SIECI NEURONOWE

Raport z laboratorium 5

Student: Van Hien Le - 257795

Narzędzia i technologie

- Python, Pandas, Numpy, Scikit-learn, PyTorch
- Jupyter Notebook

Dataset

Zbiorem danych jest kolekcja FashionMNIST, domyślnie dostępna w PyTorch.

Dostęp do zbioru danych: data = torchvision.datasets.FashionMNIST('path',download =True)

Architektura sieci

Do eksperymentów zaimplementowałem sieć jednowarstwową i sieć dwuwarstwową. Obie sieci zawierają warstwę Flatten, warstwę wejściową, po której następuje jedna lub dwie warstwy ukryte i warstwę wyjściową. Poniżej znajdują się najważniejsze elementy sieci:

Flatten layer: Warstwa przekształca dane obrazu w jednowymiarowy wektor, który można wprowadzić do warstw liniowych.

Input layer: Warstwa wejściowa ma 784 neurony, co odpowiada spłaszczonym wymiarom obrazów (28x28 pikseli).

Hidden layers: Może istnieć jedna lub dwie warstwy ukryte, każda warstwa składa się z określonej liczby neuronów. Po każdej warstwie ukrytej znajduje się funkcja aktywacji ReLU.

Output layer: Warstwa wyjściowa składa się z 10 neuronów reprezentujących dziesięć różnych klas elementów odzieży w zbiorze danych FashionMNIST. Funkcja aktywacji softmax jest używana w warstwie wyjściowej do klasyfikacji wieloklasowej.

Hiperparametry

Eksperymenty przeprowadzono z następującymi ustawieniami hiperparametrów:

Domyślne hiperparametry:

- Learning Rate: 0.001

- Number of Epochs: 30

- Optimizer: Adam

- Loss Function: Cross-Entropy loss

Hiperparametry do eksperymentowania:

- Hidden layer size: 10, 50, 200

- Batch size: 16, 100, 200

- Train dataset size: 1%, 10%

- Dataset noise: Added in test set / Added in both train and test set

Konfiguracja i procedura eksperymentów

Wstępne przetwarzanie danych

Do eksperymentów wykorzystano zbiór danych FashionMNIST. Przed trenowaniem i testowaniem zastosowałem następujące etapy przetwarzania wstępnego:

Ładowanie danych: Zbiór danych FashionMNIST został załodowany przy użyciu biblioteki DataLoader w PyTorch.

Transformacja danych: Obrazy zbioru danych zostały przekształcone w tensory i znormalizowane do zakresu [0, 1].

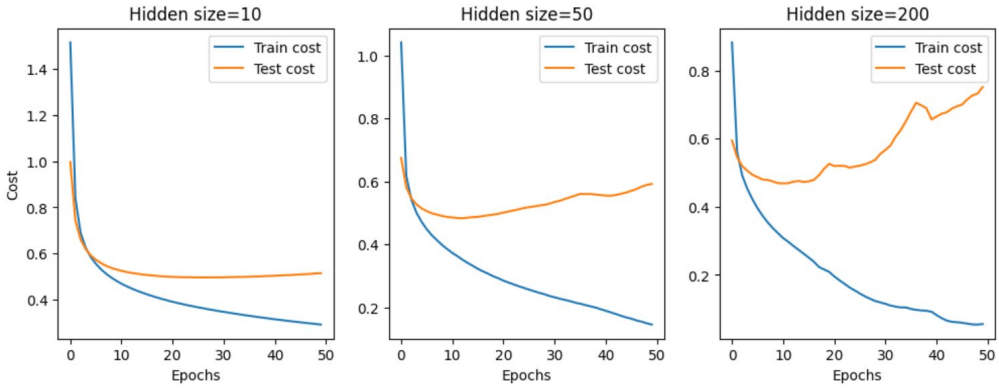
Zaburzenia danych:

Aby zbadać wpływ zaburzeń danych, do batchy danych wejściowych dodałem szum Gaussa. Szum ten został wygenerowany z różnymi odchyleniami i w niektórych eksperymentach dodany zarówno do danych uczących, jak i testowych. Celem była ocena, jak wpływa na wydajność modelu trenowanie i testowanie na zaszumionych danych.

