Umelá inteligencia: Zadanie III.
Umelé Neurónové Siete
- Dokumentácia

Fakulta Informatiky a Informčaých Technológií Slovenskej Technickej Univerzity

Ondrej Krajčovič

xkrajcovico@stuba.sk

8.Decembra 2024

1. California Housing problem

Zadanie[1]**:**

Úloha:

Vyviniete neurónovú sieť na vykonanie regresie na súbore údajov o bývaní v Kalifornii. Cieľom je predpovedať strednú hodnotu ceny domu pre okresy Kalifornie na základe niekoľkých údajov, ako sú údaje o obyvateľstve, príjme a polohe. Dátová množina obsahuje 20 640 prípadov s 8 údajmi, ako napríklad stredný príjem, vek nehnuteľnosti a priemerná obsadenosť a tiež cieľovú hodnotu, ktorou je stredná hodnota ceny domu. Táto úloha vám pomôže pochopiť, ako aplikovať neurónové siete na regresné problémy, kde výstupom je spojitá hodnota.

Zadanie:

Na riešenie úlohy použite doprednú neurónovú sieť (viacvrstvový perceptrón), kde vyskúšate viacero modelov a môžete sa snažiť nájsť ten najmenší, ale stále úspešný. Natrénujte ju pomocou algoritmov SGD, SGD s momentom a ADAM. Odmerajte trénovaciu a testovaciu chybu. V úlohe určite použite knižnicu PyTorch (stačí verzia pre CPU), iné knižnice podľa svojho uváženia.

Náčrt modelu:

Stiahnite si dátovú množinu alebo použite knižnicu scikit-learn a v nej je dataset možné priamo stiahnuť. Je dobré napísať potom k tomu triedu zdedenú z torch.utils.data.Dataset.

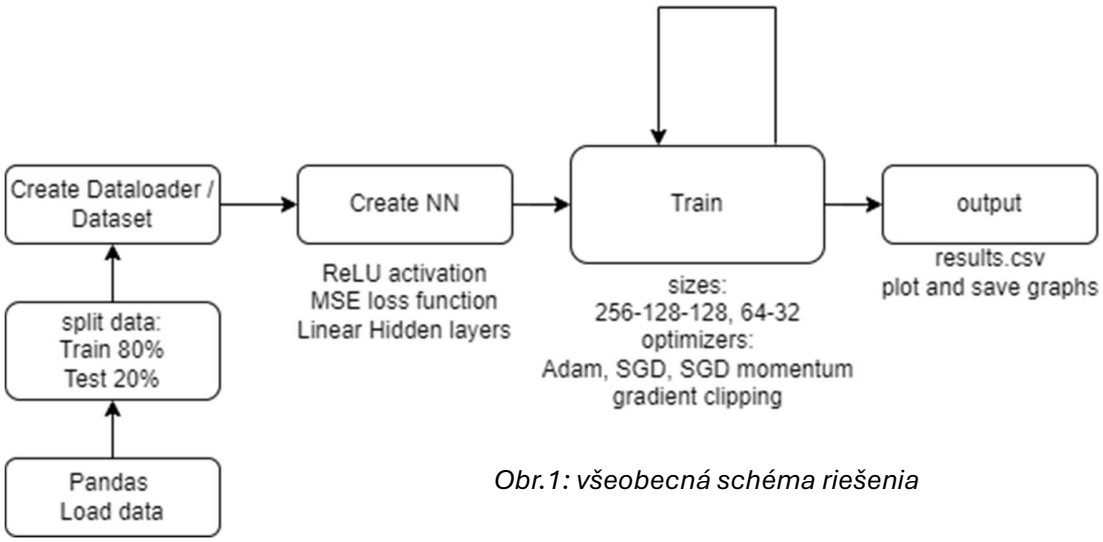
Rozdeľte dátovú množinu na trénovaciu a testovaciu (napr. 80% tréning, 20% testovanie).

Predspracujte vhodne dáta: normalizujte jednotlivé údaje. Zvážte škálovanie cieľových hodnôt na zlepšenie stability tréningu.

Navrhnite hyperparametre modelu – počet vrstiev a ich veľkosti, zvoľte si vhodné aktivačné funkcie, nastavte rýchlosť učenia a veľkosť dávky (batch) pre optimalizačný algoritmus.

Postupne vyskúšajte všetky tri optimalizačné algoritmy spomenuté v zadaní. Trénujte model vždy rovnaký počet epoch, aby ich bolo možné porovnať.

