**Slovenská technická univerzita v Bratislave**

**Fakulta informatiky a informačných technológii**

MNIST klasifikátor

**Zadanie č. 3a**

Contents

[Popis zadania 3](#_Toc183801815)

[Reprezentácia údajov a konfigurácie 3](#_Toc183801816)

[Postup riešenia 4](#_Toc183801817)

[Porovnávanie parametrov 7](#_Toc183801818)

[Testovanie 7](#_Toc183801819)

[**Zhodnotenie** 13](#_Toc183801820)

# Popis zadania

Úlohou bolo vytvoríte neurónovú sieť na klasifikáciu ručne písaných číslic z dátového súboru MNIST. Zobrať súbor údajov zo 60 000 obrázkov na trénovanie a 10 000 obrázkov na testovania, veľkosť obrázkov je 28 x 28 a reprezentuje jednu číslicu od 0 do 9.

Použiť doprednú neurónovú sieť (viacvrstvový perceptrón) a natrénujte ju pomocou algoritmov SGD, SGD s momentom a ADAM, použiť knižnicu PyTorch. Okrem trénovacej a testovacej chyb, odmerať aj presnosť modelu (t. j. koľkokrát model predikoval správnu triedu na testovacej množine).

# Reprezentácia údajov a konfigurácie

Na trénovacie a testovacie dáta použijeme údaje z databázy MNIST.

"""Datasets"""  
train\_dataset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)  
test\_dataset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)  
  
"""DataLoaders"""  
train\_loader = DataLoader(train\_dataset, batch\_size=train\_batch\_size, shuffle=True)  
test\_loader = DataLoader(test\_dataset, batch\_size=test\_batch\_size, shuffle=False)

Veľkosť dávky (batch) je zaznamenaná v konfiguračnom súbore, ktorý vyzerá nasledovne:

[Settings]

load\_model = False

optimizer(sgd, sgd\_momentum, adam) = adam

epoch\_count = 20

train\_batch\_size = 64

test\_batch\_size = 64

show\_confusion\_matrix = False

learning\_rate = 0.001

momentum = 0.9

# Postup riešenia

Na zatáčku programu bolo importovane knižnica pre náhodne generovanie, matematická Tato cast Obsahuje všetky potrebné knižnice pre Torch a MNIST klasifikátor. Okrem toho aj knižnicu pre načrtnutie štatistických grafov, knižnicu na sledovanie času, cesty k súborom, farebné výstupy a konfiguračný súbor.

* import torch
* import torchvision
* import torchvision.transforms as transforms
* from torch.utils.data import DataLoader
* import torch.nn.functional as F
* import torch.optim as optim
* import torch.nn as nn
* import matplotlib.pyplot as plt
* from sklearn.metrics import confusion\_matrix
* import seaborn as sns
* import time
* import os
* import configparser
* from colorama import Fore, Style, init

Všetky potrebné knižnice je potrebne nainštalovať použitím príkazu “pip install <názov knižnice>“.

Po implementácii knižníc bolo zadefinovaných niekoľko najdôležitejších funkcií, takých ako:

"""Parameters for the configuration file"""  
load\_model = None  
auto\_save\_model = None  
optimizer = None  
epoch\_count = 0  
train\_batch\_size = 0  
test\_batch\_size = 0  
show\_confusion\_matrix = False  
learning\_rate = 0  
momentum = 0

Ďalej sú určené začiatočné parametre, ak konfiguračný súbor neexistuje, s následným načítaním konfiguračných parametrov zo súboru.

Ďalej nasleduje trieda MLP modelu:

class MLP(nn.Module):  
 def \_\_init\_\_(self):  
 super(MLP, self).\_\_init\_\_()  
 self.fc1 = nn.Linear(784, 256)  
 self.fc2 = nn.Linear(256, 64)  
 self.fc3 = nn.Linear(64, 10)  
 #self.dropout = nn.Dropout(0.2)  
  
 def forward(self, x):

*"""Available activation functions: ReLU, Sigmoid, Tanh"""*  
 x = x.view(-1, 784)  
 x = torch.relu(self.fc1(x))  
 #x = self.dropout(x)  
 x = torch.relu(self.fc2(x))  
 x = self.fc3(x)  
 return x

V ktorej je nastavene sloji modelu a aktivačne funkcie pre rôzne sloji. Ďalej, ak treba načítavame už existujúci model, nastavujem chybovú funkciu a metód na trénovanie modelu.

if load\_model == False:  
 model = MLP()  
else:  
 model = torch.load('mnist\_model.pth')  
 print(Fore.YELLOW + 'Model loaded')  
  
"""Loss function"""  
criterion = nn.CrossEntropyLoss()  
  
