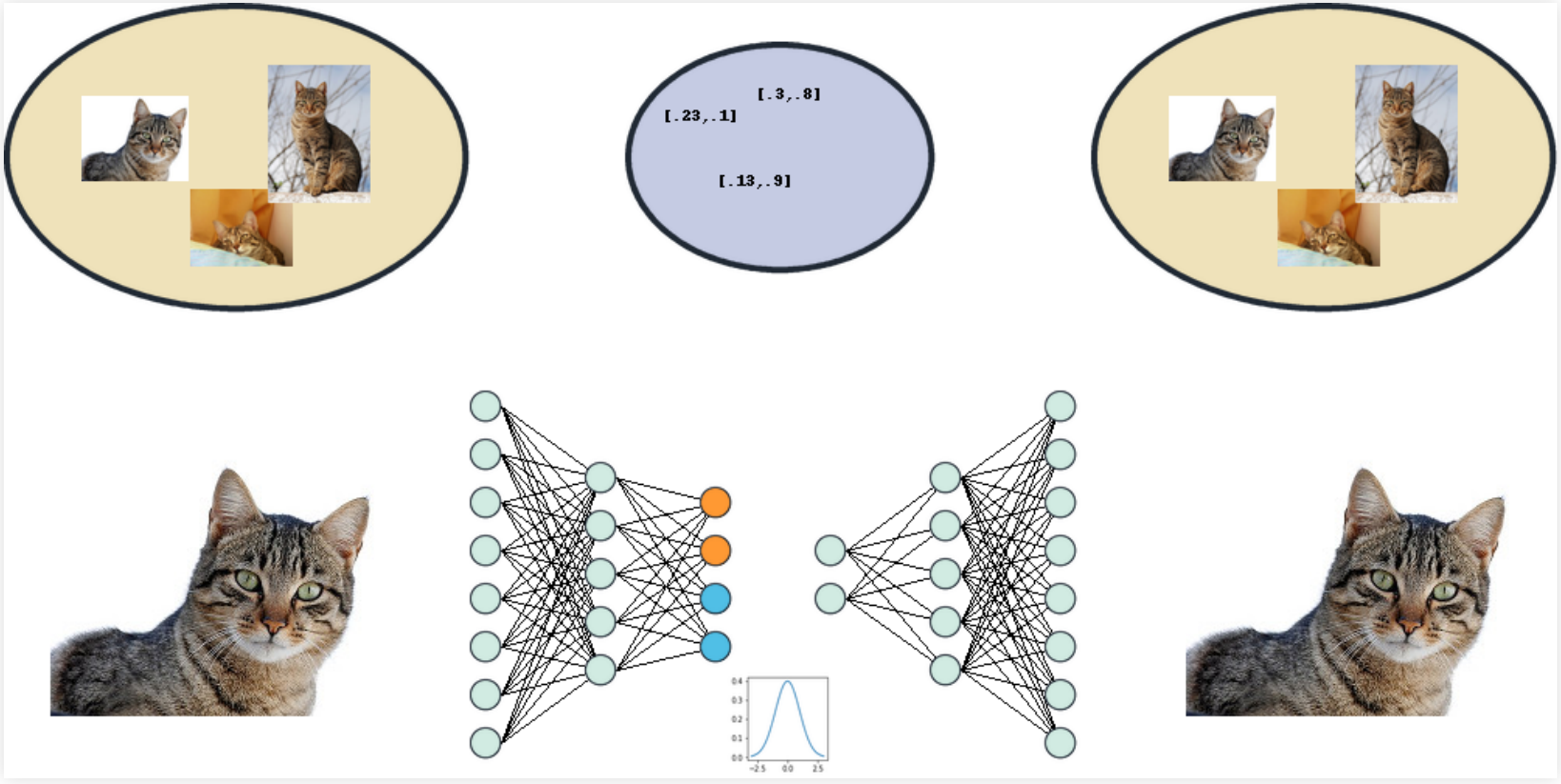
VAE implementation

```
def encoder_graph(filters, dense_neurons):
    inputs = Input(shape=(28,28,1),name='inputs')
    x = inputs
    last_conv_w = 28
    for filt in filters:
        x = Conv2D(filt, (3, 3), padding="same", )(x)
        x = Activation("relu")(x)
        x = BatchNormalization(axis=-1)(x)
        x = Conv2D(filt, (3, 3), padding="same", )(x)
        x = Activation("relu")(x)
        x = BatchNormalization(axis=-1)(x)
        x = MaxPooling2D(pool_size=(2, 2))(x)
        last_conv_w = last_conv_w // 2

    x = Flatten()(x)
    x = Dense(dense_neurons)(x)
    x = Activation("relu")(x)
    x = BatchNormalization()(x)
    z_mean = Dense(LATENT_DIM,name='means')(x)
    z_log_var = Dense(LATENT_DIM,name='log_sigma')(x)
    return inputs,z_mean,z_log_var,last_conv_w
```

VAE implementation

```
...
   z_mean = Dense(LATENT_DIM,name='means')(x)
   z_log_var = Dense(LATENT_DIM,name='log_sigma')(x)
   return inputs,z_mean,z_log_var,last_conv_w
```



VAE implementation

```
...
   z_mean = Dense(LATENT_DIM,name='means')(x)
   z_log_var = Dense(LATENT_DIM,name='log_sigma')(x)
   return inputs,z_mean,z_log_var,last_conv_w
```

- Why `log_var`?

