Rozpoznávání znaků v bitmapách pomocí neuronových sítí

David Bouška

4. září 2014

1 Úvod

V této práci se budu zabývat problémem automatického rozpoznávání znaků z bitmap. Popíšu a pokusím se vysvětlit základní myšlenky neuronových sítí. Dále osvětlím problematiku automatického rozpoznávání znaků, anglicky OCR - optical character recognition, a uvedu ji do souvislosti s neuronovými sítěmi. Budu předpokládat znalost základů lineární algebry, části matematické analýzy - funkcí více proměnných, a pro účely třetí sekce druhé kapitoly bude třeba si alespoň na intuitivní úrovni promyslet metodu spádu - optimalizační metodu pro hledání lokálních extrémů funkcí více proměnných. Vzhledem ke snaze udržet fyzický rozsah práce na minimu budu některé technicky náročnější pasáže osvětlovat méně formálním způsobem.

2 Neuronové sítě

2.1 Reálný popis

Tuto část je třeba brát s rezervou - popis nebude úplný, budu se soustředit pouze na části, které nám nějakým způsobem pomohou dobrat se užitečného výsledku.

Definice 2.1. *Synapsi budeme chápat jako spoj mezi dvěma buňkami.*

Definice 2.2. *Signál budeme simulovat numerickou hodnotou putující neurony a synapsemi.*

Definice 2.3. *Neuron budeme chápat jako buňku navázanou na vstupní a výstupní synapse, která po přijetí signálu může vyslat na výstup vlastní signál.*

Definice 2.4. *Neuronovou síť budeme chápat jako uskupení synapsemi propojených neuronů.*

Nutno dodat, že netvrdím, že synapse se dají jednoznačně klasifikovat na vstupní a výstupní, tento pojem je relativní - pro různé neurony mají dané synapse různou funkci, ovšem zjednoduší se nám tím formulace některých myšlenek.

Definice 2.5. *Práh neuronu budeme chápat jako mez pro vstupní signály, která určuje, zda neuron propustí signál na výstup.*

Definice 2.6. *Váhu synapse budeme chápat jako její schopnost vést signály.*

Práh i váhu budeme pro naše účely charakterizovat reálnými hodnotami. Přirozeně s prahem budeme porovnávat, váhou násobit posílané signály.

2.2 Matematický model

V této části popíšu myšlenkový proces přetváření výše definované neuronové sítě na její matematický model a osvětlím, k čemu taková konstrukce může sloužit.

Takovou síť budeme chtít simulovat za výpočetními účely a tedy z praktických důvodů budeme klasifikovat neurony na vstupní, skryté a výstupní. Vstupní nebudou přijímat signály od jiných

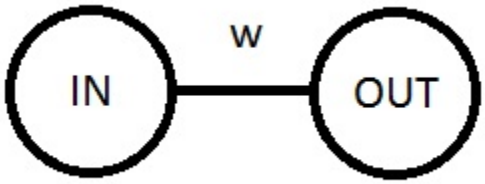
neuronů, tedy množina jejich vstupních synapsí bude prázdná, v simulaci jim budou signály nastavovány zvenku a nebudou podléhat jejich prahům. Pro skryté neurony se nic nemění. Výstupní neurony nebudou dále odesílat signály jiným neuronům, tedy množina jejich výstupních synapsí bude prázdná, ale v simulaci z nich budeme číst signály, a tyto signály budeme považovat za výstup sítě. Z praktických důvodů definujeme ještě jeden pojem související s prahem neuronu.

Definice 2.7. *Prahovou funkci budeme chápat jako reálnou funkci reálné proměnné, která na základě příchozího signálu určí hodnotu odchozího signálu.*

Dle komentáře definic v předchozí kapitole bude základní prahová funkce pro práh neuronu a definována předpisem

$$f(x) := \begin{cases} 1 & \text{pokud } x \geq a \\ 0 & \text{jinak} \end{cases}$$

Důvod volby hodnot 0, 1 za moment vyplyne z výkladu, ačkoliv zdaleka není jediný možný. Pro další krok si ukažme jednu z nejjednodušších sítí:



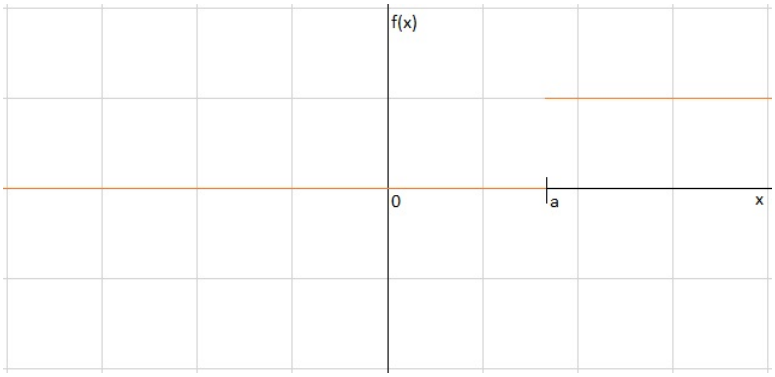
Této síti se říká perceptron. IN je vstupní neuron, OUT je výstupní. Nyní provedeme ručně simulaci chování takovéto sítě: Označme práh výstupního neuronu a, váhu synapse w, prahovou funkci f. Nastavme na vstup hodnotu x. Výstupnímu neuronu bude vyslána hodnota x, tedy na jeho vstup dorazí $w \cdot x$ a neuron zobrazí $f(w \cdot x)$.

Ač se tato síť může zdát jako samoúčelná hříčka, není tomu tak. Na této základní představě bude znázorněno několik triků, které nám umožní zjednodušit konečnou reprezentaci, a ukážeme si, jaké problémy umí tento druh sítí se správně nastavenými vahami a prahy řešit.

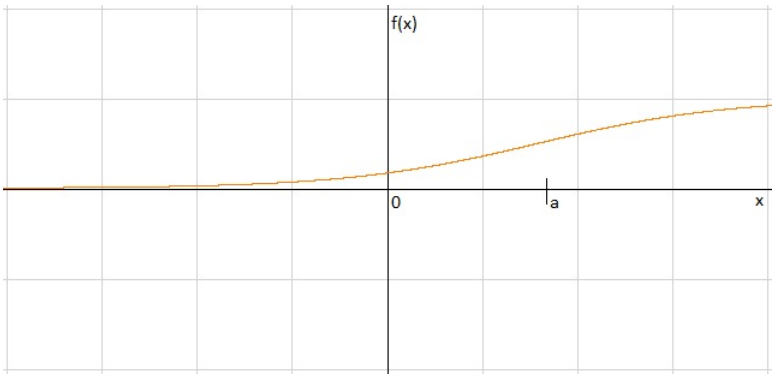
Vzorový problém je klasifikace objektů z reálného světa podle jejich vlastností. Tato triviální síť dovede, pokud má správně nastavenou váhu a práh vstupního neuronu, rozhodovat například o tom, zda při přejezdu vozidla po provizorním mostě hrozí, na základě jeho hmotnosti, reálné nebezpečí zhroucení konstrukce. Nastavme práh neuronu na maximální povolenou hmotnost vozidla a w = 1. Po nastavení váhy konkrétního vozidla na vstup a průběhu simulace bude na výstupu 1 právě tehdy, když vozidlo překračuje povolenou hmotnost a tedy nebezpečí hrozí.

Výše uvedená rozhodovací procedura má od reality ale poměrně daleko. Pokud by přes most projelo vozidlo, které by překročilo hmotnost o pár kilogramů, nemuselo by k žádné nehodě dojít, a naopak, pokud by vozidlo bylo 8x těžší, k havárii by s největší pravděpodobností došlo. Uvidíme, že neuronové sítě obecně mají daleko větší užitek v situacích, kde se vyplatí mít kromě "ano" a "ne" i možnost "možná", tedy kde se vyplatí mluvit o pravděpodobnostech.