Riešnie:



Obr.1: všeobecná schéma riešenia

Pri riešení California housing problému porovnávam dva základné koncepty, a to väčšiu a menšiu NN. Pri väčšej používam tri skryté vrstvy a to o veľkosti 256,128,128.
Pri menších používam iba dva a to o veľkosti 64 a 32.

Medzi porovnaním väčších a menších sietí porovnávam jednotlivé optimalizátory. A to konkrétne adam, SGD a SGD-momentum.

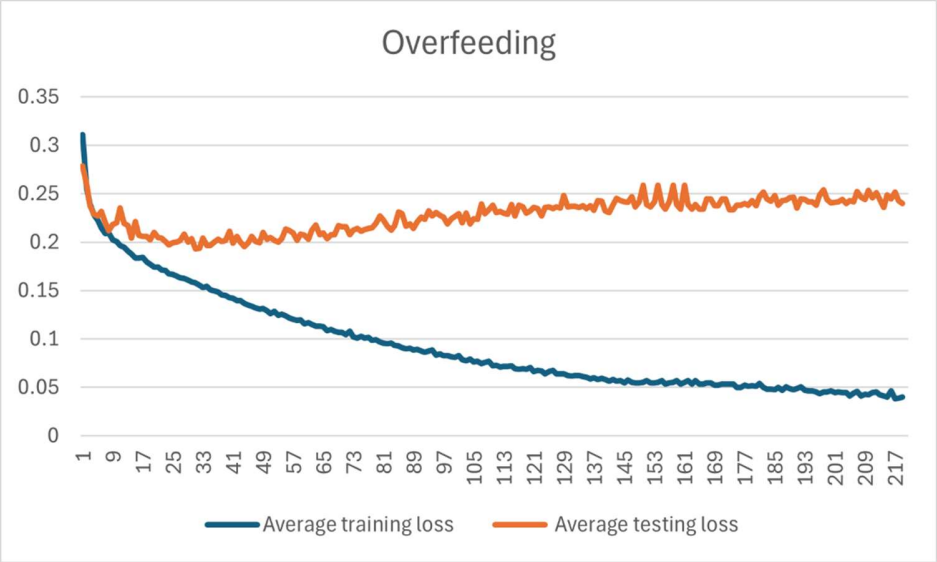
- SGD (Stochastic Gradient Descent) je základný optimalizačný algoritmus, ktorý aktualizuje váhy na základe náhodne vybraných mini-batcherov dát.
- SGD-momentum pridáva "momentum" z predchádzajúcich krokov, čím zvyšuje rýchlosť konvergencie
- Adam (Adaptive Moment Estimation) kombinuje výhody SGD a SGD-momentum.
  Používa adaptívne učenie(learning rate) pre každú váhu, sleduje prvý moment (moving average gradient descent) a druhý moment (moving squared average gradient descent)

Batch size je pri oboch porovnaniach rovnaká a to 32.

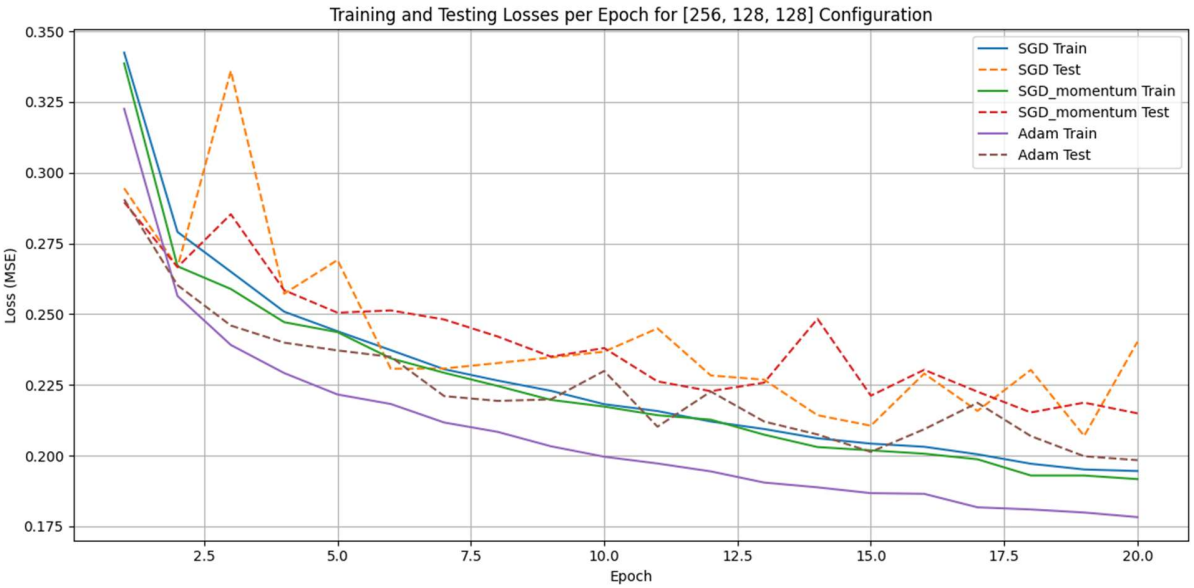
Na spustenie je nutné mať v aktuálnom pracovnom adresári housing.csv databázu

