Umelá inteligencia: Neurónové siete

Marián Kurčina

ID: 127211

xkurcinam@stuba.sk

Cvičenie: Streda 15.00 - 17.00

Obsah

MNIST klasifikátor

Zadanie

Úlohou je vytvoriť neurónovú site na klasifikáciu ručne písaných číslic z datasetu MNIST. Na riešenie použite doprednú neurónovú site (viacvrstvový perceptrón) a natrénujte ju pomocou algoritmov SGD, SGD s momentum a ADAM. Okrem trénovacej a testovacej chyby odmerajte aj presnosť modelu. Model by mal mať výslednú presnosť viac ako 97%. V úlohe použite knižnicu PyTorch.

Opis riešenia

Najprv si nahrám data do programu a transformujem ich aby mali hodnotu 0-1. Nasledovne ich rozdelím na dve časti, jedna časť bude trénovacia a jedna testovacia. Po získaní a upravení datasetu budem model trénovať a nasledovne testovať postupne pre každý algoritmus.
Trénovanie bude prebiehať niekoľko epoch, v každej epoche po banchoch vpustím obrázky číslic do modelu a získam predikciu. Z rozdielu predikcie a skutočného označenia číslice zistím gradiant, ktorý bude slúžiť na pretrénovanie siete a teda zmenu lineárnych vrstiev modelu. Spôsob menenia lineárnych vrstiev bude závisieť od algoritmu trénovania.
Po každej epoche vypíšem priemernú odchylku počas trénovania a model otestujem. Informácie o teste si uložím a pokračujem nasledujúcou epochou. Po vykonaní predom určenom počte epoch model posledný krát otestujem, toto je môj výsledný model. Nakoniec vykreslím grafy úspešnosti a trénovacích a testovacích chýb.

Opis kódu

Na začiatku mám načítanie transformovaného datasetu do train_dataset a test_dataset, prvá časť slúži na trénovanie modelu, druhá na jeho testovanie.
Ďalej si rozdelím dataset do batchov, ktoré budú v train_loaderi a test_loaderi, tieto loadery budú slúžiť na testovanie a trénovanie modelu.
Nasledovne začne proces trénovania a testovania, najprv testujem pomocou SGD – potrebujem parameter learning_rate, potom pomocou SGD s momentum - potrebujem learning_rate a momentum a ako posledný využijem ADAM – potrebujem parameter learning rate.
Samotné trénovanie začne vytvorením si modelu, ktorého vrstvy mám zadefinované v kóde.
Pre každú epochu najprv resetnem gradianty, zistím output modelu, loss a s lossom vykonám backward funkciu po ktorej môžem zmeniť váhy vineárnych vrstiev pomocou daného optimizeru (SGD/SGD_momentum/ADAM). Output modelu bude vždy uvedený ako pravdepodobnosť, že ide o dané číslo. Pre každú epochu si program uloží trénovaciu chybu a po trénovaní so všetkými obrázkami datasetu otestuje model pomocou datasetu test a uloží si úspešnosť správnej predikcie a taktiež testovaciu chybu. Tieto informácie na konci zobrazí v grafe a presnosť modelu bližšie ukáže cez confusion maticu.

Optimalizácia parametrov pre trénovacie algoritmy

Ako prvé som sa rozhodol nájsť najlepšie parameter pre jednotlivé trénovacie algoritmy, learning rate a momentum pri 30 epochách.

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

Tabuľka 1: Výsledky testovania optimálnej hodnoty learning rate pre trénovacie algoritmy SGD a 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%

Tabuľka 2: Výsledky testovania optimálnej hodnoty learning rate a momentum pre trénovací algoritmus SGD s momentom

Z testovania som zistil, že najlepšie bude používať learning rate 0.1 pri SGD a SGD s momentom a pri ADAM 0.001. Momentum pri SGD s momentom bude 0.95.

Hľadanie optimálnych hyperparametrov modelu

Ako prvé som sa pokúsil o to, nájsť správne aktivačné funkcie, pričom som si povedal, že moja site bude 2 vrstvová, to znamená, že budem mať 2 aktivačné funkcie. Budem skúšať 3 rôzne aktivačné funkcie a to ReLu, Tanh a Sigmoid. Vyskúšam všetky kombinácie.

