Unsupervised approach

Geena Kim

University of Colorado Boulder

Unsupervised approach

Yann LeCun

Need tremendous amount of information to build machines that have common sense and generalize

[LeCun-20161205-NeurIPS-keynote]

- **"Pure" Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Unsupervised/Predictive Learning (cake)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

LeCake

- (Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)

Unsupervised approach

- Density modeling (generative tasks)

- Self-supervision

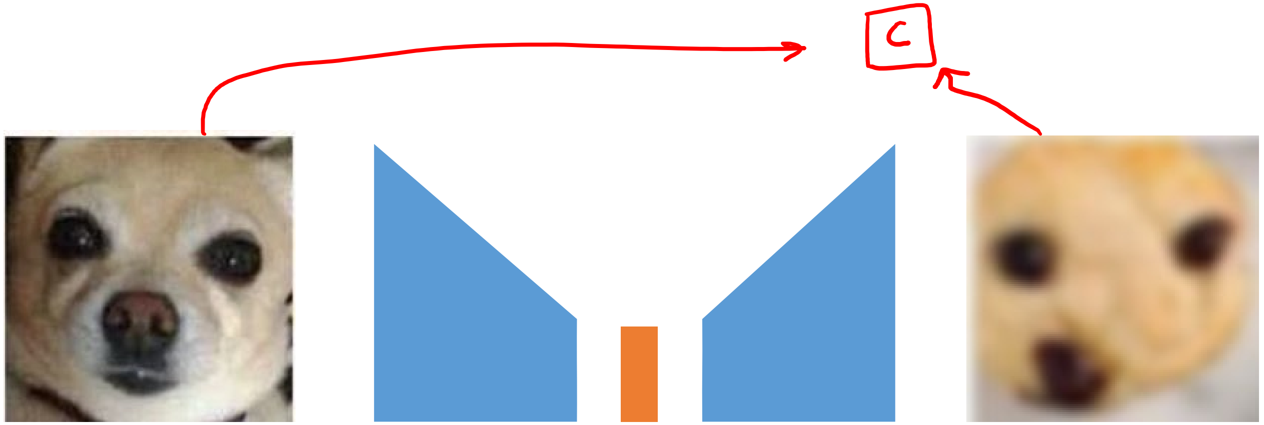
- Adversarial training

Unsupervised approach

Generative approach: generate the data distribution- generate images, text, audio, video, style transfer (density modeling, Autoencoders, GANs, etc)

Unsupervised approach

Adversarial training:



https://thispersondoesnotexist.com/

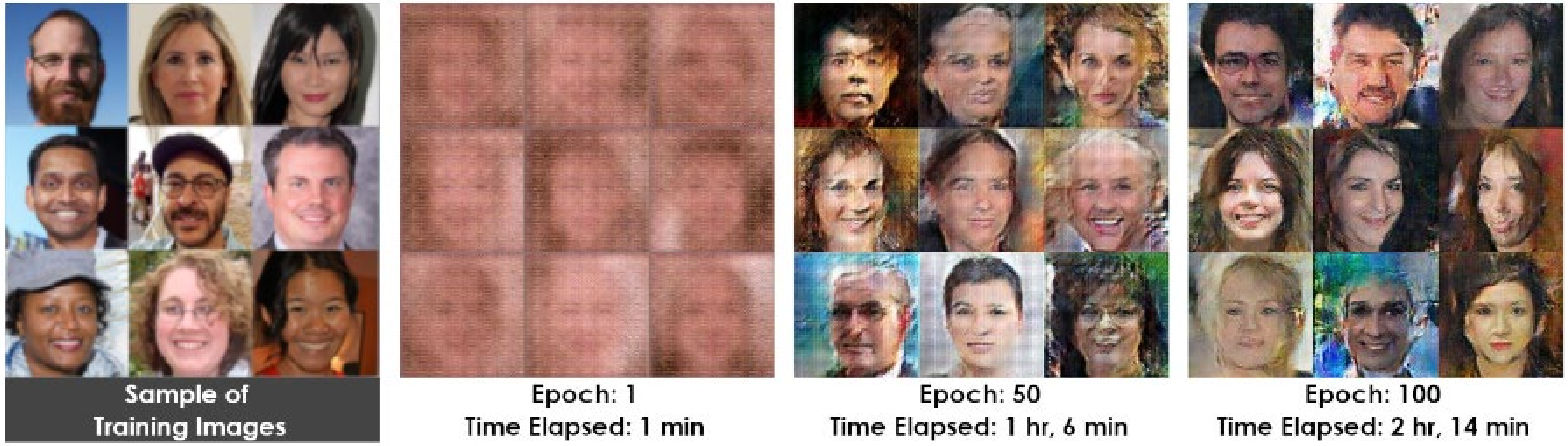
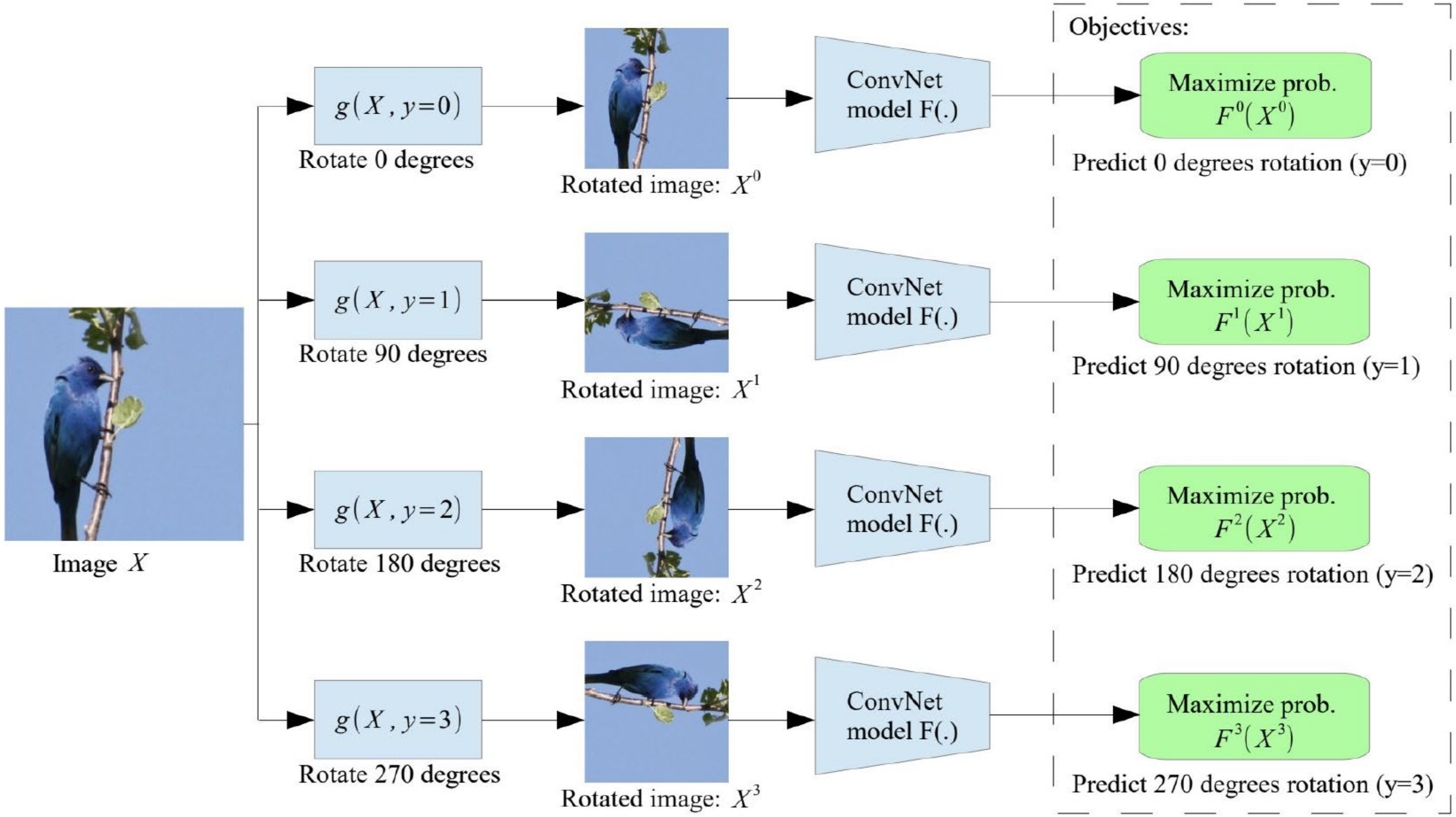


