Umelá inteligencia:  
Neurónové siete

**Marián Kurčina**

**ZS 2024**

[**xkurcinam@stuba.sk**](mailto:xkurcinam@stuba.sk)[**marian.kurcina2003@gmail.com**](mailto:marian.kurcina2003@gmail.com)

Table of Contents

[MNIST Classifier 3](#_Toc189512365)

[Task 3](#_Toc189512366)

[Solution Description 3](#_Toc189512367)

[Code Description 3](#_Toc189512368)

[Optimizing Training Parameters 4](#_Toc189512369)

[Finding the Optimal Model Hyperparameters 4](#_Toc189512370)

[Optimal Model Architecture 5](#_Toc189512371)

[Conclusion 6](#_Toc189512372)

[Backpropagation algorithm 6](#_Toc189512373)

[Zadanie 6](#_Toc189512374)

[Opis riešenia 6](#_Toc189512375)

[Opis kódu 6](#_Toc189512376)

[Model 7](#_Toc189512377)

[MSELoss 7](#_Toc189512378)

[Lineárna vrstva 7](#_Toc189512379)

[Sigmoid 8](#_Toc189512380)

[Relu 8](#_Toc189512381)

[Tanh 8](#_Toc189512382)

[Hľadanie optimálnych hyperparametrov modelu 8](#_Toc189512383)

[Architektúra optimálneho modelu 9](#_Toc189512384)

[Záver 9](#_Toc189512385)

[Zdroje 10](#_Toc189512386)

# MNIST Classifier

## Task

The goal is to create a neural network for classifying handwritten digits from the MNIST dataset. A feedforward neural network (multilayer perceptron) should be used and trained with the following optimization algorithms: SGD, SGD with momentum, and ADAM. In addition to measuring the training and test error, the model's accuracy should also be evaluated. The final model should achieve an accuracy of more than 97%. The PyTorch library should be used for implementation.

## Solution Description

First, I load the dataset into the program and transform it so that the values range between 0 and 1. Then, I split the data into two parts: a training set and a test set. After obtaining and preprocessing the dataset, I train and test the model separately for each optimization algorithm.

The training process runs for several epochs. In each epoch, batches of digit images are fed into the model, generating predictions. The difference between the predictions and the actual labels is used to compute the gradient, which updates the linear layers of the model. The way these layers are updated depends on the training algorithm.

At the end of each epoch, I print the average training error and evaluate the model on the test set. I store the test results and proceed to the next epoch. After completing a predefined number of epochs, I conduct a final test on the model, which represents the final trained version. Finally, I generate graphs displaying the training and test errors along with the model's accuracy.

## Code Description

At the beginning, I load and transform the MNIST dataset into train\_dataset and test\_dataset, where the first part is used for training and the second for testing.  
Next, I divide the dataset into batches using train\_loader and test\_loader. These loaders facilitate training and testing.  
The training and testing process follows. I first test using SGD, which requires a learning\_rate parameter. Then, I test using SGD with momentum, requiring both learning\_rate and momentum. Finally, I use ADAM, which requires only learning\_rate.  
The training process starts with defining the model and its layers in the code.  
For each epoch:

* I reset the gradients.
* I compute the model output and the loss.
* I perform backpropagation using the loss function.
* I update the linear layer weights using the selected optimizer (SGD/SGD with momentum/ADAM).

The model output is always given as probabilities for each digit.  
For every epoch, the program stores the training loss. After processing all images in the dataset, it evaluates the model using the test set, storing the accuracy of correct predictions and the test error. These results are visualized in graphs at the end, and the model’s accuracy is further analyzed using a confusion matrix.

## Optimizing Training Parameters

As a first step, I decided to find the best parameters for each training algorithm, including the learning rate and momentum over 30 epochs.

|  |  |  |
| --- | --- | --- |
|  | SGD | ADAM |
| 0.1 | **95.61%** | 86.89% |
| 0.01 | 78.48% | 97.31% |
| 0.001 | 11.35% | **97.70%** |

Table 1: Results of testing the optimal learning rate for the training algorithms SGD and ADAM

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0.1 | 0.01 | 0.001 |
| 0.95 | **98.18%** | 96.89% | 88.59% |
| 0.9 | 97.79% | 95.77% | 77.98% |
| 0.85 | 97.95% | 94.29% | 61.09% |

Table 2: Results of testing the optimal learning rate and momentum for the training algorithm SGD with momentum

From the testing, I found that the best learning rate is 0.1 for SGD and SGD with momentum, while for ADAM, it is 0.001. The optimal momentum for SGD with momentum is 0.95.