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

Tabuľka 3: Výsledky testovania všetkých kombinácií aktivačných funkcií, v stĺpci je prvá aktivačná funkcia v sieti, v riadku druhá, v tabuľke sú uvedené priemery pre všetky trénovacie algoritmy (learning rate = 0.1, momentum = 0.9, počet epoch = 30, prvá vrstva = 100, neurónov druhá vrstva = 50 neurónov)

Z výsledkov testovania som zistil, že bude najlepšie použiť aktivačné funkcie ReLU a Sigmoid. Ďalej som hľadal parametre lineárnych vrstiev a to počet neurónov v jednotlivých sieťach, s tým aby bola zachovaná čo najlepšia úspešnosť.

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

Tabuľka 4: Výsledky testovania kombinácií počtov neurónov vo vrstvách

Z testovania som zvolil, že najlepšie bude použiť 100 a 50 neurónov.

Architektúra optimálneho modelu

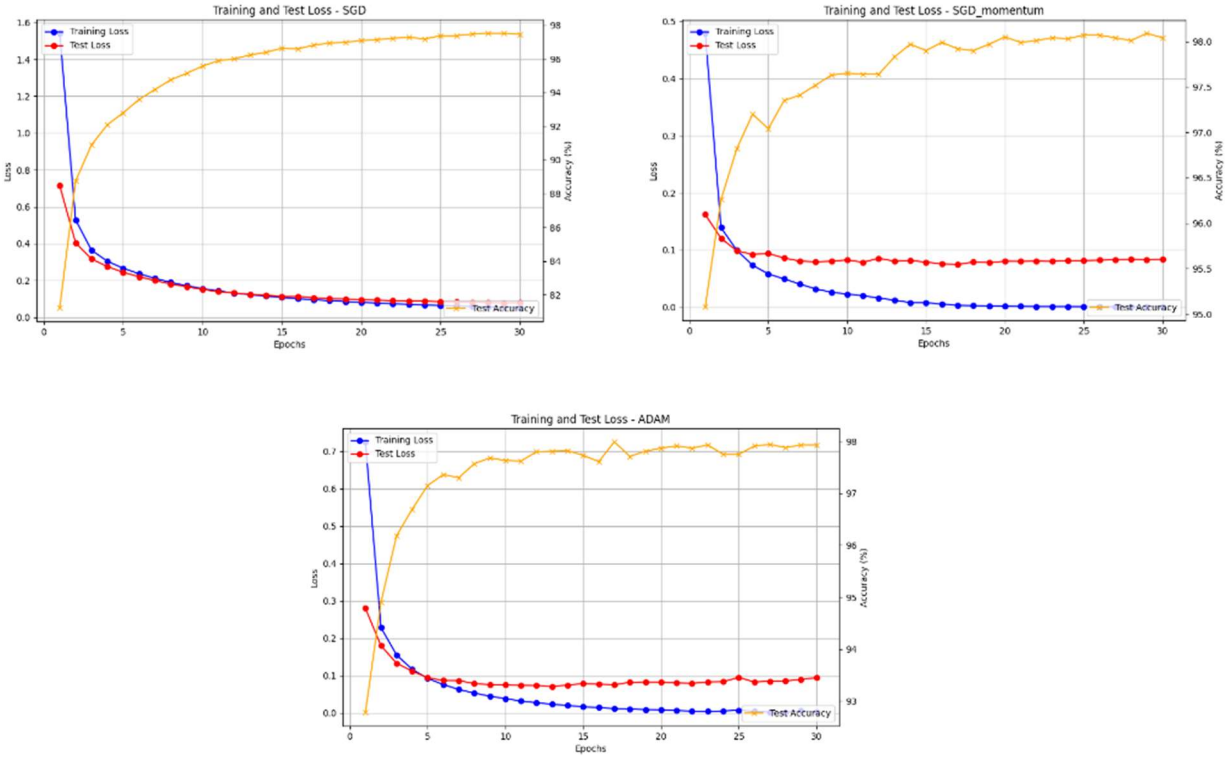
Moja sieť bude mať 2 skryté vrstvy, prvá bude mať 100 neurónov a druhá 50 neurónov, aktivačné funkcie budú ReLU a sigmoid.

Pri SGD je úspešnosť **97.46%**.
Pri SGD s momentom je úspešnosť **98.04%**.
Pri ADAM je úspešnosť **97.93%**.