Image credit: https://cpang4.github.io/gan/

Unsupervised approach

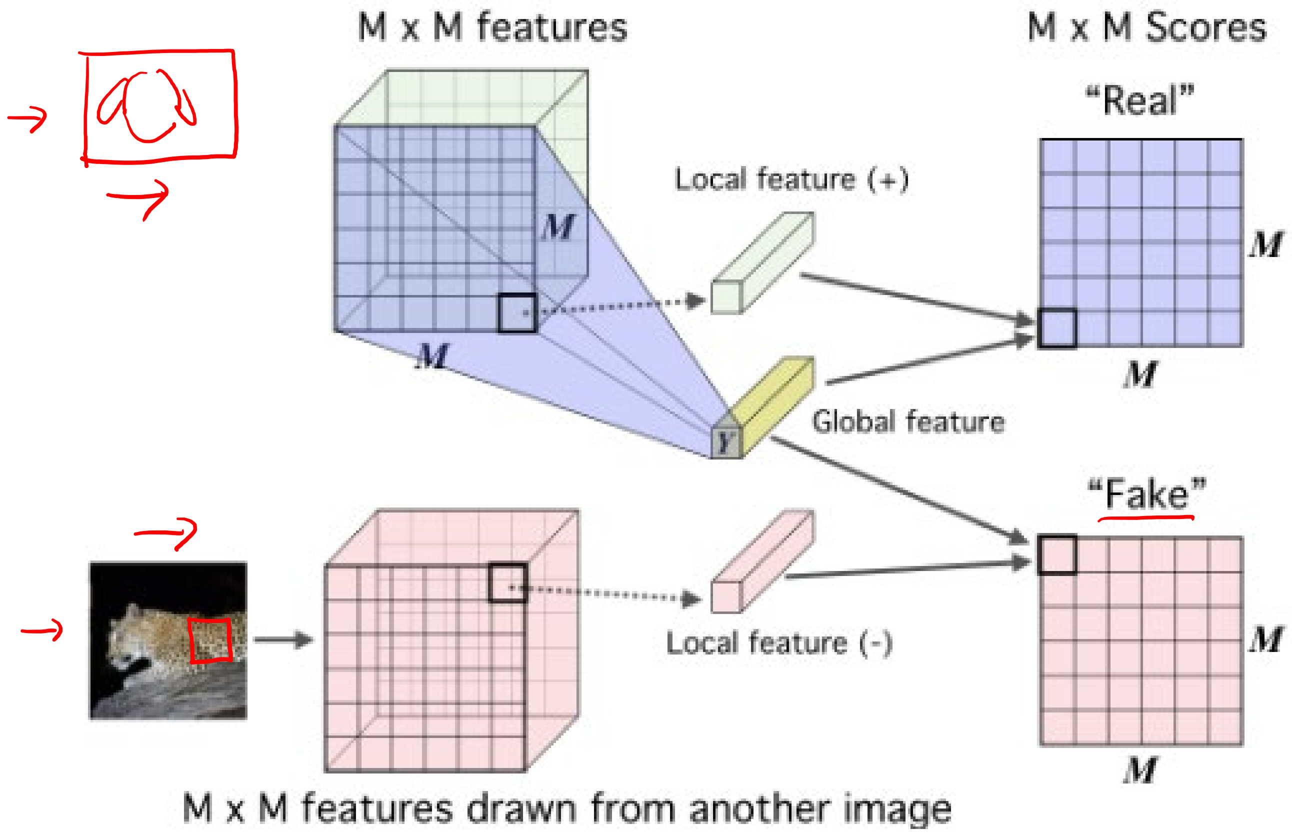
Self-supervision: Create easy label from data (surrogate task)



https://arxiv.org/pdf/1803.07728.pdf

Unsupervised approach

Self-supervision:



