VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

KONVOLUČNÍ NEURONOVÉ SÍTĚ

CONVOLUTIONAL NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. ZUZANA LIETAVCOVÁ**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. FRANTIŠEK V. ZBOŘIL, CSc.**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Lietavcová Zuzana, Bc.**
Obor: Inteligentní systémy
Téma: **Konvoluční neuronové sítě**
ConvolutionNeural Networks
Kategorie: Umělá inteligence

Pokyny:

1. Prostudujte teorii konvolučních neuronových sítí.
2. Navrhněte programové řešení konvoluční neuronové sítě, které by bylo vhodné pro výuku, tj. pro demonstraci činnosti jednotlivých vrstev sítě (především konvolučních, aktivačních a seskupujících (pooling)).
3. Navržený program implementujte.
4. Navrhněte množinu trénovacích dat, na kterých budete síť učit, opět s důrazem na demonstraci činnosti a učení sítě.
5. Na navržené množině ověřte funkčnost navržené sítě a zhodnoťte její demonstrační vlastnosti.

Literatura:

- Quoc V. Le: A Tutorial on Deep Learning, 2015, http://ai.stanford.edu/~quocle /tutorial2.pdf
- Britz, D.: Understanding Convolutional Neural Networks for NLP, 2015, http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/
- Ujjwalkarn: An Intuitive Explanation of Convolutional Neural Networks, 2016, https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
- Karpathy, A.: Convolutional Neural Networks for Visual Recognition, 2017, http://cs231n.github.io/convolutional-networks/

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zbořil František V., doc. Ing., CSc.**, UITS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Táto práca sa zaoberá problematikou učenia konvolučných neuronových sietí. Ide o druh hlbokých neuronových sietí, ktoré sa v súčasnosti hojne používajú predovšetkým v oblasti rozpoznávania obrazu a spracovania prirodzeného jazyka. Práca popisuje špecifiká konvolučných neuronových sietí oproti tradičným neuronovým sieťam a zameriava sa na vnútorné výpočty, ktoré realizujú pri učení. Konvolučné neurónové siete sa typicky skladajú z niekoľkých typov vrstiev neurónov a cieľom práce je demonštrovať výpočet jednotlivých typov vrstiev. V práci bol navrhnutý a implementovaný demonštračný program učenia jednoduchej konvolučnej siete s využitím vlastnej implementácie neurónovej siete. Správnosť implementácie bola overená natrénovaním siete pre riešenie klasifikačnej úlohy, boli vykonané experimenty s rôznymi architektúrami sietí a ich výsledky porovnané.

Abstract

This thesis deals with convolutional neural networks. It is a kind of deep neural networks that are presently widely used mainly for image recognition and natural language processing. The thesis describes specifics of convolutional neural networks in comparison with traditional neural networks and is focused on inner computations in the process of learning. Convolutional neural networks typically consist of a different types of layers of neurons and the core part of this thesis is to demonstrate computations of individual types of layers. Learning demonstrating program of a simple convolutional network was designed and implemented using own implementation of neural network. Validity of the implementation was tested by training models for solving a classification task. Experiments with different types of architectures were conducted and their performance was compared.

Klíčová slova

hlboké učenie, konvolučné neuronové siete, zostup po gradiente, spracovanie obrazu, klasifikácia

Keywords

deep learning, convolutional neural network, gradient descent, image processing, classification

Citace

LIETAVCOVÁ, Zuzana. *Konvoluční neuronové sítě*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. František V. Zbořil, CSc.

Konvoluční neuronové sítě

Prohlášení

Prehlasujem, že som túto diplomovú prácu vypracovala samostatne pod vedením pana Doc. Ing. Františka Zbořila, CSc. Uviedla som všechny literárne pramene a publikácie, z ktorých som čerpala.

. . . . . .
Zuzana Lietavcová
26. července 2018

Poděkování

Chcela by som poďakovať svojmu vedúcemu práce Doc. Ing. Františkovi Vítězslavovi Zbořilovi, CSc. za odborné vedenie práce, rady a ústretovosť pri jej riešení.

Obsah

Kapitola 1

Úvod

Strojové učenie predstavuje riešenie problémov, ktorých riešenie je veľmi zložité naprogramovať v programovacom jazyku, avšak pre človeka je jednoduché a intuitívne. Princíp strojového učenia spočíva v tom, že necháme počítač naučiť sa riešiť problém namiesto toho, aby sme mu dodali presný postup ako sa k riešeniu dopracovať. Počítač sa teda učí daný koncept na základe skúseností. Z jednoduchších konceptov je potom možné vytvoriť hierarchiu, na základe ktorej sa počítač môže naučiť zložité koncepty. Ak by sme nakreslili graf zobrazujúci ako sú tieto koncepty postavené jeden na druhom, videli by sme, že graf je hlboký, s mnohými vrstvami. Z tohto dôvodu nazývame tento prístup k umelej inteligencii ako hlboké učenie. [1] Príkladom úloh, ktoré sa riešia pomocou hlbokého učenia je napríklad rozpoznávanie objektov na obrázku, klasifikácia, preklad textu z jedného jazyka do iného a mnoho ďalších. V súčasnosti sú veľmi populárnym a skúmaným druhom hlbokých neronových sietí konvolučné neurónové siete (označované skratkou CNN). Cieľom tejto práce je navhnúť a implementovať konvolučnú neuronovú sieť vrátane algoritmu učenia a natrénovať ju pre riešenie úlohy klasifikácie obrázkov. Dôraz je kladený na vlastnú implementáciu algoritmu učenia bez použitia už implementovaných metód z frameworkov a na demonštračnú funkciu programu. Program by mal slúžiť na lepšie pochopenie fungovania konvolučných neuronových sietí pri ich štúdiu.

Prvá kapitola predstavuje problematiku strojového učenia a neurónových sietí. V druhej kapitole sú neurónové siete predstavené detailnejšie a sú vysvetlené ich základné stavebné koncepty - optimalizácia, metóda backpropagation a regularizácia. Tretia kapitola je už zameraná na konkrétny druh neuronových sietí a to konvolučné siete. Má za cieľ vysvetliť základné princípy ako konvolúciu v kontexte konvolučných sietí a typické vrstvy z ktorých sa skladá architektúra siete. Štvrtá kapitola sa zaoberá návrhom programu, ktorý bude realizovať implementáciu konvolučnej neuronovej siete vrátane algoritmu učenia a zároveň bude demonštrovať činnosť siete v grafickom rozhraní. Implementácia je zdokumentovaná v šiestej kapitole. Siedma kapitola demonštruje výpočet v implementovanom grafickom rozhraní a program pre klasifikáciu na základe natrénovaných modelov. Posledná kapitola je zameraná na experimenty s neurónovou sieťou, hodnotí a porovnáva rôzne architektúry a ich úspešnosť učenia.

Kapitola 2

Strojové učenie a neurónové siete

Cieľom tejto kapitoly je uviesť čitateľa do problematiky strojového učenia a vysvetliť kľúčové pojmy, ktoré sa budú používať v tejto práci. Ďalej bude diskutovaná problematika hlbokého učenia a neurónových sietí.

Algoritmus strojového učenia je taký algoritmus, ktorý je schopný sa učiť z dát. [1]

Mnohé úlohy sú jednoduché pre človeka, ale zároveň sú algoritmicky tažko riešiteľné počítačom - a práve tie sú vhodným kandidátom pre riešenie metódami strojového učenia. Cieľom strojového učenia je mať schopnosť riešiť nejakú úlohu, vykonávať nejakú činnosť a samotné učenie je prostriedkom ako túto schopnosť nadobudnúť. Úlohy pre strojové učenie sú zvyčajne formulované tak, že definujú ako má systém strojového učenia spracovať položku vstupných dát. Položku vstupných dát zvyčajne reprezentujeme ako kolekciu vlastností vo forme n-rozmerného vektoru a na výstupe systému potom očakávame riešenie zadanej úlohy. Ako príklad sú uvedené niektoré bežné úlohy pre strojové učenie:

- Klasifikácia
  Úlohou programu je zaradiť vstupnú vzorku dát do jednej z k kategórií. Riešenie úlohy môže mať podobu funkcie $f : R^n \rightarrow 1...k$, kedy je vstup zaradený do práve jednej kategórie alebo môže byť výstupom programu zoznam pravdepodonností, že objekt patrí do jednotlivých tried. Typickou klasifikačnou úlohou je napríklad klasifikácia objektov na obrázku. Zvláštnym typom klasifikácie je klasifikácia s chýbajúcimi vstupmi, kedy nie sú na vstup dodané kompletné dáta.

- Regresia
  Úlohou programu je priradiť vstupu numerickú hodnotu $f : R^n \rightarrow R$. Typickou úlohou je napríklad predpovedanie ceny nejakej položky na trhu.

- Transkripcia
  Program má za úlohu z neštrukturovaných dát získať štruktúrovanú informáciu vo forme textu. Patria sem napríklad problematika OCR (Optical Character Recognition) alebo rozpoznávanie reči (speech recognition).

- Strojový preklad
  Systém strojového prekladu pracuje so štruktúrovaným vstupom vo forme znakov jedného jazyka a prevádza ho na ekvivalentný výstup v inom jazyku.

- Detekcia anomálií
  Program spacováva sled udalostí alebo objektov a snaží sa označiť tie, ktoré zo sledu nejakým spôsobom vybočujú. Detekcie anomálií sa používajú napríklad pri identifikácií zneužitia účtov, pre identifikáciu potenciálne nebezpečných osôb alebo rizikového správania.

- Syntéza a vzorkovanie
  Od programu sa očakáva generovanie nových dát, ktoré sú podobné tým existujúcim. Táto schopnosť sa využíva napríklad pre generovanie odporúčaní užívateľa nejakej služby (návrhy na kúpu nových hudobných albumov podľa vkusu daného užívateľa) alebo v grafike na automatické generovanie textúry objektov.

Miera úspešnosti strojového učenia (performance measure) sa často hodnotí ako presnosť modelu (accuracy) alebo naopak ako chybovosť (error rate), ktoré vyjadrujú mieru správnych resp. nesprávnych výstupov modelu strojového učenia. Cieľom strojového učenia je, aby program vedel správne reagovať na akékoľvek vstupy v rámci svojej domény, nie len na tie, ktoré sú mu už známe. Pre objektívne hodnotenie je preto potrebné dataset rozdeliť na dve skupiny, z ktorých jednu tvoria trénovacie dáta a druhú testovacie dáta. Úspešnosť sa určuje na datách, ktoré neboli použité pre trénovanie modelu - označujú sa ako testovacie.

Dôležitou otázkou návrhu modelu strojového učenia je ujasnenie si spôsobu hodnotenia modelu, ktoré sa líši v závisloti od typu učenia, ktoré použijeme. Základné delenie strojového učenia je na metódy učenia s učiteľom (supervised learning), metódy bez učiteľa (unsupervised learning) a tzv. reinforcement learning. V prípade učenia s učiteľom je k dispozícii dataset dvojíc vstup a odpovedajúci výstup a výstup modelu sa potom porovnáva s týmto referenčným výstupom. V prípade reinforcement learning model nemá k dispozícií hodnotenie každého výstupu, ale dostáva pri učení spätnú väzbu vo forme odmeny, ktorú sa snaží maximalizovať. Naproti tomu, učenie bez učiteľa funguje na základe informácií, ktoré program sám odvozduje z trénovacích dát.

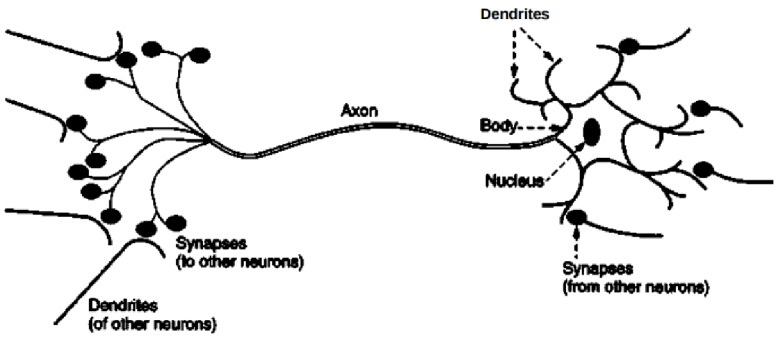
2.1 Neurónové siete

Neurónová sieťou je typom označeného orientovaného grafu, ktorého uzly vykonávajú jednoduché výpočty.[7] Neuronová sieť ako výpočetný model je inšpirovaná ľudskou nervovou sústavou, pričom uzly grafu reprezentujú jednotlivé neuróny a hrany synaptické väzby medzi nimi. Jednotlivé neuróny prijímajú cez svoje väzby signály od ostatných neurónov, spracujú ich a posielajú svojou aktiváciou signál ďalej.

Model neurónu

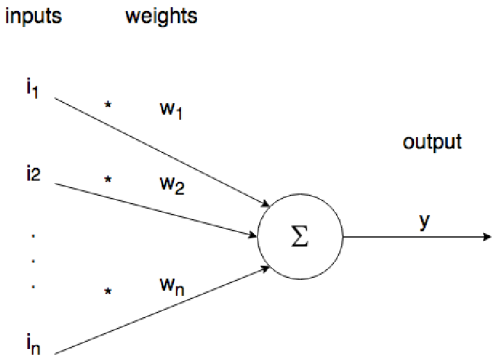
Biologický neurón sa skladá z tela, výbežku axónu a niekoľkých vlásočnicových výbežkov dendritov. Dendrity sú spojené so synapsami axónom, ktorý slúži na prenos signálu z neurónu. Dendrity slúžia na prijímanie signálov a axón na šírenie vlastného signálu do ostatných neurónov, pričom sila prenášaných signálov závisí na sile danej synapse.

Na základe biologického neurónu boli vytvorené umelé modely neurónov. Umelý neurón simuluje silu synapsí numerickými váhami tak, že signál sa danou váhou násobí a to určuje jeho silu. Ideou je, že tieto váhy sú predmetom učenia a určujú vplyv neurónov medzi sebou. Váhy môžu byť rovnako ako u biologického neurónu exhcitačné (kladné) alebo inhibičné



Obrázek 2.1: Schéma biologického neurónu [7]

(záporné). Prijaté signály sa v základnom modeli v neuróne sčítajú a výsledok je vstupom tzv. aktivačnej funkcie, ktorá rozhodne, či neurón vyšle signál a s akou silou. Inými slovami, neurón vypočíta skalárny súčin prijatých signálov a aplikuje naň nelineárnu funkciu. Rôzne typy aktivačných funkcií sú uvedené v kapitole 4.3.



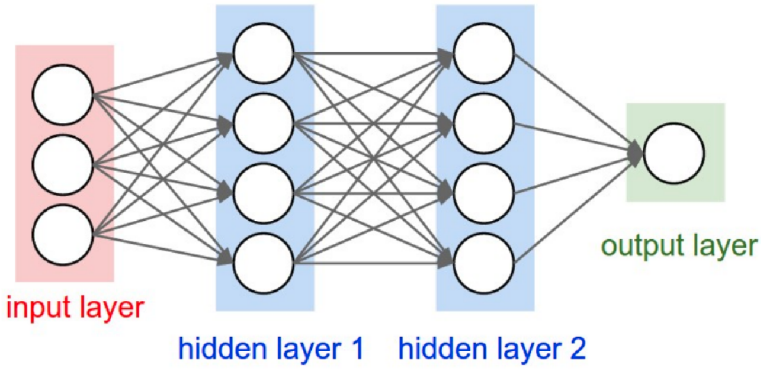
Obrázek 2.2: Jednoduchý model umelého neurónu

Štruktúra neurónovej siete

Neuronová sieť typu feedforward network sa skladá z neurónov usporiadaných do acyklického grafu. Iným typom je tzv. rekurentná neuronová sieť, ktorá obsahuje v arhcitektúre cykly (nazývaná aj feedback network). Práca sa bude ďalej zaoberať neuronovými sieťami typu feedforward.

Typicky je neurónová sieť štruktúrovaná do vrstiev. Jednotlivé vrstvy môžu byť rôzneho typu, základným typom neurónovej vrstvy je takzvane plne prepojená vrstva - každý neurón vrstvy je prepojený so všetkými neurónmi predchádzajúcej vrstvy a neuróny v jednej vrstve nie sú navzájom prepojené. Rozlišujeme medzi jednovrstvovými neurónovými sieťami a viacvrstvovými. Vrstva neurónov, ktorá nie je pripojená priamo na vstup a jej výstup nie je výstupom siete sa nazýva skrytá vrstva. Neurónové siete s aspoň jednou skrytou vrstvou sa nazývajú hlboké neurónové siete. Jednovrstvová architektúra obsahuje vstupnú vrstvu (input layer) a jednu vrstvu neurónov, korá je súčasne aj výstupnou vrstvou. Viacvrstvové siete obsahujú minimálne jednu vrstvu, ktorá nie je ani vstupnou a ani vrstvou, ktorej výs-

tupy sú výstupmi celej siete. Ukážka viacvrstvovej neurónovej siete na obrázku 2.3 má tri vrstvy (dve skryté) a obsahuje dohromady 9 neurónov.



Obrázek 2.3: Ukážka architektúry 3-vrstvovej neurónovej siete - s dvomi skytými vrstvami a jednou výstupnou. Vstup (Input) sa síce znázorňuje ako vrstva, ale nepočíta sa ako vrstva siete keď hovoríme o architektúre. [3]

Kapacita siete

Kapacitou neuronovej siete sa myslí schopnosť siete aproximovať riešenie úlohy a súvisí s množstvom informácie, ktoré možno v sieti uložiť. Zvyšovanie hĺbky siete môže viesť k zvyšovaniu kapacity siete, ale nie je to pravidlom. Závisí od typu siete a riešenej úlohy a nie je ľahké teoreticky určiť najvhodnejší počet vrstiev.

Neurónová sieť s aspoň jednou skytou vrstvou je považovaná za univerzálny aproximátor. Toto tvrdenie vychádza z tzv. universality teorému [8], ktorý hovorí, že neuronová sieť je schopná aproximovať ľubovoľnú spojitú funkciu.

Kapitola 3

Učenie neuronovej siete

Táto kapitola sa zaoberá problematikou učenia neurónových sietí a vysvetľuje mechanizmus spätnej väzby, na základe ktorého sieť upravuje svoje parametre.

3.1 Optimalizácia - zostup po gradiente

Chybová funkcia (anglicky loss function) je prostriedok pre určenie miery správnosti výstupu neurónovej siete. Jej voľba má zásadný význam pretože vo fáze trénovania neurónovej siete sa na základe nej upravujú parametre siete, ktoré sú predmetom učenia. Učenie teda zahŕňa proces minimalizácie chybovej funkcie, čo je druh optimalizačnej úlohy. Od všeobecnej optimalizácie sa ale líši v tom, že samotné minimalizovanie chybovej funkcie nie je cieľom, ale predpokladá sa, že povedie k zlepšeniu určitej metriky siete. Touto metrikou môže byť úspešnosť siete pri odpovedi na neznáme vstupy a predmetom optimalizácie je odpoveď siete na trénovaciu množinu vstupov, pričom optimalizácia na trénovacích vstupoch automaticky nemusí viesť na optimálne fungovanie siete pre neznáme vstupy.

V tejto práci sa bude uvažovať najpoužívanejšia metóda, a to metóda zostupu po gradiente (gradient descent). V multidimenzionálnom priestore je gradient vektor parciálnych derivácií v smere každej z dimenzií. Gradient v smere danej premennej chápeme ako vyjadrenie miery zmeny hodnoty funkcie pri nekonečne malej zmene hodnoty danej premennej. Ak je gradient záporný, znamená to, že zvýšenie hodnoty premennej, vzhľadom ku ktorej je vypočítaný, znamená zníženie hodnoty funkcie a naopak. Absolútna hodnota gradientu je potom miera veľkosti tejto zmeny.

Gradient udáva smer minimalizácie funkcie, ale dĺžka kroku, o ktorý sa posunú váhy v jednej iterácii v smere gradientu, je hyperparameter, ktorý treba vhodne nastaviť. Dĺžka kroku sa označuje termínom learning rate, ktorý bude používaný ďalej v tejto práci. Learning rate môže mať konštantnú hodnotu alebo sa môže v priebehu učenia meniť, typicky sa hodnota znižuje v priebehu učenia.

Pokiaľ sa gradient počíta na celej množine trénovacích dát, hovoríme o tzv. deterministickom gradiente. Gradient sa v neurónových sieťach počíta buď po každej trénovacej vzorke, ktorá prejde sieťou osobitne - vtedy sa metóda označuje ako SGD, stochastic gradient descent. Často sa však z dôvodu optimalizácie výpočtu evaluuje gradient po niekoľkých vzorkách dát (označované ako batch alebo minibatch), prípadne je možné gradient evaluovať až po celej epoche (všetkých trénovacích vzorkách).

