Autoenkodery i sieci GAN

dr inż. Sebastian Ernst

Przedmiot: Uczenie Maszynowe

Wprowadzenie

Autoenkoder

- uczy się gęstych reprezentacji danych wejściowych (*latent representations*, *codings*), czyli kopiuje wejście na wyjście
- praca z nimi polega na „utrudnianiu" im zadania – ograniczenie rozmiaru reprezentacji, dodanie szumu, itd.
- reprezentacja mniejsza niż dane wejściowe
- to detektory cech, więc można ich użyć do nienadzorowanego uczenia wstępnego
- niektóre z nich to modele *generatywne* (ale np. generowane obrazy nie są realistyczne)

GAN (*generative adversarial network*)

- generują realistyczne obrazy, por. https://thispersondoesnotexist.com
- podwyższanie rozdzielczości, kolorowanie, usuwanie obiektów, powiększanie danych
- składa się z dwóch konkurujących ze sobą sieci:
  - *generatora* tworzącego dane podobne do uczących
  - *dyskryminatora* próbującego odróżnić dane rzeczywiste od sztucznych

Reprezentacja danych

Którą z sekwencji łatwiej zapamiętać?

- 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

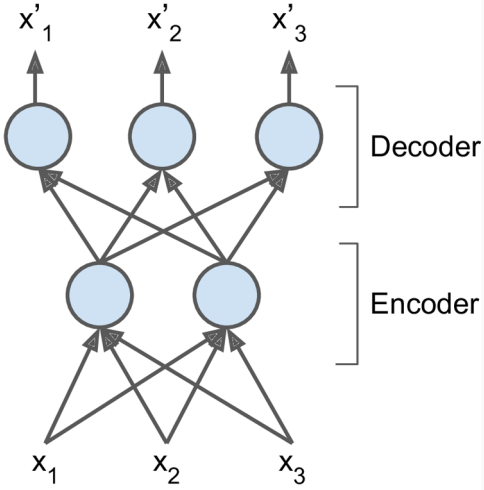
Autoenkodery

Szachy i prosty autoenkoder

Outputs
(≈ inputs)