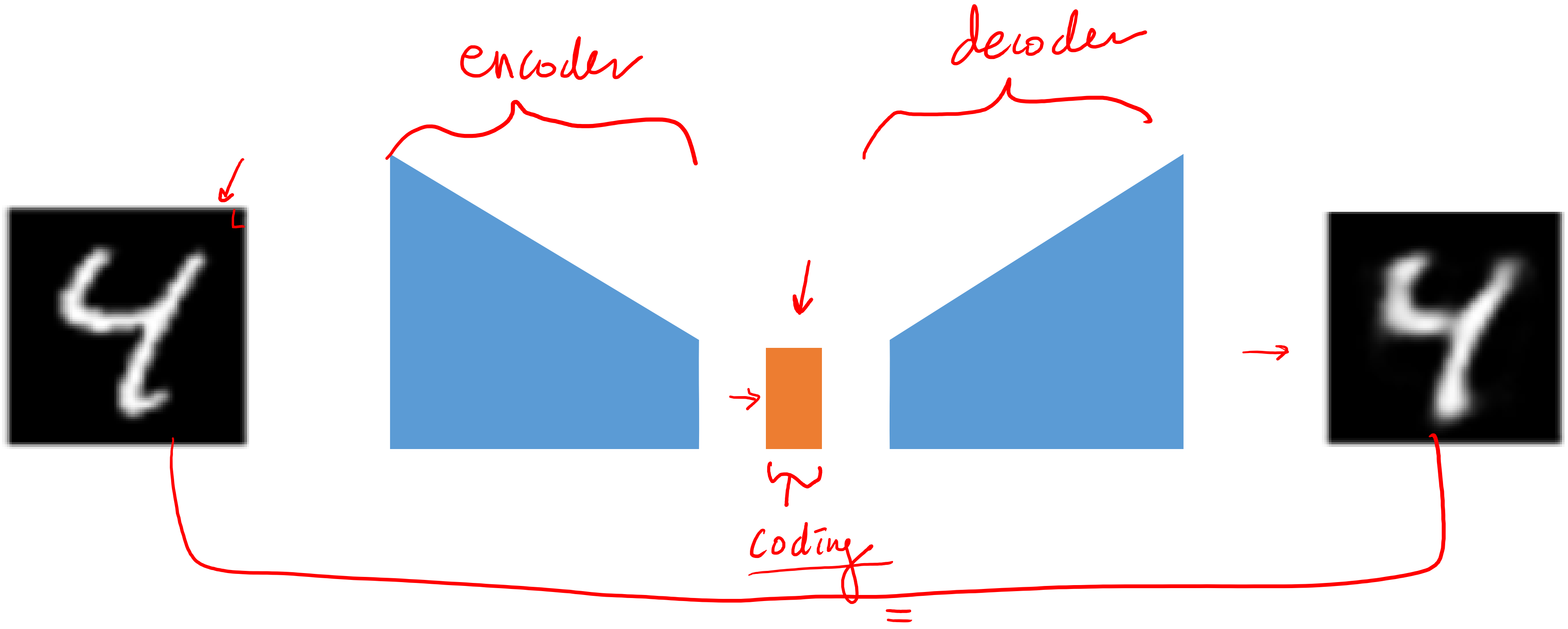
https://arxiv.org/pdf/1808.06670.pdf

Autoencoders

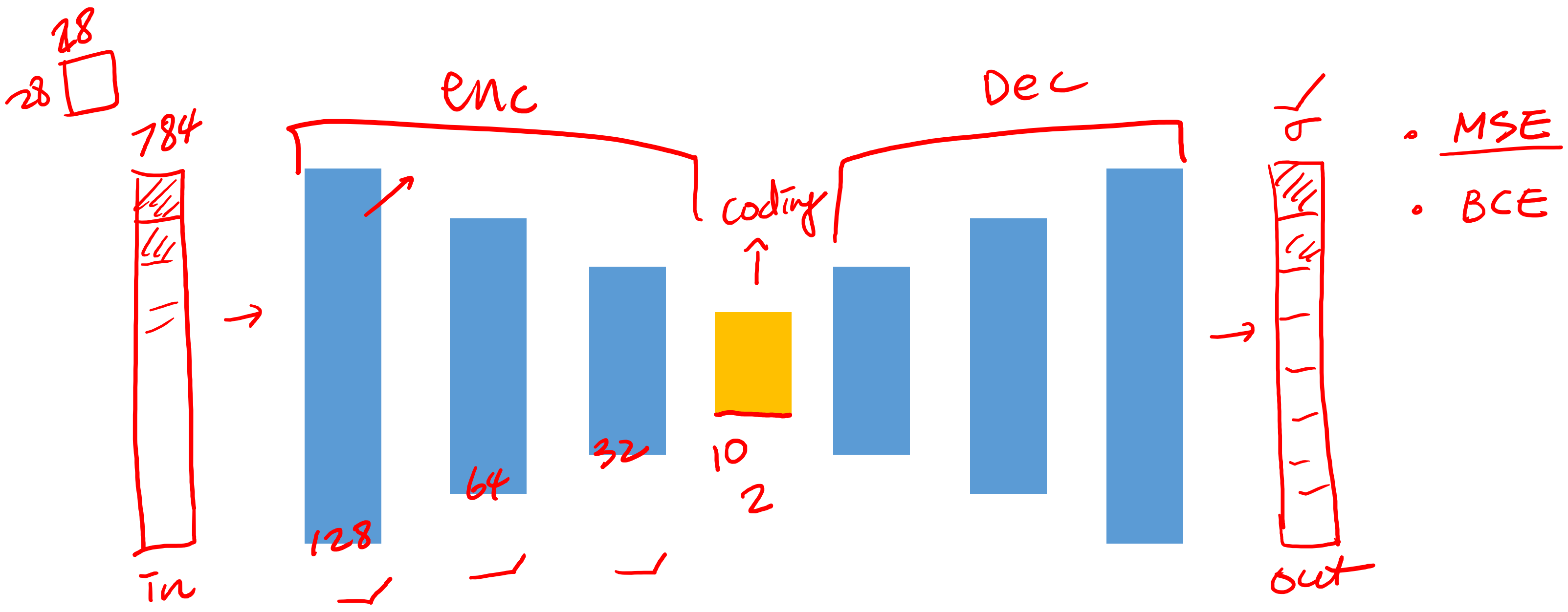
Geena Kim

University of Colorado Boulder

Autoencoders



Deep Autoencoders



