UMCS
UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
W LUBLINIE
Wydział Matematyki, Fizyki i Informatyki

Kierunek: **Informatyka**
Specjalność: **Sztuczna inteligencja**

Filip Ręka
nr albumu: 296595

Analiza porównawcza zastosowań tradycyjnych oraz wariacyjnych autoenkoderów

Comparative analysis of traditional and variational autoencoders' applications

Praca licencjacka
napisana w Katedrze Cyberbezpieczeństwa
pod kierunkiem dr hab. Michała Wydry

Lublin 2021

Spis treści

Wstęp

Autoenkoder jest jednym z rodzajów sieci neuronowych, której zadaniem jest nauczenie się zakodowania nie oznaczonych danych. Kod jest kolejnie wykorzystywany do ponownego wygenerowania wejścia sieci. Autoenkoder uczy się reprezentacji zbioru danych do zmiennych ukrytych przez ignorowanie nie istotnych danych. Wariacyjne autoenkodery są popularnymi modelami generacyjnymi. Zostały zaproponowane przez Diederika P Kingma i Maxa Wellinga w roku 2014.[1] Najczęściej zostają one skategoryzowane do modeli uczenia częściowo nadzorowanego. Znajdują zastosowanie w generacji obrazów, tekstu, muzyki oraz w detekcji anomalii. W przeciwieństwie do tradycyjnych autoenkoderów prezentują pobabilistyczne podejście do generowania zmiennych ukrytych. Swoją popularność zawdzięcza swojej budowie, która jest oparta na sieciach neuronowych oraz możliwości trenowania ich przy pomocy metod gradientowych.

Rozdział 1

Tradycyjny autoenkoder

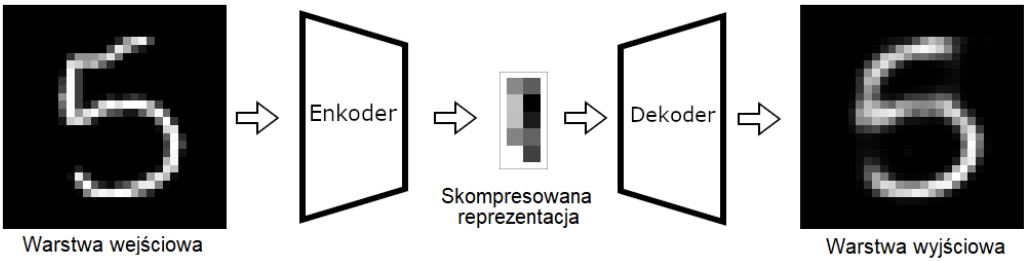
1.1 Informacje wstępne

Autoenkoder jest specyficzną wersją sieci neuronowej składającej się z dwóch części: enkodera, który koduje dane wejściowe oraz dekodera, który na podstawie kodu rekonstruuje wejście.[2] Architektura enkodera wymaga aby jego warstwa wyjściowa generująca reprezentacje danych była mniejsza niż warstwa wejściowa. Często zwężenie to jest nazywane *bottle neck*. Model na swoją warstwę wejściową oraz wyjściową dostaję te same dane. Powiedzmy że mamy dane wejściowe X o wymiarze m oraz chcemy je zakodować do wymiaru n. Formalnie możemy zapisać:

$$\text{Enkoder } E : \mathbb{R}^m \to \mathbb{R}^n$$
$$\text{Dekoder } D : \mathbb{R}^n \to \mathbb{R}^m$$
$$\text{gdzie } n < m$$

Calem *bottle neck-a* jest skompresowanie wejścia i zachowanie w ukrytych wartościach jak najwięcej informacji. W momencie, kiedy $n = m$ model przekazałby wartości z pierwszej warstwy na ostatnią bez potrzeby kompresji. Celem treningu całego autoenkodera jest zminimalizowanie błędu pomiędzy prawdziwymi danymi wejściowymi, a tymi odkodowanymi ze skompresowanych wartości. W przypadku obrazów funkcją straty może być na przykład błąd średniokwadratowy lub binarna entropia krzyżowa, która powie nam, jak wynik różni się od wejścia.

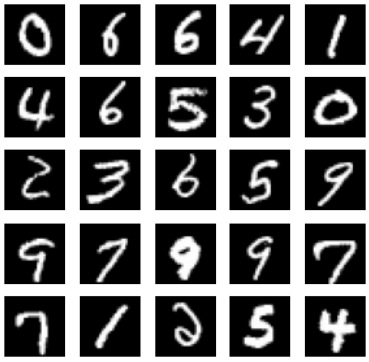
$$\mathcal{L}(x, \hat{x}) = \frac{1}{m}\sum_{i=0}^{m}(x_i - \hat{x}_i)^2 = \frac{1}{m}\sum_{i=0}^{m}(x_i - D(E(x_i)))^2$$



Rysunek 1.1: Schemat budowy autoenkodera.

