Praktická úloha riešená sieťou BP (klasifikácia)

Natália Holková (xholko02)

1 Úvod

Táto dokumentácia je písaná k projektu do predmetu SFC v akademickom roku 2021/2022. Cieľom tohto projektu bolo vyriešiť praktickú klasifikačnú úlohu sieťou využívajúcou *Backpropagation* algoritmus. Konkrétne bola vytvorená konvolučná neurónová sieť používajúca tento algorimus na klasifikáciu ručne písaných číslic. V implementácii bol použitý programovací jazyk *Python* a pre natrénovanie bol použitý dataset MNIST.

2 Algoritmus Backpropagation

Algoritmus Backpropagation je jeden z najdôležitejších algoritmov v oblasti strojového učenia a hlbokého strojového učenia. Je to algoritmus pre učenie, nastavovanie váh, acyklických a dopredných sietí. Podmienka jeho použitia je, že všetky neuróny siete musia mať spojité aktivačné funkcie, ktoré sú zderivovateľné. Cieľom je minimalizovať *objektívnu funkciu*, ktorá je funkciou všetkých váh siete a vyjadruje odchýľku odozvy siete od požadovaných hodnôt.

Backpropagation začína tak, že nastavuje náhodne váhy všetkým prepojeniam v sieti. Algoritmus potom iteratívne upravuje váhy v sieti tým, že ukazuje trénovacie dáta sieti. Váhy upravuje dovtedy, kým neurónová sieť nefunguje tak, ako je očakávané.[1]

Jadro celého algoritmu pozostáva z dvoch fáz, ktoré sa opakujú. V prvej fáze (známa aj ako *forward pass*) vstupné dáta sú vpustené do neurónovej siete a neurónové aktivácie postupne putujú celou sieťou až pokiaľ nie je vygenerovaný výstup.

Druhá fáza (známa ako *backward pass*) začína na výstupnej vrstve a postupuje naprieč sieťou opačným smerom až na vstupnú vrstvu. Tento spätný prechod začína tým, že sa pre každý neurón vo výstupnej vrstve vypočíta chyba. Táto chyba slúži na úpravu váh neurónov výslednej vrstvy. Následne sa chyba každého výstupného neurónu spätne posúva cez sieť na neuróny, ktoré sú naň pripojené. Celkový podiel neurónu na chybe sa sčíta a podľa toho sa upravia váhy. Obdobne tento proces pokračuje až na vstupnú vrstvu.

Pretože sa rieši klasifikačná úloha, kde máme > 2 triedy, použijeme nasledovnú objektívnu funkciu:

$$E(y, \hat{y}) = -\sum_{i} y_i * \log(\hat{y_i}) \tag{1}$$

kde y predstavuje skutočnú triedu a $\hat{y}$ predstavuje nami predikovanú triedu.[2]

3 Konvolučné neurónové siete

Konvolučné neurónové siete sú *acyklické neurónové siete*. Využívajú sa najmä k rozpoznávaniu a klasifikácii obrázkov, ktoré môžu byť buď čiernobiele (1 kanál) alebo farebné (3 kanály). Skladajú zo štyroch základných typov vrstiev[3]:

- konvolučné vrstvy (*convolutional layers*)
- aktivačné vrstvy (*activation layers*)
- zoskupujúce vrstvy (*pooling layers*)
- plne prepojené vrstvy (*fully connected layers*)

3.1 Konvolučná vrstva

Konvolučná vrstva je založená na malých filtroch (nazývaných aj *kernels*), ktoré sa sieť sama naučí. Tieto filtre sú použité pri operácii konvolúcie so vstupným obrázkom. Počíta sa ako skalárny súčin medzi zložkami filtrov a vstupu a výsledkom je 2D matica. Takýmto spôsobom sa sieť učí filtre, ktoré sa aktivujú pri špecifických vlastnostiach na danej pozícii v vstupe.[4]

Operáciu konvolúcie je možné popísať vzorcom:

$$f_l^k(p,q) = \sum_c \sum_{x,y} i_c(x,y) \cdot e_l^k(u,v) \tag{2}$$

kde $i_c(x,y)$ predstavuje vstupný obrázok, ktorý je po zložkách prenásobený filtrom $e_l^k(u,v)$ ($k-ty$ filter z $l-tej$ vrstvy). Konvolučná vrstva je definovaná najmä parametrami ako veľkosť filtra a veľkosť posunu filtra (*stride*).