## Finding the Optimal Model Hyperparameters

First, I focused on selecting the right activation functions. Since my network consists of two layers, I needed to choose two activation functions. I decided to test three different activation functions: ReLU, Tanh, and Sigmoid, and I evaluated all possible combinations.

|  |  |  |  |
| --- | --- | --- | --- |
|  | ReLu | Tanh | Sigmoid |
| ReLu | 97.23% | 97.56% | 97.26% |
| Tanh | 97.73% | 97.70% | 97.63% |
| Sigmoid | **97.80%** | 97.76% | 97.02% |

Table 3: Results of testing all combinations of activation functions. The first activation function in the network is listed in the columns, while the second is in the rows. The table presents the averages for all training algorithms (learning rate = 0.1, momentum = 0.9, number of epochs = 30, first layer = 100 neurons, second layer = 50 neurons).

From the test results, I found that the best activation functions to use are ReLU and Sigmoid. Next, I searched for the parameters of the linear layers, specifically the number of neurons in each layer, to maintain the highest possible accuracy.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 150 | 100 | 50 |
| 80 | 97.97% | 97.88% | 97.30% |
| 50 | 97.94% | **97.80%** | 97.29% |
| 20 | 97.87% | 97.59% | 97.04% |

Table 4: Results of testing combinations of neuron counts in layers

From the testing, I determined that the best choice is to use 100 and 50 neurons.

### Optimal Model Architecture

My network will have 2 hidden layers, the first with 100 neurons and the second with 50 neurons, using ReLU and Sigmoid as activation functions.

* Accuracy with SGD: **97.46%**
* Accuracy with SGD with momentum: **98.04%**
* Accuracy with ADAM: **97.93%**

For each run of the program, a confusion matrix is displayed.

Obrázok, na ktorom je klávesnica, snímka obrazovky, vstupné zariadenie, páka

Automaticky generovaný popis

Image 1: Confusion matrix

Obrázok, na ktorom je rad, vývoj, text, diagram

Automaticky generovaný popisObrázok, na ktorom je rad, text, vývoj, diagram

Automaticky generovaný popis

Obrázok, na ktorom je text, rad, vývoj, diagram

Automaticky generovaný popis  
Image 2: Graphs of training and test error progression and model accuracy

## Conclusion

I successfully learned to work with the PyTorch library and created a program for digit classification using the MNIST dataset and a neural network. I was able to identify the optimal model architecture and the best parameters for the training algorithms SGD, SGD with momentum, and ADAM.

# Backpropagation algorithm

## Zadanie

V tejto úlohe je potrebné implementovať plne funkčný algoritmus backpropagation, treba implementovať doprednú aj spätnú časť pre operatory a funkcie, ako aj aktualizácie parametrov siete. Algoritmus overíte pomocou natrénovania doprednej neurónovej siete. Na riešenie úlohy použite knižnicu NumPy, použitie knižníc PyTorch a TensorFlow je zakázané. Implementujte modulárnu architektúru, v ktorej bude možné jednotlivé moduly reťaziť.Implementácia musí obsahovať lineárnu vrstvu, aplikačné funkcie sigmoid, tanh a relu a chybovú funkciu MSE. Pre validáciu použite problémy AND, OR a XOR, vyskúšajte siete s 1 a 2 skrytými vrstvami.