Interpretácia gradientu

Majme funkciu $f(x, y) = xy$. Parciálne derivácie podľa jednej a druhej premennej sú $\frac{\partial f}{\partial dx} = y$ a $\frac{\partial f}{\partial x} = x$. Na deriváciu sa môžme pozerať z hľadiska limity:

$$\frac{df}{dx} = lim_{x \to 0} \frac{f(x+h) - f(x)}{h} \tag{3.1}$$

Intuitívne teda derivácia označuje citlivosť funkcie f na zmenu hodnoty premennej x resp. y. Gradient je $l_n + 1$ vektor parciálnych derivácii, a teda gradient funkcie f:

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}] = [y, x] \tag{3.2}$$

Dôležitou funkciou používanou v tejto práci je funkcia $f(x) = max(x, y)$ 4.3. Parciálne derivácie funkcie sú $\frac{\partial f}{\partial x} = 1 \wedge (x >= y)$ a $\frac{\partial f}{\partial y} = 1 \wedge (y >= x)$. Gradient vzhľadom k premennej s väčšou hodnotou je 1 a vzhľadom k premennej s menšou hodnotou 0, pretože menšia z dvoch hodnôt nemá na honotu funkcie vplyv. Toto je možné v kontexte výpočtu gradientu tvrdiť, pretože gradient pracuje s nekonečne malou zmenou premennej. Počíta sa s tým, že nekonečne malá zmena hodnoty hodnota premennej nespôsobí to, že by prevýšila pôvodne vyššiu hodnotu druhej premennej.

3.1.1 Problémy optimalizácie

Súčasťou optimalizácie je problém inicializácie parametrov. Voľba počiatočných hodnôt váh v sieti môže mať zásadný vplyv na úspešnosť trénovacieho algoritmu alebo dokonca na to, či bude vôbec konvergovať. Vzhľadom k tomu, že hlbšie chápanie fungovania neuronových sietí v súčasnosti ešte stále chýba, existujú len všeobecné odporúčania na zvýšený úspech voľby iniciálnych hodnôt. Prvým pravidlom je voliť parametre tak, aby nevznikla symetria. Ak majú dva neuróny v rovnakej vrstve rovnaké vstupy a aktivačné funkcie, mali by mať odlišné počiatočné hodnoty váh. V opačnom prípade by to viedlo k tomu, že tieto neuróny by v prípade deterministického učiaceho algoritmu pokračovali v zdieľaní rovnakých váh. Existujú viaceré možnosti ako túto vlastnosť dosiahnuť, ale najjednoduchšou a relatívne spoľahlivou je náhodná inicializácia podľa Gaussovej distribúcie vrátane normalizácie. Používanou stratégiou je aj batch normalization [2]. Normalizácia vstupných dát sa často používa jednorazovo ako súčasť predspracovania dát. V priebehu učenia neurónovej siete sa však menia jej parametre a to spôsobí, že normalizované vstupné dáta môžu byť v príliš vysokých alebo nízkych hodnotách. Práve tomu je možné zabrániť technikou batch normalization tak, že sa budú vstupné data normalizovať v priebehu učenia a to pridaním vrstvy s týmto účelom do architektúry siete.

3.2 Backpropagation

Metóda backpropagation tvorí podstatu učenia hlbokých neuronových sietí, je to spôsob akým sa spätná väzba z výstupnej vrstvy dostáva k ďalším vrstvám v sieti.

Smeru výpočtu od vstupu k výstupu neuronovej siete hovoríme o tzv. forward propagation (dopredná propagácia) a opačnému smeru od výstupu k vstupu siete hovoríme backpropagation. Po doprednom výpočte výstupu neurónovej siete je potrebné upraviť váhy neurónov v jednotlivých vrstvách na základe miery správnosti výstupu siete. Hodnoty váh sa upravujú typicky po spracovaní jedného vstupu alebo po spracovaní jednej dávky vstupov

(batch). Prvým krokom je spočítanie gradientov chyby vzhľadom na všetky váhy v sieti, teda propagácia informácie smerom k vstupom. Druhým krokom je použitie metódy zostupu po gradiente, ktorá aktualizuje hodnoty váh s cieľom minimalizovať chybu na výstupe siete. Platí, že váhy sú upravené proporcionálne k ich príspevku k celkovej chybe siete.

Gradient je možné vypočítať dvoma metódami, numericky a analyticky. Numerický výpočet gradientu je priamočiary, avšak výpočetne náročný, pričom je potrebné odvodiť výraz vyjadrujúci gradient pre konkrétnu architektúru siete. Z tohto dôvodu sa používa analytický výpočet gradientu pomocou výpočetného grafu. Výpočetný graf je tvorený uzlami reprezentujúcimi premenné a je sprevádzaný množinou operácií. Operáciu definujeme ako jednoduchú funkciu s viacerými vstupnými premennými. Spájaním operácií získame zložitejšie funkcie. Vo výpočetnom grafe orientovaná hrana z uzlu x do uzlu z znamená, že hodnota premennej z sa získa aplikovaním operácie na premennú x.

Reťazové pravidlo slúži na odvodenie derivácie funkcie na základe reťazenia derivácií funkcií, ktoré sú známe. Backpropagation je algoritmus, ktorý počíta reťazové pravidlo so špecifickým poradím operácií.[1] Predpokladajme funkcie f a g definované v obore reálnych čísel tak, že $y = g(x)$ a $z = f(g(x)) = f(y)$. Potom podľa reťazového pravidla platí:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x} \tag{3.3}$$

V kontexte neuronových sietí je dôležitá varianta, kedy x a y nie sú skaláry, ale vektory, napríklad $\vec{x} \in \mathbb{R}^n$ a $\vec{y} \in \mathbb{R}^m$. V tom prípade bude výpočet derivácie vyzerať nasledovne:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{3.4}$$

Vyššie uvedené môžme vyjadriť aj pomocou Jacobiho matice tak, že násobime Jacobiho maticu funkcie g(derivácia každého elementu g vzhľadom ku každému elementu $\vec{x}$) a gradient $\nabla_{\vec{y}} z$. Notácia $\nabla_{\vec{x}} z$ vyjadruje gradient hodnoty z vzhľadom k $\vec{x}$.

$$\nabla_{\vec{x}} z = (\frac{\partial \vec{y}}{\partial \vec{x}})^T \nabla_{\vec{y}} z \tag{3.5}$$

Každý uzol vo výpočetnom grafe reprezentujúci funkciu je schopný vypočítať lokálny gradient - teda parciálne derivácie svojho výstupu podľa svojich vstupov. V procese backpropagation sa potom z lokálneho a propagovaného gradientu určí vplyv vstupov uzlu na výstup celej siete. Cieľom celého procesu backpropagation je pri hodnotiacej funkcii L dostať pre každú váhu w_i, ktorá je predmetom učenia gradient $\nabla_{w_i} L$.

Princíp backpropagation sa dá implementovať rôznymi spôsobmi, základom je ale mať prístup k nasledovným informáciám [1] :

- operácia, ktorá určí hodnotu uzlu
- zoznam premenných, ktoré sú následníkmi uzlu v grafe
- zoznam premenných, ktoré sú predkami uzlu v grafe

Základný postup výpočtu siete:

1. Na vstup siete sa privedie položka trénovacích dát. Vstupná vrstva vypočíta svoj výstup a dá ho na vstup nasledujúcej vrstve, tá ďalšej vrstve, a tak ďalej, až po poslednú vrstvu siete.

2. Na základe výstupu poslednej vrstvy sa vypočíta funkcia loss a výstupná vrstva dostane späť gradient chyby vzhľadom k svojim výstupom.

3. Každá vrstva, začínajúc od poslednej, postupne vypočíta lokálny gradient svojho výstupu vzhľadom k váham $\nabla_{\vec{w}} z$ a vzhľadom k svojmu vstupu $\nabla_{\vec{x}} z$ a oba vynásobí ho gradientom prijatým z vyššej vrstvy. Gradient loss funkcie vzhľadom k vstupu vrstvy $\nabla_{\vec{x}} L$ pošle ďalej do nižšej vrstvy. Gradient loss funkcie vzhľadom k svojim váham $\nabla_{\vec{w}} L$ použije na update svojich váh.

4. Výpočet a predávanie gradientu pokračuje medzi vrstvami až k počiatočnej, vstupnej, vrstve.

Aktualizácia parametrov vo vrstve (weight update) prebieha na základe vzorca 3.6, kde w_i označuje jednolivé váhy (parametre) vrstvy a λ je spomínaý hyperparameter learning rate.

$$w_i \mathrel{+}= -\lambda \times \nabla_{w_i} L \tag{3.6}$$

3.3 Vlastnosti neurónových sietí a regularizácia

Najdôležitejšou vlastnosťou neurónovej siete, ktorú chceme učením dosiahnuť, je generalizácia. Znamená to, že sieť správne klasifikuje vstupné dáta, ktoré neboli súčasťou datovej sady použitej pri učení. Chybu siete na trénovacích datách nazývame trénovacia chyba a proces jej minimalizácie sa nazýva optimalizácia. Oproti tomu, generalizačná alebo testovacia chyba t.j. chyba na testovacích dátach, je očakávaná hodnota chyby na nových dátach, ktoré sieť nedostala na vstup počas trénovania. [1]. Aby bolo možné spoľahlivo určiť generalizačnú chybu z testovacích dát, je potrebné, aby testovacia a trénovacia sada boli na sebe nazávislé a obe s identicky distribuovaými pravdepodobnosťami. Cieľom učiaceho algoritmu je dosiahnuť nízku trénovaciu chybu a zároveň malý rozdiel medzi trénovacou a testovacou chybou. Nesplnenie týchto cieľov vedie k dvom fenoménom a to underfitting a overfitting siete. Prvý z nich znamená, že sieť dosahuje vysokú trénovaciu chybu, zatiaľčo druhý je v praxi častejšie riešeným roblémom a znamená, že sieť napriek nízkej trénovacej chybe dosahuje vysokú testovaciu chybu.

Underfitting aj overfitting súvisia s kapacitou siete, čiže množstvom funkcií, ktoré je schopná sa sieť naučiť. Sieť s vysokou kapacitou je náchylná na pretrénovanie a naopak sieť s nízkou kapacitou na underfitting. Intuitívne by mala kapacita siete zodpovedať zložitosti problému, ktorý rieši. Podľa princípu Occamovej britvy by malo byť z množiny riešení s rovnakým hodnotením vybrané to najmenej zložité, čo sa uplatňuje v oblasti neuronových sietí. To, ktoré riešenia sieť vyberie môžme ovlyvniť, okrem voľby architektúry siete, aj metódami tzv. regularizácie. Metódy regularizácie umožňujú vyjadriť preferenciu pre riešenia s určitou vlastnosťou.

Pretrénovanie neurónovej siete je jedným z najčastejších problémov strojového učenia. Prejavuje sa tak, že neurónová sieť dosahuje výborné výsledky na trénovacej sade dát, ale na testovacej sade dát zlyháva. Znamená to, že sieť hodnoty parametrov prispôsobila trénovacím vstupom na úkor generalizácie t.j. nie je schopná aplikovať naučené znalosti na vstupy, ktoré dostáva po prvýkrát. Môže to byť dôsledkom nedostatočnej pestrosti trénovacích dát alebo ich malému objemu alebo prílišnej zložitosti riešenia, ktoré sa sieť naučila. Zníženie

zložitosti naučeného riešenia sa dosiahne niektorou z metód regularizácie. Regularizácia môže mať podobu regularizačného termu. ktorý je súčasťou chybovej funkcie. Regularizačný term je hyperparameter neurónovej siete, označíme ako $R(\vec{w})$. Nasleduje stručný popis najčastejších metód regularizácie.

Penalizácia normy parametra

Penalizácia funguje vo forme hodnoty pričítanej k hodnotiacej funkcii (objective function) a tým limituje kapacitu modelu neurónovej siete. Penalizácia sa počíta v závislosti na aktuálnych hodnotách váh neurónu. Nižšie je uvedený všeobecný vzorec vyjadrujúci regularizovanú hodnotiacu funkciu $R(\vec{w})$ 3.7, kde $\vec{w}$ reprezentuje predmet učenia (váhy), $\vec{x}$ vstupy a L hodnotiacu funkciu siete. Hyperparameter λ určuje mieru regularizácie funkcie tak, že $\lambda = 0$ znamená žiadnu regularizáciu. Ako motiváciu k zavedeniu regularizačného termu môžme uviesť situáciu, keď neuronová sieť dospeje k riešeniu reprezentovanému váhami $\vec{w}$, ktoré vykazuje nulovú chybu. Zároveň však aj násobky riešenia $n \times \vec{w}$, kde $n > 1$, dosahujú nulovú chybu. Regularizačný term nám potom pomôže vyjadriť preferenciu medzi týmito riešeniami a zmierniť nejednoznačnosť.

$$\tilde{L} = L + \lambda R(\vec{w}) \tag{3.7}$$

Regularizácia máva formu tzv. L^2 alebo L^1 regularizácie. L^2 regularizácia vyjadruje preferenciu hodnôt váh bližšie k iniciálnej hodnote t.j. preferuje malé zmeny hodnôt váh. Regularizačný term je vyjadrený rovnicou 3.8 a pripočítava sa k hodnotiacej funkcii.

$$R(\vec{w}) = \sum_k \sum_l w_{k,l}^2 \tag{3.8}$$

Regularizačný term pre L^1 má tvar uvedený v 3.9, čiže sumu absolutných hodnôt všetkých váh v modeli. L^1 regularizácia v porovnaní s L^2 vedie k redšej váhovej reprezentácii, čím sa myslí to, že viac váh pri L^1 regularizácii má veľkosť blízku nule. Preto sa L^1 regularizácia používa pre mechanizmus výber prvkov (feature selection). L^2 regularizácia je však používaná častejšie.

$$R(\vec{w}) = \sum_i |w_i| \tag{3.9}$$

Druhou možnosťou regulácie váh v sieti je explicitne určiť obmedzenia váh neurónu vo forme horného alebo dolného obmedzenia hodnôt.

Augmentácia datasetu

Pri strojovom učení všeobecne platí, že čím je trénovací dataset väčší, tým lepšie výsledky možno dosiahnuť. V prípade, keď nemáme k dispozícii viac trénovacíh dát, môžme v niektorých prípadoch vygenerovať z existujúcich dát nové. Toto funguje dobre napríklad v klasifikačných úlohách, pretože nové data sa dajú vytvoriť pomocou jednoduchých transformácií (napríklad posunutie alebo rotácia pri klasifikácii obrázkov). Inou formou zväčšenia objemu trénovacieho datasetu je vkladanie šumu do vstupov neurónovej siete. Dokonca je možné vkladať šum aj do skrytých vrstiev siete, teda augmentáciu datasetu na rôznych úrovniach abstrakcie.

Robustnost voči šumu

Šum môžno pridať k vstupným dátam alebo k váham neurónov, prípadne k požadovaným výstupom siete. Šum pridaný k váham siete môže byť alternatívou k použitiu regularizačného termu a prispieva k stabilite naučenej funkcie. Pridávanie šumu k očakávaným výstupom siete (labels) vychádza z predpokladu, že trénovacie data obsahujú s istou pravdepodobnosťou aj chybné očakávané výstupy.

Semi-supervised learning

V tzv. semi-supervised učení pozostáva trénovacia množina z označených aj neoznačených dát. Cieľom je previesť vstupy siete x na inú reprezentáciu $h = f(x)$, ktorá v sebe obsahuje informáciu o podobnosti vstupných dát patriacich do rovnakej triedy. Prevod vstupných dát na inú reprezentáciu býva implementovaný ako krok predspracovania dát.

Early stopping

V priebehu učenia vo väčšine prípadov pozorujeme, že trénovacia chyba sa časom zmenšuje, avšak testovacia chyba sa zväčšuje. Riešením je ukladanie aktuálneho stavu siete (stav parametrov) v priebehu učenia, vždy keď sa trénovacia chyba zníži. Vďaka tomu, keď sa trend zvyšovania testovacej chyby objaví, je možné sa vrátiť k stavu, kedy bola táto chyba najnižšia. Výhodou tejto metódy je zanedbateľná pamäťová náročnosť ukladania najlepšieho stavu parametrov a to, že nezasahuje do dynamiky učiaceho procesu. Pri iných formách regularizácie, napr. regularizačným termom, je potrebná opatrnosť, aby nemali negatívny vplyv na učiaci proces algoritmu.

Viazanie a zdieľanie parametrov

Vzájomné viazanie parametrov je používané v prípade, že medzi parametrami existuje nejaká závislosť, ktorú je vhodné do modelu preniesť. Vzťah medzi parametrami môže vyplývať zo znalosti domény alebo architektúry modelu. Penalizácia pre vyjadrenie preferencie blízkych hodnôt váhy w_x a w_y môže mať s využitím L^1 normalizácie tvar $\Omega(w_x, w_y) = |w_x - w_y|$. Extrémnou závislosťou, keď sa požaduje, aby boli si parametre rovné, je zdieľanie parametrov. Technika zdieľania paramerov tvorí základ konvolučných neuronových sietí.

Riedka reprezentácia

Penalizácia je umiestnená na aktivačné funkcie neurónov a nepriamo spôsobuje penalizáciu parametrov modelu. Vyššie je zmienené, že L^1 regularizácia spôsobuje riedku parametrizáciu 3.9. Naproti tomu riedka reprezentácia znamená, že vstup siete prevedieme na reprezentáciu s nulovými hodnotami resp. hodnotami blízkymi nule. Na obrázku je ilustrovaná riedka parametrizácia 3.1 a riedka reprezentácia vyjadrená vektorom $\vec{h}$ 3.2. Regularizačný term pre riedku reprezentáciu s normou L^1 môže mať tvar $\Omega(h) = \|h\| = \sum_i |h_i|$.

Bagging

Pre riešenie úlohy je nezávisle natrénovaných viacero modelov a finálnym riešením je výstup siete vyberaný pomocou hlasovania jednotlivých modelov. Tento prístup sa označuje

$$\underset{\boldsymbol{y} \in \mathbb{R}^m}{\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix}} = \underset{\boldsymbol{A} \in \mathbb{R}^{m \times n}}{\begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix}} \underset{\boldsymbol{x} \in \mathbb{R}^n}{\begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}}$$

Obrázek 3.1: Riedka parametrizácia lineárnej regresie [1]. $\vec{x}$ sú vstupné data, $\vec{A}$ je riedka matica váh a $\vec{y}$ je výstup.

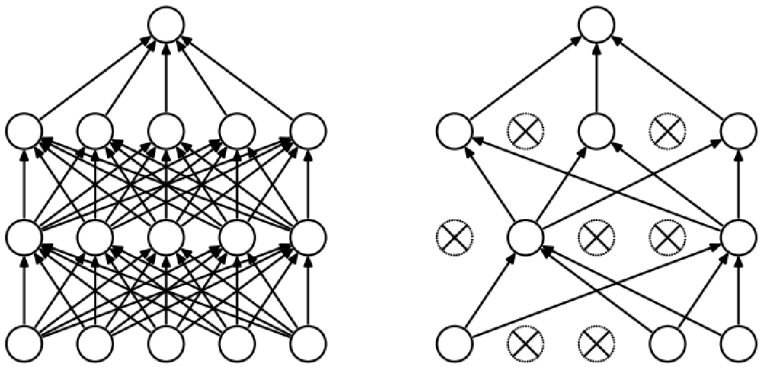
$$\underset{\boldsymbol{y} \in \mathbb{R}^m}{\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix}} = \underset{\boldsymbol{B} \in \mathbb{R}^{m \times n}}{\begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix}} \underset{\boldsymbol{h} \in \mathbb{R}^n}{\begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}}$$

Obrázek 3.2: Riedka reprezentácia lineárnej regresie [1]. $\vec{h}$ je riedka reprezentácia vstuných dát, pričom $\vec{h}$ je funckiou vstupných dát $\vec{x}$, $\vec{B}$ je matica váh a $\vec{y}$ je výstup.

ako priemerovanie modelov (model averagging) a je založený na predpoklade, že rôzne modely sa nedopúšťajú rovnakých chýb. Modely bývajú typicky trénované na odlišných trénovacích datách (vytvorených z originálneho trénovacieho datasetu) avšak vďaka náhodnosti a nedeterminizmu, ktoré neurónové siete obsahujú (náhodná inicializácia, výber hyperparametrov atď.) sa aj pri použití identických trénovacích dát modely líšia.

