Laboratorium: Autoenkodery

June 20, 2023

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

2.1 Pobieranie danych

Pobierz zbiór danych MNIST:

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

2.2 Autoenkoder

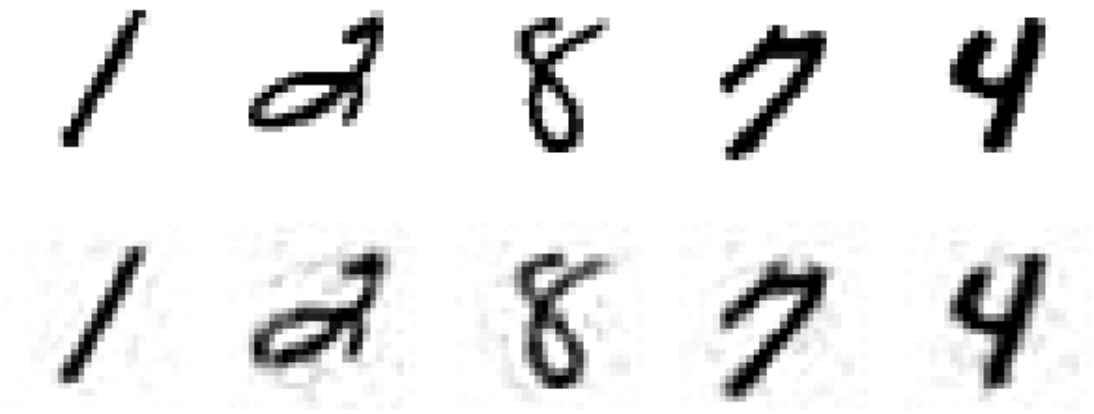
Przygotuj głęboki autoenkonder zdolny do reprezentowania obrazów cyfr w przyzwoitej jakości. Poeksperymentuj z liczbą warstw oraz liczbą neuronów na każdej z nich, szczególnioe na ostatniej warstwie enkodera – ona determinuje rozmiar reprezentacji.

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

```
def plot_reconstructions(model, images=X_valid, 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(stacked_ae)
```

```
1/1 [==============================] - 0s 10ms/step
```



2.3 Autoenkoder konwolucyjny

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`.

2.4 Odszumianie

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`.

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

Skonstruuj model usuwający szum z obrazu. Dane zaszumione możesz przygotować wcześniej w osobnej zmiennej lub dodać odpowiedni element do samego modelu. W drugim przypadku, pamiętaj o zastosowaniu analogicznego zabiegu przy wyświetlaniu wyników:

```
dropout = tf.keras.layers.Dropout(0.1)
plot_reconstructions(dropout_ae, dropout(X_valid[100:], training=True))
```

2.5 Autoenkoder wariacyjny

Skonstruuj autoenkoder wariacyjny dla zbioru MNIST. Ze względu na rozgałęzienie, konieczne będzie skorzystanie z API funkcyjnego Keras – API sekwencyjne w tym wypadku nie wystarczy.

Przyucz model zbiorem danych i spróbuj wykorzystać go do wygenerowania nowych, nieistniejących cyfr.

2.6 Podnosimy poprzeczkę: CIFAR-10

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 $[32, 32, 3]$.