Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd edition by Aurélien Geron

Chapter 17 Representation Learning and Generative Learning Using Autoencoders and GANs

San Diego Machine Learning
2022 MAR 19
Discussion Leader: Robert Kraig

Chapter 17: Main Ideas

Methods

- Autoencoders
- Generative Adversarial Networks (GANs)

Goals

- Representation Learning
- Denoising
- Data Generation

Efficient Data Representations

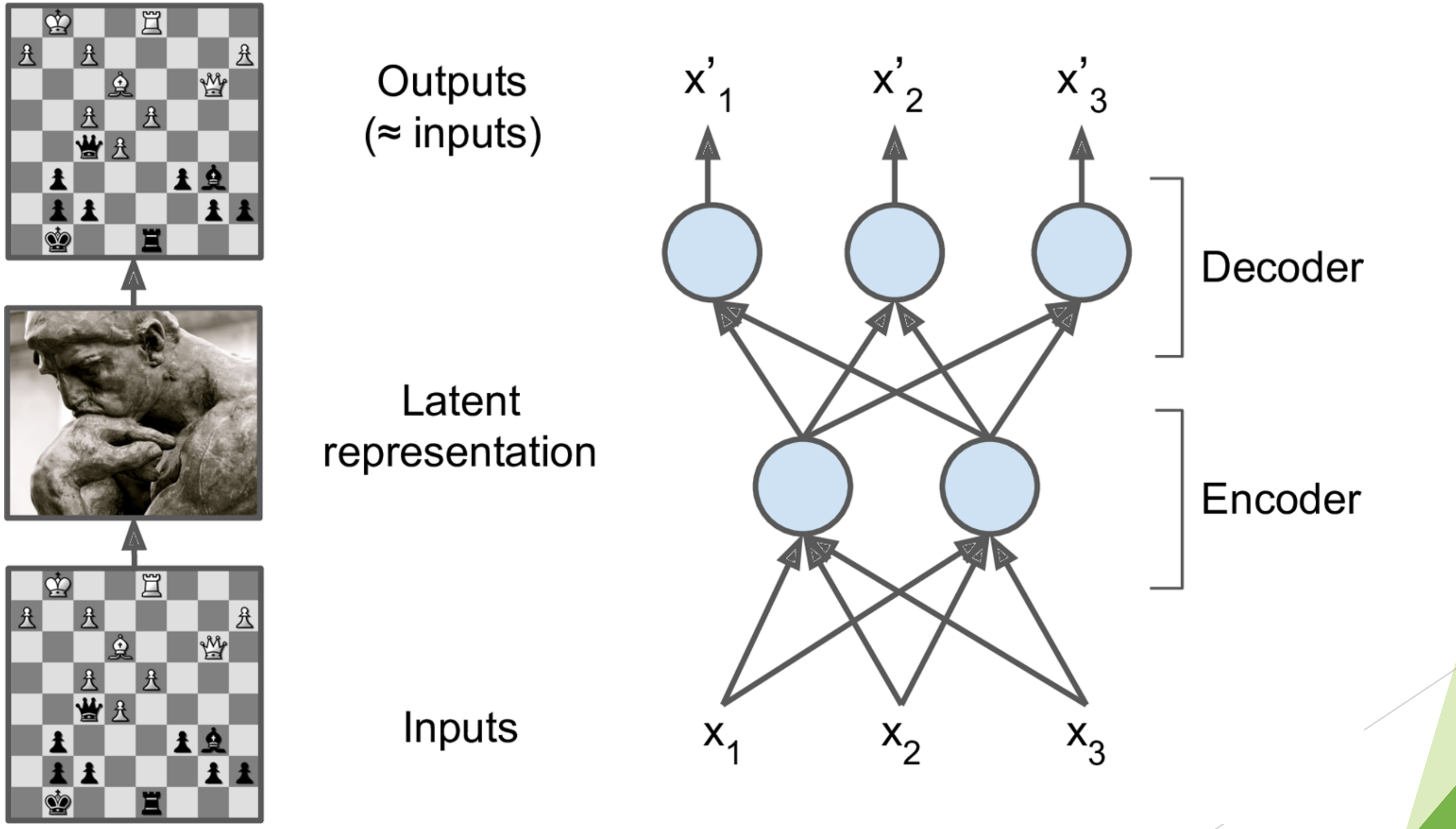
Which of the following number sequences do you find the easiest to memorize?

- 40, 27, 25, 36, 81, 57, 10, 73, 19, 68
- 50, 48, 46, 44, 42, 40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14

Efficient Data Representations (cont’d)

- The quick brown fox jumps over the lazy dog.
- An inscrutable lemur ponders a northern bill.
- Box jaundice toward hula daal platypus vacate.
- Gupar gui yapeo bijar numpevac ilu fubo zoad.
- Guwkd fruywv rwjrk wripsdkdf w eer ewpglpae.

Efficient Data Representations

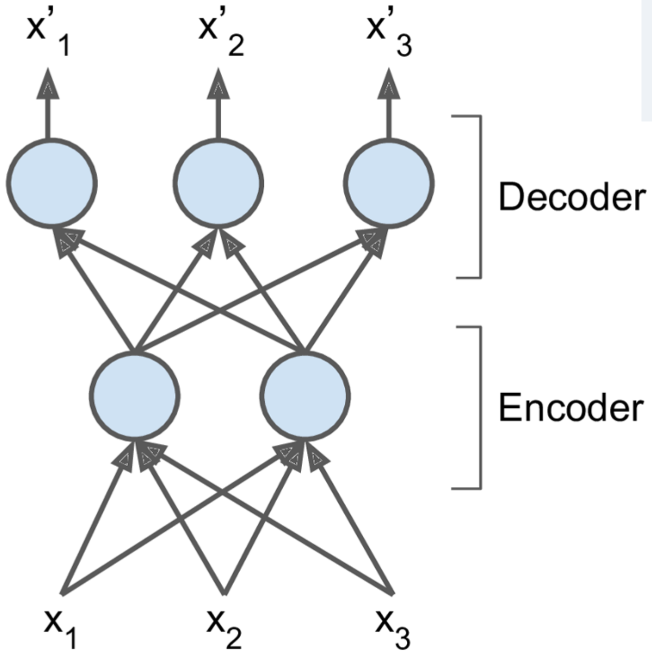


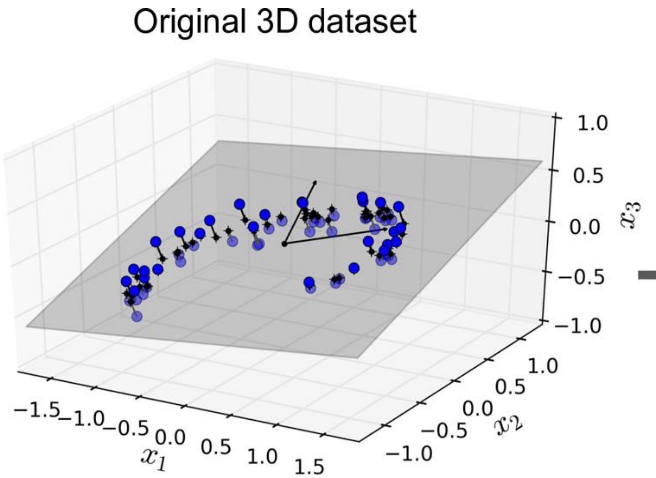
Relation between Cross Entropy and Information Theory

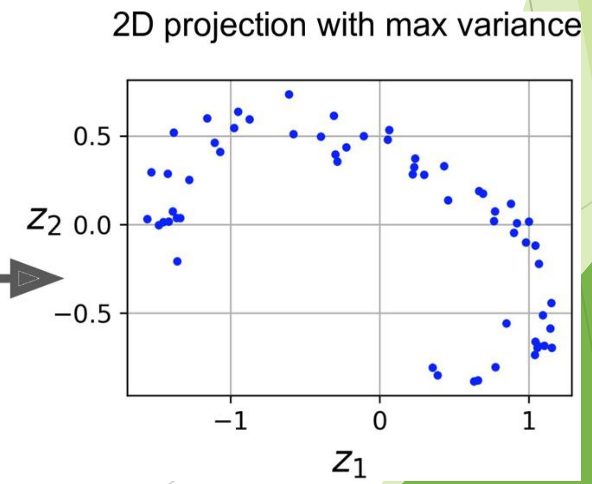
PCA is an Undercomplete Autoencoder

```
encoder = keras.models.Sequential([keras.layers.Dense(2, input_shape=[3])])
decoder = keras.models.Sequential([keras.layers.Dense(3, input_shape=[2])])
autoencoder = keras.models.Sequential([encoder, decoder])

autoencoder.compile(loss="mse", optimizer=keras.optimizers.SGD(learning_rate=1.5))
```

```
history = autoencoder.fit(X_train, X_train, epochs=20)
codings = encoder.predict(X_train)
```