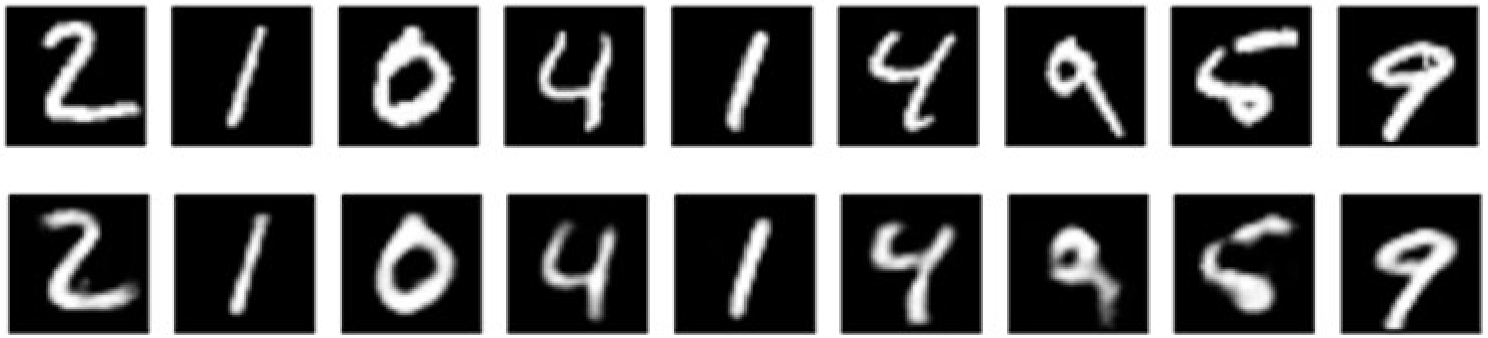


Image credit: François Chollet, https://blog.keras.io/building-autoencoders-in-keras.html

Denoising Autoencoders

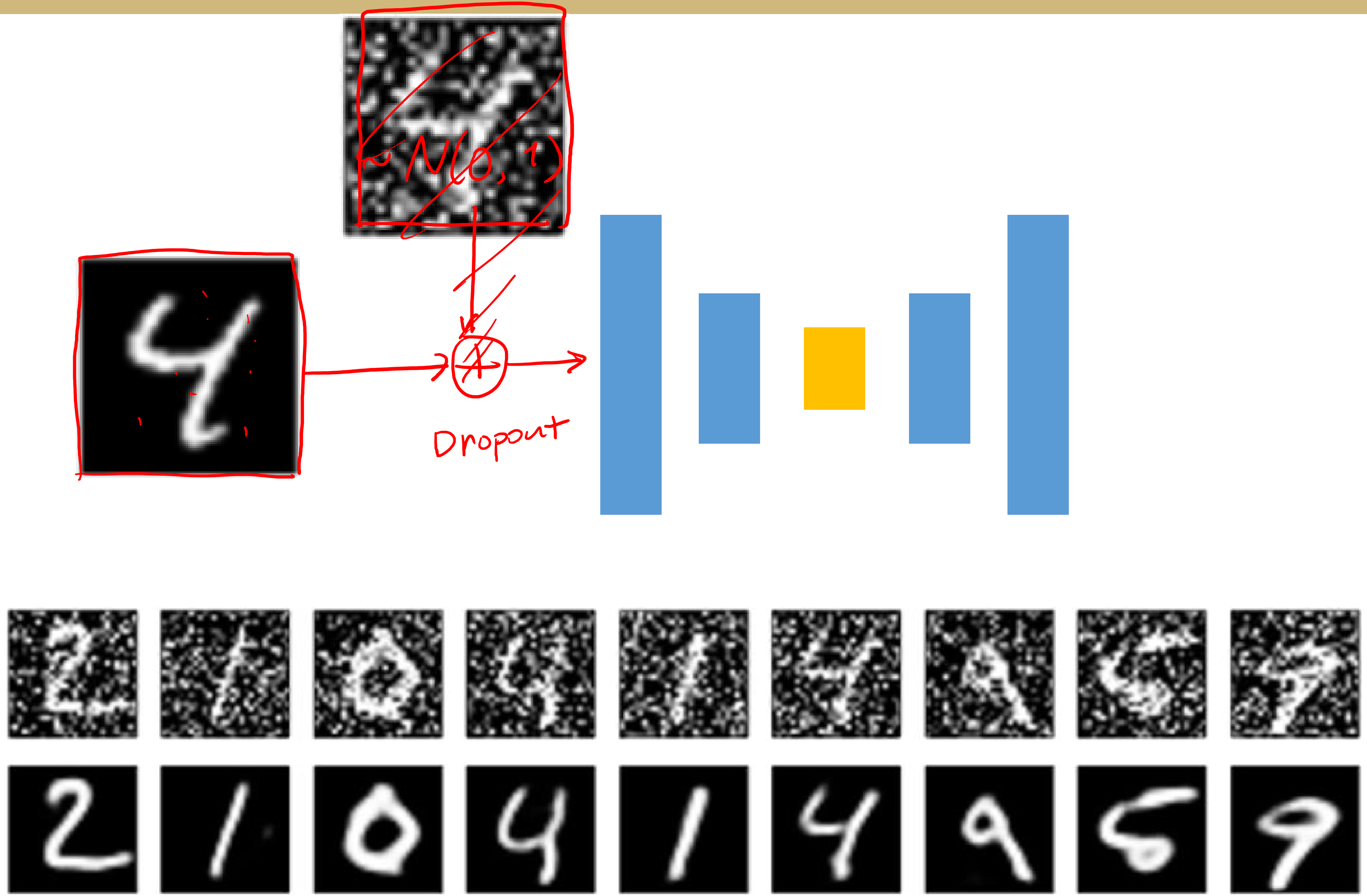
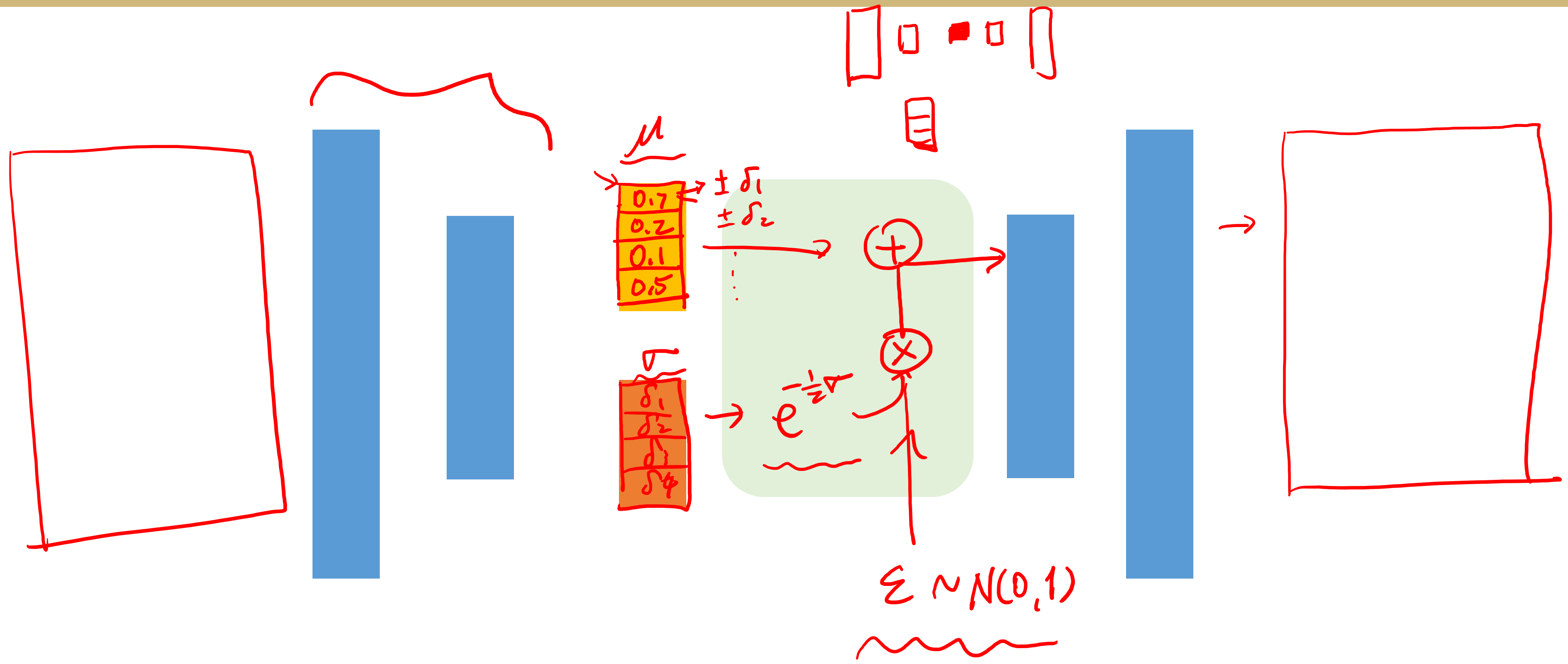


