Sieci neuronowe

Krzysztof Halawa

Obliczenia neuronowe - autoenkodery, GAN, Keras

2024

Treść prezentacji

Autoenkodery

Generatywne sieci przeciwstawne GAN (ang. Generative Adversarial Networks)

Polecana literatura do samodzielnej nauki

[AGeron] Aurélien Géron, Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Wyd. II, Helion, 2020

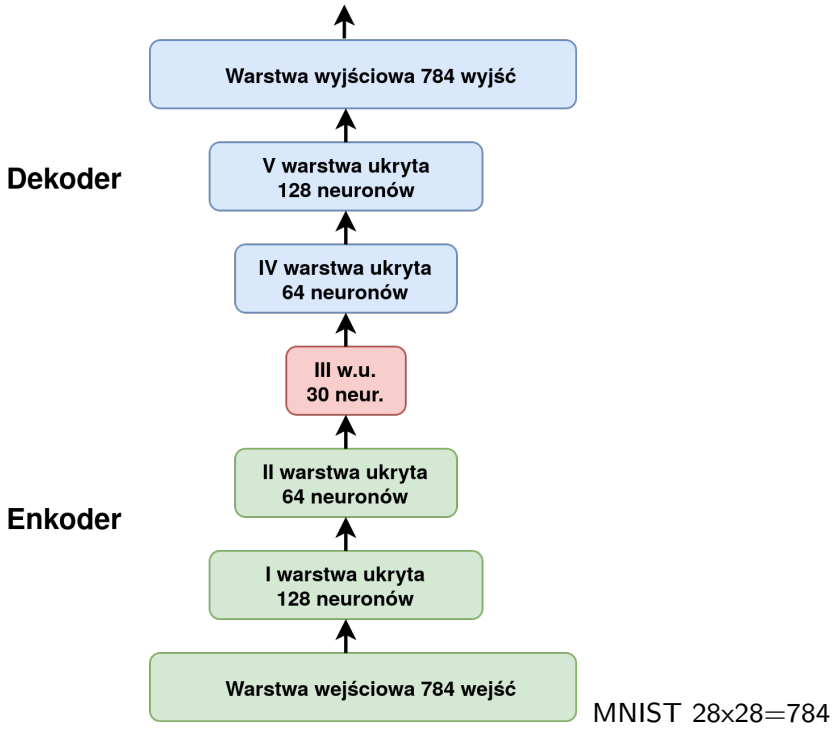
[FChollet] Francois Chollet, Deep Learning. Praca z językiem Python i biblioteką Keras, Helion, 2019

[DFoster] David Foster, Deep learning i modelowanie generatywne. Jak nauczyć komputer malowania, pisania, komponowania i grania, Helion, 2021

Autoenkodery

Autoenkoder stara się zrekonstruować dane wejściowe. Dane wyjściowe są nazywane rekonstrukcjami.
Autoenkoder składa się z dwóch części:

- enkoder (ang. encoder) zwany też koderem lub siecią rozpoznania (warstwy zielone+czerwona).
- dekoder nazywany również siecią generatywną (ang. decoder, generative network).



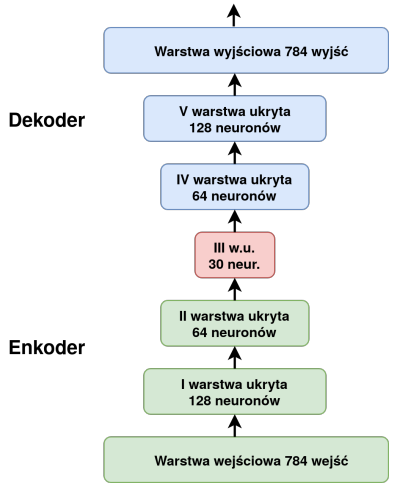
MNIST 28x28=784

kod - reprezentacja ukryta

Enkoder przekształca dane wejściowe do reprezentacji ukrytej.

Autoenkodery

Proste autenkodery mogą mieć tylko dwie warstwy neuronów (gęste) np. 784 wejścia, 30 neuronów w warstwie ukrytej, 784 neurony w warstwie wyjściowej.