3.2 Aktivačná vrstva

Aktivačné funkcie definujú výstup neurónov v sieti na základe vstupu. Pri konvolučných neurónových sieťach sa najčastejšie používajú:

- *ReLU* - aplikuje sa na výstup konvolučnej vrstvy[5]

$$f(x) = \begin{cases} x_i & pre\ x_i \geq 0 \\ 0 & pre\ x_i < 0 \end{cases} \tag{3}$$

- *Softmax* - používa sa na poslednej vrstve, normalizuje do intervalu $< 0, 1 >$[5]

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{4}$$

3.3 Zoskupujúca vrstva

Zmyslom zoskupujúcej vrstvy (*pooling*) je znížiť dimenzionalitu, čo zníži počet potrebných parametrov a tým pádom aj výpočtovú náročnosť celého modelu. Pooling pracuje nad 2D maticou a používa zvolenú funkciu na škálovanie vstupu. Najčastejším typom zoskupujúcej vrstvy je *max pooling*, ktorý vyučíva ako škálovaciu funkciu výber maxima.

Pooling vrstva je definovaná veľkosťou filtre (typicky 2x2) a posunom (*stride*). Týmto sa dosiahne zoškálovanie na 25% pôvodnej veľkosti.

3.4 Plne prepojená vrstva

Táto vrstva slúži na prevod vstupu v tvare matice na výstup v tvare vektora, ktorý môže už byť vstupom klasifikácie.[4]

3.5 Učenie konvolučných neurónových sietí

Existuje viacero spôsobov učenia konvolučných neurónových sietí, ako napríklad *Stochastic Gradient Descent (SGD)* a jeho modifikácie *Batch Gradient Descent (BGD)* a *Mini-Batch Gradient Descent (MBGD)*. Všetky tieto metódy využívajú algoritmus backpropagation. Pri implementácii bol použitý populárny optimalizačný algoritmus *Adam*, ktorý je tiež variantou SGD.

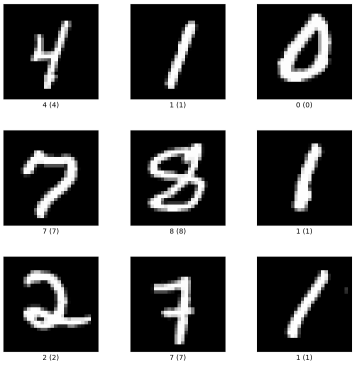
Zatiaľ čo klasický SGD má iba jeden konštantný parameter rýchlosti učenia (*learning rate*), Adam využíva hneď niekoľko. Kombinuje v sebe výhody algoritmu adaptívneho gradientu (*AdaGrad*), ktorý pomáha pri malých gradientoch, a tiež algoritmu *Root Mean Square Propagation (RMSProp)*, ktorý upravuje parameter rýchlosti učenia na základe posledných zmien váh.

4 Implementácia

V tomto projekte bola implementovaná konvolučná neurónová sieť, ktorá je schopná po naučení klasifikovať ručne písané číslice, teda predikovať, do ktorej triedy z 10tich číslic patria. Implementácia je v programovacom jazyku *Python*, bez akýchkoľvek knižníc pre strojové učenie. Využíva iba základné knižnice ako `numpy` na vektorové a maticové výpočty, `matplotlib` pre vykresľovanie priebehu učenia alebo `pickle` pre uloženie natrénovaného modelu.

Aplikácia funguje v dvoch režimoch:

- trénovanie neurónovej siete (`train.py`)
- testovanie výsledného modelu na časti testovacích dát (`test.py`)



Obr. 1: Ukážka MNIST datasetu

4.1 Dataset

Pre trénovanie konvolučnej neurónovej siete bol zvolený dataset *MNIST*[1], ktorý obsahuje čiernobiele obrázky ručne písaných číslic spolu s príslušnými anotáciami. Zahŕňa 60000 obrázkov na trénovanie a 10000 obrázkov na testovanie.

