Zadanie 05

I want to ride my bicycle!

Naszym głównym zadaniem będzie zaimplementowanie klasycznego generatywnego modelu dyfuzyjnego znanego jako DDPM (*Denoising Diffusion Probabilistic Models*). By nie przytłoczyła nas skala koniecznego treningu, tym razem ograniczymy się do mniejszego, sztucznego problemu. Nasz zbiór treningowy to 50 tysięcy obserwacji, każda z nich w formie dwuwymiarowego wektora. Obserwacje zapisane są w dołączonym do zadania pliku `bicycle.txt`, a ich rozkład na płaszczyźnie 2D układa się w charakterystyczny kształt. Mimo to, postaramy się by reszta parametrów (np. ilość kroków dyfuzji) modelowała bardziej realistyczny scenariusz generowania prawdziwych obrazów.

W wykonaniu zadania bardzo pomocny może byc szczegółowy tutorial omawiający metody dyfuzyjne autorstwa Lilian Weng: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/. Jako inspiracja dla ogólnego kształtu implementacji użyteczny może być też ten tutorial kerasowy: https://keras.io/examples/generative/ddpm/.

- W przypadku punktów oznaczonych ikoną ⌨️ poinformuj w jaki sposób je zrealizowano - wspomnij kluczowe klasy/metody/funkcje lub załącz powiązany fragment kodu źródłowego.
- W przypadku punktów oznaczonych ikoną 🖼️ załącz w raporcie obraz przedstawiający stan kanału będącego wynikiem danej operacji.

I want to ride my bike!

1. Zacznijmy od wczytania naszego zbioru danych. Skorzystaj z takiego narzędzia, które potem umożliwi ci wygodny trening w twoim ulubionym frameworku. Narysuj na płaszczyźnie ich losowy pozdbiór, upewniając się, że układają się w odpowiedni rozkład. [🖼️]
2. Następnie przetestujmy dyfuzję "w przód".
    1. Proces ten opisuje poniższe równanie.

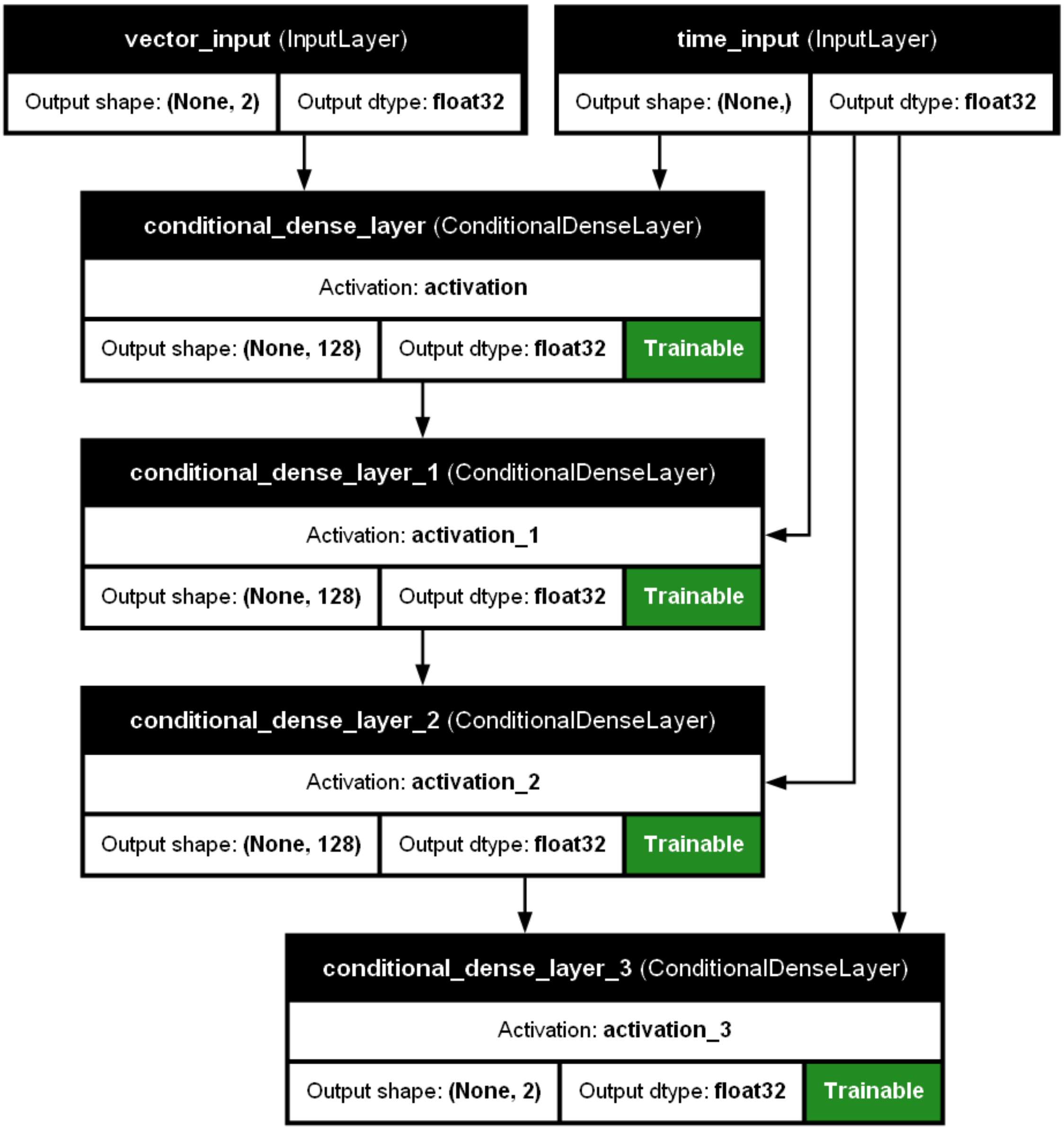
    $$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