MNIST 28x28=784

Efektywne reprezentacje danych - liczba wyjść enkodera mniejsza od liczby wejść

Automatyczna znajdowanie istotnych wzorców - ekstrakcja istotnych cech.
Redukcja wymiarowości,
Kompresja danych - można użyć np. do obrazów, sygnałów audio
Dawniej do nienadzorowanego uczenia wstępnego

Dla obrazów funkcją straty jest kategorialna krzyżowa entropia (w warstwie wyjściowej można keras.activation.sigmoid ale nie nalezy zapomnieć o przeskalowaniu wejść i wyjść, jeżeli kodowanie barwy pikseli poza przedziałem [0,1])

Prosty autencoder do MNIST - Keras

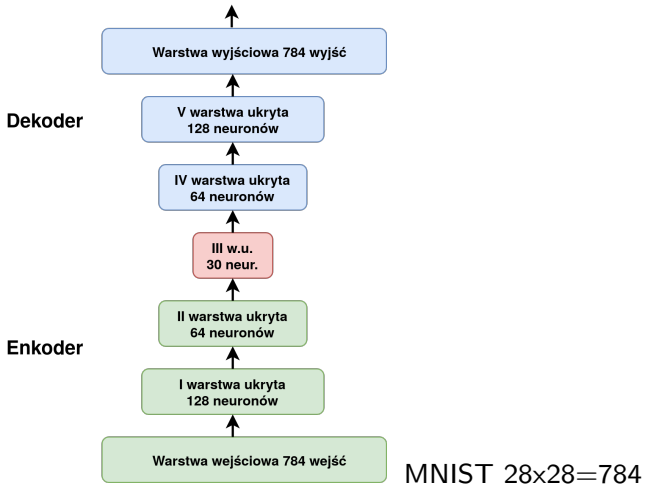
```
stacked_encoder = keras.models.Sequential([
  keras.layers.Flatten(input_shape=[28, 28]),
  keras.layers.Dense(90, activation="selu"),
  keras.layers.Dense(25, activation="selu"),
])

stacked_decoder = keras.models.Sequential([
keras.layers.Dense(90, activation="selu", input_shape=[30]),
keras.layers.Dense(28 * 28, activation="sigmoid"),
keras.layers.Reshape([28, 28])
])

stacked_autoencoder = keras.models.Sequential([stacked_encoder, stacked_decoder])
stacked_autoencoder.compile(loss="binary_crossentropy",
  optimizer=keras.optimizers.SGD(lr=1))
```

Wiązanie wag (ang. tying weights) dekodera z enkoderem

Rozwiązanie często stosowane dla symetrycznych lub częściowo symetrycznych autenkoderów.
Niech W_L oznacza wagi w L-tej warstwie autoenkodera z rysunku:

$$\begin{aligned} W_1 &= W_5^T \\ W_2 &= W_4^T. \end{aligned} \tag{1}$$

(tzw. transponowane warstwy konwolucyjne)



Zalety: Przeważnie lepsze wyniki mimo dwa razy mniejszej liczby parametrów zmienianych podczas treningu.

Autoencodery

```
dense1 = keras.layers.Dense(90, activation="selu")
dense2 = keras.layers.Dense(25, activation="selu")

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

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

tied_autoencoder = keras.models.Sequential([encoder, decoder])
```

Autoenkodery splotowe i rekurencyjne

Autoenkodery splotowe - przydatne do większych obrazów - enkoder i dekoder zawierają warstwy splotowe - pamiętamy, że sieci splotowe dobrze nadają się do przetwarzania obrazów.

