**Umělá inteligence**

Matyáš Egem

2023/24

Předmět: *IT*

Třída: *Kvarta*

Vedoucí práce: *Robert Haken*

Rozsah:

* *Celkem: 10 str.*
* *Text: 7 str.*
* *Přílohy: 4*

Formát: *PDF*

Poznámky:

*K této práci je vypracovaný program, který najdete na stránkách GitHubu, v repositáři jménem „AI\_Sem“, pod uživatelem „ALQETRE“.*

Prohlášení:

*Prohlašuji, že jsem svou práci vypracoval samostatně*

**Anotace:**

Skrze tuto práci jsem vysvětlil zjednodušeně, jak funguje umělá inteligence, a jak ji vylepšit jednoduše, ale efektivně. Nepouštěl jsem se do žádného lepšího, a hlavně komplikovaného učícího procesu, než je používání náhody. V pythonu programuji už téměř dva roky, a i přes to jsem si spoustu problémů musel vyhledat, také je to můj třetí pokus a řekl bych, že první úspěšný (*A první, o kterém jsem napsal nějakou práci.*).

**Klíčová Slova:**

Dataset – Statistický soubor, používaný k výcviku a následnému testování umělé inteligence.

Neural Network – Česky neuronová síť, je síť neuronů, ve které každý neuron dělá specifické operace, a tím dohromady dokáží přemýšlet.

**Obsah:**

Anotace

Klíčová Slova

Obsah

Úvod

Teoretické Části:

č.1 – Prostředky

č.2 – Neuronová Síť

č.3 – Učící se proces

č.4 – Princip generací

Praktická část:

č.1 – Základy používání programu

č.2 – Nastavení .json souboru

Závěr

Seznam Příloh

**Úvod:**

Tato práce spočívala ve vytváření umělé inteligence, která by měla být schopná se učit a použít své znalosti k určení počasí v nějaké oblasti. V mém příkladu jsem používal dataset z Londýna, o tom budu mluvit později v první teoretické části. A proč jsem si vlastně toto vybral téma? Mám dvě motivace, ta první je pochopit a naučit se, jak fungují neuronové sítě a umělá inteligence obecně. To je ta teoretická část a druhou hlavní motivaci jsem našel v praktické části, a to umět z replikovat a vytvořit svojí vlastní síť neuronů za pomocí knihovny Pandas. Tento nástroj mi umožnil dobře pracovat s datama a lépe strukturovat můj projekt a nejen to, jedna z velkých výhod Pandas či Numpy je rychlost zpracovávání větších objemů dat, v čemž python není obecně moc dobrý. Zároveň jsem použil méně komplikovaný způsob učení, nebo spíše opravování chyb co moje umělá inteligence udělá, jednoduše jsem náhodně pozměnil nějaké hodnoty a vyzkoušel ji znovu.

**Teoretická část č. 1 – Prostředky:**

Jak už jsem zmiňoval, k realizaci tohoto projektu jsem použil python, což je interpretovaný jazyk, který je dynamicky psaný a má jednoduchou syntaxi. Já osobně jsem používal python teď už přes rok a půl a považoval bych se za zkušeného programátora. Dále jak už jsem také zmiňoval jsem používal nějaké pythnovské knihovny, jako je například: Pandas, Numpy, Math, Matplotlib, Json a Tqdm.

Pandas a Numpy jsou knihovny na práci s daty a umožňují ulehčení vytváření a následné používání a analýzu datové struktury. Pandas byl původně vytvořen a vydán v roce 2008 americkým developerem Westem McKinneyem. Pandas staví na datové struktuře Numpy a dále rozšiřuje a ulehčuje práci s daty.

Knihovna Math je jedna ze standartních pythnovských knihoven a otevírá možnost používat komplikovanější a pokročilejší matematický funkce, já jsem tuto knihovnu používal hlavně pro funkce floor() a ceil(), které zaokrouhlují nahoru nebo dolu. (V celkovém kódu jsem je použil jen dvakrát.)

Matplotlib je další užitečná knihovna, tentokráte jen, pro analýzu dat a umožňuje vytvářet grafy a ostatní grafické znázornění dat. Já jsem ji použil pro finální zobrazení úspěšnosti mého systému.

A jako poslední tu máme dvě knihovny a to Json, což je knihovna, která umožňuje práci s .json soubory, kde mám uložené veškeré nastavení pro spuštění této neuronové sítě. A Tqdm, což je knihovna, která mi umožnila zobrazit hezký progres bar během učící fáze.

Jako IDE: Integrated Developer Enviroment, jsem používal VS Code od Microsoftu, což je perfektní pro nějaké jednoduché, ale i jak vidíte, tak trochu komplexnější projekty, hlavně kvůli jeho jednoduchosti a intuitivnosti k používání.