- Variance must be > 0, but a linear output can be any number.
- So we output `log(var)` and then use `exp(log_var)`

VAE implementation

- Why `log_var`?

- Variance must be > 0, but a linear output can be any number.
- So we output `log(var)` and then use `exp(log_var)`

```
def sampler_graph(epsilon_sigma):
    means = Input(shape=(LATENT_DIM,))
    log_vars = Input(shape=(LATENT_DIM,))
    epsilon = K.random_normal(shape=(K.shape(means)[0], LATENT_DIM),
                              mean=0., stddev=epsilon_sigma)
    z_sample = means + K.exp(0.5*log_vars) * epsilon
    return means, log_vars, z_sample
```

- `0.5*log_vars` because $\sigma = \sqrt{var}$
- `epsilon_sigma` is a "cheat" to make it easier to train

- In this case if we have too much variance in sampling the decoder has a hard time learning

VAE implementation

```
def decoder_graph(filters, dense_neurons, last_conv_w):
    decoder_inputs = Input(shape=(LATENT_DIM,))
    x = Dense(dense_neurons)(decoder_inputs)
    x = Activation("relu")(x)
    x = BatchNormalization(axis=-1)(x)
    x = Dense(last_conv_w*last_conv_w*filters[-1])(x)
    x = Reshape((last_conv_w,last_conv_w,filters[-1]))(x)
    x = Activation("relu")(x)
    for filt in filters[::-1]:
        x = UpSampling2D()(x)
        x = Conv2D(filt, (3, 3), padding="same")(x)
        x = Activation("relu")(x)
        x = BatchNormalization(axis=-1)(x)
        x = Conv2D(filt, (3, 3), padding="same")(x)
        x = Activation("relu")(x)
        x = BatchNormalization(axis=-1)(x)
    x = Conv2D(1, (3, 3), padding="same")(x)
    decoder_outputs = Activation("sigmoid")(x)
    return decoder_inputs,decoder_outputs
```

VAE implementation

- The loss function:

```
def vae_loss(inputs, outputs, z_mean, z_log_var):
    reconstruction = binary_crossentropy(inputs, outputs)
    kl_loss = 1 + z_log_var - K.square(z_mean) - K.exp(z_log_var)
    kl_loss = -0.5 * K.sum(kl_loss, axis=-1)
    return K.mean(reconstruction,axis=[-1,-2])+ 0.1*kl_loss
```

$$D_{KL} = -\frac{1}{2}\left(\sum_i (log\ \sigma_i^2 + 1) + \sum_i log\ \sigma_i^2 + \sum_i \mu_i^2\right)$$

- Calibrate the weight of the regularization

- Too little and the encoding manifold may not be well-behaved
- Too much and we lose in the reconstruction and have a lower quality output

VAE implementation

```
def create_models(filters,dense_neurons, epsilon_sigma):
    inputs,z_mean,z_log_var,conv_width = encoder_graph(filters,dense_neurons)
    input_means, input_vars, z_sample = sampler_graph(epsilon_sigma)
    decoder_inputs,decoder_outputs = decoder_graph(filters,dense_neurons,conv_w

    encoder = Model(inputs= inputs,outputs = [z_mean,z_log_var])
    sampler = Model(inputs = [input_means, input_vars], outputs = z_sample)
    decoder = Model(inputs = decoder_inputs, outputs = decoder_outputs)

    vae_outputs = decoder(sampler(encoder(inputs)))
    vae = Model(inputs = inputs, outputs = vae_outputs)

    loss = vae_loss(inputs,vae_outputs,z_mean, z_log_var)
    return vae,encoder,decoder,loss
```

VAE implementation

```
filters = [8,16]
dense_neurons = 16
epsilon_sigma = 1

vae,encoder,decoder,loss = create_models(filters,dense_neurons,epsilon_sigma)
encoder.compile(optimizer='adam', loss = 'mse')
decoder.compile(optimizer='adam', loss = 'mse')
vae.add_loss(loss)
vae.compile(optimizer='adam')

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.astype('float32').reshape((-1,28,28,1)) / 255
x_test = x_test.astype('float32').reshape((-1,28,28,1)) / 255
vae.fit(x_train, x_train,
        epochs=200,
        batch_size=128,
        validation_data=(x_test, x_test))
```

VAE implementation

- Visualizing the mapping, using the means, for the test set

```
x_test_mean,x_test_std = encoder.predict(x_test)
plt.figure(figsize=(6, 6))
plt.scatter(x_test_mean[:, 0], x_test_mean[:, 1], c=y_test)
plt.colorbar()
plt.show()
```