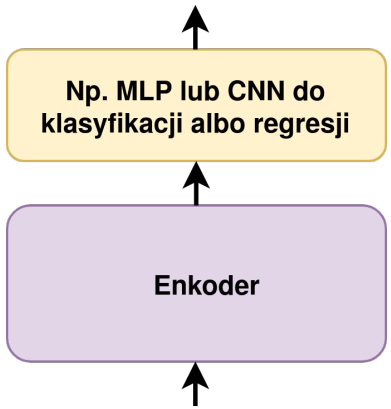
Autoenkodery rekurencyjne - przetwarzanie sekwencji (np. przetwarzanie tekstów, języka naturalnego, szeregów czasowych) - enkoder jest siecią rekurencyjną, która na wyjściu generuje reprezentacje w postaci wektora, dekoder dekoduje wektor do postaci odpowiedniej sekwencji

Autoencoder splotowy

```
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)
])

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, 4 padding="same", activation="selu"),
  keras.layers.Conv2DTranspose(1, kernel_size=3, strides=2, padding="same", activation="sigmoid"), keras.layers.Reshape([28, 28])
])
conv_autoencoder = keras.models.Sequential([encoder, decoder])
```

Gdy mamy dużo danych nieoznakowanych (bez etykiet) i mało danych oznakowanych (z etykietami) - sytuacja bardzo często spotykana w praktyce

Można wytrenować autoenkoder na wszystkich danych, a następnie wykorzystać enkoder do ekstrakcji cech oraz zastąpić dekoder odpowiednią niezbyt dużą siecią neuronową, którą należy wytrenować na danych oznakowanych
(podobny trik do wykorzystania bazy konwolucyjnej razem z transfer learning). - uczenie dużej sieci na małej liczbie danych oznakowanych spowodowałoby prawdopodobnie przeuczenie sieci !!!



Ręczne etykietowanie danych

Jest pracochłonne - może być bardzo żmudne oraz wymagać zatrudnienia sporej grupy osób.

Zachłanne uczenie warstwowe (ang. greedy layerwise training)

Zamiast trenować autoenkoder z dużą liczbą warstw można trenować małe enkodery pojedynczo i stworzyć z nich głęboki autoenkoder. Po wytrenowaniu małego autoenkodera reprezentacja danych z jego enkodera jest wykorzystywana do trenowania następnego niewielkiego autoenkodera. Następnie składa się jak kanapkę duży autoenkoder z małych enkoderów.

Historia - Początki XXIw - uczenie nienadzorowane autoenkoderów oraz sieci Deep Belief

W celu przezwyciężenia m.in. problemu zanikania gradientu stosowano dwie fazy:
– najpierw uczenie nienadzorowane w celu wstępnego doboru wag - zachłanne uczenie warstwowe stosowano do uczenia nienadzorowanego autoenkoderów lub sieci Deep Belief złożonych z ograniczonych maszyn Boltzmanna RBM (ang. Restricted Boltzmann Machines).
– następnie uczenie nadzorowane głębokiej sieci Deep belief złożonej z wielu RBM lub głebokiego autoenkodera złożonego z małych enkoderów

Autoenkodery odszumiające (ang. denoising autoencoders) i rzadkie

Nie trzeba stosować autoenkoderów z efektywną reprezentacji. Można zaszumić dane wejściowe. Jeżeli autoenkoder nauczy się rekonstruować dane mimo zakłóconych wejść, to znaczy, że nauczył się wyznaczać przydatne cech na podstawie których można dokonać dobrej rekonstrukcji.

Najczęściej szum gaussowski (wartość oczekiwana 0) dodawany do wejść lub losowe porzucanie części wejść (można łatwo zaimplementować używając warstwy dropout).

Inny sposób do wyodrębnienia istotnych cech jest stosowany w autoenkoderach rzadkich (ang. sparse) - można wymusić aby tylko kilka procent neuronów w ostatniej warstwie enkodera była aktywna poprzez zastosowanie np. regularyzacji L1. W tej warstwie często stosowana jest logistyczna funkcja aktywacji keras.activations.sigmoid. W [AGeron] podano też inne sposoby tworzenia enkoderów rzadkich.