V súbore `cnn/util.py` bolo definovaných niekoľko funkcií, ktoré stiahnu dataset z internetu (`download_MNIST_zip`), rozbalia ho (`unzip_dataset`) a načítajú zvolený počet obrázkov spolu s popisom (`extract_data`, `extract_labels`).

4.2 Architektúra konvolučnej neurónovej siete

Je vytvorená samostatná trieda `Network` v súbore `cnn/cnn.py`, ktorá obsahuje definíciu architektúry konvolučnej neurónovej siete. Vzhľadom na fakt, že boli použité iba základné knižnice a výpočty nie sú zoptimalizované, bola volená iba minimálna štruktúra, ktorá by predstavila všetky zložky konvolučných neurónových sietí a zároveň sa dala trénovať v prípustnostnom čase. Architektúra preto vyzerá nasledovne:

1. *vstupná vrstva* príjmajúca čiernobiele obrázky veľkosti 28x28 px
2. *konvolučná vrstva* s 8 filtrami veľkosti 5x5 px
3. *ReLU* aplikované na výstup 1. konvolučnej vrstvy
4. *konvolučná vrstva* s 8 filtrami veľkosti 5x5 px
5. *ReLU* aplikované na výstup 2. konvolučnej vrstvy
6. *zoskupujúca vrstva* s *max* funkciou a 2x2 px filtrom
7. *plne prepojená vrstva* transformujúca vstup na vektor

[1] `https://deepai.org/dataset/mnist`

8. *ReLU* na výstup plne prepojenej vrstvy
9. *plne prepojená vrstva*
10. *softmax*

Pre každý typ vrstvy je vytvorená samostatná trieda v súbore `cnn/layers.py`. Tieto triedy obsahujú funkcie pre obe fázy algoritmu backpropagation: propagáciu vstupov na výstup (`feedforward`) a spätný výpočet podielov váh na chybe (`backpropagate`).

4.3 Parametre a priebeh učenia

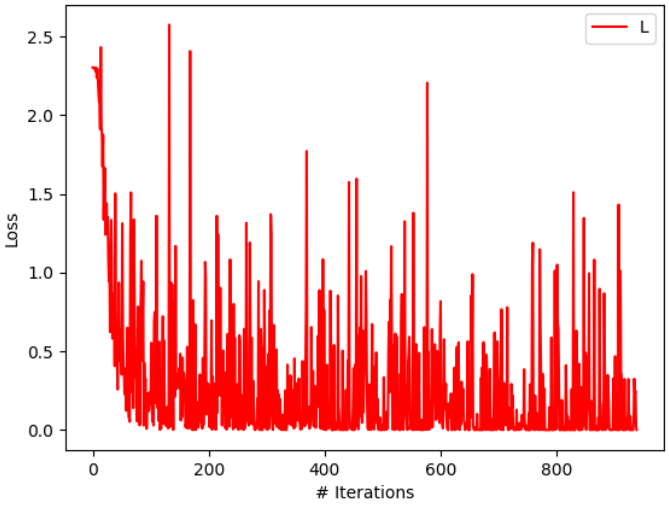
Model bol primárne trénovaný s hodnotou *learning rate = 0.01* na 5000 vstupných dátach po dobu 3 epóch pri *batch size = 16*. Všetky váhy, filtre, bias sú náhodne inicializované s normálnym rozložením. Učenie avšak prebieha veľmi dlhú dobu nakoľko nie je optimalizované pre využívanie GPU.

Nakoľko nie zavedená žiadna normalizácia, stáva sa, že dôjde k vybuchnutiu gradientov na výstupe. Toto znamená problém pre `softmax`, ktorý vyhodnotí e^x ako `nan`. Bolo preto zavedené najskôr posunutie hodnôt o najväčšiu hodnotu, až nasledovne samotný výpočet softmax.

Samotné učenie siete prebieha po epochách. Na začiatku každej epochy sa trénovacie dáta zamiešajú pre zníženie šance na pretrénovanie, a sú rozdelené na časti o rovnakej veľkosti (*batches*). Nad každou skupinkou vstupných obrázkov sa vykoná *Adam gradient descent*.