VAE implementation

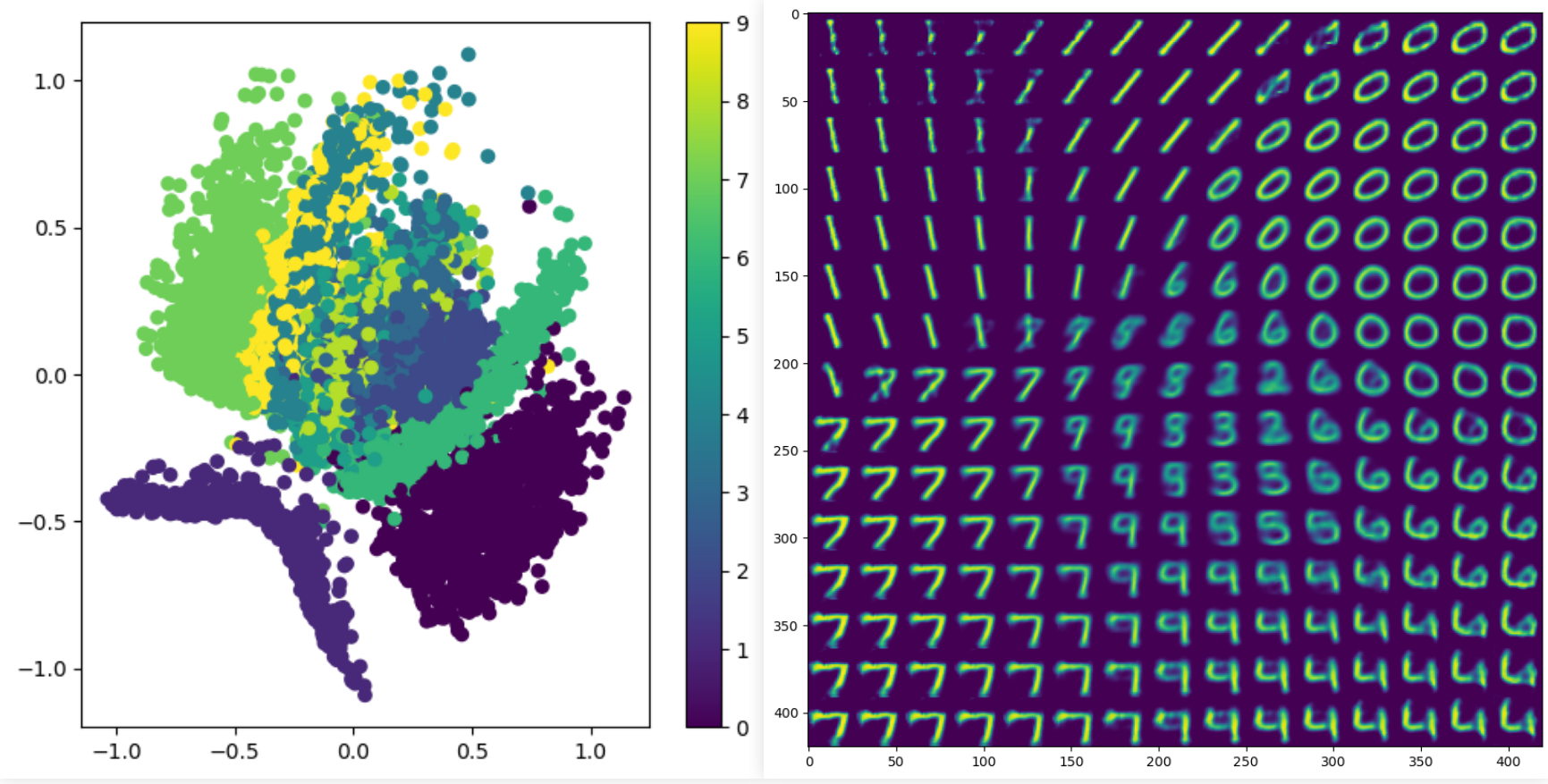
- Visualizing the manifold

```
n = 15  # figure with 15x15 digits
digit_size = 28
figure = np.zeros((digit_size * n, digit_size * n))
# We will sample n points within [-15, 15] standard deviations
grid_x = np.linspace(-15, 15, n)
grid_y = np.linspace(-15, 15, n)

for i, yi in enumerate(grid_x):
    for j, xi in enumerate(grid_y):
        z_sample = np.array([[xi, yi]])*epsilon_sigma
        x_decoded = decoder.predict(z_sample)
        digit = x_decoded[0].reshape(digit_size, digit_size)
        figure[i * digit_size: (i + 1) * digit_size,
               j * digit_size: (j + 1) * digit_size] = digit

plt.figure(figsize=(10, 10))
plt.imshow(figure)
plt.show()
```

