Rozpoznávání ručně psaných číslic pomocí neuronových sítí

William Tatarko

16. září 2017

Abstrakt

V úvodu popíšeme základní princip neuronových sítí. Dále se budeme zabývat aplikací takových sítí, konkrétně rozpoznáváním ručně psaných číslic. Detailně rozebereme matematický model, stojící za neuronovými sítěmi. Nakonec stručně zmíníme možnosti implementace, efektivitu a možnosti vylepšení.

1 Neuronové sítě obecně

1.1 Základní princip

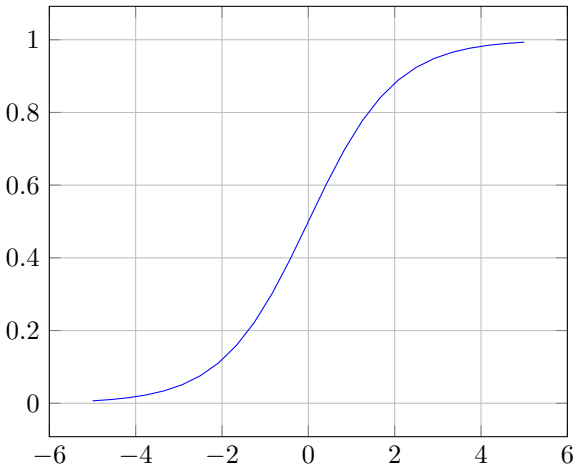
Umělé neuronové sítě jsou oblastí umělé inteligence. Neuronové sítě obsahují mnoho parametrů, které určují, jaký výstup bude při daném vstupu. Při tréninku jsou sítě schopny tyto parametry samy měnit tak, aby byl výstup dle nějakého daného pravidla co nejlepší. Cílem je, aby měly parametry po konci tréninku takové hodnody, aby výstup sítě byl co nejlepší i pro vstupy, se kterými neuronová síť netrénovala.

V našem případě, tj. klasifikace ručně psaných číslic, bude vstupem obrázek číslice (přesněji dále) a výstupem jaká číslice je na obrázku.

1.2 Neuron

Neuron je základním prvkem neuronové sítě. Každý neuron má $n \in \mathbb{N}$ vstupů $x_i \in [0,1]$, kde $i \in \{1,\dots,n\}$, a jeden výstup $y \in [0,1]$. Krom toho má také n tzv. synaptických vah w_i a práh b. Každý vstup x_i si můžeme představit jako jistý podnět, který má důležitost w_i. Práh pak intuitivně představuje, jak moc je neuron ochotný na všechny podněty zareagovat, tj. vyslat na výstup y signál. Právě w_i a b jsou parametry, o kterých jsme mluvili v úvodu. Výstup neuronu je dán vztahem

$$y = \sigma\Big(\sum_{i=1}^{n} x_i w_i + b\Big), \tag{1}$$



Obrázek 1: Logistická funkce σ

kde σ je tzv. logistická funkce daná předpisem

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{2}$$

Graf funkce σ je znázorněn na Obrázku 1. Pro přehlednost můžeme y přepsat pomocí skalárního součinu

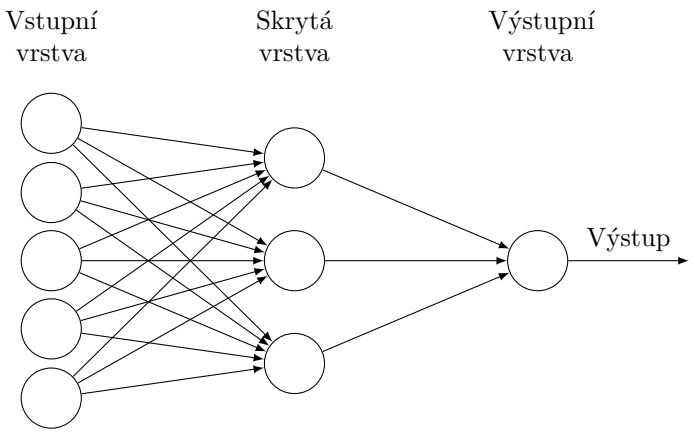
$$y = \sigma(\mathbf{x} \cdot \mathbf{w} + b), \tag{3}$$

kde $\mathbf{x}$ je vektor obsahující v i-té složce x_i a podobně $\mathbf{w}$ je vektor, který má v i-té složce w_i.

Výstup neuronu je (s výjimkou těch posledních) použitý jako vstup dalších neuronů. Pokud bychom tedy z definice y vynechali logistickou funkci, tak by hrozilo, že by vstup do dalšího neuronu nebyl z intervalu $[0, 1]$. Místo funkce σ lze použít i jiné funkce, např. hyperbolický tangens.

1.3 Neuronová síť

Neuronová síť je složena z mnoha neuronů rozdělených do vrstev. Každý neuron ve vrstvě B má jako vstupy výstupy všech neuronů z vrstvy A. Například na Obrázku 2 jsou 3 vrstvy. První vrstva (nakreslená nalevo) se nazývá vstupní a obsahuje v tomto případě 5 neuronů. Poznamenejme, že vstupní vrstva neobahuje skutečné neurony popsané v podkapitole 1.2, neboť nemá žádný vstup, ale přesto má výstup. Označovat jednotlivé vstupy za neurony je spíše konvence.



Obrázek 2: Neuronová síť

Poslední vrstvě se říká výstupní a v našem případě obsahuje pouze jeden neuron. Všechny ostatní vrstvy se nazývají skryté. Na obrázku je tedy pouze jedna skrytá vrstva (obsahující 3 neurony).

1.4 Loss function

Základní myšlenkou neuronových sítí je nalezení vhodných parametrů, tj. synaptických vah w_i a prahu b pro každý neuron. Potřebujeme tedy nějakou funkci, která bude vyhodnocovat jak dobrý je výstup při různých parametrech a daném vstupu. Tuto funkci budeme nazývat loss function (případně cost function) a definujeme ji předpisem

$$L(W, B) = \frac{1}{2n} \sum_{\mathbf{x}} \|f(\mathbf{x}) - y(\mathbf{x})\|^2, \tag{4}$$

kde W je množina všech $\mathbf{w}$ (pro každý neuron jedna), B je množina všech b (pro každý neuron jeden), n je počet vstupů, $\mathbf{x}$ je vstup (do vstupní vrstvy), $f(\mathbf{x})$ je výstup (výstupní vrstvy) a $y(\mathbf{x})$ je správný výstup, kterého se snažíme dosáhnout. V případě rozpoznávání číslic bude $y(\mathbf{x})$ značit, která číslice je skutečně na obrázku a $f(\mathbf{x})$ bude symbolizovat, jaká číslice si neuronová síť "myslí", že je na obrázku.