K učení a pozdějšímu testování jsem používal dataset, který obsahuje datum, oblačnost, světelnost, globální radiaci, maximální teplotu, průměrnou teplotu, minimální teplotu, srážky, tlak, a nakonec hloubku sněhu. Každá z těchto hodnot by měla být zachycena jednou za den od 1. 1. 1979 až do 31. 12. 2020, celkem 15341 dní bylo zachyceno. Ale ne všechny tyto data používám můj systém automaticky vyřadí datum, tlak a hloubku sněhu, a to z těchto důvodů: datum je nám k ničemu, a navíc je zapsaný v této podobě: 20201231 (31. 12. 2020), což je strašně velké číslo a hodně by to ovlivnilo výsledek, podobný důvod je i u tlaku, protož tlak je v tomto datasetu zapsaný například takto: 100500.0 a to je opět moc velké číslo. A dále hloubka sněhu nějak moc nepomáhá s učícím procesem, a tak jsem ji prostě vyřadil. Také je důležité zmínit, že srážky se také nepoužívají jako vstup, ale naopak, jako kontrola výsledku, protože tento systém si nakonec vybere mezi dvěma možnostmi a to, že srážky byli 0, nebo více. A pak výsledky zkontrolujeme s reálnýma datama.

**Teoretická část č. 2 – Neuronová síť:**

Jak to vlastně funguje? Neuronová síť, dále už jen jako NW (Neural Network), vytváří imaginární hranice v několika dimenzionálním bodovém grafu. A poté určuje, zda je bod za hranicí či ne a jak „daleko“ je od hranic.

Př.:

Máme NW, který nám řekne, jakou pizzu má rád a jakou ne. Pro pizzu si definujeme jen dva parametry, abychom si ten graf mohli představit, například v mém hlavním NW je jich 6: oblačnost, světelnost, globální radiaci, maximální teplotu, průměrnou teplotu, minimální teplotu. Ale to by byl šesti dimenzionální graf, a to přeji hodně štěstí si ho představit. Takže použijeme dva, a to kolik salámu máme na pizze a kolik ananasu máme na pizze. Jinými slovy, pokud tyto dva parametry jsou stejné pro dvě pizzy pro náš NW jsou totožné. K tomu abychom mohli trénovat a učit náš NW budeme potřebovat nějaký dataset různých pizz, a to i s datama, kde je napsáno, zda ji má náš NW rád či ne. Ten může vypadat jako [*tabulka č.1*](#Tabulka1) v seznamu příloh na konci této práce, nebo klikněte na text *tabulka č.1*. Tuto tabulku můžeme zakreslit do grafu, [*graf č.1*,](#Graf1) zároveň v grafu můžete vidět tu hranici, o které jsem před chvílí mluvil, všechny body v zeleném poli jsou dobré pizzy, ale všechny ostatní jsou špatné. A ten NW docílí této imaginární hranice dvěma operacemi, a to prvně násobení a poté sčítání. Například náš NW má dva vstupy, a to ananas a salámy, a zároveň dva výstupy, a to buďto ano mám rád pizzu nebo ne nemám rád pizzu. To budou jednotlivé neurony a mezi ně můžeme přidat několik pater s dalšími neurony. Poté každý neuron spojíme se všemi neurony v následujícím patře. Viz [*obrázek č.1*](#Obrázek1)*. Kolečka jsou neurony a čáry jsou spoje mezi neurony, zelená jsou vstupní, magenta je první skryté patro, oranžová je druhé skryté patro a modrá je výsledek. Na obrázku jsou také vypsané váhy spojů mezi vstupním patrem a prvním skrytým patrem, co to znamená je vysvětleno v dalším odstavci.*

Na začátku, každý spoj dostane jedno číslo, které se pohybuje okolo jedničky, toto číslo je tak zvaná váha (weight v angličtině). Toto číslo říká, jak je spoj důležitý a v průběhu vyučování našeho NW ho budeme upravovat. Dále každý z našich neuronů, kromě vstupního patra, dostane také jedno číslo a tomu se říká bias, to nám zase dovolí trochu jinak upravovat naše imaginární hranice, které náš NW bude zakreslovat. Na začátku, každé komputace vstupů se podél všech spojů vynásobí vstupní daný neuron a váha toho spoje. Například zadáme našemu NW vzorek č. 2 z naší *tabulky č.1* a vidíme, že první spoj je 2.0, takže vynásobíme 2, protože máme dva salámy, takže máme 4 (2 \* 2.0 = 4.0), dále máme váhu 1.2 ta je spojená se spodním vstupním neuronem, to znamená 5, to vynásobíme a dostaneme 6 (1.2 \* 5 = 6.0). To uděláme pro všechny ostatní spoje. Dále ke každému spoji přičteme náš bias podle toho, k jakému neuronu je připojen, takže první spoj je připojen k prvnímu neuronu v prvním skrytém patru a řekněme, že má například přiřazené číslo 1.3, jako bias, to znamená, že přičteme 1.3 k 4.0, protože to je co nám předtím vyšlo a dostaneme hodnotu 5.3. Dále to samé uděláme u druhého spoje, ten je připojený ke stejnému neuronu, to znamená, že má stejný bias (1.3), takže 1.3 + 6 = 7.3, ale protože obě tyto hodnoty směřují do stejného neuronu, tak je sečteme. To znamená, že u každého neuronu sečteme všechny hodnoty, které do něho směřují, a to je naše finální hodnota jednotlivých neuronů. Tento proces poté zopakujeme u každého patra, ale pokaždé použijeme, jako vstupní hodnoty výsledky z předchozího patra. Na konci jen porovnáme výstupní neurony a určíme za správný ten, co má nejvyšší hodnotu. Podle toho určíme, zda má rád pizzu nebo ne.