Dropout

Motiváciou metódy je znížiť závislosť neurónov na sebe navzájom. Samozrejme, závislosť neurónov je podstatou neurónových sietí, ale z hľadiska generalizácie je prínosné znížiť závislosť jedného neurónu na inom konkrétnom neuróne. Technika dropout je realizovaná tak, že počas učenia sú zo siete vynechané niektoré neuróny (okrem neurónov z výstupnej vrstvy), tým, že sa označia ako neaktívne a ich výstup je nulovaný. Ide o relatívne novú, ale populárnu techniku. Dropout možno chápať ako regularizáciu vkladaním šumu k skrytým jednotkám siete alebo ako implementáciu metódy bagging. V kontexte metódy bagging ide o trénovanie modelov líšiacich sa architektúrou, pričom množina možných modelov je množina podsietí neuronovej siete, ktoré vzniknú vynechaním vnútorných jednotiek (neurónov) z pôvodnej architektúry siete. Tieto podsiete majú identický učiaci algoritmus a navyše ich parametre sú na sebe závislé. Postupným prepínaním medzi subsieťami je teda iniciálnym stavom parametrov danej podsiete posledný stav tej predchádzajúcej. Je možné ju realizovať tak, že jednotlivé neuróny sú nastavené ako aktívne s istou pravdepodobnosťou a v každom nastavení pravdepodobností sieť spracuje niekoľko vstupov, typicky jeden batch dát. Po skončení trénovania je každý neurón v siete prítomný vždy. Treba mať na pamäti, že aj v tomto prípade, za cenu zníženia generalizáčnej chyby sa zníži kapacita modelu. Technika je znázornená na obrázku 3.3.



(a) Standard Neural Net (b) After applying dropout.

Obrázek 3.3: Ilustrácia techniky regularizácie dropout [9]. Sieť vľavo je štandardná neurónová sieť so skrytými vrstvami. Vpravo je daná sieť po aplikovaní operácie dropout. Krúžky s krížikom sú neuróny, ktoré sú zo siete vynechané (a teda sú vynechané aj ich väzby na ostatné neuróny).

3.4 Učenie klasifikačných úloh

Klasifikačné úlohy mapujú vstupné dáta siete na klasifikačné triedy, pričom každý vstup siete patrí väčšinou do jednej triedy. Pri riešení klasifikačných úloh je kľúčovými komponentami tzv. score funkcia a loss funkcia.
Funkcia score mapuje dáta vystupujúce z neurónovej siete na klasifikačné triedy. Táto funkcia sa väčšinou modeluje ako posledná vrstva neurónovej siete, ktorá je typu plne prepojená vrstva (fully connected layer). Jej vstupom sú dáta z predošlej vrstvy siete, lubovoľného typu a výstupom je takzvané skóre pre jednotlivé triedy. Skóre pre danú triedu vyjadruje presvedčenie, že vstup má byť zaradený do danej triedy, avšak na rozdiel od pravdepdobnosti, nie sú hodnoty z intervalu $(0, 1)$. Preto je potrebné toto skóre interpretovať, čo je možné realizovať porovnaním hodnôt v rámci daného výstupu. Teda, najvyššia hodnota skóre označuje najpravdepoobnejšiu triedu klasifikácie a naopak najnižšia hodnota tú najmenej pravdepodobnú triedu. Obecný predpis funkcie score, pre vstupné dáta dimenzie D a K klasifikačných tried: $f = R^D \rightarrow R^K$. Plne prepojená vrstva modelujúca score function potom bude obsahovať maticu váh s rozmermi $D \times K$ a prípadne vektor bias s rozmerom $1 \times K$.
Pre vysvetlenie funkcií score a loss predpokladajme lineárny klasifikátor (alebo analogicky posledná plne prepojená vrstva neuronovej siete). Funkcia score, označme f, sa dá vyjadriť rovnicou 3.10, kde x_i označuje data na vstupe, W označuje váhy a b je hodnota bias. Pre zjednodušenie sa ale používa zápis 3.11, kde je bias súčasťou váh W alebo nemusí byť vôbec použitý. [3]

$$f(x_i, \vec{w}, b) = \vec{w}x_i + b \tag{3.10}$$

$$f(x_i, \vec{w}) = \vec{w}x_i \tag{3.11}$$

Funkcia loss sa aplikuje na výstu funkcie score a vyjadruje spokojnosť s výstupom siete. Pomocou funkcie loss je možné interpretovať výsledok siete, v kontexte klasifikácie teda zaradenie do jednej z definovaných tried. Funkcia loss sa v literatúre označuje aj ako

cost funkcia alebo tzv. objective function. Predpokladáme učenie s učiteľom (supervised learning) a teda sú k dispozícii dvojice trénovacie dáta a label. Label obsahuje informáciu o triede klasifikácie, do ktorej vstupné dáta skutočne patria a preto sa používa pre určenie chyby siete. Existuje niekoľko typov funkcie loss, v ďalšej podkapitole budú podrobnejšie popísané dve najpoužívanejšie z nich. Spoločnou vlastnosťou funkcií loss je to, že čím vyššia hodnota je na ich výstupe, tým väčšia je nespokojnosť s výstupom siete.

3.5 SVM klasifikátor

Prvou z loss funkcií je funkcia označovaná SVM, ktorá je skratkou termínu Support Vector Machine, konkrétne Multiclass Support Vector Machine. Je založená na princípe, že skóre správnej triedy (určenej labelom) má byť vyššie ako skóre ostatných, nesprávnych, tried minimálne o určenú hodnotu delta Δ. Funkcia score 3.11 teda vypočíta skóre pre jednotlivé triedy, označme ako vektor $\vec{s}$, $f(x_i, \vec{w}) = \vec{s}$. Zložka s_j označuje skóre pre tiedu j a správna trieda je označená ako y_i, hodnota funkcie SVM loss L_i je potom vyjadrená rovnicou 3.13. Funkcia loss pre celý dataset je priemerná hodnota L_i 3.12, kde N j počet tried klasifikácie. V praxi sa k priemernej hodnote loss funkcie pričítava ešte tzv. regularization loss, v prípade, že chceme regularizovať váhy siete 3.14. Ideou je, že jednej hodnote loss funkcie zodpovedajú rôzne násobky váh λW a pomocou tohto regularizačného termu je možné ovplyvniť, ktorú variantu váh sa sieť naučí. Možnosti regularizácie sú bližšie popísané v kapitole 3.3.

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \Delta) \tag{3.12}$$

$$L = \frac{1}{N} \sum_i L_i \tag{3.13}$$

$$L = \frac{1}{N} \sum_i L_i + \lambda \vec{w} \tag{3.14}$$

3.6 Softmax funkcia

Druhou z najčastejšie používaných hodnotiacich funkcií neurónových sietí je funkcia softmax. Funkcia má na vstupe N-dimenzionálny vektor čísel a na výstupe vektor reálnych čísel s hodnotami v intervale $(0, 1)$, ktorých súčet je rovný 1. Funkcia softmax teda produkuje distribúciu pravdepodobnosti, ktorú je možné prirodzene interpretovať pri klasifikačných úlohách. Jednotlivé hodnoty vo výstupnom vektore zodpovedajú pravdepodobnostiam, že vstup patrí do danej triedy. Počet prvkov výstupného vektora zodpovedá počtu prvkom tried klasifikácie. Funkcia softmax sa používa v kombinácii s tzv. cross-entropy loss funkciou, ktorá vypočítava samotnú hodnotu loss pre výstup siete. Softmax funkcia má na vstupe priamo vektor so skóre pre jednotlivé triedy a na výstupe vektor normalizovaných hodnôt pravdepodobností. Cross-entropy loss funkcia má na vstupe tento vektor pravdepodobností a na výstupe výslednú hodnotu loss. V praxi sa ale často používa pojem len softmax funkcia pre označenie ich kombinácie.
Za predpokladu, že neurónová sieť realizuje funkciu $f(x_i, W) = Wx_i$, v kontexte klasifikácie znamená Wx_i skóre pre jednotlivé triedy klasifikácie. Toto skóre interpretujeme ako nenormalizované logaritmické pravdepodobnosti pre každú triedu pomocou cross-entropy loss funkcie. Ak K je počet tried klasifikácie, j index pre označenie triedy a hodnota jej

skóre je f_j, potom môžme funkciu cross-entropy loss pre vstup i vyjadriť rovnicou 3.15.

$$L_i = -log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}) \tag{3.15}$$

Z pohľadu pravdepodobnosti rovnica 3.16 vyjadruje pravdepodobnosť, že pre vstup x_i je správna trieda y_i.

$$P(y_i|x_i; \vec{w}) = \frac{e^{f_{y_j}}}{\sum_j e^{f_j}} \tag{3.16}$$

V ukážke kódu 3.4 je postup výpočtu základnej verzie funkcie softmax a cross-entropy loss a jej stabilnej verzie. Problémom, ktorý v praxi môže nastať je, že pri výpočte exponentu z dôvodu obmedzenej reprezentácie desatinných čísel, dôjde k tzv. pretečeniu hodnoty premennej. Aby sa obmedzila hodnota, ktorá vstupuje do exponentu a zabránilo sa tejto situácii, používa sa numericky stabilná verzia funkcie softmax. Numerickú stabilitu dosiahneme tým, že pred výpočtom exponenciálnej funkcie ku všetkým hodnotám pričítame konštantu. Hodnota tejto konštanty sa typicky volí $-max_j(f_j)$, teda odčítanie maximálnej hodnoty z daného vektora skóre. Funkcia softmax možno chápať ako generalizáciu multinomiálnej logistickej regresie. V logistickej regresii sa predpokladá binárny výstup, zatiaľčo multinomiálna logistická regresia počíta s ľubovoľným počtom tried klasifikácie.

```
import numpy as np

def softmax(x):
    exps = np.exp(x)
    return exps / np.sum(exps)

def softmax_stable(x):
    exps = np.exp(x) - np.max(x)
    return exps / np.sum(exps)
```

Obrázek 3.4: Algoritmus pre funkciu softmax v jazyku python

SVM vs. Softmax

Môžme si všimnúť, že funkcia Softmax na rozdiel od SVM nepovažuje za správny výsledok (t.j. propaguje nulovú chybu pre jednotlivé hodnoty vektora so skóre) žiaden prípad okrem toho, keď je pravdepodobnosť správnej triedy rovná 1 a ostaných rovná 0. Naopak, funkcia SVM považuje za správny výsledok taký, keď je každé skóre nesprávnej triedy menšie od skóre správnej triedy aspoň o hodnotu δ. Všeobecne sa v literatúre sa uvádza, že výsledok pri použití oboch funkcií je porovnateľný.

Kapitola 4

Konvolučné neuronové siete

Konvolučné neurónové siete sú druhom hlbokých neurónových sietí a do popredia záujmu sa dostali v roku 2012, kedy sieť s touto architektúrou zvvíťazila v súťaži ImageNet. Konvolučné neurónové siete majú rovnako ako iné neurónové siete parametre (váhy), ktorých hodnoty sú predmetom učenia. Ich architektúru rovnako popisujeme vrstvami obsahujúcimi neuróny, ktorých výpočet sa líši v závislosti od ich typu. Každý neurón v sieti počíta skalárny súčin svojho vstupu a váh a typicky aktivuje svoj výstup na základe nelineárnej funkcie. Od všeobecných neurónových sietí sa ale odlišujú tým, že obsahujú tzv. konvolučnú vrstvu.

V bežnej neurónovej sieti je každý neurón v prvej vrstve napojený na každú hodnotu zo vstupu (za predpokladu, že jeden vstup sa skladá z niekoľkých hodnôt). Tento prístup ale spôsobuje prudký nárast parametrov pokiaľ je vstup siete vysokodimenzionálny. Príkladom je vstupný obrázok s rozmermi 100×100 pixelov, kedy by mal každý neurón napojený na vstup počet parametrov 10000. Konvolučné neurónové siete naproti tomu zavádzajú princíp, kedy je každý neurón napojený len na malú časť vstupu (časť susedných položiek vstupu, napríklad výsek obrázku). Tento jav označujeme pojmom lokálna konektivita. Lokálna konektivita sa v konvolučných neurónových sieťach využíva nielen vo vrstve napojenej na vstupný dáta, ale aj v skyrých vrstvách a je propagovaná do celej siete. Typy neurónových sietí, ktoré využívajú princíp lokálnej konektivity sa potom označujú ako lokálne prepojené vrstvy (locally connected networks) alebo tzv. local receptive field [4].

Druhým typickým rysom všeobecnej neurónovej siete je, že každý neurón môže obsahovať unikátne váhy. Konvolučná neurónová sieť využíva techniku zdieľania parametrov (parameter sharing) 3.3. Znamená to, že neuróny v jednej vrstve zdieľajú rovnaké hodnoty parametrov. Toto prináša výhodu v pamäťovej náročnosti, pretože počet parametrov, ktoré treba uložiť sa výrazne znižuje. Na príklade obrázku s rozmerom 100×100 pixelov sa za predpokladu lokálnej konektivity veľkosti výrezu 9 pixelov a počtu rôznych hodnôt váh 4, zredukuje počet parametrov vo vrstve z 10000 na 36 (9×4). Zdieľanie parametrov vychádza z prepdokladu, že každá vzorka vstupu (myslí sa tým vektor, matica alebo obecne tenzor) obsahuje nejaké vlastnosti, vzory, ktoré sa v rámci nej opakujú. Má teda zmysel detekovať rovnakú vlastnosť na všetkých častich vstupu. Vlastnosť, ktorú detekujeme, je reprezentovaná práve setom váh, ktorý je zdieľaný. Zároveň je zachovaná pozícia, kde bola vlastnosť vo vstupe detekovaná, tým, že konvolučné vrstvy majú na výstupe tzv. aktivačnú mapu. Inituitívne, jedna aktivačná mapa detekuje jednu vlasnosť a mapuje ju na pozície vstupu. Je možné detekovať niekoľko vlastností v jednej vrstve a je to aj veľmi typické. Vrstva potom obsahuje niekoľko setov váh a na výstupe sa potom objaví niekoľko aktivačných máp, každá

pre jednu vlastnosť (jeden set váh).

Na základe vyššie uvedeného vysvetlenia môžme formálnejšie zhrnúť, že konvolučné neurónové sete stoja na troch základných princípoch [1]:

- riedka interakcia

  Princíp inak označovaný ako riedke váhy alebo riedka konektivita znamená, že jeden neurón reaguje len na malé množsto hodnôt vstupu (veľkosť je daná počtom váh neurónu).

- zdieľanie parametrov

  Znamená to, že rovnaký parameter sa použije pre viac ako jednu funkciu v modeli (zdieľa sa medzi neurónmi). Môžme sa naň pozerať aj ako na prípad viazaných váh parametrov (tied weights), kedy váhy aplikované na každú časť vstupu musia byť rovnaké. Konvolučná vrstva obsahuje váhy označené ako filter, ktoré simulujú niekoľko neurónov s rovnakými váhami, pričom každý neurón je napojený len na výsek vstupných dát.

- ekvivariantné reprezentácie

  Konvolučná vrstva spôsobuje vlastnosť ekvivariancie voči posunutiu, čo znamená, že ak sa vstup posunie, posunie sa rovnakým spôsobom aj výstup.

Ďalej v kapitole bude popísaný princíp konvolúcie, predstavené typické vrstvy konvolučnej neurónovej siete, z ktorých sa skladá konvolučná neuronová sieť a to konvolučnej vrstvy, aktivačnej vrstvy, vrstvy pooling a plne prepojenej vrstvy, tiež ich typické usporiadanie a aplikácie. Konvolučné neuronové siete sa budú ďalej označovať skratkou CNN.

4.1 Konvolúcia

Konvolučné neurónové siete sa od neurónových sietí všeobecného typu odlišujú tým, že obsahujú aspoň jednu vrstvu neurónov, ktorá pracuje na základe operácie konvolúcie. Konvolúcia je lineárna matematická operácia definovaná nasledujúcou rovnicou:

$$s(t) = (x * w)w(t) \tag{4.1}$$

Znak '*' označuje operáciu konvolúcie, x označuje v terminológii konvolučných neuronových sietí vstup (input) a w označuje jadro konvolúcie (kernel). Vstupom konvolúcie je typicky viacrozmerné pole hodnôt, ktoré sa nazýva tenzor. Jadro konvolúcie má zvyčajne rovnakú dimenziu ako je dimenzia vstupu, teda pri spracovaní obrazu to typicky bývajú dve alebo tri dimenzie (čiernobiely alebo RGB obrázok). Výsledok $s(t)$ sa ozačuje ako aktivačná mapa (activation map alebo feature map). Konvolúcia v kontexte neuronových sietí teda vyzerá nasledovne:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{4.2}$$

Mnohé knižnice pre neuronové siete však implementujú operáciu konvolúcie ako vzájomnú korelácia (cross-correlation), t.j. jadro konvolúcie sa pri násobení neotáča (nerobí sa operácia kernel flipping). Táto varianta je možná, pretože to nemá vplyv na algoritmus učenia siete. Konvolúcia je v prípade vynechania operácie kernel flipping definovaná nasledovne:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{4.3}$$

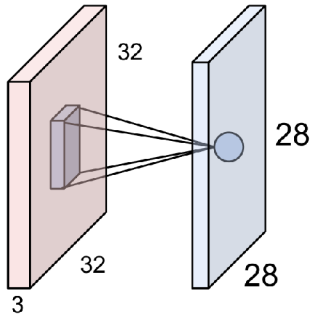
4.1.1 Konvolúcia v CNN

V úvode kapitoly bolo uvedené, že vrstvy neurónovej siete obsahujú parametre (váhy). Tieto váhy sa v kontexte neurónových sietí nazývajú kernel alebo jadro konvolúcie alebo podľa terminológie spracovania signálov, filter. Pre upresnenie, jedna skupina váh vo vrstve sa označuje ako filter, pričom jedna konvolučná vrstva môže obsahovať niekoľko skupín váh, teda niekoľko filtrov. Pri prechode vrstvou sa teda realizuje niekoľko konvolúcii, podľa počtu filtrov v danej vrstve.

Jadro konvolúcie má rovnakú dimenziu ako vstup, ale niekoľkonásobne menšiu veľkosť. V každom elementárnom kroku operácie konvolúcie sa jadro "priloží"na vstup, čím sa definuje aktuálny výsek vstupu a hodnoty tohto výseku vstupu a jadra konvolúcie sa vynásobia ako skalárny súčin - výsledkom násobenia teda bude jedno číslo. Potom sa jadro posunie (zmení pozíciu na vstupe) o jeden krok, čím definuje nový výsek vstupu a vypočíta sa druhý skalárny súčin. Výsledky jednotlivých konvolúcií sa ukladajú do aktivačnej mapy na príslušné pozície (dané pozíciou výseku na vstupe).

Aletrnatívne sa na jadro konvolúcie môžme pozerať ako na tenzor rovnakých rozmerov ako vstup, avšak len s niekoľkými nenulovými hodnotami umiestnenými pri sebe, pričom pri výpočte sa táto skupina nenulových hodnôt bude postupne posúvať po inak nuly obsahujúcom tenzore o daný krok.

Konvolúcia v neuronových sieťach teda prebieha tak, že filter postupne prikladáme so zvoleným krokom na vstup, vzájomne násobíme hodnoty aktuálneho prekrytia vstupu a filtra. Výsledky jednotlivých konvolúcií sa zapisujú do aktivačnej mapy. Aktivačná mapa po skončení celej operácie konvolúcie na vstupe obsahuje hodnoty skalárneho násobenia úseku vstupu a filtra po celom priestore vstupu.



Obrázek 4.1: Konvolučná vrstva s jedným filtrom o veľkosti 5x5x3. Ružový hranol označuje vstup, modrý aktivačnú mapu. Obrázok znázorňuje výsek vstupu (menší hranol vnútri vstupu) a výsledok konvolúcie (guličkа v modrej vrstve). Zdroj:[3]

Krok konvolúcie (stride) vyjadruje počet počet hodnôt vstupu, o ktorý sa posunie výsek vstupu určený jadrom konvolúcie medzi dvoma skalárnymi násobeniami v operácii konvolúcie. V kontexte ilustračného prikladania filtra na vstup si krok môžme predstaviť ako počet

hodnôt indexu, o ktorý posúvame jadro po vstupe. Krok konvolúcie spolu s rozmermi filtra určuje rozmery výstupnej aktivačnej mapy. Veľkosť kroku býva rovnaká vo všetkých dimenziách, pričom jeden krok znamená posun len v jednej dimenzii. Čím väčší krok zvolíme, tým menšiu aktivačnú mapu dostaneme.

Uvažujme 2-rozmerný vstup s rozmermi $n * n$ a filter s rozmermi $m * m$, krok dĺžky k a spôsob konvolúcie kedy filter nepresiahne hranice vstupu. Výstupná aktivačná mapa bude mať potom štvorcový rozmer s dĺžkou hrany: $(n - m)/k + 1$. Veľkosť kroku volíme tak, aby bol uvedené podiel celé číslo, z dôvodu zachovania symetrie výstupu.