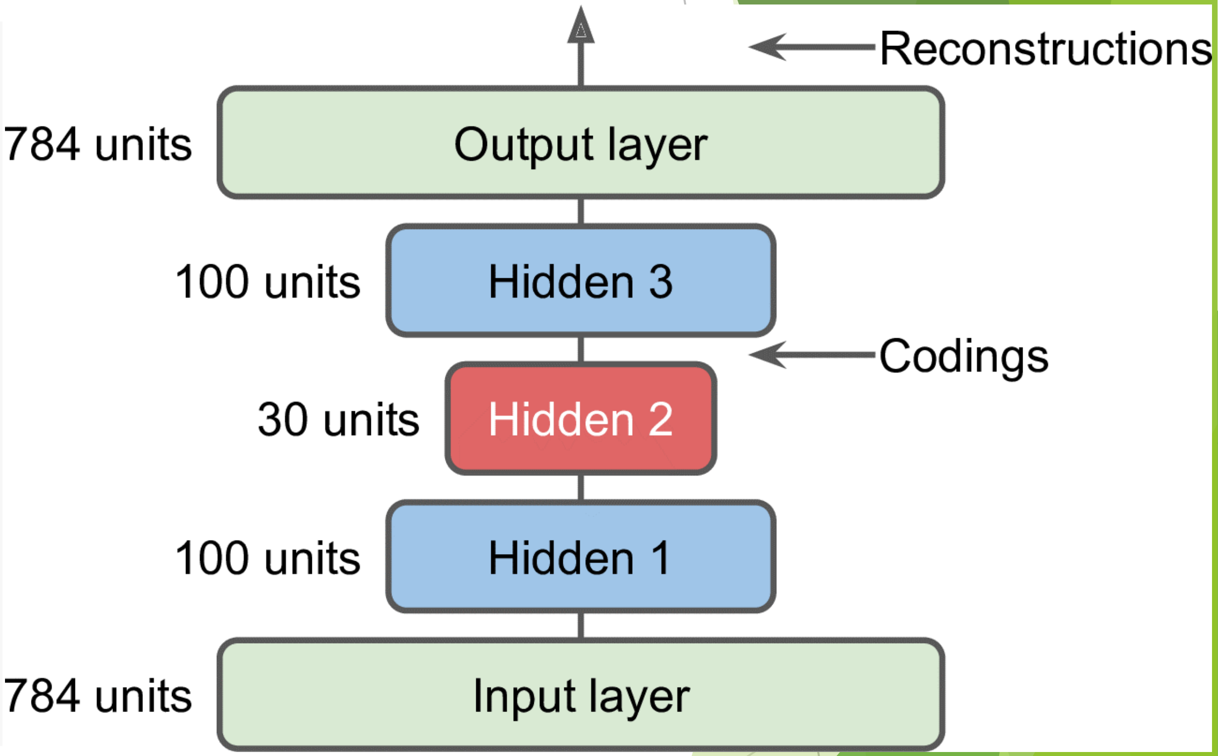






Stacked (Deep) Autoencoders

```
stacked_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(30, activation="selu"),
])
stacked_decoder = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[30]),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
stacked_ae = keras.models.Sequential([stacked_encoder, stacked_decoder])
stacked_ae.compile(loss="binary_crossentropy",
                   optimizer=keras.optimizers.SGD(learning_rate=1.5),
                   metrics=[rounded_accuracy])
history = stacked_ae.fit(X_train, X_train, epochs=20,
                         validation_data=(X_valid, X_valid))
```