"""Optimizers"""  
if optimizer == 'sgd':  
 optimizer = optim.SGD(model.parameters(), lr=learning\_rate)   
elif optimizer == 'sgd\_momentum':  
 optimizer = optim.SGD(model.parameters(), lr=learning\_rate, momentum=momentum)  
elif optimizer == 'adam':  
 optimizer = optim.Adam(model.parameters(), lr=learning\_rate)

Ďalej ide metóda na trénovanie modelu a na testovanie. Tato metóda obsahuje informácie po každej epoche, ako: úspešnosť trénovania a testovania, percent chyb počas trénovania a najlepšia úspešnosť

def train\_model(model, criterion, optimizer, train\_loader, test\_loader, epochs=10):  
 global save\_model  
 max\_accuracy = 0  
 accepting\_rate = 0.97  
 training\_losses = []  
 evaluation\_accuracies = []  
  
 """"TTL - Time to live. If model does not update its accuracy by count of lives,  
 process of find best model will be terminated"""  
 live = 50  
 TTL = live  
  
 for epoch in range(epochs):  
 model.train()  
  
 """Training information"""  
 training\_info = {  
 'running\_loss': 0.0,  
 'correct': 0,  
 'total': 0  
 }  
  
 for inputs, labels in train\_loader:  
 optimizer.zero\_grad()  
  
 outputs = model(inputs)  
 loss = criterion(outputs, labels)  
 loss.backward()  
 optimizer.step()  
  
 """Append training information"""  
 training\_info['running\_loss'] += loss.item()  
 \_, predicted = torch.max(outputs, 1) # predicated output  
 training\_info['total'] += labels.size(0)  
 training\_info['correct'] += (predicted == labels).sum().item()  
  
 training\_accuracy = training\_info['correct'] / training\_info['total']  
 training\_losses.append(training\_info['running\_loss'] / len(train\_loader))  
  
 """Evaluation model"""  
 evaluation\_accuracy = evaluate\_model(model, test\_loader)  
 evaluation\_accuracies.append(evaluation\_accuracy)  
  
 if evaluation\_accuracy > max\_accuracy:  
 max\_accuracy = evaluation\_accuracy  
 TTL = live  
 if save\_model and max\_accuracy > accepting\_rate:  
 torch.save(model.state\_dict(), 'mnist\_model.pth')  
  
 """Logging"""  
 print(Fore.GREEN + "--------------------------------------------------------------------------------\n",  
 Fore.LIGHTMAGENTA\_EX + f"Epoch {epoch + 1},",  
 Fore.LIGHTRED\_EX + f" Train Loss: {training\_info['running\_loss'] / len(train\_loader):.4f}, ",  
 Fore.LIGHTGREEN\_EX + f"Train Accuracy: {training\_accuracy:.4f}, ",  
 Fore.LIGHTCYAN\_EX + f"Test Accuracy: {evaluation\_accuracy:.4f}",  
 Fore.GREEN + "\n--------------------------------------------------------------------------------")  
  
 if evaluation\_accuracy < max\_accuracy:  
 TTL -= 1  
  
 if TTL == 0:  
 print(Fore.GREEN + f"Best model was not reached by {live} hope.")  
 break  
  
 print(Fore.LIGHTGREEN\_EX + "Best accuracy: ", max\_accuracy)  
  
 plot\_metrics(training\_losses, evaluation\_accuracies)  
  
 return model  
  
  
def evaluate\_model(model, test\_loader):  
 model.eval()  
  
 training\_info = {  
 'correct': 0,  
 'total': 0  
 }  
  
 with torch.no\_grad():  
 for inputs, labels in test\_loader:  
 outputs = model(inputs)  
 \_, predicted = torch.max(outputs, 1)  
 training\_info['total'] += labels.size(0)  
 training\_info['correct'] += (predicted == labels).sum().item()  
  
 evaluation\_accuracy = training\_info['correct'] / training\_info['total']  
 return evaluation\_accuracy

# Porovnávanie parametrov

Počas testovanie prišli sme k takým výsledným charakteristikám trénovaniach metód:

* SGD: ma pomalsiu konvergenciu v porovnaini s ostatnymi, ale výsledky je viac stabilné
* SGD s momentumom: konvergencia je rýchlejšia, ako pri SGD
* Adam: ma najrýchlejšiu konvergenciu a je oveľa presnejšia

Počas testovanie prišli sme k takým výsledným charakteristikám aktivačných funkcii:

* ReLU: oproti iných funkcii viac vyhovuje, je rýchlejšia
* Sigmoid: konvergencia je pomalšia
* Tanh: lepšia ako Sigmoid, ale horšia, ako ReLU

Počet vrstiev a ich veľkosť je tiež veľmi dôležitá. Preto potrebujeme nájsť najlepší počet vrstiev a ich rozmery.