Image credit: François Chollet

Variational Autoencoders



Variational Autoencoders

VAE loss

$$\text{VAE Loss} = \underbrace{\text{Reconstruction loss}}_{= \text{MSE / BCE}} + \underbrace{\text{KL loss}}$$

$$\mathcal{L}(\boldsymbol{\theta}, \phi; \mathbf{x}^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\right) + \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)} | \mathbf{z}^{(i,l)})$$

$$\text{where} \quad \mathbf{z}^{(i,l)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(l)} \quad \text{and} \quad \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(0, \mathbf{I})$$

Kingma and Welling, https://arxiv.org/pdf/1312.6114.pdf

VAE implementation example

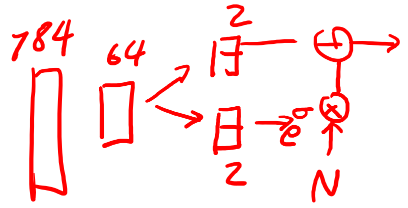
```python
original_dim = 28 * 28
intermediate_dim = 64
latent_dim = 2
```

```python
inputs = keras.Input(shape=(original_dim,))
h = layers.Dense(intermediate_dim, activation='relu')(inputs)
z_mean = layers.Dense(latent_dim)(h)
z_log_sigma = layers.Dense(latent_dim)(h)
```

```python
from keras import backend as K

def sampling(args):
    z_mean, z_log_sigma = args
    epsilon = K.random_normal(shape=(K.shape(z_mean)[0], latent_dim),
                              mean=0., stddev=0.1)
    return z_mean + K.exp(z_log_sigma) * epsilon

z = layers.Lambda(sampling)([z_mean, z_log_sigma])
```



Source: https://blog.keras.io/building-autoencoders-in-keras.html

VAE implementation example

```
# Create encoder
encoder = keras.Model(inputs, [z_mean, z_log_sigma, z], name='encoder')

# Create decoder
latent_inputs = keras.Input(shape=(latent_dim,), name='z_sampling')
x = layers.Dense(intermediate_dim, activation='relu')(latent_inputs)
outputs = layers.Dense(original_dim, activation='sigmoid')(x)
decoder = keras.Model(latent_inputs, outputs, name='decoder')
```

```
# instantiate VAE model
outputs = decoder(encoder(inputs)[2])
vae = keras.Model(inputs, outputs, name='vae_mlp')
```