Intuitivně je zřejmé, že čím bude $L(W, B)$ menší, tím lépe bude naše neuronová síť fungovat. Chceme tedy najít minimum funkce $L(W, B)$.

Loss function lze definovat i jinými způsoby (např. tzv. cross entropy), avšak všechny musí splňovat dvě podmínky [1]. Zaprvé musí být funkcí výstupu neuronové sítě a zadruhé musí být aritmetickým průměrem ohodnocení jednotlivých vstupů. Bez těchto předpokladů bychom nemohli aplikovat metodu backpropagation (viz dále). Definice (4) splňuje obě podmínky.

Důležité je si uvědomit, že pokud daná neuronová síť má alespoň jednu skrytou vrstvu, tak loss function L není konvexní. Předpokládejme, že jsme "prohodili" vektory $\mathbf{w}$ a prahy b neuronů n_1 a n_2 ve stejné vrstvě a následně také složky vektorů $\mathbf{w}$ ve všech neuronech další vrstvy tak, aby výstup neuronu n_1 byl nyní násoben synaptickými vahami, kterými jsme původně násobili výstup neuronu n_2 a obráceně. Pak máme jinou posloupnost neznámých, která ovšem dává stejný výstup $f(\mathbf{x})$ pro každý vstup $\mathbf{x}$ (vstupní vrstvy). Proto bude mít i $L(W, B)$ stejnou hodnotu. Ukázali jsme tedy, že pokud má funkce L v nějakém bodě lokální minimum, existuje ještě jiný bod, ve kterém také nabývá (stejného) lokálního minima. Proto funkce L nemůže být konvexní.

1.5 Gradient descent

Množina W obsahuje mnoho $\mathbf{w}$ a každé z nich má mnoho složek. Podobně je na tom B. Proto by bylo velmi obtížné hledat minimum klasickou metodou analýzy. Místo toho použijeme metodu zvanou gradient descent, která hledá lokální minimum funkce více proměnných metodou postupných aproximací. Za předpokladu, že funkce $f : \mathbb{R}^n \to \mathbb{R}$, kde $n \in \mathbb{N}$, má všechny parciální derivace spojité, pak pro malá Δx_i ($i \in \{1, \ldots, n\}$) platí

$$\Delta f \approx \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \Delta x_i. \tag{5}$$

Pokud označíme $\Delta \mathbf{x} = (\Delta x_1, \ldots, \Delta x_n)^T$ tak můžeme rovnici (5) přepsat jako

$$\Delta f \approx \nabla f \cdot \Delta \mathbf{x}. \tag{6}$$

Platí tedy také

$$f(\mathbf{x} + \Delta \mathbf{x}) \approx f(\mathbf{x}) + \nabla f \cdot \Delta \mathbf{x}. \tag{7}$$

Jelikož chceme minimalizovat f, chceme najít takové (dostatečně malé) $\Delta \mathbf{x}$, aby platilo $\nabla f \cdot \Delta \mathbf{x} < 0$. Protože ∇f nám říká, v jakém směru roste funkce f nejrychleji, víme, že ve směru $-\nabla f$ nejrychleji klesá. Proto budeme za směr $\Delta \mathbf{x}$ volit právě $-\nabla f$. Aby platily výše uvedené přibližné rovnosti, je potřeba zajistit, aby $\Delta \mathbf{x}$ bylo dostatečně malé. K tomu nám poslouží konstanta $\eta > 0$. Změnu $\Delta \mathbf{x}$ tedy budeme volit

$$\Delta \mathbf{x} = -\eta \nabla f. \tag{8}$$

Metoda gradient descent nám tedy říká, jak změnit parametry funkce, abychom se přiblížili k jejímu minimu. Konstanta η se běžně nazývá learning rate a hraje důležitou roli. Pokud by byla příliš malá, pak by bylo hledání minima příliš pomalé, neboť bychom si vždy přilepšili pouze o malý kus. Naopak, pokud by learning rate byl příliš velký, tak by již nemusela platit aproximace ve vztahu (5) a (7).

V našem případě chceme pomocí metody gradient descent nalézt minimum funkce L. Proto budeme dle vztahu (8) měnit parametry z množin W a B. To však vyžaduje znalost všech parciálních derivací funkce L, které vypočítáme v další kapitole.

Připomeňme, že funkce L není konvexní, a tak metodou gradient descent pravděpodobně nezískáme globální minimum, ale pouze nějaké lokální (respektive se k němu přiblížíme). To se však ukáže jako nepříliš velký problém, neboť obvykle i lokální minima dávají poměrně dobré výsledky.

1.6 Backpropagation

Metoda zvaná backpropagation slouží k nalezení parciálních derivací všech w a b pomocí zpětného průchodu neuronové sítě. Jedná se o matematicky poměrně komplikovaný postup, a tak se jím budeme zabývat až v další kapitole na konkrétním případě. Poznamenejme však, že nám umožní spočítat gradient pouze pro jeden vstup. Vzhledem k tomu, že požadujeme, aby loss function byla aritmetickým průměrem napříč všemi vstupy, tak dává smysl vypočítat celkový gradient funkce L (a následně dle (8) změnu všech parametrů) jako aritmetický průměr přes všechny vstupy.

1.7 Konvoluční neuronové sítě

1.7.1 Úvod

V klasické neuronové síti popsané v podkapitole 1.3 je každý neuron v rámci vrstvy rovnocenný, neboť mezi dvěma vrstvami je propojený každý neuron s každým. V našem případě by nám tedy nic nebránilo, abychom měli pro každý pixel vstupního obrázku vyhrazený jeden vstup. Hodnota by pak značila, jak moc je daný pixel černý, respektive bílý. Na takto relativně jednoduchý příklad (rozpoznávání ručně psaných číslic) by tato neuronová síť v jistém slova smyslu stačila. Pro komplikovanější úkoly, jako je například rozlišování obrázků psů a koček, bychom s klasickou metodou neuspěli. Je potřeba sofistikovanější neuronová síť. Tou je právě konvoluční neuronová síť. Nejenže umožňuje řešit obtížnější zádání, ale také – jak v další kapitole uvidíme – je schopná lepší klasifikace ručně psaných číslic.

Již nebudeme na pixely nahlížet jako na hromadu čísel, ale vezmeme v potaz strukturu obrázku. Pokud bychom používali pouze klasickou neuronovou síť, tak

bychom po tréninku například nemohli poznat číslici, která je sice napsaná stejně (jako ty tréninkové), ale je o kus posunutá [4]. Důvodem je to, že posunutím změní řada pixelů barvu.

