VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Soft Computing – technická zpráva

Praktická úloha řešená sítí s pomocí algoritmu backpropagation – klasifikace

26. listopadu 2022 David Hudák, xhudak03

1 Úvod a teorie

Jako téma semestrální práce v předmětu Soft Computing (dále jako SFC) jsem si zvolil zadání týkající se řešení praktických úloh s pomocí algoritmu backpropagation za účelem klasifikace dopředných neuronových sítí. V rámci řešení projektu jsem pak zvolil primárně dvě hlavní úlohy – identifikace prvku s největší četností v datasetu a klasifikace obrázků MNIST[1]. Zdroji pro řešení projektu byly přednášky a materiály v předmětech SFC, SUI a IZU a kniha zabývající se hlubokým učením od Goodfellowa a dalších [1].

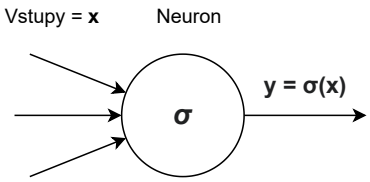
Výstupem tohoto projektu je jednoduchý framework pro tvorbu neuronových sítí s libovolnými aktivačními a objektivními funkcemi s několika natrénovanými sítěmi a jednoduché uživatelské rozhraní pro tvorbu a spouštění sítí nad vybranými úlohami. V případě vhodně zvolené architektury lze řešit i jiné typy úloh než klasifikace (například regrese).

1.1 Klasifikace

Jednou ze základních disciplín obecně strojového učení jsou klasifikační úlohy. Ty se řadí mezi takzvané úlohy s učitelem, kdy vstupem úlohy jsou jak samotná data, tak i jejich správné klasifikace (labely). Cílem takovéto úlohy je na základě těchto dat správně identifikovat (generalizovat) nová vstupní data. Při učení tak máme za cíl minimalizovat nějakou objektivní/chybovou funkci s ohledem na trénovací sadu, neboli zjednodušeně se snažíme identifikovat co nejvíce vstupů správně.

1.2 Neuronová síť

Základním stavebním kamenem moderního strojového učení jsou neuronové sítě založené na jednoduchém matematickém modelu neuronu. Ten se skládá ze své bázové funkce (výpočet vstupu x ze vstupů) a aktivační funkce viz obrázek 1. Tyto neurony se pak uspořádávají do vrstev, kdy v dopředných sítích jsou vstupem každého neuronu pouze výstupy neuronů z vrstvy předchozí a posuvy (biasy, učené hodnoty, které se přičítají) a výstupy neuronů směřují pouze do vrstev bezprostředně následujících.



Obrázek 1: Příklad obecného neuronu, který ze vstupních hodnot vytvoří skalár x, který upraví aktivační funkcí σ a pošle hodnotu do dalších vrstev.

Dopředná síť pak zvládá dvě hlavní funkce – takzvanou dopřednou (forward) propagaci, která postupně počítá ze vstupů vrstev jejich výstupy a výsledky posílá dál až na výstup, a funkci učení. Proces učení lze implementovat mnoha různými způsob, ale dnes se setkáváme primárně s algoritmem backpropagation v kombinaci s gradientním sestupem, jehož princip bude popsán v následující podsekci. Cíl tohoto algoritmu je výpočet gradientu pro učení vah a biasů, které se používají pro přidávání/odebírání významu v rámci propojení mezi jednotlivými vrstvami. Učení se může týkat ještě takzvaných konvolučních jader, které ale nejsou součástí tohoto projektu.

Je vhodné zmínit, že při práci s neuronovými sítěmi se můžeme setkat s různými typy vrstev. Hovoříme o takzvaných vstupních, skrytých a výstupních vrstvách. Rozdíly jsou v tom, že

[1] http://yann.lecun.com/exdb/mnist/

vstupní vrstvy pouze předávají svoji vstupní hodnotu (například hodnotu pixelu), výstupní vrstvy aplikují jiné aktivační funkce na své vstupy (například softmax) a skryté provádí postup uvedený v předchozích částech tohoto textu.

1.3 Backpropagation

Cílem optimalizačních algoritmů je minimalizovat nějakou chybovou funkci a backpropagation jde na to z pohledu matematické analýzy s principem počítání gradientů. V případě neuronových sítí se jedná o vektor hodnot příslušících každé váze a každému biasu, který říká, kterým směrem jít pro dosažení největšího růstu. Tedy pokud budeme mít například dvě váhy w_1 a w_2 a k nim gradient $\Delta w = [1, 2]$, pak víme, že ve směru $[1, 2]$ nejvíce rostou jejich váhy (a v opačném směru nejvíce klesají).

