Raport z zadania 5. - DDPM Diffussion Models

Michał Stefanik

28 listopada 2023

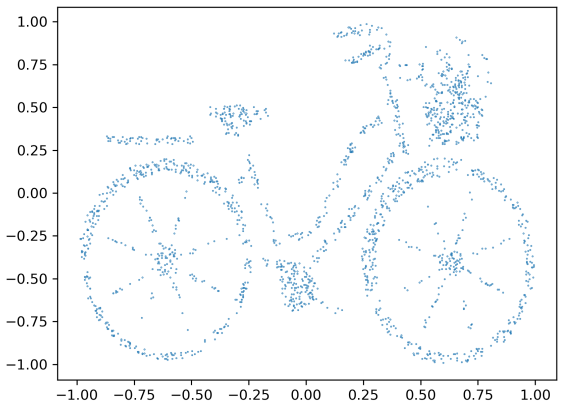
1 Wstęp

Do wykonania zadania używałem języka Python 3.10.13 z biblioteką PyTorch 2.1.0.

2 Zadanie

Należało zaimplementować model dyfuzyjny działający na wektorach dwuwymiarowych. Zadaniem modelu, jest przesuwanie wektora o wartościach z rozkładu normalnego standardowego w takim kierunku by pokrywał się ze zbiorem danych wejściowych.

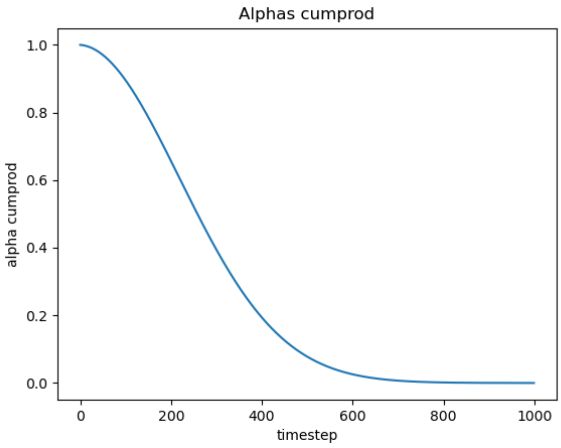
Jako dane wejściowe został użyty zbiór punktów z płaszczyzny, reprezentowanych na rysunku 1 przez niebieskie kropki.



Rysunek 1: Model dyfuzyjny

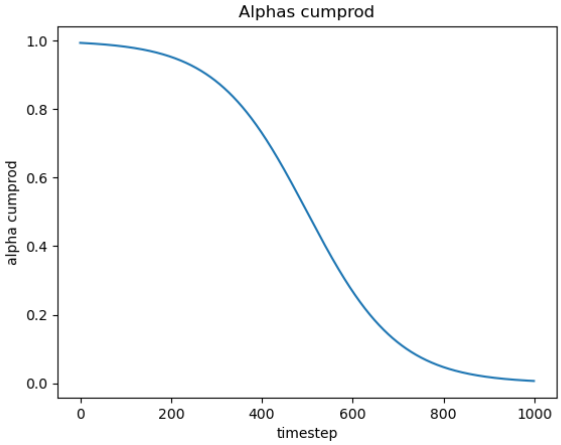
3 Dyfuzja

Pierwotny schemat wartości β był zdefiniowany jako funkcja liniowa rosnąca od wartości $\beta_1 = 0.0001$ do $\beta_T = 0.02$ w $T = 1000$ krokach. Wartości α zostały wyliczone ze wzoru $\alpha_t = 1 - \beta_t$. Dalej potrzebne były wartości $\overline{\alpha_t} = \prod_{i=1}^{t} \alpha_i$. Wykres wartości $\overline{\alpha_t}$ został pokazany na rysunku 2.



Rysunek 2: Wartości $\overline{\alpha_t}$

Wykres ten zbyt wcześnie osiąga wartość 0, co powoduje, że uczenie jest mniej płynne. Żeby temu zapobiec, wartości $\overline{\alpha_t}$ zostały wzięte z wartości przeskalowanej funkcji sigmoidalnej z przedziału $[-5, 5]$. Na jej podstawie zostały wyliczone wartości α_t i β_t.