1.7.2 Konvoluce

Na vstupu (vstupní vrstvy) máme 3-dimenzionální tenzor. Jeden rozměr značí vrstvy, druhý výšku a třetí šířku. V případě barevného vstupu bychom tedy měli 3 vrstvy, zatímco u černobílého jen jednu. U klasické neuronové sítě jsme měli na vstupu jen 1-dimenzoinélní tenzor, neboli vektor. Pro každou vrstvu vstupního obrázku budeme mít jeden tzv. filter (neboli kernel), což je vlastně matice synaptických vah. Tento filter bude mít menší rozměry než vstupní obrázek. To je proto, abychom ho mohli "posouvat" (vždy o stejnou hodnotu, tzv. stride) po vstupním obrázku, respektive jeho jedné vrstvě. V každé pozici filteru vynásobíme všechny hodnoty filteru s příslušnými (těmi při dané pozici "pod sebou") hodnotami vstupního obrázku. Všechny součiny sečteme a navíc přičteme ještě práh (viz později). Takto získáme pro každou pozici filteru ("na vstupním obrázku") jednu hodnotu. Ve skutečnosti však nemáme pro každou vrstvu vstupního obrázku jen jeden filter, ale rovnou několik. Z každého filteru získáme matici hodnot, a tedy celkem získáme opět 3-dimenzionální tenzor. Pro každou vrstvu výstupu máme jen jeden práh, který je sdílen všemi vrstvami vstupu. Jelikož chceme, aby každý výstup mohl být vstupem další vrstvy, tak na konci ještě aplikujeme logistickou funkci.

Nechť má vstupní tenzor X rozměry $a \times b \times b$, kde $a, b \in \mathbb{N}$. Číslo a tedy značí počet vrstev a b výšku a šířku vstupního obrázku (pro jednoduchost uvažujeme čtvercový obrázek). Nechť pro každou vrstvu je c filterů, každý má rozměr $d \times d$ (opět máme pro jednoduchost čtvercový filter), kde $d \in \mathbb{N}, d \leq b$, a nechť $F(i, j, k, l)$ značí hodnotu i-tého filteru pro vrstvu j na pozici (k, l), kde $i \in \{1, \ldots, c\}$, $j \in \{1, \ldots, a\}$ a $k, l \in \{1, \ldots, d\}$. Nechť $b(i)$ značí práh příslušný k i-té výstupní vrstvě. Nechť stride je s. Pak má výstupní tenzor Y rozměry $c \times e \times e$, kde $e = \frac{b-d}{s} + 1$. Nechť $Y(i, m, n)$ značí hodnotu na pozici (m, n) v i-té vrstvě Y, kde $m, n \in \{1, \ldots, e\}$. Pak platí

$$Y(i,m,n) = \sigma\left(\sum_{o=1}^{a}\sum_{p=1}^{d}\sum_{q=1}^{d}\Big(X\Big(o, s(m-1)+p, s(n-1)+q\Big)F(i,o,p,q)\Big)+b(i)\right). \tag{9}$$

Operace obsažená v definici (9) se obvykle nazývá korelace. Konvoluce vypadá podobně a lze ji (v tomto případě) dosáhnout stejnou definicí otočením každého filteru o 180°. Jakou operaci použijeme je ve výsledku irelevantní a korelace je zde lepší na představivost.

Podívejme se ještě, jak konvoluce vypadá na konkrétním příkladě. Předpokládejme, že na vstupu máme černobílý obrázek velikosti 12×12. Vstup má tedy pouze

Tabulka 1: Vstupní vrstva s vyznačenou polohou filteru

3	1	6	4	4	1	3	6	3	2	3	5
4	5	6	1	5	6	5	2	2	4	6	4
5	1	4	5	1	3	4	3	1	1	5	4
1	4	4	5	2	2	2	5	4	3	1	1
4	2	1	4	4	4	2	6	3	1	3	6
2	5	5	3	5	5	4	2	1	5	1	4
4	3	2	5	1	5	2	5	3	1	1	5
6	3	2	4	2	4	1	5	3	6	6	5
2	5	1	6	4	4	3	2	1	1	5	2
6	2	4	2	1	5	4	1	6	4	6	5
3	5	2	2	5	6	2	5	3	2	4	1
6	2	1	3	2	5	5	6	5	6	6	3

Tabulka 2: První filter

1	4	4	5	2
5	2	3	5	2
4	5	1	5	4
3	4	1	3	5
4	5	2	1	4

jednu vrstvu. Dále přepokládejme, že tato konvoluční vrstva obsahuje 6 filterů, každý velikosti 5×5. Ke každému filteru patří také jeden práh. Uvažujme stride této konvoluce $s = 1$. Výstup bude mít tedy 6 vrstev (neboť máme 6 filterů). Rozměr výstupní vrstvy spočítáme dle vzorečku $e = \frac{b-d}{s} + 1 = \frac{12-5}{1} + 1 = 8$. Ukážeme, jak by se počítal například prvek v 3. řádku a 7. sloupci první vrstvy výstupu. Všechny uvedené hodnoty jsou pouze ilustrační. V Tabulce 1 vidíme 12×12 hodnot, při čemž každá reprezentuje 1 pixel vstupu. Neboť chceme vypočítat hodnotu (první vrstvy) výstupu v 3. řádku a 7. sloupci, "posuneme" filter (velikosti 5×5) tak, by jeho levý horní roh byl na pozici $(3, 7)$.

V Tabulce 2, která reprezentuje filter, pak vidíme 5×5 hodnot. Předpokládejme, že práh prvního filteru má hodnotu 4.

Abychom získali hodnotu prvku v 3. řádku a 7. sloupci první vrstvy výstupu, musíme udělat součet všech součinů příslušných hodnot pixelů vstupu a filteru a na závěr ještě přičíst práh. Výpočet je tedy následující.

$$
\begin{aligned}
&4 \cdot 1 + 3 \cdot 4 + 1 \cdot 4 + 1 \cdot 5 + 5 \cdot 2 + \\
&2 \cdot 5 + 5 \cdot 2 + 4 \cdot 3 + 3 \cdot 5 + 1 \cdot 2 + \\
&2 \cdot 4 + 6 \cdot 5 + 3 \cdot 1 + 1 \cdot 5 + 3 \cdot 4 + \\
&4 \cdot 3 + 2 \cdot 4 + 1 \cdot 1 + 5 \cdot 3 + 1 \cdot 5 + \\
&2 \cdot 4 + 5 \cdot 5 + 3 \cdot 2 + 1 \cdot 1 + 1 \cdot 4 + \\
&4 = 227
\end{aligned} \tag{10}
$$

Nakonec bychom ještě měli aplikovat logistickou funkci σ. Hodnota prvku v 3. řádku a 7. sloupci první vrstvy výstupu je tedy $\sigma(227)$. Podobně bychom vypočítali hodnoty na všech pozicích každého filteru.

