



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	5
1 Tradycyjny autoenkoder	7
1.1 Informacje wstępne	7
1.1.1 Zbiór danych MNIST	8
1.2 Budowa	9
1.3 Zastosowania	10
1.3.1 Redukcja wymiarów	10
1.3.2 Odszumianie obrazów	10
1.3.3 Uzupełnianie obrazów	11
1.4 Problemy z generacją nowych danych	11
2 Wariacyjny autoenkoder	13
2.1 Informacje ogólne	13
2.2 Motywacja statystyczna	13
2.3 Wnioskowanie wariacyjne	14
2.3.1 Dywergencji Kullbacka-Leiblera	14
2.4 Sztuczka reparametryzacyjna	16
2.5 Budowa tmp	17
3 Implementacja	19
3.1 Tensorflow oraz Keras	19
Spis tabel	21
Spis rysunków	23

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 dostaje te same dane. Powiedzmy że mamy dane wejściowe X o wymiarze m oraz chcemy je zakodować do wymiaru n . Formalnie możemy zapisać:

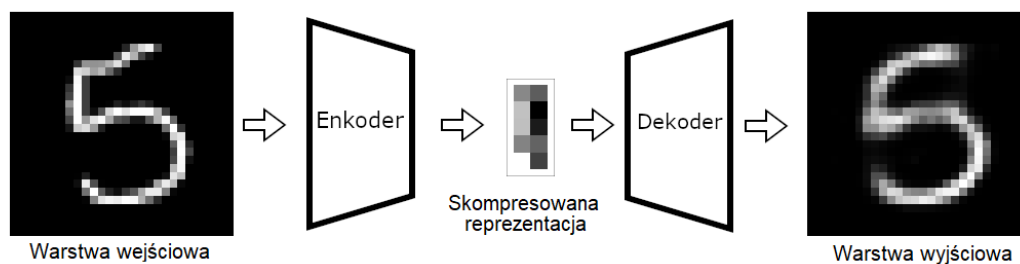
$$\text{Enkoder } E : \mathbb{R}^m \rightarrow \mathbb{R}^n$$

$$\text{Dekoder } D : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$\text{gdzie } n < m$$

Celem *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.

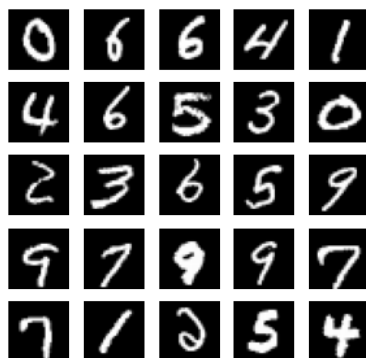
$$\mathcal{L}(x, \hat{x}) = \frac{1}{m} \sum_{i=0}^m (x_i - \hat{x}_i)^2 = \frac{1}{m} \sum_{i=0}^m (x_i - D(E(x_i)))^2$$



Rysunek 1.1: Schemat budowy autoenkodera.