Adam iteruje cez dáta kde vykonáva zároveň *forward pass* aj *backward pass* fázy algoritmu backpropagation. Týmto získava gradienty a hodnotu objektívnej funkcie, ktoré sa neskôr sčítajú pre celý *batch*. Úprava váh, filtrov a bias nastáva iba raz pre každý *batch*.

Na obrázku 2 je zobrazený priebeh učenia siete a hodnota loss funkcie pri jednotlivých iteráciách:



Obr. 2: Loss funkcia v závislostí na iteráciách

4.4 Testovanie modelu

Po natrénovaní modelu je možné ho otestovať otestovať na testovacích dátach z MNIST datasetu a vyhodnotiť presnosť jeho predikcií. Model najskôr načíta (`load_params`) naučené váhy, filtre a bias, ktoré sa ukladajú do `.pkl` súboru. Pri evaluácii modelu (`predict`) sa už vykonáva iba *forward pass* nakoľko už nedochádza k úprave váh. Úspešnosť modelu je počítaná ako percentuálny podiel správnych predikcií z celkového počtu testov.

Model trénovaný na 5000 vzorkoch po dobu 3 epóch, ktorý bol testovaný na 1000 testovacích čísliciach dosiahol presnosť 96%.

5 Spustenie aplikácie

Aplikácia je spustiteľná na referenčnom serveri Merlin alebo aj akomkoľvek stroji pomocou `venv` a požiadavkov definovaných v `requirements.txt`

5.1 Príklad spustenia

Na stroji, ktorý nespĺňa požiadavky knižníc, je potrebná najskôr inštalácia pomocou:

```
python3 -m venv ./env-sfc
source ./env-sfc/bin/activate
pip3 install -r requirements.txt
```

5.1.1 Trénovanie

Trénovanie konvolučnej neurónovej siete možno spustiť príkazom v tvare:

```
python3 train.py [-s uloženie modelu] [-n počet dát] [-e epochy] [-b batch]
```

Konkrétne napríklad:

```
python3 train.py -s model.pkl -n 500 -e 5 -b 32
```

Kvôli dlhej dobe trénovania je už priložený natrénovaný model `model.pkl`, ktorý bol trénovaný na 5000 vzorkoch po dobu 3 epóch.

5.1.2 Testovanie

Testovanie natrénovaného modelu možno spustiť príkazom v tvare:

```
python3 test.py [-m cesta k modelu] [-n počet dát]
```

Konkrétne napríklad:

```
python3 test.py -m model.pkl -n 10
```

6 Záver

V projekte bola vytvorená funkčná konvolučná neurónová sieť využívajúca algoritmus backpropagation. Implementácia avšak nie je vhodná na reálne použitie kvôli dlhej dobe potrebnej na trénovanie a absencii normalizačných techník. Napriek tomu sa dá využiť ako ukážka praktickej klasifikačnej úlohy riešenej algoritmom backpropagation.

Referencie

[1] J.D. Kelleher. *Deep Learning*. MIT Press Essential Knowledge series. MIT Press, 2019. ISBN: 9780262537551. URL: https://books.google.sk/books?id=1wICwQEACAAJ.

[2] Kiprono Elijah Koech. *Cross-Entropy Loss Function*. Ed. Towards Data Science. cit. 2021-12-05. Okt. 2020.

[3] Zbořil František V. *4. Neocognitron a konvoluční neuronové sítě*. https://www.fit.vutbr.cz/study/courses/SFC/private/20sfc_4.pdf. Accessed: 2021-12-05. 2020.

[4] Keiron O'Shea a Ryan Nash. "An Introduction to Convolutional Neural Networks". In: *arXiv.org* (2015). ISSN: 2331-8422. URL: http://search.proquest.com/docview/2083864399/.

[5] Chigozie Nwankpa et al. "Activation Functions: Comparison of trends in Practice and Research for Deep Learning". In: *arXiv.org* (2018). ISSN: 2331-8422. URL: http://search.proquest.com/docview/2131541613/.