Konvolúcie v kontexte neuronových sietí sa týka problém okrajových hodnôt vstupu. K tomuto problému existujú dva hlavné prístupy. Prvým prístupom je neriešiť problém menšej reprezentácie okrajových hodnôt vstupu, teda nepridávať k vstupným dátam žiadne zarovnanie a filter prikladať na vstup len tak, aby žiadna jeho časť neprekročila hranice vstupu. To znamená, že rozmer výstupu bude menší (v prípade kroku veľkosti 1) o $(m - 1)$ hodnôt, kde m je rozmer filtru v danej dimenzii. Ďalším dôsledkom bude to, že okrajové hodnoty budú mať menší vplyv na výstup celej neurónovej siete, pretože do výpočtu konvolúcie prispejú menejkrát. Druhým prístupom je pridať na okraje vstupu zarovnanie nulami (zero padding). Týmto spôsobom potom možno prikladať filter aj za hranic vstupu a dosiahnuť nezmenšené rozmery výstupnej aktivačnej mapy a zároveň reprezentovať na výstupe neuronovej siete hodnoty na okrajoch vstupu rovnakou váhou ako hodnoty v strede vstupu. Nevýhodou je, že doplnenie hodnôt na okraje môže spôsobiť detekciu neexistujúcih vlastností v týchto miestach. V praxi sa ale táto metóda napriek zmienej nevýhode často používa. Ďalším spôsobom riešenia je zreplikovať okrajové hodnoty, padding rozšíri pôvodný vstup zopakovaním jeho okrajových hodnôt.

Je vhodné pripomenúť, že v oblasti spracovania obrazu býva typicky na vstupe viackanálový, typicky 3-kanálový, 2-rozmerný vstup. Jadro konvolúcie kopíruje celú dimenziu vstupu, teda pre 2D obraz s 3 kanálmi bude mať dimenziu 3. Pri konvolúcii sa vstup posúva len po dvoch dimenziách a teda výstupom je potom aktivačná mapa s dimenziou 2. Rovnaký prípad nastáva, keď je vstupom konvolučnej vrstvy výstup inej konvolučnej vrstvy s viacerými aktivačnými mapami. Tretia dimenzia jadra konvolúcie sa zhoduje s počtom aktivačných máp od predchádzajúcej vrstvy.

Operácia konvolúcie býva často implementovaná z dôvody výpočetnej efektivity spôsobom, tak že každý úsek vstupu aj filter sa transformujú do jednorozmerného vektora a vykonáva sa operácia násobenia dvoch vektorov.

4.2 Konvolučná vrstva

Konvolučná vrstva je taká vrstva neurónov v konvolučnej neuronovej sieti, ktorá počíta operáciu konvolúcie vysvetlenú v sekcii 4.1.1. Jedna konvolučná vrstva typicky obsahuje niekoľko filtrov. Dôvodom je, že každý filter detekuje inú vlastnosť vstupu a ignoruje ostatné vlastnosti. Ak by boli filtre umiestnené nie paralelne, ale sériovo, ochudobnili by sme sa o detekciu vzájomne nezávislých prvkov vo vstupe. Napríklad, každý filter v prvej vrstve môže detekovať vo vstupnom obraze hranu v inom smere. Ak by prvá vrstva obsahovala len jeden filter, všetky ďalšie vrstvy by dostali informáciu len o tejto jednej hrane a informácie o ostatných smeroch hrán by boli stratené. Pre pripomenutie, počet filtrov vo vrstve určuje

počet aktivačných máp na výstupe vrstvy.

V porovnaní s plne prepojenou vrstvou neurónov má konvolučná vrstva výrazne menší počet parametrov, čo znižuje jej pamäťovú náročnosť a v dôsledku aj časovú náročnosť výpočtu. Počet parametrov konvolučnej vrstvy je počet hodnôt konvolučného jadra plus jeden parameter pre reprezentáciu parametru bias (ak je použitý). Parameter bias je hodnota, ktorá sa pripočítava k výsledku súčinu, v rovnici 4.4 označený ako b. Z pohľadu neurónu je rozdiel v tom, že neurón v plne prepojenej vrstve vidí celý vstup vrstvy, zatiaľčo neurón v konvolučnej vrstve vidí len malý úsek vstupu (určený veľkosťou filtra).

$$f(x_i, \vec{w}, b) = \vec{w}x_i + b \tag{4.4}$$

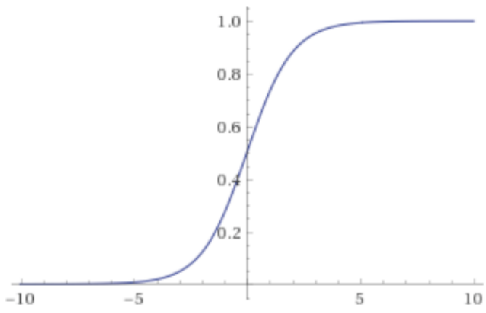
4.3 Aktivačná vrstva

Aktivačná funkcia v neurónovej sieti určuje hodnoty výstupov jednotlivých neurónov (ich aktivácie) na základe ich vnútorného potenciálu (vnútorný potenciál sa vypočíta násobením váh so vstupom). Ide o nelineárnu funkciu operujúcu nad jedným vstupným číslom. Existuje mnoho druhov aktivačných funkcií, v súčasnosti najpoužívanejšie sú:

- Sigmoidová funkcia

  Sigmoidová funkcia má tvar 4.5 a jej výstupom je číslo z intervalu $(0, 1)$ 4.3. Historicky bola často používaná, v súčasnosti jej popularita klesla. Prvým problémom je, že dochádza k saturácii aktivácie neurónov. Vstupy do aktivačnej funkcie sa môžu pohybovať v rozsahu vysokých resp. nízkych hodnôt a funkcia ich takmer zhodne mapuje na hodnotu 1 resp. O. To spôsobí, že gradient má takmer nulové hodnoty a spätný tok informácie sieťou je takmer žiaden. Druhým negatívom tejto aktivačnej funkcie je, že jej interval výstupných hodnôt nemá stred v hodnote 0. To znamená, že nasledujúce vrstvy neuronovej siete nebudú dostávať hodnoty, ktoré sú centrované v nule.

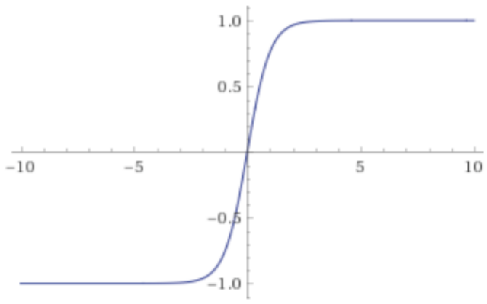
$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.5}$$



Obrázek 4.2: Simoidová aktivačná funkcia

- Tanh

  Hyperbolický tangens ako aktivačná funkcia má výhodu voči sigmoidovej funkcii v tom, že jej obor hodnôt je (-1,1) a má teda stred v hodnote 0.

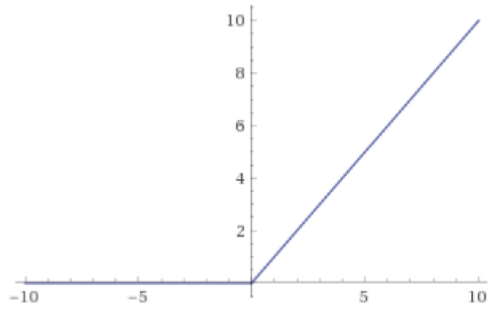


Obrázek 4.3: Aktivačná funkcia tanh

- ReLU

  Skratka ReLU označuje tzv. Rectified Linear Unit a je momentálne najpoužívanejšou aktivačnou funkciou 4.4. Ide vlastne o spojenie dvoch lineárnych funkcií, typická implementácia je uvedená v rovnici 4.6, čo je oproti ostatným aktivačným funkciám výpočetne nenáročné. Obor hodnôt je teda zhora neobmedzený a vďaka tomu nespôsobuje nulový gradient pri vysokých hodnotách aktivácie. Stále však môže spôsobiť, že neurón uviazne v stave, kedy sa nikdy neaktivuje a gradient na ňom ostane nulový - tento problém sa označuje ako tzv. dying ReLU.

$$f(x) = max(0, x) \tag{4.6}$$



Obrázek 4.4: Aktivačná funkcia ReLU

- Leaky ReLU

  Úprava funkcie ReLU za účelom riešenia problému dying ReLU. Namiesto nulovej hodnoty má funkcia mierne klesanie: $f(x) = 1(x < 0)(\alpha x) + 1(x >= 0)(x)$, kde α je malá konštanta.[3].

- Maxout

Funkcia realizuje skalárny súčin medzi váhami neurónu a vstupnými datami a generalizuje ReLU. Neuróny sa rozdelia do skupín a maxout funkcia vyberie spomedzi neurónov v každej skupine maximálnu hodnotu, pre dva neuróny vyjadrené rovnicou 4.7. Výhodou navyše oproti ReLU je odstránenie problému dying ReLU a nevýhodou zase zdvojnásobenie počtu parametrov vo vrstve.

$$f(x) = max(w_1^T x + b_1, w_2^T x + b_1) \tag{4.7}$$

4.4 Pooling vrstva

Úlohou pooling vrstvy v architektúre konvolučnej neuronovej siete je zmenšiť rozmery vstupu. Typickým spôsobom ako docieliť zmenšenie vstupu je tzv. max pooling. Ide o výber maximálnej hodnoty z danej oblasti vstupu. Iným, niekedy používaným typom operácie v pooling vrstve je average pooling (priemerovanie hodnôt). Parametrami operácie sú podobne ako v konvolučnej vrstve rozmery filtra a krok (stride). Pri operácii pooling sa často volí veľkosť kroku tak, aby sa oblasti prikladania filtra neprekrývali.

Pooling vrstva zároveň prináša do neuronovej siete nezávislosť na posunutí vstupu, t.j. ak napríklad vstup posunieme oproti pôvodnému o jeden pixel a filter vo vrstve pooling má rozmer v danej dimenzii 2, výsledok operácie maxpooling na danej pozícii bude rovnaký (nezávisí na ktorej z dvoch pozícii sa nachádzala maximálna hodnota).

Pooling vrstva sa používa aj pre zmenšenie vstupov rôznej veľkosti na jednodnotné rozmery, vtedy je umiestnená na začiatku architektúru siete. V tomto prípade sa parametre operácie volia dynamicky, podľa veľkosti konkrétneho vstupu. V prípade, že sa pooling vrstva nachádza vnútri architektúry, tj. je predchádzaná aspoň jednou konvolučnou vrstvou, dostáva na vstupe aktivačné mapy. Operácia sa potom vykoná nad každou aktivačnou mapou osobitne. Pooling vrstva teda na rozdiel od konvolučnej vrstvy vždy zachováva počet aktivačných máp. Pre dosiahnutie zmenšenia vstupu môžme vrstvu pooling nahradiť konvolučnou vrstvou s väčším krokom, ktorá rovnako zmenší vstup, ale tieto riešenia nie sú ekvivalentné (každé má iný vplyv na výstup neurónovej siete).

4.5 Plne prepojená vrstva

Plne prepojená vneurónová rstva je vrstva v ktorej je každý neurón pripojený na každý neurón v predchádzajúcej vrstve, ale neuróny z rovnakej vrstvy navzájom prepojené nie sú. Táto vrstva je typická pre klasické architektúry neuronových sietí (feedforward networks), ale v konvolučných neurnových sieťach sa vyskytuje tiež. Môže byť v architektúre umiestnená ako skrytá vrstva alebo ako posledná, výstupná vrstva architektúry. Pokiaľ je umiestnená ako výstupná vrstva siete, slúži pri klasifikačných úlohách pre zaradenie vstupu do tried. Vtedy je počet jej výstupov (počet neurónov) rovný počtu tried klasifikácie a jej výstupy označujeme ako skóre pre jednotlivé triedy. Tieto skóre bývajú potom reprezentované hodnotiacou funkciou.

Zaujímavosťou je, že plne prepojená vrstva môže byť prevedená na konvolučnú vrstvu a naopak. Neuróny v oboch vrstvách totiž počítajú skalárny súčin svojich vstupov a váh, avšak neurón v plne prepojenej vrstve počíta nad celým vstupom vrstvy, zatiaľčo neurón v konvolučnej vrstve len nad časťou vstupu a že niektoré neuróny v konvolučnej vrstve

zdieľajú parametre (v prípade jedného filtra v konvolučnej vrstve zdieľajú všetky neuróny rovnaké parametre).

4.6 Aplikácie konvolučných neuronových sietí

Existuje niekoľko typov úloh, ktoré sa v súčasnosti efektívne riešia konvolučnými neuronovými sieťami. Hlavnými dvoma doménami, v ktorých sa v súčasnoti tieto siete používajú sú spracovanie obrazu a spracovanie reči.

V doméne spracovania obrazu ide o úlohy rozpoznávania, klasifikácie, označovania, porovnávania atď. Pre človeka nie je spracovanie obrazových vnemov zložitá vec, resp. nie je na to potrebné veľké vedomé úsilie. Je pre nás jednoduché identifikovať známe objekty na obraze, aj v prípade, že sú zobrazené z netradičných uhlov alebo je väčšina objektu zakrytá prekážkou. Rovnako jednoduché je určiť, že dva obrazy sú takmer identické bez ohľadu na posunutie, rotáciu, grafickú úpravu alebo mierne poškodenie obrazu. Pre počítač sú tieto úlohy naopak značne komplikované ak sa ich znažíme riešiť algoritmicky, ale konvolučné neuronové siete sú schopné dosiahnuť invariantnosť voči týmto transformáciám. Výhodou konvolučných neurňových sietí je aj to, že zachovávajú informáciu o pozície nejakej detekovanej vlastnosti v obraze a často slúžia na lokalizáciu objektov na obraze. V súčasnosti je veľmi bežná detekcia ľudskej tváre na fotografiách pri zaotrovaní fotoaparátu alebo pri identifikácii na sociálnych sieťach.

V doméne spracovania reči je podstatný prevod na vhodnú reprezentáciu dát. Jednou možnosťou je reprezentácia pomocou tzv. n-gramov. Modely založené na n-gramoch definujú podmienenú pravdepobnosť n-tého tokenu vzhľadom k predchádzajúcim n-1 tokenom. Token reprezentuje najčastejšie slovo, ale môže to byť aj písmeno, či slovné spojenie. Táto práca sa primárne zaoberá doménou spracovania obrazu, preto problematika spracovania reči nebude v tejto práci podrobnejšie rozobraná.

Architektúry CNN

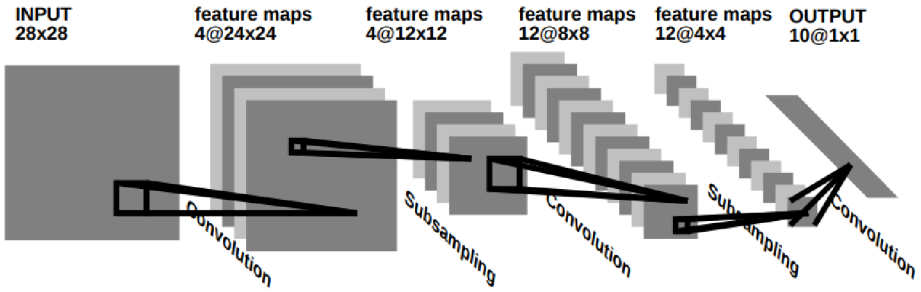
V architektúrach konvolučných neuronových sietí je možné pozorovať istú šablónu, podľa ktorej sú jednotlivé vrstvy usporiadané. Predpokladáme vyššie vymenované vrstvy, označené skratkami: Conv (konvolučná vrstva), ReLU (akivačná vrstva), Pool (vrstva pooling) a FC (plne prepojená vrstva). Ako najjednoduchšiu architektúru konvolučnej neuronovej siete môžme označiť takú, ktorá obsahuje jednu konvolučnú vrstvu, aktiváciu a plne prepojenú vrstvu 4.8. Konvolučné neuronové siete sa však typicky skladajú z niekoľkých skrytých vrstiev, často (ale nie nutne) poskladané podľa šablóny 4.9.

Skryté vrstvy sieti umožňujú zachytiť viac vlastností a tiež väčšiu abstrakciu. Pre ilustráciu je uvedená ukážka neuronovej siete typu LeNet 4.5. Výstupy jednotlivých vrstiev sa označujú ako aktivačné mapy, pričom pre súhrnné označenie vstupu siete a aktivačných máp možno použiť výraz feature map.

$$Conv- > ReLU- > FC \tag{4.8}$$

$$Conv- > [[ReLU- > FC] \times n]- > Pool] \times m- > [FC- > ReLU] \times k- > FC[3] \tag{4.9}$$

Jednou zo záladných architektúr je tzv. LeNet [6].



Obrázek 4.5: Ukážka architektúry konvolučnej neuronovej siete s názvom LeNet-1. Každý zo sivých štvorcov reprezentuje mapu vlastností (feature map). Rozmery máp sú v tvare $d@m \times n$, kde d označuje hĺbku, danú počtom filtrov v predchádzajúcej vrstve, m a n sú rozmery jednej mapy, t.j. rozmery po aplikácii jedného konvolučného filtra. Convolution označuje konvolučnú vrstvu a subsampling vrstvu pooling. Zdroj:[6]

Kapitola 5

Návrh programu

Navrhnutý program pre trénovanie konvolučných neuronových sietí má plniť predovšetkým demonštračnú úlohu. Cieľom je vytvoriť študijnú pomôcku pri štúdiu konvolučných neuronových sietí, ktorá pomôže intuitívne pochopiť princíp činnosti týchto neuronových sietí. Program vizualizuje architektúru neuronovej siete, algoritmus učenia a demonštruje činnosti jednotlivých typov vrstiev konvolučnej neuronovej siete. Program bude obsahovať jednoduché grafické užívateľské rozhranie a bude implementovaný v programovacom jazyku Python.

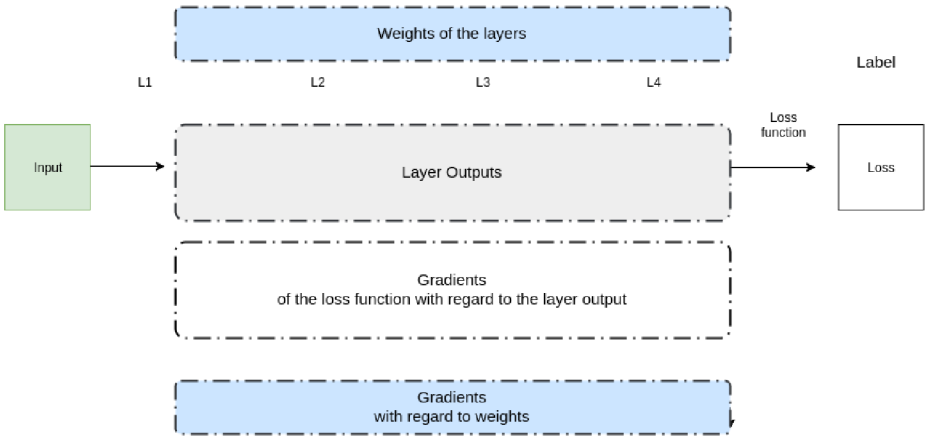
V súčasnej dobe je na internete dostupných mnoho knižníc a frameworkov pre prácu s neuronovými sieťami, ktoré umožňujú abstrahovať od detailov fungovania neuronových sietí. Ich druhou nespornou výhodou je optimalizovaná implementácia pre konkrétne CPU alebo GPU. Ide napríklad o framework Tensorflow, Coffee, Torch alebo Keras. Vysoká forma abstrakcie ponúkaná týmito knižnicami dovoľuje sústredenie na design architektúry a výber hyperparametrov, ale na druhú stranu neumožňuje hlbšie pochopenie princípu fungovania neuronových sietí. V tejto práci sú preto nie sú využité a samotný algoritmus výpočtu neuronovej siete je svojpomocne implementovaný podľa dostupnej literatúry, za cenu nevyužitia optimalizovaných matematických operácií, ktoré ponúkajú tieto knižnice. Tento prístup k návrhu síce znamená obmedzenie v otázke rýchlosti výpočtu a teda aj v komplexnosti architektúry implementovanej neurónovej siete, ale toto hľadisko nie je prioritou zadania. Pri stavbe neurónovej siete je na výber viacero architektúr a preto bude súčaťou úlohy experimentovaním vybrať najvhodnejšie varianty.

Program bude fungovať v dvoch základných módoch.

- Prvým módom je demonštračné krokovanie algoritmu výpočtu. Neuronová sieť jednoduchej architektúry je zobrazená prostredníctvom jednotlivých vrstiev a váh (filtrov) v nich. Užívateľ v grafickom rozhraní bude môcť krokovať výpočet a sledovať priechod vstupných dát sieťou (konvolučnými, aktivačnými a pooling vrstvami) a tiež spätný chod - propagáciu chyby sieťou, výpočet gradientov a aktualizáciu váh.
- Druhým módom je výpočet bez demonštračných prvkov, kedy je cieľom natrénovať sieť pre riešenie klasifikačnej úlohy. Vo fáze trénovana bude výstup programu len konzolový resp. do sôboru. Sieť bude po každej epoche ukladať svoj stav a tiež bude hodnotená na trénovacej množine. Natrénovaná sieť sa môže sa použiť na klasifikáciu buď celého datasetu alebo vybraného vzorku.