VAE implementation

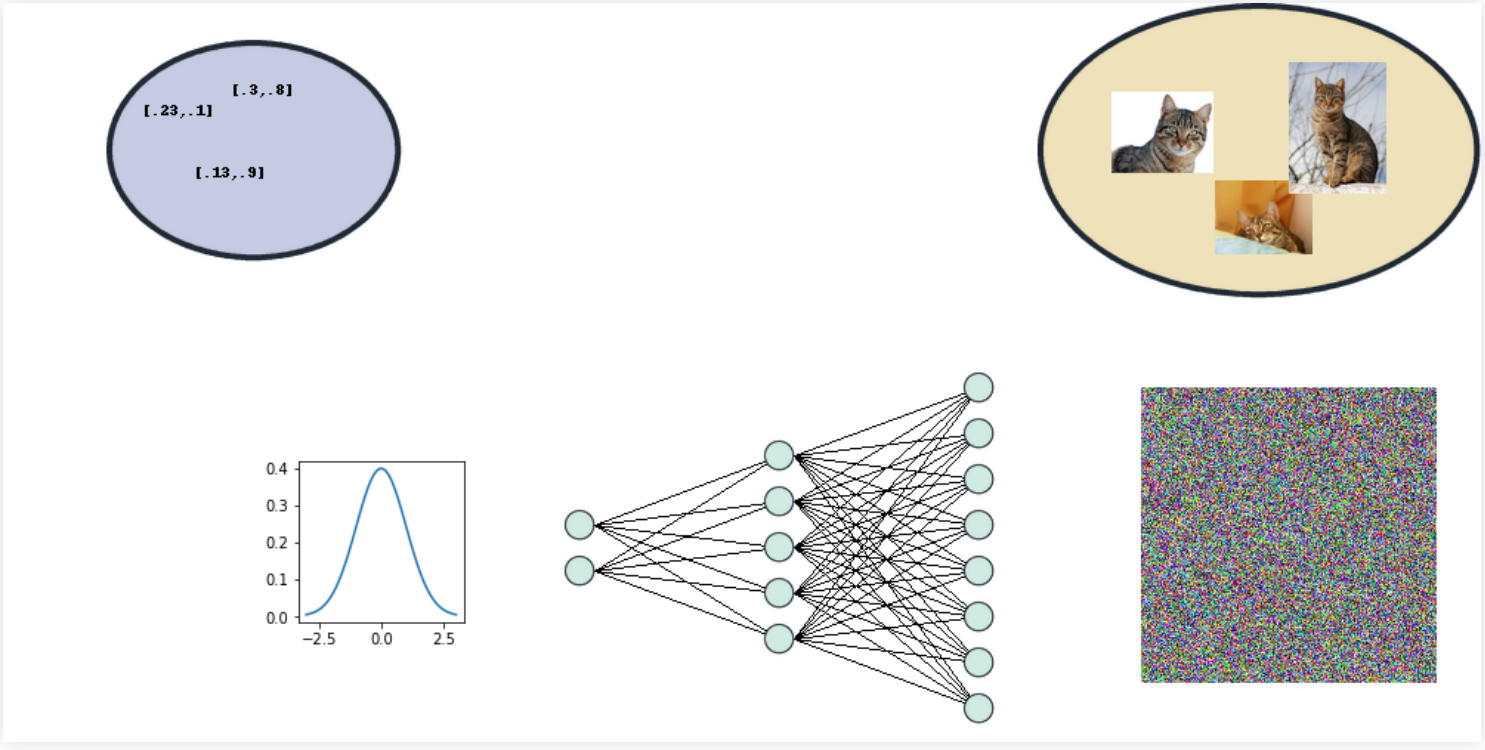


Generative adversarial networks

Generative adversarial networks

- Fix the latent space with some distribution (e.g. Gaussian mixture)

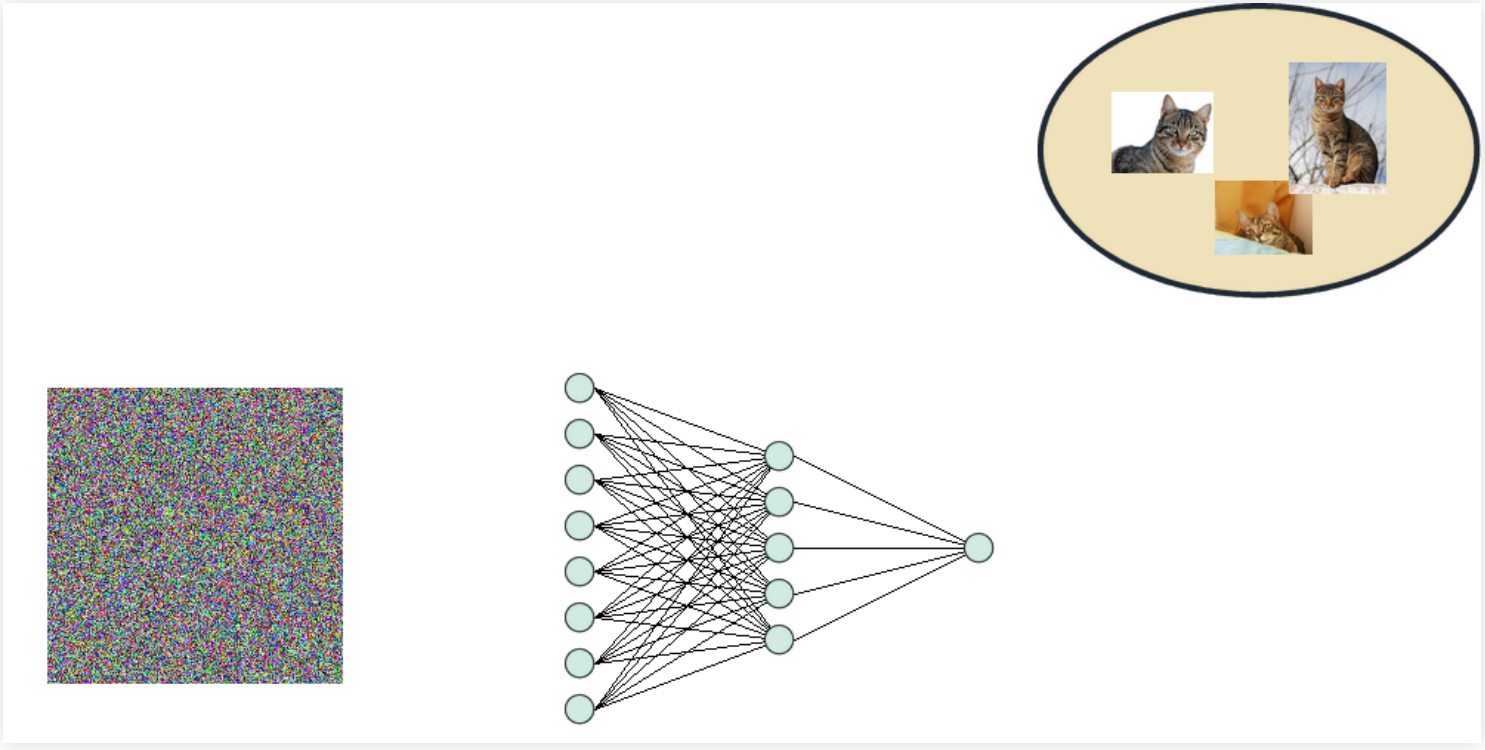
- The result will be garbage because net not trained