Latent
representation

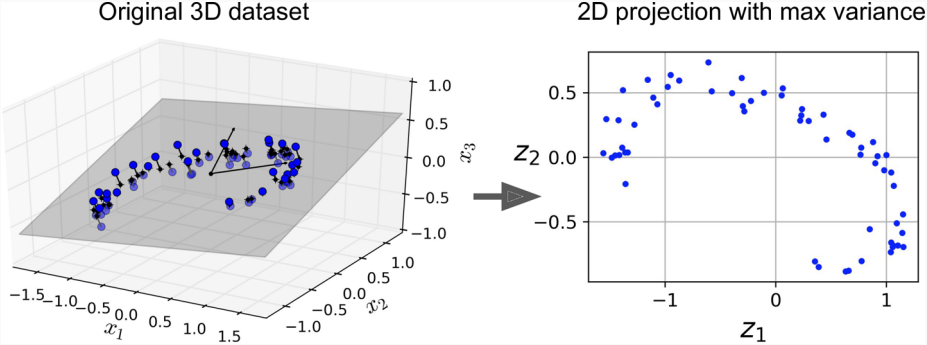
Inputs



PCA przy pomocy autoenkodera

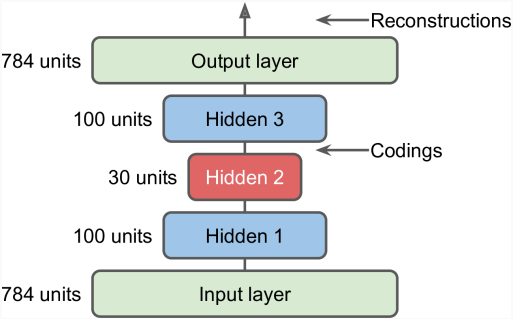
```
encoder = keras.models.Sequential(
    [keras.layers.Dense(2, input_shape=[3])])
decoder = keras.models.Sequential(
    [keras.layers.Dense(3, input_shape=[2])])
autoencoder = keras.models.Sequential([encoder, decoder])

# Używamy modeli jako warstw w modelu nadrzędnym!

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

Redukcja wymiarowości, wynik



Autoenkodery głębokie

- nazywane *stacked* lub *deep*
- struktura „kanapki”



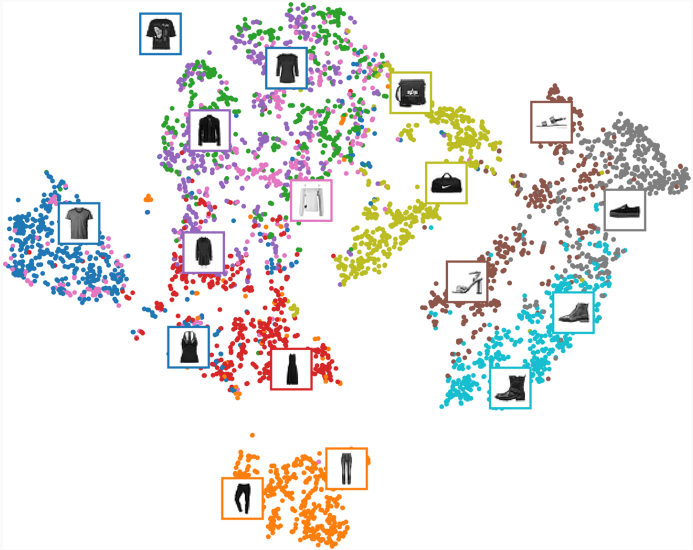
Autoenkoder dla Fashion MNIST

```
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])
])
```

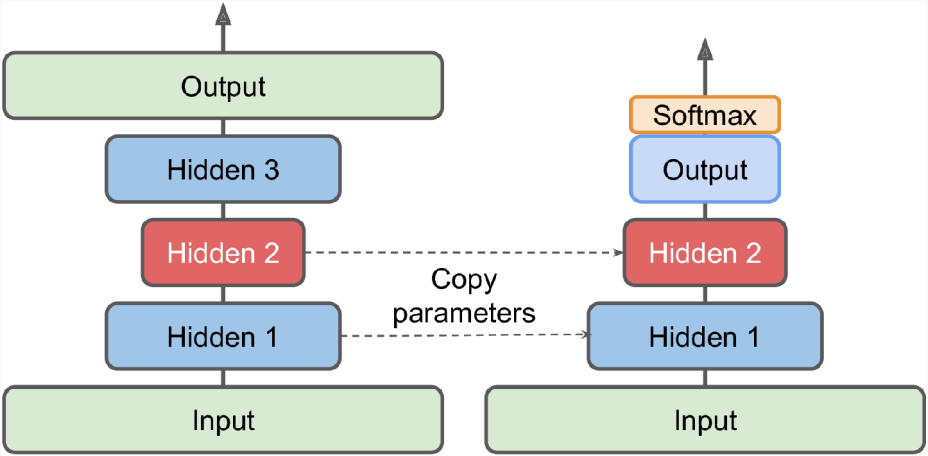
Weryfikacja jakości rekonstrukcji

```
def show_reconstructions(model, images=X_valid, n_images=5):
    reconstructions = model.predict(images[:n_images])
    fig = plt.figure(figsize=(n_images * 1.5, 3))
