Gymnázium Christiana Dopplera, Zborovská 45, Praha 5

ROČNÍKOVÁ PRÁCE

Regresní neuronové sítě

Vypracoval: Lukáš Caha
Třída: 8.M
Školní rok: 2017/2018
Seminář: Seminář z programování

Prohlašuji, že jsem svou ročníkovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím s využíváním práce na Gymnáziu Christiana Dopplera pro studijní účely.

V Praze dne 21. ledna 2018 Lukáš Caha

Obsah

1. Úvod

Ve světě se nachází mnoho dat v mnoha podobách a v dnešní době se dostáváme do bodu, kdy nestačíme všechny třídit a využívat na 100 %.

Neuronové sítě jsou vrcholem lidské práce v oblasti informačních technologií. Mohl bych je přirovnat k lidskému mozku. A důvodem proč je zde zmiňuji je právě jejich všestranost. Sítí můžeme pouštět dva typy dat. Jedak lidmi vyhodnocené, a poté nevyhodnocené u nichž budu chtít výsledek. Neuronové sítě se podle prvního typu dat naučí jaká je souvislost mezi vstupem a výstupem a pak můžou přibližně určit výstupy dat druhého typu. Pokud byla v prvé řadě síť správně designovaná můžeme očekávat výsledky s poměrně velikou přesností a rychlostí zpracování, na jakou jsem zvyklí u počítačů.

Touto prací bych chtěl rozebrat neuronové sítě na úroveň pochopitelnou i pro středoškoláky, kteří by chtěli začít se strojovým učením, což je obor zahrnující moderní způsoby práce s umělou inteligencí.

2. Základní pojmy

2.1 Neuron

2.1.1 Jádro

Aktivace je hodnota mezi nulou a jedničkou ($a = 0.73$). Tahle hodnota určuje míru zapnutosti neuronu. Více aktivované neurony můžou mít větší vliv na neurony v sítí přímo následující. Aktivace neuronů jsou závislé na datech, takže není možné měnit jejich hodnoty přímo.

Normalizační funkce upravuje příchozí signály, tak aby následně vytvořená hodnota zapadala do rozmezí aktivací. Přijdou-li do neuronu 4 signály všechny s maximální hodnotou, bude aktivace neuronu velmi blízko 1.

2.1.2 Synapse

Synapse je spojení mezi dvěma neurony. Toto spojení zajiťuje, že aktivace neuronu v síti je závislá na aktivacích předchozích neuronů.

Váha ovlivňuje spoje mezi neurony. Váhy spojení tvoří dohromady povahu sítě. Z libovolných vstupních dat můžu upravováním síly spojení (synapsí) vyvodit libovolné výstupní data.

2.2 Síť

Síť se skládá s několik vrstev, které jsou navzájem propojené.

Vrstva je několik neuronů, které se navzájem neovlivňují, ale jsou ovlivněny stejnými neurony a zároveň ovlivňují stejé neurony.

Bias je míra vlivu nezávislého na datech. Tato externí síla se stará o vyrovnání sítí s menším počtem neuronů a tím uspoří výpočetní výkon. V překladu je bias šum, který rozostřuje data, aby se výsledná síť nepřizpůsobila až příliš moc trénovacím datům.

2.3 Pohyb dat

2.3.1 Vstup

Typy vstupních dat

Trénovací data jsou data u nichž používáme vstupy i výstupy pro vylepšování sítě. Pokud výsledná síť uvidí znovu tato data bude na nich mít mnohem lepší výsledky, jelikož je trénovaná speciálně na tyto data a až jako vedlejší produkt je tréning na data podobná.

Testovací data je soubor vstupů, u nichž je známý i výsledek. Ten ale nikdy není ukázán síti, slouží totiž pro porovnání výsledku sítě s pravidvým výsledkem. Takto získává uživatel statistiky o kvalitě sítě.

Produkční data jsou důvod proč síť vůbec programujeme. Tyto data dostává síť během běžného používání a počítá k nim výsledky. Není však možnost určit jak by tyto výsledky měli vyjít, a proto nám už zbývá pouze doufat, že síť funguje jak popisuje teorie.

Scaling je metoda upravení hodnot z našich vstupních dat, tak aby v síti tato data vystupovala pouze jako aktivace. Dobrým příkladem je vstupní hodnota věk. V našich datech se vyskytuje člověk s maximální věkem 100 a minimálním 0. Odpovídající hodnoty aktivace potom budou $100 \rightarrow 1.0$ a $0 \rightarrow 0.0$.