Pokud takto spočítáme gradient, pak ho můžeme předat nějakému algoritmu, který tyto gradienty použije k příslušné úpravě vah a biasů (v našem případě klasickému gradientnímu sestupu). To bez použití optimalizátorů učení a dalších úprav obvykle vypadá následovně:

$$\mathbf{w}_{new} = \mathbf{w}_{old} - \mu\Delta\mathbf{w}$$

kde θ označuje staré a nové nastavitelné parametry, μ označuje koeficient učení a $\Delta\theta$ gradient [1].

Požadovaný gradient pak algoritmus backpropagation počítá dvěma průchody neuronovou sítí – dopředným, kdy se vypočítají excitace a výstup, a zpětným, který přebírá data z dopředné propagace, vypočítá chybu a následně prochází síť zpětně. Při zpětném průchodu se pro každou vrstvu spočítá její gradient, který se následně použije k úpravě vah a biasů.

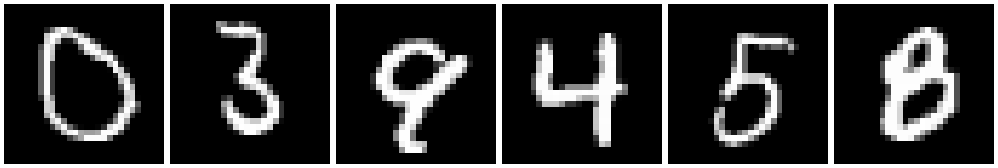
Pro výpočet gradientu obecně používáme takzvané delta pravidlo:

$$\Delta w_{ji}^l = \mu\delta_j^l x_i^l$$

kde Δw_{ji}^l je derivace váhy z i-tého neuronu předchozí vrstvy do j-tého neuronu aktuální vrstvy l, μ koeficient učení, δ_j^l propagovaná chyba z aktivační funkce neuronu a x_i^l hodnota přicházející do neuronu přes danou váhu. Hlubší rozbor je probírán na přednáškách.

1.4 Datové sady

Klasifikační úlohy se neomezují na žádnou konkrétní úlohu, kterou by mohly řešit – problémem bývá spíše dostupnost dat. Obecně je potřeba totiž jak vstupní data, tak i jejich požadované výstupy. V tomto projektu jsem zkoušel uměle tvořit datovou sadu (hledání nejčastější hodnoty v datasetu, XOR problém) a také použít již vytvořenou datovou sadu MNIST (viz 2).



Obrázek 2: Příklady vstupních obrázků z datové sady MNIST.

2 Technické detaily

2.1 Popis architektury

Klíčová část řešení spočívá ve dvou třídách NNET a Layer, kdy první zmíněná třída implementuje hlavní funkce neuronové sítě (učení a klasifikace) a v sobě obsahuje seznam objektů třídy Layer.

Třída Layer pak zapouzdřuje jak plně propojené vrstvy, tak aktivační vrstvy – toto rozlišení pak znamená, že síť se dvěma vrstvami teoretickými má vnitřně vrstvy čtyři. Rozdělení bylo zvoleno z důvodu snadnější implementace zpětné propagace. Vstupní vrstva není reprezentována vrstvou a na váhy „druhé“ vrstvy jsou tak pouze napojeny hodnoty vstupu.

Každá fyzická vrstva v sobě pak obsahuje krátký řetězec obsahující typ vrstvy (plně propojená/aktivační) a potřebné atributy. V případě plně propojené váhy a biasy, v případě aktivační vrstvy pak obsahuje atribut odkazu na aktivační funkce a atribut odkazu na její derivaci. Oba typy si pak pamatují své poslední vstupy a výstupy.

Obě třídy pak ještě spoléhají na soubor, kde jsou implementace všech používaných funkcí pro neuronové sítě (více v podsekci 2.3). Výsledek zaobalují skripty tools.py a main.py, které obsahují pomocné skripty pro inicializace dat a sítí a implementaci jednoduchého uživatelského rozhraní. Datasety i naučené sítě pak jsou ukládány ve formě pickle souborů, data pro testování majority jsou generována za běhu programu.

2.2 Technické minimum

Pro správné fungování projektu je nutné mít Python alespoň ve verzi 3.8 s balíky Numpy, Matplotlib a Pickle (dostupné na Merlinovi). V případě zájmu o stažení kompletního datasetu MNIST je nutné mít navíc balík Keras s balíkem Tensorflow (na Merlinu by měly být dostupné), které se používají v load_mnist.py. Staženými soubory test_x.pkl a další je pak nutno nahradit soubory ve složkách *test* a *train*. Preferovaná varianta je používat omezenou datovou sadu poskytnutou v rámci programu.

