FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

KATEDRA
KYBERNETIKY

Implementace a trénování neuronové sítě

Semestrální práce NSES

Západočeská Univerzita v Plzni
Katedra Kybernetiky

Tomáš LEBEDA
19. listopadu 2023

1 Zadání

1. Natrénujte:

   (a) jednovrstvou neuronovou síť

   (b) dvouvrstvou neuronovou síť

   Pro klasifikaci předmětů popsaných dvěma příznaky do 5 tříd. Trénovací množiny jsou v souborech `tren_data1.txt` a `tren_data.txt`. Každou síť natrénujte pro každou trénovací množinu zvlášť.

   Každá řádka souborů s daty odpovídá jedné trénovací dvojici. Na každém řádku jsou tři položky, které jsou od sebe odděleny mezerou. První dvě položky reprezentují vstup sítě, třetí položka je informace o zařazení klasifikovaného předmětu do konkrétní třídy.

2. Graficky znázorněte:

   - průběh chyby v závislosti na trénovacích cyklech
   - oblast ve vstupním prostoru, jak výsledná síť klasifikuje vstupní body

3. Funkční program osobně předveďte vyučujícímu.

Upozornění: Vyučující může v bodě 3 požadovat, aby student změnil trénovací množinu, strukturu sítě, parametry trénování apod. Dále může po studentovi požadovat vysvětlení, proč se síť v daném případě chová tak, jak se chová.

2 Řešení

Jako první bylo potřeba rozhodnout jakou podobu bude řešení mít a v jakém jazyce či prostředí bude implementováno. Po důkladném posouzení problému bylo rozhodnuto, že implementace bude mít podobu CLI aplikace psané v jazyce Rust s volitelnými vizualizacemi v reálném čase pomocí softwaru `Rerun.io`.

Navíc bylo řešení vytvořeno univerzálně tak, aby bylo možné pomocí argumentů příkazové řádky specifikovat počet a velikosti vrstev, jejich aktivační funkce a další parametry jako konstantu učení, velikost dávky, momentum, …

2.1 Běh programu

Celý běh programu lze shrnout v následujících krocích:

Inicializace:

V tomto kroku jsou načtené a aplikované vstupní parametry z příkazové řádky, případně je vypsáno kde nastal problém při validaci vstupní konfigurace.

Načtení trénovacích dat:

V tomto kroku jsou z poskytnuté cesty na disku načtena trénovací data do paměti. Očekávaný formát je popsaný v zadání s následujícími doplňky:

- Data mohou obsahovat více vstupních příznaků.
- Data mohou obsahovat libovolný počet klasifikačních tříd.
- Počet tříd i příznaků lze specifikovat ručně, nebo je odhadne automaticky z dat.
- Třídy je možné značit libovolnými přirozenými čísly, ne nutně 1, 2, 3, …

Vytvoření modelu neuronové sítě:

V tomto kroku je vytvořen samotný model neuronové sítě podle konfigurace:

- Počet vstupů je odvozený automaticky z počtu vstupních příznaků.
- Počet výstupů (= velikost poslední vrstvy) je odvozený automaticky z počtu tříd.
- Aktivační funkce výstupní vrstvy je možné specifikovat pomocí argumentu příkazové řádky `--final-layer-af` ve formátu `S:P`, viz následující bod a Tabulky 1 2.
- Skryté vrstvy je možné specifikovat pomocí argumentu příkazové řádky `--hidden-layers`. Očekávaná hodnota tohoto argumentu je řetězec skládající se z částí ve formátu `N:S:P` nebo `N:S` oddělených čárkami (bez mezer). Význam složek `N`, `S`, `P` je v Tabulce 1.