1.1.1 Zbiór danych MNIST

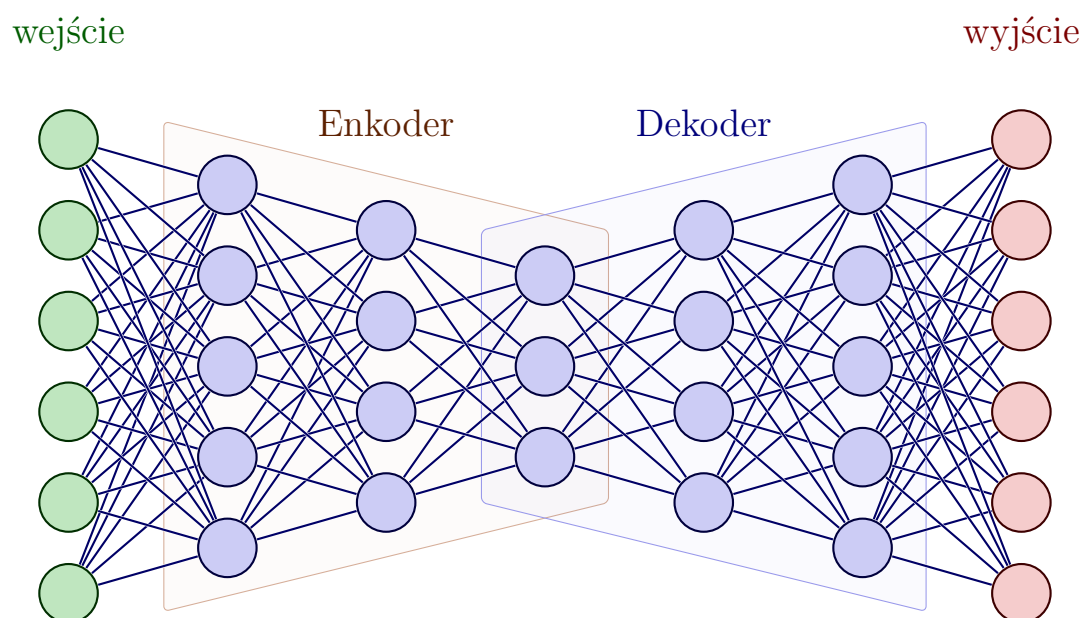
Zbiór danych MNIST (*Modified National Institute of Standards and Technology*)[3] jest zbiorem wielu odręcznie pisanych cyfr. Znajduje szerokie zastosowanie w nauce i prezentacjach możliwości modeli uczenia maszynowego. W jego skład wchodzi 60,000 obrazów przeznaczonych do treningu modeli oraz 10,000 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 dekodery są w pełni połączonymi sieciami neuronowymi.



Rysunek 1.3: Wizualizacja sieci tworzącej autoenkoder.

Obrazek przedstawia prosty autoenkoder kompresujący wejście siedmiowymiarowe do kodu o długości trzy. Enkoder jak i dekodery mają po dwie warstwy ukryte. Odbicie lustrzane architektury nie jest konieczne aby model działał poprawnie, jednak zazwyczaj taką architekturę się używa.

Hyperparametry 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 "oszukiwał" i zostanie 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

odzworowanie przez dekodery. Z drugiej strony używając, dłuższego wektora zmienionych ukrytych dostajemy gorszą kompresję danych. Długość kodu musimy dobrać w zależności od problemu, który chcemy rozwiązać używając modelu.

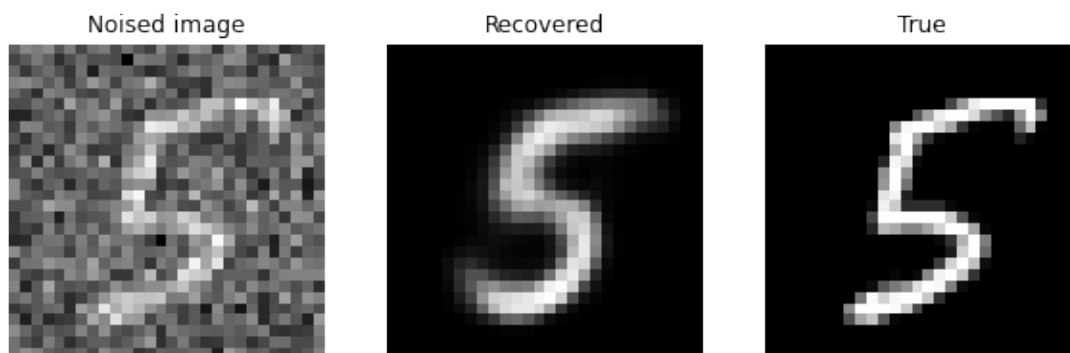
1.3 Zastosowania

1.3.1 Redukcja wymiarów

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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.