Cat images: Joaquim Alves Gaspar CC-SA

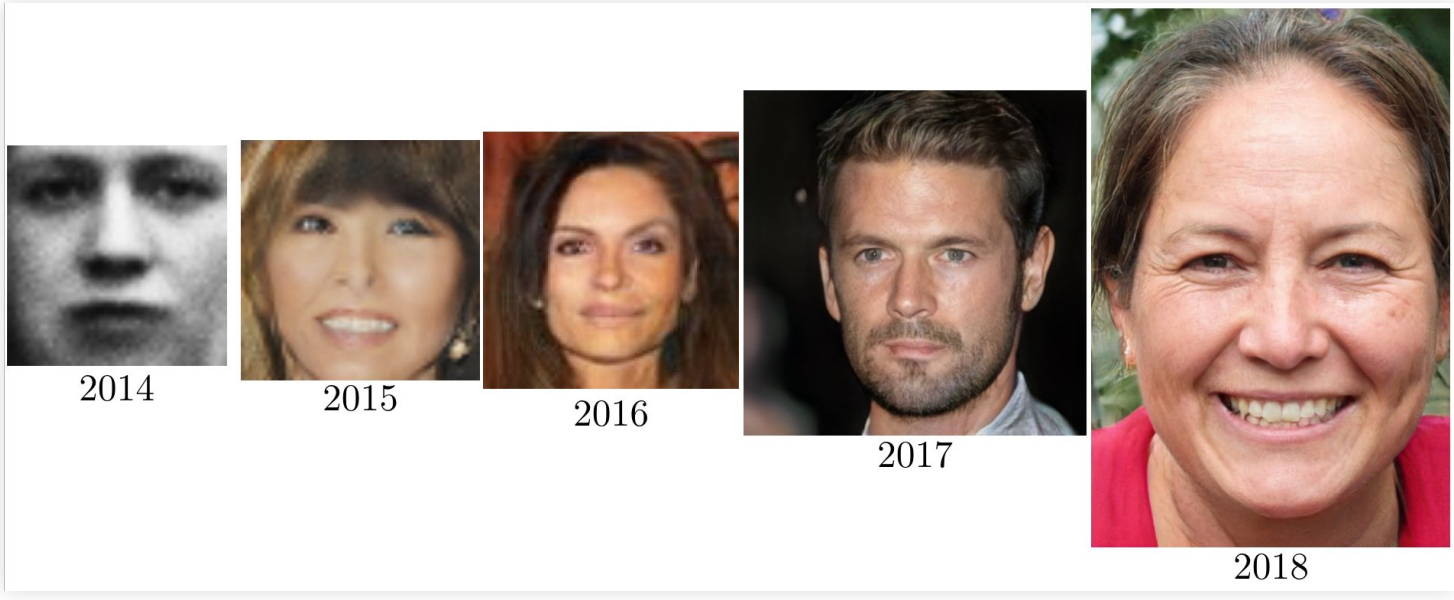
Generative adversarial networks

- Train a network to distinguish the real examples from fakes



Generative adversarial networks

- One network creates examples from given distribution
- The other distinguishes real from fake
- Train both, alternating, so each becomes increasingly better



Ian Goodfellow

Generative adversarial networks

- One network creates examples from given distribution
- The other distinguishes real from fake
- Train both, alternating, so each becomes increasingly better
- As a result, the generator learns to map our fixed initial distribution to the space of our target examples.

