

# Autoenkodery

Autoenkodery to sieci neuronowe próbujące odtworzyć wejście sieci na jej wyjściu. Funkcją kosztu jest różnica między wejściem a wyjściem, stąd dane do uczenia autoenkodera nie muszą być opisane. Najczęściej, autoenkoder posiada w środku tzw bottleneck (warstwę, o mniejszym wymiarze niż wejście i wyjście). Uczenie takiego enkodera rekonstrukcji wejścia powoduje znalezienie ukrytej reprezentacji danych wejściowych o mniejszym wymiarze.

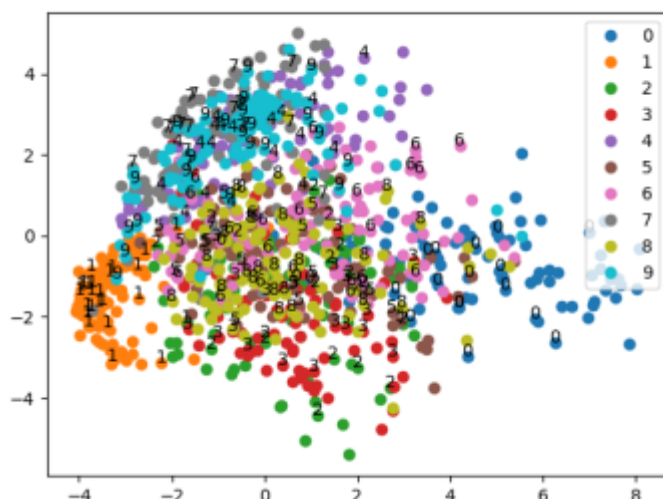
W ramach ćwiczenia, Państwa zadaniem będzie próba redukcji wymiarowości przykładów zbioru MNIST do dwóch wartości rzeczywistych z wykorzystaniem autoenkoderów.

Na początku proszę zaimportować odpowiednie pakiety:

```
import numpy as np
from keras.datasets import mnist
import keras
from keras import layers
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from matplotlib import pyplot as plt
```

## Redukcja wymiarowości z wykorzystaniem PCA

W pierwszej kolejności użyjemy do tego celu PCA, aby mieć jakiś punkt odniesienia. Zastosowanie PCA na zbiorze MNIST i wybranie dwóch pierwszych składowych, następnie wyrysowanie ich na płaszczyźnie może dać poniższych efekt.



Powyższy wykres przedstawia dwie najbardziej znaczące (mające najwięcej informacji) kombinacje liniowe wszystkich pikseli obrazu. Oznacza to na przykład, że jedynki mają swoje charakterystyczne piksele gdzie indziej, oraz niektóre zera, w porównaniu z resztą. Siódemki natomiast są nieco wymieszane z dziewiątkami (dzielą więcej tych samych pikseli).

Do wyrysowania powyższego przykładu proszę użyć mniej więcej 10% danych ze zbioru testowego. Przykładowy kod realizujący to zadanie:

```
X_train_flatten = np.reshape(X_train, (X_train.shape[0],
X_train.shape[1]*X_train.shape[2]))

pca = PCA()
pca.fit(X_train_flatten)

X_test_flatten = np.reshape(X_test, (X_test.shape[0], X_test.shape[1] *
X_test.shape[2]))
Z_test_flatten = pca.transform(X_test_flatten)
Z_test_subset, _, y_test_subset, _ = train_test_split(Z_test_flatten,
y_test, train_size=0.1, random_state=12345)

plt.figure()
for i in range(10):
    X_tmp = Z_test_subset[y_test_subset == i]
    plt.plot(X_tmp[:, 0], X_tmp[:, 1], 'o')
    for j in range(0, len(X_tmp), 5):
        plt.annotate(i, (X_tmp[j, 0], X_tmp[j, 1]), ha='center')
plt.legend(range(10))
plt.show()
```

## Redukcja wymiarowości z wykorzystaniem Autoenkodera

Aby przygotować dane do uczenia, proszę zastosować wskazówki z poprzednich zajęć.