1.3.2 Odszumianie obrazów

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

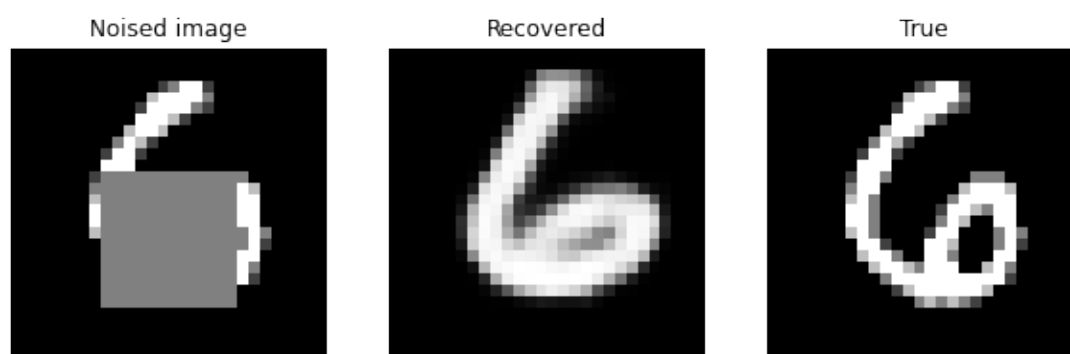


Rysunek 1.4: Grafika przedstawia porównanie obrazu z szumem, prawdziwego oraz wynikowego odcodowanego z czterech wartości.

1.3.3 Uzupełnianie obrazów

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.



Rysunek 1.5: Grafika przedstawia porównanie obrazu wejściowego, prawdziwego oraz wynikowego odcodowanego z czterech wartości.

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 odcodować 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, że punkty wzięte z dystrybucji muszą dawać wyniki mające sens

Tradycyjna architektura nie zapewnia nam a priori czy przestrzeń zmiennych ukrytych będzie spełniała te warunki. 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 by nam pasowała, ale jest to bardzo mało prawdopodobne. Jeśli chcemy zbudować model generacyjny musimy mieć zagwarantowane, że za każdym razem dostaniemy rozkład spełniający nasze warunki.

Rozdział 2

Wariacyjny autoenkoder

2.1 Informacje ogólne

Wariacyjny autoenkoder ma inne podejście do generowania zmiennych ukrytych. Zamiast generować jedną zmienną dla każdego wymiaru, generuje dwie liczby, σ oraz μ , które traktujemy jako odchylenie standardowe oraz średnią rozkładu normalnego. Dla przykładu jeśli uznamy że chcemy dane reprezentować jako siedmio-wymiarowy wektor, nasz enkoder wygeneruje dwa wektory siedmio-wymiarowe, z którego jeden będzie przechowywał wartości średniej a drugi odchylenia standardowego dla każdego z siedmiu rozkładu normalnego. Kolejną istotną zmianą jest funkcja straty, która oprócz błędu rekonstrukcji obrazu składa się z dywergencji Kullbacka-Leiblera.

2.2 Motywacja statystyczna

Powiedzmy że istnieje zmienna ukryta z , która generuje obserwację x . Mamy tylko informację o x i chcemy się dowiedzieć jakiej jest z . Aby to zrobić 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 zwyczajnie kombinatorycznie za duża aby korzystać z takich metod jak próbkowanie Monte Carlo łańcuchami Markowa.

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.

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

gdzie Q to rodzina prostych dystrybucji, na przykład rozkładu Gaussa

Policzmy:

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

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 $\mathcal{N}(0, \mathbf{I})$. 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:

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

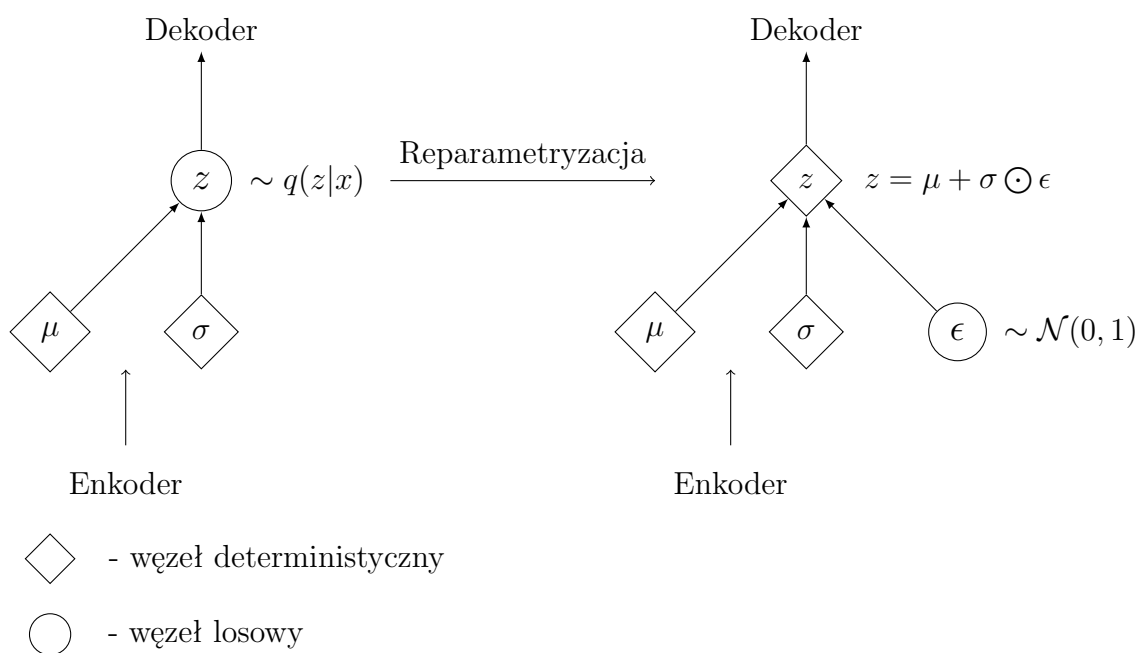
Jest to pierwsza część naszej funkcji straty.

