MRO - zadanie 1: Bliskie spotkania trzeciego stopnia z Pytorch'em

Piotr Zawiślan

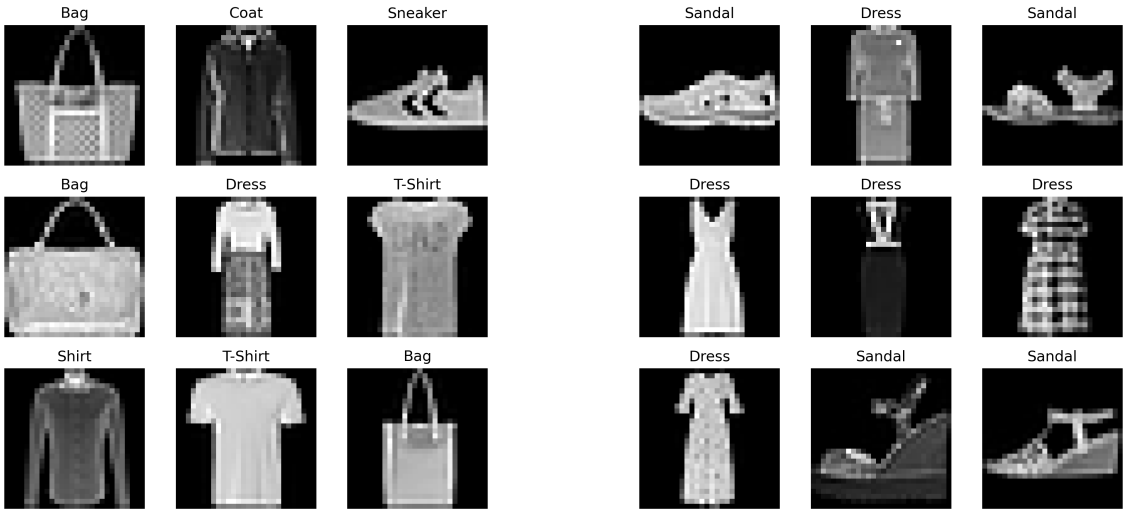
17 Październik 2022

1 Wykorzystane technologie

Kod pisałem w języku Python z wykorzystaniem frameworku Pytorch. Jeżeli chodzi o motywację mojego wyboru - wybrałem Pytorch'a ze względu na opinie mówiące, że jest bardziej "pythoniczny"od Tensorflow, aczkolwiek zamierzam też kiedyś napisać coś w tym drugim :).

2 Zbiór danych

Do pozyskania zbioru danych posłużyłem się wbudowaną funkcją z modułu *torchvision*. Następnie wybrałem interesujące nas 2 klasy (zgodnie z zaleceniami: sukienki - 3 i sandały - 5) i zamieniłem etykiety na 0 i 1. W wykonaniu tych zadań pomogły mi tablice 'numpy'.
Ponieważ w tutorialu Pytorch'a była mowa o obiekcie Dataset i Dataloader, zaimplementowałem własny Dataset, który jednak nie wczytywał danych z pliku, ale po prostu z pamięci (jako, że ograniczony zbiór danych nie zajmował jej zbyt dużo). Przy operacji $\_\_getitem\_\_$ użyłem też transformacji *ToTensor()*, która normalizuje tablicę zawierającą obraz.



Obrazy przed transformacją. Obrazy po transformacji.

Rysunek 1: Losowe elementy zbioru przed i po ogarniczeniem go do 2 klas.

3 Model decyzyjny

3.1 Konstrukcja modelów w Pytorch'u

Budowa modelów uczenia maszynowego (a tak naprawdę sieci neuronowych) w Pytorch'u odbywa się poprzed stworzenie metod *__init__* oraz *forward* w klasie dziedziczącej po *torch.nn.module*.

3.2 Definicja modelu regresji logistycznej

```
class LogRegNN(nn.Module):
    def __init__(self, in_features=28*28, out_features=1):
        super(LogRegNN, self).__init__()
        self.stack = nn.Sequential(
            nn.Flatten(),
            nn.Linear(in_features, out_features),
            nn.Sigmoid()
        )

    def forward(self, x):
        probs = self.stack(x)
        return probs
```

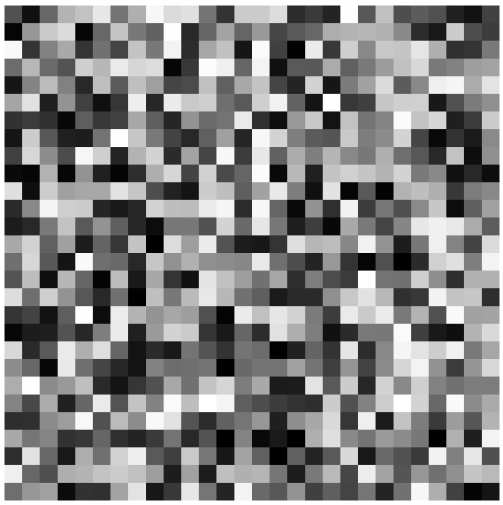
Rysunek 2: Moja definicja modelu regresji logistycznej.