Następnie utworzyć model autoenkodera:

```
input_shape = (28, 28, 1)

inputs = layers.Input(shape=input_shape)

flatten = layers.Flatten()(inputs)
dense1 = layers.Dense(50, activation='tanh')(flatten)
dense2 = layers.Dense(50, activation='tanh')(dense1)
latent = layers.Dense(2, activation='linear')(dense2)
dense3 = layers.Dense(50, activation='tanh')(latent)
dense4 = layers.Dense(50, activation='tanh')(dense3)
dense5 = layers.Dense(28 * 28, activation='relu')(dense4)
outputs = layers.Reshape(target_shape=(28, 28, 1))(dense5)

model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(loss=keras.metrics.mse, optimizer="adam")
```

Proszę zwrócić uwagę na stopniową redukcję wymiarowości aż do osiągnięcia warstwy środkowej

(latent). Kolejne warstwy to próba rekonstrukcji wejścia z formy skompresowanej. Tutaj, będziemy próbowali zakodować wejście z pomocą dwóch liczb rzeczywistych (funkcja aktywacji linear). Funkcje aktywacji 'tanh' wprowadzają nieliniowość do modelu. Natomiast funkcja kosztu do średni kwadrat różnic między pikselami na wejściu i na wyjściu.

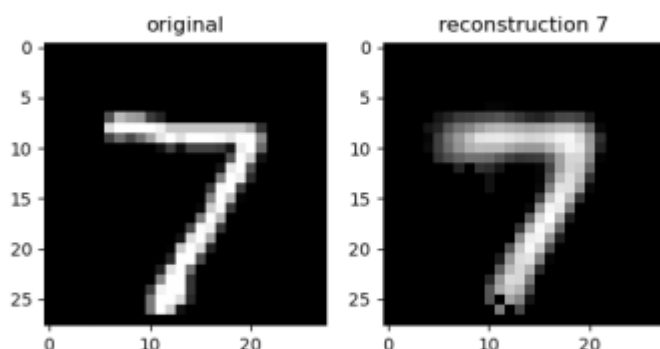
Tak przygotowany model można poddać uczeniu:

```
batch_size = 128
epochs = 50
model.fit(X_train, X_train, batch_size=batch_size, epochs=epochs,
          validation_split=0.2)
```

Nauczony model można poddać predykcji (rekonstrukcji na podstawie oryginalnego obrazu):

```
X_gen = model.predict(X_test)

example_id = 0
plt.figure()
fig, axs = plt.subplots(1, 2)
axs[0].imshow(X_test[example_id, :, :, 0], cmap='gray')
axs[0].set_title('original')
axs[1].imshow(X_gen[example_id, :, :, 0], cmap='gray')
axs[1].set_title(f'reconstruction {y_test[example_id]}')
plt.show()
```



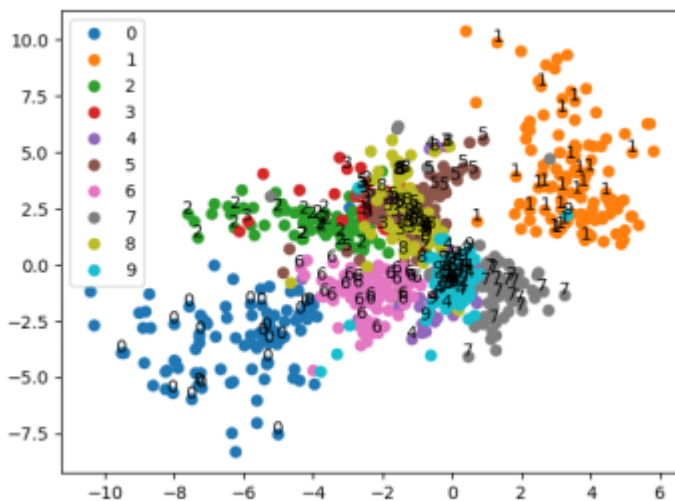
Proszę zwrócić uwagę, że wynik rekonstrukcji został odtworzony z dwóch wartości warstwy latent. Aby zwizualizować gdzie znajdują się poszczególne przykłady w ich reprezentacji ukrytej, możemy “przeciąć” naszą sieć w pół i wygenerować odpowiedzi warstwy latent:

```
compression_model = keras.Model(inputs=inputs, outputs=latent)

X_test_subset, _, y_test_subset, _ = train_test_split(X_test, y_test,
                                                         train_size=0.1, random_state=12345)
X_compressed = compression_model.predict(X_test_subset)
```

```
plt.figure()
for i in range(10):
    X_tmp = X_compressed[y_test_subset == i]
    plt.plot(X_tmp[:, 0], X_tmp[:, 1], 'o')
    for j in range(0, len(X_tmp), 5):
        plt.annotate(i, (X_tmp[j, 0], X_tmp[j, 1]), ha='center')
plt.legend(range(10))
plt.show()
```