Najväčšiu úspešnosť som zaznamenal pri optializátore adam, avšak pri menších NN bol obšas relatívne úspešnejší aj SGD-momentum. Priemerná úspešnosť(loss) sa pohybovala okolo 0.3 - 0.2

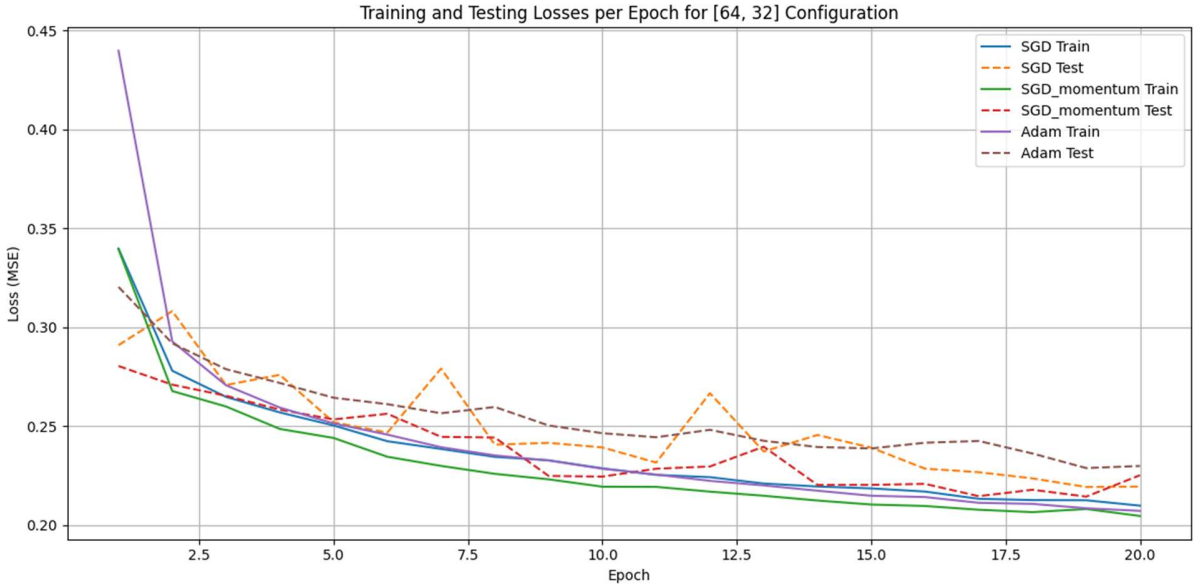
Počas iniciálneho trénovania sa mi stalo že kým som úplne nerozumel tomuto konceptu tak som sa snažil spraviť NN s čo najmenšou loss. Počas testovania NN som vytváral veľmi veľkú neurónovú sieť. Čudoval smo sa ako je možné že aj keď sa mi trénovací model čoraz viac a viac zlepšoval, testovací stále stagnoval, dokonca sa až zhoršoval. Potom som si uvedomil že sa moja NN učí výsledky z trénovacieho datasetu naspamäť (tzv. Overtraining/overfeeding)



Obr.2: Graf príkladu overtrainingu (model adam)



Obr.3: porovnanie straty modelov s väčšou NN



Obr.4: porovnanie straty modelov s menšou NN

1. Backpropagation Algorithm

Zadanie[1]**:**

Úloha:

V tejto úlohe budete implementovať plne funkčný algoritmus backpropagation, ktorý je kľúčovým komponentom tréningu neurónovej siete. Umožňuje jej učiť sa pomocou minimalizácie zadanej chybovej funkcie. Treba implementovať doprednú aj spätnú časť pre jednotlivé operátory a funkcie, ako aj aktualizácie parametrov siete. Algoritmus overíte natrénovaním jednoduchej doprednej neurónovej siete (viacvrstvového perceptrónu).