5.1 Demonštrácia v GUI

Grafické rozhranie bolo navrhnuté s cieľom zobraziť elementárne kroky toku dát sieťou. Zobrazuje krok po kroku putovanie vstupu cez jednotlivé vrstvy, prostredníctvom výstupov vrstiev. Na rovnakej obrazovke prebieha potom spätný tok dát sieťou - backpropagation, ktorý je znázornený o riadok nižšie prostredníctvom derivácií vstupov danej vrstvy. Okrem toho sú zobrazené tabuľky váh vrstiev a ekvivalentne, derivácií týchto váh. Derivácia vstupu sa predá predchádzajúcej vrstve, ktorá na základe nej (a derivácie svojho vstupu z doprednej fázy) určí deriváciu výstupu siete podľa svojho vstupu. Tú potom predá predchádzajúcej vrstve a tak ďalej, až kým sa týmto postupom nedosiahne prvá vrstva v architektúre.
Derivácie váh v danej vrstve sa nepredávajú ďalším vrstvám, ale slúžia pre aktualizáciu váh. Tabuľka váh je umiestnená nad danou vrstvou, derivácie váh pod danou vrstvou a výstupy a vstupy putujúce medzi vrstvami sú zobrazené v tabuľkách medzi vrstvami 5.1.



Obrázek 5.1: Schéma grafického užívateľského rozhrania zobrazujúca rozmiestnenie jednotlivých údajov.

Vstupy

V grafickom rozhraní sa ešte nachádzajú dve tabuľky označené zelenou farbou, ktoré sú editovateľné. Ide o vstupné dáta siete (Tabuľka *Input*) a požadované výstupné dáta (Tabuľka *Labels*). Tabuľky sú na začiatku inicializované hodnotami a užívateľ môže manuálne zadávať vlastné hodnoty a sledovať reakcie siete. Vstupom môžu byť ľubovoľné desatinné čísla. V GUI sa počíta s veľkosťou maximálne 5 číslic, plus znamienko, v prípade väčšieho čísla výpočet nie je ovplyvnený, ale zobrazenie v rozhraní môže byt neprehľadné (prítomnosť posuvníka v tabuľkách). Bežným postupom trénovania siete by bolo zaistiť, aby boli vstupné dáta nejakým spôsobom normalizované, ale vzhľadom k tomu, že v tejto časti úlohy nejde o faktické trénovanie siete, ale o demonštráciu algoritmu, vstupné dáta sa po prečítaní ďalej neupravujú. Užívateľ ale inituitívne môže získať predstavu o vhodných vstupných datách z inicializačných hodnôt.
Vstupné dáta sú z užívateľského rozhrania načítané v okamžiku začatia novej iterácie algoritmu, hodnoty labelu sú načítané pred vyhodnotením výstupu siete. Užívateľ teda vstup môže nastaviť v ľubovoľnej fáze a nespôsobí to prerušenie bežiacej iterácie algoritmu, ale jeho vstupy sa použijú v nasledujúcej iterácii.

Tabuľka pre zadávanie labelu, teda želaného výstupu siete sa nachádza vpravo hore, nad hodnotiacou funkciou. Pri klasifikácii do tried je jej sémantika taká, že index, na ktorom je jednotka, označuje index triedy, do ktorej daný vstup skutočne patrí. Ostatné indexy sú teda nulové. Toto obmedzenie sa viaže na konkrétny typ úlohy, ktorý v demonštračnej časti nezohľadňujeme, preto je jej nastavenie užívateľom ponechané bez obmedzenia. Rozlišuje sa len medzi nulovou a nenulovou hodnotou daného indexu a teda je možné testovať aj scenáre, keď budú všetky indexy nenulové alebo nulové. Vysvetlenie, ako súvisí label s hodnotením výstupu siete (hodnotiacou funkciou) je v teoretickej časti, kapitole o hodnotiacej funkcii SVM.

Výstupy vrstiev

Vrstvy neurónovej siete sú znázornené zvislými čiarami a označené používanými skratkami:

Conv: Konvolučná vrstva

ReLU: Aktivačná funkcia (Rectified Linear Unit)

Pool: Pooling vrstva

FC: Plne prepojená vrstva (Fully Connected)

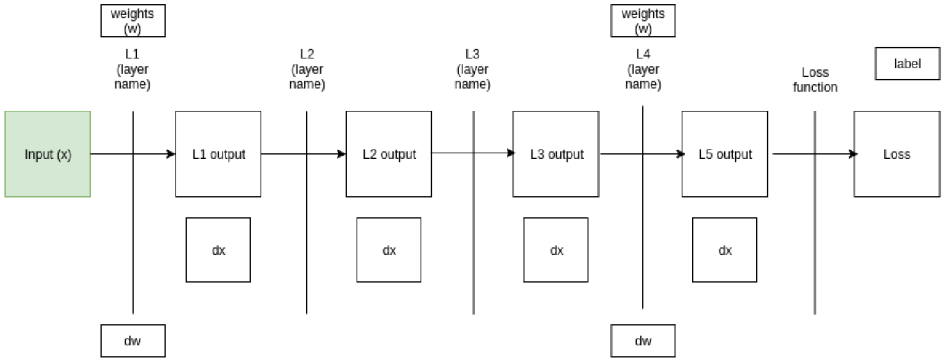
Cost: Hodnotiaca funkcia

Výstupy vrstiev sú zobrazené v tabuľkách umiestnených medzi jednotlivými vrstvami, rozdelených do dvoch riadkov. Vrchný riadok zobrazuje dopredný prechod sieťou a spodný zase spätný prechod (backpropagation). Každá tabuľka vo vrchnom riadku, nachádzajúca sa medzi vrstvou l_n a l_{n+1} reprezentuje výstup vrstvy l_n a zároveň vstup vrstvy l_{n-1}. Vrstva prijme na vstupe dáta, na základe svojej triedy vypočíta výstup a vráti ho na výstupe, pričom iterácia v doprednom prechode prebieha zľava doprava.
Po dosiahnutí poslednej vrstvy v architektúre, čo je funkcia hodnotiaca výstup siete na základe zadaného labelu (tabuľka *Label*), začne fáza backpropagation. Tá prebieha v obrátenom poradí vrstiev, teda sprava doľava a výstupy vrstiev sú zobrazené v tabuľkách v spodnom riadku. Tabuľka medzi vrstvou l_n a l_{n+1}, označená dx reprezentuje deriváciu chyby siete vzhľadom na vstup vrstvy $l_n + 1$ a zároveň vstup pre vrstvu l_n, ktorá na základe reťazového pravidla vypočíta deriváciu chyby vzhľadom na svoje vstupy. Princíp backpropagation a použitie reťazového pravidla je vysvetlené v prvej kaptole 3.2. Účelom počítania gradientu vzhľadom na vstupy je použitie pre výpočet gradientu vzhľadom na váhy predchádzajúcej vrstvy a tento gradient sa použije na aktualizáciu hondôt váh vrstvy.

Váhy

Vrstvy tvoriace neurónovú sieť obsahujú váhy-parametre, ktoré sú predmetom učenia. V prípade tejto práce ide teda o vrstvy konvolučnú a plne prepojenú. Vrstva Pooling neobsahuje vlastné parametre a rovnako vrstvy ReLU a hodnotiaca funkcia, pretože len modelujú určitú funkciu. V niektorých architektúrach je ReLU modelovaná ako súčasť inej vrstvy (väčšinou konvolučnej), inde ako samostatná vrstva, tu bolo zvolené druhé riešenie.
Váhy pre vrstvy konvolučnú a plne prepojenú sú zobrazené vo vrchnej časti, nad zvislou čiarou reprezentujúcu danú vrstvu. Typicky obsahuje jedna konvolučná vrstva niekoľko filtrov, teda niekoľko matíc váh, ale z dôvodu prehľadnosti obsahuje zobrazená neurónová sieť

len jeden filter. Váhy sa aktualizujú vo fáze backpropagation, po výpočte derivácií v danej vrstve. V grafickom rozhraní ich aktualizácia prebehne v rovnakom kroku ako zobrazeni derivácie vstupu a aktuálnych váh. Pre ich aktualizáciu je použitá metóda popísaná v úvodnej kapitole 3.6. Hypermarametrom, ktorý určuje ako prudko sa budú váhy na základe gradientu meniť je tzv. learning rate, ktorá je v programe zvolená konštantne (pre demonštráciu je použitá defaultná hodnota, v učiacej časti sa s jej hodnotou experimentovalo).
Spodný riadok tabuliek obsahuje tabuľky derivácií celkovej chyby siete vzhľadom k jednotlivým váham. Tabuľky sa nachádzajú pod odpovedajúcou vrstvou neurónovej siete a ich hodnota sa aktualizuje vo fáze backpropagation, spolu s tabuľkami derivácií podľa vstupov vrstvy (dx) a aktualizáciou hodnôt váh (dw). Princíp výpočtu obsahu tabuľky derivácií vzhľadom k váham je vysvetlený v prvej kapitole, v časti 3.6.



Obrázek 5.2: Detailná schéma umiestnenia prvkov v grafickom užívateľskom rozhraní. Derivácie podľa váh sú označené dw, podľa vstupov dx.

Ovládanie

Program obsahuje tlačidlá *Step* a *Reset* a toolbar s nápovedou *Help*. Tlačidlo *Step* slúži na krokovanie výpočtu, jedno stlačenie znamená posun o jednu vrstvu dopredu resp. dozadu. Aktuálny stav krokovania algoritmu je znázornené farebným označením práve aktualizovaných tabuliek. Tlačidlo *Reset* slúži na uvedenie programu do počiatočného stavu, teda nastavenie vstupov a váh na inicializačné hodnoty a stavu algoritmu na počiatočný stav.

5.2 Trénovanie siete pre riešenie problému

Druhou časťou práce je potom natrénovať konvolučnú neuronovú sieť pre riešenie vybraného problému. Je vybraný problém klasifikácie obrázkov za datasete MNIST[5], ktorý sa bežne používa pre evaluácií klasifikačných konvolučných neuronových sietí. Súčasťou úlohy bolo aj nájsť architektúru siete a nastavenie hyperparametrov, pre ktorú bude klasifikátor dosahovať najlepšie výsledky.

Trénovanie neurónovej siete si vyžaduje čas a supervíziu procesu učenia. Učenie prebieha tak, že na vstup neuronovej siete vložíme trénovacie dáta (po jednom alebo po určených množstvách tzv. batch) a tie prejdú sieťou od prvej k poslednej vrstve a zase naspäť, do prvej vrstvy siete. Prechod všetkých trénovacích dát sieťou práve jedenkrát nazývame jedna trénovacia epocha. Učenie siete vyžaduje niekoľko epôch, ich počet je ale treba určiť ex-

perimentálne. Líši sa v závislosti od typu riešenej úlohy, podľa povahy vstupných dát, ich rozmanitosti a ďalších faktoroch.

Pri trénovaní siete sa používa regularizačná technika early stopping 3.3. Teda sa priebežne ukladá stav siete a po skončení trénovania sa vyberie model s najlepším hodnotením, čo nemusí byť nutne model z poslednej iterácie. Často sa stáva, že sieť s počtom iterácii horšie klasifikuje trénovacie dáta, čo je znakom pretrénovania 3.3.

Kapitola 6

Implementácia

Programové riešenie je implementované v jazyku Python (verzia 2.7) ako program s konzolovým výstupom a grafickým rozhraním. Grafické užívateľské rozhranie je implementované s využitím knižnice PyQt, čo je verzia populárneho grafického frameworku Qt pre jazyk python. PyQt je verzia multiplatformného frameworku Qt, originálne napísaná v jazyku C++. Pozitívom je takmer jednotný vzhľad na rôznych plaformách, čo je výhodou vzhľadom na jeho využiteľnosť pri výuke. Grafická verzia programu plní demonštračnú úlohu, konzolová zase slúži na skutočné natrénovanie siete a klasifikáciu. Obe verzie programu však používajú rovnakú implemnetáciu neurónove siete, knižnicu *Cnn.py*.

6.1 Implementácia CNN

Konvolučná neuronová sieť je implementovaná ako modul (Cnn.py) z dôvodu jednoduchého použitia v programoch (pomocou mechanizmu import). V tabuľke nižšie sú uvedené jednotlivé triedy v programe spolu s vysvetlením, čo reprezentujú 6.1.

Konvolučná neuronová sieť je objektom triedy *ConvNetwork* a obsahuje objetky ďalších tried, ktoré reprezentujú jednotlivé vrstvy siete. Vrstvy neurónovej siete sú objekty príslušnej triedy a sú uložené v zozname s názvom *layers*.

Každá trieda reprezentujúca vrstvu obsahuje v sebe implementáciu výpočetného kroku vpred (*step_forward*) a krok operácie backpropagation (*step_backprop*). Vďaka tomu je možné z vonkajšieho pohľadu pracovať so všetkými vrstvami jednotným spôsobom. Inicializácia prebieha pri vytváraní objektu v metóde *___init___*.

Trieda	Význam
ConvNetwork	konvolučná neuronová sieť
ConvLayer	konvolučná vrstva
ReLU_Conv	aktivácia ReLU konvolučnej vrstvy
ReLU_FC	aktivácia ReLU plne prepojenej vrstvy
PoolLayer	vrstva pooling
CostFun	funkcia loss (softmax)

Ďalej budú v tejto kapitole uvedené implementačné riešenia kľúčových častí programu.

Inicializácia parametrov

Pred začatím trénovania neurónovej siete je potrebné nastaviť parametre v jednotlivých vrstvách (váhy) na počiatočnú hodnotu. Určenie počiatočnej hodnoty sa ukazuje byť podstatným pre úspešné trénovanie siete. Prvou myšlienkou by mohlo byť inicializovať váhy na rovnaké hodnoty, napríklad na hodnotu 0. Táto idea by však spôsobila problémy, pretože na začiatku by každý neurón v sieti vypočítal rovnaký výstup a teda aj rovnaký gradient, na základe ktorého upraví svoje váhy o rovnakú hodnotu. Preto je treba váhy nastaviť na rôzne hodnoty, v tejto práci bola zvolená inicializácia podľa normálneho rozloženia, so stredom $\mu = 0$ a rozptylom $\sigma = 0.01$, na základe odporúčania [3]. V jazyku python má inicializácia tvar 6.1.

$$weights = 0.01 \times np.random.randn(...) \tag{6.1}$$

Normalizácia vstupných dát

Vstupné dáta siete sú pred vstupom do učiaceho algoritmu normalizované (to sa netýka užívateľom zadaných dát v GUI). Normalizácia je implementovaná tak, že každá hodnota trénovacích aj testovacích dát je prevedená do intervalu $< -0.5, 0.5 >$ tak, že pôvodná hodnota je podelená maximálnou hodnotou (v prípade obrazového vstupu, hodnota 255) a je od nej odčítaná hodnota 0.5 6.2.

$$[(x/255.0 - 0.5) \quad for \quad x \quad in \quad image] \tag{6.2}$$

Určenie rozmerov

Každá vrstva neurónovej siete dostane pri inicializácii informáciu o tom, aké rozmery bude dostávať pri výpočte na vstupe a podľa nich potom určí rozmer svojho výstupu. Nech $output_h$ označuje výšku a $output_w$ šírku aktivačnej mapy na výstupe konvolučnej a pooling vrstvy. Vstupný rozmer vrstvy označme $[input_w, input_h, input_c]$. Rozmer filtrov neurónovej siete má obecne trojdimenzionálny tvar, označme $[filter_h, filter_w, filter_c]$.

Tvar výstupu konvolučnej neurónovej vrstvy za predpokladu f konvolučných filtrov je $[output_h, output_w, f]$ kde $output_h$ a $output_w$ sú určené rovnicami 6.3 a 6.4.

$$output_h = 1 + \frac{input_h + 2 \times padding - filter_h}{stride} \tag{6.3}$$

$$output_w = 1 + \frac{input_w + 2 \times padding - filter_w}{stride} \tag{6.4}$$

Vrstva pooling má pri rovnakom značení výstup tvaru $[output_h, output_w, input_c]$, kde $output_h$ a $output_w$ sú určené rovnicami 6.5 a 6.6.

$$output_h = 1 + \frac{input_h - filter_h}{stride} \tag{6.5}$$

$$output_w = 1 + \frac{input_w - filter_w}{stride} \tag{6.6}$$

Vrstva realizujúca aktiváciu ReLU zahcováva rozmery vstupu a výstup plne prepojenej vrstvy je vektor s počtom prvkov rovným počtu výstupných tried.

Implementácia konvolúcie

Pri výpočte konvolúcie vyvstáva otázka, či a aký druh zarovnania (padding) použiť. V tejto implementácii padding nie je použitý, aj vzhľadom na povahu klasifikačnej úlohy. Argument, prečo padding použiť je hlavne ten, že sa nestratí informácia na okrajoch vstupu, avšak v tejto práci sú však použité vstupné dáta MNIST obsahujú po okrajoch prázdne miesto - informácia je centrovaná v strede obrazu.

Konvolúcia pri doprednom prechode sieťou je implementovaná bez otáčania filtru (kernelu). Podľa definície teda ide fakticky o koreláciu, ale na výsledok to nemá vplyv. Implementácia korelácie namiesto konvolúcie sa niekedy volí z vizualizačného dôvodu, čo je aj prípad tejto práce.

6.2 Implementácia učenia

Učiaci algoritmus je implementovaný s predpokladom dvoj- alebo troj-rozmerných vstupných dát siete (dvojrozmerné dáta sú prevedené na trojrozmerné pridaním dimenzie s veľkosťou jedna). Pri učení je použitý konštantný hyperparameter learning rate s veľkosťou 0.01, hodnota bola určená exprimentálne.

6.2.1 Backpropagation v konvolučnej vrstve

Výstupom sú gradient dx - parciálna derivácia výstupu siete podľa vstupov konvolučnej vrstvy a gradient dw - parciálna derivácia výstupu siete podľa váh vrstvy.

- výpočet $\vec{dx}$

  Gradient dx má rovnaké rozmery ako vstup konvolučnej vrstvy x. Položka $dx_{i,j}$ gradientu $dx = \nabla_{\vec{x}} L$ sa vypočíta ako v rovnici 6.7. L označuje ohodnotenie výstupu siete hodnotiacou funkciou a $\vec{out}$ výstup aktuálnej vrstvy (vrstva dostane gradient funkcie loss vhľadom k svojmu výstupu).

  $$dx_i = \sum_k \frac{\partial L}{\partial out_k} \frac{\partial out_k}{\partial x_i} \tag{6.7}$$

  Jedna položka vstupu x má vplyv na malý počet položiek výstupu (aktivačnej mapy) vrstvy. Konkrétne je danou položkou vstupu ovplyvnený počet položiek aktivačnej mapy rovný počtu položiek konvolučného filtra. Gradientu vzhľadom k vstupom konvolučnej vrstvy sa bude propagovať ďalej po sieti v spätnom smere a bude slôžiť na určenie ďalších gradientov pomocou reťazovéjo pravidla. Rošírením základnej implementácie by bolo pridanie viacerých filtrov do jednej vrstvy, ďalší rozmer jej vstupu (farebný kanál pri obrazových datách) prípadne padding. Efektívnosť by sa zvýšila aj využitím transformačných operácií - prevádzať matice na vektory.

- výpočet $\vec{dw}$

  Gradient vzhľadom k váham vrstvy sa vypočíta podobným spôsobom ako gradient vzhľadom k vstupom. Jeho rozmery sa opäť zhodujú s pôvodnými rozmermi matice váh. Vypočítaný gradient sa potom použije pri aktualizovaní hodnôt váh, pričom miera zmeny hodnoty je daná hyperparametrom learning rate.

Kapitola 7

Demonštrácia programu

Táto kapitola popisuje použitie a ovládania implementovaného programu. Program má dva základné prípady použitia a to demonštráciu výpočtu konvolučnej neurónovej siete (program *demo.py*) a demonštráciu klasifikácie obrázkov (konzolový program *classify_demo.py*). Obe prípady použitia využívajú rovnakú knižnicu implementujúcu neuronovú sieť. Pre úplnosť sú súčasťou riešenia aj programy pre trénovanie konvolučnej siete a pre klasifikáciu pre štatistické účely.