    for image_index in range(n_images):
        plt.subplot(2, n_images, 1 + image_index)
        plot_image(images[image_index])
        plt.subplot(2, n_images, 1 + n_images + image_index)
        plot_image(reconstructions[image_index])
```

Fashion MNIST, wizualizacja



Nienadzorowane uczenie wstępne



Autoenkodery konwolucyjne, enkoder

```
conv_encoder = keras.models.Sequential([
    keras.layers.Reshape([28, 28, 1], input_shape=[28, 28]),
    keras.layers.Conv2D(16, kernel_size=3, padding="SAME",
                        activation="selu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(32, kernel_size=3, padding="SAME",
                        activation="selu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(64, kernel_size=3, padding="SAME",
                        activation="selu"),
    keras.layers.MaxPool2D(pool_size=2)
])
```

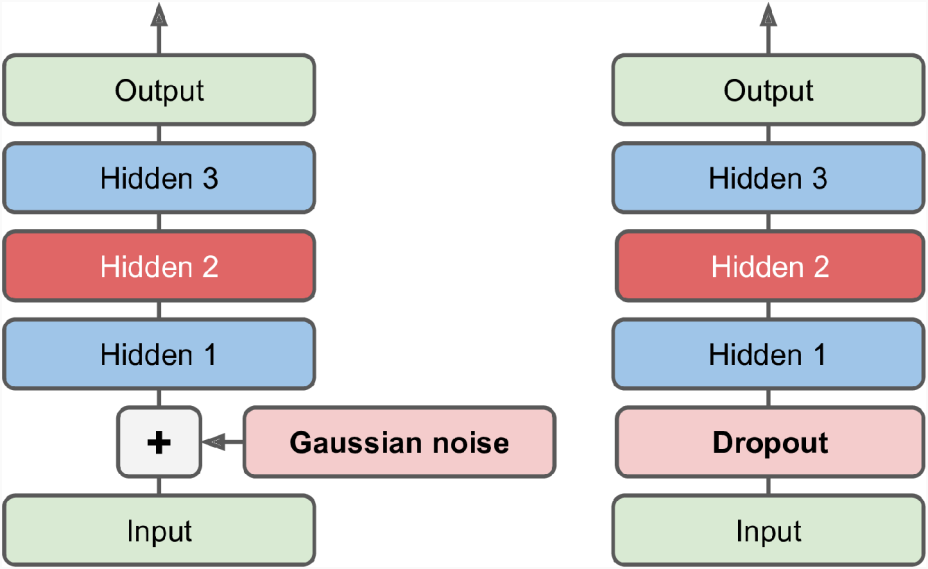
Autoenkodery konwolucyjne, dekoder