Zkratka	Význam	Datový typ	Příklad
`N`	velikost vrstvy	přirozené číslo	„9“, „12“
`S`	typ aktivační funkce	řetězec, viz Tabulka 2	„Linear“, „bibi“, „r“
`P`	parametr aktivační funkce λ	reálné číslo	„0.5“, „12.36“

Tabulka 1: Význam složek `N:S:P`

Aktivační funkce	Předpis	Možné způsoby zadání (`S`)

Binární bipolární	$f(x) = \operatorname{sgn}(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases}$	`BinaryBipolar`
		`bibi`
		`bb`

Binární unipolární	$f(x) = \begin{cases} +1 & x \geq 0 \\ 0 & x < 0 \end{cases}$	`BinaryUnipolar`
		`biun`
		`bu`

Sigmoida bipolární	$f(x) = \frac{2}{1 + \exp(-\lambda \cdot x)} - 1$	`SigmoidBipolar`
		`sibi`
		`sb`

Sigmoida unipolární	$f(x) = \frac{1}{1 + \exp(-\lambda \cdot x)}$	`SigmoidUnipolar`
		`siun`
		`su`

p-ReLU	$f(x) = \begin{cases} \lambda \cdot x & x \geq 0 \\ 0 & x < 0 \end{cases}$	`ReLU`
		`relu`
		`r`

Lineární	$f(x) = \lambda \cdot x$	`Linear`
		`lin`
		`l`

Tabulka 2: Aktivační funkce použitelné v programu

Například argument ve tvaru „`--hidden-layers 5:lin:0.8,7:biun,4:relu:0.9`“ vytvoří neuronovou síť se čtyřmi vrstvami:

  - První (skrytá) vrstva bude mít 5 neuronů s lineární aktivační funkcí s parametrem $\lambda = 0.8$.
  - Druhá (skrytá) vrstva bude mít 7 neuronů s binární unipolární aktivační funkcí.
  - Třetí (skrytá) vrstva bude mít 4 neurony s p-ReLU aktivační funkcí s parametrem $\lambda = 0.9$.
  - Výstupní vrstva bude mít počet neuronů daný počtem tříd.

- Velikost trénovacích dávek lze specifikovat přirozeným číslem pomocí argumentu `--batch-size`. Výchozí hodnota je plná velikosti trénovací množiny.
- Konstanta učení lze specifikovat reálným číslem pomocí argumentu `--learning-rate`.
- Momentová konstanta lze specifikovat reálným číslem pomocí argumentu `--momentum`.

Trénovací cyklus

Samotný trénovací cyklus lze popsat v následujících bodech:

1. Inicializace pomocných proměnných, mapování mezi třídou a její one-hot-vector reprezentací.
2. Náhodné zamíchání trénovacích dat a jejich rozdělení na dávky.
3. Spočtení gradientu pomocí algoritmu zpětné propagace a celkové ztráty dané dávky. Gradient je počítán pro každý trénovací bod samostatně a nakonec je udělán průměr přes celou dávku (pro lepší robustnost).
4. Změna parametrů sítě podle právě spočteného gradientu momentovou metodou.
5. Kontrola přesnosti sítě (na všech trénovacích datech). Pokud je přesnost dostačující (dle konfigurace argumentem `--target-acc`), tak trénování končí (úspěšně).

   V opačném případě kontrola, zda počítadlo epoch dosáhlo maximální hodnoty (dle konfigurace argumentem `--max-epochs`). Pokud ano, trénování také končí (neúspěšně).

   Jinak kontrola další dávky: Pokud existuje další dávka v současné epoše, pokračování do bodu 3 s další dávkou. Pokud neexistuje, inkrementace počítadla epoch a pokračování do bodu 2.

Pro implementační detaily viz přiložený zdrojový kód.

Poznámka: současná implementace nerozděluje vstupní data na trénovací a evaluační množinu, což v praxi vede k nežádoucímu přetrénování sítě, nicméně pro účely této práce to nepředstavuje problém.