Autoencoder odzumiający - zakłócenie (szum) z 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")
])

decoder = keras.models.Sequential([
  keras.layers.Dense(100, activation="selu", 25),
  keras.layers.Dense(28 * 28, activation="sigmoid"),
  keras.layers.Reshape([28, 28])
])
dropout_autoencoder = keras.models.Sequential([encoder, decoder])
```

Autoenkoder rzadki - reprezentacja rzadka z L1

```
encoder = keras.models.Sequential([
  keras.layers.Flatten(input_shape=[28, 28]),
  keras.layers.Dense(90, activation="selu"),
  keras.layers.Dense(200, activation="sigmoid"),
  keras.layers.ActivityRegularization(l1=1e-3)
])

decoder = keras.models.Sequential([
  keras.layers.Dense(90, activation="selu", input_shape=200),
  keras.layers.Dense(28 * 28, activation="sigmoid"),
  keras.layers.Reshape([28, 28])
])
sparse_autoencoder = keras.models.Sequential([encoder, decoder])
```

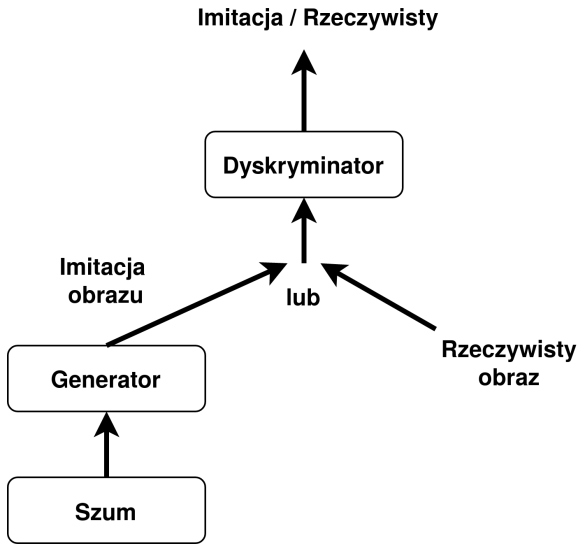
Autoenkodery

Są też inne autoenkodery...
(np. autoenkodery wariacyjne, które znacznie straciły na popularności po opracowaniu sieci GAN).

Generatywne sieci przeciwstawne GAN (ang. Generative Adversarial Networks)

Sieci GAN zaproponowano w 2014 w artykule https://arxiv.org/pdf/1406.2661.pdf
Mogą np. generować obrazy imitujące prawdziwe.

Na wejścia generatora podawane są liczby pseudolosowe (przeważnie rozkład gaussowski), które można traktować podobnie jak reprezentacje ukryte w autoenkoderach. Generator generuje z tych liczb imitację obrazu (podobnie jak dekoder w autoenkoderach). Dyskryminator (krytyk) rozpoznaje czy na jego warstwę wejściową podano obraz rzeczywisty czy imitację.



Podczas uczenia generator i dyskryminator mają przeciwstawne cele. Dyskryminator stara

Generatywne sieci przeciwstawne GAN (ang. Generative Adversarial Networks)

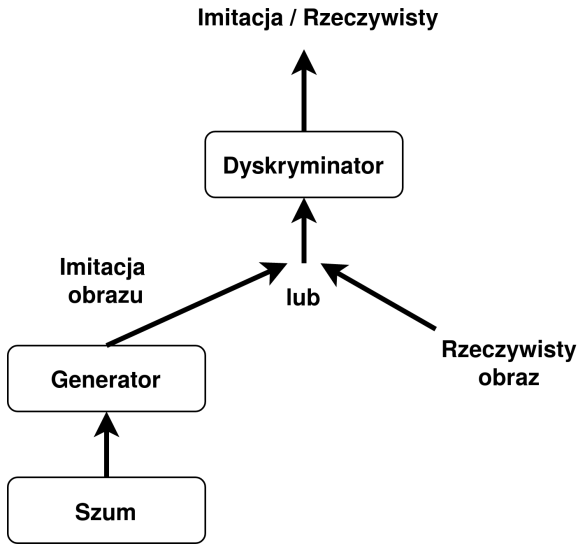