**Teoretická část č. 3 – Učící se proces:**

Vzhledem k tomu, že všechny váhy a biasy jsou na začátek iniciovaný náhodně musíme je nějak upravit, aby náš NW dělal, co chceme, aby dělal. Tomuto procesu říkáme učení a je několik způsobů, jak docílit lepších hodnot. Já jsem použil jednu z nejjednodušších metod, a to pomocí náhody, stejně jako to dělá evoluce.

Začneme s náhodnými hodnotami a vyzkoušíme je, poté si zaznamenáme tak zvanou ztrátu, tu můžeme počítat různými způsoby, já ji počítám pomocí sigmoid funkce, která náš výstup zmenší, tak aby byl mezi 1 a 0, ale zároveň neztratil žádné informace. Jinými slovy, pokud dostaneme velké číslo jako výstup sigmoid funkce ho zmenší, ale tak aby to bylo spíše blíže k 1, protože bylo velké, ale naopak, kdybychom měli malé, či negativní číslo tato funkce to zkrátí spíše k 0. Dále vezmu hodnotu neuronu, který vím, že je správně a zprůměruju jí se všemi ostatními výsledky, které jsem vyzkoušel, a to je potom moje ztráta. Dále si uložíme nezměněnou kopii všech různých hodnot našeho NW a originál náhodně trochu pozměníme, ale ne že nastavíme všechny hodnoty úplně náhodně. Ne, my jen přičteme náhodné číslo (Může být i záporné) ke každé hodnotě. A vyzkoušíme náš NW znovu. Opět vypočítáme novou ztrátu a porovnáme je, pokud jsem se od posledně zlepšili, super nemusíme dělat nic navíc a tu originální nezměněnou kopii můžeme smazat a celí tento proces znovu zopakovat, v tomto případě jsme úspěšně zlepšili náš NW, ale pokud ztráta zůstane stejná, nebo se zhorší, tak jen zahodíme upravenou verzi a obnovíme uloženou kopii originální verze tu potom znovu poupravíme a zopakujeme celí proces. Tímto způsobem se zvládneme učit, je to stejný způsob, který používá evoluce, ale místo poupravování hodnot existují mutace, pokud jsou špatné jedinec umře, ale jeho nezmutovaný druh bude mutovat jinak a možná dokonce i lépe (Tomuto procesu se říká přirozený výběr.).

**Teoretická část č. 4 – Princip generací:**

V našem NW používáme něco, čemu se říká generace. To znamená, že máme nějaký dataset, třeba o 20 vzorcích, a každý ze vzorků protáhneme našim NW jednou, ale jsou tam dva problémy.

První problém je, že každý test bude mít jen jeden velmi specifický případ a to znamená, že poté NW nemusí dobře určovat vzorky, které nikdy neviděl. Abychom měli po každém testu, co nejobecnější výsledky a tím zvýšili šanci, že pokryjeme více úplně nových vzorků, používáme něco, čemu se říká batches (Do češtiny by se to dalo přeložit, jako skupinky, nebo balíčky.), je to vlastně skupina vzorků o nějaké velikosti, která projde NW najednou. To zajistí, aby výsledný test byl více obecný a zároveň urychluje komputace, protože místo jednotlivých generací, můžeme spustit jednu, která ale otestuje několik vzorků najednou.

Druhý problém je, že se spoléháme na náhodu, aby vyšla na první pokus, a to je strašně malá šance. K vyřešení tohoto problému prostě a jednoduše projdeme náš dataset několikrát. Tím pádem budeme mít více šancí nato, aby se náš systém nastavil správně. To docílíme generacemi, každá generace znamená, že jsme jednou prošli celí dataset.