2.2 Výsledky a vliv meta-parametrů

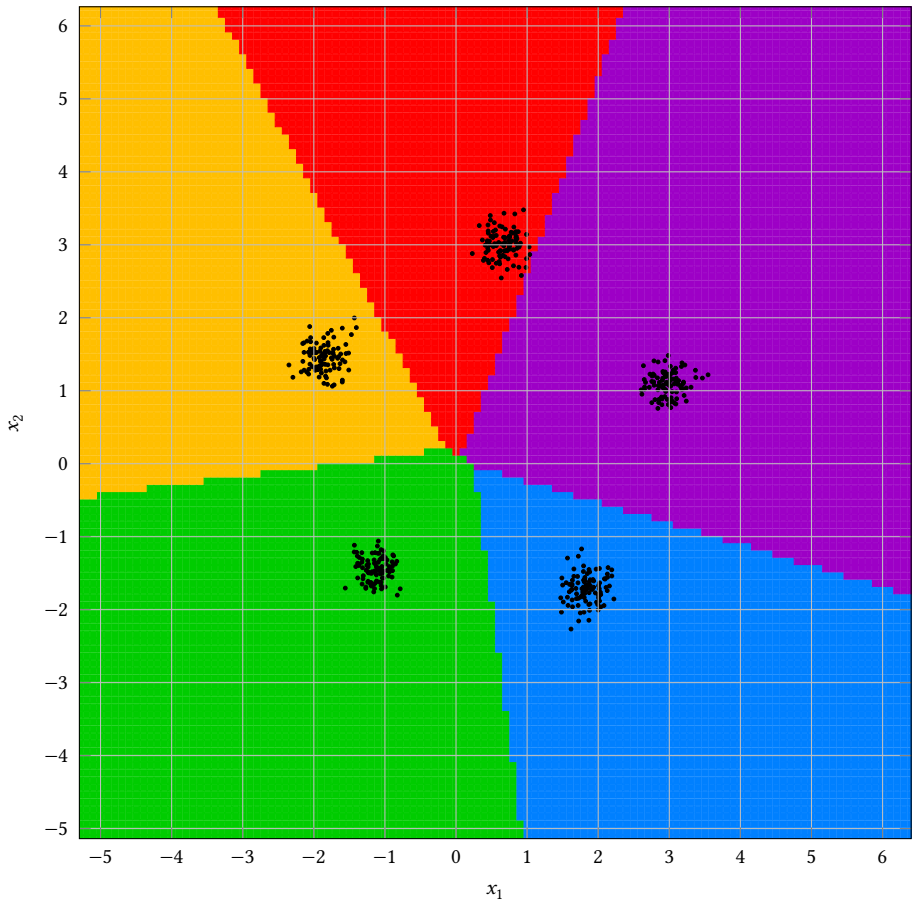
V následující části budou zobrazené a diskutované vlivy jednotlivých meta-parametrů na průběh a výsledek trénování.

2.2.1 Výsledné rozdělení vstupního prostoru

Na Obrázku 1 je zobrazeno finální rozdělení vstupního prostoru včetně znázorněných trénovacích dat. Jak je možné z této vizualizace pozorovat, vstupní rovina je rozdělena přímkami na pět oblastí, což odpovídá charakteristice jednovrstvé neuronové sítě.

Dále lze vidět, že oblasti nejsou centrované kolem shluků trénovacích dat, což je způsobeno velmi dobrou lineární separabilitou dat a ukončením trénování sítě okamžitě ve chvíli, kdy jsou všechny body správně klasifikované.

V tomto případě byla úspěšnost 100%, proto zde nejsou body rozlišené barevně, patří vždy do třídy, v jejíž oblasti se nacházejí.