2.3 Podporované funkce a vrstvy

Při zpracování projektu bylo nutné využít několik dvojic funkcí a jejich derivací, které je nyní možné použít při inicializaci neuronové sítě. Jedná se o aktivační funkce: ReLU, tanh, sigmoid, softmax. Pro učení mohou být použity objektivní funkce střední kvadratická chyba a křížová entropie.

2.4 Řešené problémy učení

Při řešení projektu bylo nutné řešit situace, kdy se nedařilo učit, jak by bylo očekáváno. Jednak se ukázalo jako klíčové používat objektivní funkci křížové entropie namísto kvadratické chyby, což je ale u klasifikačních sítí poměrně běžné. Dále bylo problematické určení počátečních vah – na výběr byly varianty náhodných hodnot v intervalu $[-0.5, 0.5]$ a v intervalu závisejícím na počtu vstupů n $[\frac{-3}{\sqrt{n}}, \frac{3}{\sqrt{n}}]$. Jako lepší se ukázala první varianta, která však u některých situací vede k přetečení hodnot (viz 4.3). Druhé variantě se naopak příliš často stával velmi pomalý start učení, který vedl k jejímu zamítnutí.

Další možnou podporou bylo podpořit učení nějakými optimalizátory učení. Z existujících byl z důvodu jednoduchosti aplikován algoritmus momenta, ale ukázal se jako neefektivní a z výsledného řešení byl odstraněn. Dále bylo experimentováno s postupným snižováním koeficientu učení, ale to také nevedlo k lepšímu řešení.

Třetí, nakonec implementovanou variantou vylepšení učení, bylo rozdělování trénovacích dat do menších skupin – algoritmus zpětné propagace v tomto projektu prochází epochami, přičemž v každé se aplikuje pro každý jeden vzor algoritmus backpropagation. Přidané vylepšení pak spočívalo v tom, že se zmenšil celkový počet epoch pro menší skupinu dat s tím, že se trénovací data postupně střídají. U tohoto řešení se ukázalo, že na stejném množství trénovacích dat se dařilo dosáhnout i o několik procent lepších výsledků.

3 Návod k použití

Projekt lze používat dvěma způsoby:

- Naimportováním modulů nnet a functions do vlastního skritpu (není součástí návodu). Funkcionalita spočívá v inicializaci třídy NNET a volání jejích metod predict a learn.
- Zavoláním `$ python3 main.py` pro spuštění uživatelského rozhraní (primární).

Při spuštění hlavního skriptu se uživatel dostává do jednoduchého uživatelského rozhraní, kde si volbou přes klávesnici lze vybírat různé aktivity. Lze načítat či tvořit vlastní sítě, načíst testovací dataset, trénovat, predikovat. V případě vlastních dat lze vytvořit dvě dvojice souborů (trénovací a testovací) obsahující data ve formě posloupností čísel (od -1.0 do 1.0) o velikosti požadovaného vstupu/výstupu. Příklad zápisu pro XOR problém lze nalézt ve složce *train*. V rámci natrénovaných sítí je poskytnuta jedna menší ReLU síť pro klasifikaci MNIST obrázků. V případě tvorby vlastní sítě je možné si zvolit počet vrstev, počty neuronů a aktivační a objektivní funkce.

Pro experimentování se sítěmi lze využít buď predikování nad testovacím datasetem, nebo lze danou síť učit. Možností je například si na základě čísla vybrat n-té trénovací dato, kterému se vypíše (vykreslí) vstup a síť na výstupu vypíše potenciální klasifikační třídy, případně lze agregačně testovat na všech testovacích datech. Pro učení se v programu nachází nabídka na počet obrázků z trénovacího setu, počet iterací pro set a dotaz na rozdělení vstupu na podskupiny s příslušným periodickým rozdělením (ukázalo se jako někdy výhodné).

4 Experimentování

V této sekci jsou popsány pozorované výsledky na dříve uvedených úlohách a některé vedlejší poznatky získané z experimentování.