1.1.1 Zbiór danych MNIST

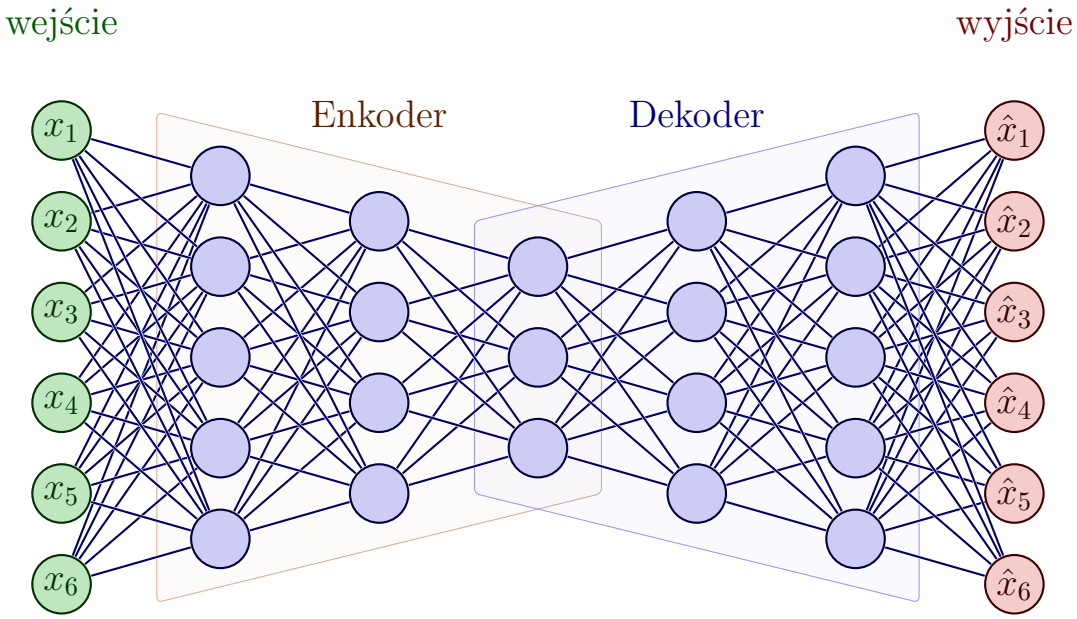
Zbiór danych MNIST (*Modified National Institute of Standards and Technology*) jest zbiorem wielu odręcznie pisanych cyfr.[3] Znajduje szerokie zastosowanie w nauce i prezentacjach możliwości modeli uczenia maszynowego. W jego skład wchodzi 60,000 obrazów przeznaczonych do treningu modeli oraz 10,000 do testów. Obrazy są czarno-białe i mają wymiary 28 na 28 pikseli.



Rysunek 1.2: Przykładowe obrazy ze zbioru danych.

1.2 Budowa

Jak już zostało napisane, autoenkoder składa się z dwóch sieci neuronowych. Enkoder jak i dekoder są w pełni połączonymi sieciami neuronowymi.



Rysunek 1.3: Wizualizacja sieci tworzącej autoenkoder.

Obrazek 1.3 przedstawia prosty autoenkoder kompresujący wejście siedmiowymiarowe do kodu o długości trzy. Enkoder jak i dekoder mają po dwie warstwy ukryte. Odbicie lustrzane architektury nie jest konieczne aby model działał poprawnie, jednak zwyczajem jest używanie takiej architektury.
Hiperparametramy modelu, które możemy ustalić przed jego treningiem to:

- Ilość warstw ukrytych - jeśli wiemy, że nasze dane są skomplikowane, dodatkowe warstwy ukryte będą miały pozytywny wpływ na otrzymywane rezultaty, ponieważ większa ilość warstw sprawia, że model jest w stanie nauczyć się bardziej skomplikowanych funkcji.[4, 5]

- Ilość neuronów w poszczególnych warstwach - autoenkoder powinien posiadać w każdej warstwie mniej neuronów niż w poprzedniej. W ten sposób model nie będzie "oszukiwa" i zostaje zmuszony do reprezentacji jak najlepszej kompresji.

- Funkcja straty - najlepszymi funkcjami straty do treningu autoenkodera jest błąd średniokwadratowy lub binarna entropia krzyżowa w przypadku, kiedy dane są w przedziale od 0 do 1.

- Rozmiar kodu - jest najistotniejszy parametr dla nas i dla modelu. Dłuższa długość kodu oznacza zachowanie więcej istotnych elementów, a co za tym idzie lepsze

odwzorowanie przez dekoder. Z drugiej strony używając, dłuższego wektora zmiennych ukrytych dostajemy gorszą kompresje danych. Długość kodu musimy dobrać w zależności od problemu, który chcemy rozwiązać używając model.