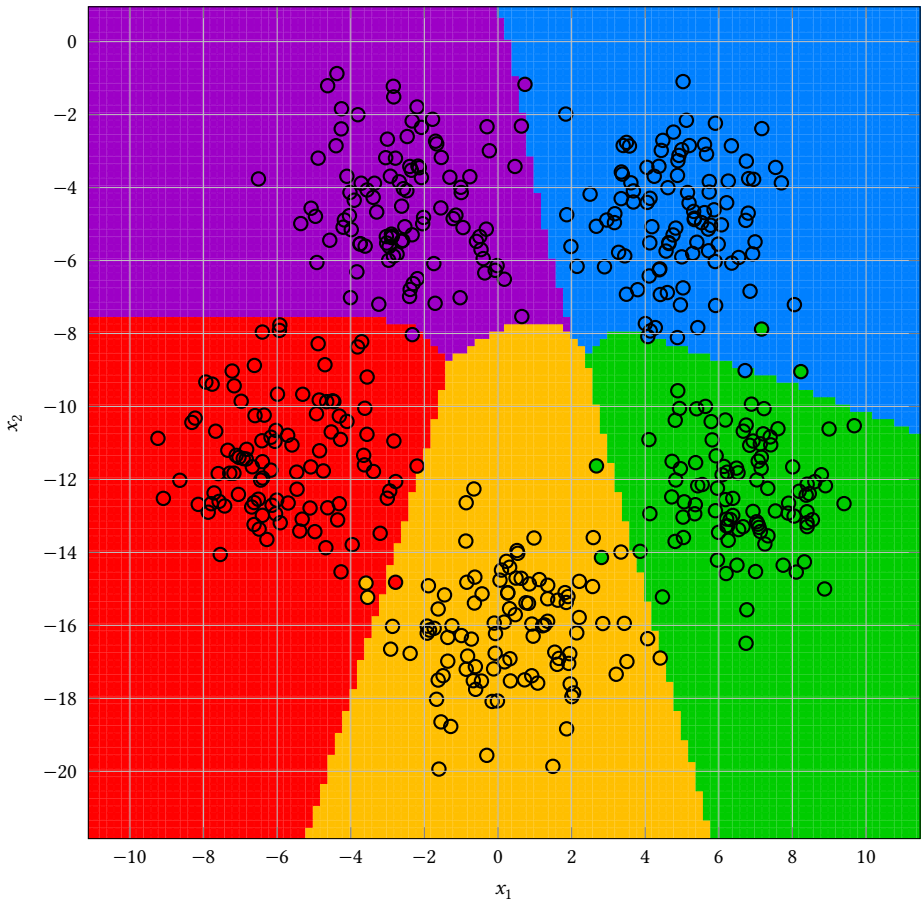


Obrázek 1: Výsledné rozdělení prostoru jednovrstvou sítí

Na Obrázku 2 je pak zobrazeno finální rozdělení vstupního prostoru druhé sady dat, která již nebyla dobře lineární separovatelná. Neuronová síť trénovaná na tento případ měla dvě vrstvy, což se projevilo ve tvaru oblastí, které nápadně připomínají polynomy druhého řádu - paraboly.

Cílová úspěšnost byla nastavena na 98% a jak je možné z vizualizace vidět, některé body byly klasifikované chybně (barva bodu odpovídá jeho správné klasifikaci, barva pozadí odpovídá jeho predikované klasifikaci).

Náhodně zvolené počáteční hodnoty vah a prahů způsobí pro opakované běhy mírně rozdílné výstupy, nicméně výše popsaná pozorování jsou konzistentní pro opakované běhy s malou mírou variability.



Obrázek 2: Výsledné rozdělení prostoru dvouvrstvou sítí

2.2.2 Průběh ztráty a přesnosti

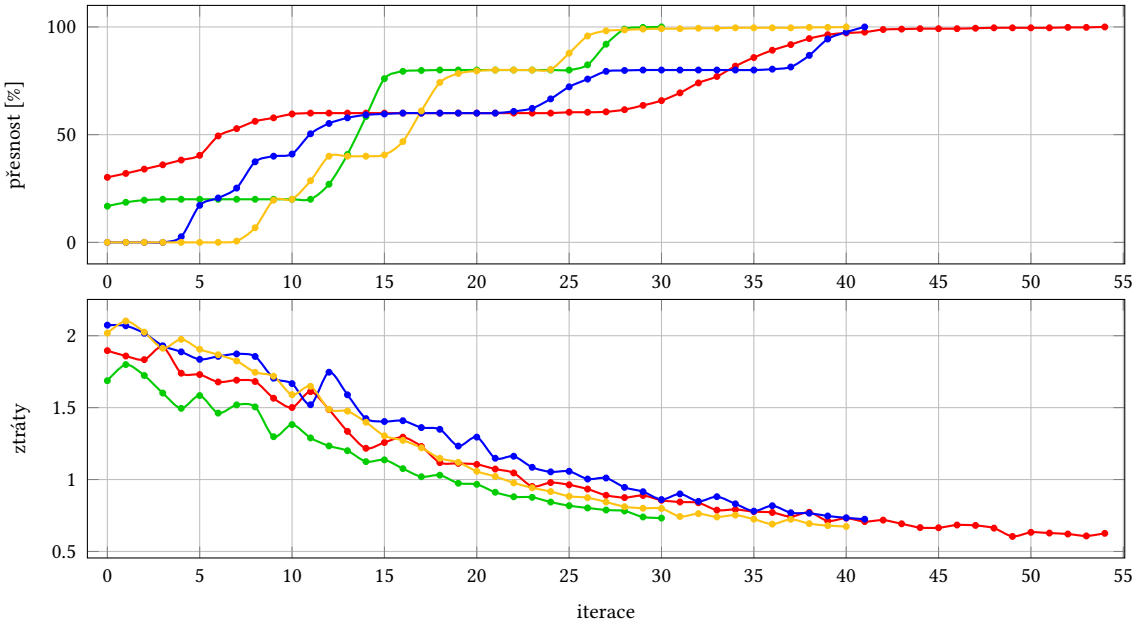
Na Obrázku 3 jsou zobrazené průběhy přesností 4 různých běhů jednovrstvé neuronové sítě se stejnými parametry (viz Tabulka 3). Z grafu je zřejmé, že počáteční náhodné nastavení vah a prahů