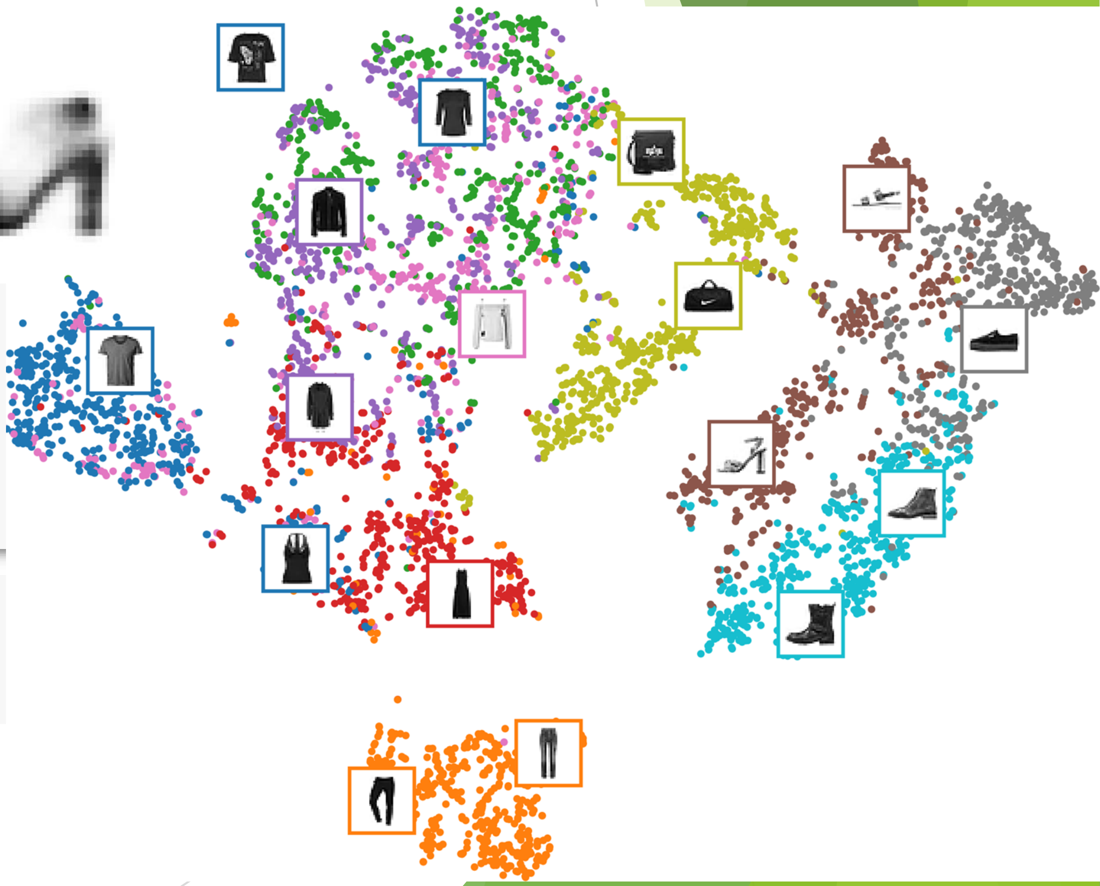
Visualizing Fashion MNIST

original

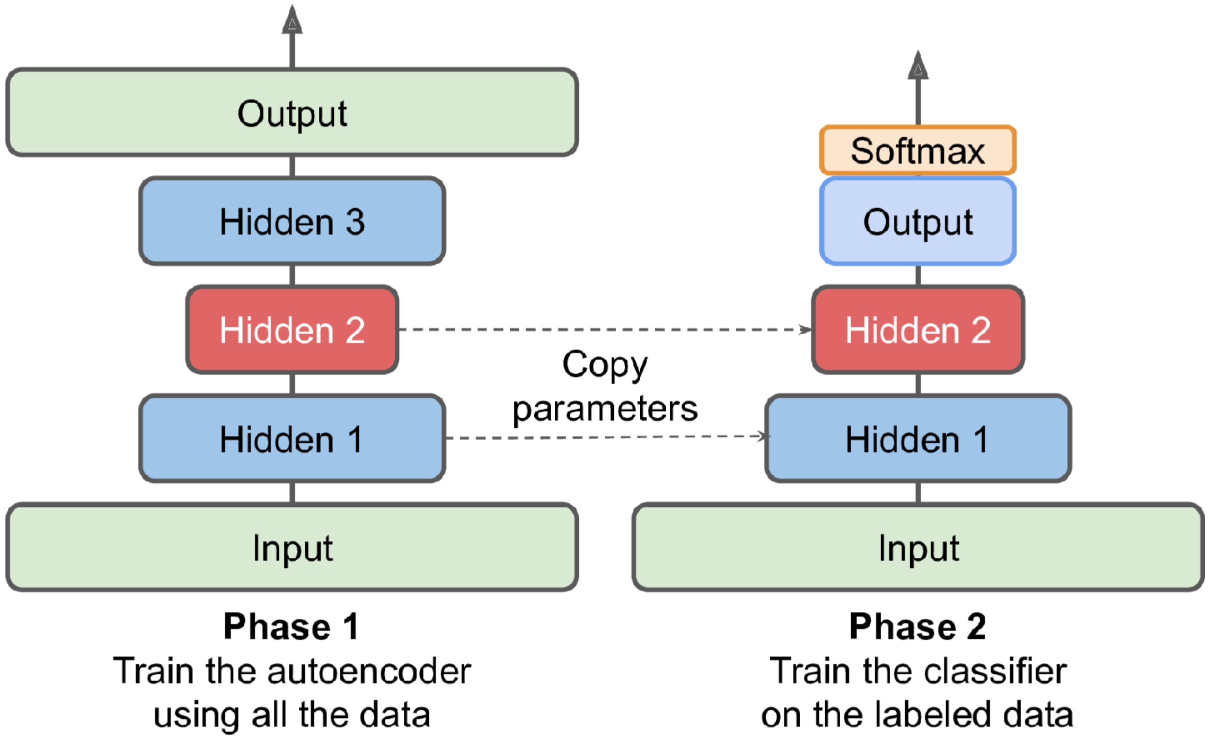
reconstructed

```python
from sklearn.manifold import TSNE

X_valid_compressed = stacked_encoder.predict(X_valid)
tsne = TSNE()
X_valid_2D = tsne.fit_transform(X_valid_compressed)
X_valid_2D = (X_valid_2D - X_valid_2D.min()) / (X_valid_2D.max() - X_valid_2D.min())
```

```python
plt.scatter(X_valid_2D[:, 0], X_valid_2D[:, 1], c=y_valid, s=10, cmap="tab10")
plt.axis("off")
plt.show()
```



Unsupervised Pretraining



Tying Weights

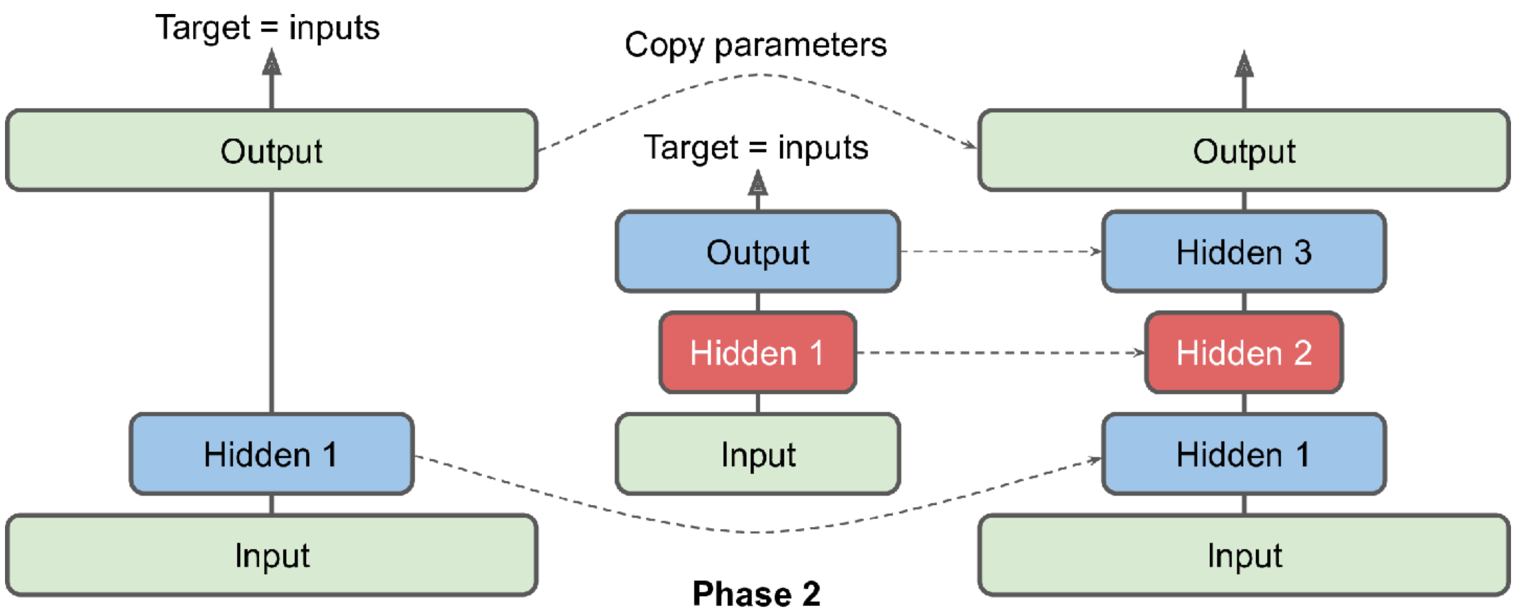
- Use the same matrices for encoding and decoding!

```python
class DenseTranspose(keras.layers.Layer):
    def __init__(self, dense, activation=None, **kwargs):
        self.dense = dense
        self.activation = keras.activations.get(activation)
        super().__init__(**kwargs)
    def build(self, batch_input_shape):
        self.biases = self.add_weight(name="bias",
                                      shape=[self.dense.input_shape[-1]],
                                      initializer="zeros")
        super().build(batch_input_shape)
    def call(self, inputs):
        z = tf.matmul(inputs, self.dense.weights[0], transpose_b=True)
        return self.activation(z + self.biases)
```

```python
keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

dense_1 = keras.layers.Dense(100, activation="selu")
dense_2 = keras.layers.Dense(30, activation="selu")

tied_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    dense_1,
    dense_2
])

tied_decoder = keras.models.Sequential([
    DenseTranspose(dense_2, activation="selu"),
    DenseTranspose(dense_1, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])

tied_ae = keras.models.Sequential([tied_encoder, tied_decoder])

tied_ae.compile(loss="binary_crossentropy",
                optimizer=keras.optimizers.SGD(learning_rate=1.5), metrics=[rounded_accuracy])
history = tied_ae.fit(X_train, X_train, epochs=10,
                      validation_data=(X_valid, X_valid))
```

Training one Autoencoder at a Time

- Greedy Layerwise Training



Phase 1
Train the first autoencoder

Phase 2
Train the second autoencoder on the training set encoded by the first encoder

Phase 3
Stack the autoencoders

[G. Hinton et al. (2006): A Fast Learning Algorithm for Deep Belief Nets](#)
[Y. Bengio et al. (2007): Greedy Layer-Wise Training of Deep Networks](#)

Convolutional Autoencoders

```
conv_encoder.summary()

Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 reshape_2 (Reshape)         (None, 28, 28, 1)         0

 conv2d (Conv2D)             (None, 28, 28, 16)        160

 max_pooling2d (MaxPooling2D  (None, 14, 14, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 32)         0
 2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 64)          18496

 max_pooling2d_2 (MaxPooling  (None, 3, 3, 64)         0
 2D)

=================================================================
Total params: 23,296
Trainable params: 23,296
Non-trainable params: 0
```

```
conv_decoder.summary()

Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_transpose (Conv2DTra  (None, 7, 7, 32)         18464
 nspose)

 conv2d_transpose_1 (Conv2DT  (None, 14, 14, 16)       4624
 ranspose)

 conv2d_transpose_2 (Conv2DT  (None, 28, 28, 1)        145
 ranspose)

 reshape_3 (Reshape)         (None, 28, 28)            0

=================================================================
Total params: 23,233
Trainable params: 23,233
Non-trainable params: 0
```