Pre každý beh programu sa zobrazí confusion matica.

Predicted->	0	1	2	3	4	5	6	7	8	9
0	969	0	2	0	0	0	4	2	2	1
1	0	1130	1	1	0	0	0	0	3	0
2	5	1	1010	3	3	0	1	6	3	0
3	0	0	2	992	0	6	0	4	2	4
4	1	0	3	0	962	1	2	3	1	9
5	3	0	0	7	1	870	5	0	3	3
6	2	2	3	0	6	4	940	0	1	0
7	1	4	7	1	0	0	0	1003	3	9
8	2	0	3	5	4	2	1	2	950	5
9	2	3	1	5	9	2	1	4	4	978

Obrázok 1:Confusion matica



Obrázok 2:Grafy vývoju trénovacej a testovacej chyby a úspešnosti modelu

Záver

Podarilo sa mi naučiť sa pracovať s knižnicou PyTorch a vytvoriť program na klasifikáciu číslic z datasetu MNIST za pomoci neurónovej siete. Podarilo sa mi identifikovať optimálnu architektúru modelu a optimálne parametere pre trénovacie algoritmy algoritmy SGD, SGD s momentum a ADAM.

Backpropagation algoritmus

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: $\frac{\sum(predicted - ta\quad)^2}{output\ size}$ – priemerný rozdiel na druhú

Backward metóda: $\frac{\sum 2*(predicted - targ\quad)}{output\ size}$ – 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: $input * W + b$
Pri forward metóde bude input vynásobený maticou váh a bude k nemu pripočítaný vektor bias.

Backward metóda: $dW = grad * input^T$

db = grad

S momentom:

$$vW = momentum * vW - learning\ rate * dW$$

$$vb = momentum * vb - learning\ rate * db$$

$$W = W + vW$$

$$b = b + vb$$

Bez momentu:

$$W = W - learning\ rate * dW$$

$$b = b - learning\ rate * db$$

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: $\frac{1}{1+e^{-x}}$

Backward metóda: $grad * output * (1 - output)$

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: $\max(0, x)$

Backward metóda: $grad * (input > 0)$

Tanh

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

Forward metóda: $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

Backward metóda: $grad * (1 - output^{2})$

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

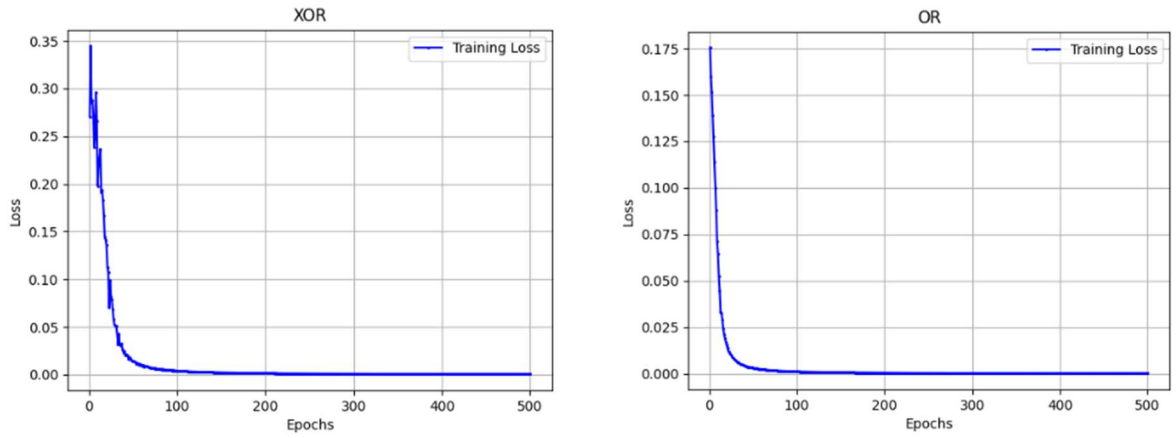
Ideálny model pre riešenie logických členov bez momentu je model s jednou skrytou vrstvou a za použitia Tanh aktivačnej funkcie.

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 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://eli.thegreenplace.net/2018/backpropagation-through-a-fully-connected-layer/
https://www.youtube.com/watch?v=Ilg3gGewQ5U