Parametr sítě	**Hodnota**
Aktivační funkce výstupní vrstvy	sigmoida unipolární, $\lambda = 0.6$
Velikost trénovací dávky	100
Momentová konstanta	0.8
Konstanta učení	0.4
Skryté vrstvy	žádné

Tabulka 3: Nastavení sítě z Obrázku 3

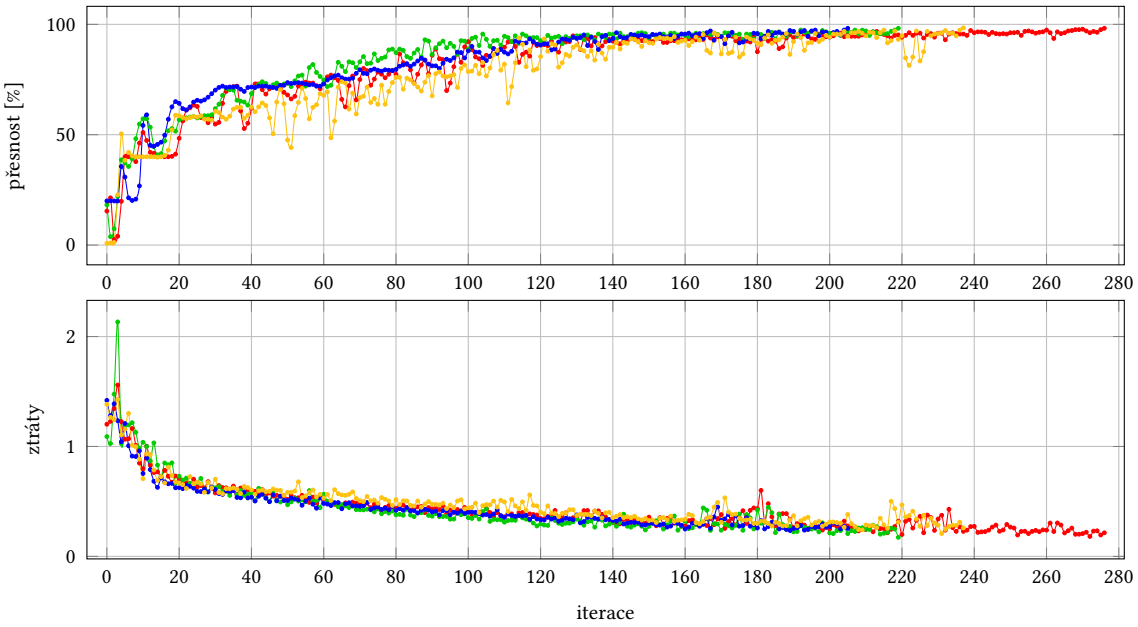
výrazně ovlivní počáteční přesnost sítě, nicméně s dostatečným počtem iterací tato síť konverguje spolehlivě ke 100% přesnosti.

„Schodovitá“ charakteristika průběhu přesností je pravděpodobně způsobena rozmístěním trénovacích dat, kdy k prudkému vzrůstu přesnosti dojde když se nějaká z rozdělujících přímek dostane do oblasti nějakého ze shluků.



