Jozef Waldhauser

Úloha 7

Učenie neurónovej siete s učiteľom

Obsah

[Úloha 7 – zadanie 2](#_Toc197289346)

[Úloha 7 – zdroje 2](#_Toc197289347)

[Teória 2](#_Toc197289348)

[Učenie s učiteľom 3](#_Toc197289349)

[Optimalizátory 3](#_Toc197289350)

[Aktivačné funkcie 3](#_Toc197289351)

[Stratová funkcia 3](#_Toc197289352)

[Design algoritmu 4](#_Toc197289353)

[Implementácia 5](#_Toc197289354)

[Dataset MNIST 5](#_Toc197289355)

[Definovanie architektúry modelu 6](#_Toc197289356)

[Trénovanie modelu 6](#_Toc197289357)

[Vyhodnotenie modelu 7](#_Toc197289358)

[Experimenty/analýza 7](#_Toc197289359)

[Záver 8](#_Toc197289360)

# Úloha 7 – zadanie

1. stáhněte si vhodná trénovací + testovací data z <https://www.tensorflow.org/datasets/catalog/overview#all_datasets>
2. předzpracujte vhodně data
3. zvolte vhodnou topologii sítě (struktura a typ vrstev, přechodové funkce), učící algoritmus a měření chyby
4. naučte síť, aby dosahovala přesnosti alespoň 90% na testovacích datech

**Úlohou bolo navrhnúť a natrénovať neurónovú sieť na rozpoznávanie ručne písaných číslic tak, aby dosahovala presnosť minimálne 90 % na testovacích dátach. Sieť je učená učiteľom.**

# Úloha 7 – zdroje

<https://www.tensorflow.org/guide/keras/sequential_model>

<https://www.tensorflow.org/datasets/catalog/mnist>

<https://www.deeplearningbook.org/>

<https://www.youtube.com/watch?v=i8NETqtGHms> – tensorflow in 100 seconds

<https://www.youtube.com/watch?v=Mubj_fqiAv8&list=PLeo1K3hjS3uu7CxAacxVndI4bE_o3BDtO> – tensorflow learning playlist

<https://www.youtube.com/watch?v=6_2hzRopPbQ> - tensorflow

# Teória

**Umelé neurónové siete (UNS)** sú výpočtové modely inšpirované biologickými neurónmi v mozgu. Skladajú sa z uzlov (neurónov) usporiadaných do vrstiev – vstupnej vrstvy, jednej alebo viacerých skrytých vrstiev a výstupnej vrstvy.

Každý neurón vezme svoje vstupy, vynásobí ich tzv. váhami, spočíta ich, a výsledok upraví pomocou aktivačnej funkcie. Tým sa pridáva „nelinearita“ – teda schopnosť modelu robiť zložitejšie rozhodnutia.

kde , ... sú vstupy a sú váhy, ***b*** je bias - tzv. posun.

## Učenie s učiteľom

Ide o typ učenia, pri ktorom má model k dispozícii vstupy aj správne výstupy (tzv. „označené dáta“). Cieľ je jednoduchý: **naučiť sa predpovedať správne výstupy na základe vstupov**. Model počíta chybu medzi tým, čo vypočítal, a tým, čo mal povedať. Pomocou **backpropagation** sa upravujú váhy, aby bola chyba v ďalšom kole menšia. Ďalej tu sú optimalizačné algoritmy ako je napríklad **Adam**, **SGD** a ďalšie.

## Optimalizátory

Sú to algoritmy, ktoré hovoria, **ako meniť váhy siete**, aby sa znižovala chyba. Medzi najčastejšie používané patria:

* **SGD (Stochastic Gradient Descent)** – klasický algoritmus založený na stochastickom zostupe po gradiente.
* **Adam (Adaptive Moment Estimation)** – adaptívny optimalizátor, ktorý kombinuje výhody Adagrad a RMSprop.
* **RMSprop, Nadam, Adadelta, Adamax** – ďalšie varianty, ktoré upravujú učenie podľa historických hodnôt gradientov.

## Aktivačné funkcie

Podobne ako v predošlej úlohe 6 taktiež používame **ReLU a softmax.**

**ReLU (Rectified Linear Unit) –** najpoužívanejšia funkcia v deep neural networks, nahrádza záporné hodnoty nulami.

**Softmax –** používa sa v poslednej vrstve classification modelov pre viac tried, prevádza výstupy na pravdepodobnostné rozdelenie.