1.3 Zastosowania

1.3.1 Redukcja wymiaru

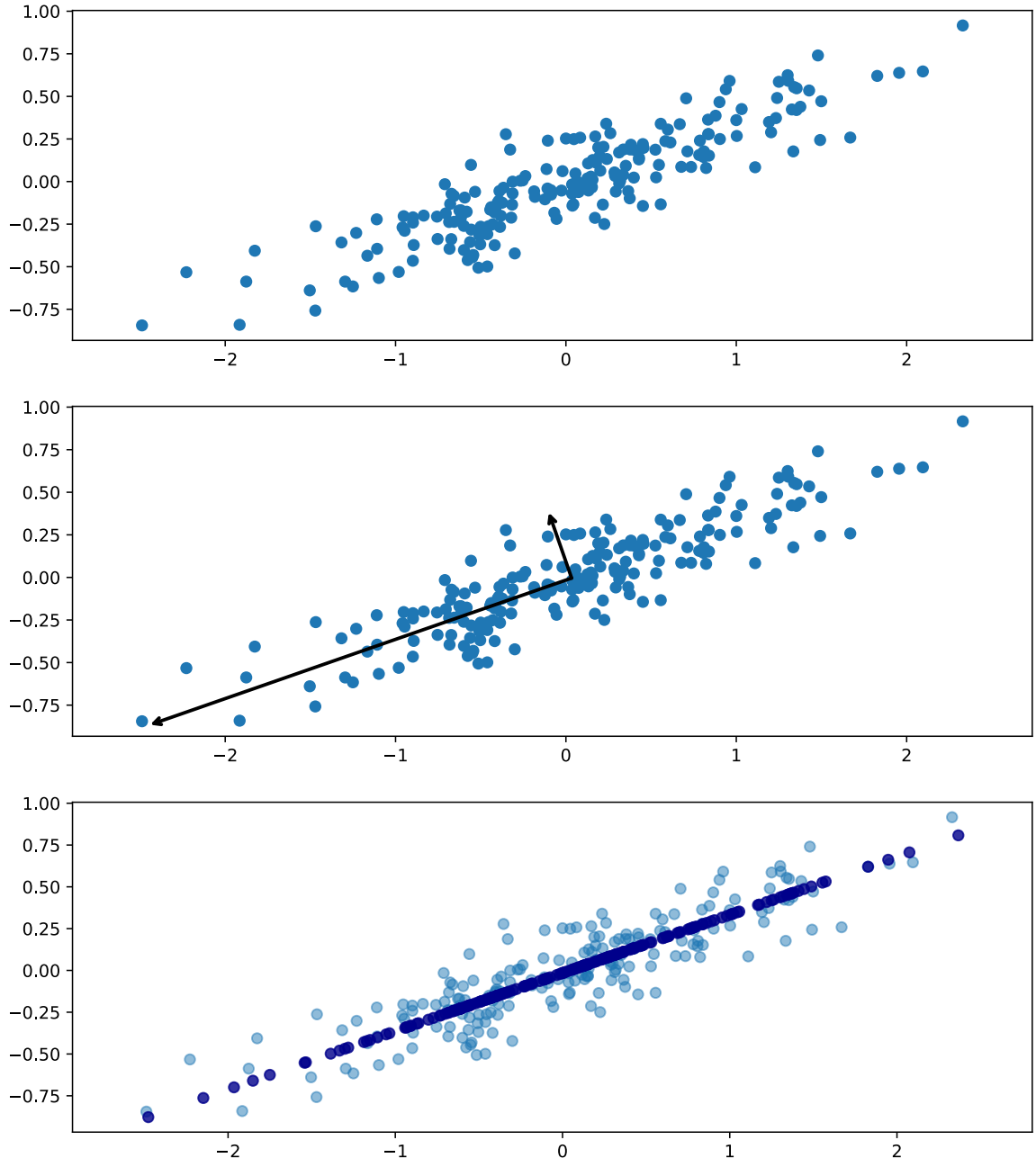
Redukcja wymiaru jest procesem zmniejszenia liczby zmiennych przeznaczonych do analizy, a zarazem zachowanie w nich jak najwięcej istotnych informacji. Powody dla których chcemy zmniejszyć wymiarowość danych to między innymi:

- Część zmiennych opisująca dane jest ze sobą nadmiernie skorelowana lub niesie ze sobą cechy, które nie są istotne statystycznie i usunięcie ich nie wpływa na poprawę działania modeli statystycznych lub uczenia maszynowego.
- Dane bardzo wysokiego wymiaru są trudne do analizy lub operacje na nich zajmują tak dużo czasu i zasobów że stają się one bezużyteczne.
- Wielowymiarowe dane jest ciężej zwizualizować. Możemy je zredukować do jedno, dwu, lub trzy wymiarowej reprezentacji co pozwoli nam na proste narysowanie wykresu zrozumiałego przez każdego.

Autoenkoder nie jest jednym sposobem na redukcję wymiarów. Najbardziej rozpowszechnioną metodą jest *PCA*.

