WSI zad. 5@e23f8dc087ae7d0a8c1fe515df65ac5580b354f1 **Grzegorz Rypeść**

Proszę podzielić się w pary. W tym zadaniu kontynuujemy temat aproksymowania zachowania człowieka w grze *Snake* przy użyciu klasyfikatora. Należy stworzyć wielowarstowy perceptron, który pokieruje wężem korzystając z biblioteki PyTorch. Dane można wziąć z zadania 4., dogenerować więcej, lub wygenerować je od nowa. Należy je podzielić na zbiór treningowy, walidujący i testowy w proporcji 8:1:1. Atrybuty można zmienić, ale niech nadal będzie ich co najmniej 8. Proszę opisać je w raporcie.

Należy zaimplementować następujące moduły:

- BCDataset - Klasa dziedzicząca po *torch.utils.data.Dataset*, należy nadpisać jej metody __init__ , __get_item__, __len__ https://pytorch.org/docs/stable/data.html#map-style-datasets. Ma ona za zadanie zaopatrywać ładowarkę danych (ang. DataLoader) w przykłady danych.
- MLP (ang. Multi Layer Perceptron) - Perceptron wielowarstowy, pochodna *torch.nn.Module*. Klasyfikator, który dla wektora opisującego stan gry zwróci kierunek/akcję węża: góra, prawo, dół, bądź lewo.
- Metryki - Należy raportować wartości funkcji kosztu oraz dokładność klasyfikacji na zbiorze treningowym oraz walidującym dla każdej epoki. Można użyć bibliotek: *logging*, T*ensorboard, torch.metrics*.
- Skrypt trenujący - Powinien dokonywać wsadowego trenowania modelu (batch size > 1) korzystając z ładowarek danych i integrować powyższe moduły. Jako optymalizator należy wykorzystać optymalizator SGD. Skrypt powinien generować plik zawierający wagi modelu (tzw. *state dict*) z najlepszej epoki, tj. epoki, na której model uzyskał najwyższą dokładność na zbiorze walidującym.
- Skrypt testujący - Skrypt, który oblicza metryki wytrenowanego modelu na zbiorze testowym.
- MLPAgent - Agent analogiczny do klasy *HumanAgent*, który wykorzystuje wytrenowany perceptron do sterowania wężem. Wagi do sieci musi wczytywać z pliku generowanego przez skrypt trenujący.

MLP składa się z warstwy wyjściowej oraz warstw ukrytych z funkcjami aktywacji. Proszę wytrenować MLP dla następujących funkcji aktywacji: {tożsamościowa, ReLU, LeakyReLU z odchyleniem 0.01 i jedna dowolna} oraz liczb warstw ukrytych: {1, 2, 5, 30}. Proszę opisać zaobserwowane rezultaty i wyciągnąć wnioski. Dla liczby warstw ukrytych 30 i ReLU proszę dla macierzy wag każdej warstwy zaraportować średnią normę (ang. matrix norm) gradientów w czasie pierwszej epoki trenowania. Rezultatem będzie 31 skalarów. Co można zaobserwować i z czego to wynika?

Następnie dla 1 warstwy ukrytej oraz funkcji aktywacji ReLU proszę zbadać wpływ liczby neuronów w warstwie ukrytej na wyniki. Czy pojawiło się niedouczenie lub przeuczenie? Dlaczego? Jak temu zaradzić? Proszę zastosować środki zapobiegawcze przeuczeniu, takie jak odrzucanie oraz regularyzacja L2. Jak wpłynęły one na wyniki?

Proszę wybrać najlepszy model i przetestować go na zbiorze testowym. Wnioski. Proszę zawrzeć log z trenowania tego modelu w raporcie. Następnie proszę stworzyć klasę MLPAgent wyposażając ją w najlepszy model i przetestować w grze Snake w 100 rozgrywkach. Proszę porównać wynik z wynikiem z zadania 3 i opisać wnioski.

Raport oraz pliki proszę spakować do pliku o nazwie WSI-5-NAZWISKO-IMIE.zip i przesłać na adres grzegorz.rypesc.dokt@pw.edu.pl.

Wskazówki:
Pamiętać o stosowaniu *model.train()*, *model.eval()*, *with torch.no_grad()*, będę pytał co te metody robią.

torch.nn.CrossEntropyLoss dokonuje funkcji softmax sam w sobie. Jest to funkcja kosztu, z której warto w zadaniu skorzystać.

Pierwszą osią w tensorach **musi** być zawsze i wszędzie rozmiar wsadu. Ładowarka danych powinna zwracać cały wsad danych, np. tensor o kształcie [64x10] i przykłady danych we wsadzie powinny być równolegle przetwarzane na potrzeby uczenia. 64 to tutaj rozmiar wsadu, a 10 to liczba atrybutów. Przetworzenie jednego wsadu w czasie treningu nazywane jest iteracją.

Ładowarka danych musi połączyć przykłady danych zwracane przez funkcję *__get_item__* w jeden wsad. Korzysta do tego z funkcji *collate_fn*.

Do trenowania można wykorzystać procesor graficzny.

Na czas implementacji warto ograniczyć sobie zbiór danych i atrybutów do absolutnego minimum, które jest łatwe w interpretacji. Po napisaniu kodu warto sprawdzić, czy sieć jest w stanie się przeuczyć na jednym wsadzie danych. Jeżeli nie jest w stanie tego zrobić to znaczy, że implementacja zawiera błąd, bądź wsad zawiera sprzeczne ze sobą dane.

Parametr *momentum* w optymalizatorze SGD można przestawić na 0.9 i sprawdzić co się stanie.