Pri veľkej rýchlosti upečenia model môže nedoisť ku optimálnemu riešeniu. Losses môžu byt väčšie. Pri nízkej model bude pomalé hľadať optimálne riešenie, môže ovisnúť v lokálnom minimume.

Pri malej veľkosti dávky učenie je rýchlejšie, ale je menej stabilne, ako pri veľkej dávky.

# Testovanie

Počas testovania používal a uviedol rôzne trénovanie metódy(sgd, sgd\_momentum, adam), počet vrstiev a ich veľkosti, rôzne aktivačné funkcie(ReLU (Rectified Linear Unit), Sigmoid, Tanh), rýchlosť učenia(learning rate) a veľkosť dávky.

Prvý test som začal zo vstupnými parametrami, chcel som porovnať rôzne trénovanie metódy:

optimizer(sgd, sgd\_momentum, adam) = adam

epoch\_count = 200

train\_batch\_size = 64

test\_batch\_size = 64

learning\_rate = 0.001

Tak ako máme na vstupe obrázok 28x28, budeme mať 784 neurónov. V najprv použili sme dve vrstvy, vstupnú a výstupnú.

Vstupná vrstva:

* Vstup: 784 neurónov
* Výstup: 32 neurónov
* Aktivácia: ReLU

Výstupná vrstva:

* Vstup: 32
* Výstup: 10

Posledné 3 epochy(trénovanie a testovanie):

Epoch 18, Train Loss: 0.0674, Train Accuracy: 0.9793, Test Accuracy: 0.9673

Epoch 19, Train Loss: 0.0642, Train Accuracy: 0.9809, Test Accuracy: 0.9691

Epoch 20, Train Loss: 0.0609, Train Accuracy: 0.9813, Test Accuracy: 0.9680

Ma dosť rýchly rast.

Зображення, що містить текст, схема, ряд, Графік

Автоматично згенерований опис

optimizer(sgd, sgd\_momentum, adam) = sgd

epoch\_count = 200

train\_batch\_size = 64

test\_batch\_size = 64

learning\_rate = 0.001

Vstupná vrstva:

* Vstup: 784 neurónov
* Výstup: 32 neurónov
* Aktivácia: ReLU

Výstupná vrstva:

* Vstup: 32
* Výstup: 10

Posledné 3 epochy(trénovanie a testovanie):

Epoch 18, Train Loss: 0.4669, Train Accuracy: 0.8793, Test Accuracy: 0.8863

Epoch 19, Train Loss: 0.4545, Train Accuracy: 0.8816, Test Accuracy: 0.8878

Epoch 20, Train Loss: 0.4439, Train Accuracy: 0.8835, Test Accuracy: 0.8895

Ako môžeme vidieť, pre tých to parametroch je dosť neúspešná, dosť malý počet epoch.

Зображення, що містить схема, ряд, текст, Графік

Автоматично згенерований опис

optimizer(sgd, sgd\_momentum, adam) = sgd\_momentum

epoch\_count = 200

train\_batch\_size = 64

test\_batch\_size = 64

learning\_rate = 0.001

momentum = 0.9

Vstupná vrstva:

* Vstup: 784 neurónov
* Výstup: 32 neurónov
* Aktivácia: ReLU

Výstupná vrstva:

* Vstup: 32
* Výstup: 10

Posledné 3 epochy(trénovanie a testovanie):

Epoch 18, Train Loss: 0.2210, Train Accuracy: 0.9383, Test Accuracy: 0.9381

Epoch 19, Train Loss: 0.2159, Train Accuracy: 0.9394, Test Accuracy: 0.9404

Epoch 20, Train Loss: 0.2114, Train Accuracy: 0.9405, Test Accuracy: 0.9406

Ako môžeme vidieť, pre tejto optimalizačnej funkcie ma stabilný rast úspešnosti, lepšie bolo b, keď sme zväčšili počet epoch.

Зображення, що містить текст, схема, ряд, Графік

Автоматично згенерований опис

Výsledky pre rôznych aktivačných funkciách(ReLU, Sigmoid, Tanh):

Test Accuracy: 0.9554, Test Accuracy: 0.9607, Test Accuracy: 0.9541

Зображення, що містить текст, схема, ряд, Графік

Автоматично згенерований описЗображення, що містить текст, схема, ряд, Графік

Автоматично згенерований опис Зображення, що містить текст, схема, Графік, ряд

Автоматично згенерований опис

Segmid v tomto testovanii mal najlepsie vysledky z pohľadu časovej zložitosti a presnosti. Ale ReLU bol richlesij.