    2. Przyjmijmy, że dyfuzja będzie skutkiem wykonania $T = 1000$ kroków.
    3. Musimy również dobrać sekwencję parametrów opisującą wariancję β_t każdego z kroków.
        1. Na początek przyjmij, że będą rosnąć liniowo od $\beta_1 = 0.0001$ do $\beta_T = 0.02$.
        2. W oparciu o nie oblicz wartości pomocnicze $\alpha_t = 1 - \beta_t$ oraz $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$.
        3. Prześledź, jak zmienia się $\bar{\alpha}_t$ wraz z kolejnymi krokami t. [🖼️]
        4. By zapewnić większą płynność procesu odszumiania zastąp tę sekwencją $\bar{\alpha}_t$ inną, opartą o dowolną funkcję sigmoidalną ($\bar{\alpha}_t$ powinno stopniowo zanikać od 1 do 0, z wartością 0.5 dla $t = T/2$).
        5. Mając nową sekwencję $\bar{\alpha}_t$ oblicz zależne od niej α_t i β_t - będą nam potrzebne w kolejnych krokach.
    4. Zwróć uwagę, że dzięki trikowi z reparametryzacją efekt t kroków dyfuzji można obliczyć korzystajac ze wzoru poniżej.

    $$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

5. Przetestuj implementację, sprawdź jak stopniowo zmienia się położenie początkowych przykładó ze zbioru danych wraz z kolejnymi etapami dyfuzji.
6. Postaraj się, by implementacja była możliwie efektywna i potrafiła działać na całych batchach, unikając niepotrzebnych pętli.

3. Teraz zajmiemy się zdefiniowaniem modelu predykcyjnego. Powinien mieć następującą wysokopoziomową strukturę:



1. `conditional_dense_layer` to blok, który:
    1. przekształca liniowo wejściowy wektor danych `x` (przekształcenie zależne jest od wyuczalnych parametrów) w nowy wektor o rozmiarze 128,
    2. zamienia wejściowy krok dyfuzji `t` na reprezentację wektorową będącą efektem bloku `learnable_sinusoidal_embedding` (również o rozmiarze 128),
    3. sumuje uzyskane reprezentacje `x` i `t`,
    4. wykonuje na rezultacie aktywację ReLU (za wyjątkiem ostatniej, czwartej warstwy - tam nie wykorzystujemy na koniec żadnej aktywacji).
2. `learnable_sinusoidal_embedding` to blok, który:
    1. wykorzystuje sinusoidalne kodowanie pozycji do zamiany kroku `t` na reprezentujący go wektor cech o rozmiarze 50,
    2. przekształca ten wektor z wykorzystaniem warstwy gęsto połączonej o rozmiarze 128 i aktywacji ReLU,

        3. wykonuje jeszcze jedno przekształcenie liniowe w nowy wektor o rozmiarze 128 (tym razem bez aktywacji) i zwraca jego rezultat.
    3. Dodatkowo zakładamy, że będziemy operowali na batchach składających się z 64 przykładów i wykorzystywali optymalizator Adam z *learning rate* równym 0.0001.
    4. Sprawdź, czy (niewytrenowany) model przyjmuje i zwraca dane w odpowiednim formacie. [ ]
4. Mając przygotowany zbiór danych, narzędzia do zaszumiania go, oraz model predykcyjny, pora rozpocząć trening, zgodnie z algorytmem DDPM.

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

    1. Pamiętaj by próbkować x_0 z pierwotnego zbioru danych, t z równą szansą z całego zakresu, a ϵ z odpowiednią wariancją.
    2. Proces powinien przebiegać stabilnie i bez większych zawirowań. Uzyskanie wyników wysokiej jakości może zająć do 1000 kroków, ale już po kilkudziesięciu krokach treningu procedura generacji powinna zacząć dawać rozsądne rezultaty.
    3. Wytrenuj model, zapisz kilka jego wariantów w trakcie treningu, obserwuj zmiany funkcji straty. [ ]
5. Jeżeli udało się zakończyć trening, to teraz pora przetestować procedurę generacji (korzystając z drugiego z algorytmów powyżej).
    1. Wygeneruj zbiór losowych punktów w oparciu o rozkład normalny.
    2. Rozpocznij procedurę "odszumiania" tak uzyskanych próbek dla kolejnych t wykorzystując wytrenowany - jak zmienia się ich rozkład (możesz nawet pokusić się o animację)? [ ]
        1. Uwaga - na potrzeby tego procesu zakładamy, że wspomniane w algorytmie $\sigma_t = \sqrt{\beta_t}$.
    3. Sprawdź finalny rezultat generacji dla wariantów modelu zapisanych podczas wcześniejszych epok - czy spadek jakości jest proporcjonalny do różnic w funkcji loss? Dlaczego? [ ]