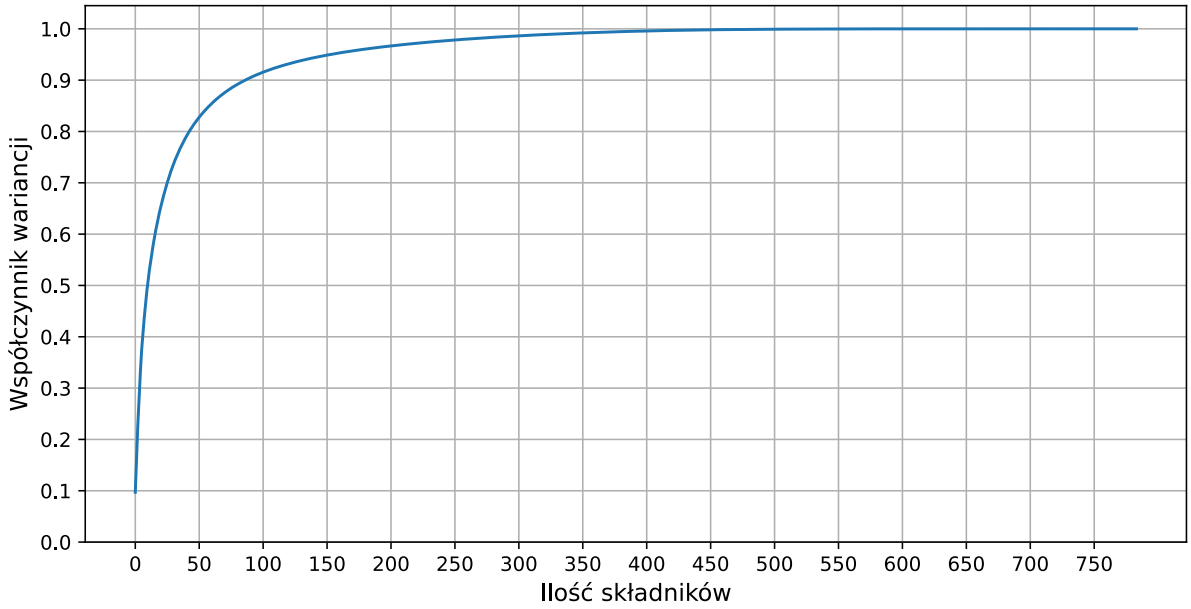
Analiza składowych głównych

Analiza składowych głównych (*ang. Principal Components Analysis,* ***PCA***) służy do wyznaczania jak najmniejszej ilości nowych zmiennych, mówiących jak najwięcej o zbiorze danych. Wielowymiarowe dane koncentrują się w pewnych podprzestrzeniach oryginalniej przestrzeni. Analiza PCA pozwala znaleźć te podprzestrzenie, które są wektorami pełniącymi rolę nowych osi, które je lepiej opisują.[6] Używając tej metody ograniczamy się tylko do przekształceń liniowych. Ilość wektorów względem których można zredukować dane jest równa wymiarowi danych. Środkowy obrazek na rysunku 1.4 przedstawia właśnie te wektory na przykładzie dwuwymiarowego zbioru punków. Linia względem której spłaszczane są dane jest tą, która minimalizuje odległość do niej od wszystkich punktów. Dolny wykres rysunku 1.4 pokazuje już spłaszczone dane do jednego wymiaru. Nowe wektory są wybierane w taki sposób, aby wariancja rzutów poszczególnych obserwacji była jak największa, co gwarantuje nam odwzorowanie jak największej ilości danych. Każdy kolejny wektor zachowuję się w taki sam sposób oraz jest ortonormalny do poprzednich.

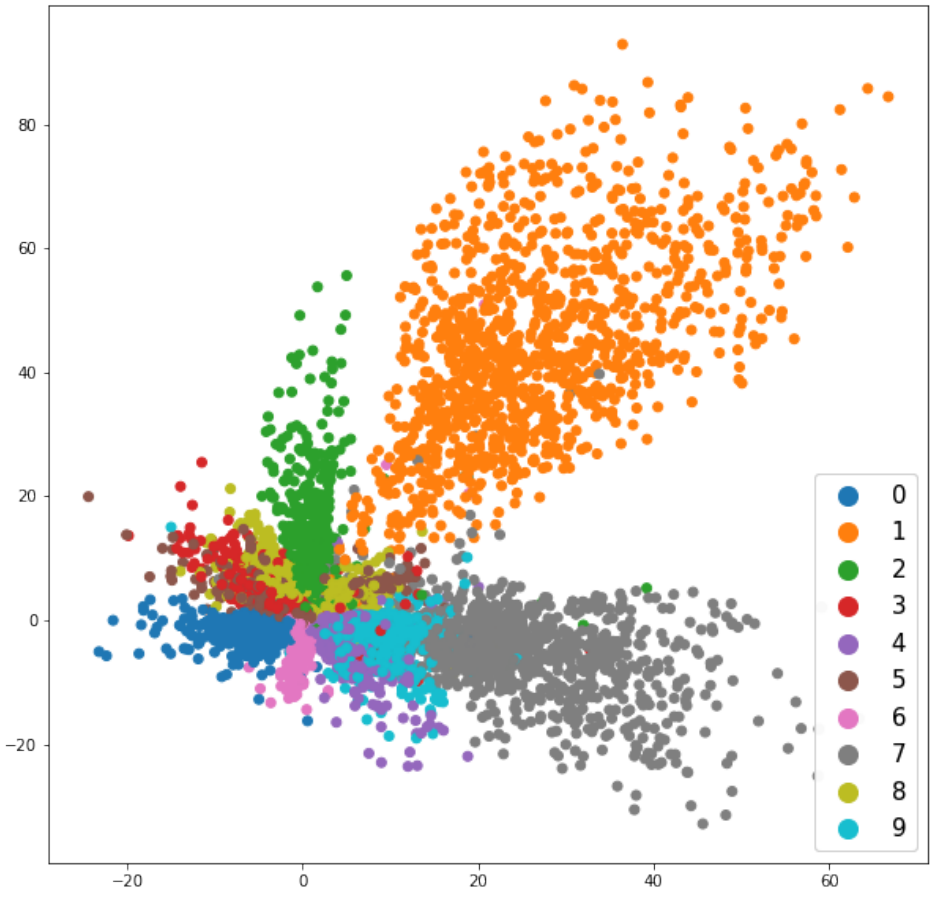


Rysunek 1.4: Wizualizacja PCA na zbiorze punktów w przestrzeni dwuwymiarowej.

Wykres 1.5 pokazuje zależność między ilością składników PCA, a procentem wariancji opisywanym przez składniki na podstawie zbioru danych MNIST. Ilość składników mieści się w przedziale od 1 do 784 (28 razy 28 pixeli). Jak można zauważyć dane reprezentowane przez około 80 wartości są w stanie opisać 90% wariancji danych co jest znaczą redukcją z 784. Przy 400 składnikach osiągamy praktycznie 100% pokrycia.



Rysunek 1.5: Procent wariancji opisywany przez ilość składników.