J. Masci et al. (2011): Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction

Recurrent Autoencoders

```
recurrent_encoder.summary()
```

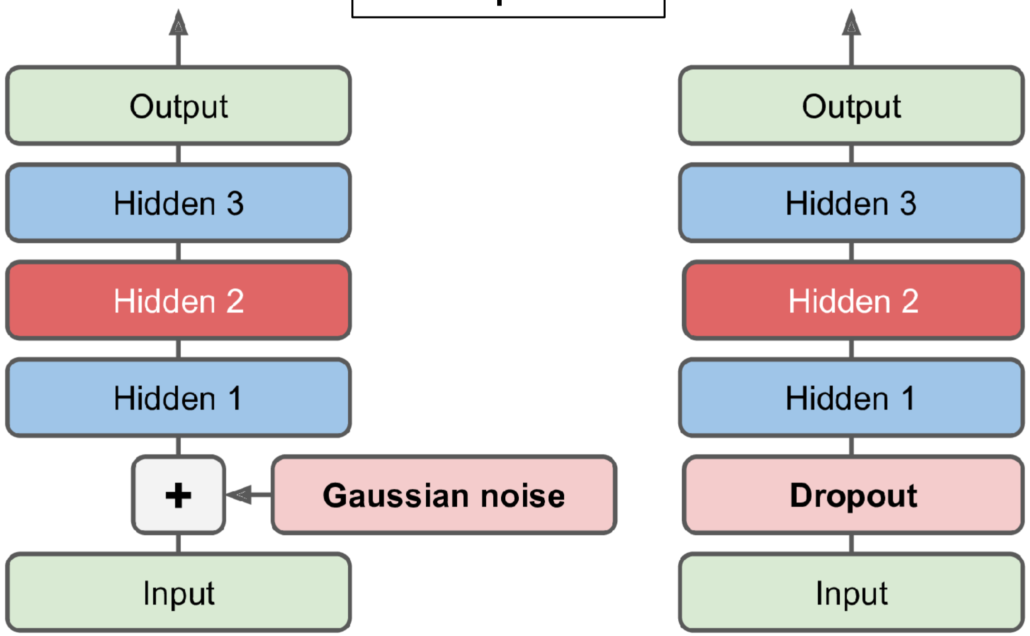
```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 28, 100)           51600

 lstm_1 (LSTM)               (None, 30)                15720

=================================================================
Total params: 67,320
Trainable params: 67,320
Non-trainable params: 0
```

```
recurrent_decoder.summary()
```

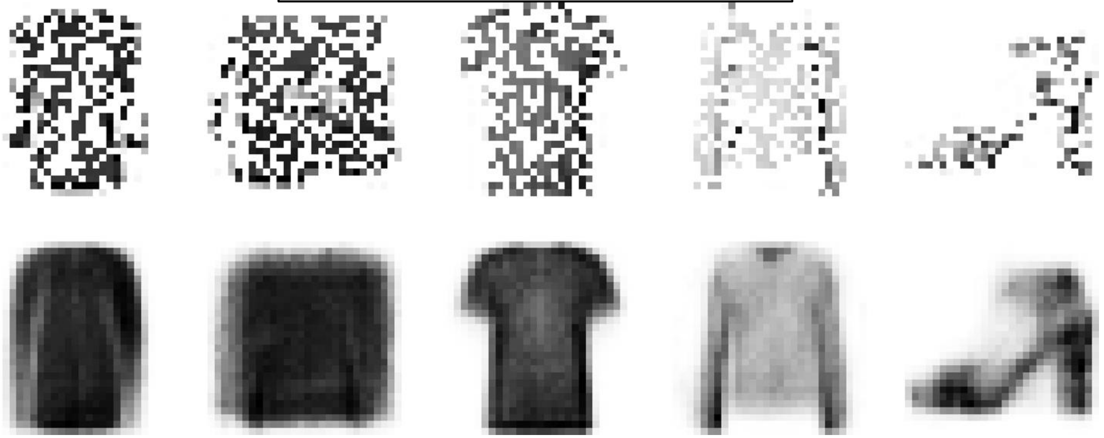
```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 repeat_vector (RepeatVector  (None, 28, 30)           0
 )

 lstm_2 (LSTM)               (None, 28, 100)           52400

 time_distributed (TimeDistr  (None, 28, 28)           2828
 ibuted)

=================================================================
Total params: 55,228
Trainable params: 55,228
Non-trainable params: 0
```

Denoising Autoencoders

Two options:



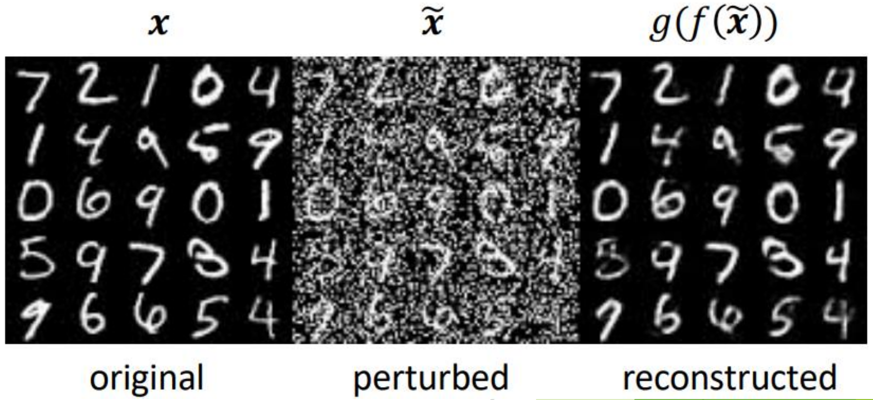
Results from Dropout:



Denoising Autoencoder

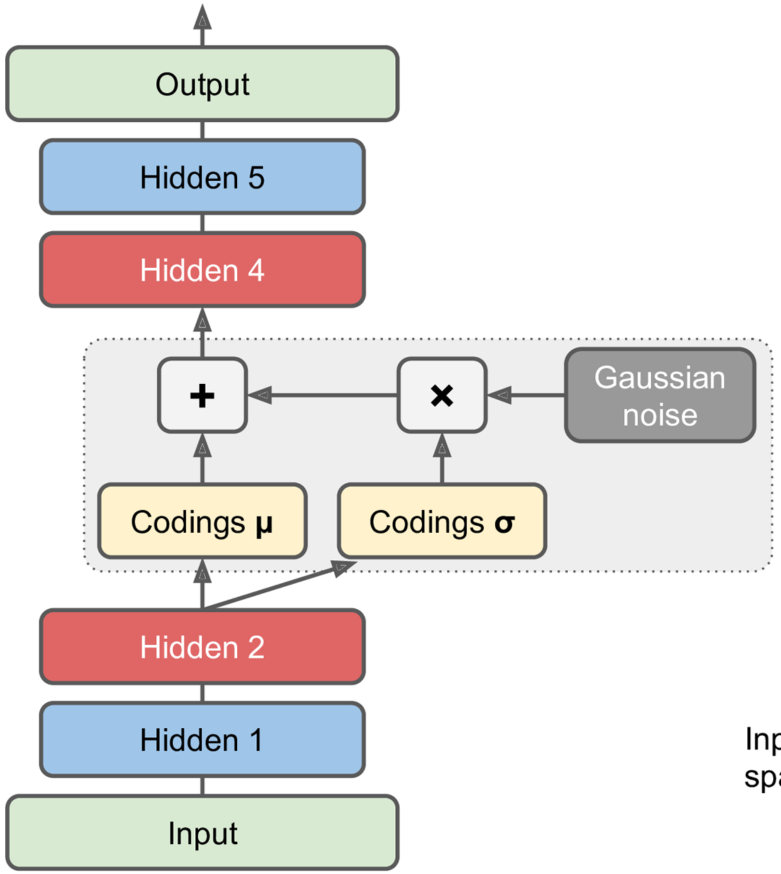
- Consider noisy version $\tilde{\boldsymbol{x}}$ of the input $\boldsymbol{x}$
- Data denoising