Z testovanie vyššie vieme, že potrebujeme zmeniť počet a rozsah vrstiev.

optimizer(sgd, sgd\_momentum, adam) = adam

epoch\_count = 20

train\_batch\_size = 64

test\_batch\_size = 64

show\_confusion\_matrix = False

learning\_rate = 0.01

dropout (0.2)

Vstupná vrstva:

* Vstup: 784 neurónov
* Výstup: 256 neurónov
* Aktivácia: ReLU
* dropout(x)

Skrytá vrstva:

* Vstup: 256 neurónov
* Výstup: 64 neurónov
* Aktivácia: ReLU

Výstupná vrstva:

* Vstup: 64
* Výstup: 10

Posledné 3 epochy(trénovanie a testovanie):

Epoch 18, Train Loss: 0.1022, Train Accuracy: 0.9757, Test Accuracy: 0.9732

Epoch 19, Train Loss: 0.0923, Train Accuracy: 0.9778, Test Accuracy: 0.9747

Epoch 20, Train Loss: 0.0914, Train Accuracy: 0.9782, Test Accuracy: 0.9762

Зображення, що містить текст, схема, Графік, ряд

Автоматично згенерований опис

Nástupným krokom vyskúšame podobrať optimálnu rýchlosti upečenia:

Vyskúšal som znížiť lr na 0.001

optimizer(sgd, sgd\_momentum, adam) = adam

epoch\_count = 20

train\_batch\_size = 64

test\_batch\_size = 64

show\_confusion\_matrix = False

learning\_rate = 0.001

dropout (0.2)

Vstupná vrstva:

* Vstup: 784 neurónov
* Výstup: 256 neurónov
* Aktivácia: ReLU
* dropout(x)

Skrytá vrstva:

* Vstup: 256 neurónov
* Výstup: 64 neurónov
* Aktivácia: ReLU

Výstupná vrstva:

* Vstup: 64
* Výstup: 10

Posledné 3 epochy(trénovanie a testovanie):

Epoch 18, Train Loss: 0.0101, Train Accuracy: 0.9966, Test Accuracy: 0.9800

Epoch 19, Train Loss: 0.0075, Train Accuracy: 0.9973, Test Accuracy: 0.9781

Epoch 20, Train Loss: 0.0071, Train Accuracy: 0.9978, Test Accuracy: 0.9796

Зображення, що містить текст, схема, ряд, Графік

Автоматично згенерований опис

Výsledok je oveľa lepší ako pri 0.01

Pri nasledujúcich testovaniach som doimplementoval vypísanie najlepšieho dosiahnutého výsledku.

optimizer(sgd, sgd\_momentum, adam) = adam

epoch\_count = 200

train\_batch\_size = 64

test\_batch\_size = 64

learning\_rate = 0.001

optimizer(sgd, sgd\_momentum, adam) = adam

epoch\_count = 200

train\_batch\_size = 64

test\_batch\_size = 64

learning\_rate = 0.001

Vstupná vrstva:

* Vstup: 784 neurónov
* Výstup: 256 neurónov
* Aktivácia: ReLU

Skrytá 1 vrstva:

* Vstup: 256 neurónov
* Výstup: 128 neurónov
* Aktivácia: ReLU

Skrytá 2 vrstva:

* Vstup: 128 neurónov
* Výstup: 64 neurónov
* Aktivácia: ReLU

Výstupná vrstva:

* Vstup: 64
* Výstup: 10

Posledné 3 epochy(trénovanie a testovanie):

Najlepšia uspesnost: 98.5

Зображення, що містить текст, схема, знімок екрана, Графік

Автоматично згенерований опис

# Najlepšie nájdene parametre

# **Zhodnotenie**

Vytvorená neurónová sieť na klasifikáciu ručne písaných číslic z dátového súboru MNIST je určený na trénovanie a testovanie vytvoreného systému s rôznymi parametrami, ako napríklad: optimalizačný metód, vhodné aktivačné funkcie, rýchlosť učenia a veľkosť dávky (batch) pre optimalizačný algoritmus. Taktiež treba bolo cestou pokusov a omylov navrhnúť počet vrstiev a ich veľkosti pre vytvorenú model. Otestovať, porovnať výsledky testovania a vyhodnotiť úspešnosť nerehneného modelu.

Výtvory, natrénovaný model je schopný klasifikovať číslice s percentom úspešnosti 98 a vyššie. Závisí od poctu epoch a nastavení modelu na trénovaní. Najlepší výsledok bol opísaný v testovacej časti a dokázal schopnosť siete riešiť podobne problémy na klasifikáciu číslic.

Taktiež program umožňuje konfigurovať konfiguráciu pred behom programu. Čo výrazne zrýchľuje prístup k jednotným špecifikáciám a prispôsobeným nástrojom pre určite potreby klienta. Ako napríklad používateľ môže si zvoliť spôsob učenia site, aká najviac mu vyhovuje.

Dosť dôležitou úlohou bolo správne vybrať a nastrojiť parametre pre učenia. Boli porovnané rôzne spôsoby konfigurácie, a navrhnutá najlepšia konfigurácia, ktorú sa dala vytvoriť.