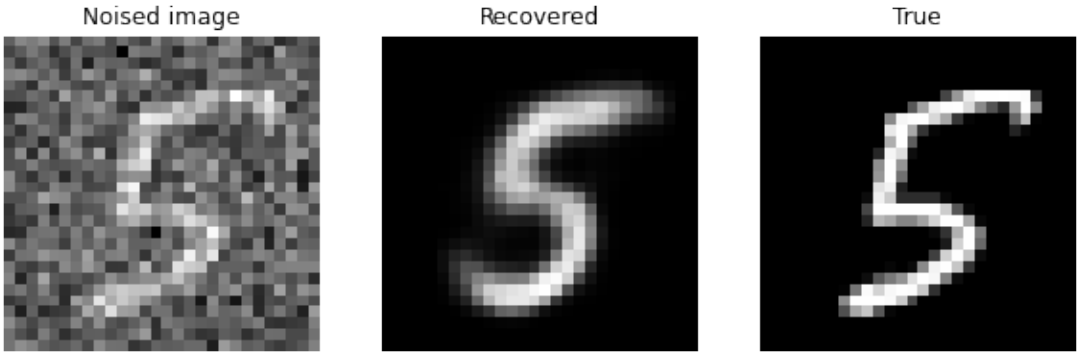
Rysunek 1.6: Przestrzeń dwuwymiarowej zmiennej ukrytej.

Autoenkoder

1.3.2 Odszumianie obrazów

Model autoenkodera przeznaczony do odszumiania danych, często dostaje swoją nazwę i jest określany mianem DAE *(Denoising autoencoder)*. DAE, tak jak zwykły auto-

enkoder, próbuje w jak najlepszy sposób skompresować dane, zachowując jak najwięcej istotnych informacji. Najważniejszą różnicą między tymi modelami są dane, które dostają na wejście i wyjście. Obrazy przyjmowane na warstwę wejściową są zaszumione natomiast te z warstwy wyjściowej pochodzą prosto ze zbioru danych. Powodem dla którego autoenkodery tak dobrze nadają się do odszumiania jest ich umiejętność kompresji danych. Kompresja, której dokonują te modele jest stratna. W przypadku kiedy naszym głównym zadaniem jest jak najlepsze odtworzenie danych wejściowych jest to kłopot, jednak w tym przypadku możemy wykorzystać tą własność na naszą korzyść. Porównując wyjście modelu z danymi bez szumu, zapewniamy, że model nauczy się w jakimś stopniu odtwarzać je poprawnie. Zmuszamy w ten sposób model do ignorowania nieistotnych części naszych danych oraz zapamiętywanie tylko tych, na podstawie których będzie możliwe jak najlepsze odwzorowanie wejścia sieci. Rysunek 1.7 pokazuje możliwości modelu na przykładzie zbioru danych MNIST. Do obrazów przeznaczonych do treningu został dodany szum. Zaszumiony obraz powstał przez dodanie do oryginalnego obrazu losowo wybranych wartości z rozkładu Gaussa $\mathcal{N}(0, 1)$ przemożonych przez stałą, która w tym przypadku wynosi 0.4. Następnie obrazy wejściowe zostały skompresowane do kodu o długości pięć, a następnie odkodowane przez dekoder.



Rysunek 1.7: Obraz z szumem, odzyskany oraz prawdziwy.

Wadą autoenkoderu przeznaczonego do odszumiania danych jest jego ścisłe powiązanie ze zbiorem danych, na których został wytrenowany. Model z parametrami wytrenowanymi na jednym zbiorze danych nie będzie się nadawał do innego zbioru, z którego danymi będziemy chcieli pracować. Jedynym rozwiązaniem tego problemu jest stworzenie nowego modelu przeznaczonego do użytku na nowych danych.

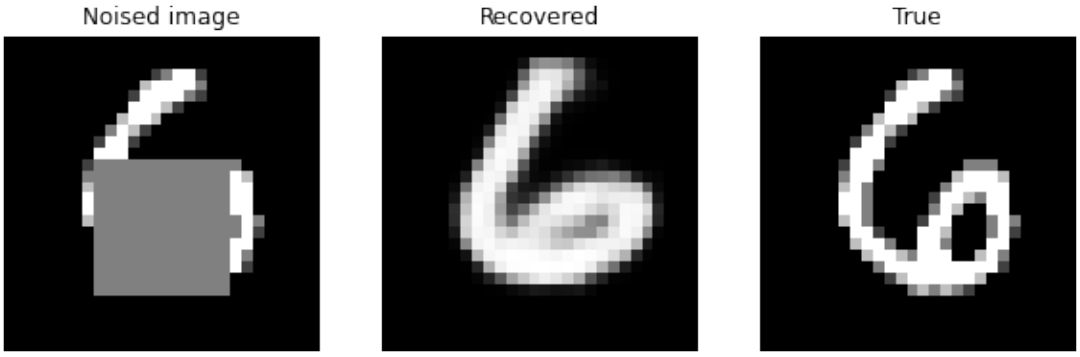
1.3.3 Uzupełnianie obrazów

Uzupełnianie obrazów ma na celu wypełnienie brakującej lub zamaskowanej części obszaru. Człowiek bez problemu jest sobie poradzić z tym zadaniem jednak dla komputera nie jest ono oczywiste. Bierze się to z tego, że jest ogromna ilość możliwości wypełnienia

nawet niewielkiej brakującej przestrzeni. Można wyróżnić dwa główne podejścia wypełniania obrazów:

- sieć posiada informację w którym miejscu obrazu jest luka
- sieć musi sama się nauczyć, które miejsce obrazu musi wypełnić

Rysunek 1.8 przedstawia drugie podejście.



Rysunek 1.8: Obraz z luką, odzyskany oraz prawdziwy.