## Opis riešenia

Kedže model musí byť schopný reťazenia, implementujeme jednotlivé vrstvy a celý model pomocou objektovo orientovaného programovania, pričom každá vrstva bude vlastný objekt s funkciami. Model bude schopné definovať pomocou vrstiev a teda bude možné jednoducho neurónovú site upravovať. Trénovanie bude prebiehať opakovaným vkladaním inputov 1 a 0 do siete, získania predikcie siete a zmeny váh lineárnych vrstiev pomocou rozdielu a jeho gradient.  
Trénovanie bude prebiehať v epochách, v každej epoche vložím do inputu siete všetky kombinácie dvojíc 0 a 1. Model by dá predikciu, z tejto predikcie pomocou MSELoss zistím chybu, z ktorej zistím gradient. Tento gradient pošlem spätným chodom a v každej vrstve sa zmení. Ak prejde lineárnou vrstvou zmení pomocou matíc gradientov, prípadne matíc rýchlosti za použitia momentu váhy a biasy vrstvy. Po prejdení gradientu všetkými vrstvami proces začína s novým inputom, prípadne novou epochou. Po danom počte epoch vypíše predikcie pre všetky inputy a užívateľ môže skontrolovať, či je jeho predikcia správna.

## Opis kódu

Model, chyba MSELoss a všetky vrstvy sú definované ako objekty s forward a backward funkciou. Model je schopný vyvolať postupne všetky funkcie jeho vrstiev, pri prednom chode vyvoláva funkcie forward, pri spätnom vyvolá funkcie backward.   
Proces trénovania funguje tak, že model dostane za každým nejaký input, ktorý bude vložený do modelu, model vykoná postupne pre každú jeho vrstvu funkciu forward, pričom do vstupu bude dávať output z predchádzajúcej vrstvy.Po prejdení celej vrstvy vzkoná predpoklad. Zistí rozdiel predpokladu a skutočného označenia pomocou MSELoss.forward, z tejto chyby zistí gradient, ktorý potom pošle do spätného prechodu modelom. Pre spätný chod vyvolá model postupne pre všetky vrstvy odzadu funkcie backward, pričom output predchádzajúcej vrstvy sa stáva inputom tej ďalšej. Keď prejde gradient lienárnou vrstvou zmení matice váh a biasov.  
Po spätnom chode tento process pokračuje pre ďalší input.

### Model

Model je definovaný jeho vrstvami, forward metóda vyvolá forward metódy pre všetky jeho vrstvy, pričom z outputu vrstvy sa stáva input pre ďalšiu vrstvu. Backward metóda vyvolá odzadu metódy backward všetkých vrstiev, pričom z output vrstvy sa stáva input ďalšej vrstvy.

### MSELoss

Tento objekt slúži na výpočet chyby modelu a výpočet gradient.

Forward metóda: – priemerný rozdiel na druhú

Backward metóda: – priemer pre všetky zložky outputu

### Lineárna vrstva

Lineárna vrstva je jediná vrstva, ktorá mení svoje parameter pri backpropagation. Táto vrstva si musí pamätať maticu váh(W) a maticu bias(b) pre forward metódu. Ďalej si musí vrstva pamätať posledný input a pre počítanie s momentom si musí pamätať matice rýchlostí (velocity) pre matice váh(vW) a bias(vb).

Forward metóda:   
Pri forward metóde bude input vynásobený maticou váh a bude k nemu pripočítaný vektor bias.

Backward metóda:

= grad

S momentom:

Bez momentu:

Pri backward metóde sa najprv vypočítajú matice gradientov(dW,db), ktoré budú slúžiť na aktualizovanie matice váh a matice bias. Ak máme momentum vypočítajú sa ešte matice velocity pomocou momentu a matíc gradientu. Tieto matice slúžia na aktualizovanie matíc váh.

### Sigmoid

Aktivačná funkcia sigmoid si musí zapamätať hodnotu posledného outputu, táto hodnota sa používa pri výpočte backward metódy.

Forward metóda:

Backward metóda:

### Relu

Táto aktivačná funkcia vráti 0 ak ide o záporné číslo alebo x ak ide o kladné číslo, tento objekt si musí pamätať posledný input metódy forward.

Forward metóda:

Backward metóda:

### Tanh

Aktivačná funkcia tanh si musí zapamätať hodnotu postedného output, táto hodnota sa používa pri výpočte backward metódy.