VAE implementation example

```
reconstruction_loss = keras.losses.binary_crossentropy(inputs, outputs)
reconstruction_loss *= original_dim
kl_loss = 1 + z_log_sigma - K.square(z_mean) - K.exp(z_log_sigma)
kl_loss = K.sum(kl_loss, axis=-1)
kl_loss *= -0.5
vae_loss = K.mean(reconstruction_loss + kl_loss)
vae.add_loss(vae_loss)
vae.compile(optimizer='adam')
```

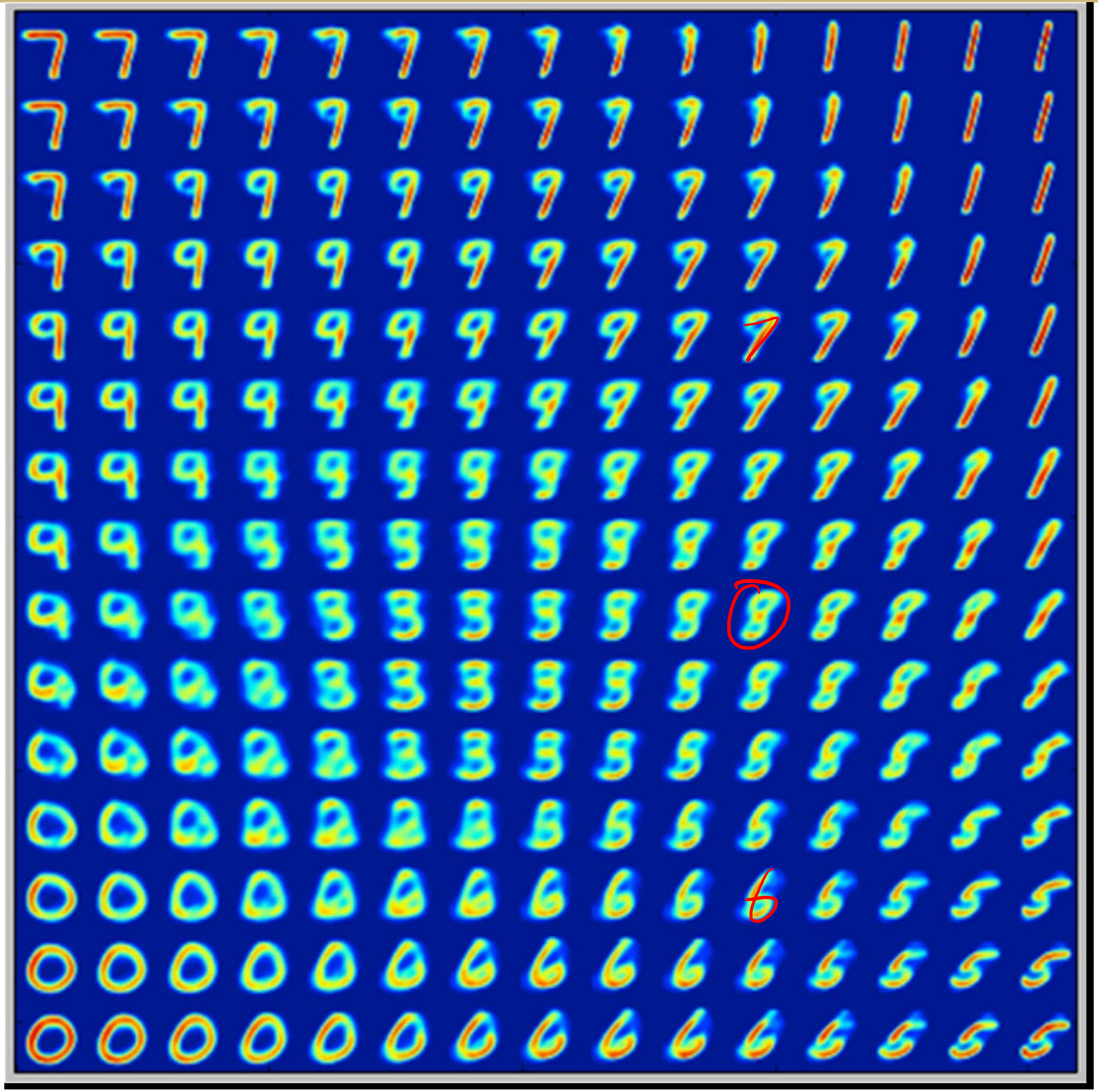
$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\right) + \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)} | \mathbf{z}^{(i,l)})$$

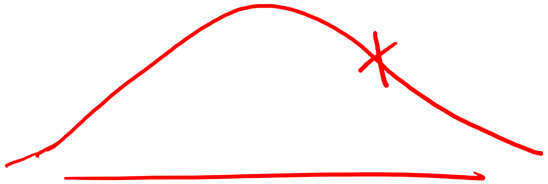
where $\quad \mathbf{z}^{(i,l)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(l)} \quad$ and $\quad \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(0, \mathbf{I})$

Why is VAE useful?

- Good for learning latent representation
- We can control the model
- Generative model
- Can interpolate in latent feature space