1.7.3 Max-pooling

Jelikož konvoluce vyžaduje poměrně velké množství operací, které se musí během tréninku mnohokrát vykonávat, je výhodné zmenšit velikost tenzorů. Obvyklou metodou je tzv. max-pooling, který spočívá v tom, že v každé vrstvě výstupu z konvoluční vrstvy se z jisté oblasti vezme pouze největší hodnota. V našem případě takovou oblastí vždy bude čtverec 2×2.

Matematicky řečeno máme na vstupu tenzor X velikosti $a \times b \times b$ (jelikož uvažujeme vždy jen čtvercové vstupy), kde $a, b \in \mathbb{N}$ a výstupem bude tenzor Y velikosti $a \times \frac{b}{2} \times \frac{b}{2}$. Požadujeme tedy, aby b bylo dělitelné dvěma. Pro hodnotu výstupu ve vrstvě $i \in \{1, \ldots, a\}$ a na pozici (j, k) pro $j, k \in \{1, \ldots, \frac{b}{2}\}$ pak platí

$$
Y(i, j, k) = max\Big(X(i, 2j - 1 + l, 2k - 1 + m);\ l, m \in \{0, 1\}\Big) \tag{11}
$$

1.7.4 Vektoralizace

Konvoluční neuronová síť má své výhody, ale výstupem je 3-dimenzionální tenzor, což se obvykle těžko interpretuje jako výsledek. Proto je běžné mít za konvoluční neuronovou sítí ještě klasickou neuronovou síť. V tomto kontextu se vrtsvám s konvolucí říká konvoluční vrtsvy, podobně vrstvy s max-poolingem se nazývají max-pooling vrstvy a všem vrstvám klasické neuronové sítě na konci se říka plně propojené vrstvy (neboť je každý neuron spojen s každým mezi sousedními vrstvami).

Jelikož výstupem konvoluční vrstvy je 3-dimenzionální tenzor a vstupem plně propojené vrstvy je vektor, musíme tenzor deformovat. Všechny hodnoty se intuitivně "seřadí za sebe". Matematicky to poněkud neformálně popíšeme funkcí V

$$
\mathbf{v} = V(T), \tag{12}
$$

kde T je 3-dimenzionální tenzor (na výstupu konvoluční vrstvy) a $\mathbf{v}$ je příslušný vektor po vektoralizaci.

1.7.5 Backpropagation

Backpropagation se konvolučními (a max-poolingovými) vrstvami samozřejmě značně komplikuje. Rozebereme ji až v další kapitole na konkrétním příkladě.

2 Rozpoznávání rukou psaných číslic

2.1 Úvod

Cílem je správně klasifikovat ručně psanou číslici. Na vstupu tedy bude černobílý obrázek velikosti 28×28. Síť bude mít 10 výstupů, pro každou číslici jeden. Ten s největší hodnotou bude vyhodnocen jako ten správný. Data získáme z databáze MNIST [3], která obsahuje 50000 obrázků číslic na trénování sítě a dále 10000 (jiných) na testování.

2.2 Architektura sítě

Průchod sítí bude vypadat přibližně takto.

0. Na vstupu bude tenzor velikosti $1 \times 28 \times 28$.
1. Následovat bude první konvoluční vrstva s 16 filtry velikosti 5×5 a stridem $s = 1$. Výstupem této vrstvy tedy bude tenzor velikosti $16 \times 24 \times 24$, neboť (dle vzorce z kapitoly 1.7.2) $\frac{28-5}{1} + 1 = 24$.
2. Dále zmenšíme množství dat max-poolingem (s faktorem 2), takže výsledkem bude tenzor velikosti $16 \times 12 \times 12$.
3. Následovat bude další konvoluce. Tentokrát bude mít 32 filtrů velikosti 5×5 a opět stride $s = 1$. Výstupem tedy bude tenzor velikosti $32 \times 8 \times 8$.
4. Znovu aplikujeme max-pooling, čímž získáme tenzor velikosti $32 \times 4 \times 4$.
5. Nyní použijeme vektoralizaci, takže deformujeme tenzor po max-poolingu na vektor velikosti $32 \cdot 4 \cdot 4 = 512$.
6. Dále bude plně propojená vrstva s 128 neurony.
7. Nakonec bude další plně propojená vrstva s 10 neurony.

Naše síť (a loss function) tedy bude mít $16 \cdot (1 \cdot 5^2 + 1) + 32 \cdot (16 \cdot 5^2 + 1) + 128 \cdot (512 + 1) + 10 \cdot (128 + 1) = 80202$ parametrů.

2.3 Backpropagation

2.3.1 Úvod

Průchod konvoluční neuronovou sítí je sice poměrně komplikovaný, ale vypočítat ze vstupu výstup je přímočaré. Obtížné je spočítat gradient loss function L. Jak již bylo naznačeno, při výpočtu parciálních derivací se postupuje od konce.

2.3.2 Značení

Horním indexem budeme značit, ke které vrstvě se daná neznámá váže. Neznámé se stejným významem budeme značit stejným písmenem (odlišené horním indexem). Případná čísla v závorkách za neznámou budeme (stejně tak jako doposud) využívat k indexování. Budeme používat tyto neznámé:

- I^0 ... vstup $(1 \times 28 \times 28)$,
- K^1 ... filtry první konvoluční vrstvy $(16 \times 1 \times 5 \times 5)$,
- $\mathbf{b^1}$ ... prahy první konvoluční vrstvy (16),
- Z^1 ... výstup první konvoluční vrstvy před aplikací funkce σ $(16 \times 24 \times 24)$,
- C^1 ... výstup první konvoluční vrstvy po aplikaci funkce σ $(16 \times 24 \times 24)$,
- S^2 ... výstup prvního max-poolingu $(16 \times 12 \times 12)$,
- K^3 ... filtry druhé konvoluční vrstvy $(32 \times 16 \times 5 \times 5)$,
- $\mathbf{b^3}$ ... prahy druhé konvoluční vrstvy (32),
- Z^1 ... výstup druhé konvoluční vrstvy před aplikací funkce σ $(32 \times 8 \times 8)$,
- C^3 ... výstup druhé konvoluční vrstvy po aplikaci funkce σ $(32 \times 8 \times 8)$,
- S^4 ... výstup druhého max-poolingu $(32 \times 4 \times 4)$,
- $\mathbf{v^5}$ ... výstup vektoralizace (512),
- W^6 ... synaptické váhy první plně propojené vrstvy (128×512),
- $\mathbf{b^6}$ ... prahy první plně propojené vrstvy (128),
- $\mathbf{z^6}$ ... výstup první plně propojené vrstvy před aplikací funkce σ (128),
- $\mathbf{f^6}$ ... výstup první plně propojené vrstvy po aplikaci funkce σ (128),
- W^7 ... synaptické váhy druhé plně propojené vrstvy (10×128),
- $\mathbf{b^7}$ ... prahy druhé plně propojené vrstvy (10),
- $\mathbf{z^7}$ ... výstup druhé plně propojené vrstvy před aplikací funkce σ (10),
- $\mathbf{f^7}$ ... výstup druhé plně propojené vrstvy po aplikaci funkce σ (10),
- $\mathbf{o^7}$ ... správny výstup (jeden z vektorů kanonické báze) (10).