Zadanie:

Na riešenie úlohy použite knižnicu NumPy pre maticové a vektorové operácie. Použitie knižníc ako PyTorch a TensorFlow, ktoré obsahujú autograd je zakázané. Implementujte modulárnu architektúru, v ktorej bude možné jednotlivé moduly reťaziť. Implementácia bude obsahovať lineárnu vrstvu, aktivačnú funkciu sigmoid, tanh, relu a chybovú funkciu MSE (mean squared error).

Náčrt modelu:

Pre validáciu algoritmu použite XOR problém, použite dvojvrstvovú sieť, ktorá bude mať na skrytej vrstve 4 neuróny, na výstupnej vrstve 1 neurón a trénovať sa bude pomocou MSE chybovej funkcie. Rýchlosť učenia môže byť 0.1 až 0.01 a počet epoch by mal stačiť okolo 500.

Najskôr implementujte dopredný smer pre jednotlivé moduly a spojte ich do jedného modelu (zoznam modulov), ktorý vám zo vstupných údajov vráti výstup.

Začnite od MSE funkcie a pre každý modul spočítajte jeho deriváciu a implementujte spätný smer. Dajte si pozor, lebo pri počítaní derivácie je potrebné udržiavať aj niektoré výsledky z doprednéhо smeru. Moduly môžu mať svoj stav, alebo môžu byť aj bezstavové a výsledky môžu byť odložené niekde inde. To ponechávame na vašej voľbe.

Implementujte pravidlo pre aktualizáciu váh bez momentu a potom aj s momentom.

Volanie modelu by malo byť jednoduché, pre dopredný smer model.forward(vstup) a pre spätný smer model.backward(chyba) a následne volanie aktualizácie parametrov (či už to bude implementovať model, alebo iná samostatná trieda je opäť na vás).

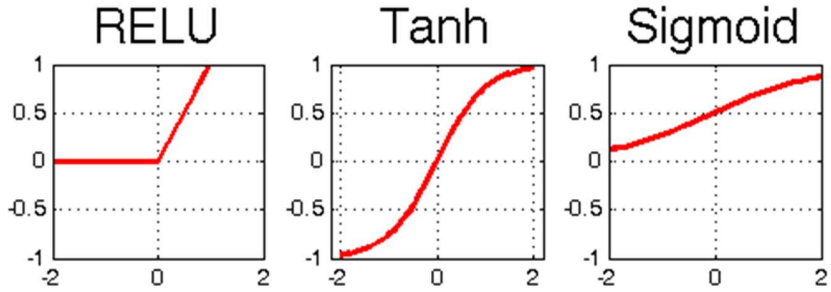
Riešnie:

Trénovanie prebieha iteratívne: vpred vypočítam výstup, spočítam chybu (Mean Squared Error), spätne odvodením (backpropagation) zistíme gradienty a aktualizujeme váhy.

Testoval som s 1 a 2 skrytými vrstvami (hidden layers), pri trénovaní som použil learning rate = 0.1 a momentum = 0.9 po 500 epoch. Moja strata sa po priemernom trénovaní limitne blíži nule.

Ako aktivačné funkcie používam ReLU, Tanh a Sigmoid:
Extrémne skrátene podané:

ReLU vracia 0 pre záporné hodnoty a lineárne rastie pre kladné.
Tanh sa mení od -1 po 1 a je symetricky(nepárne) okolo nuly.
Sigmoid sa limitne mení od 0 po 1 (s f(0) = 0,5) výsledkom je vždy kladné číslo.



Obr.5: porovnanie aktivaých funkcií[1]

ReLU a Tanh používam pri skrytých vrstvách pri výstupe používam sigmoid

Chybu počítam pomocou MSE Loss Function (Mean Squared Error): ktorú rátam aj pri forward aj pri backward procese. Ide o matematickú operáciu ktorá meria priemerný rozdiel na druhú medzi predikovanými a skutočnými hodnotami.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