Wyniki eksperymentów

A. Eksperymenty z jednowarstwową siecią

- Różne rozmiary warstwy ukrytej: 10, 50, 200

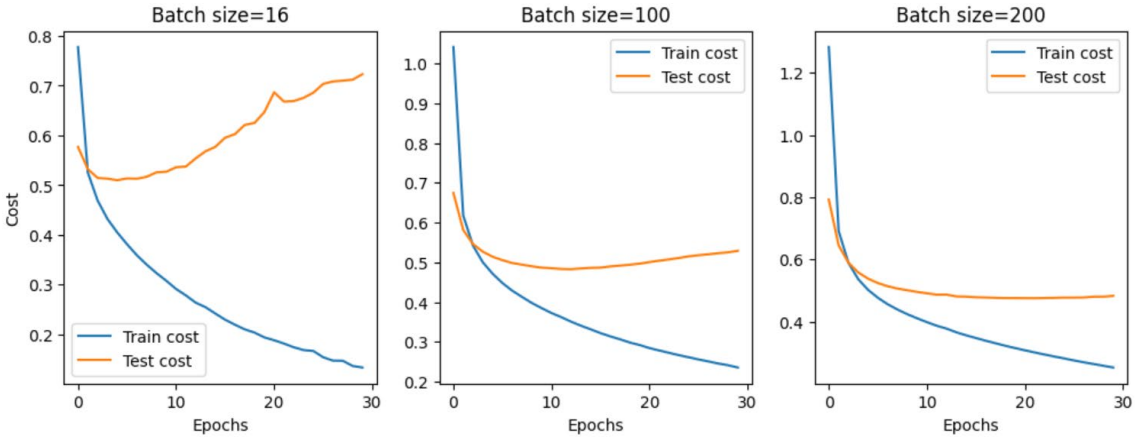


+ Metryki Accuracy:

Hidden size=10 : tensor(0.8286, device='cuda:0')
Hidden size=50 : tensor(0.8305, device='cuda:0')
Hidden size=200 : tensor(0.8416, device='cuda:0')

+ Wnioski: Zwiększanie rozmiaru warstwy ukrytej może poprawić koszt uczenia i metrykę Accuracy na danych testowych, ale może również zwiększyć ryzyko nadmiernego dopasowania.

- Różne rozmiary batchy:



+ Metryki Accuracy:

Batch size=16 : tensor(0.8250, device='cuda:0')

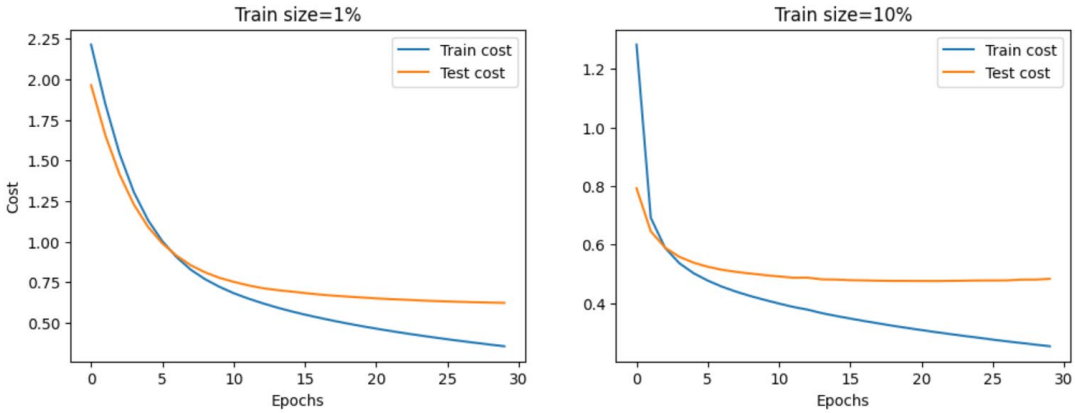
Batch size=100 : tensor(0.8302, device='cuda:0')

Batch size=200 : tensor(0.8343, device='cuda:0')

+ Wnioski: Mniejsze rozmiary batchy mogą prowadzić do szybszego uczenia się, ale mogą skutkować większą rozbieżnością w kosztach testowania i potencjalnym nadmiernym dopasowaniem. Większe rozmiary batchy mogą sprawić, że modele będą lepiej uogólniać, ale osiągnięcie zbieżności będzie wymagało większej liczby epok.

Przy mniejszych rozmiarach batchy trenowanie jest niestabilne. Z drugiej strony, większe rozmiary batchy wykazują gładsze krzywe kosztów, co wskazuje na bardziej stabilne estymacje gradientu.