Implementing a simple GAN

Implementing GAN

Based on a tutorial by Jason Brownlee

Outline:

- Discriminator model, trained on mixture of true and fake images
- Generator model, inputs random (Gaussian) values, generates image
- Train the generator model combined with the (frozen) discriminator

Implementing GAN

Generator:

- Note: we do not need to compile the generator

```
def define_generator(latent_dim):
    model = Sequential()
    n_nodes = 128 * 7 * 7
    model.add(Dense(n_nodes, input_dim=latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Reshape((7, 7, 128)))
    model.add(UpSampling2D())
    model.add(Conv2D(128, (4,4), padding='same'))
    model.add(UpSampling2D())
    model.add(Conv2D(128, (4,4), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Conv2D(1, (7,7), activation='sigmoid', padding='same'))
    return model
```

Implementing GAN

- Discriminator:

```
def define_discriminator(in_shape=(28,28,1)):
    model = Sequential()
    model.add(Conv2D(64, (3,3), strides=(2, 2),
                     padding='same', input_shape=in_shape))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.4))
    model.add(Conv2D(64, (3,3), strides=(2, 2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.4))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt,
                  metrics=['accuracy'])
    return model
```

Implementing GAN

- Combining generator and discriminator:

```
def define_gan(g_model, d_model):
     d_model.trainable = False
     model = Sequential()
     model.add(g_model)
     model.add(d_model)
     opt = Adam(lr=0.0002, beta_1=0.5)
     model.compile(loss='binary_crossentropy', optimizer=opt)
     return model
```

- The `trainable` flag in the model is applied to all layers when compiling.
- The discriminator in this model is frozen, but the parameters are the same.

Implementing GAN

- Bring it all together:

```
latent_dim = 100
d_model = define_discriminator()
g_model = define_generator(latent_dim)
gan_model = define_gan(g_model, d_model)
(trainX, _), (_, _) = load_data()
dataset = trainX.reshape((-1,28,28,1)).astype('float32')/255
train(g_model, d_model, gan_model, dataset, latent_dim)
```

Implementing GAN

- The training function:

```
def train(g_model, d_model, gan_model, dataset,
          latent_dim, n_epochs=100, n_batch=256):
    bat_per_epo = int(dataset.shape[0] / n_batch)
    half_batch = int(n_batch / 2)
    for i in range(n_epochs):
        for j in range(bat_per_epo):
            X_real, y_real = generate_real_samples(dataset, half_batch)
            X_fake, y_fake = generate_fake_samples(g_model,
                                                   latent_dim,
                                                   half_batch)
            X, y = np.vstack((X_real, X_fake)), np.vstack((y_real, y_fake))
            d_loss, _ = d_model.train_on_batch(X, y)
            X_gan = generate_latent_points(latent_dim, n_batch)
            y_gan = np.ones((n_batch, 1))
            g_loss = gan_model.train_on_batch(X_gan, y_gan)
```

- `train_on_batch` returns loss or list with loss and metrics
  - (e.g. the discriminator, was compiled with accuracy metric)

Implementing GAN

- Generating examples:

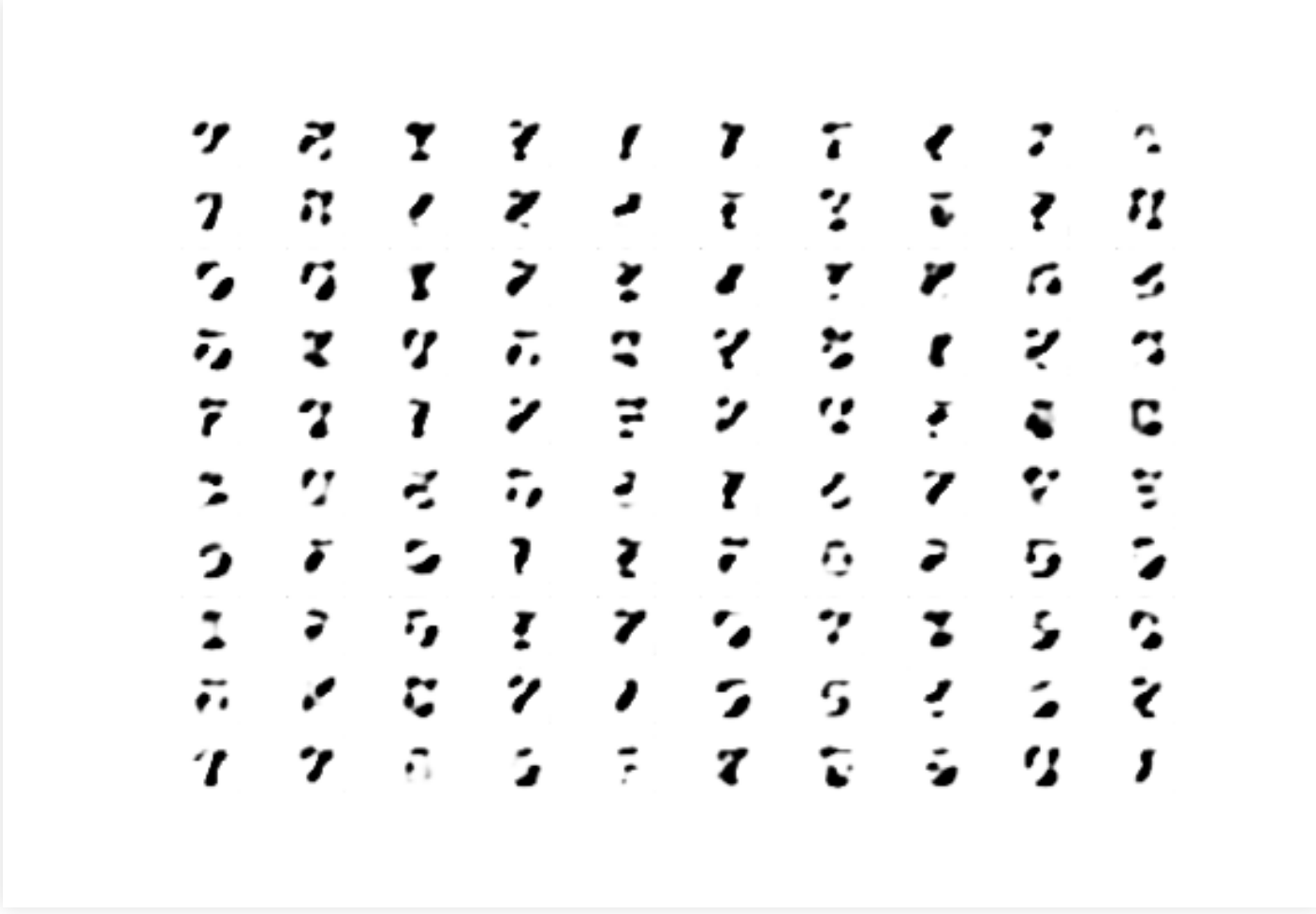
```
def generate_real_samples(dataset, n_samples):
    ix = np.random.randint(0, dataset.shape[0], n_samples)
    X = dataset[ix]
    y = np.ones((n_samples, 1))
    return X, y

def generate_latent_points(latent_dim, n_samples):
    x_input = np.random.randn(latent_dim * n_samples)
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input

def generate_fake_samples(g_model, latent_dim, n_samples):
    x_input = generate_latent_points(latent_dim, n_samples)
    X = g_model.predict(x_input)
    y = np.zeros((n_samples, 1))
    return X, y
```