1.4 Problemy z generacją nowych danych

Dobrym pytaniem jest czy przy pomocy kodu jesteśmy generować nowe dane bardzo podobne do tych co model otrzymał na wejściu. Wytrenowaliśmy sieć, która jest w stanie ze zmiennych ukrytych odkodować obraz, więc ustawiając wejście dekodera na losowy punkt z przestrzeni zmiennych powinniśmy być w stanie dostać obraz, który jest podobny do tych na których sieć została wytrenowana. Aby model mógł generować nowe dane muszą zostać spełnione dwa warunki:

- Nasza przestrzeń kodu (tzw. zmiennych ukrytych) musi być ciągła co znaczy że dwa punkty znajdujące się obok siebie będą dawać podobne dane jak zostaną odkodowane
- Przestrzeń musi być kompletna co znaczy, ze punkty wzięte z dystrybucji muszą dawaj wyniki mające sens

Tradycyjna architektura nie zapewnia nam przed treningiem, czy przestrzeń zmiennych ukrytych będzie spełniała te warunki. Spoglądając na rysunek 1.6 możemy zaobserwować, że przestrzeń zawiera luki. Szczególnie dobrze to widać między klasami oznaczającymi jedynki oraz siódemki. Kolejnym problemem widocznym na tej grafice jest brak separacji między klasami. Niektóre z nich są dobrze odseparowane od siebie, jednak inne całkowicie na siebie nachodzą, jak siódemki z dziewiątkami czy trójki z piątkami. Zadaniem

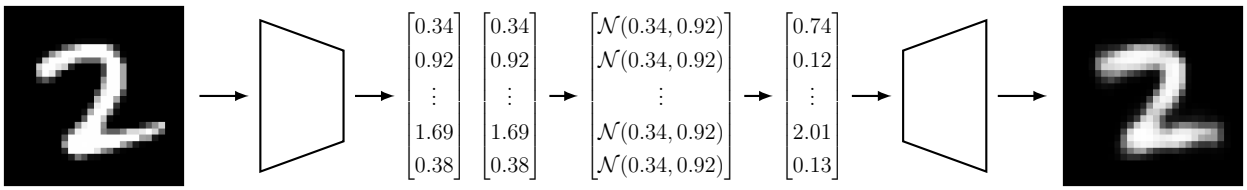
modelu jest jak najlepsze odzwierciedlenie skompresowanych danych, a nie dbanie o to, czy rozkład zmiennych kodu spełnia nasze warunki. Może się tak zdarzyć, że sieć nauczy się akurat takiej dystrybucji, która nam pasuje, ale jest to bardzo mało prawdopodobne. Jeśli chcemy zbudować model generacyjny musimy mieć zagwarantowane, że za każdym razem dostaniemy rozkład spełniający odpowiednie warunki.