- Różne ilości procentowe zbioru danych uczących:



+ Metryki Accuracy:

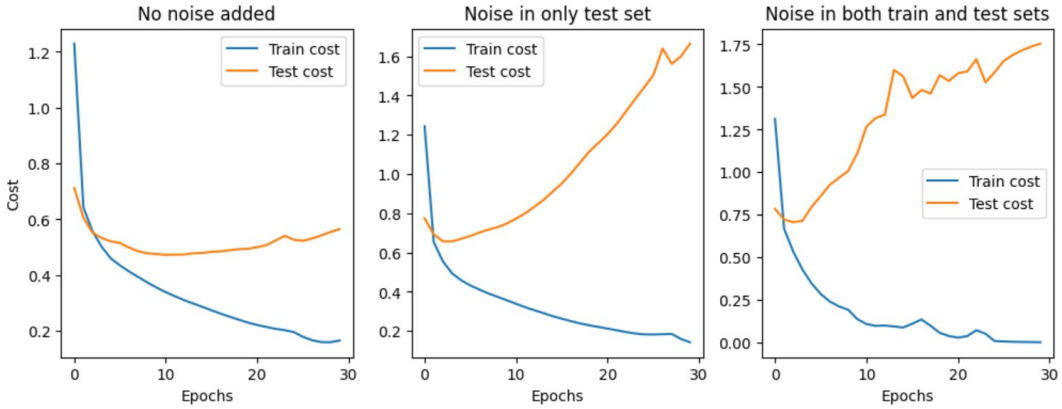
Train size=1% : tensor(0.7775, device='cuda:0')

Train size=10% : tensor(0.8343, device='cuda:0')

+ Wnioski: Zarówno w przypadku 1% danych treningowych , jak i 10% danych testowych, koszty trenownaia i testowania zaczynają się od wysokich i szybko spadają, ale koszty w pierwszym przypadku są znacznie wyższe niż koszty w drugim przypadku.

Więcej danych treningowych oznacza większą metrykę Accuracy.

- Zaburzenia danych:



+ Metryki Accuracy:

No noise added : tensor(0.8415, device='cuda:0')

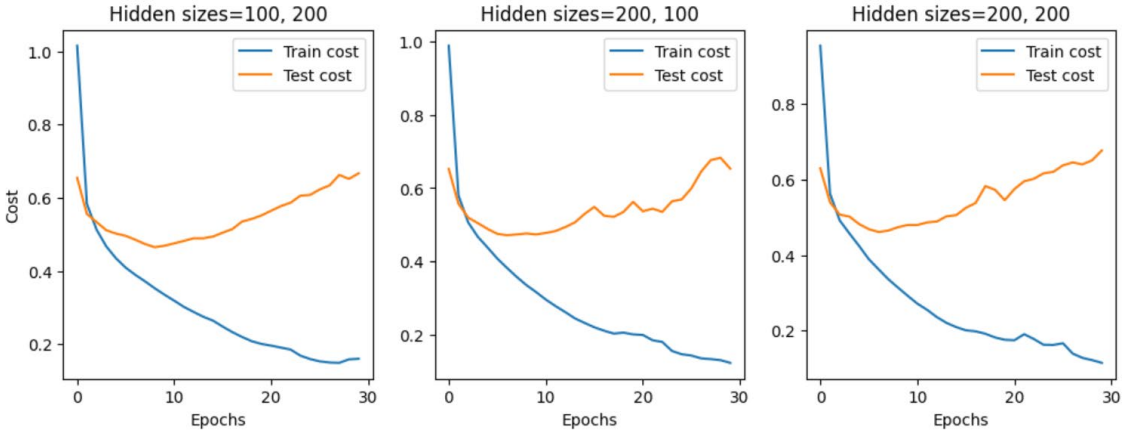
Noise in only test set : tensor(0.7575, device='cuda:0')

Noise in both train and test sets : tensor(0.7385, device='cuda:0')

+ Wnioski: Gdy nie doda się żadnego szumu, model może być zbieżny i dobrze uogólniać. Po dodaniu szumu do zbioru danych testowch, koszty testowania zaczynają znacznie rosnąć, jednak po dodaniu szumu także do zbioru danych treningowych, koszty trenowania nie rosną tak bardzo jak koszty testowania, może to sugerować, że model nauczył się ignorować szum lub znajdowanie wzorców w danych treningowych pomimo szumu.

B. Eksperymenty z dwuwarstwową siecią

- Różne rozmiary warstwy ukrytej: (100, 200), (200, 100), (200, 200)