Graf naší prahové funkce f vypadá pro práh a takto:

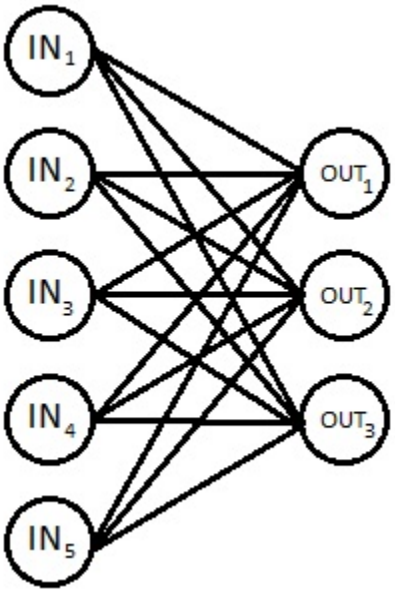


Pro pravděpodobnostní pojetí reálné situace se nám ale mnohem více hodí funkce spojitá, hladká. Jedna taková nabízející se třída funkcí je tvaru $f(x) = \frac{1}{1+e^{-\alpha(x-a)}}; \alpha \in \mathbb{R}^{>0}$. Funkci tohoto tvaru se říká sigmoida. Pro kanonickou volbu $\alpha = 1$ vypadá graf funkce takto:



Povšimněme si, že funkce je pro pravděpodobnostní pojetí situací vhodná ve smyslu chování v $\pm\infty$, a spojitosti. S takovouto funkcí bude výstup sítě vypadat jinak: $f(w \cdot x)$ nenabývá hodnot z $\{0, 1\}$ ale z $(0, 1)$. Síť nyní nerozhoduje formou ano/ne, ale udává pravděpodobnosti, zda dojde k nehodě, za předpokladu, že má rozumně nastavenu váhu synapse a práh vstupního neuronu−nyní již nemusí jít o maximální povolenou hmotnost.

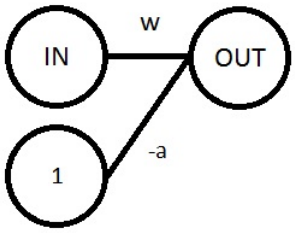
Uveďme další, již méně triviální příklad neuronové sítě:



Této síti se stále říká perceptron, a jinými druhy sítí se zde zabývat nebudeme. Vlevo je vstupní vrstva, vpravo pak výstupní. Obecně této síti nastavíme na vstup vektor a po průběhu simulace dostaneme na výstup jiný vektor, tedy tato konkrétní síť může sloužit ke klasifikaci libovolných situací, ve kterých je k dispozici 5 parametrů. Jelikož nám jde o klasifikaci, přisuzujeme signálu každého výstupního neuronu míru podobnosti jedné konkrétní položce z předem známých možností. Pokud budeme chtít klasifikovat ovoce na základě hmotnosti, diametru (dvou nejvzdálenějších bodů projekce) a barvy o rgb složkách, a naše tři možnosti budou banán, jablko a grep, budeme to schopni realizovat touto sítí, při správném nastavení vah synapsí a prahů výstupních neuronů.

Zatím jsme ovšem neřešili, jakým způsobem se na vstupu do neuronu sloučí signály příchozí z různých zdrojů. Jelikož signály jsou elektrochemické povahy, nabízí se je sčítat.

Nyní osvětlím trik, který umožňuje zanedbávat práhy neuronů. Vraťme se k našemu prvnímu příkladu sítě - 2 neuronům. Místo abychom porovnávali s prahem výstupního neuronu, popř. o něj posouvali hodnotu argumentu naší sigmoidy, můžeme si vedle vstupního neuronu přimyslet ještě jeden neuron trvale nastavený na 1, a synapsi nastavit váhu na $-a$, kde a je práh výstupního neuronu. Po této úpravě můžeme vždy za práh neuronu považovat 0.



Můžeme si ověřit, že výsledek bude stejný jako v původní situaci. V prvním příkladu jsme došli ke kritériu $w \cdot x < a$, zde máme $w \cdot x - a < 0$, a pro sigmoidu je ověření také jen o rozepsání výrazu $f(w \cdot x)$. V dalších úvahách tento neuron navíc nebudeme uvažovat, dolepit do řešení se dá pohodlně až při programování odvozených algoritmů.