Rozdział 2

Wariacyjny autoenkoder

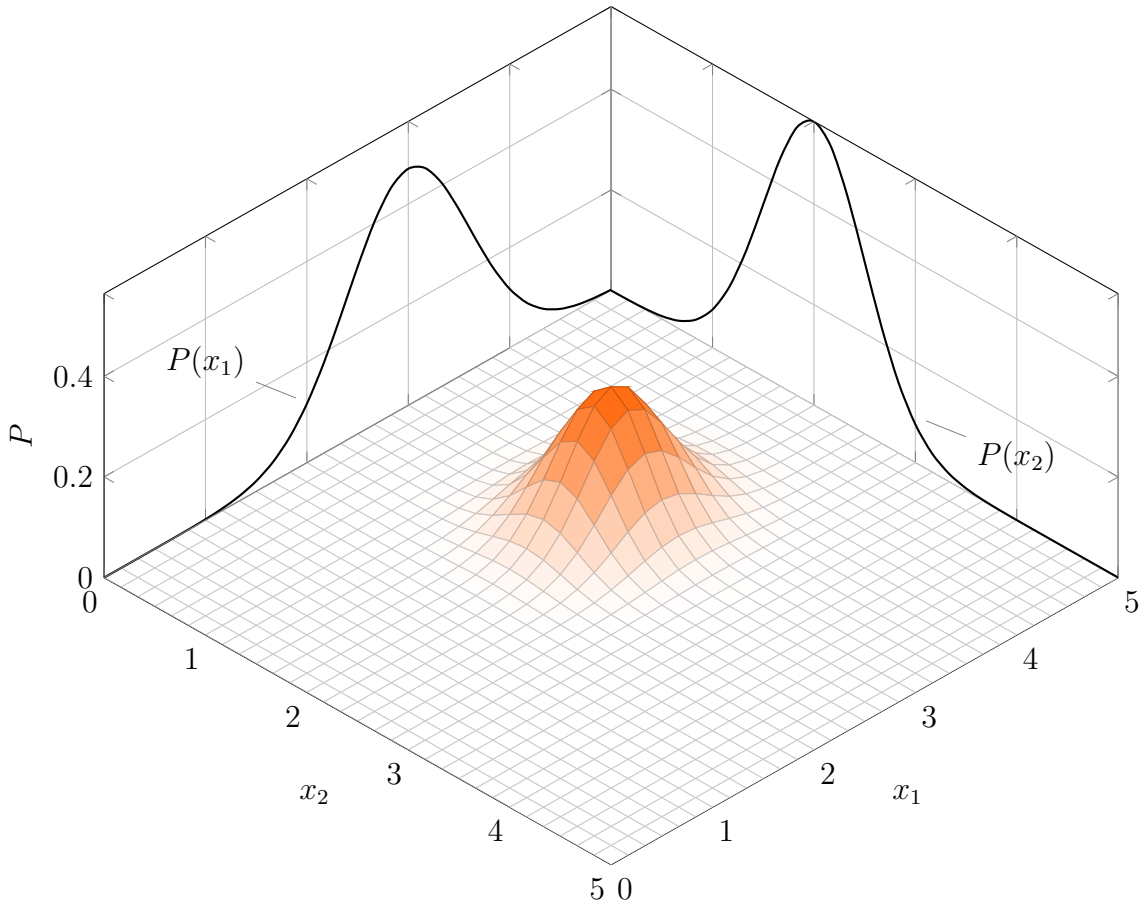
2.1 Informacje ogóle

Wariacyjny autoenkoder rozwiązuje problemy generacyjny tradycyjnego modelu. VAE ma na celu skompresowanie danych do określonego wielowymiarowego rozkładu ukrytego, a następnie z próbki z tej dystrybucji, próbuje jak najlepiej zrekonstruować wejście. Najczęściej rozkładami wybieranymi do tego celu są rozkłady normalne. Rozkład normalny jest opisywany przy pomocy dwóch wartości: średnia, która oznaczana jest znakiem μ oraz odchylenie standardowe oznaczane σ. Jeśli chcemy dane skompresować do kodu o długości n, enkoder wygeneruje dwa wektory n-wymiarowe, z którego jeden będzie przechowywał wartości średniej a drugi odchylenia standardowego dla każdego z n rozkładów normalnych.



Rysunek 2.1: Schemat budowy wariacyjnego autoenkodera.

Ograniczając enkoder do nauki wyłącznie tej dystrybucji z której próbkujemy losowo zmienne ukryte z których rekonstruujemy obrazy zapewniamy sobie gładką oraz ciągłą dystrybucje zmiennych losowych. Oczekujemy, że dowolna próbka z przestrzeni zostanie celnie odwzorowana przez dekoder. Dlatego wartości, które sa blisko będą generowały podobne wyjście.



Rysunek 2.2: Wielowymiarowy rozkład zmiennej.

2.2 Motywacja statystyczna

Powiedzmy ze istnieje zmienna ukryta z, która generuje obserwację x. Posiadamy tylko informację o x i chcemy się dowiedzieć z jakiego z dana obserwacja powstała. Aby tego dokonać powinniśmy policzyć $p(z|x)$. Z twierdzenia Bayesa wiemy że:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

Aby obliczyć rozkład marginalny $p(x)$ musimy policzyć:

$$p(x) = \int_z p(x,z)dz$$

Obliczenie tej całki jest bardzo trudne ponieważ z jest często wielowymiarowe i przestrzeń przeszukiwań jest kombinatorycznie zwyczajnie za duża aby korzystać z takich metod jak próbkowanie Monte Carlo łańcuchami Markowa.*citation needed :(*

2.3 Wnioskowanie wariacyjne

Rozwiązaniem tego problemu jest próba policzenia rozkładu $q(z|x)$, które będzie jak najlepiej odzwierciedlać $p(z|x)$ i będzie miał rozkład, który będziemy mogli policzyć.

2.3.1 Dywergencji Kullbacka-Leiblera

Jest to miara określająca rozbieżność między dwoma rozkładami prawdopodobieństwa. Nie można określić jej mianem metryki ponieważ nie jest symetryczna ($D_{KL}(P\|Q) \neq D_{KL}(Q\|P)$).
Naszym celem będzie zminimalizowanie jej.

$$q^*(z|x) = \text{argmin}_{q(z|x)\in Q}(D_{KL}(q(z|x)\|p(z|x)))$$

gdzie Q to rodzina prostych dystrybucji, na przykład rozkładu Gaussa

Policzmy:

$$D_{KL}(q(z|x)\|p(z|x)) = \mathbb{E}_{z\sim q(z|x)} \log \frac{p(z|x)}{q(z|x)} = \int_z q(z|x) \log \frac{q(z|x)}{p(z|x)} dz$$

Natrafiamy na kolejny problem ponieważ nie możemy $p(z|x)$ jednak jesteśmy w stanie to przepisać jako:

$$p(z|x) = \frac{p(z,x)}{p(x)}$$

Tu jest dużo matmy której nie chce mi się na razie pisać ale tu będzie ELBO (dolna granica dowodów).

Wybieramy sobie że nasza funkcja $q(z|x)$ będzie $\mathcal{N}(0, \boldsymbol{I})$. Dywergencja dla dwóch rozkładów normalnych wygląda w następujący sposób.

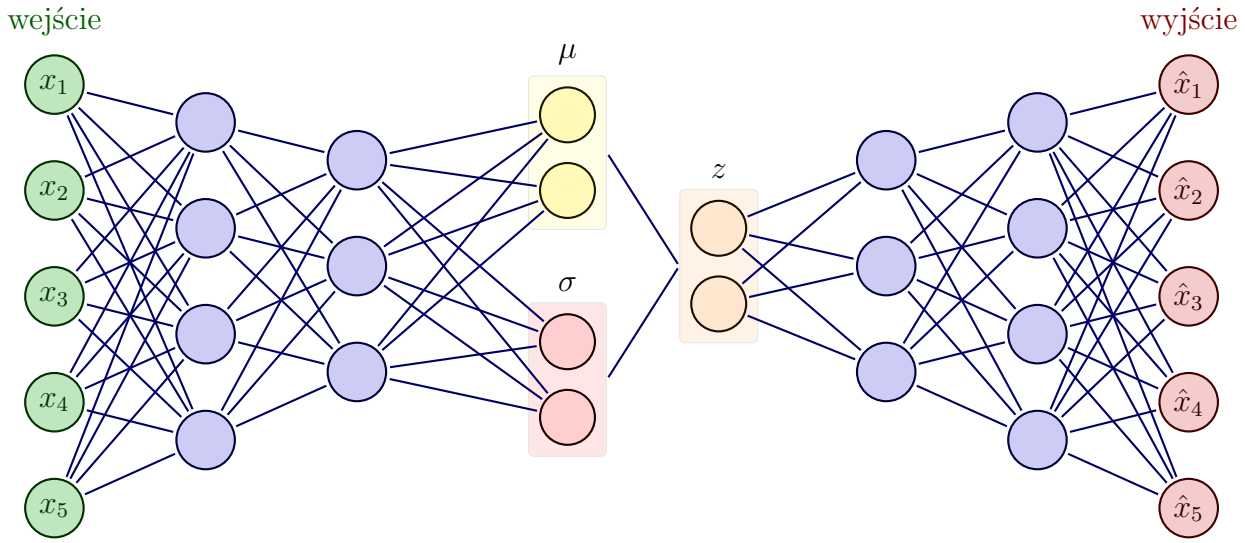
$$\frac{1}{2}\left\{\left(\frac{\sigma_0}{\sigma_1}\right)^2 + \frac{(\mu_1 - \mu_0)^2}{\sigma_1^2} - 1 + 2\log\frac{\sigma_1}{\sigma_0}\right\}$$