VAE applications





VAE applications

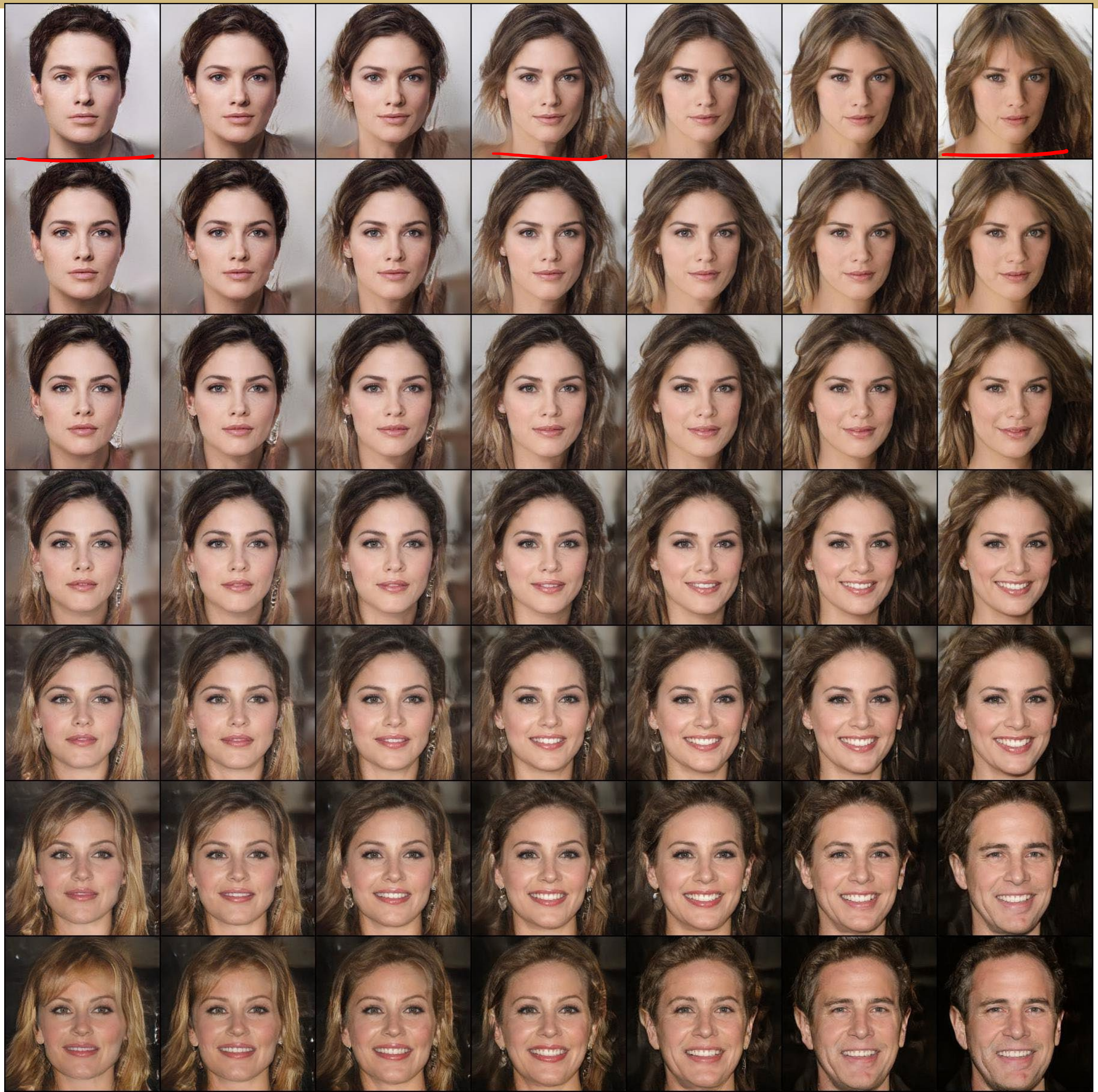


Image credit: Tal Daniel and Aviv Tamar, https://arxiv.org/pdf/2012.13253.pdf

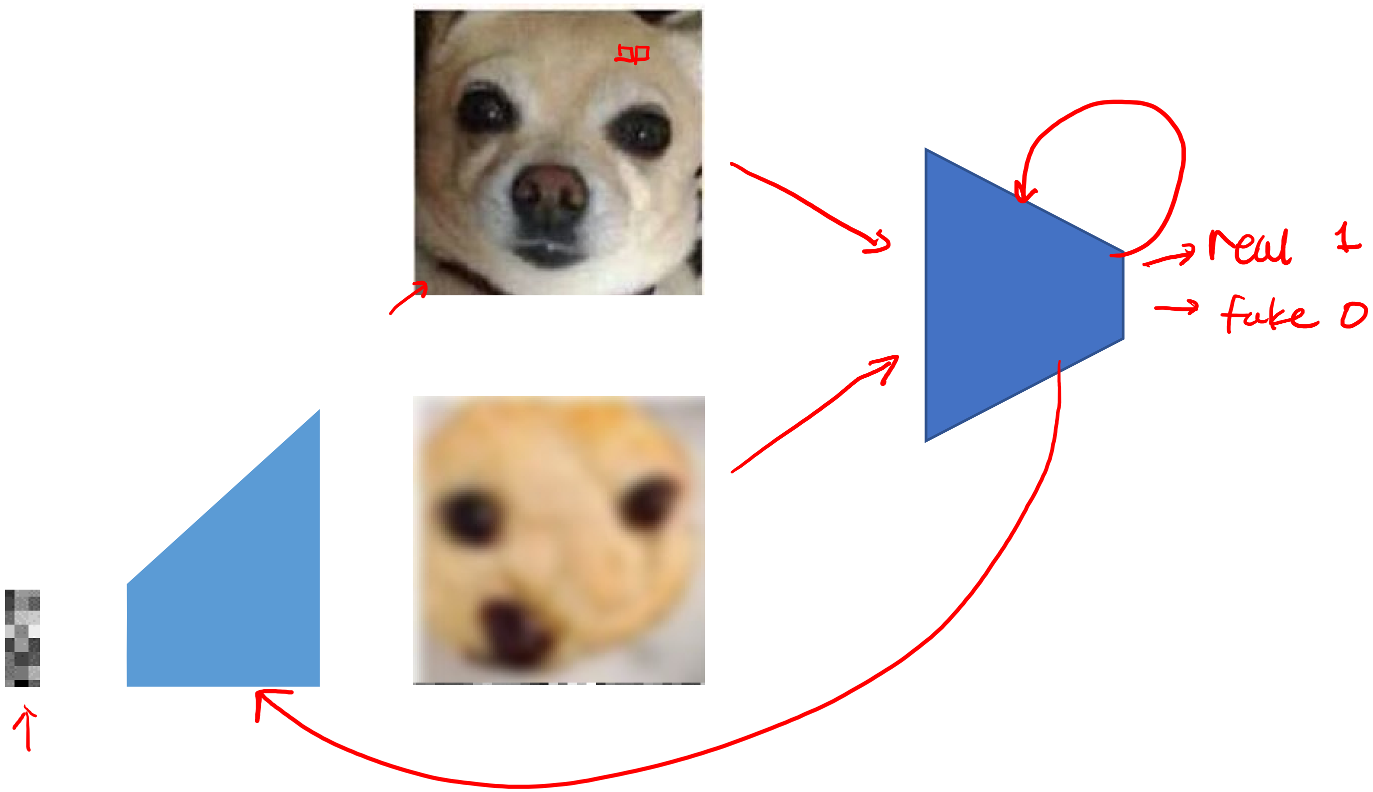
VAE applications

https://magenta.tensorflow.org/music-vae

Generative Adversarial Networks

Geena Kim

Generative Adversarial Networks



Minimax Loss

$$\min_{G} \max_{D} V(D,G)$$

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

https://arxiv.org/abs/1406.2661

GAN training

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

1 D

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

$$\underbrace{-\log(D(G(z)))}$$

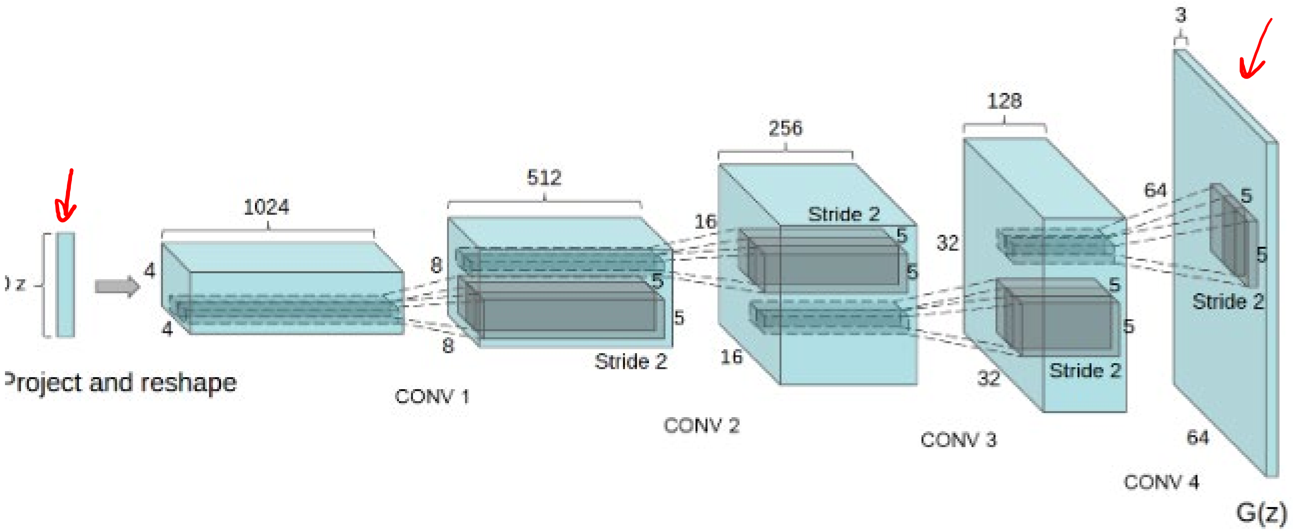
modified minimax

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

https://arxiv.org/abs/1406.2661

DCGAN



Minimax loss + SGD optimizer

https://arxiv.org/abs/1511.06434

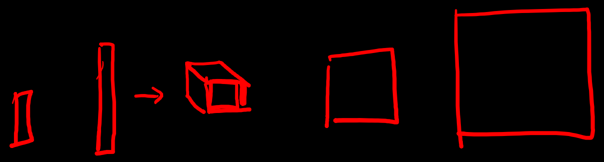
DCGAN implementation example

```
discriminator = keras.Sequential(
    [
        keras.Input(shape=(64, 64, 3)),
        layers.Conv2D(64, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(128, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(128, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Flatten(),
        layers.Dropout(0.2),
        layers.Dense(1, activation="sigmoid"),
    ],
    name="discriminator",
)
discriminator.summary()
```

0 – 1

https://keras.io/examples/generative/dcgan_overriding_train_step/

DCGAN implementation example



```python
latent_dim = 128

generator = keras.Sequential(
    [
        keras.Input(shape=(latent_dim,)),
        layers.Dense(8 * 8 * 128),
        layers.Reshape((8, 8, 128)),
        layers.Conv2DTranspose(128, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(256, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2DTranspose(512, kernel_size=4, strides=2, padding="same"),
        layers.LeakyReLU(alpha=0.2),