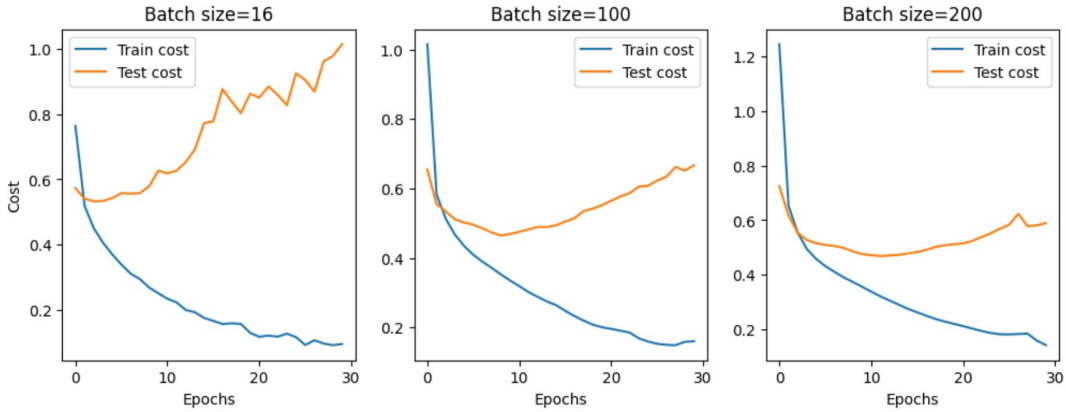
+ Metryki Accuracy:

Hidden sizes=100, 200 : tensor(0.8364, device='cuda:0')

Hidden sizes=200, 100 : tensor(0.8389, device='cuda:0')

Hidden sizes=200, 200 : tensor(0.8408, device='cuda:0')

- Różne rozmiary batchy: 16, 100, 200



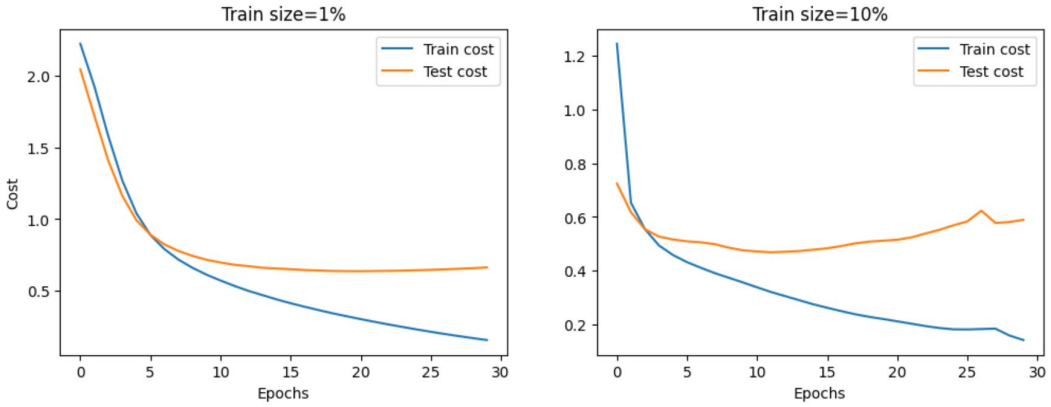
+ Metryki Accuracy:

Batch size=16 : tensor(0.8331, device='cuda:0')

Batch size=100 : tensor(0.8364, device='cuda:0')

Batch size=200 : tensor(0.8364, device='cuda:0')

- Różne ilości procentowe zbioru danych uczących:

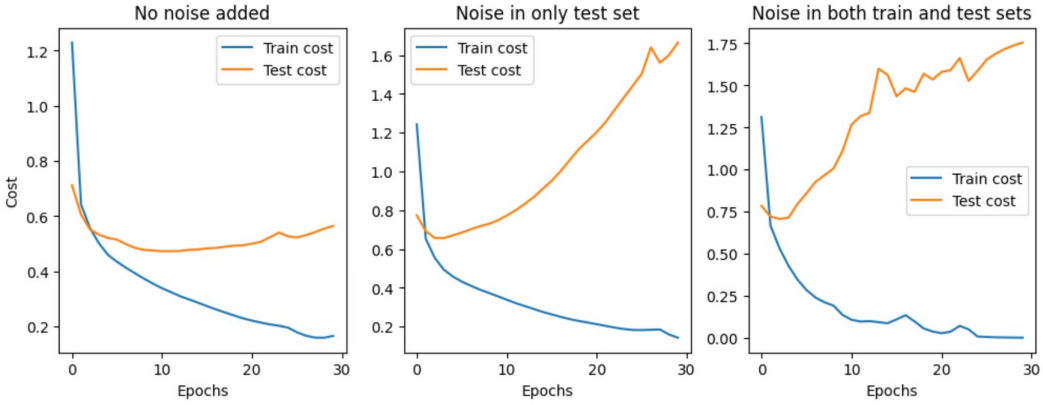


+ Metryki Accuracy:

Train size=1% : tensor(0.7775, device='cuda:0')

Train size=10% : tensor(0.8364, device='cuda:0')

- Zaburzenia danych:



+ Metryki Accuracy:

No noise added : tensor(0.8415, device='cuda:0')

Noise in only test set : tensor(0.7575, device='cuda:0')

Noise in both train and test sets : tensor(0.7385, device='cuda:0')