```
conv_decoder = keras.models.Sequential([
    keras.layers.Conv2DTranspose(32, kernel_size=3, strides=2,
                                 padding="VALID", activation="selu",
                                 input_shape=[3, 3, 64]),
    keras.layers.Conv2DTranspose(16, kernel_size=3, strides=2,
                                 padding="SAME", activation="selu"),
    keras.layers.Conv2DTranspose(1, kernel_size=3, strides=2,
                                 padding="SAME",
                                 activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
```

Autoenkodery niekompletne i przepełnione

- dotychczas nasze autoenkodery były niekompletne (*undercomplete*), tzn. ograniczaliśmy rozmiar warstwy kodującej aby uzyskać oszczędną reprezentację
- w niektórych zastosowaniach wybieramy autoenkodery przepełnione (*overcomplete*), w których rozmiar ten jest równy rozmiarowi wejścia, lub wręcz większy

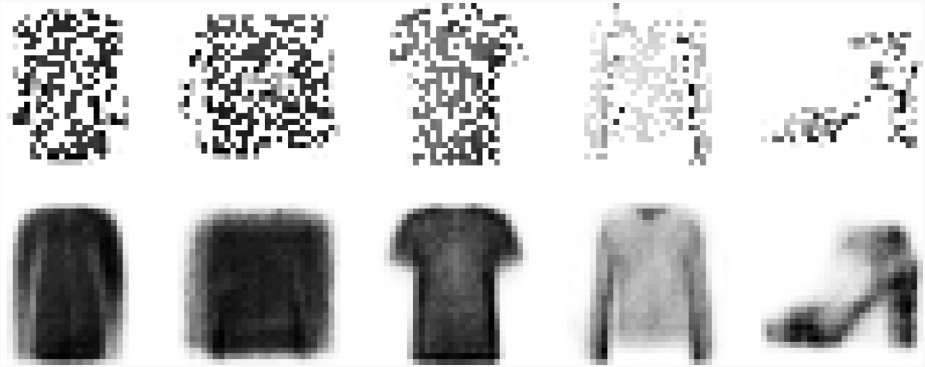
Odszumianie



Odszumianie, implementaccja

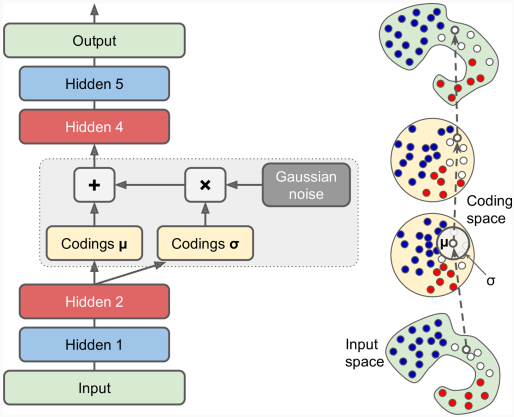
```
dropout_encoder = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(30, activation="selu")
])
dropout_decoder = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[30]),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
```

Odszumianie, wyniki



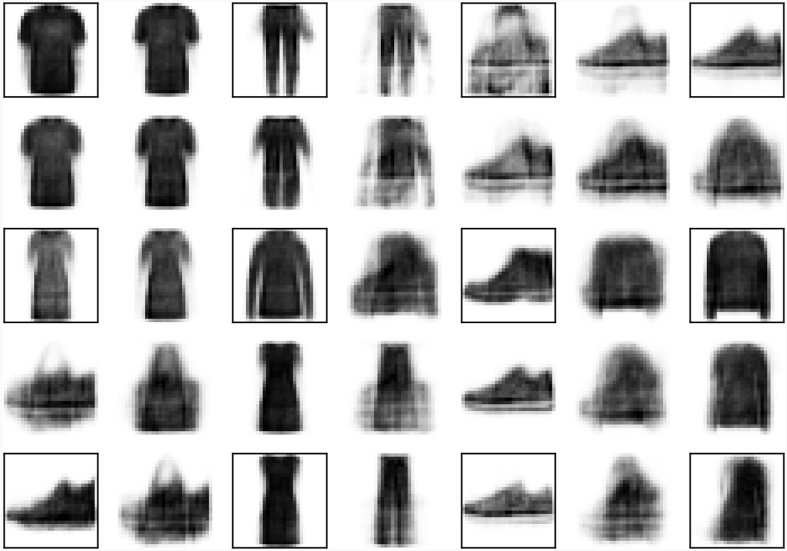
Autoenkodery wariacyjne

- są *probabilistyczne* – ich wyjścia nawet po uczeniu są częściowo losowe
- są *generatywne* – mogą generować nowe instancje podobne do tych ze zbioru uczącego



Generowanie obrazów Fashion MNIST

Interpolacja semantyczna

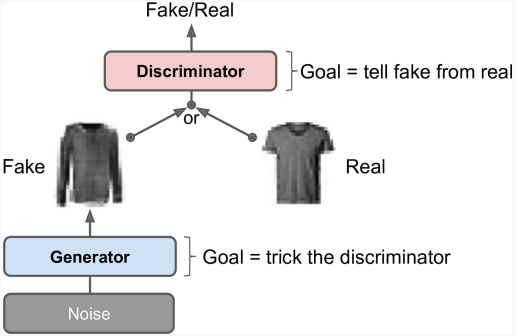


GAN

Generatywne sieci przeciwstawne (*Generative Adversarial Networks*)

Składają się z dwóch sieci:

1. *Generator* przyjmuje losowy rozkład i produkuje dane (np. obraz) – tak jak dekoder autoenkodera wariacyjnego
2. *Dyskryminator* ocenia, czy obraz jest prawdziwy czy sztuczny



Uczenie GAN

Faza 1: uczenie dyskryminatora

- wsad prawdziwych obrazów ze zbioru uczącego (etykieta 1) i sztucznych z generatora (etykieta 0)
- uczymy przez jedną iterację
- wagi dyskryminatora zablokowane

Faza 2: uczenie generatora

- generujemy wsad obrazów
- dyskryminator je ocenia, jego wagi są zablokowane
- etykiety ustawione na 1 – szukamy obrazów na które się „nabierze" dyskryminator

GAN dla Fashion MNIST