        layers.Conv2D(3, kernel_size=5, padding="same", activation="sigmoid"),
    ],
    name="generator",
)
generator.summary()
```

https://keras.io/examples/generative/dcgan_overriding_train_step/

DCGAN implementation example

```
# Combine them with real images
combined_images = tf.concat([generated_images, real_images], axis=0)

# Assemble labels discriminating real from fake images
labels = tf.concat(
    [tf.ones((batch_size, 1)), tf.zeros((batch_size, 1))], axis=0
)
# Add random noise to the labels - important trick!
labels += 0.05 * tf.random.uniform(tf.shape(labels))

# Train the discriminator
with tf.GradientTape() as tape:
    predictions = self.discriminator(combined_images)
    d_loss = self.loss_fn(labels, predictions)
grads = tape.gradient(d_loss, self.discriminator.trainable_weights)
self.d_optimizer.apply_gradients(
    zip(grads, self.discriminator.trainable_weights)
)
```

https://keras.io/examples/generative/dcgan_overriding_train_step/

DCGAN implementation example

```
# Sample random points in the latent space
random_latent_vectors = tf.random.normal(shape=(batch_size, self.latent_dim))

# Assemble labels that say "all real images"
misleading_labels = tf.zeros((batch_size, 1))

# Train the generator (note that we should *not* update the weights
# of the discriminator)!
with tf.GradientTape() as tape:
    predictions = self.discriminator(self.generator(random_latent_vectors))
    g_loss = self.loss_fn(misleading_labels, predictions)  BCE
grads = tape.gradient(g_loss, self.generator.trainable_weights)
self.g_optimizer.apply_gradients(zip(grads, self.generator.trainable_weights))
```

https://keras.io/examples/generative/dcgan_overriding_train_step/

DCGAN implementation example

DCGAN implementation in Keras

https://keras.io/examples/generative/dcgan_overriding_train_step/

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

Using tf.GradientTape

https://www.tensorflow.org/guide/autodiff