## Stratová funkcia

Slúži na výpočet chyby medzi predikciou siete a reálnym výstupom. V úlohe klasifikácie viac tried sa najčastejšie používa:

* **Sparse Categorical Crossentropy** – vhodná pre úlohy, kde výstupom je celé číslo označujúce triedu.

# Design algoritmu

* **Načítanie dát:**
  + Použije sa dataset MNIST (obsahuje 60 000 trénovacích a 10 000 testovacích obrázkov ručne písaných číslic 0–9).
  + Obrázky majú rozmer 28 × 28 pixelov (čiernobiele).
  + Každý obrázok je reprezentovaný ako matica hodnôt (jas pixelov).
* **Predspracovanie:**
  + Všetky pixely sú škálované na hodnoty medzi 0 a 1 (pôvodne 0–255).
  + Tým sa urýchľuje učenie modelu a stabilizuje správanie optimalizačných algoritmov.
* **Vizualizácia:**
  + Vybrané obrázky sa zobrazia pomocou knižnice (napr. matplotlib).
  + Umožňuje rýchlu vizuálnu kontrolu dát – či sú správne načítané a či majú očakávaný formát.
* **Definícia modelu:**
  + Model je typu Keras Sequential (ide vrstvy za sebou).
* **Architektúra:**
  + Flatten: premení 2D obrázok (28×28) na 1D vektor (784 prvkov).
  + Dense: plne prepojená vrstva so 128 neurónmi a ReLU aktiváciou.
  + Dropout: náhodne vypína 20–50 % neurónov (zabraňuje preučeniu).
  + Dense: výstupná vrstva s 10 neurónmi (pre číslice 0–9) a softmax funkciou (pravdepodobnosti).
* **Tréning modelu:**
  + Sieť sa trénuje na trénovacích dátach s príslušnými štítkami (číslami).
  + Používajú sa rôzne optimalizéry (napr. Adam, SGD), ktoré upravujú váhy tak, aby sa znižovala chyba.
  + Model prechádza viacerými epochami (kolami učenia).
* **Vyhodnotenie výkonu:**
  + Na testovacej množine sa vypočíta presnosť (accuracy).
  + Zobrazí sa matica zámien (confusion matrix) – ukazuje, ktoré číslice si model mýli.
  + Vygenerujú sa grafy učenia (strata a presnosť v čase).
* **Porovnanie:**
  + Výkon modelu sa porovnáva pre rôzne optimalizéry.
  + Zobrazia sa rozdiely v rýchlosti učenia, finálnej presnosti a správaní modelu.
  + Výsledky sa prezentujú pomocou tabuliek a grafov.

# Implementácia

Použil som dataset **MNIST** (obrázky rukou písaných číslic) na základe toho čo si vyberali spolužiaci aby som uľahčil debugovanie.

## Dataset MNIST

Dataset MNIST obsahuje 60 000 trénovacích a 10 000 testovacích obrázkov ručne písaných číslic od 0 do 9. Obrázky majú veľkosť 28x28 pixelov v odtieňoch šedej. Je to štandardný dataset pre testovanie klasifikačných úloh v oblasti strojového učenia.

A group of numbers in squares

AI-generated content may be incorrect.

Obrázok 1

Načítanie dát:

|  |
| --- |
| # dataset MNIST (ze vraj to kazdy pouziva)  mnist = tf.keras.datasets.mnist  # nacitanie trenovacich a testovacich dat  (train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()  # normalizacia obrazkov (z 0-255 na 0-1)  train\_images = train\_images / 255.0  test\_images = test\_images / 255.0 |

Normalizácia obrázkov prevádza hodnoty pixelov z rozsahu 0–255 na 0–1, čo **zlepšuje stabilitu a rýchlosť trénovania modelu.** Pomáha optimalizátorom efektívnejšie sa učiť a vedie k vyššej presnosti.