```
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")
])
```

Uczenie GAN, Fashion MNIST

```
def train_gan(gan, dataset, batch_size, codings_size, n_epochs=50):
    generator, discriminator = gan.layers
    for epoch in range(n_epochs):
        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)
```

GAN dla Fashion MNIST, wyniki po 1 iteracji

Głębokie konwolucyjne GAN

Zalecenia wg. Radford et al.:

- zastąpić warstwy zbierające rozkrokiem (dykryminator) i konwolucją z transpozycją (generator)
- BN w obu częściach, oprócz wyjścia generatora i wejścia dyskryminatora
- usunąć warstwy gęste
- ReLU w generatorze (wyjście — tanh)
- leaky ReLU w dyskryminatorze

Konwolucyjny GAN, generator

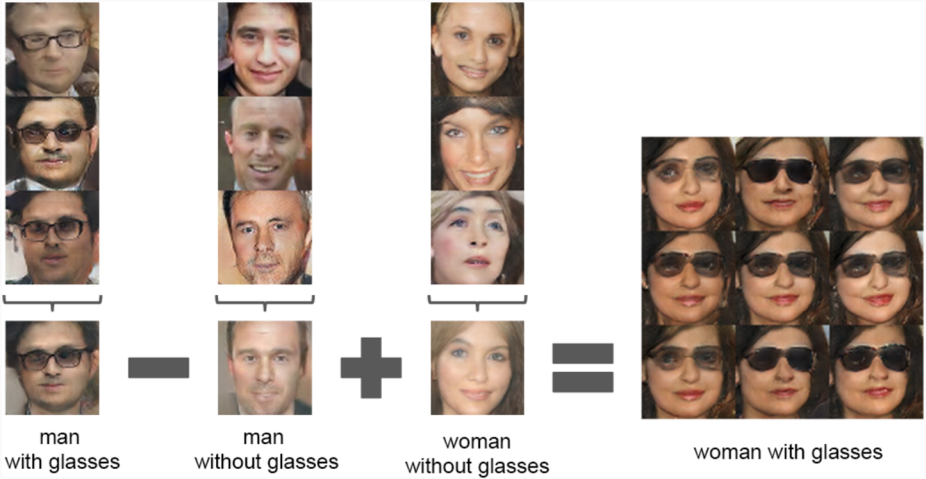
```
generator = keras.models.Sequential([
    keras.layers.Dense(7 * 7 * 128, input_shape=[codings_size]),
    keras.layers.Reshape([7, 7, 128]),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(64, kernel_size=5, strides=2,
                                 padding="SAME", activation="selu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(1, kernel_size=5, strides=2,
                                 padding="SAME", activation="tanh"),
])
```

Konwolucyjny GAN, dyskryminator

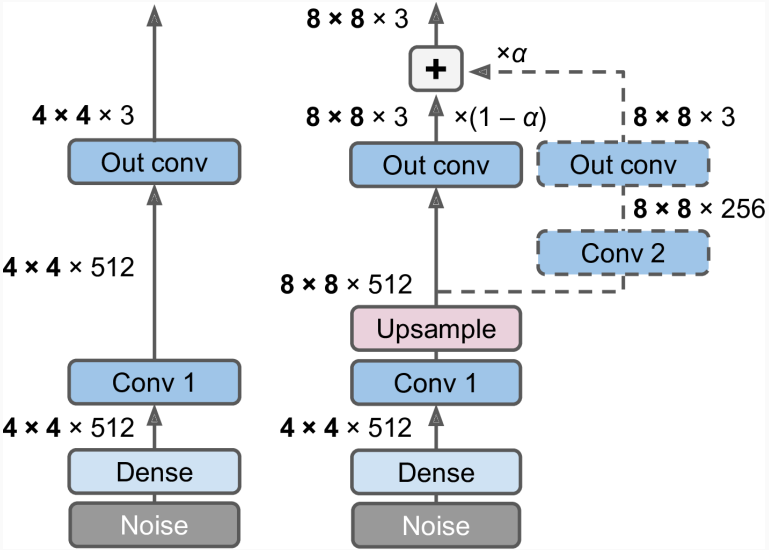
```
discriminator = keras.models.Sequential([
    keras.layers.Conv2D(64, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2),
                        input_shape=[28, 28, 1]),
    keras.layers.Dropout(0.4),
    keras.layers.Conv2D(128, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2)),
    keras.layers.Dropout(0.4),
    keras.layers.Flatten(),
    keras.layers.Dense(1, activation="sigmoid")
])
```

Konwolucyjny GAN, wyniki

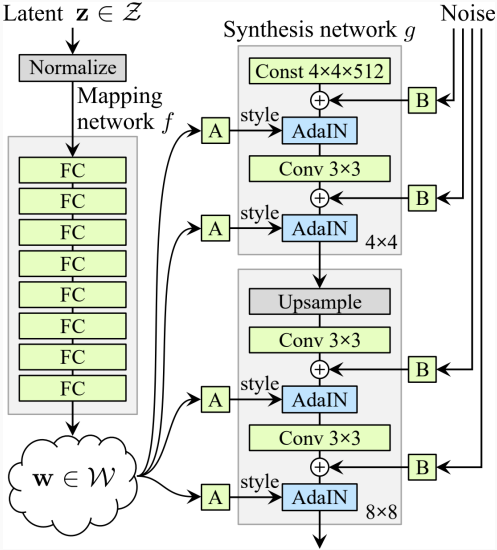
Konwolucyjny GAN, łączenie reprezentacji



GAN, stopniowa rozbudowa



StyleGAN


Latent $\mathbf{z} \in \mathcal{Z}$
Normalize
Mapping network f
FC
FC
FC
FC
FC
FC
FC
FC
$\mathbf{w} \in \mathcal{W}$
Synthesis network g
Noise
Const 4×4×512
B
style
A
AdaIN
Conv 3×3
B
style
A
AdaIN
4×4
Upsample
Conv 3×3
B
style
A
AdaIN
Conv 3×3
B
style
A
AdaIN
8×8