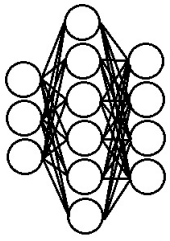
Nyní definuji jednoduchou obecnou funkci, abychom mohli naše představy shrnout do formulky.

Definice 2.8. *Nechť $(x_1, x_2, ..., x_n)^T \in \mathbb{R}^n, f : \mathbb{R} \to \mathbb{R}$. Potom definujeme funkci $P_f : \mathbb{R}^n \to \mathbb{R}^n$ předpisem $P_f((x_1, x_2, ..., x_n)^T) = (f(x_1), f(x_2), ..., f(x_n))^T$*

Tvrzení 2.1. *Nechť C je perceptron o m vstupních a n výstupních neuronech, f sigmoida, $w_{j,i}$ váha synapse vedoucí z i-tého neuronu ve vstupní vrstvě do j-tého neuronu ve výstupní vrstvě. Potom po provedení simulace se vstupním vektorem x o složkách $x_1, ..., x_m$ bude na j-tém výstupním neuronu hodnota $f(w_{j,1} \cdot x_1 + w_{j,2} \cdot x_2 + \cdots + w_{j,m} \cdot x_m)$. Pokud prvky $w_{j,i}$ zapíšeme do matice W tak, že $W_{j,i} = w_{j,i}$, a perceptron budeme chápat jako zobrazení, bude $C_{f,W}(x) = P_f(W \cdot x)$.*

Matici W můžeme právem říkat váhová matice. Nyní se můžeme zamyslet nad tím, že v tomto modelu nastává problém s lineárními zavislostmi. Závislosti posloupností vstupních vektorů se na výstupu neprojeví lineárně, ale sigmoida tento problém nemůže plně napravit. Tento problém vedl k nápadu použití skryté vrstvy neuronů, která ho skutečně v praxi odstraňuje. Simulace ale funguje na úplně stejném principu - nejdříve zobrazí vektor ze vstupní vrstvy na skrytou vrstvu, a z té potom na výstupní. Takto upravené síti se říká vícevrstvý perceptron.

Označme takovou síť C, váhové matice W_1 ze vstupní do skryté vrstvy, W_2 ze skryté do výstupní. Potom $C_{f,W_1,W_2}(x) = P_f(W_2 \cdot P_f(W_1 \cdot x))$. Nyní zbývá jedna z posledních otázek - jakým způsobem lze hledat váhové matice, aby síť dávala výsledky blízko požadovaným. Tato otázka bude zodpovězena v následující podkapitole. Naznačím, že tato úloha bude řešena plně automaticky za pomoci vzorových dat.



2.3 Hledání vah synapsí

2.3.1 Metoda spádu

V této části zjednodušeně popíšu metodu, kterou je možno hledat extrémy funkcí více proměnných. Tuto metodu potom aplikuji na problém hledání rozumných váhových matic. Nejdříve uvedu 1-dimenzionální připad:

Mějme funkci $f : \mathbb{R} \to \mathbb{R}$, všude diferencovatelnou. Naše úloha je najít pro tuto funkci lokální extrém - bez újmy na obecnosti budeme hledat minimum. Budeme pracovat s iterační metodou o předpisu $x_{n+1} = x_n - \epsilon f'(x_n)$, kde ϵ zvolíme rozumně malé, kladné (jak ho ve skutečnosti volit je již spíše inženýrská otázka). Můžeme nahlédnout na některé důležité vlastnosti této metody: Pokud je $f'(x_n) = 0$, potom $x_{n+1} = x_n$, tedy pokud je x_n stacionárním bodem f, a tedy potenciálním lokálním extrémem, bude jím i následník. Pokud $f'(x_n) > 0$, tedy funkce roste, bude $x_{n+1} < x_n$, tedy u rozumně se chovajících funkcí $f(x_{n+1}) < f(x_n)$, a obdobně pro $f'(x_n) < 0$ bude $x_{n+1} > x_n$, a tedy $f(x_{n+1}) < f(x_n)$. Tyto vlastnosti naznačují, že naše iterační metoda je schopná přibližně hledat lokální minima některých funkcí, pokud existují a správně zvolíme x_0. Jelikož se jedná spíše o praktickou záležitost a existují situace, kdy tato metoda může selhat, zdržím se shrnutí této myšlenky do věty, ale tento postup nyní zobecním pro funkce více proměnných.