Każda iteracja uczenia jest podzielona na dwie fazy:

1. **Trening dyskryminatora (wagi generatora niezmieniane - zamrożone)** - w celu utworzenia zbioru danych uczących losowana jest grupa rzeczywistych obrazów i generowana jest taka sama liczba imitacji. Rzeczywiste obrazy mają etykiety 1, a imitacje 0. Funkcją straty jest binarna entropia (w ostatniej warstwie neuron z funkcja aktywacji keras.activations.sigmoid, ponieważ zadanie klasyfikacji binarnej).

2. **Trening generatora (wagi dyskryminatora zamrożone !!!)** - generator tworzy grupę obrazów podawanych na dyskryminator. Przypisane są im etykiety 1, mimo iż nie są obrazami prawdziwymi - są one używane tylko w tej fazie, więc nie zepsują dyskryminatora. Od wyjścia dyskryminatora propagujemy jest błąd metodą wstecznej propagacji, co powoduje zmianę wag generatora umożliwiającą tworzenia lepszych imitacji (które powodują, że wartość wyjścia dyskryminatora jest bliższa 1). Przypomnienie - wagi dyskryminatora w tej fazie są zamrożone.

Warto zwrócić uwagę, że generator nigdy nie ma bezpośredniego dostępu do rzeczywistych obrazów i uczy się tylko na podstawie wstecznej propagacji błędów z dyskryminatora. Im lepszy dyskryminator, tym lepiej uczy się generator.

Generatywne sieci przeciwstawne GAN (ang. Generative Adversarial Networks)



Generator i dyskryminator tworzy się w Kerasie najczęściej w sposób sekwencyjny ze standardowych warstw. Generator i dyskryminator mogą być sieciami konwolucyjnymi. Uczenie wymaga napisania własnej funkcji składającej się z kilku linii kodu. Przykład sieci GAN dla zbioru MNIST wraz z kodem funkcji do nauki pokazany jest w książce [AGeron]. W przykładzie tym prosty generator i dyskryminator składają się z trzech warstw gęstych (architektura MLP).

Generatywne sieci przeciwstawne GAN (ang. Generative Adversarial Networks)

