Laboratorium: Autoenkodery

June 10, 2025

1 Zakres ćwiczeń

Dzisiejsze laboratorium poświęcone jest autoenkoderom (in. autokoderom) – specyficznej klasie sieci, których charakterystyczną właściwością jest zdolność do znajdowania wąskich reprezentacji danych.

2 Ćwiczenia

Pobierz i przygotuj zbiór danych MNIST:

```
dataset = tf.keras.datasets.mnist.load_data()
(X_train_full, y_train_full), (X_test, y_test) = dataset
X_train_full = X_train_full.astype(np.float32) / 255
X_test = X_test.astype(np.float32) / 255
X_train, X_valid = X_train_full[:-5000], X_train_full[-5000:]
y_train, y_valid = y_train_full[:-5000], y_train_full[-5000:]
```

2.1 Autoenkoder z warstwami gęstymi

10 p.

Przygotuj głęboki autoenkonder zdolny do reprezentowania obrazów cyfr. Poeksperymentuj z liczbą warstw oraz liczbą neuronów na każdej z nich, szczególnie na ostatniej warstwie enkodera – ona determinuje rozmiar reprezentacji.

```
encoder = tf.keras.Sequential([
    ...
])
decoder = tf.keras.Sequential([
    ...
])
ae = tf.keras.Sequential([encoder, decoder])

ae.compile(..., metrics=['mae'])
history = ae.fit(...)
```

Uwaga: Niezależnie od stosowanej funkcji straty, każdy z modeli powinien gromadzić wartości metryki MAE – jest ona używana w procesie sprawdzania.

Przeprowadź trening sieci przez 5 epok. Sprawdź wartość przyjętej metryki jako baseline:

```
ae.evaluate(X_test, X_test, return_dict=True)
```

Działanie tego modelu, jak również każdego z kolejnych, możesz zaobserwować korzystając z funkcji podobnej do tej z podręcznika:

```
def plot_reconstructions(model, images=X_test, n_images=5):
    reconstructions = np.clip(model.predict(images[:n_images]), 0, 1)
    fig = plt.figure(figsize=(n_images * 1.5, 3))
    for image_index in range(n_images):
        plt.subplot(2, n_images, 1 + image_index)
        plt.imshow(images[image_index], cmap="binary")
        plt.axis("off")
        plt.subplot(2, n_images, 1 + n_images + image_index)
        plt.imshow(reconstructions[image_index], cmap="binary")
        plt.axis("off")

plot_reconstructions(ae)
```

Zapisz model w pliku `ae_stacked.keras`.

```
ae.save('ae_stacked.keras')
```

2.2 Autoenkoder konwolucyjny

10 p.

Ponieważ mamy do czynienia z obrazami, skuteczny może się okazać model oparty nie o warstwy gęste, a o warstwy konwolucyjne.

Enkoder powinien mieć strukturę typową dla sieci konwolucyjnych – warstwy konwolucyjne „przeplatane" z warstwami zbierającymi. Rozmiar map cech („obrazów") powinien się stopniowo zmniejszać (warstwy zbierające), przy jednoczesnym zwiększaniu ich liczby (liczba filtrów w warstwach konwolucyjnych). Wyjściem powinno być n map cech o rozmiarze 1×1, odpowiadającym poszczególnym elementom reprezentacji (gdzie n jest jej wielkością).

Struktura dekodera powinna być podobna, przy czym oczywiście tu kolejność jest odwrotna (rozmiar map cech rośnie, a ich liczba maleje) – „powiększanie" map cech uzyskamy przy pomocy warstw dekonwolucyjnych realizowanych przy pomocy klasy `Conv2DTranspose`.

Zbuduj model i przeprowadź jego trening, podobnie jak poprzednio, przez 5 epok.