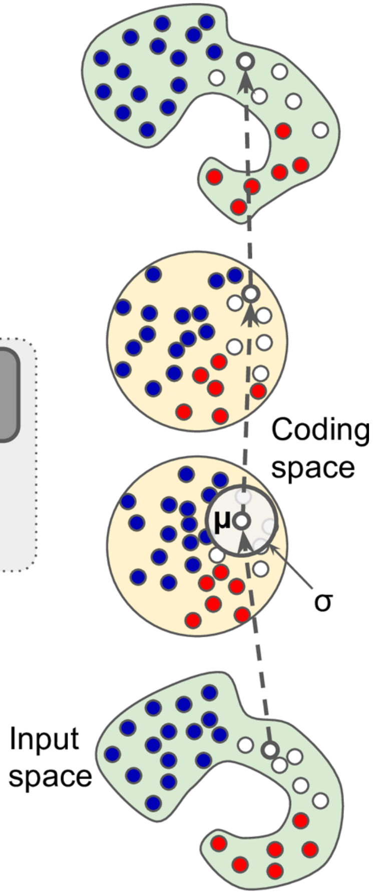
$$\min_{\boldsymbol{W}} \frac{1}{2}\sum_n \left\| g\left(f(\tilde{\boldsymbol{x}}_n; \boldsymbol{W}_f); \boldsymbol{W}_g\right) - \boldsymbol{x}_n \right\|_2^2 + c \left\| f(\tilde{\boldsymbol{x}}_n; \boldsymbol{W}_f) \right\|_1$$



https://cs.uwaterloo.ca/~ppoupart/teaching/cs480-spring19/slides/cs480-lecture20.pdf

Variational Autoencoders





- Idea: train encoder $\Pr(\boldsymbol{h}|\boldsymbol{x};\boldsymbol{W}_f)$ to approach a simple and fixed distribution, e.g., $N(\boldsymbol{h};\boldsymbol{0},\boldsymbol{I})$
- This way we can set $\Pr(\boldsymbol{h})$ to $N(\boldsymbol{h};\boldsymbol{0},\boldsymbol{I})$
- Objective:

$$\max_{\boldsymbol{W}} \sum_n \log \Pr(\boldsymbol{x}_n;\boldsymbol{W}_f,\boldsymbol{W}_g) - c\,\underbrace{KL(\Pr(\boldsymbol{h}|\boldsymbol{x}_n;\boldsymbol{W}_f)\,||N(\boldsymbol{h};\boldsymbol{0},\boldsymbol{I}))}_{\text{Kullback-Leibler divergence}}$$

Kullback-Leibler divergence
Distance measure for distributions

https://cs.uwaterloo.ca/~ppoupart/teaching/cs480-spring19/slides/cs480-lecture21.pdf

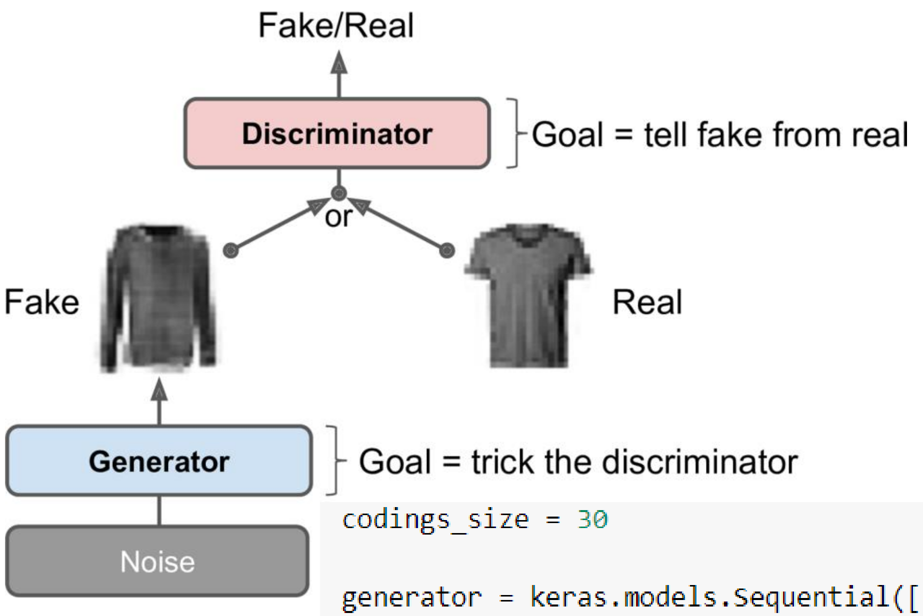
12 samples:

Equation 17-1. Kullback–Leibler divergence

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

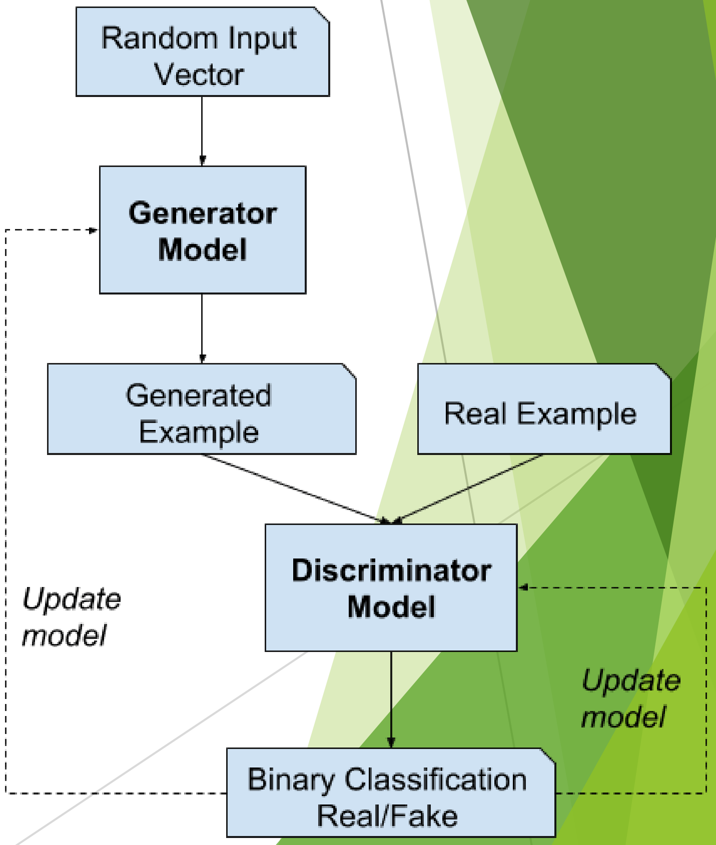
Generative Adversarial Networks (GANs)



I. Goodfellow et al. (2014): Generative Adversarial Nets

https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

```
codings_size = 30

generator = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[codings_size]),
    keras.layers.Dense(150, activation="selu"),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
discriminator = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(150, activation="selu"),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(1, activation="sigmoid")
])
gan = keras.models.Sequential([generator, discriminator])
```

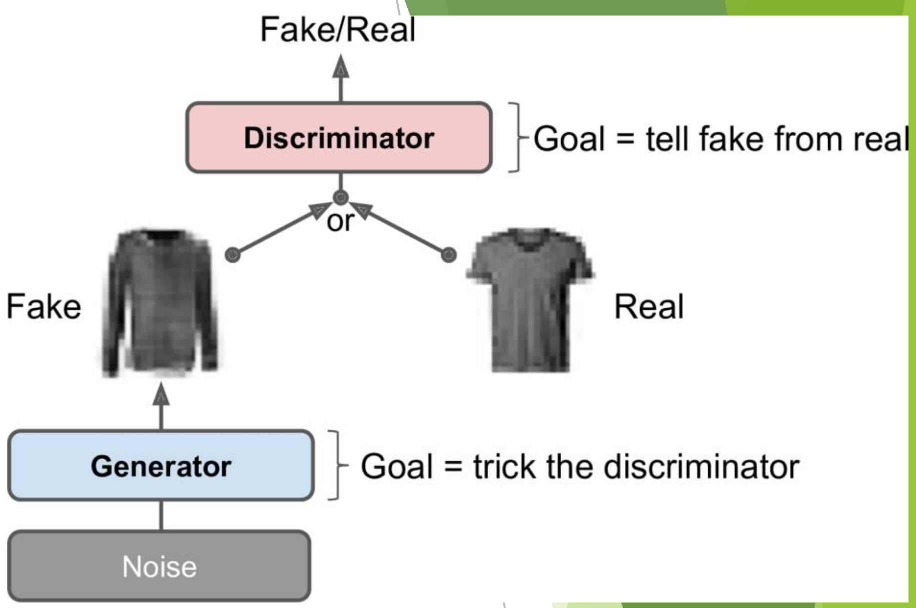


Training GANs

- Discriminator and Generator are Trained Separately

```
discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")
discriminator.trainable = False
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```