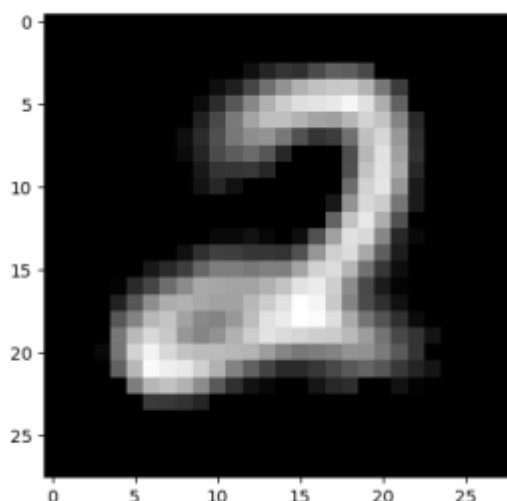
Przykładowy wynik może być następujący.



Widać wyraźnie, że autoenkoder lepiej poradził sobie z kompresja danych wejściowych do dwóch wartości (widoczne są wyraźne skupiska danych wejściowych, że względu na klasę).

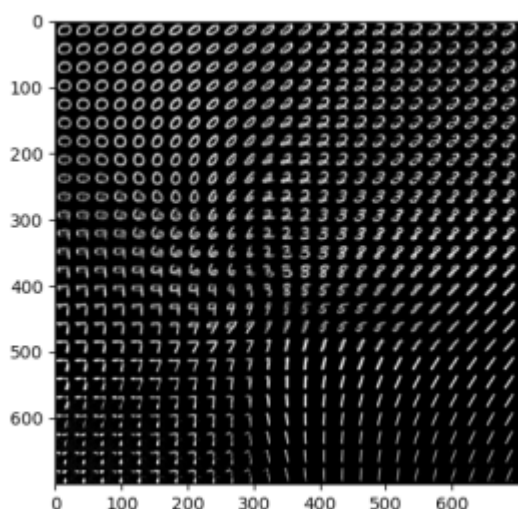
Oprócz weryfikacji rozkładu enkodera, można wydzielić drugą część sieci (dekoder) i wygenerować kilka przykładów podając na wejściu przykładowe dwie wartości:

```
decoder = keras.Model(inputs=latent, outputs=outputs)
input_data = np.array([[ -12.0, -2.0]])
random_img = decoder.predict(input_data)
plt.imshow(random_img[0, :, :, 0], cmap='gray')
plt.show()
```




Bądź zbadać całą przestrzeń z odpowiednim próbkowaniem:

```
xs = np.linspace(-10, 6, 25)
ys = np.linspace(-7, 10, 25)
all_img = np.zeros([len(xs)*28, len(ys)*28], dtype=np.float32)
for i, x in enumerate(xs):
    for j, y in enumerate(ys):
        input_data = np.array([[x, y]])
        random_img = decoder.predict(input_data)
        all_img[i * 28:(i + 1) * 28, j * 28:(j + 1) * 28] = random_img[0, :,
        :, 0]
all_img[all_img > 1.0] = 1.0
plt.imshow(all_img[:, :], cmap='gray')
plt.show()
```



From:  
<https://home.agh.edu.pl/~mdig/dokuwiki/> - **MVG Group**

Permanent link:  
[https://home.agh.edu.pl/~mdig/dokuwiki/doku.php?id=teaching:data\\_science:ml\\_pl:topics:nn\\_autoencodery](https://home.agh.edu.pl/~mdig/dokuwiki/doku.php?id=teaching:data_science:ml_pl:topics:nn_autoencodery) 

Last update: **2023/06/01 17:40**