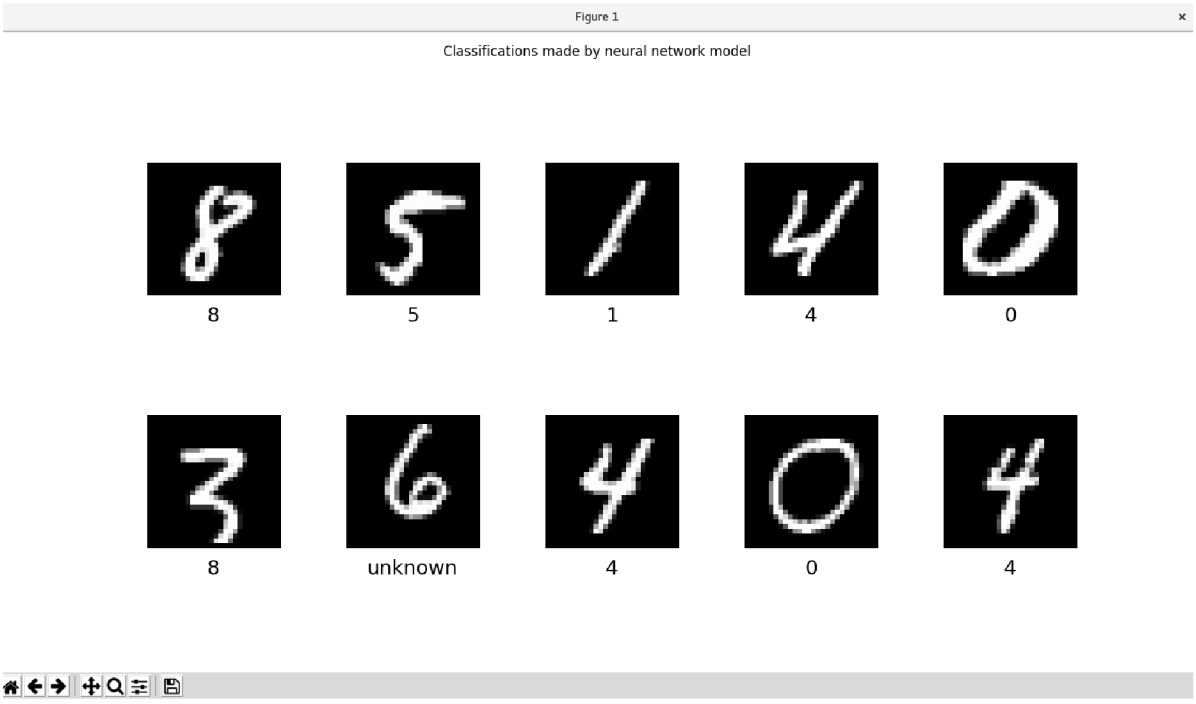
7.1 Demonštrácia a ovládanie programu pre klasifikáciu

Program, ktorý demonštruje klasifikáciu sa nazýva *classify_demo.py* a spúšťa sa z príkazového riadku. Po spustení sa vytvorí neurónová sieť na základe uloženého modelu zo súboru, t.j. zvolí sa príslušná architektúra a váhy v sieti sa nastavia na naučené váhy podľa modelu. Následne sa načíta trénovací dataset (s veľkosťou 10000 vzoriek) a náhodne sa vyberie 10 vzoriek. Tie sa potom predajú neurónovej sieti na klasifikáciu. Výsledok klasifikácie je zobrazený v grafickom okne 7.1, kde je pod každým vizualizovaným obrázkom číslo, ktoré reprezentuje výsledok klasifikácie. V prípade, že program predpovedá pre daný obrázok všetkým triedam rovnakú pravdepobonosť, zobrazí sa namiesto čísla text *unknown*. Správnosť klasifikácie je možné jednoducho visuálne zhodnotiť, keďže klasifikovanými obrázkami sú ručne písané číslice, ľahko čitateľné pre človeka.

Typické použitie je napr. $classify\_demo.py \quad -m \quad Conv2x3\_models/cnn\_model9.py$, čiže spustenie s posledným modelom (číslovka 9 označuje poslednú iteráciu učenia daného modelu, modely sú číslované od 0) neurónovej siete typu $Conv2x3$. Parameter súboru s modelom $-m$ nie je povinný, v prípade, že nie je zadaný, použije sa jeho defaultná hodnota - natrénovaný model s najlepšími výsledkami. Prehľad možných argumentov programu je v tabuľke 7.1.

Argument		Význam
-h,	--help	výpis nápovedy
-m,	--model	cesta k súboru s natrénovaným modelom neurónovej siete
-p,	--mnist_path	cesta k adresáru s datasetom MNIST

Tabulka 7.1: Parametre programu *classify_demo.py*



Obrázek 7.1: Ukážkový screenshot programu *classify_demo.py*. Obrázky sú náhodne vybrané testovacie dáta, pod každým obrázkom je zobrazený výsledok klasifikácie - číslica, ktorú program rozpoznal. Na obrázku vidieť, že konrétny model (vybraný za účelom demonštrácie aj chybných predpovedí) neurčil správne číslicu 3, ktorá má istú podobnosť s predpovedanou číslicou 8 a že číslicu 6 nevedel jednoznačne klasifikovať. Predpovede sú vykonané modelom neurónovej siete vybraným užívateľom.

Klasifikácia bez demonštrácie
Program sa okrem vyššie popísaného spôsobu s demonštračnou klasifikáciou dá ešte použiť pre klasifikáciu veľkého množstva dát napríklad pre štatistické účely. Spustením programu *classify.py* sa vykoná klasifikácia trénovacích dát vybraným modelom (pokiaľ nie je parametrom zadaný, použije sa hodnota default). Argumenty programu sú uvedené v tabuľke 7.2. Klaifikácia programu je implementovaná ako vykonanie dopredného prechodu sieťou bez fázy back-propagation.

Argument		Význam
-h,	--help	výpis nápovedy
-m,	--model	cesta k súboru s natrénovaným modelom neurónovej siete
-p,	--mnist_path	cesta k adresáru s datasetom MNIST
	--full_dataset	použiť celý dataset (defaultne sa použije subset)

Tabulka 7.2: Parametre programu *classify.py*

7.2 Ovládanie programu pre trénovanie siete

Program pre trénovanie *train.py* slúži pre získanie modelu neurónovej siete, ktorý bude plniť klasifikačnú úlohu, teda klasifikovať ručne písané číslice. Typické spustenie programu môže byť bez parametrov, čo znamená, že program začne trénovať model od začiatku na trénovacom subsete dát. Prehľad možných parametrov programu pri spustení je v tabuľke 7.3. Program je možné použiť aj na pokračovanie trénovania existujúceho modelu, v tom prípade je vhodné uviesť číslo počiatočnej iterácie kvôli zachovaniu prehľadnosti číslovania priebežne ukladaných súborov s modelmi.
Učenie modelu prebieha v iteráciách, pričom po každej iterácii prebehne klasifikácia na testovacích datách a jej výsledky sa uložia do súboru *stats.out*. Tieto štatistiky sa potom môžu použiť na hodnotenie priebehu učenia, napríklad vizualizáciu chyby a presnosti klasifikácie.

Program sa učí na trénovacích dátach datasetu MNIST (parametrom je nutné špecifikovať lokáciu trénovacích dát len v prípade, že boli v rámci adresárovej štruktúry premiestnené).

Argument		Význam
-h,	--help	výpis nápovedy
-m,	--input_model	cesta k súboru s natrénovaným modelom neurónovej siete
-p,	--mnist_path	cesta k adresáru s datasetom MNIST
-o,	--out_model	meno súboru pre uloženie natrénovaného modelu
-t,	--num_iterations	počet iterácií učenia
-s,	--start_iteration	počiatočnej číslo iterácie
	--full_dataset	použiť celý dataset (defaultne sa použije subset)
-a,	--architecture	typ achritektúry neuronovej siete

Tabulka 7.3: Parametre programu *classify_demo.py*

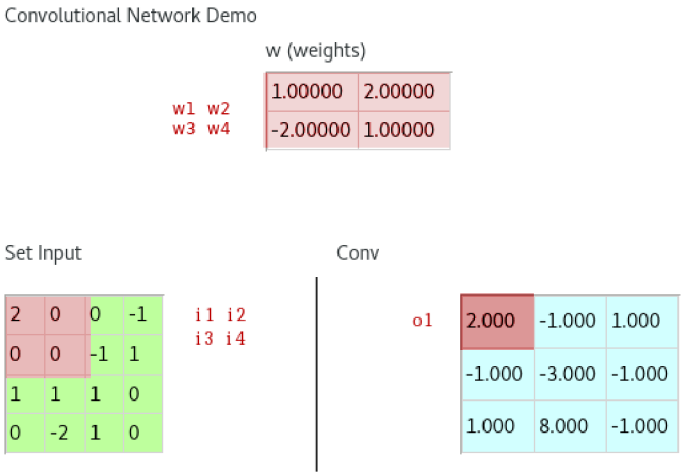
7.3 Demonštrácia GUI

Grafické rozhranie programu sa spúšťa príkazom *./demo.py*. Rozmiestnenie a vysvetlenie jednotlivých prvkov v rozhraní je uvedené v kapitole o návrhu programu 5.1. V tejto kapitole je uvedená ukážka krokovania výpočtu a činnosti jednotlivých vrstiev konvolučnej neurónovej siete vrátane vzorcov pre výpočet. Pre vysvetlenie výpočtu sú použité screenshoty z implementovaného grafického rozhrania, v ktorých sú doplnené značenia jednotlivých hodnôt vo vstupoch, filtroch a výstupoch vrstiev.

Výpočet prebieha v dvoch fázach, doprednej fáze a fáze backpropagation. Výpočet je krok za krokom vysvetlený v nasledujúcich dvoch podkapitolách (každá pre jednu fázu výpočtu).

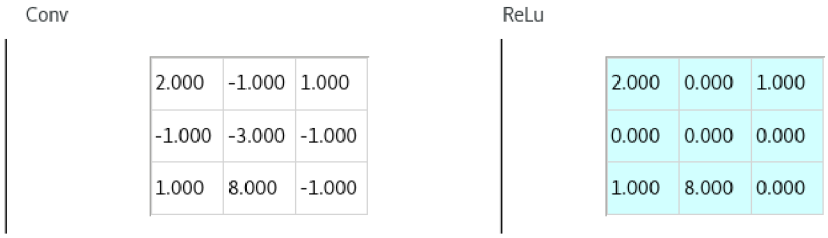
7.3.1 Dopredný prechod sieťou

Dopredný prechod sieťou zahŕňa prechody cez jednotlivé vrstvy zobrazené na obrázkoch - prechod cez konvolučnú vrstvu 7.2, aktivačnú vrstvu 7.3, vrstvu pooling 7.4, plne prpeojenú vrstvu 7.5 a vrstvu realizujúcu funkciu softmax 7.6.



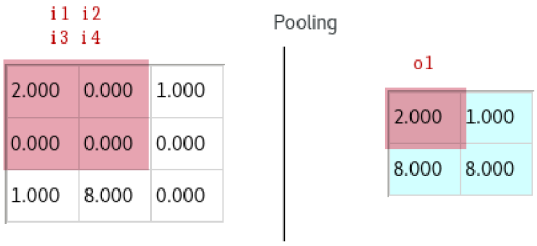
Obrázek 7.2: Demoštrácia výpočtu konvolučnej vrstvy (znázornená zvislou čiarou). V tabuľke vľavo sa nachádza vstup pooling vrstvy, hore matica váh (konvolučný filter) vrstvy a vpravo výstup vrstvy. Červenou farbou je označený aktuálny výsek vstupu, ktorý sa skalárne násobí s váhami matice a výsledok je označený v matici výstupov. Znázornená položka výstupu sa v tom prípade vypočíta rovnicou 7.1. V ďalšom kroku sa výsek vstupu posunie o jednu pozíciu doprava, podobne sa vypočíta skalárny súčin a uloží sa do aktivačnej mapy na pozíciu o jednu vpravo od o_1.

$$o_1 = w_1 \times i_1 + w_2 \times i_2 + w_3 \times i_3 + w_4 \times i_4 \tag{7.1}$$



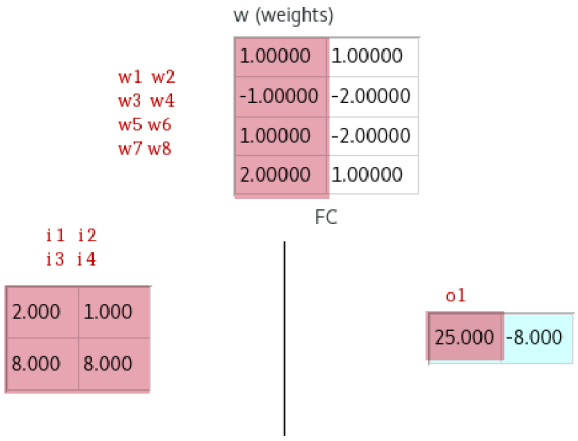
Obrázek 7.3: Demoštrácia výpočtu aktivačnej vrstvy ReLU, nasledujúcej po konvolučnej vrstve (znázornená zvislou čiarou). Aktivačná vrstva realizuje funkciu ReLU 4.3. V tabuľke vľavo sa nachádza vstup aktivačnej vrstvy, vpravo výstup vrstvy. Aktivačná funkcia pre každú hodnotu vstupu realizuje funkciu 7.2. Vidíme, že aktivačná vrstva neobsahuje vlastné parametre a nemení tvar ani rozmer vztupných dát.

$$o_j = max(0, i_j) \tag{7.2}$$



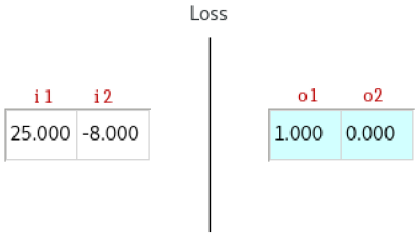
Obrázek 7.4: Demoštrácia výpočtu pooling vrstvy (znázornená zvislou čiarou), počítajúcu operáciu max pooling (výber maximálnej hodnoty). V tabuľke vľavo sa nachádza vstup vrstvy, vpravo výstup vrstvy. Pooling vrstva nebosahuje vlastné hodnoty parametrov, jej filter je určený len veľkosťou, v tomto prípade 2×2. Červenou farbou je označený aktuálny výsek vstupu, z ktorého sa vyberie maximálna hodnota a výsledok sa uloží do označeného miesta v aktivačnej mape na výstupe. Výstup sa v tom prípade vypočíta rovnicou 7.3. V ďalšom kroku sa výsek vstupu posunie o jednu pozíciu doprava, podobne sa vyberie maximálna hodnota a uloží sa do aktivačnej mapy na pozíciu o jednu vpravo od o_1.

$$o_1 = max(i_1, i_2, i_3, i_4) \tag{7.3}$$



Obrázek 7.5: Demoštrácia výpočtu plne prepojenej vrstvy (znázornená zvislou čiarou) realizujúcej skalárny súčin svojich váh a vstupu. V tabuľke vľavo sa nachádza vstup vrstvy, vpravo výstup vrstvy. Počet váh plne prepojenej vrstvy je určený počtom prvkov vstupu a počtom výstupov plne prepojenej vrstvy, v tomto prípade 4×2. Červenou farbou je označený aktuálny výsek vstupu, vypočíta sa skalárny súčin s vyznačenou časťou váh a výsledok sa uloží do označeného miesta na výstupe. Výstup sa v tom prípade vypočíta rovnicou 7.4 . V ďalšom kroku sa vstup skalárne vynásobí s druhým stĺpcom matice váh a hodnota a uloží sa do aktivačnej mapy na pozíciu o jednu vpravo od o_1.

$$o_1 = w_1 \times i_1 + w_3 \times i_2 + w_5 \times i_3 + w_7 \times i_4 \tag{7.4}$$



Obrázek 7.6: Demoštrácia výpočtu funkcie softmax. V tabuľke vľavo sa nachádza výstup z plne prepojenej vrstvy, na základe ktorého určí funkcia Softmax pravdepodobnosti jednotlivých tried (pričom súčet pravdepodobností je rovný 1). Výpočet funkcie softmax pre konkrétny prípad je vyjadrený rovnicami 7.7, kde o_1 a o_2 sú výstupy funkcie softmax (normalizované hodnoty skóre). Na obrázku je vidieť, že triedy boli prepdovedané s pravdepodobnosťami 0 a 1, čo sú pre demononštračný účel zaokrúhlené hodnoty reprezentujúce veľmi vysokú a veľmi nízku pravdepodobnosť (nulová pravdepodobnosť označuje pravdepodobnosť s exponentom 10^{-15}).

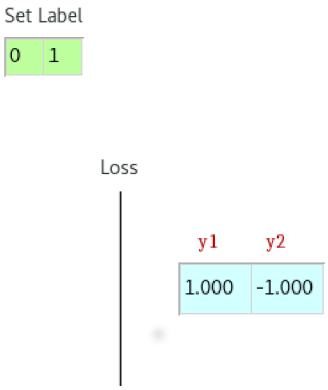
$$o_1 = \frac{e^{i_1}}{e^{i_1} + e^{i_2}} \quad o_2 = \frac{e^{i_2}}{e^{i_1} + e^{i_2}} \tag{7.5}$$

7.3.2 Fáza backpropagation

V nasledujúcom kroku výpočtu začína fáza backpropagation, čiže propagácia gradientu chyby. V GUI bude výpočet postupovať v dolnom riadku smerom zľava doprava. Vo fáze backpropagation sa pre jednotlivé vrstvy počítajú gradienty chybovej funkcie vzhľadom na ich vstupy, bližšie popísané v kapitole 3.2. Zároveň s výpočtom príslušných gradientov vzhľadom k váham sa aktualizujú hodnoty váh vo vrstve (ak daná vrstva váhy obsahuje). Prechod vrstvami je zobrazený na obrázkoch, začína od funkcie softmax 7.7 k plne prepojenej vrstve 7.8 (kde sa aktualizujú váhy 7.9), pokračuje cez vrstvu pooling 7.10, aktivačnú vrstvu ReLU 7.11 až ku konvolučnej vrstve 7.12 s aktualizáciou váh 7.13.

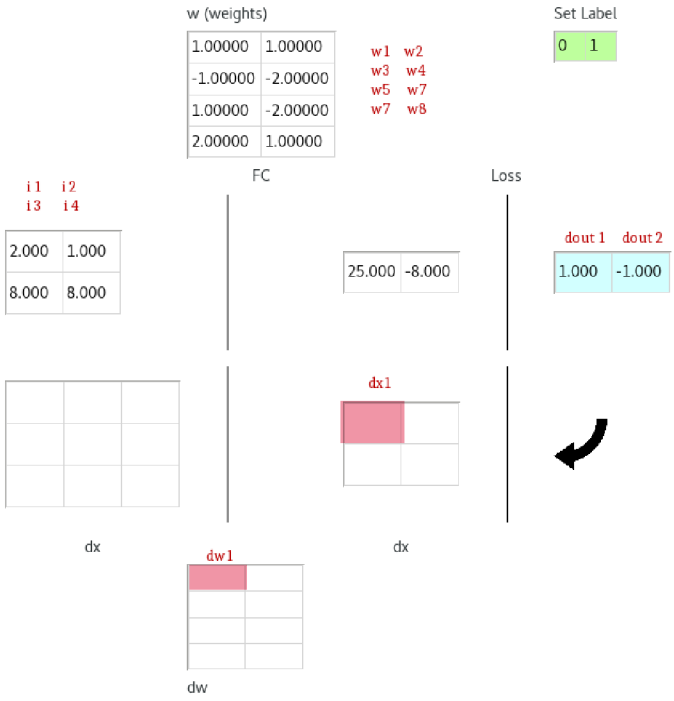
Počiatočným krokom výpočtu je určenie derivácie funkcie loss (určená funkciou softmax a cross-entropy loss). Derivácia funkcie loss podľa výstupov siete je vyjadrená rovnicou 7.6, kde L označuje funkciu loss, o_i i-tý výstup siete znázornený na obrázku 7.6 a v_{label} je vektor, ktorý má na všetkých svojich pozíciách nuly, okrem pozície označenej labelom ako správna trieda klasifikácie (tzv. one-hot vektor).

$$\frac{\partial L}{\partial y} = o_i - v_{label} \tag{7.6}$$



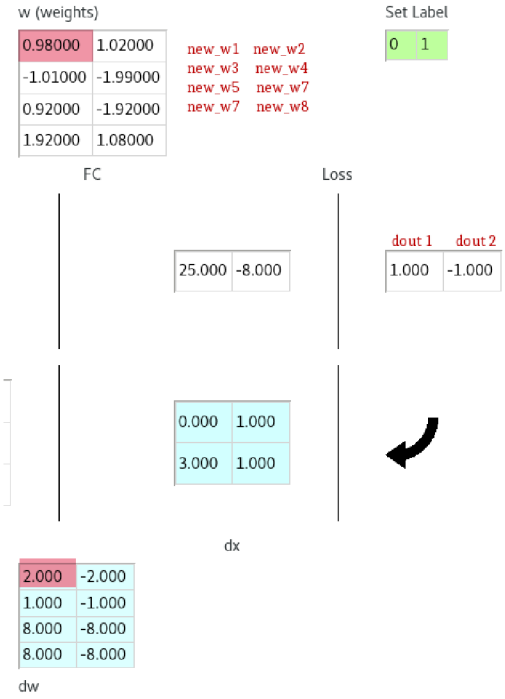
Obrázek 7.7: Prvým krokom fázy backpropagation je vypočítanie chyby pre jednotlivé skóre tried (v predchádzajúcom kroku označené o_j). Ten prebieha tak, že hodnoty pravdepodobností pre jednotlivé triedy sa zachovajú, s výnimkou triedy označenej labelom - od pravdepodobnosti tejto triedy sa odčíta hodnota 1. V ilustračnom prípade sú na výstupe len dve triedy, pričom druhá v poradí je označená ako správna 7.7.