```
def train_gan(gan, dataset, batch_size, codings_size, n_epochs=50):
    generator, discriminator = gan.layers
    for epoch in range(n_epochs):
        print("Epoch {}/{}".format(epoch + 1, n_epochs))              # not shown in the book
        for X_batch in dataset:
            # phase 1 - training the discriminator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            generated_images = generator(noise)
            X_fake_and_real = tf.concat([generated_images, X_batch], axis=0)
            y1 = tf.constant([[0.]] * batch_size + [[1.]] * batch_size)
            discriminator.trainable = True
            discriminator.train_on_batch(X_fake_and_real, y1)
            # phase 2 - training the generator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            y2 = tf.constant([[1.]] * batch_size)
            discriminator.trainable = False
            gan.train_on_batch(noise, y2)
        plot_multiple_images(generated_images, 8)                     # not shown
        plt.show()                                                    # not shown
```




```
train_gan(gan, dataset, batch_size, codings_size, n_epochs=1)
```

Epoch 1/1

GAN Training Difficulties

```
train_gan(gan, dataset, batch_size, codings_size)
```

Mode Collapse!

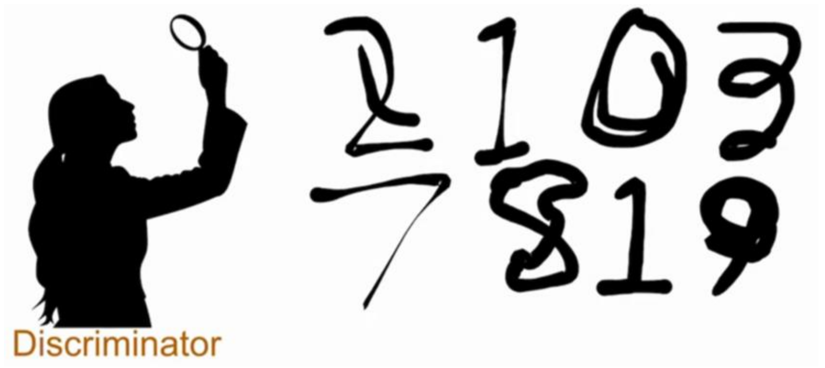
Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

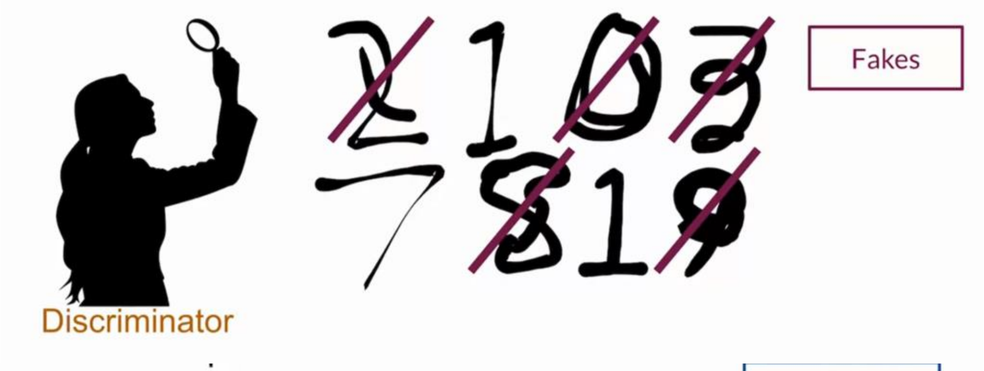
Mitigation Measures

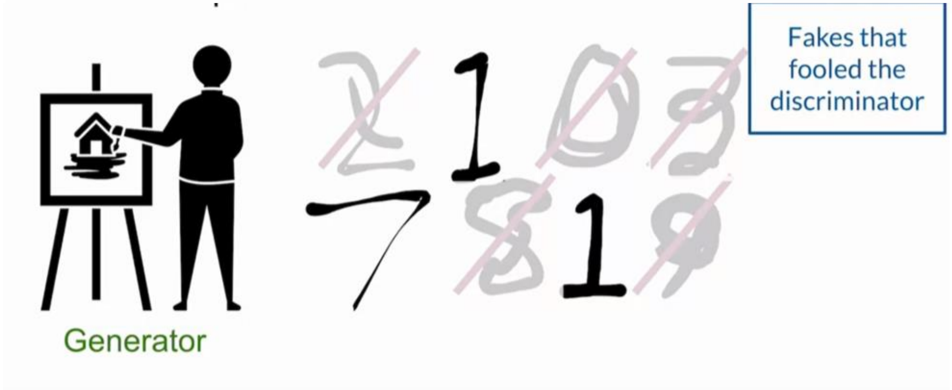
- Experience Replay
- Mini-Batch Discrimination
- Conditional Generation
- Change the Loss Function (e.g. Wasserstein Loss)

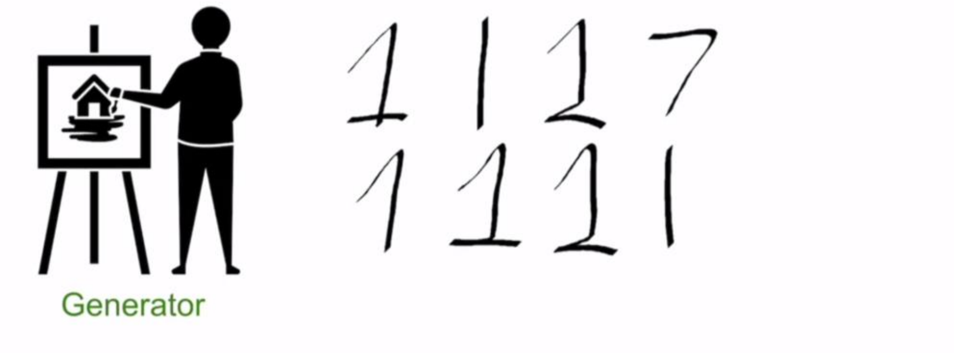
Mode Collapse: Example

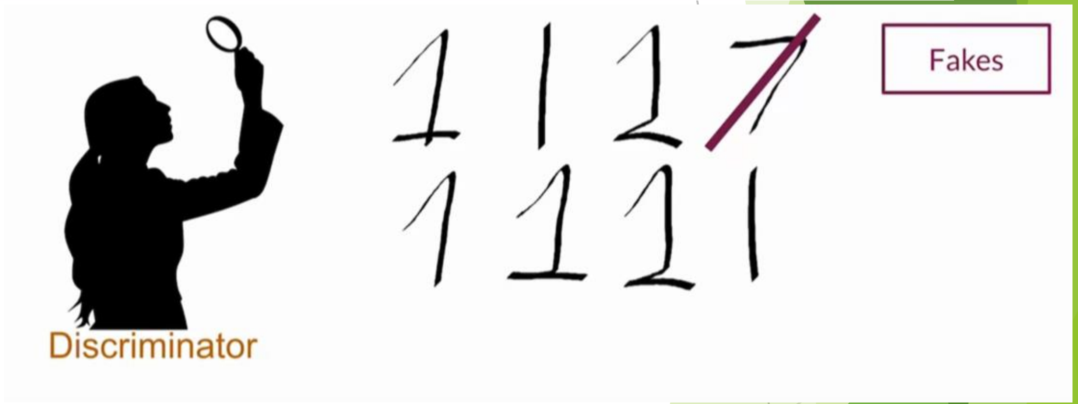
Source: https://www.coursera.org/learn/build-basic-generative-adversarial-networks-gans/home/week/3

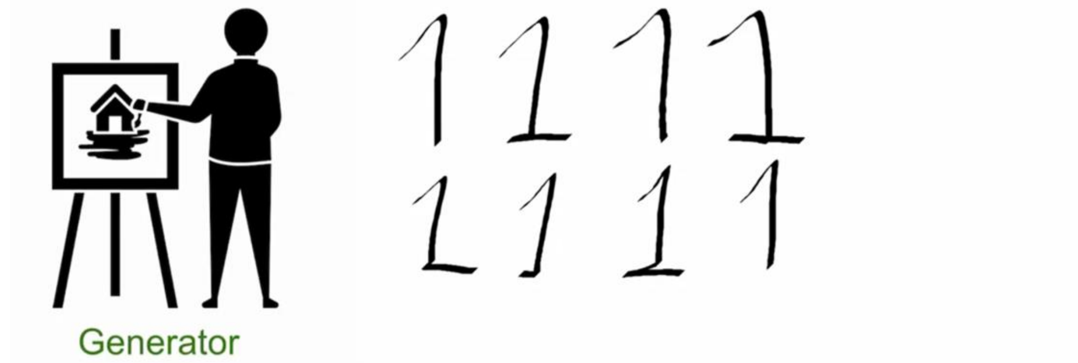
1


2


3


4


5


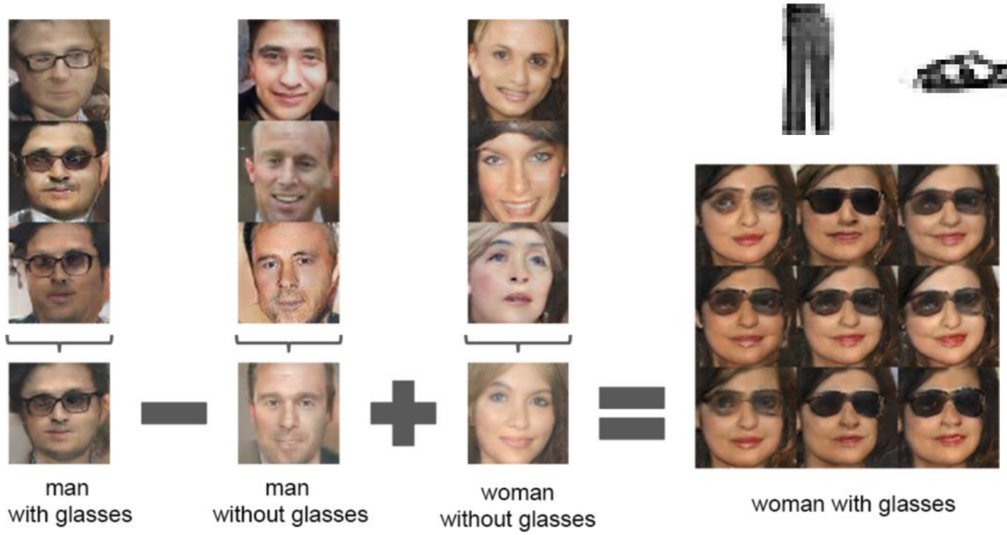
6


Deep Convolutional GANs (DCGANs)

A. Radford et al. (2016): Unsupervised Representation Learning with Deep Convolutional GANs

- Strided and Transposed Convolutions, no Pooling
- Batch Normalization
- No Dense Layers
- Activations
  - Generator: ReLU (output tanh)
  - Discriminator: Leaky ReLU

After 50 epochs:



A different approach:
M. Mirza, S. Osindero (2014) Conditional GANs

Progressive Growing

T. Karras et al. (2017): Progressive Growing of GANs for Improved Quality, Stability and Variation

Video Demonstration

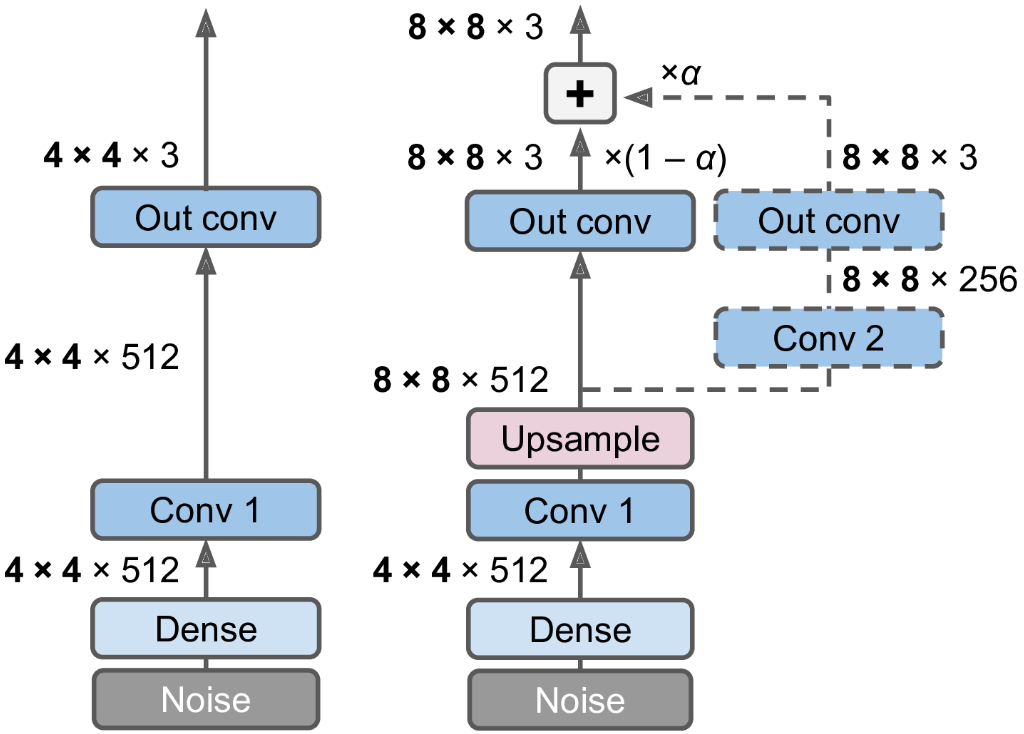


Figure 17-19. Progressively growing GAN: a GAN generator outputs 4 × 4 color images (left); we extend it to output 8 × 8 images (right)

- Similar Growing at Beginning of Discriminator
- Other Techniques introduced in this paper:
  - Mini-batch standard deviation layer
  - Learning Rate effectively scaled layerwise by $\sqrt{\frac{2}{n_{inputs}}}$
  - Pixelwise Normalization Layer

StyleGAN

[T. Karras et al. (2018): A Style-Based Generator Architecture for Generative Adversarial Networks]

[Video Demonstration]

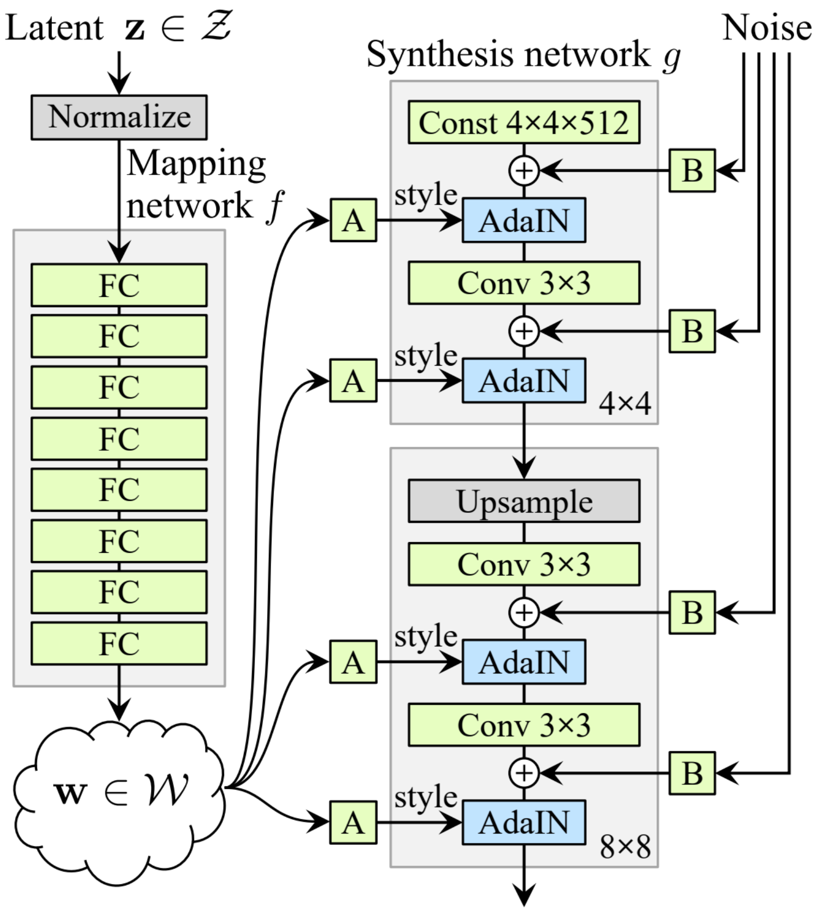
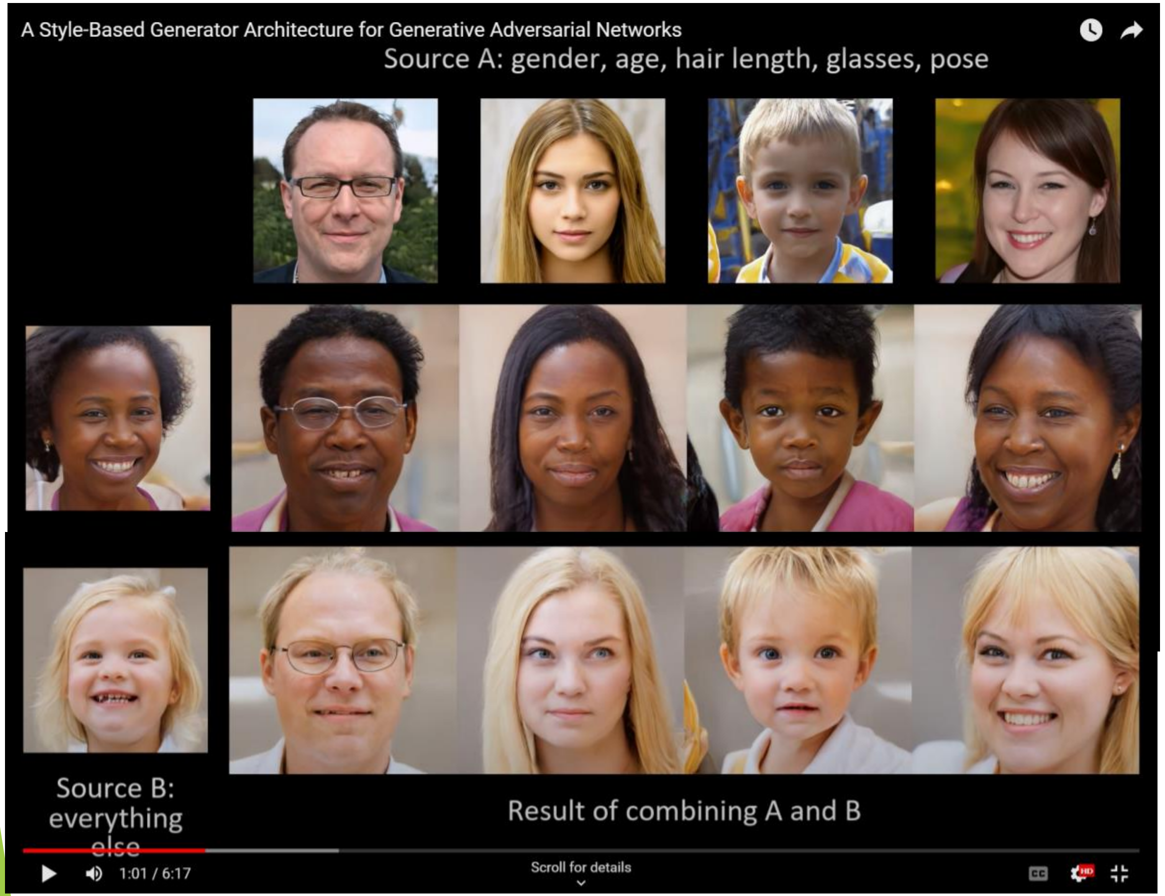


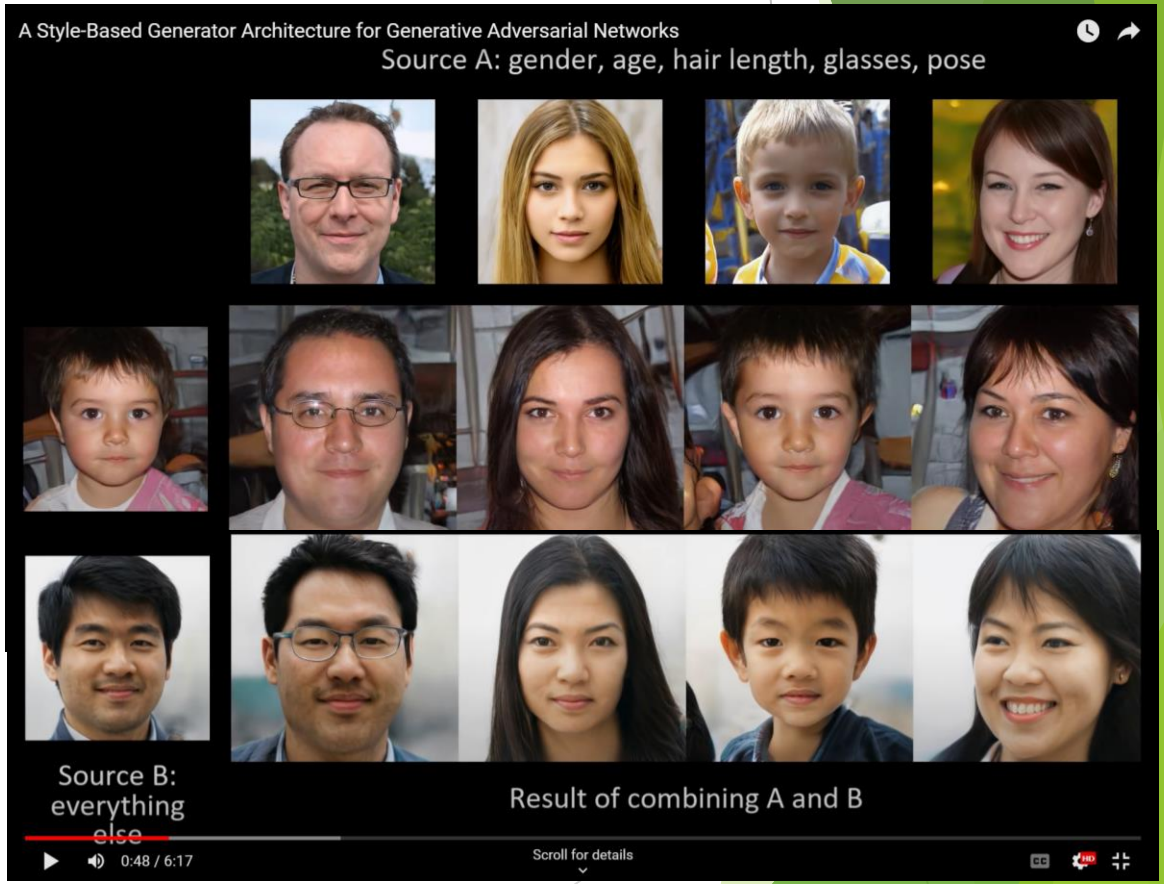
Figure 17-20. StyleGAN's generator architecture (part of figure 1 from the StyleGAN paper)

- Styles learned at varying scales
- Architecture
  - Mapping network
  - Synthesis network
  - Adaptive Instance Normalization

StyleGAN Results

- Video Demonstration





The End