Mějme funkci $f : \mathbb{R}^n \to \mathbb{R}$ a nechť ∇f existuje v každém bodě $\mathbb{R}^n$. Naše úloha zůstává stejná - hledáme lokální minimum f. Budeme pracovat s obdobnou iterační metodou: $x_{n+1} = x_n - \epsilon \nabla f(x_n)$. Stejně jako v předchozím případě lze nahlédnout na důležité vlastnosti stejné povahy, tedy chování při kladných, záporných či nulových složkách gradientu v bodě, jen je třeba použít více představivosti. Tedy i tato iterační metoda nás může dovést do lokálního extrému funkce. Pokud bychom hledali maximum místo minima, použili bychom $x_{n+1} = x_n + \epsilon \nabla f(x_n)$. Pro detailnější, obsáhlejší a případně rigoroznější vysvětlení této metody bych čtenáře rád odkázal na stránky Petra Tichého o numerické optimalizaci, anglický web OnMyPhd, nebo příslušnou literaturu zabývající se tématem optimalizace.

2.3.2 Algoritmus pro hledání vah

Nyní odvodím postup, kterým lze hledat váhové matice pro vícevrstvý perceptron. Budu se zabývat pouze případem jedné skryté vrstvy. Případ více vrstev si čtenář po pochopení níže uvedeného postupu při patřičné práci odvodí sám.

Ještě mírně upravím značení - výstup perceptronu o váhových maticích X, Y, sigmoidě f a vstupu t budu značit $C_f(t, X, Y)$, jelikož za moment budeme derivovat dle prvků v X, Y.

Obecná úloha, kterou budeme řešit, zní následovně: Nechť $X \in \mathbb{R}^{p \times m}, Y \in \mathbb{R}^{n \times p}$ jsou vahové matice, f je sigmoida, C je vícevrstvý perceptron zobrazující vektory z $\mathbb{R}^m$ do $\mathbb{R}^n$ předpisem $C_f(t, X, Y) = P_f(Y \cdot P_f(X \cdot t))$ a $\{({}_ir, {}_is); i \in [k]\}$ je "vzorová" množina dvojic vektorů. Po C požadujeme $\forall_{i \in [k]} C_f({}_ir, X, Y) = {}_is$. Ve skutečnosti se nám nepodaří dosáhnout přímo tohoto kritéria, ale budeme schopni se mu přiblížit a zároveň zachovat rozumné chování funkce.

V dalším textu bude pro vektory notace x_i značit i-tou složku vektoru x. Naším klíčem k úspěchu bude definice následující funkce, která bude charakterizovat chybu sítě pro danou dvojici vstupu a očekávaného výstupu následujícím způsobem:

$E_{f,r,s}(X, Y) = \frac{1}{2} \cdot \sum_{i=1}^{n} (C_f(r, X, Y)_i - s_i)^2$

Co nás ovšem bude ve skutečnosti zajímat není hodnota této funkce pro nějkou konkrétní dvojici (r, s), ale suma těchto chyb přes všechny zadané dvojice z množiny $\{({}_ir, {}_is); i \in [k]\}$.

$E'_f(X, Y) = \sum_{j=1}^{k} E_{f,{}_jr,{}_js}(X, Y)$

Začneme tedy s nějakými náhodně či kanonicky zvolenými váhovými maticemi, a náš cíl bude minimalizovat chybu, jakou bude perceptron vykazovat pro vzorovou množinu. Aplikujeme tedy spádovou metodu přiblíženou v části 2.3.1 na všechny prvky matic X, Y a po jednom je aktualizujeme podle následujících dvou předpisů:

$$Y_{i,j} := Y_{i,j} - \epsilon \frac{\partial E'_f(X, Y)}{\partial Y_{i,j}}$$

$$X_{i,j} := X_{i,j} - \epsilon \frac{\partial E'_f(X, Y)}{\partial X_{i,j}}$$

Při případné implementaci algoritmu založeného na této metodě budou tyto úpravy téměř jistě prováděny po prvcích, tedy stačí spočíst pro každou dvojici indexů výše uvedené parciální derivace a není třeba hledat nějakou maticovou formulku, která by zahrnovala všechny úpravy.