2.3.3 Výpočet

Výpočty v této podkapitole byly inspirovány výpočty uvedenými v [2].

Nejprve musíme vypočítat derivaci logistické funkce σ a parciální derivaci loss function L dle $\mathbf{f^7}(i)$ (pro jeden vstup, tedy $n = 1$), kde $i \in \{1, \dots 10\}$,

$$\sigma'(x) = \frac{0 - 1 \cdot e^{-x} \cdot (-1)}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-1}} - \frac{1}{(1 + e^{-x})^2} = \sigma(x)\Big(1 - \sigma(x)\Big), \quad (13)$$

$$\frac{\partial L}{\partial \mathbf{f^7}(i)} = \frac{1}{2} \cdot 2\Big(\mathbf{f^7}(i) - \mathbf{o^7}(i)\Big) \cdot 1 = \mathbf{f^7}(i) - \mathbf{o^7}(i). \quad (14)$$

Abychom mohli vypočítat parciální derivace neznámých sedmé vrstvy, musíme prve vypočítat $\Delta\mathbf{z^7}$. Pro každé $i \in \{1, \dots, 10\}$ platí dle řetízkového pravidla a vztahů (13) a (14)

$$\begin{aligned} \Delta\mathbf{z^7}(i) &\overset{\text{def}}{=} \frac{\partial L}{\partial \mathbf{z^7}(i)} = \frac{\partial L}{\partial \mathbf{f^7}(i)} \cdot \frac{\partial \mathbf{f^7}(i)}{\partial \mathbf{z^7}(i)} = \Big(\mathbf{f^7}(i) - \mathbf{o^7}(i)\Big) \cdot \sigma'\Big(\mathbf{z^7}(i)\Big) \\ &= \Big(\mathbf{f^7}(i) - \mathbf{o^7}(i)\Big) \cdot \mathbf{f^7}(i) \cdot \Big(1 - \mathbf{f^7}(i)\Big), \end{aligned} \quad (15)$$

a tedy

$$\Delta\mathbf{z^7} = \Big(\mathbf{f^7} - \mathbf{o^7}\Big) \odot \mathbf{f^7} \odot \Big(1 - \mathbf{f^7}\Big), \quad (16)$$

kde $\odot$ značí násobení vektorů po složkách. Nyní můžeme vypočítat ΔW^7. Pro všechna $i \in \{1, \dots, 10\}$ a $j \in \{1, \dots, 128\}$ platí

$$\begin{aligned} \Delta W^7(i,j) &\overset{\text{def}}{=} \frac{\partial L}{\partial W^7(i,j)} = \frac{\partial L}{\partial \mathbf{z^7}(i)} \cdot \frac{\partial \mathbf{z^7}(i)}{\partial W^7(i,j)} = \Delta\mathbf{z^7}(i) \cdot \frac{\partial \mathbf{z^7}(i)}{\partial W^7(i,j)} \\ &= \Delta\mathbf{z^7}(i) \cdot \frac{\partial}{\partial W^7(i,j)}\Bigg(\sum_{l=1}^{128}\Big(W^7(i,l) \cdot \mathbf{f^6}(l)\Big) + \mathbf{b^7}(i)\Bigg) \\ &= \Delta\mathbf{z^7}(i) \cdot \mathbf{f^6}(j), \end{aligned} \quad (17)$$

z čehož plyne

$$\Delta W^7 = \Delta\mathbf{z^7}(\mathbf{f^6})^T. \quad (18)$$

Můžeme také vypočítat $\Delta\mathbf{b^7}$. Pro každé $i \in \{1, \dots, 10\}$ totiž platí

$$\begin{aligned} \Delta\mathbf{b^7}(i) &\overset{\text{def}}{=} \frac{\partial L}{\partial \mathbf{b^7}(i)} = \frac{\partial L}{\partial \mathbf{z^7}(i)} \cdot \frac{\partial \mathbf{z^7}(i)}{\partial \mathbf{b^7}(i)} = \Delta\mathbf{z^7}(i) \cdot \frac{\partial \mathbf{z^7}(i)}{\partial \mathbf{b^7}(i)} \\ &= \Delta\mathbf{z^7}(i) \cdot \frac{\partial}{\partial \mathbf{b^7}(i)}\Bigg(\sum_{j=1}^{128}\Big(W^7(i,j) \cdot \mathbf{f^6}(j)\Big) + \mathbf{b^7}(i)\Bigg) \\ &= \Delta\mathbf{z^7}(i), \end{aligned} \quad (19)$$

a proto

$$\Delta\mathbf{b}^{\mathbf{7}} = \Delta\mathbf{z}^{\mathbf{7}}. \tag{20}$$

Nyní vypočítáme $\Delta\mathbf{f}^{\mathbf{6}}$. Pak spočítáme parciální derivace neznámých další (tj. šesté) vrstvy zcela analogicky (s vrstvou sedmou). Pro každé $i \in \{1, \ldots, 128\}$ platí

$$\begin{aligned}
\Delta\mathbf{f}^{\mathbf{6}}(i) &\overset{\text{def}}{=} \frac{\partial L}{\partial \mathbf{f}^{\mathbf{6}}(i)} = \sum_{j=1}^{10} \frac{\partial L}{\partial \mathbf{z}^{\mathbf{7}}(j)} \cdot \frac{\partial \mathbf{z}^{\mathbf{7}}(j)}{\partial \mathbf{f}^{\mathbf{6}}(i)} \\
&= \sum_{j=1}^{10} \Delta\mathbf{z}^{\mathbf{7}}(j) \cdot \frac{\partial}{\partial \mathbf{f}^{\mathbf{6}}(i)} \Big(\sum_{l=1}^{128} \Big(W^7(j,l) \cdot \mathbf{f}^{\mathbf{6}}(l) \Big) + \mathbf{b}^{\mathbf{7}}(j) \Big) \\
&= \sum_{j=1}^{10} \Delta\mathbf{z}^{\mathbf{7}}(j) \cdot W^7(j,i),
\end{aligned} \tag{21}$$

takže

$$\Delta\mathbf{f}^{\mathbf{6}} = \Big(W^7\Big)^T \Delta\mathbf{z}^{\mathbf{7}}. \tag{22}$$