2.3.2 Výstup

Back-scaling je forma získání dat zpět z neuronové sítě. Pokud zrovna trénujeme, není nutné data získávat a pak je porovnávat s očekávanými výsledky, lepší způsob je očekávané výsledky převést na jazyk, kterým komunikuje síť. Tímto samozdřejmě myslím použít scaling a převést výsledek na hodnotu mezi nulou a jedničkou.

3. Vstup

3.1 Design vstupní vrstvy

3.1.1 Velikost vrstvy

Do vstupní vrstvy musíme dát dostatečné množství neuronů, aby mohli obsáhnout všechny důležité informace. Pokud budu například používat jako data obrázky o velikosti 28×28px, tak vhodný počet vstupních neuronů je 784, takto neztratím žádná data a dokonce budou mít takhle všechny pixely rovnocený vliv. Pokud budu, ale používat jako data databázi s údaji o osobě, bude mi stačit neuronů zhruba stejný počet jako je sloupců v databázi.

3.2 Scaling

Jak již dobře víme, scaling je používán na získání hodnot mezi 0 a 1 ze vstupních dat. Našim vstupním číslům by síť totiž nemusela rozumět hned od začátku, jelikož se věk uvádí v řádu desítek, ale výplaty dosahují přibližně o 4 řády více. Vlivy těchto vstupních hodnot budou značně odlišné, což způsobí nechtěné nepřesnosti. Až později pochopíme jak funguje teoretické učení sítě zjistíme, že se tomuto kroku můžeme vyhnout, avšak z praktických zkušeností pak usoudíme, že scaling je docela užitečný krok.

3.3 Data

U začátku designování a programování sítě stojí vždy nějaká data. Každý trochu pokročilý programátor jistě dokáže vyjmenovat desítky případů, kdy mu znalost dat ulehčila práci. Tato znalost je u neuronových sítí naprosto zásadní. Když nevím co moje data znamenají nemůžu čekat, že to strojové učení odhadne za mě.

Zároveň se musím postarat, abych použil data pravdivá, ve kterých se můžou opravdu existovat korelace, které můžu potvrdit letmou úvahou. Budu-li trénovat síť na míchání barev, musím použít reálné data o míchání barev. Kdybych si výsledné barvy vymýšlel náhodně můžu čekat, že odhady sítě budou také celkem náhodné. Když se později poohlédneme za cestou od začátku do vytrénované sítě uvidíme, že dvě stejné sítě můžou být vytrénovány na dvě různé činnosti. Celkově bych to shnul upraveným příslovím: "Síť nepadá daleko od dat."

3.3.1 Trénovací

Tento pojem již známe. Pojďme se tedy podívat na příklad míchaní barev. Na trénovacích datech obzvlášť záleží. Musíme z existujících údajů tedy vybrat co nejvíc náhodně vzorky,

jejichž smíchání bylo ovlivněno co nejvíce faktory. Jednoduše pokud si vyberu na trénovaní pouze temné barvy, nemůžu potom očekávat, že síť správně smíchá dvě světlé barvy.

3.3.2 Testovací

Právě na těchto datech se uvidí, zda byly dosavadní kroky provedené úspěšně. Dosavadními kroky myslím celý postup, jelikož testovací data přichází skoro až na konci vývoje sítě. Jsou to právě tyto data, která nám oznámí jak dobře jsme odvedli práci trénování. Pokud se však nenachází podobné korelace mezi trénovacími a testovacími daty, bude náš test sítě neúspěšný a budeme se muset vrátit zpět ke trénování.

4. Forward-propagation

Je to právě tento postup, který nám umožňuje pracovat s neuoronovými sítěmi. V průběhu vývoje se snažíme, aby tato fáze proběhla co nejlépe. Je to totiž výpočet toho co si síť myslí. Když tedy spustíme forward-propagation na nějakých datech dostaneme na výstup pořád pár divných čísel, podobných těm co vzniknou scalováním. To můžeme ale jednoduše změnit back-scalingem, což nám poskytne opravdu výstup, jaký by se dal čekat od živé bytosti se slušným uvažováním. Tento výstup bude zezačátku pravděpodobně většinou chybný, ale postupem času a trénováním se dostaneme do bodu, kdy výsledky opravdu odpovídají realitě.

4.1 Typy sítí

Při náhledu do lidského mozku asi nenajdeme takhle hezky uspořádané sítě z rovnocených neuronů. Stále však reprezentují dostatečně na to, aby to celé mohlo fungovat. Je dobré vědět, že existují i jiné typy sítí. Existují modifikace jenž umožňují pracovat s pamětí, vracejí již propočítaná data, nebo sami uspořádavají síť za běhu.