## Definovanie architektúry modelu

|  |
| --- |
| def vytvor\_model():      # Použitie funkcionálneho API namiesto Sequential pre odstránenie varovania      inputs = tf.keras.Input(shape=(28, 28))      x = tf.keras.layers.Flatten()(inputs)      x = tf.keras.layers.Dense(128, activation='relu')(x)      x = tf.keras.layers.Dropout(0.2)(x)      outputs = tf.keras.layers.Dense(10, activation='softmax')(x)        model = tf.keras.Model(inputs=inputs, outputs=outputs)      return model  model = vytvor\_model() |

1. Vstupná vrstva: 28×28 (flatten)
2. Skrytá vrstva: 128 neurónov, aktivácia **ReLU**
3. Dropout: 0.2
4. Výstupná vrstva: 10 neurónov, aktivácia **Softmax**

Dropout: Ak je sieť „príliš dobre naučená“ potom je síce veľmi presná na tréningových dátach, ale zlyháva na nových, neznámych dátach (napr. testovacích).

...Je to ako študent, ktorý sa naučil odpovede naspamäť, ale nerozumie látke.

## Trénovanie modelu

|  |
| --- |
| # Kompilacia      model.compile(          optimizer=opt,          loss='sparse\_categorical\_crossentropy',          metrics=['accuracy']      )        # Cas zaciatku trenovania      start\_time = time.time()        # Trenovanie      history = model.fit(          train\_images, train\_labels,          epochs=epochs,          validation\_split=0.1,          verbose=1      ) |

Po definovaní modelu ho musíme **kompilovať**, aby sme určili optimalizátor, funkciu straty a metriky, ktoré chceme sledovať. Model sa trénuje na **trénovacích dátach** (súbor train\_images a train\_labels) pomocou metódy .fit().

## Vyhodnotenie modelu

Po tréningu je model vyhodnotený na **testovacích dátach** pomocou .evaluate().

|  |
| --- |
| # Vyhodnotenie      test\_loss, test\_accuracy = model.evaluate(test\_images, test\_labels, verbose=0)      print(f"Presnost na testovacich datach: {test\_accuracy:.4f}") |

Toto umožňuje vidieť ako dobre sa model natrénoval.

# Experimenty/analýza

|  |  |  |
| --- | --- | --- |
| Optimizer | Epochs | Accuracy |
| SGD | 5 | 0.9377 |
| SGD | 7 | 0.9453 |
| Adam | 5 | 0.9775 |
| RMSProp | 5 | 0.9727 |
| Adamax | 5 | 0.9733 |
| Nadam | 5 | 0.9774 |
| Ftrl | 5 | 0.9319 |
| Ftrl | 10 | 0.9486 |
| Adam | 1 | 0.9534 |
| Ftrl | 1 | 0.8821 |

Adam a Nadam dosiahli najvyššiu presnosť (0.9775 a 0.9774) po 5 epóch, sú rýchle a stabilné. RMSProp a Adamax mali o niečo nižšiu presnosť (0.9727 a 0.9733). Ftrl bol najslabší po 1 epóche (0.8821), no po 10 epóch dosiahol 0.9486. SGD sa zlepšil z 0.9377 (5 epôch) na 0.9453 (7 epôch), ale zaostáva za Adamom a Nadamom.