Nyní spočítáme $\Delta\mathbf{z}^{\mathbf{6}}$. Pro každé $i \in \{1, \ldots, 128\}$ platí

$$\begin{aligned}
\Delta\mathbf{z}^{\mathbf{6}}(i) &\overset{\text{def}}{=} \frac{\partial L}{\partial \mathbf{z}^{\mathbf{6}}(i)} = \frac{\partial L}{\partial \mathbf{f}^{\mathbf{6}}(i)} \cdot \frac{\partial \mathbf{f}^{\mathbf{6}}(i)}{\partial \mathbf{z}^{\mathbf{6}}(i)} = \Delta\mathbf{f}^{\mathbf{6}}(i) \cdot \sigma'\Big(\mathbf{z}^{\mathbf{6}}(i)\Big) \\
&= \Delta\mathbf{f}^{\mathbf{6}}(i) \cdot \mathbf{f}^{\mathbf{6}}(i) \cdot \Big(1 - \mathbf{f}^{\mathbf{6}}(i)\Big),
\end{aligned} \tag{23}$$

a tedy

$$\Delta\mathbf{z}^{\mathbf{6}} = \Delta\mathbf{f}^{\mathbf{6}} \odot \mathbf{f}^{\mathbf{6}} \odot \Big(1 - \mathbf{f}^{\mathbf{6}}\Big). \tag{24}$$

Již můžeme vypočítat ΔW^6. Pro všechna $i \in \{1, \ldots, 128\}$ a $j \in \{1, \ldots, 512\}$ platí

$$\begin{aligned}
\Delta W^6(i,j) &\overset{\text{def}}{=} \frac{\partial L}{\partial W^6(i,j)} = \frac{\partial L}{\partial \mathbf{z}^{\mathbf{6}}(i)} \cdot \frac{\partial \mathbf{z}^{\mathbf{6}}(i)}{\partial W^6(i,j)} = \Delta\mathbf{z}^{\mathbf{6}}(i) \cdot \frac{\partial \mathbf{z}^{\mathbf{6}}(i)}{\partial W^6(i,j)} \\
&= \Delta\mathbf{z}^{\mathbf{6}}(i) \cdot \frac{\partial}{\partial W^6(i,j)} \Big(\sum_{l=1}^{512} \Big(W^6(i,l) \cdot \mathbf{v}^{\mathbf{5}}(l) \Big) + \mathbf{b}^{\mathbf{6}}(i) \Big) \\
&= \Delta\mathbf{z}^{\mathbf{6}}(i) \cdot \mathbf{v}^{\mathbf{5}}(j),
\end{aligned} \tag{25}$$

z čehož plyne

$$\Delta W^6 = \Delta \mathbf{z^6} (\mathbf{v^5})^T. \tag{26}$$

Podobně vypočítáme $\Delta \mathbf{b^6}$, neboť pro každé $i \in \{1, \ldots, 128\}$ platí

$$\begin{aligned} \Delta \mathbf{b^6}(i) &\overset{\text{def}}{=} \frac{\partial L}{\partial \mathbf{b^6}(i)} = \frac{\partial L}{\partial \mathbf{z^6}(i)} \cdot \frac{\partial \mathbf{z^6}(i)}{\partial \mathbf{b^6}(i)} = \Delta \mathbf{z^6}(i) \cdot \frac{\partial \mathbf{z^6}(i)}{\partial \mathbf{b^6}(i)} \\ &= \Delta \mathbf{z^6}(i) \cdot \frac{\partial}{\partial \mathbf{b^6}(i)} \left(\sum_{j=1}^{512} \left(W^6(i,j) \cdot \mathbf{v^5}(j) \right) + \mathbf{b^6}(i) \right) \\ &= \Delta \mathbf{z^6}(i), \end{aligned} \tag{27}$$

a tak

$$\Delta \mathbf{b^6} = \Delta \mathbf{z^6}. \tag{28}$$

Opět analogicky (s $\Delta \mathbf{f^6}$) vypočítáme $\Delta \mathbf{v^5}$. Pro každé $i \in \{1, \ldots, 512\}$ totiž platí

$$\begin{aligned} \Delta \mathbf{v^5}(i) &\overset{\text{def}}{=} \frac{\partial L}{\partial \mathbf{v^5}(i)} = \sum_{j=1}^{128} \frac{\partial L}{\partial \mathbf{z^6}(j)} \cdot \frac{\partial \mathbf{z^6}(j)}{\partial \mathbf{v^5}(i)} \\ &= \sum_{j=1}^{128} \Delta \mathbf{z^6}(j) \cdot \frac{\partial}{\partial \mathbf{v^5}(i)} \left(\sum_{l=1}^{512} \left(W^6(j,l) \cdot \mathbf{v^5}(l) \right) + \mathbf{b^6}(j) \right) \\ &= \sum_{j=1}^{128} \Delta \mathbf{z^6}(j) \cdot W^6(j,i), \end{aligned} \tag{29}$$

a tedy

$$\Delta \mathbf{v^5} = \left(W^6 \right)^T \Delta \mathbf{z^6}. \tag{30}$$

Nyní se dostáváme k vektoralizaci. Jelikož se jedná pouze o přeskládání složek 3-dimenzionálního tenzoru do podoby vektoru, můžeme odvodit ΔS^4 přímo z $\Delta \mathbf{v^5}$. Neformálně tuto skutečnost zapíšeme pomocí funkce V^{-1}

$$\Delta S^4 = V^{-1}(\Delta \mathbf{v^5}). \tag{31}$$

Skrze vrstvu max-poolingu se backpropagation dostane snadno. Vzhledem k tomu, že ze tři ze čtyř hodnot výsledek (a tedy ani funkci L) vůbec neovlivní, je parciální derivace dle těchto třech prvků ΔC^3 nulová. Parciální derivace zbylé (čtvrté) proměnné je pak shodná s příslušnou parciální derivací prvku ΔS^4. Nechť M^4 je množina všech trojic (i,j,l), kde $i \in \{1,\ldots,32\}$ a $j,l \in \{1,\ldots,8\}$, pro které je $C^3(i,j,l) = S^4(i, \lceil \frac{j}{2} \rceil, \lceil \frac{l}{2} \rceil)$. Pak můžeme pro každé $i \in \{1,\ldots,32\}$ a $j,l \in \{1,\ldots,8\}$ vyjádřit $\Delta C^3(i,j,l)$ pomocí ΔS^4

$$\Delta C^3(i,j,l) \stackrel{\text{def}}{=} \frac{\partial L}{\partial C^3(i,j,l)} = \begin{cases} \Delta S^4(i, \lceil \frac{j}{2} \rceil, \lceil \frac{l}{2} \rceil) & \text{pokud } (i,j,l) \in M^4, \\ 0 & \text{jinak.} \end{cases} \tag{32}$$