Co w naszym przypadku gdzie $\mu_1 = 0$ oraz $\sigma_1 = 1$ uprości się do:

$$\frac{1}{2}\sum_{m}^{i=1} \sigma_i^2 + \mu_i^2 - 2\log(\sigma_i) - 1$$

Jest to pierwsza część naszej funkcji straty.

2.4 Budowa tmp

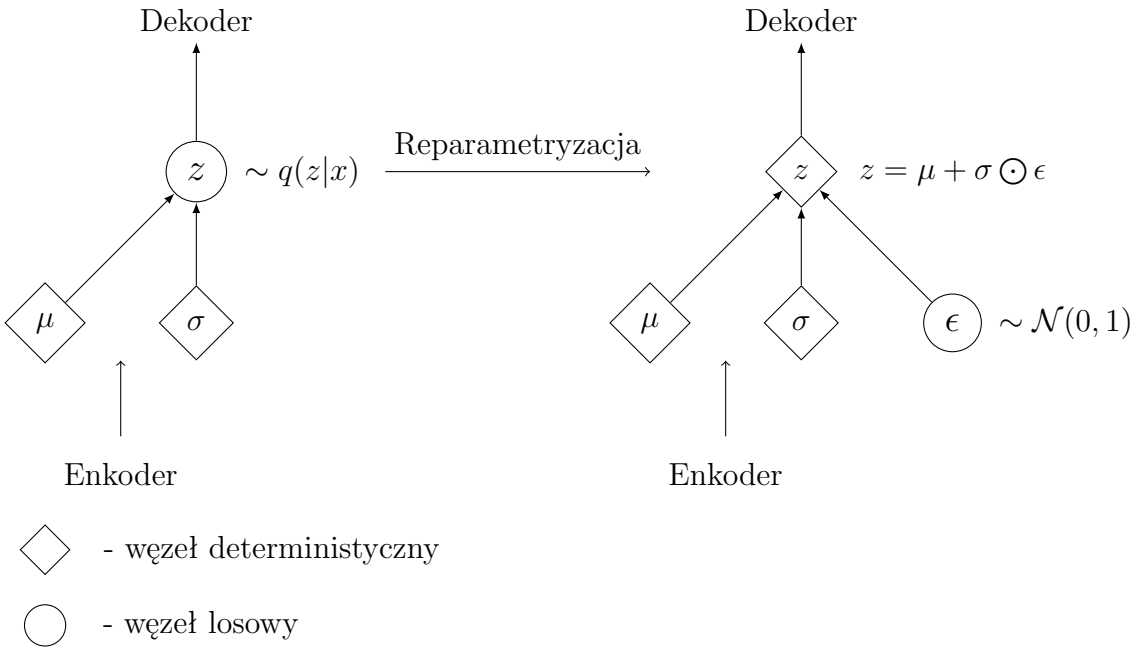


2.5 Sztuczka reparametryzacyjna

Model *VAE* po zakodowaniu wejścia dokonuje operacji próbkowania (*sampling*) z dystrybucji na nauczonych parametrach. Przy propagacji do przodu nie jest to problem, jednak podczas propagacji wstecznej jest to nie możliwe. Operacja próbkowania nie jest różniczkowalna co sprawia, że nie możemy policzyć gradientu po którym będziemy schodzić. Sposobem obejścia tego problemu jest zastosowanie sztuczki (*reparameterization trick*). Próbkowanie z dystrybucji $z \sim \mathcal{N}(\mu, \sigma)$ jesteśmy w stanie zapisać jako:

$$\epsilon \sim \mathcal{N}(0, 1)$$
$$z = \mu + \sigma \bigodot \epsilon$$

Pozornie nic się nie zmieniło, jednak teraz jesteśmy w stanie poprowadzić gradient przez z, które jest teraz deterministycznie. W poprzednim przypadku było ono losowe wybierane z dystrybucji.



Rysunek 2.3: Graficzna reprezentacja reparametryzacji.

Rozdział 3

Implementacja

3.1 Tensorflow oraz Keras

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.[7]

Spis tabel

Spis rysunków

Bibliografia

[1] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” 2014.

[2] D. Bank, N. Koenigstein, and R. Giryes, “Autoencoders,” 2021.

[3] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010.

[4] M. Telgarsky, “Benefits of depth in neural networks,” 2016.

[5] R. Eldan and O. Shamir, “The power of depth for feedforward neural networks,” 2016.

[6] J. Gramack and A. Gramacki, “Wybrane metody redukcji wymiarowości danych oraz ich wizualizacji,” 2008.

[7] C. Doersch, “Tutorial on variational autoencoders,” 2021.