Obrázek 3: Přesnost a ztráty 4 různých běhů jednovrstvé sítě

Na Obrázku 4 jsou pak znázorněné průběhy přesností a ztrát pro dvouvrstvou neuronovou síť s konfigurací v Tabulce 4. Na tomto grafu je patrné, že pro dosažení přesnosti 98% bylo zapotřebí mnohem více iteračních cyklů. Dále je možné z vizualizace sledovat, že průběhy jednotlivých trénování mají velmi podobný charakter a vliv rozdílných počátečních není tolik zřetelný jako u jednodušší sítě s lépe separovanými daty.



Obrázek 4: Přesnost a ztráty 4 různých běhů dvouvrstvé sítě

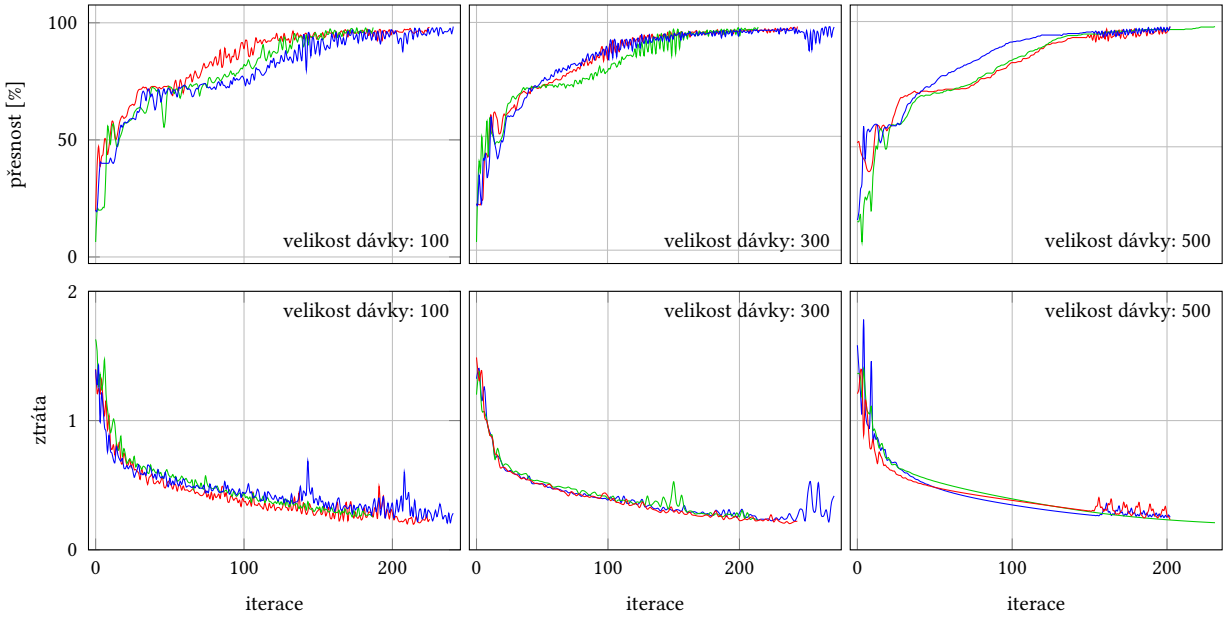
Parametr sítě	Hodnota
Aktivační funkce výstupní vrstvy	sigmoida unipolární, $\lambda = 0.6$
Velikost trénovací dávky	100
Momentová konstanta	0.8
Konstanta učení	0.4
Skryté vrstvy	10 neuronů, lineární, $\lambda = 0.6$

Tabulka 4: Nastavení sítě z Obrázku 4

2.2.3 Vliv velikosti trénovací dávky

Na Obrázku 5 je zobrazen vliv velikosti trénovací dávky na průběh ztráty a přesnosti dvouvrstvé neuronové sítě. Jak je možné vidět, tak velikost trénovací dávky souvisí s hladkostí (počtem lokálních extrémů).

Čím větší je trénovací dávka, tím více reprezentativní bude výsledný vektor gradientu a tím pádem bude úprava vah a prahů „přesnější“. Nevýhodou velkých trénovacích dávek je nutnost velkého počtu trénovacích a vyšší výpočetní náročnost.