**Najlepšie:** Adam a Nadam (97.75%, 5 epôch).

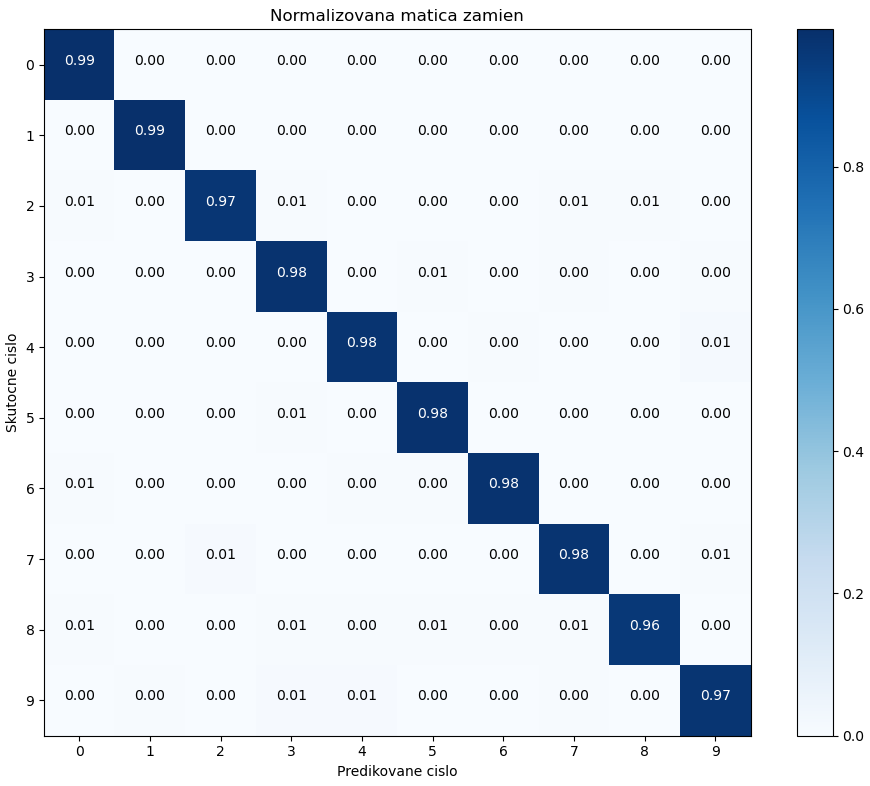
**Ftrl:** Pomalší, slušné výsledky po 10 epóch (0.9486).

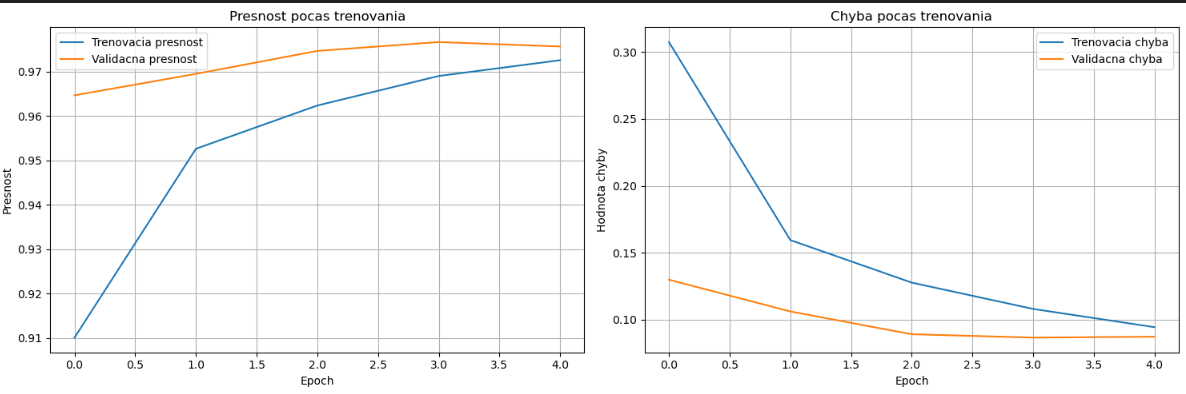
**SGD:** Lepší po 7 epóch (0.9453), ale menej efektívny.

**RMSProp a Adamax:** Dobré, no Adam a Nadam sú efektívnejšie.



Obrázok 2 Výsledky trénovania FTRL

A screenshot of a graph

AI-generated content may be incorrect.

Obrázok 4 Matica zámien po 5 epochách Adam

Obrázok 3 Matica zámien po 1 epoche ftrl

Obrázok 5 Trénovanie pomocou opt. algo Adam

# Záver

Neurónová sieť navrhnutá na rozpoznávanie ručne písaných číslic z datasetu MNIST dosiahla požadovanú presnosť nad 90 % na testovacích dátach.

Najlepšie výsledky ukázali optimalizátory Adam a Nadam s presnosťou ~97,75 % po 5 epóch, pomáhaju rýchlej a stabilnej konvergencii.

RMSProp a Adamax dosiahli o niečo nižšiu presnosť (~97,3 %), no stále sú efektívne. SGD a Ftrl vyžadujú viac epôch lepších výsledkov.

Architektúra siete s vrstvami Flatten, Dense (128 neurónov, ReLU), Dropout (0.2) a výstupnou vrstvou (10 neurónov, Softmax) spolu s normalizáciou dát zabezpečila stabilné a presné učenie. Adam a Nadam sú optimálnou voľbou pre túto úlohu kvôli ich rýchlosti a vysokej presnosti.