4.1.1 Feed-forward

Tento typ je asi nejjednoduší případ využití strojového učení. Veškerá data postupují organizovaně dále do sítě a na konci dosáhnou výsledných neuronů.

4.1.2 Deep learning

Od slova deep (hluboké) jsou tyto sítě používány na zpracování komplikovanějších dat, jako je například rozpoznávání znaků. Uvnitř mají totiž více vrstev a každá funguje jako vstupní vrstva pro další. Takto si můžou jednotivé vrstvy předpracovat data a vrstva s výstupem už jen posbírá skoro hotové výsledky.

5. Back-propagation

Použitím této metody můžeme síť opravdu něco naučit. Stručně řečeno si síť porovná výsledky s očekáváními a poté upraví váhy v síťi, tak aby nám další feed-forward fungoval trochu lépe.

5.1 Loss function

Celá síť je naprosto definovaná svým rozložením a vahami jednotlivých synapsí. Rozložení je fixní, takže pokud chceme ze sítě dostat maximum musíme upravovat váhy. Není však žádný jasný směr, kterým se vydat. Váh v síti je často od stovek k tisícům. Proto si zavedeme funkci loss neboli ztrátu. Tato funkce nám říká, jak moc je výsledek špatný. A najitím minimální hodnoty můžeme najít ideální stav sítě.

6. Praktická práce s PC

6.1 Jazyk

Jako vývojový jazyk jsem si zvolil Python. Konkrétně používám verzi 3.6.3 na operačním systému Windows 7 64-bit. Tento jazyk jsem si zvolil především, protože je v oboru výrazně preferovaný. Tomu odpovídá i množství knihoven, které pro python v oblasti strojového učení vzniklo.

6.1.1 Vývojové prostředí

Pro vývoj používám prostředí Spyder, které již podle názvu "Scientific PYthon Development EnviRonment"bylo navrženo čistě pro práci s Pythonem. Dva nejdůležitější nástroje poskytnuté prostředím jsou IPython console a Variable explorer.

IPython console umožňuje mít program zapnutý po celou dobu práce s modelem neuronové sítě. Ze začátku se zapne Python kernel, do kterého postupně nahráváme řádky kódu a ty se vyhodnocují. Styl jakým se nejčastější píše kód z tohoto nástroje velmi benefituje. Jednotlivé části kódu na sebe totiž navazují, a proto je píšeme postupně. S náhledem do již vyhodnocené části kódu se nám následující program bude psát i opravovat rychleji. Tento nástroj opravdu oceníme, až budeme chtít upravit výpis dlouho trénované neuronové sítě, bez nutnosti pouštět celý program znovu.

Variable explorer nám dovolí dívat se živě na proměnné, které v programu existují. Největší uplatnění je asi na začátku, když se snažíme dostat data ze souboru do vstupní vrstvy a můžeme se dívat v jaké fázi se zrovna nachází.

6.1.2 Knihovny

Knihovny usnadňují práci s programem. Můžeme si je představit právě jako knihy z knihovny plné funkcí, které jsou většinou velmi jednoduché, ale i přes to vyžadují čas na vytvoření a organizaci zdrojových souborů. Navíc jsou dost často optimalizované na to co dělají.

Pandas čte vstupní soubory a nahrává data do lehce čitelných proměnných.

NumPy poskytuje značné množství jednoduchých matematických operací a funkcí, ze kterých je strojové učení složené.

Scikit-learn je podstatná knihovna pro práci s daty. Do tohoto oboru spadají i neuronové sítě. Využívám ji na rozdělení dat na testovací a trénovací, a také pro určení chyby a účinosti vytrénované sítě.

Keras je velmi pokročilá knihovna určená přímo pro neuronové sítě. Běží na základě knihoven Tensorflow od Googlu a Theano. Umožní nám zaměřit se přímo na stavění sítí a modelů, místo toho, abychom všechno psali od začátku můžeme začít pouze s teoretickými znalostmi.

MatPlotLib není knihovna, co by se značně podílela na funkčnosti. Poskytuje nám však značné možnosti ve vykreslování statistických údajů pomocí grafů.

6.2 Git

Pro zálohování a verzování používám Git. Konkrétně webovou službu GitHub (github.com/LukasCaha kde se nachází veškerý kód asociovaný k této práci.

7. Závěr

Toto je závěr mé ročníkové práce.

Literatura

[1] Birge J. R., Wets R. J.-B. (1987): Computing bounds for stochastic programing problems by means of a generalized moment problem. *Mathematics of Operations Research* **12**, 149-162.

Přílohy