4.1 Vybrané výsledky

V této podsekci je uvedena tabulka natrénovaných sítí na MNISTu. Sítě byly z dokumentačních důvodů tvořeny se stejně velkými skrytými vrstvami v rámci sítě. V rámci odevzdaného archivu se kvůli velikosti nachází pouze jedna menší síť (*nnets/mnist1x128.pkl*) s jednou vrstvou a aktivační funkcí ReLU. Všechny sítě byly trénovány na 90 epochách na vzorku 2400 trénovacích dat, všechny sítě používaly jako objektivní funkci křížovou entropii v kombinaci s funkcí výstupní soft max na výstupu. Testy proběhly na kompletním testovacím MNIST datasetu (10 tisíc obrázků). Více detailů viz tabulka 1.

Aktivační vrstvy	Počet vrstev	Velikost vrstvy	Úspěšnost
ReLU	2	256	92.19 %
tanh	2	256	87.47 %
sigmoid	2	256	89.38 %
ReLU	1	128	88.52 %

Tabulka 1: Úspěšnost různých neuronových sítí na MNIST datasetu.

Experimentováno bylo ještě se zmíněnou majoritou, ale primárním výstupem práce jsou sítě natrénované a testované na MNIST datasetu.

4.2 Výběr parametrů sítě a učení

Při experimentování se ukázalo, že koeficient učení by měl být oproti původním očekáváním poměrně malý, tedy maximálne zhruba 0.01, ale obvykle bylo vhodné použít i řádově menší.

Také se ukázalo, že čím hlubší síť, tím komplikovanější učení – mnohdy širší a méně hluboká sítě dosahovaly lepších výsledků než sítě s více vrstvami.

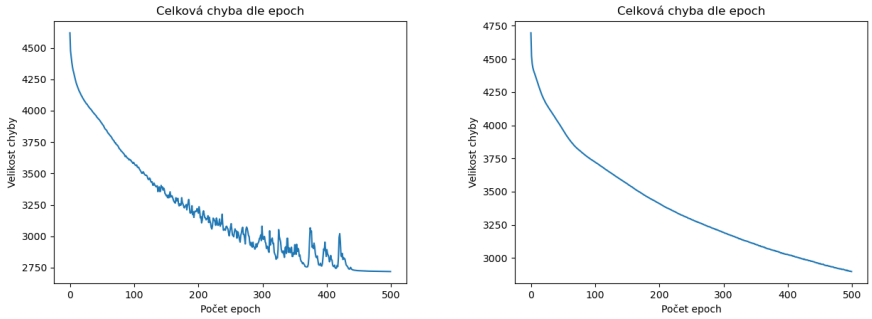
Co se týká počtu vzorků, tak se ukázalo, že byť je zásadní poskytnout dostatečné množství trénovacích dat, jejich vliv postupně klesá. U MNIST databáze se tak empiricky ukázalo, že vzít 2400 obrázků je poměrně dostačující množství (maximum v neredukované sadě je 60000, v odevzdaném archivu je pouze redukovaná). Větší množství by samozřejmě vedlo ještě k lepším výsledkům, ale rozdíl by pravděpodobně nebyl řádový.

4.3 Problém přetečení

Při některých experimentech dochází k tomu, že kvůli kombinaci náhodně zvolených vah dojde[2] k přetečení maximální povolené hodnoty v datovém typu float. Tento problém je ošetřen znovuinicializováním sítě s nižšími hodnotami vah (děje se automaticky), případně změnou architektury či přidáním aktivační funkce tanh, která mnohem lépe usměrňuje extrémní hodnoty než například ReLU.

4.4 Nestabilní učící křivka

Jeden z efektů, který se děl při některých trénováních, byl vznik zubů na učící křivce – chyba v některých momentech přestala monotónně klesat a začala se naopak v některých učících iteracích zvyšovat. K tomuto efektu jednak docházelo při příliš vysokém koeficientu učení, jednak obecně při zvětšování množství vrstev v sítí. Příklad tohoto efektu můžeme vidět na obrázku 3.



Obrázek 3: Záškuby při učení pětivrstvé sítě se 1 tanh, 3 ReLU a 1 softmax vlevo proti čisté křivce při učení třívrstvé sítě s 1 tanh, 1 ReLU a 1 softmax vrstvou vpravo. Klasifikace většiny.

5 Závěr

Úspěšně se zdařilo vytvořit jednoduchý framework pro tvorbu (klasifikačních) neuronových sítí. Také se zdařilo vytvořit poměrně kvalitní sítě pro klasifikaci obrázků z databáze MNIST s až 92% úspěšností. Dále se v rámci řešení projektu zdařilo vyzkoušet některé užitečné principy učení neuronových sítí.

Reference

[1] Goodfellow, I. J.; Bengio, Y.; Courville, A.: *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, `http://www.deeplearningbook.org`.

[2] Primárně v rámci funkce softmax, kde se nachází exponenciální funkce