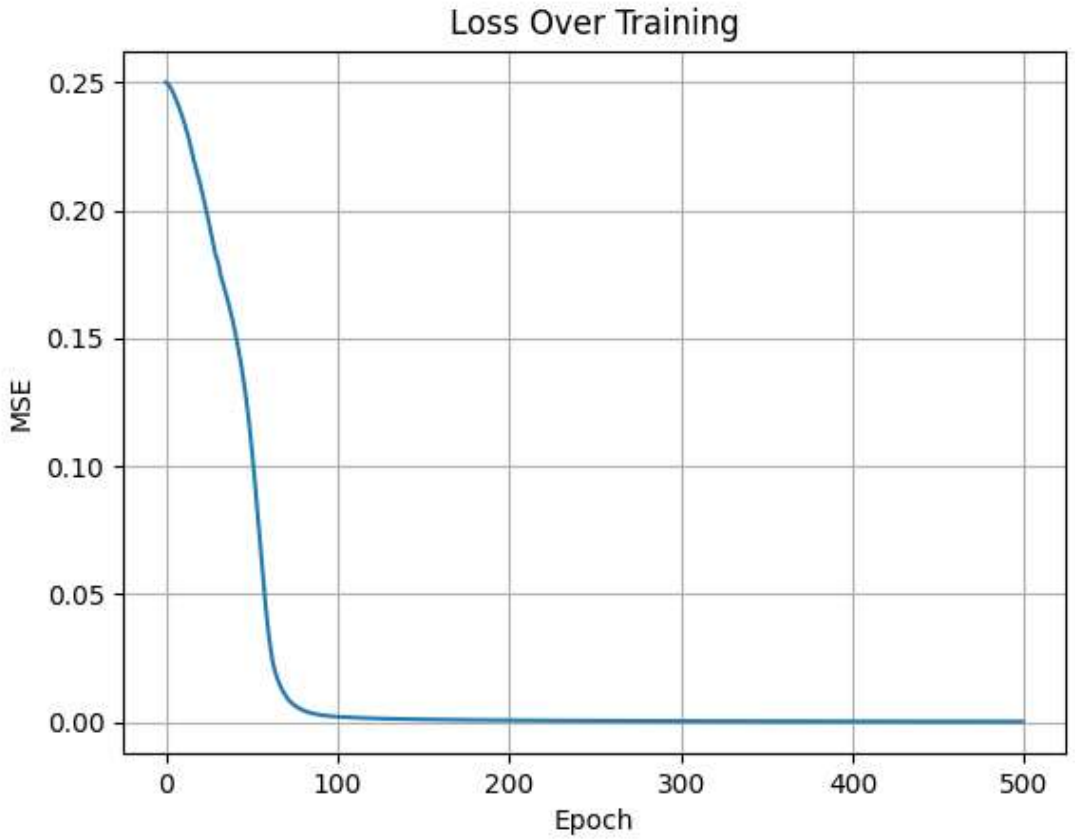
Obr.6: všeobecný zorec na výpočet MSE[2]

Skúšal som trénovať s momentum aj bez momenta, čo ovplyvnilo rýchlosť a stabilitu konvergencie

zdroje:

- 1 https://www.researchgate.net/figure/Activation-Functions-ReLU-Tanh-Sigmoid_fig4_327435257

- 2 https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-c7dde9a26b93/



Obr.6: loss počas trénovania mojej NN

```
Epoch 1/500 | Loss: 0.2346
Epoch 50/500 | Loss: 0.0052
Epoch 100/500 | Loss: 0.0013
Epoch 150/500 | Loss: 0.0009
Epoch 200/500 | Loss: 0.0006
Epoch 250/500 | Loss: 0.0005
Epoch 300/500 | Loss: 0.0004
Epoch 350/500 | Loss: 0.0003
Epoch 400/500 | Loss: 0.0003
Epoch 450/500 | Loss: 0.0003
Epoch 500/500 | Loss: 0.0002
Predicted:
[1 0 0 0]
Actual:
[1 0 0 0]
```

Obr.7:štandardný výstup pri trénovaní na AND problém

Záver

Pri porovnaní optializátorov v California housing problem som zistil že je pre môj prípad najefektívnejšie použiť adam optimalizér, s veľmi podobnými výsledkami dosiahnutými pmocou SGD-momentum.
Po spustení programu a následnom trénovaní sa uložia výsledky do .csv súboru a grafy ako .png súbory.

Pri backpropagation som úspešne implementoval vlastnú, relatívne presnú neuronovú sieť.
Po jej spustení sa taktiež ako v predošlej úlohe vytvára results.csv databáza s výsledkami a results.png graf loss funkcie.

Použité knižnice (California housing problem):

- *pytorch*
- *numpy* - *pre maticové operácie*
- *pandas* - *pre efektívnosť načítania údajov z .csv súborov*
- *matplotlib* - *pre tvorenie grafov*
- *csv* - *pre jednoduchší zápis do results.csv*
- *sklearn* -*StadardScaler & data split*

Použité knižnice (Backpropagation):

- *numpy* - *pre maticové operácie*
- *matplotlib* - *pre tvorenie grafov*

Obsah