Powyższy zrzut ekranu przedstawia moją implementację regresji logistycznej. Normalizacją i zamianą etykiet klas zajmuje się już *Dataset* i *Dataloader*, tak więc wewnątrz modelu znajduje się reszta potrzebnych "klocków":

- Funkcja "spłaszczająca"obraz $((28, 28) \implies (28 * 28))$
- Warstwa gęsta/liniowa $((28 * 28) \implies (1))$
- Funkcja logistyczna, aby uzyskać prawdopodobieństwo z logitów $((1) \implies (1))$

Jeśli chodzi o debugowanie "na żywo", modelu można używać jak funkcji, więc testowałem jego zachowanie na przykładowych danych. Nie używałem narzędzia podobnego do *Tensorboard*, ponieważ nie jestem pewnien, czy w ogóle taki istnieje w Pytorchu :).

3.3 Początkowe wagi funkcji liniowej



Rysunek 3: Początkowe wagi przekształcone do obrazu 28×28.

Nie mają większego sensu - są inicjalizowane losowo.

4 Uczenie i testowanie

4.1 Funkcja kosztu oraz optimizer

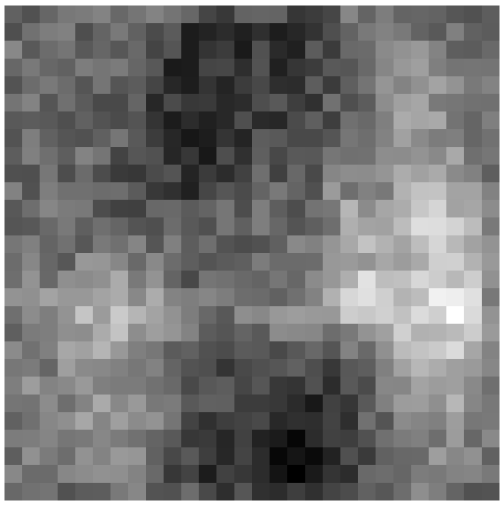
Jako funkcji kosztu przyjąłem *torch.nn.BCELoss* (binarna entropia krzyżowa), a jako mechanizmu optymalizacji *torch.optim.SGD* (stochastyczny spadek po gradiencie).

4.2 Strojenie do danych treningowych i ewaluacja

W celu treningu utworzyłem funkcję uczącą model. Pobiera ona dane w losowych batch'ach po 32 elementy, zwraca dla każdego z nich prawdopodobieństwo przynależności do klasy 1 (sukienki), oblicza binarną entropię krzyżową pomiędzy predykcjami a etykietami, oblicza gradienty po każdej cesze (dzięki *autograd*) i w końcu wykonuje krok *SGD* (czy tak naprawdę, *Mini-Batch Gradient Descend*).
W ramach ewaluacji modelu, utworzyłem też funkcję testującą, zliczającą TP, FP, FN i FP. Na tej podstawie, obliczam dokładność, precyzję i czułość, których wartości wyświetlam po każdej epoce.
Na jedną epokę składa się wykonanie pętli treningowej oraz pętli testującej. Po 16 epokach (trengu na *GPU*), uzyskałem następujące wyniki:

- dokładność: 99.4%
- precyzja: 99.2%
- czułość: 99.5%
- średni koszt: 0.041879

4.3 Wagi funkcji liniowej po treningu



Rysunek 4: Wagi po treningu, przekształcone do obrazu 28×28.

Tutaj już da się coś zauważyć - model głównie bierze pod uwagę środkowo-górną i środkowo-dolną część obrazu. Być może, to, że sukienki są ulokowane bardziej na środku oraz to, że są dłuższe (w pionie) niż sandały sprawia, że model sprytnie wykorzystał te duże różnice w lokacjach pikselów na obrazie i na ich podstawie decyduje o tym jaka to część garderoby.
Sprawdziłem na 10 losowych przykładach, że model działa jak należy :).