Budeme potřebovat ne-maticovou formulku pro i-tou složku výstupu sítě:

$C_f(t, X, Y)_i = P_f(Y \cdot P_f(X \cdot t))_i = f(\sum_{u=1}^{p} Y_{i,u} \cdot P_f(X \cdot t)_u) = f(\sum_{u=1}^{p} Y_{i,u} \cdot f(\sum_{v=1}^{m} X_{u,v} \cdot t_v))$

V tuto chvíli již stačí pouze aplikovat známá pravidla pro derivace součinu funkcí a složených funkcí na vzorec pro celkovou chybu sítě:

$$\frac{\partial E'_f(X,Y)}{\partial Y_{i,j}} = \sum_{q=1}^{k} \frac{\partial E_{f,{}_q r,{}_q s}(X,Y)}{\partial Y_{i,j}} = \sum_{q=1}^{k} \sum_{w=1}^{n} (C_f({}_q r, X, Y)_w - {}_q s_w) \cdot \frac{\partial C_f({}_q r, X, Y)_w}{\partial Y_{i,j}}$$

$$\frac{\partial C_f({}_q r, X, Y)_w}{\partial Y_{i,j}} = \begin{cases} f'(\sum_{u=1}^{p} Y_{i,u} \cdot f(\sum_{v=1}^{m} X_{u,v} \cdot {}_q r_v)) \cdot f(\sum_{v=1}^{m} X_{j,v} \cdot {}_q r_v) & \text{pokud } w = i \\ 0 & \text{jinak} \end{cases}$$

To dohromady dává:

$$\frac{\partial E'_f(X,Y)}{\partial Y_{i,j}} = \sum_{q=1}^{k} (C_f({}_q r, X, Y)_i - {}_q s_i) \cdot f'((Y \cdot P_f(X \cdot {}_q r))_i) \cdot f((X \cdot r)_j)$$

Obdobně pro druhou matici:

$$\frac{\partial E'_f(X,Y)}{\partial X_{i,j}} = \sum_{q=1}^{k} \frac{\partial E_{f,{}_q r,{}_q s}(X,Y)}{\partial X_{i,j}} = \sum_{q=1}^{k} \sum_{w=1}^{n} (C_f({}_q r, X, Y)_w - {}_q s_w) \cdot \frac{\partial C_f({}_q r, X, Y)_w}{\partial X_{i,j}}$$

$$\frac{\partial C_f({}_q r, X, Y)_w}{\partial X_{i,j}} = f'(\sum_{u=1}^{p} Y_{w,u} \cdot f(\sum_{v=1}^{m} X_{u,v} \cdot {}_q r_v)) \cdot Y_{w,i} \cdot f'(\sum_{v=1}^{m} X_{i,v} \cdot {}_q r_v) \cdot {}_q r_j$$

Což dohromady dává:

$$\frac{\partial E'_f(X,Y)}{\partial X_{i,j}} = \sum_{q=1}^{k} \sum_{w=1}^{n} (C_f({}_q r, X, Y)_w - {}_q s_w) \cdot f'((Y \cdot P_f(X \cdot {}_q r))_w) \cdot Y_{w,i} \cdot f'((X \cdot {}_q r)_i) \cdot {}_q r_j$$