Rysunek 3: Wartości $\overline{\alpha_t}$

4 Sprawdzenie poprawności modelu

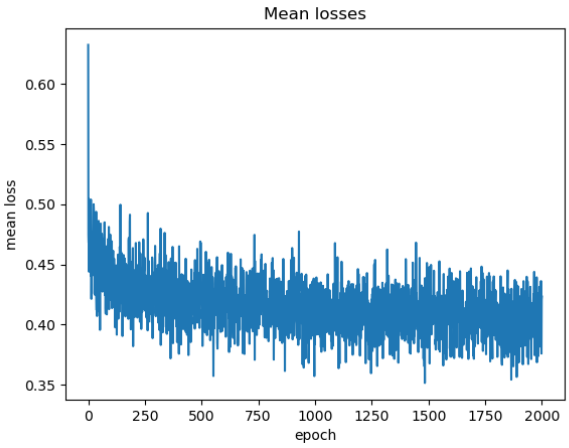
Jedyne sprawdzenie poprawności modelu, to sprawdzenie czy dane wyjściowe modelu są takiego samego rozmiaru jak dane wejściowe. W tym celu użyłem poniższego kodu:

```
model = DiffussionGenerator(0.0001, 0.02, timesteps=1000)
random_input = torch.randn(10, 2)
random_cond = torch.randint(0, 1000, (1,))
result = model(random_input, random_cond)

assert result.shape == random_input.shape
```

5 Trening modelu

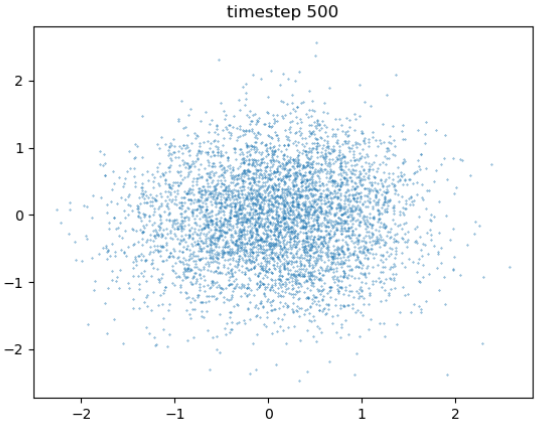
Podczas 2000 epok treningu modelu, funkcja kosztu spadała niewiele, co widać na rysunku 4.



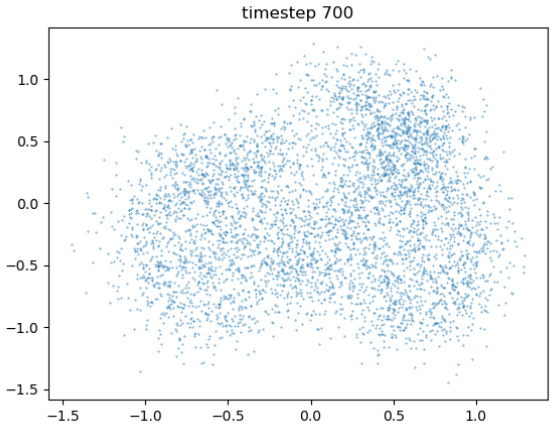
Rysunek 4: Wartości funkcji straty

6 Przykładowe Odszumianie 5000 punktów

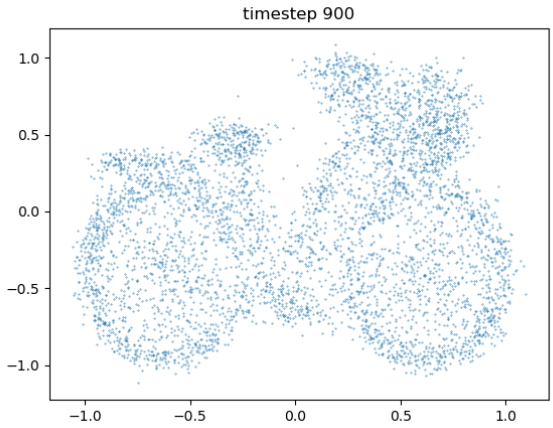
W tych wynikach użyty model został wytrenowany na 2000 epokach.



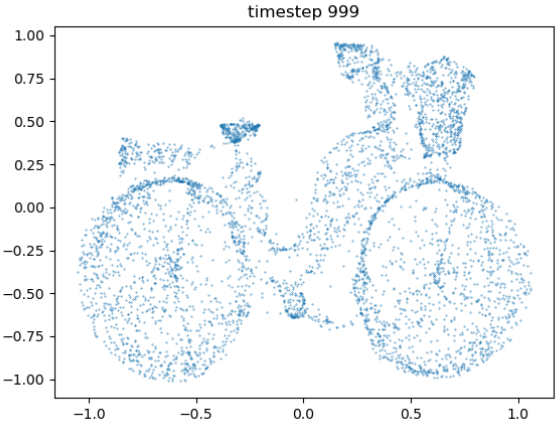
Rysunek 5: Odszumianie 5000 punktów po 500 krokach