```
generator = keras.models.Sequential([
  keras.layers.Dense(100, activation="selu", input_shape=25),
  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])
```

Generatywne sieci przeciwstawne GAN (ang. Generative Adversarial Networks)

Zbiór MNIST jest prosty. Obrazy są niewielkie 28x28. W przypadku bardziej skomplikowanych zbiorów, nie jest tak łatwo - potrzebne są dodatkowe ulepszenia, żeby przezwyciężyć problemy:

Załamanie modu (ang. mode collapse) - występuje gdy wyniki generatora stają się coraz mniej zróżnicowane. Załóżmy, że generatorowi udaje się tworzyć lepsze imitacje jednych klas od innych, co w konsekwencji doprowadzi, że będzie produkował coraz więcej imitacji tych klas i zapominał jak tworzyć inne klasy. Po jakimś czasie dyskryminator się udoskonali, więc generator może się cyklicznie przerzucać na generowanie kilku innych klas i nigdy nie osiągnąć dokładności w żadnej.

Niestabilny trening - oscylacje i rozbieganie się parametrów trenowanych do walki ze sobą sieci - generatora i dyskryminatora.

Opracowano różne metody i zalecenia w celu zmniejszenia tych problemów. Jest publikowana spora liczba badań na ten temat.

Głębokie splotowe sieci GAN (DCGAN ang. Deep convolutional GAN)

W związku z problemami z uczeniem opracowano różnorodne modyfikacje np. model WGAN (GAN Wassersteina).

Różnice między WGAN i standardową GAN:

– inna funkcja straty - f. staty Wassersteina.

– etykiety 1 (dla obrazów autentycznych) i –1 (dla wygenerowanych imitacji)

– liniowa funkcja aktywacji (zamiast logistycznej) w ostatniej warstwie dyskryminatora

– szkolenie dyskryminatora jest wykonywane znacznie częściej niż uczenie generatora (dodatkowo przycinanie wag dyskryminatora)

Później powstała udoskonalona wersja WGAN-GP (ang. Wassertein GAN-Gradient Penalty), w którym wprowadzono modyfikację do dyskryminatora:
– ograniczenie gradientu w f. strat dyskryminatora zamiast przycinania jego wag
– brak normalizacji wsadowej w dyskrminatorze

Głębokie splotowe sieci GAN (DCGAN ang. Deep convolutional GAN)

W związku z problemami z uczeniem przydatne mogą być niektóre wskazówki autorów pracy https://arxiv.org/abs/1511.06434 dotyczące tworzenia głębokich splotowych GAN :

– w dyskryminatorze i generatorze zastosowanie normalizacji wsadowej (ang. batch normalization) - oprócz warstw wyjściowych

– we wszystkich warstwach pooling wartość parametru stride > 1 np. stride=2

– w głębszych sieciach usunąć warstwy gęste

– w warstwie wyjściowej generatora zastosować f.aktywacji tanh, a w pozostałych ReLU

– we wszystkich warstwach dyskryminatora przeciekającą ReLU - keras.activations.LeakyReLU

$$f(x) = \begin{cases} x & \text{for} \quad x \geqslant 0, \\ -\alpha x & \text{for} \quad x < 0, \end{cases}$$

gdzie α jest niewielką dodatnią liczbą $<< 1$.

Generatywne sieci przeciwstawne GAN (ang. Generative Adversarial Networks)

Przykładowe dodatkowe sposoby zmniejszające problemy uczenia [AGeron]:

Odtwarzanie doświadczenia (ang. experience replay) - obrazy utworzone przez generator w każdej iteracji trafiają do bufora odtwarzania, z którego stopniowo usuwane są starsze imitacje. Dyskryminator uczony jest za pomocą wszystkich imitacji z bufora, a nie tylko imitacji tworzonych na bieżąco przez generator. W ten sposób w uczeniu dyskryminatora biorą udział też starsze obrazy, co powoduje, że jest mniejsze ryzyko jego nadmiernego dopasowania wyłącznie do najnowszych obrazów.

Rozróżnianie mini-batchy (ang. mini batch discripation) - mierzone jest podobieństwo obrazów w mini-batchu utworzonych przez generator. Jest ono przekazywane jako dodatkowe wejście do dyskryminatora, co umożliwia łatwe odrzucenie przez dyskryminator imitacji o zbyt małym zróżnicowaniu. Zmusza to generator do tworzenia zróżnicowanych obrazów i zapobiega załamaniu modu.

Generatywne sieci przeciwstawne GAN (ang. Generative Adversarial Networks)