Toto je téměř kompletní řešení našeho problému a na papíře se nevyplatí ho dále rozvádět, v této formě je již možné ho přenést do podoby kódu programu. Na závěr poznamenejme, že tyto iterace je třeba provést řádově 10^2 až 10^8 krát, v závislosti na velikosti sítě, která může jít do tisíců neuronů. Ačkoliv je tato metoda poměrně spolehlivá, tak s sebou přináší problémy "inženýrského typu"- základní otázky jsou např. jak volit ϵ, aby algoritmus našel vahové matice spolehlivě, a přesto rozumně rychle, nebo právě kolikrát opakovat iteraci - pokud jich bude málo, síť nebude pracovat spolehlivě, pokud jich bude příliš mnoho, síť může zdegenerovat do stavu, kdy se v okolí vzorových vstupů bude chovat divoce, téměř nespojitě. Hlavní, těžko odstranitelný problém je fakt, že výše definovaná chybová funkce obvykle obsahuje velké množství lokálních minim, a nelze zaručit, že má ve všech takových bodech hodnotu blízkou globálnímu minimu. I přes tyto nepříjemnosti je však tato metoda stále dostatečně výhodná. Jedna malá, často používaná modifikace tohoto algoritmu je neměnit jednotlivé váhy v závislosti na všech chybách v jednom kroku ale provádět úpravy po jednotlivých dvojicích $({}_i r, {}_i s)$. V praxi se ukázalo, že se tím stává algoritmus o trochu odolnější vůči stacionárním bodům.

To, že tyto chybové funkce skutečně v netriviálních případech mají více lokálních minim není na první pohled zřejmé, a zkonstruovat nějakou jednoduchou situaci, ze které by byla možnost existence více minim na první pohled vidět také není úplně jednoduché. Pokud by se čtenář chtěl přesvědčit, že tomu tak je, ať zkusí pomocí matematického softwaru prozkoumat lokální extrémy chybové funkce příslušné perceptronu $\mathbb{R} \to \mathbb{R}$ s jednou skrytou vrstvou o jednom neuronu, a se dvěma požadavky na hodnoty v bodech, neboli 2-prvkovou vzorovou množinou.

3 Popis problému rozpoznávání znaků z bitmap

V této poslední části práce stručně přiblížím problematiku rozpoznávání znaků z bitmap, zjednoduším tento problém a uvedu do souvislosti s vícevrstvým perceptronem, a na závěr učiním pár poznámek k převádění obrazů textu do elektronické podoby obecně.

Pro začátek je třeba si promyslet, že při čtení ručně psaného textu se člověk snaží najít v paměti nějaký znak - vzor, který se nejvíce podobá instanci znaku, který se snaží rozpoznat. To nás vede k myšlence, že pokud budeme po někom/něčem vyžadovat vykonání takového úkolu, musíme nejdříve poskytnout samotné vzory, ke kterým chceme přiřazovat jednotlivé instance - rozpoznávat je. V našem případě budeme uvažovat jen černobílé znaky, a v obecném případě bychom stejně nejspíše redukovali na černobílé, abychom bez použití alpha kanálu dovedli simulovat průhlednost znaku.

Reprezentace bitmap blíže nebyla nijak určena, a dokud zůstane numerická, je téměř libovolná, ale vyplatí se pro jednotlivé pixely považovat reálné hodnoty v intervalu $[0; 1]$, zapsané v

matici. Hodnota 0 odpovídá bílému pixelu, 1 černému. Trik zvaný vektorizace (převedení matice po sloupcích na vektor) nám je již z přednášek známý, tedy není třeba ho komentovat. Ten použijeme.

Nyní se dostáváme do části, ve které je třeba zredukovat problém do formy, na kterou půjde aplikovat perceptron a tato redukce lze udělat velkým množstvím méně či více podivných způsobů. Já zde vyložím jeden, který je v praxi rozumné použít. V dalším textu budu znakem V mínit vektorizaci - funkci a znakem C blíže neurčený vícevrstvý perceptron.

Zvolme tedy nějakou sadu znaků, ze které budeme chtít rozpoznávat. Pro teď ji budeme chápat jako indexovanou množinu M. Může to být česká abeceda, číslice v rozsahu 0-9, azbuka, či cokoliv jiného, co si vymyslíme. Zvolme zobrazení $F : M \to Bitmapy$, které nám ke každému znaku v M poskytne jeho obrázek (skutečně zde neztotožňuji znaky a jejich obrázky). Jedna podmínka, kerá zjednodušuje situaci, i když není nezbytná, je, aby všechny bitmapy v $F(M)$ měly stejnou velikost. Pro teď budu předpokládat, že je splněna. Pokud by nebyla, nejspíše bychom bitmapy zarovnávali či měnili jejich velikost. Nyní zkonstruujme množinu $T := \{(V(F(M_i)), e_i); i \in [|M|]\}$ a předložme ji perceptronu C k úpravě váhových matic.