2.4 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:

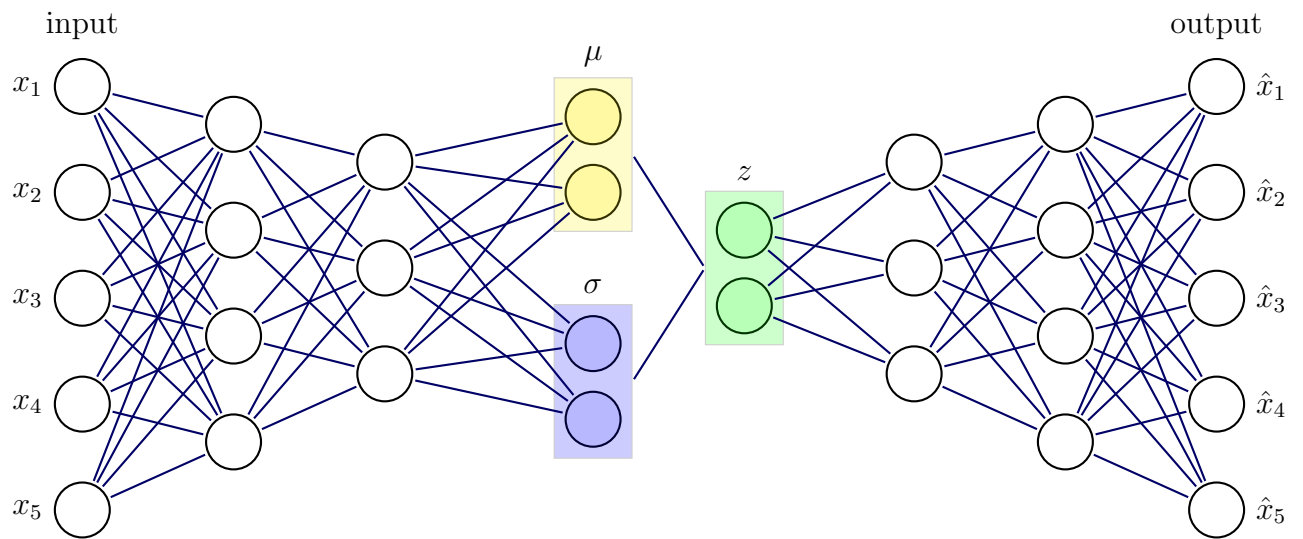
$$\begin{aligned}\epsilon &\sim \mathcal{N}(0, 1) \\ z &= \mu + \sigma \odot \epsilon\end{aligned}$$

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 2.1: Graficzna reprezentacja reparametryzacji.

2.5 Budowa tmp



Rozdział 3

Implementacja

3.1 Tensorflow oraz Keras

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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.[6]

Spis tabel

Spis rysunków

1.1	Schemat budowy autoenkodera.	8
1.2	Przykładowe obrazy ze zbioru danych.	8
1.3	Wizualizacja sieci tworzącej autoenkoder.	9
1.4	Grafika przedstawia porównanie obrazu z szumem, prawdziwego oraz wynikowego odkodowanego z czterech wartości.	10
1.5	Grafika przedstawia porównanie obrazu wejściowego, prawdziwego oraz wynikowego odkodowanego z czterech wartości.	11
2.1	Graficzna reprezentacja reparametryzacji.	16

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