$$loss_1 = o_1 - 0, \quad loss_2 = o_2 - 1 \tag{7.7}$$



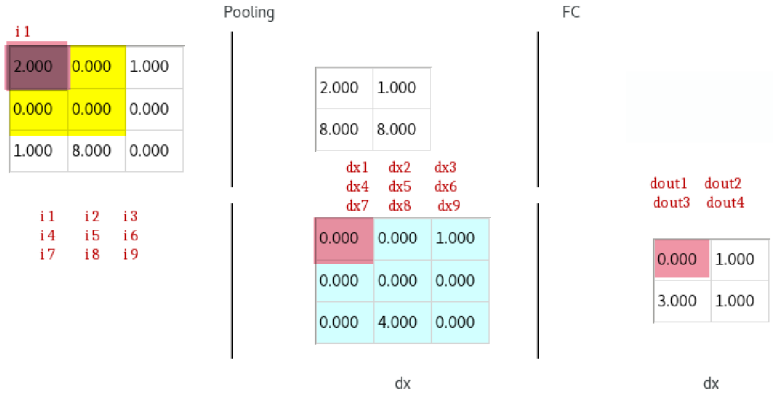
Obrázek 7.8: Spätný prechod sieťou vo fáze backpropagation a výpočet gradientov v plne prepojenej (FC) vrstve. Gradient chyby vzhľadom na vstupy siete $\vec{dx}$ sa vypočíta ako maticové násobenie gradientov z vyššej vrstvy $\vec{dout}$ a transponovanej matice váh. Výpočet hodnoty gradientu dx_1 vzhľadom na hodnotu vstupu i_1 je vyjadrená vzťahom 7.8. Podobne vynásobíme vstup s transponovanou maticou príchozích gradientov, konkrétne teda hodnotu gradientu vzhľadom na váhu w_1 získame rovnicou 7.9.

$$dx_1 = dout_1 \times w_1 + dout_2 \times w_2, \quad dw_1 = i_1 \times dout_1 \tag{7.8}$$



Obrázek 7.9: S výpočtom gradientu prebehla aktualizácia hodnôt váh v plne prepojenej vrstve na základe vzorca 3.6, pre váhy w_1 vyjadrené rovnicou 7.9, kde hodnota 0.01 reprezentuje parameter learning rate.

$$new\_w_1 = w_1 - 0.01 \times dw_1 \tag{7.9}$$



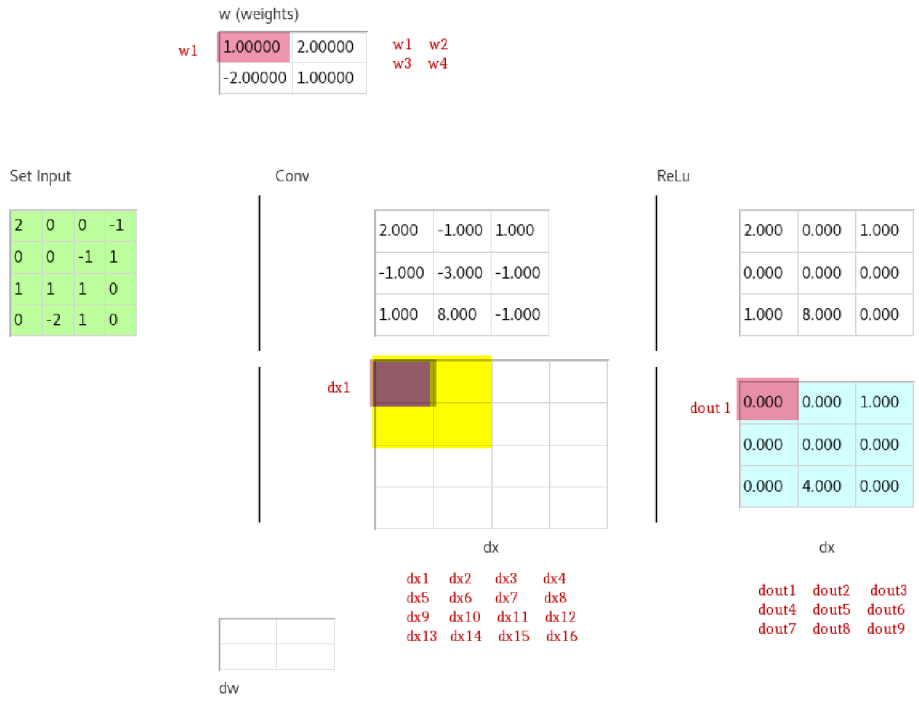
Obrázek 7.10: Krok backpropagation vrstvou pooling. Všeobecne vo vrstve pooling, hodnota gradientu dx_j vzhľadom na vstup i_j sa rovná 0 pokiaľ v danom výseku vstupu nebola maximálna hodnota i_j. Ak bola hodnota v danom výseku vstupu spomedzi ostatných maximálna, jej hodnota je $i_j \times dout_j$. Na obrázku je znázornený žltou farbou výsek definovaný štyrmi hodnotami, pričom i_1 je z nich maximálna hodnota. V rovnici 7.10 uvedený výpočet gradientu pre maximálnu hodnotu a aj pre hodnotu, ktorá nebola vybraná ako maximum. Pokiaľ sa v operácii pooling prekrývajú jednotlivé výseku vstupu, čo je náš prípad pri zvolenom korku 1 a veľkosti filtra 2×2, treba výsledky priložení filtra v jednotlivých bunkách pri spätnom prechode sčítať.

$$\begin{aligned} dx_1 &= dout_1, \\ dx_2 &= 0 + 0, \\ dx_3 &= dout_2, \\ dx_4 &= 0, \\ dx_5 &= 0 + 0, \\ dx_6 &= 0 + 0, \\ dx_7 &= 0, \\ dx_8 &= dout_3 + dout_4, \\ dx_9 &= 0 \end{aligned} \tag{7.10}$$

V obrázku 7.4 vidíme, že napríklad položka dx_3 je nastavená na hodnotu $dout_3$, pretože hodnota vstupu na indexe 3 v doprednom prechode i_3 bola kladná a v doprednom prehcode sieťou sa jej hodnota zachovala. Naopak, napríklad hodnota dx_2 má nulovú hodnotu, pretože zodpovedajúca hodnota vstupu i_2 mala zápornú hodnotu a aktivácia ReLU nastavila príslušný výstup na indexe 2 na nulu (pôvodná hodnota i_2 teda nemala žiaden vplyv na výstup neurónovej siete).

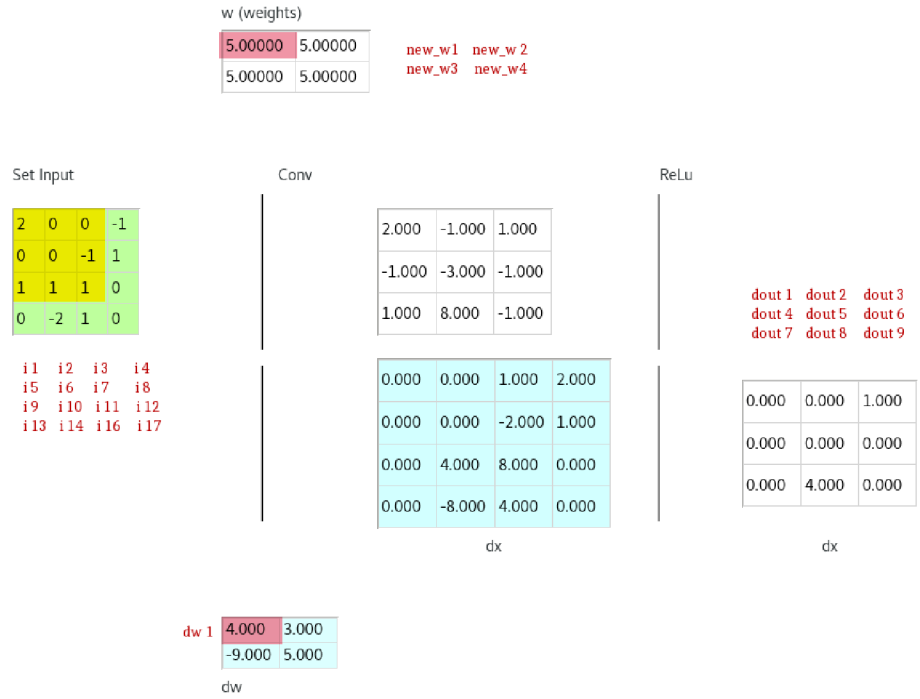
Obrázek 7.11: Krok backpropagation vrstvou aktivácie ReLU. Jednotlivé hodnoty matice dx_j sú nulové, pokiaľ hodnota s rovnakým indexom na vstupe vrstvy i_j bola záporná. Naopak, ak bolo na danej pozícii v doprednej fáze kladné číslo, derivácia podľa tohto vstupu je vyjadrená ako $dx_j = dout_j$, pre konkrétne prípady vyjadrené rovnicami 7.11.

$$\begin{aligned} dx_1 &= dout_1 \\ dx_2 &= 0 \\ dx_3 &= dout_3 \\ dx_4 &= 0 \\ dx_5 &= 0 \\ dx_6 &= 0 \\ dx_7 &= dout_7 \\ dx_8 &= dout_8 \\ dx_9 &= 0 \end{aligned} \tag{7.11}$$



Obrázek 7.12: Krok backpropagation konvolučnou vrstvou, výpočet gradientu vzhľadom k vstupom. Filter s váhami sa postupne prikladá na pozície gradientu podľa vstupu dx (prvé priloženie filtra označené žltou farbou) s pri každom priložení sa výseku pričítajú hodnoty filtra násobené číslom $dout_1$. V rovniciach 7.12 sú uvedené vzorce pre konkrétne položky gradientu.

$$\begin{aligned}
dx_1 &= w_1 \times dout_1, \\
dx_2 &= w_2 \times dout_1 + w_1 \times dout_2, \\
dx_3 &= w_2 \times dout_2 + w_1 \times dout_3, \\
dx_4 &= w_2 \times dout_3 \\
dx_5 &= w_3 \times dout_1 + w_1 \times dout_4, \\
dx_6 &= w_4 \times dout_1 + w_3 \times dout_2 + w_2 \times dout_4 + w_1 \times dout_5 \\
dx_7 &= w_4 \times dout_2 + w_3 \times dout_3 + w_2 \times dout_5 + w_1 \times dout_6 \\
&\dots
\end{aligned} \tag{7.12}$$



Obrázek 7.13: Krok backpropagation konvolučnou vrstvou - výpočet gradientu vzhľadom na váhy a aktualizácia váh. Filter s váhami sa postupne prikladá na pozície vstupu (so zvoleným krokom jeden) a pri každom priložení sa k jednotlivej váhe filtra pripočíta súčin hodnoty vstupu, ktorú prekrýva a gradientu z predchádzajúcej vrstvy $dout_j$. Na obrázku sú žltou farbou znázornené hodnoty vstupu, ktoré sa použijú pri výpočte gradientu vzhľadom k váhe dw_1, vyjadrené rovnicou 7.13. Následne sa hodnota váhy w_1 aktualizuje na základe hodnoty dw_1 podľa rovnice 7.14.

$$dw_1 = i_1 \times dout_1 + i_2 \times dout_2 + i_3 \times dout_3 + i_5 \times dout_4 + i_6 \times dout_5 + i_7 \times dout_6 + i_9 \times dout7 + i_10 \times dout_8 + i_11 \times dout_9 \tag{7.13}$$

$$new\_w_1 = w_1 - 0.01 \times dw_1 \tag{7.14}$$

Po dokončení fáze backpropagation výpočet pokračuje z aktuálneho stavu ďalšou iteráciou. Užívateľ môže v tomto bode výpočtu zmeniť hodnoty vstupov alebo labelu pre ďalšiu iteráciu.

Kapitola 8

Výsledky a experimenty

Táto kapitola popisuje experimenty s učením implementovanej neurónovej siete a diskutuje ich výsledky. Vzhľadom k tomu, že programové riešenie siete v tejto práci má v prvom rade demonštračný charakter, rýchlosť výpočtu nebola prioritou. Neuronová sieť bola implementovaná krok po kroku bežnými matematickými operáciami, bez heuristík a využitia existujúcich frameworkov, aby bolo možné vidieť jednoduché kroky, ktoré tvoria učiaci algoritmus. Preto ako demonštračná úloha bola zvolená jedna zo základných a často používaných úloh - klasifikácia ručne písaných číslic. Trénovacie a testovacie dáta pochádzajú z dostupného zdroja [5] a označujú sa ako MNIST dataset.

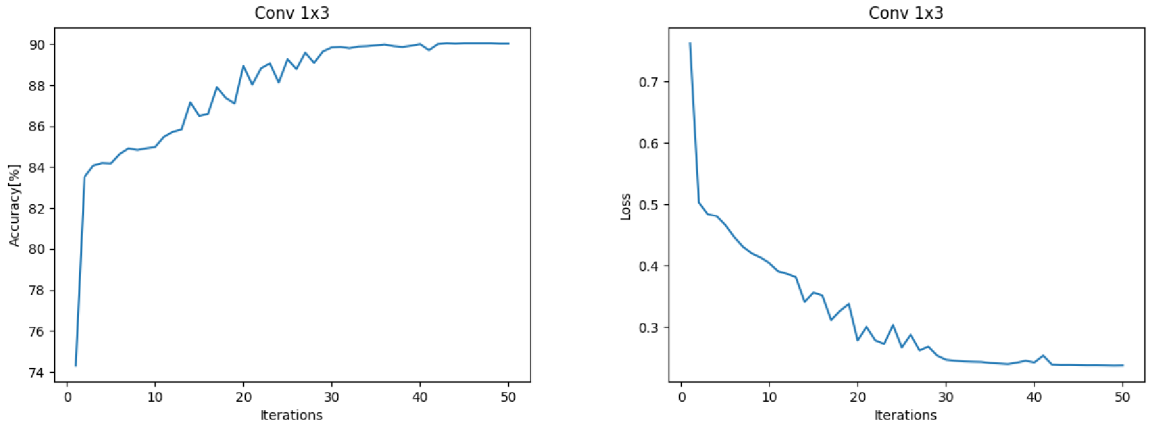
Popis datasetu
Plná verzia datasetu MNIST je rozdelená na dve datové sady, trénovacia obsahuje 60 000 vzoriek a testovacia 10 000 vzoriek dát. Dáta v testovacej a trénovacej sade majú odlišných pisateľov číslic, aby sa vylúčila závislosť na konkrétnom rukopise. Pre účely experimentov boli v niektorých prípadoch použité podsety týchto datasetov, bližšie špecifikované v jednotlivých prípadoch.

Spôsob hodnotenia
Proces učenia prebieha tak, že neurónová sieť vypočíta niekoľko iterácií (epôch), po každej iterácií uloží svoj aktuálny stav do súboru a vykoná klasifikáciu na testovacích datách. Pri klasifikácii sa za správnu považuje trieda s najvyššou pravdepodobnosťou, pričom pravdepodobnosti jednotlivých tried sa určujú na základe funkcie softmax.
V klasifikácii sa vyhodnocuje priemerná hodnota funkcie loss (v tomto prípade funkcie softmax a cross entropy loss 3.16) a presnosť klasifikácie (accuracy). Výsledok klasifikácie sa považuje za správny, pokiaľ sa trieda s najvyššou pravdepodobnosťou zhoduje s triedou určenou labelom (správny label je súčasťou trénovacieho aj testovacieho datasetu). Accuracy sa vypočíta ako percentuálny podiel počtu spávne určených vzoriek a celkového počtu vzoriek klasifikácie 8.1.

$$accuracy = \frac{correct\_predictions}{total\_predictions} \times 100 \tag{8.1}$$

8.1 Testovanie algoritmu učenia

Cieľom prvého experimentu bolo overiť, či je sieť schopná sa učiť aj s extrémne jednoduchou architektúrou. K spomínanému datasetu MNIST sú pre porovnanie k dispozícii výsledky s



(a) Accuracy (b) Loss

Obrázek 8.1: Učenie s jednou konvolučnou vrstvou s troma filtrami a počtom iterácií 50. Testovacie dáta sú podmnožinou trénovacích dát, dôvodom je overenie kapacity modelu pre riešenie danej úlohy.

rôznymi typmi neurónových sietí. [5] Najjednoduchším typom konvolučnej neuronovej siete, ktorá sa v danom porovnaní nachádza, je sieť typu LeNet-1 [6], ktorá sa skladá z troch konvolučných vrstiev so štyrmi až dvanástimi filtrami. Architektúra použitá v tomto teste sa skladá len z jednej konvolučnej siete s tromi konvolučnými filtrami a ReLU aktivačnou funkciou, pooling vrstvy a plne prepojenej vrsvy s ReLU aktiváciou. Z výsledkov učenia je zrejmé, že sieť sa učí (maximálna dosiahnutá presnosť klasifikácie 84.56%), ale pre lepší výsledok by bolo treba buď rozšíriť architektúru alebo zväčšiť trénovací dataset. V ďalšej kapitole sa bude experimentovať s týmito nastaveniami. Ďalej podľa grafov odhadujeme, že počet iterácií je možné znížiť, pretože najväčší pokrok dosiahne sieť v prvých iteráciách učenia.

8.2 Experimenty s architektúrou siete

Experimenty s architektúrou siete prebiehali v dvoch fázach. V prvej fáze bol použitý subset datasetu MNIST s veľkostou 20 000 trénovacích vzoriek a 3000 testovacích vzoriek (vybrané podľa poradia od začiatku datasetu). V druhej fáze boli vybrané dve architektúry s najlepšími výsledkami v rámci desiatich iterácií a boli natrénovaná a ohodnotené podľa kompletného datasetu (t.j. 60 000 trénovacích vzoriek a 10 000 testovacích). Okrem toho boli podľa krivky učenia niektoré siete dotatočne trénované niekoľko ďalších iterácií. Dotrénovanie sa realizovalo v prípade, keď mala krivka v grafe prudko stúpajúci charakter v posledných iteráciách a teda bol predpoklad, že výsledky siete sa ešte zlepšia. Naopak niektoré siete dosiahli svoje maximum v prvých iteráciách trénovania a ďalej sa už ich hodnotenie výrazne nemenilo.

V popisoch architektúry sú použité nasledujúce označenia:

- $Conv(n)$
  konvolučná vrstva s n filtrami (výstupom n aktivačných máp)

- *ReLU*
  aktivácia funkciou ReLU
- *Pool*
  vrstva typu max pooling
- *FC*
  plne prepojená vrstva

Testované boli nasledujúce architektúry s jednou konvolučnou vrstvou obsahujúcou len jeden filter 8.2, s jednou konvolučnou vrstvou obsahujúcou tri filtre 8.3, ďalej architektúra obsahovala dve konvolučné vrstvy 8.4, každá s troma filtrami a nakoniec architektúra s troma konvlučnými vrstvami, v každej tri filtre a to v dvoch variantách - s dvoma pooling vrstvami 8.5 a s troma pooling vrstvami 8.6.

$$Conv(1) -> ReLU -> Pool -> FC -> ReLU \tag{8.2}$$

$$Conv(3) -> ReLU -> Pool -> FC -> ReLU \tag{8.3}$$

$$[Conv(3) -> ReLU] \times 2 -> Pool -> FC -> ReLU \tag{8.4}$$

$$[Conv(3) -> ReLU -> Pool] \times 2 -> Conv -> ReLU -> FC -> ReLU \tag{8.5}$$

$$[Conv(3) -> ReLU -> Pool] \times 3 -> FC -> ReLU \tag{8.6}$$

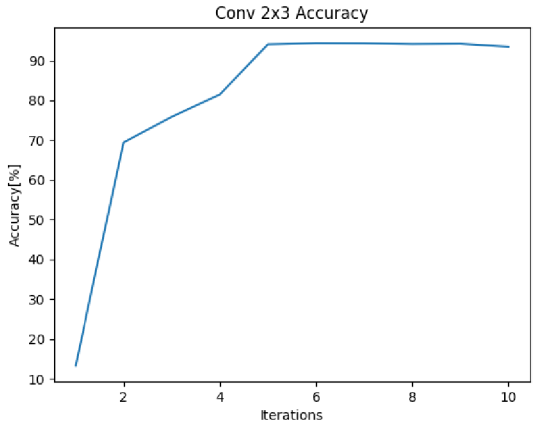
8.3 Prehľad výsledkov

Výsledky učenia neuronových sietí sú zobrazené vo forme grafov, ako závislosť hodnoty funkcie loss a presnosti (accuracy) na počte iterácií. Označenia architektúr neuronových sietí použité vo výsledkových tabuľkách odkazujú na príslušnú schémy, ktoré ich vizualizujú. Ďalej sú najlepšie výsledky každej siete v rámci desiatich iterácií uvedené v tabuľkách pre subset trénovacieho datasetu 8.1. Neurónové siete, ktoré boli trénované vyšší počet iterácií (na trénovacom subsete) sú uvedené spolu s dosiahnutými výsledkami v tabuľke 8.2 a siete natrénované na kompletnom datasete v tabuľke8.3.