Mějme bitmapu x, ze které chceme rozpoznat dosud neznámý znak. Necháme spočíst vektor $C(V(x))$ a zjistíme index i takový, že $C(V(x))_i$ má největší hodnotu ze všech složek $C(V(x))_i$. Potom x považujeme za obrázek znaku M_i.

A to je celá myšlenka této metody - zkonstruovali jsme zobrazení, které se pokouší analyticky zjistit, jak moc je předložený znak podobný jednotlivým již známým vzorům, a vzor s nejvyšší mírou podobnosti považujeme za správnou odpověď konkrétní úlohy identifikovat obrázek. Záměrně jsem použil slovo "pokouší", jelikož vše opět závisí na parametrech algoritmu pro úpravu váhových matic perceptronu. Pokud budou nesprávně zvoleny, bude perceptron podávat nesmyslné výsledky.

4 Poznámky na závěr

Zdá se, že jsem se zbytečně dlouho zabýval samotným perceptronem, když bylo možné tento problém konstrukce zobrazení o požadovaných vlastnostech "vlepit"do hlavního tématu této práce - rozpoznávání znaků. Není tomu tak. Kdybych tato 2 témata skutečně sloučil, sice bychom také dospěli k rozumnému řešení problému, ovšem jiným, pravděpodobně techničtějším a méně obecným způsobem. Perceptrony a obecně neuronové sítě pokrývají mnohem větší třídu problémů, a tedy z pedagogických důvodů bylo vhodné je uvést do povědomí čtenáře jako samostatné téma.

Čtenář by mohl namítnout, že k řešení našeho problému neuronových sítí vůbec není třeba, že existují efektivní porovnávací metody, které fungují na pevně určeném algoritmu, jehož parametry nejsou automaticky modifikovány. Ačkoliv takové algoritmy existují, a v praxi dosahují nějakého úspěchu, většinou selhávají na "hloupých"pastech, které je za znalosti daného algoritmu jednoduché zkonstruovat. Navíc operují s podobnostmi znaků v takovém smyslu, že je obtížné odchytávat případy, kdy se rozpoznávaný znak žádnému vzoru z pohledu člověka příliš nepodobá.

Tyto "hloupé"pasti samozřejmě existují i pro vícevrstvý perceptron - stačí mu poskytnout vzor znaku X, a následně instanci, ve které bude každý pixel posunutý o 1 v libovolné ose. Na takovéto situace existuje množství triviálních triků, jako vybírání maxima ne z jedné, ale z více simulací průběhů, ve kterých se vstup právě posune v různých směrech, nebo zarovnávání, a několik dalších, které jsou natolik zřejmé, že je pro zájemce ponechám jako mentální cvičení.

Jednu poznámku bych rád věnoval převodu obrazu textu do elektronické podoby - řetězce znaků obecně, a to sice, že se jedná o stále otevřený problém. Námi používaná abeceda znaků je poměrně bezproblémová co se týče čtení, a přesto je v praxi obtížné se dostat přes 99% úspěšnost, a to i s použitím pokročilejších metod, které berou ohled na délku celých slov nebo jeho částí a předpokládají, že text "dává smysl", ať už v rámci jednotlivých slov, slovních spojení, nebo celých vět. Daleko větší problém dělají různé nestandartní fonty - kurzíva nebo klasické psací písmo, které naprostá většina z nás zažila na základní škole, zde se spolehlivosti rozpoznávacích metod pohybují okolo 70-90%. Dále jsou problémové několikatisícové sady znaků pocházející z Asie – není těžké si rozmyslet, že zde by se stal vícevrstvý perceptron poměrně drahou simulací, alespoň v porovnání s perceptronem pro našich 26 znaků.

5 Zdroje

- Neuronové sítě - předmět o úvodu do umělé inteligence absolvovaný ve školním roce 2012/2013 na SSPŠ
- Záznam z 12. přednášky zimního semestru 2010 předmětu 6.034 - Artificial Intelligence na MIT přístupný na YouTube
- Shrnutí optimalizační metody gradientu na stránce webu OnMyPhd