Forward metóda:

Backward metóda:

## Hľadanie optimálnych hyperparametrov modelu

Mám dané, že site má mať 1 skrytú vrstvu a to s 4 neurónami, mám avšak vyskúšať aj site s 2 skrytými vrstvami, pre toto som sa rozhodol, že budem mať 2 vrstvy o 2 neurónoch, čím porovnám účinnosť dvoch sietí s rovnakým počtom neurónov ale iným počtom vrstiev. Nájdem pre prvú sieť ideálne parametre a s týmito parametrami siete porovnám za použitia momentu aj bez momentu.  
Parametre, ktoré musím nájsť sú learning rate a ktoré aktivačné funkcie sú na vyriešenie problému najvýhodnejšie, aktivačné vrstvy som hľadal osobitne pre obe siete. Pre každú sieť budem dávať rovnaké aktivačné funkcie. Hodnota momentu bude rovnaké ako v prvej úlohe, takže 0.95.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0.1 | 0.05 | 0.01 |
| Sigmoid | 1.4237 | 1.8026 | 1.9729 |
| Tanh | **0.0511** | 0.0776 | 0.2674 |
| ReLU | 0.9904 | 0.9990 | 1.0001 |

Tabuľka 5: Výsledky testovania siete s jednou skrytou vrstvou pre 500 epoch pre problém XOR bez momentu. Úspešnosť porovnávam na základe celkovej odchylky finálneho modelu

Z výsledkov testovania som zistil, že najlepšie aktivačné funkcie pre sieť s jednou skrytou vrstvou s 4 neurónmi sú tanh a najlepším learning rate je 0.1. NAsledovne som našiel najlepšie aktivačné funkcie pre sieť s 2 skrytými vrstvami spolu s 4 neurónmi.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Sigmoid | Tanh | Relu |
| 0.1 | 1.9773 | 0.4794 | 2 |

Tabuľka 6: Výsledky testovania s 2 skrytými vrstvami pre 500 epoch pre problém XOR bez momentu

Pri 2 skrytých vrstvách som spozoroval, že Tanh sa drží na rovnakej loss a iba niekedy sa model zlepší, Relu zase nie je schopný učenia, jeho hodnoty sa nemenia. Preto na tento problem je lepšie použiť iba 1 skrytú vrstvu.

### Architektúra optimálneho modelu

Ideálny model pre riešenie logických členov bez momentu je model s jednou skrytou vrstvou a za použitia Tanh aktivačnej funkcie.

Môj model bude mať 1 skrytú vrstvu s 4 neurónmi, bude mať 2 aktivačné funkcie Tanh.

|  |  |  |  |
| --- | --- | --- | --- |
|  | XOR | OR | AND |
| Bez momentu | 0.0663 | 0.0413 | 0.039 |
| S momentom | 1.0002 | 1.0051 | 3 |

Tabuľka 7: Výsledná úspešnosť na základe lossu pre jednotlivé logické problémy

Pre môj ideálny model som vyskúšal aj trénovanie s momentom. Ak by som použil momentum 0.95 sieť nebolo možné natrénovať s aktivačnými funkciami Tanh, avšak keď som vyskúšal aktivačnú funkciu Sigmoid, trénovanie sa podarilo.

Obrázok, na ktorom je text, rad, vývoj, diagram

Automaticky generovaný popis Obrázok, na ktorom je text, rad, vývoj, snímka obrazovky

Automaticky generovaný popis

Obrázok 3: Grafy zobrazujúce vývoj chyby cez epochy

## Záver

Podarilo sa mi implementovat backpropagation algoritmus, implementoval som všetky zadané časti a pre každú som naprogramoval predný aj spätný chod. Môj model je schopný reťazenia a je možné zmeniť jeho vrstvy. Podarilo sa mi nájsť optimálny model pre všetky problémy XOR, OR a AND za použitia 1-vrstvového a 2-vrstvového modelu (skryté vrstvy), s momentum aj bez momentu.

# Zdroje

[Building a neural network FROM SCRATCH (no Tensorflow/Pytorch, just numpy & math)](https://www.youtube.com/watch?v=w8yWXqWQYmU)

<https://eli.thegreenplace.net/2018/backpropagation-through-a-fully-connected-layer/>

<https://www.youtube.com/watch?v=Ilg3gGewQ5U>