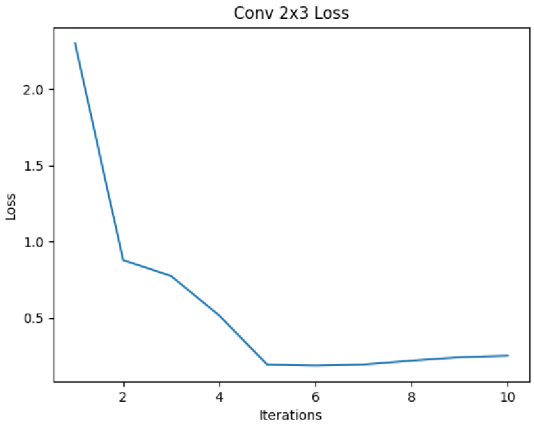
8.3.1 Trénovanie s menším datasetom

Architektúra	Accuracy	Loss
Conv2x3 8.4	94.37%	0.19
Conv1x1 8.2	83.44%	0.52
Conv3x3x 8.6	71.90%	0.92
Conv1x3 8.3	56.53%	1.32
Conv3x3 8.5	17.63%	2.30

Tabulka 8.1: Výsledky po desiatich iteráciách, použitý subset trénovacích dát

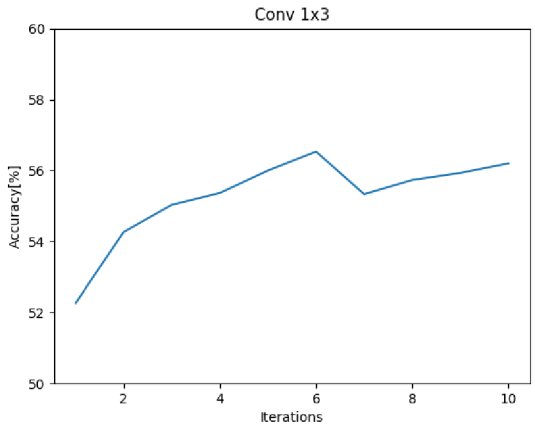


(a) Accuracy

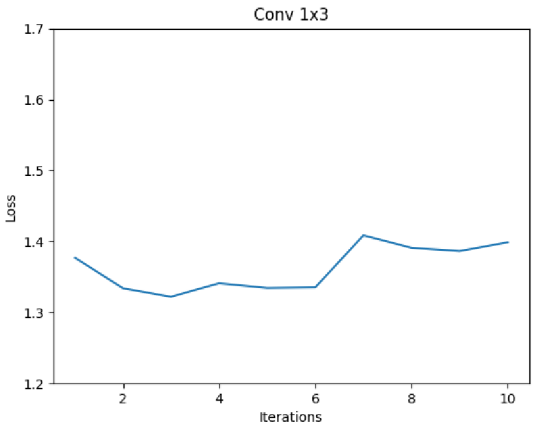


(b) Loss

Obrázek 8.2: Architektúra s dvoma konvolučnými vrstvami obsahujúcimi tri filtre. Trénované s počtom iterácií 10.

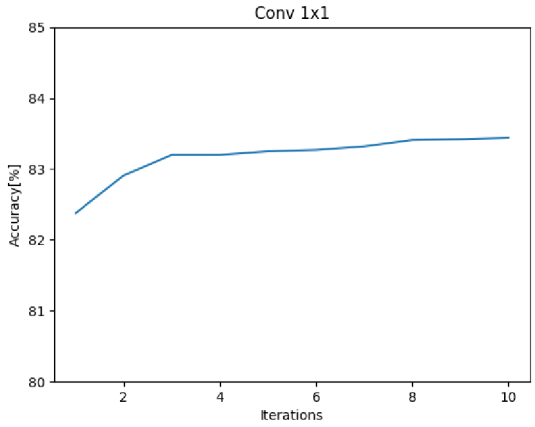


(a) Accuracy

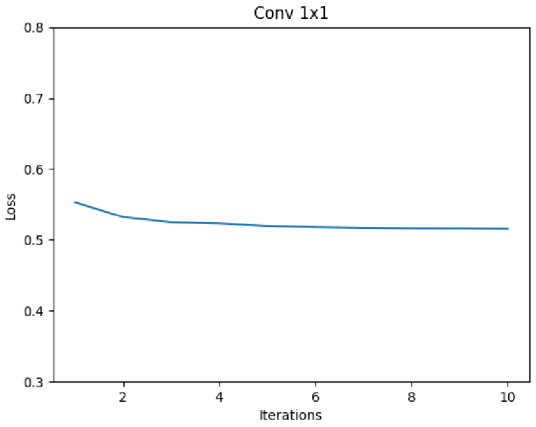


(b) Loss

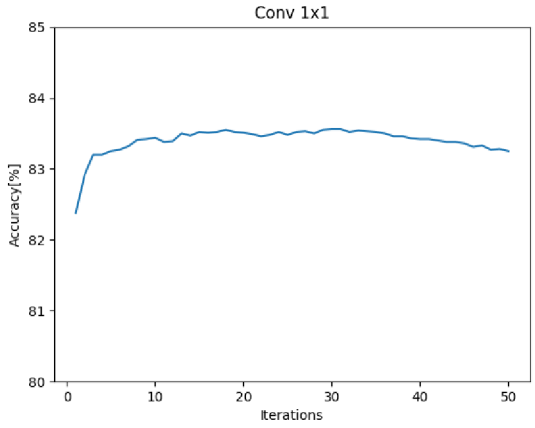
Obrázek 8.3: Architektúra s jednou konvolučnou vrstvou obsahujúcou tri filtre. Trénované s počtom iterácií 10.



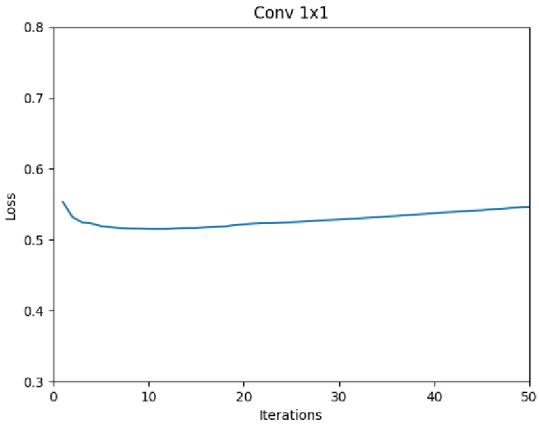
(a) Accuracy



(b) Loss



(c) Accuracy

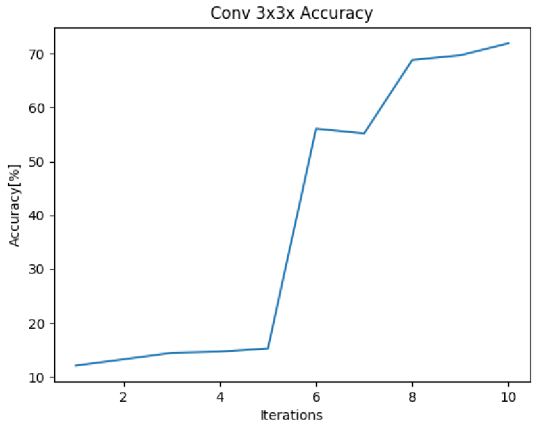


(d) Loss

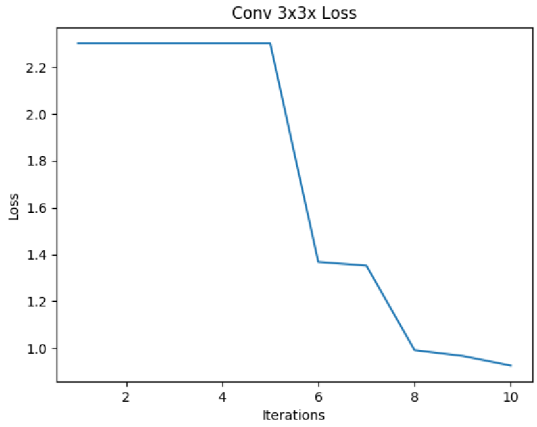
Obrázek 8.4: Architektúra s jednou konvolučnou vrstvou obsahujúcou jeden filter. Trénované s počtom iterácií 10, v spodnom riadku s počtom 15.

V prípade neuronovej siete obsahujúcou len jeden konvolučný filter bolo experimentálne vykonaných až 50 iterácií, avšak presnosť jej klasifikácie sa od začiatočných fáz klasifikácie takmer nemenila 8.4.

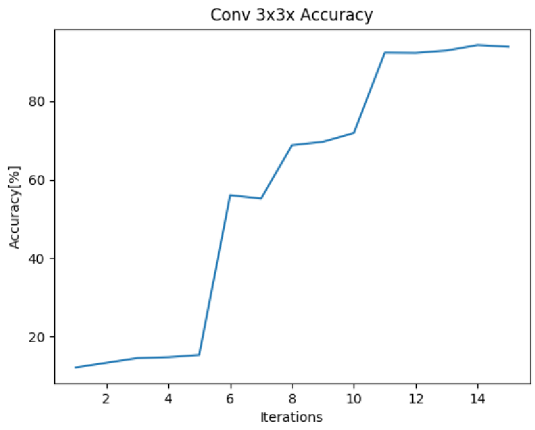
Neuronová sieť skladajúca sa z troch konvlučných vrstiev (každá po tri filtre) a z troch pooling vrstiev (označená Conv 3x3) dosiahla po pätnástich iteráciách, v porovnaní s výsledkom po desiatich iteráciách, značné zlepšenie presnosti klasifikácie 8.5. Podľa výsledkových tabuliek vidíme, že jej presnosť stúpla z 71.9% 8.1 na 94.3% 8.2, čiže v tomto prípade sa zvýšenie počtu iterácií vyplatilo.



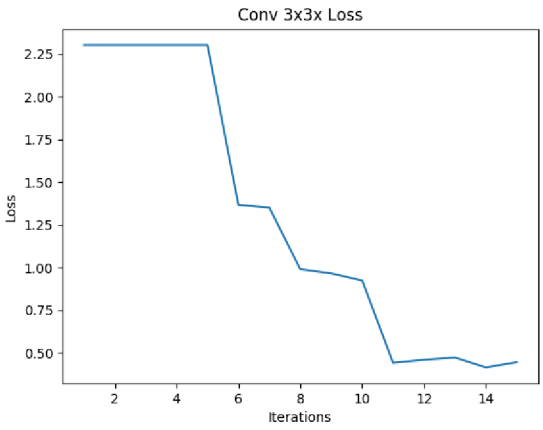
(a) Accuracy



(b) Loss



(c) Accuracy



(d) Loss

Obrázek 8.5: Architektúra s troma konvolučnými vrstvami obsahujúcimi tri filtre. Trénované s počtom iterácií 10, v spodnom riadku s počtom 15.

Architektúra	Accuracy	Loss	Počet iterácií
Conv3x3x 8.6	94.30%	0.41	15
Conv1x1 8.2	83.56%	0.52	50
Conv1x3 8.3	56.57%	1.32	15

Tabulka 8.2: Výsledky po viac ako desiatich iteráciách, použitý subset trénovacích dát

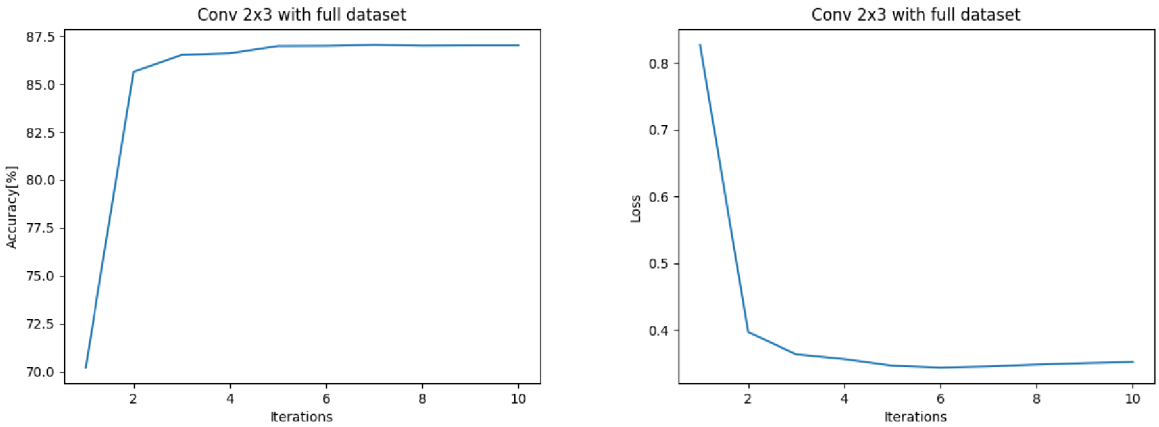
8.3.2 Trénovanie s kompletným datasetom

Cieľom trénovania na subsete trénovacích dát bolo získať približnú predstavu o výsledkoch jednotlivých architektúr. Najlepšie výsledky po desiatich iteráciách dosahovala architektúra s dvoma konvolučnými vrstvami, každá s troma filtrami a po nej najmenšia sieť obsahujúca len jeden filter v jednej konvolučnej vrstve.

Pri trénovaní neurónovej siete nie je cieľom dosiahnuť najlepšiu úspešnosť na malom datasete, ale na celých testovacích datách, ktoré sú k dispozícii. Predpokladáme, že čím väčší dataset je pre trénovanie použitý, tým lepšie bude sieť generalizovať po skončení učenia na nových dátach. Zároveň musí mať ale neurónová sieť dostatočnú kapacitu pre zachytenie väčšieho množstva informácií, ktoré prináša trojnásobne väčší dataset pre učenie. V tabuľke s výsledkami 8.3 vidieť, že výsledky oboch sietí sa v porovnaní s menším datasetom 8.1 zhoršili. Dvojvrstvová neuronová sieť ($Conv2x3$) bola schopná sa naučiť po desiatich iteráciciách na presnosť klasifikácie 87.03%, teda dosiahla po zhodnom počte iterácií o približne 7.77% horšiu presnosť pri klasifikácii na testovacom datasete (rozdiel v presnosti je 7.34%). Sieť s jednou konvolučnou vrstvou a jedným filtrom ($Conv1x1$) dosiahla úspešnosť klasifikácie 48%, čo znamená zhoršenie až o 42.17% (rozdiel v presnosti je 35.19%). Toto je zrejme spôsobené problémom kapacity sietí a výraznejšie sa prejavil na menšej sieti, čo bolo však očakávané vzhľadom na veľmi malú exprimentálnu architektúru.

Architektúra	Accuracy	Loss
Conv2x3 8.4	87.03%	0.34
Conv1x1 8.2	48.25%	1.48

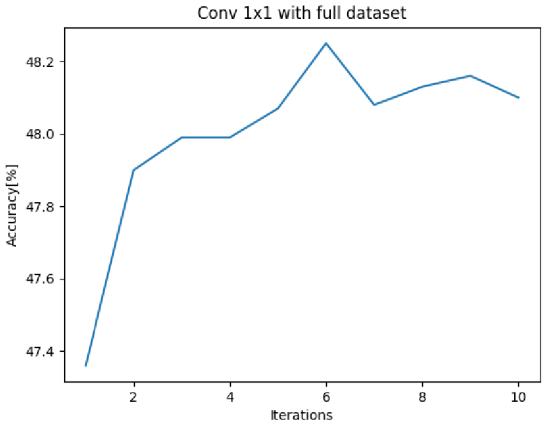
Tabulka 8.3: Výsledky po 10 iteráciách, použitý kompletný dataset



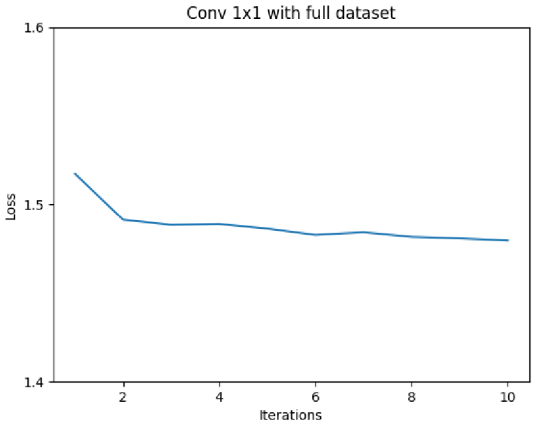
(a) Accuracy

(b) Loss

Obrázek 8.6: Architektúra s dvoma konvolučnými vrstvami obsahujúcimi tri filtre. Trénované s počtom iterácií 10 na kompletno datasete.



(a) Accuracy



(b) Loss

Obrázek 8.7: Architektúra s jednou konvolučnou vrstvou obsahujúcou jeden filter. Trénované s počtom iterácií 10 na kompletnom datasete.

8.3.3 Zhodnotenie experimentov

Cieľom trénovania neurónovej siete pomocou vlastnej implementácie bolo overiť jej funkčnosť. Dosiahnuté výsledky neuronových sietí ukazujú, že sieť je schopná sa učiť pri trénovaní klasifikačnej úlohy na datasete MNIST. Hoci boli zvolené experimentálne architektúry veľmi malé, najlepší dosiahnutý výsledok na subsete trénovacích dát (20 000 trénovacích vzoriek) bol 94.37% (na kompletnom datasete 87.03%). V porovnaní s uvedenými najlepšími výsledkami na stránke datasetu [5], ktoré udávajú chybu pre konvolučné neurónové siete maximálne 1.7% je to značne horší výsledok, avšak treba vziať do úvahy, že najjednoduchšia konvolučná vrstva na stránke obsahuje celkom 17 filtrov (4 v prvej, 12 v druhej a 1 v poslednej konvolučnej vrstve). Môžme teda konštatovať, že pre riešenie úlohy v praxi by sme zvolili komplexnejšie architektúry neuronových sietí.

Kapitola 9

Záver

Práca uviedla čitateľa do problematiky strojového učenia a hlbokých neuronových sietí, vysvetlila základné pojmy a koncepty. Bližšie sa zamerala na konvolučné neuronové siete, pričom sa diskutovali možné voľby funkcií, algoritmov a hyperparametrov pre tento druh sietí. Práca predstavila návrh programového riešenia demonštračnej úlohy a jeho implementáciu, vrátane knižnice implemnetujúcej neurónovú sieť. Grafické užívateľské rozhranie demonštruje výpočet jednotlivých vrstiev a je ním možné krokovať tok trénovacích dát sieťou, pričom užívateľ môže zadávať vlastné vstupy do siete.

Funkčnosť vlastnej implementácie neurónovej siete a algoritmu učenia overila natrénovaním modelov pre riešenie jednoduchej klasifikačnej úlohy na datasete MNIST. Boli vykonané experimenty s architektúrami konvolučnej neurónovej siete a ich výsledky porovnané a diskutované, pričom za hlavné kritérium sa považovala presnosť klasifikácie na testovacích dátach. Súčasťou riešenia je aj program pre názornú klasifikáciu pomocou ľubovoľného z uložených modelov, ktorý slúži na názorné overenie výsledkov presnosti klasifikácie jednotlivých modelov.

Možným pokračovaním tejto práce by bolo rozšíriť grafické užívateľské rozhranie pre detailnejšiu demonštráciu a pridať vlastné nastavenia hyperparametrov učenia. Rozšírením experimentov by bolo natrénovanie neurónovej siete pre iný typ úlohy a použiť k tomu robustnejšie architektúry neurónovej siete.

Literatura

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

[2] Ioffe, S.; Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, JMLR.org, 2015, s. 448–456.
URL http://dl.acm.org/citation.cfm?id=3045118.3045167

[3] Karpathy, A.: CS231n: Convolutional Neural Networks for Visual Recognition. 2017.
URL http://cs231n.stanford.edu/

[4] Le, Q. V.; Brain, G.; Inc, G.: A Tutorial on Deep Learning Part 2: Autoencoders, Convolutional Neural Networks and Recurrent Neural Networks. 2015.

[5] LeCun, Y.; Cortes, C.: MNIST handwritten digit database. 2010.
URL http://yann.lecun.com/exdb/mnist/

[6] LeCun, Y.; Jackel, L.; Bottou, L.; aj.: Comparison of Learning Algorithms for Handwritten Digit Recognition. In *INTERNATIONAL CONFERENCE ON ARTIFICIAL NEURAL NETWORKS*, 1995, s. 53–60.

[7] Mehrotra, K.; Mohan, C. K.; Ranka, S.: *Elements of Artificial Neural Networks*. Cambridge, MA, USA: MIT Press, 1997, ISBN 0-262-13328-8.

[8] Nielsen, M. A.: Neural Networks and Deep Learning. 2015.
URL http://neuralnetworksanddeeplearning.com/

[9] Srivastava, N.; Hinton, G.; Krizhevsky, A.; aj.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, ročník 15, č. 1, Leden 2014: s. 1929–1958, ISSN 1532-4435.
URL http://dl.acm.org/citation.cfm?id=2627435.2670313