Dostali jsme se k nejtěžší části – backpropagation u konvolunčních vrstev. Začneme výpočtem ΔZ^3. Pro každé $i \in \{1,\ldots,32\}$ a $j,l \in \{1,\ldots,8\}$ platí

$$\begin{aligned} \Delta Z^3(i,j,l) &\stackrel{\text{def}}{=} \frac{\partial L}{\partial Z^3(i,j,l)} = \frac{\partial L}{\partial C^3(i,j,l)} \cdot \frac{\partial C^3(i,j,l)}{\partial Z^3(i,j,l)} \\ &= \Delta C^3(i,j,l) \cdot \sigma'\Big(Z^3(i,j,l)\Big) \\ &= \Delta C^3(i,j,l) \cdot C^3(i,j,l) \cdot \Big(1 - C^3(i,j,l)\Big). \end{aligned} \tag{33}$$

Pokračujme výpočtem ΔK^3. Pro všechna $i \in \{1,\ldots,32\}$, $j \in \{1,\ldots,16\}$ a $l,m \in \{1,\ldots,5\}$ platí

$$\begin{aligned} \Delta K^3(i,j,l,m) &\stackrel{\text{def}}{=} \frac{\partial L}{\partial K^3(i,j,l,m)} \\ &= \sum_{n=1}^{8}\sum_{o=1}^{8} \frac{\partial L}{\partial Z^3(i,n,o)} \cdot \frac{\partial Z^3(i,n,o)}{\partial K^3(i,j,l,m)} \\ &= \sum_{n=1}^{8}\sum_{o=1}^{8} \Delta Z^3(i,n,o) \cdot \frac{\partial}{\partial K^3(i,j,l,m)} \Bigg(\\ &\sum_{p=1}^{16}\sum_{q=0}^{4}\sum_{r=0}^{4} \Big(S^2(p,n+q,o+r) \cdot K^3(i,p,q+1,r+1)\Big) + \mathbf{b^3}(i) \Bigg) \\ &= \sum_{n=1}^{8}\sum_{o=1}^{8} \Delta Z^3(i,n,o) \cdot S^2(j,n+l-1,o+m-1). \end{aligned} \tag{34}$$

Tvar se třemi sumacemi vychází z definice konvoluce (9) pro $s = 1$. Poznamenejme, že výsledek by opět bylo možné zapsat stručněji pomocí konvoluce. Nyní vypočítáme $\Delta\mathbf{b^3}$. Pro každé $i \in \{1,\ldots,32\}$ platí

$$\begin{aligned}
\Delta\mathbf{b^3}(i) &\stackrel{\text{def}}{=} \frac{\partial L}{\partial \mathbf{b^3}(i)} = \sum_{j=1}^{8}\sum_{l=1}^{8} \frac{\partial L}{\partial Z^3(i,j,l)} \cdot \frac{\partial Z^3(i,j,l)}{\partial \mathbf{b^3}(i)} \\
&= \sum_{j=1}^{8}\sum_{l=1}^{8} \Delta Z^3(i,j,l) \cdot \frac{\partial}{\partial \mathbf{b^3}(i)} \Bigg(\\
&\sum_{m=1}^{16}\sum_{n=0}^{4}\sum_{o=0}^{4} \Big(S^2(m, j+n, l+o) \cdot K^3(i, m, n+1, o+1) \Big) + \mathbf{b^3}(i) \Bigg) \\
&= \sum_{j=1}^{8}\sum_{l=1}^{8} \Delta Z^3(i,j,l).
\end{aligned} \tag{35}$$

Tenzor ΔS^2 vypočítáme trochu odlišně, neboť je konvoluční vrstva nejen před ním, ale i po něm. Pro každé $i \in \{1, \ldots, 16\}$ a $j, l \in \{1, \ldots, 12\}$ platí

$$\begin{aligned}
\Delta S^2(i,j,l) &\stackrel{\text{def}}{=} \frac{\partial L}{\partial S^2(i,j,l)} \\
&= \sum_{m=1}^{32} \sum_{\substack{n=0,\\ j-n\geq 1}}^{4} \sum_{\substack{o=0,\\ l-o\geq 1}}^{4} \frac{\partial L}{\partial Z^3(m, j-n, l-o)} \cdot \frac{\partial Z^3(m, j-n, l-o)}{\partial S^2(i,j,l)} \\
&= \sum_{m=1}^{32} \sum_{\substack{n=0,\\ j-n\geq 1}}^{4} \sum_{\substack{o=0,\\ l-o\geq 1}}^{4} \Delta Z^3(m, j-n, l-o) \cdot \frac{\partial}{\partial S^2(i,j,l)} \Bigg(\\
&\sum_{p=1}^{16}\sum_{q=0}^{4}\sum_{r=0}^{4} \Big(S^2(p, j-n+q, l-o+r) \cdot K^3(m, p, q+1, r+1) \Big) + \mathbf{b^3}(m) \Bigg) \\
&= \sum_{m=1}^{32} \sum_{\substack{n=0,\\ j-n\geq 1}}^{4} \sum_{\substack{o=0,\\ l-o\geq 1}}^{4} \Delta Z^3(m, j-n, l-o) \cdot K^3(m, i, n+1, o+1).
\end{aligned} \tag{36}$$

Skrze další max-pooling vrstvu se dostaneme stejně jako přes tu první. Nechť M^2 je množina všech trojic (i,j,l), kde $i \in \{1, \ldots, 16\}$ a $j, l \in \{1, \ldots, 24\}$, pro které je $C^1(i,j,l) = S^2(i, \lceil \frac{j}{2} \rceil, \lceil \frac{l}{2} \rceil)$. Pak pro každé $i \in \{1, \ldots, 16\}$ a $j, l \in \{1, \ldots, 24\}$ platí

$$\Delta C^1(i,j,l) \stackrel{\text{def}}{=} \frac{\partial L}{\partial C^1(i,j,l)} = \begin{cases} \Delta S^2(i, \lceil \frac{j}{2} \rceil, \lceil \frac{l}{2} \rceil) & \text{pokud } (i,j,l) \in M^2, \\ 0 & \text{jinak.} \end{cases} \tag{37}$$

Již stačí spočítat parciální derivace neznámých první konvoluční vrstvy. Výpočet je analogický s druhou konvoluční vrstvou. Nejprve spočítáme ΔZ^1. Pro každé $i \in \{1, \ldots, 16\}$ a $j, l \in \{1, \ldots, 24\}$ platí

$$\begin{aligned}
\Delta Z^1(i,j,l) &\overset{\text{def}}{=} \frac{\partial L}{\partial Z^1(i,j,l)} = \frac{\partial L}{\partial C^1(i,j,l)} \cdot \frac{\partial C^1(i,j,l)}{\partial Z^1(i,j,l)} \\
&= \Delta C^1(i,j,l) \cdot \sigma'\Big(Z^1(i,j,l)\Big) \\
&= \Delta C^1(i,j,l) \cdot C^1(i,j,l) \cdot \Big(1 - C^1(i,j,l)\Big).
\end{aligned} \tag{38}$$