Jest wiele różnorodnych innych metod poprawy uczenia i uzyskiwanie bardziej realistycznych obrazów za pomocą sieci GAN, np. w [AGeron] opisano m.in. sieci StyleGAN, odpowiednią metodę inicjalizacji wag GAN, rozrost progresywny GAN (kolejno dodawane są warstwy splotowe w generatorze i dyskryminatorze pozwalające uzyskiwać coraz większe obrazy 4x4, 8x8, . . . , 512x512). Niektóre techniki zostały opracowane przez zespół z NVidia.

Uczenie sieci GAN może wymagać sporej liczby eksperymentów, dużej mocy obliczeniowej i stosowania różnorodnych usprawnień.

Sieci GAN są obecnie intensywnie badane i rozwijane.

Generatywne sieci przeciwstawne GAN (ang. Generative Adversarial Networks)

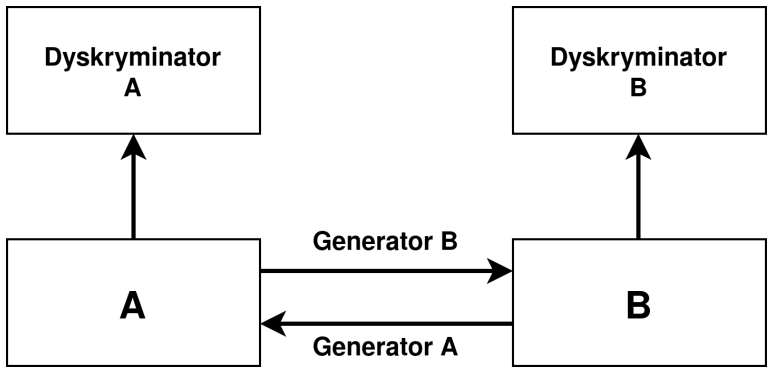
Uwaga

Sieci GAN nie muszą być zastosowane do generowania obrazów 2D ale mogą zostać użyte do wielu innych celów.

Cycle GAN

Uwaga

Transfer stylów np. generowanie obrazów w stylu słynnego malarza
Dwa generatora i dyskryminatory



A i B oznaczają domeny, np. A może zawierać zdjęcia, B - zbiór obrazów w stylu słynnego malarza wygenerowanych z A.

Cycle GAN

Całkowita f.straty jest sumą ważoną następujących kryteriów [DFoster]:

Poprawność - skuteczność oszukiwania dyskryminatorów przez obrazy utworzone w generatorach

Rekonstrukcja - błąd rekonstrukcji po przepuszczeniu obrazu przez obydwa generatory - w obu kierunkach (czy uzyska my znowu obraz zbliżony do początkowego)

Tożsamość - czy obraz nie zostaje zmieniony, jeżeli np. obrazu z domeny A przepuścimy przez generator przekształcający na A

Cycle GAN

Zamiana wszystkich zebr na konie na obrazie - deep fake

modele dyfuzyjne (rozpraszające)

Już w 2015 https://arxiv.org/abs/1503.03585

W 2020 DDPM (ang. Denoising Diffusion Probabilistic), który generował realistyczne obrazy.

W 2021 badacze Alex Nichol, Prafulla Dhariwal z OpenAI w https://arxiv.org/abs/2102.09672 pt. "Improved Denoising Diffusion Probabilistic Models" zaproponowali w DDPM kilka usprawnień, które pozwoliły uzyskiwac lepsze wyniki od GAN. https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/

modele dyfuzyjne (rozpraszające)

DDPM są łatwiejsze do wytrenowania od GAN i pozwalają uzyskiwać wyższej jakości, bardzo zróżnicowane obrazy.

modele dyfuzyjne (rozpraszające)

Dobre wyjaśnienie funkcjonowania: https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/

modele dyfuzyjne (rozpraszające)

Zaczynamy od obrazu początkowego x_0 i w kolejnych taktach dodajemy do obrazu trochę szumu gaussowskiego o wartości oczekiwanej (średniej) zero i wariancji β_t, gdzie t oznacza numer taktu. Szum ten jest niezależny dla każdego piksela. Liczba taktów $T = 4000$ (od 2021r. dawniej na ogół mniej taktów). Współczynnik β_t jest zmieniany według odpowiedniego harmonogramu w kolejnych taktach (wolniej na początku i na końcu). W każdym takcie wartości pikseli są nieznacznie skalowane przez czynnik $\sqrt{1-\beta_t}$ tzn.
rozkład prawdopodobieństwa q procesu rozpraszania do przodu

$$q(x_t|x_{t-1}) = \mathcal{N}\left(\sqrt{1-\beta_{t-1}} \cdot x_{t-1}, \quad \beta_t \mathrm{I}\right), \tag{2}$$

gdzie x_t oznacza obraz w takcie t, I jest macierzą jednostkową. W kolejnych taktach coraz bardziej obszar początkowy jest zanurzany w szumie (dyfuzja).

Proces wsteczny

W procesie wstecznym model (sieć U-net) jest trenowany do generowania x_t z x_{t-1}. Musimy wykonać aż $T = 4000$ taktów wnioskowań, żeby uzyskać obraz wynikowy - dlatego nie jest to szybka metoda.

Polecana literatura do samodzielnej nauki

[AGeron] Aurélien Géron, Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Wyd. III, Helion, 2023

[FChollet] Francois Chollet, Deep Learning. Praca z językiem Python i biblioteką Keras, Helion, 2019

[DFoster] David Foster, Deep learning i modelowanie generatywne. Jak nauczyć komputer malowania, pisania, komponowania i grania, Helion, 2021

Dziękuję za uwagę

Dziękuję za uwagę