Rysunek 6: Odszumianie 5000 punktów po 700 krokach

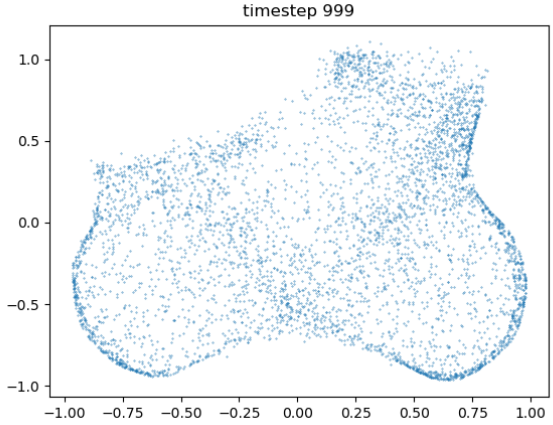


Rysunek 7: Odszumianie 5000 punktów po 900 krokach

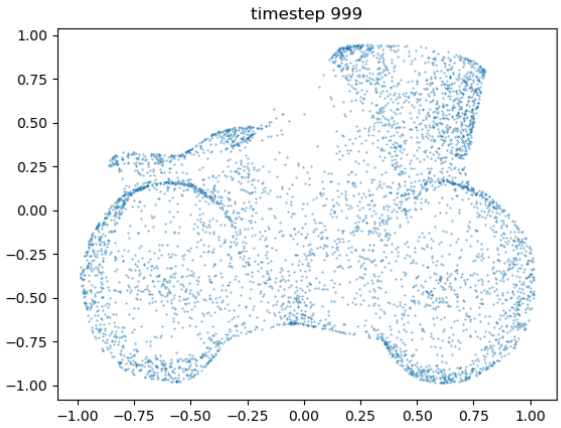


Rysunek 8: Odszumianie 5000 punktów po 999 krokach

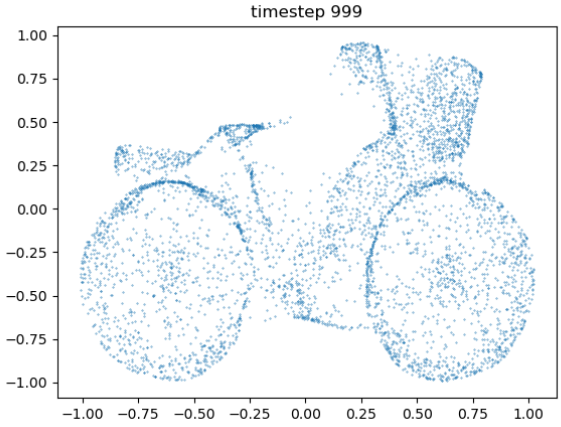
7 Porównanie z wariantami modelu zapisanymi podczas treningu



Rysunek 9: Wynik generacji 5000 punktów na modelu trenowanym przez 200 epok



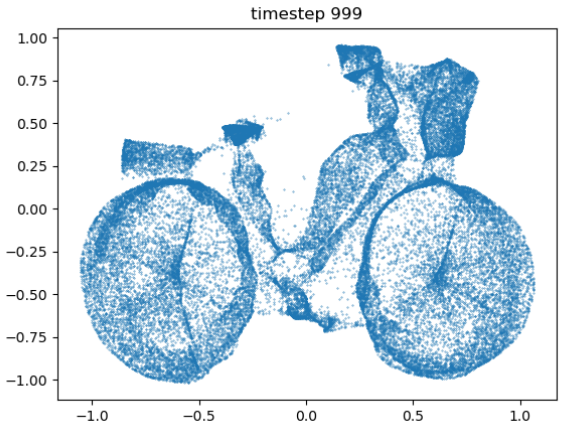
Rysunek 10: Wynik generacji 5000 punktów na modelu trenowanym przez 500 epok



Rysunek 11: Wynik generacji 5000 punktów na modelu trenowanym przez 1000 epok

Jak widać, funkcja loss, nie spadła zbyt mocno, a pomimo to jakość generacji znacząco się poprawiła. Być może dane zaszumiane były na zbyt mało sposobów, by model mógł się nauczyć ich odszumiać. Generowany szum był za każdym razem inny, więc model powoli się uczył, jak go odszumiać. Dodatkowo, może pomogłaby sieć o większej pojemności, która mogłaby się nauczyć bardziej skomplikowanych zależności.

8 Generacja 40000 punktów



Rysunek 12: Wynik generacji 40000 punktów na modelu trenowanym przez 2000 epok