Následuje vypočet ΔK^1. Pro všechna $i \in \{1, \ldots, 16\}$ a $j, l \in \{1, \ldots, 5\}$ platí

$$\begin{aligned}
\Delta K^1(i,1,j,l) &\overset{\text{def}}{=} \frac{\partial L}{\partial K^1(i,1,j,l)} \\
&= \sum_{m=1}^{24} \sum_{n=1}^{24} \frac{\partial L}{\partial Z^1(i,m,n)} \cdot \frac{\partial Z^1(i,m,n)}{\partial K^1(i,1,j,l)} \\
&= \sum_{m=1}^{24} \sum_{n=1}^{24} \Delta Z^1(i,m,n) \cdot \frac{\partial}{\partial K^1(i,1,j,l)} \Bigg(\\
&\sum_{o=0}^{4} \sum_{p=0}^{4} \Big(I^0(1, m+o, n+p) \cdot K^1(i,1,o+1,p+1)\Big) + \mathbf{b^1}(i) \Bigg) \\
&= \sum_{m=1}^{24} \sum_{n=1}^{24} \Delta Z^1(i,m,n) \cdot I^0(1, m+j-1, n+l-1).
\end{aligned} \tag{39}$$

Jako poslední spočítáme $\Delta \mathbf{b^1}$. Pro každé $i \in \{1, \ldots, 16\}$ platí

$$\begin{aligned}
\Delta \mathbf{b^1}(i) &\overset{\text{def}}{=} \frac{\partial L}{\partial \mathbf{b^1}(i)} = \sum_{j=1}^{24} \sum_{l=1}^{24} \frac{\partial L}{\partial Z^1(i,j,l)} \cdot \frac{\partial Z^1(i,j,l)}{\partial \mathbf{b^1}(i)} \\
&= \sum_{j=1}^{24} \sum_{l=1}^{24} \Delta Z^1(i,j,l) \cdot \frac{\partial}{\partial \mathbf{b^1}(i)} \Bigg(\\
&\sum_{m=0}^{4} \sum_{n=0}^{4} \Big(I^0(1, j+m, l+n) \cdot K^1(i,1,m+1,n+1)\Big) + \mathbf{b^1}(i) \Bigg) \\
&= \sum_{j=1}^{24} \sum_{l=1}^{24} \Delta Z^1(i,j,l).
\end{aligned} \tag{40}$$

Rovnice (18), (20), (26), (28), (34), (35), (39) a (40) nám říkají, jak vypočítat gradient loss function L. To nám umožní aplikovat na funkci L metodu gradient descent.

2.4 Program

Implementací programu se zde nebudeme do detailu zabývat. Ačkoliv by bylo možné napsat program dle výše zmíněných rovnic bez využití jakékoliv knihovny v prakticky libovolném programovacím jazyce, obvykle se nějaká knihovna používá. Důvodem je nejen o mnoho jednoduší implementace, ale také znatelně rychlejší program. Knihovny jsou zpravidla založeny na nějakém rychlém programovacím jazyce, obsahují řadu optimalizací a často k výpočtu používájí nejen CPU, ale i GPU [7], které je designované na násobení matic apod., což jsou zde velmi využívané operace.

Testovali jsme 2 programy. První byl přesně dle architektury popsané v kapitole 2.2 a využíval knihovnu TensorFlow [7]. Druhý program [1] byl podstatně jednoduší, neboť se jednalo pouze o klasickou (plně propojenou) neuronovou síť bez konvolučních vrstev. Konkrétně pak obsahoval $784 = 28 \cdot 28$ neuronů ve vstupní vrstvě (pro každý pixel jeden), 45 v jediné skryté vrstvě a 10 ve výstupní vrstvě. Backpropagation této sítě lze velmi jednoduše odvodit z popisu backpropagation pro naši primární síť. Narozdíl od prvního programu využíval pouze knihovny Numpy [8], která slouží jen k algebraickým operacím. Jak již bylo naznačeno, rozpoznávat číslice je schopná i takto jednoduchá architektura, neboť se jedná o poměrně jednoduchý problém. Cílem druhé neuronové sítě bylo porovnat výsledky s tou první a ukázat, že konvoluční vrstvy mají (i zde) smysl.

2.5 Výsledky

První program byl schopný správně klasifikovat ručně psanou číslici v 98.72% případech, zatímco úspěšnost druhého programu v rozpoznávání číslic byla pouze 95.83%. Oba programy vyžadovaly k tréninku přibližně 20 minut. Ani jednoduchá architektura si tedy nevedla příliš špatně. Na druhou stranu jsou konvoluční sítě i v tomto problému znatelně lepší, ačkoliv samozřejmě mnohem komplikovanější.

3 Závěr

Přestože výše popsané neuronové sítě dosahují poměrně dobré úspěšnosti (takřka 99%), pro reálnou aplikaci by to chtělo ještě vylepšit. Tyto technologie se mohou používat například na poštách k rozpoznávání číslic poštovního směrovacího čísla, kde by bylo poněkud problamtické, pokud by každý stý dopis přišel na špatnou adresu. Naštěstí existuje řada komplikovanějších neuronových sítí, které jsou však nad rámec této práce. Ty pak využívají daleko více vrstev a například tzv. dropout, který slouží k ignoraci náhodných neuronů, čímž se nejen

snižuje množství operací, ale především se tím zabraňuje tzv. overfittingu, což je v kontextu neuronových sítí zjednodušeně to, že se parametry uzpůsobí příliš přesně tréninkovým vstupům a následně mají problém se vstupy novými. Dále se běžně používá augmentace dat, která slouží k zobecnění vstupů. Například u zpracování obrázků to znamená náhodné posunutí, otočení a oříznutí či změnu kontrastu, jasu apod. Výhodnější také mohou být jiné aktivační či ohodnocovací funkce. Například tzv. cross entropy může umožňovat (především na začátku tréninkového procesu) značně rychlejší učení než námi uvedená loss function [1]. Možných modifikací na zlepšení je mnoho a vzhledem k tomu, že neuronové sítě jsou nyní v rozkvětu, tak se každým rokem objevuje řada nových metod.

Reference

[1] neuralnetworksanddeeplearning.com

[2] web.eecs.utk.edu/žzhang61/docs/reports/2016.10%20-%20Derivation%20of%20Backpropagation%20in%20Convolutional%20Neural%20Network%20(CNN).pdf

[3] yann.lecun.com/exdb/mnist/

[4] medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

[5] cs231n.github.io/convolutional-networks/

[6] www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

[7] www.tensorflow.org/

[8] numpy.org