Sprawdź wartość funkcji straty. Powinna być znacznie lepsza niż poprzednio:

```
conv_ae.evaluate(X_test, X_test)
```

Obejrzyj wyniki działania:

```
plot_reconstructions(conv_ae)
plt.show()
```

Zapisz model w pliku `ae_conv.keras`.

```
conv_ae.save('ae_conv.keras')
```

2.3 Wizualizacja wyników

Gęste reprezentacje produkowane przez autoenkodery często wykorzystywane są do wizualizacji danych.

```
from sklearn.manifold import TSNE

X_valid_compressed = conv_encoder.predict(X_valid)
tsne = TSNE(init="pca", learning_rate="auto", random_state=42)
X_valid_2D = tsne.fit_transform(X_valid_compressed)

plt.figure(figsize=(10, 5))
plt.scatter(X_valid_2D[:, 0], X_valid_2D[:, 1], c=y_valid, s=10, cmap="tab10")
plt.show()
```

Na wykres można nanieść również próbki z poszczególnych klastrów. Posłużmy się kodem z podręcznika:

```
import matplotlib as mpl

plt.figure(figsize=(10, 5))
cmap = plt.cm.tab10
Z = X_valid_2D
Z = (Z - Z.min()) / (Z.max() - Z.min())  # normalize to the 0-1 range
plt.scatter(Z[:, 0], Z[:, 1], c=y_valid, s=10, cmap=cmap)
image_positions = np.array([[1., 1.]])
for index, position in enumerate(Z):
    dist = ((position - image_positions) ** 2).sum(axis=1)
    if dist.min() > 0.02: # if far enough from other images
        image_positions = np.r_[image_positions, [position]]
        imagebox = mpl.offsetbox.AnnotationBbox(
            mpl.offsetbox.OffsetImage(X_valid[index], cmap="binary"),
            position, bboxprops={"edgecolor": cmap(y_valid[index]), "lw": 2})
        plt.gca().add_artist(imagebox)

plt.axis("off")
plt.show()
```

2.4 Odszumianie

10 p.

Autoenkodery często używane są do usuwania szumu z obrazu. Szum do danych możemy dodać np. przy pomocy warstwy `GaussianNoise`, lub po prostu dodając warstwę `Dropout`:

```
dropout_encoder = tf.keras.Sequential([
    tf.keras.layers.Flatten(),
```

```
    tf.keras.layers.Dropout(0.5),
    # ...
])
dropout_decoder = tf.keras.Sequential([
    # ...
])
dropout_ae = tf.keras.Sequential([dropout_encoder, dropout_decoder])

dropout_ae.compile(...)
history = dropout_ae.fit(...)
```

Poeksperymentuj z działaniem sieci (parametr `rate` warstwy `Dropout` lub `stddev` warstwy `GaussianNoise`). Zobacz jak wygląda jeden i drugi rodzaj zaburzenia obrazu.

Zbuduj i wytrenuj model przez 5 epok.

Zwizualizuj wyniki działania. Ponieważ szum dodawany jest w samej sieci, musimy go też dodać, pogladowo, w podczas wizualizacji.

```
dropout = tf.keras.layers.Dropout(0.3)
plot_reconstructions(dropout_ae, dropout(X_valid, training=True))
```

Zapisz model w pliku `ae_denoise.keras`.

```
dropout_ae.save('ae_denoise.keras')
```

2.5 Dla zainteresowanych: CIFAR-10

Zadanie dodatkowe, nieoceniane

Przetwarzanie zbioru MNIST jest bardzo łatwym zadaniem nawet dla prostych modeli.

Aby przekonać się o tym, jak charakter danych wpływa na trudność zadania, spróbuj powtórzyć niektóre z powyższych ćwiczeń dla zbioru zawierającego obrazy wprawdzie niewiele większe, ale kolorowe (3 kanały – RGB) i o zupełnie innym charakterze (są to pomniejszone zdjęcia).

Zbiór danych CIFAR-10 zawiera obrazy 32×32 należące do 10 różnych klas – po 6000 ($5000 + 1000$) na klasę:

0. samolot
1. samochód
2. ptak
3. kot
4. jeleń
5. pies
6. żaba
7. koń
8. statek
9. ciężarówka

Pobierz zbiór danych:

```
cifar10 = tf.keras.datasets.cifar10.load_data()
(X_train_full, y_train), (_, _) = cifar10
X_train_full = X_train_full.astype(np.float32) / 255
X_train, X_valid = X_train_full[:-5000], X_train_full[-5000:]
```

Poeksperymentuj ze strukturą sieci oraz rozmiarem reprezentacji. Czy udaje się uzyskać rozsądne wyniki?

Pamiętaj, aby dostosować modele do nowego rozmiaru danych – teraz zamiast $[28, 28]$ mamy $[32, 32, 3]$.

3 Wyślij rozwiązanie

Skrypt realizujący powyższe punkty zapisz w pliku `lab13/lab13.py` w swoim repozytorium.