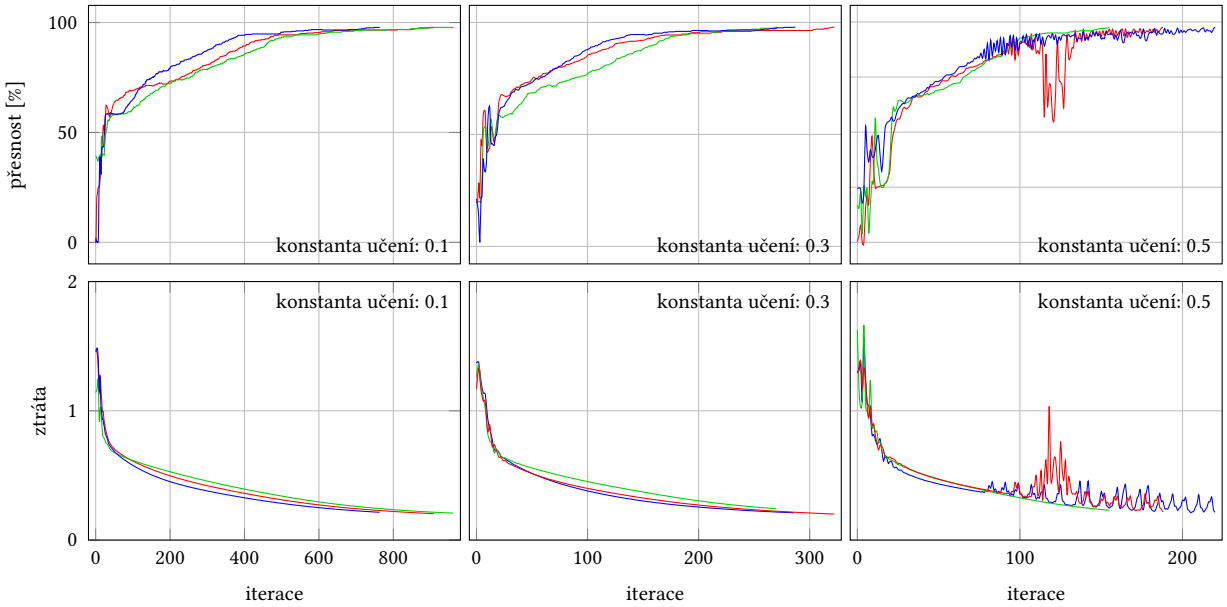


Obrázek 5: Vliv velikosti dávky na průběh ztráty a přesnosti

2.2.4 Vliv konstanty učení

Na Obrázku 6 je zobrazen vliv konstanty učení na průběh trénování dvouvrstvé neuronové sítě. Jak je možné z grafů vidět, s rostoucí hodnotou konstanty učení prudce klesá počet iterací potřebných k dosažení požadované přesnosti.

Cenou za rychlejší trénování je však vyšší pravděpodobnost, že změna vah a prahů bude příliš agresivní a „přeskočí“ optimální bod. Toto chování je vidět ke konci trénovacích cyklů s konstantou učení 0.5, kde již jsou potřeba pouze malé změny prahů a vah.



Obrázek 6: Vliv konstanty učení na průběh ztráty a přesnosti

2.2.5 Vliv dalších faktorů sítě

Dále bylo provedeno několik experimentů s aktivačními funkcemi, jejich parametry, a počtem neuronů i počtem vrstev ve skrytých vrstvách. Obecně bylo problematické odhadovat vliv jednotlivých faktorů, zejména kvůli problémům s numerickou stabilitou, která nebyla v této implementaci optimalizována.

Celkově lepší a stabilnější výsledky poskytovaly omezené aktivační funkce (sigmoidy) s nižším parametrem (v řádu desetin), než skokové funkce nebo neomezené (lineární a ReLU).

Některé konfigurace trénování způsobily konvergenci výrazně rychlejší či stabilnější, pro jiné zase byla nižší citlivost na změnu počátečního stavu.

Pro podrobnější analýzu vlivu jednotlivých aktivačních funkcí a dalších faktorů sítě by bylo potřeba zlepšit numerickou stabilitu implementace a provést rozsáhlejší sadu experimentů, což je nad rámec této práce.