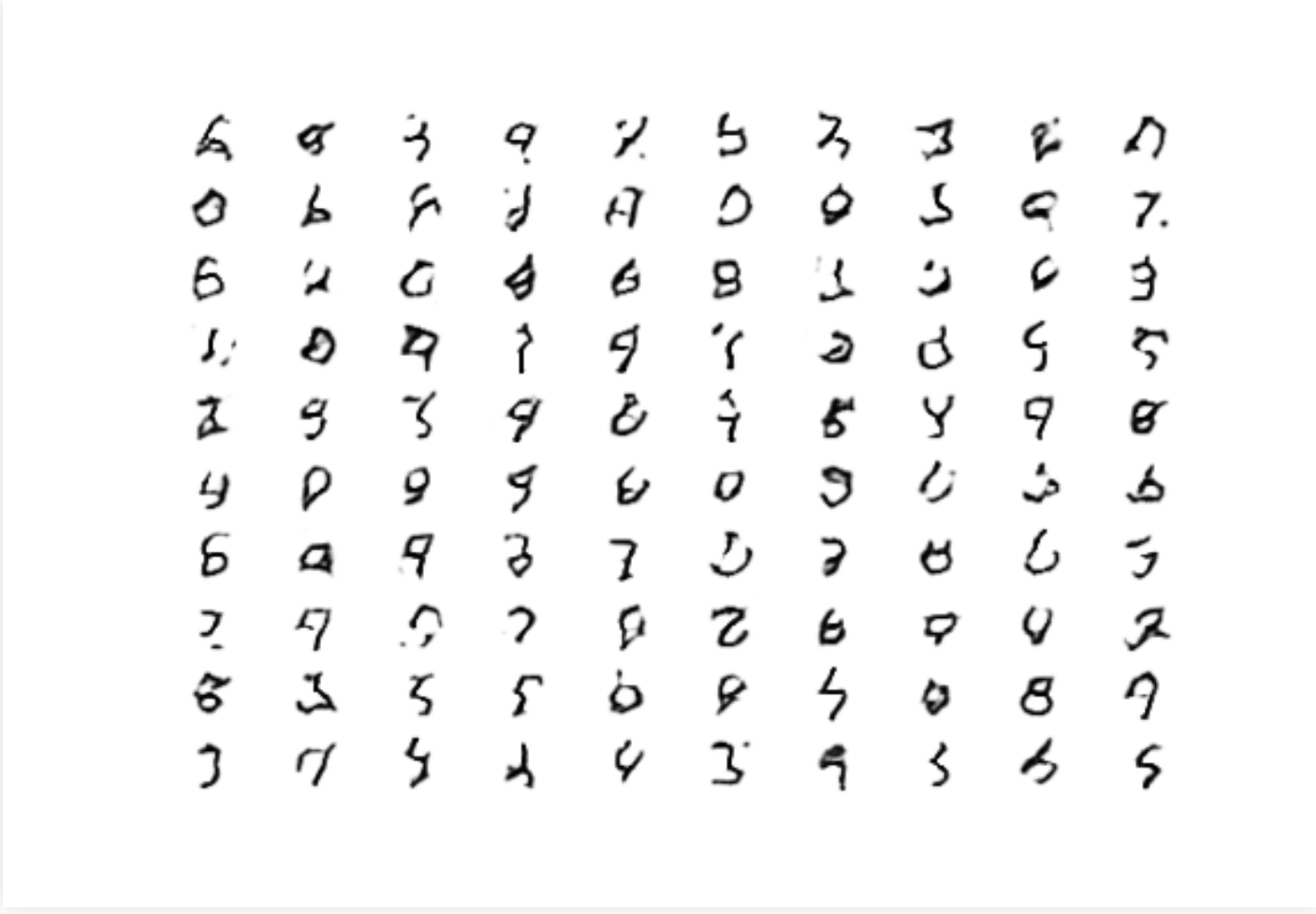
Implementing GAN

- Results after 1 epoch



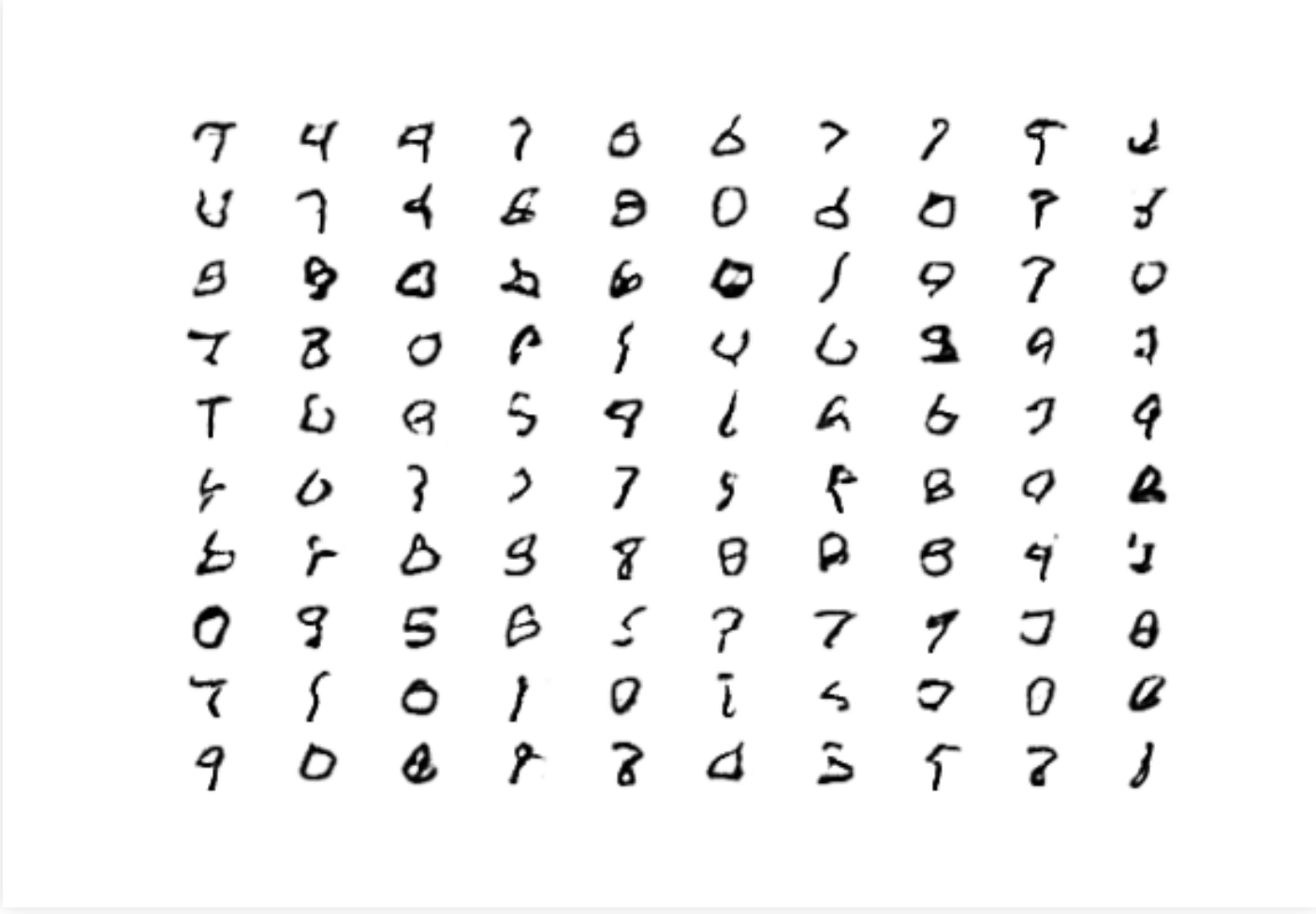
Implementing GAN

- Results after 25 epochs



Implementing GAN

- Results after 50 epochs



Summary

Generating images

Summary

- Variational Autoencoders

- Encoder learns parameters of probability distributions
- Decoder is fed random samples from such distribution
- A regularization term controls the distributions to improve the manifold
- This makes it easy to generate new examples

- Generative Adversarial Networks

- Generator learns to fool discriminator by creating images from random values
- Discriminator is trained in paralel to force generator to improve

Further reading:

- Goodfellow et.al, Deep learning, Sections 20.10.3 and 20.10.4
- (Optional) Brownlee, Generative Adversarial Networks with Python