**Praktická část č. 1 – Základy používání programu:**

Program, který jsem napsal v pythonu začne tím, že se uživatele zeptá na příkaz. Tento systém podporuje hned několik možností, jako třeba příkaz „run“, nebo zkráceně „r“, tento příkaz spustí několik předdefinovaných funkcí: „get\_proprietes()“, „get\_dataset()“, „init\_layers()“, „reset\_plot\_data()“ a nakonec „Learn()“. Kompletní seznam příkazů najdete v *[Tabulce č.2](#Tabulka2)*[.](#Tabulka2)

Po spuštění tohoto příkazu se prvně načne veškeré nastavení NW z „Network\_Proprietes.json“, poté se načtou potřebná data z datasetu. Dále se inicializují všechna patra našeho NW, a nakonec se spustí funkce „learn()“.

Ta projde všechny batche a generace a zapíše data do našeho plotu, ten pak můžeme zobrazit a pomocí příkazu „plot“.

**Praktická část č. 2 – Nastavení .json souboru:**

V tomto souboru jsou všechna potřebná nastavení pro náš NW. Úplně nahoře je nastavení „Data Set“, to definuje, jaký soubor se používá, jako dataset. Dále tu máme nastavení „Dataset size“, které definuje, jakou část datasetu budeme používat. Další nastavení „Layers“, definuje jednotlivé velikosti pater našeho NW pomocí python listu, avšak vynechá první patro, protože to se definuje v dalším nastavení „Input“. Další nastavení jménem „Batch Size“ definuje velikosti jednotlivých batchů, které systém spracovává. Poté se definuje počet generací pomocí nastavení „Generations“. Dále tu máme „Intensity“, což je trochu komplikovanější nastavení a udává o kolik se náhodně můžou upravovat jednotlivé neurony. Toto nastavení má tři módy a to „static“, „gen“ a „loss“. V každém případě se použije list se apespoň třemi hodnotami, poslední hodnota bude vždy jméno módu, který chcete použít. Při módu „static“, se do první hodnoty zadá statická intenzita, která bude po celou dobu stejná a druhé pole se nechá prázdné, protože se v tomto módu nepoužije. V módu „gen“, se na první místo dá maximální intenzita a na druhé minimální intenzita, a to znamená, že každá generace bude mít trochu jinou intenzitu, a to tak že každá další generace bude mít menší a menší intenzitu. Poslední mód je „loss“, což dělá úplně to samé, jako mód „gen“, a také se na první místo zadává maximální intenzita a na druhé minimální intenzita, až na to, že bude záležet na tom, jaká je ztráta, a čím vyšší bude tím větší bude intenzita a naopak. Zároveň v tomto případu se použije čtvrtá hodnota na posledním místě, která říká, na jaké ztrátě bude maximální intenzita. Nastavení hned po intenzitě se jmenuje „Drop colunms“ a znamená jaké sloupce se z datasetu nepoužijí vůbec. Podobné nastavení je „Output“ a to značí jakou hodnotu predikujeme, ve smyslu, zda je 0, nebo > 0, a proto jí také vyhodíme z datasetu, který používáme k učení našeho NW. Předposlední nastavení je „Test batch count“ a to definuje, kolik batchů nepoužijeme k učení, ale testování našeho NW. A nakonec tu máme „Test frequency“, což definuje, jak často děláme ty testy, a udává za kolik batchů udělá další test.

**Závěr:**

S výsledkem mého projektu jsem dost spokojen, ale to rozhodně neznamená, že bych se k touto tématu nechtěl vracet a zkoušet udělat jinak. Z celého procesu jsem si odnesl základy Pandas a rozšířil jsem si znalosti různých knihoven mimo pythnovskou standartní knihovnu. Avšak jsou zde věci, co bych udělal jinak, jako třeba, když jsem začínal tento projekt, tak jsem neměl až tak velké zkušenosti s OOP (Object Oriented Programing) a to poté vedlo k používání tříd jen, tak okrajově, což vlastně bylo mnohdy bylo horší, než kdybych je vůbec nepoužíval. Ale zároveň by to bez tříd šlo jen těžko. Také je nutno říci, že na první projekt, kde jsem se zabýval nějaký velkým počtem dat, toto téma nebylo vůbec jednoduché a radši bych si vybral nějaké jednodušší na začátek. Na druhou stranu jsem se naučil hned několik způsobů, jak data zobrazovat a ukládat alespoň trochu rozumně.

**Seznam příloh:**

*Tabulka č.1:*

|  |  |  |  |
| --- | --- | --- | --- |
| Vzorek č. | Salámy | Ananas | Má NW rád tuto pizzu |
| 1 | 0 | 3 | true |
| 2 | 2 | 5 | true |
| 3 | 3 | 0 | false |
| 4 | 2 | 0 | false |
| 5 | 2 | 2 | true |
| 6 | 1 | 1 | false |
| 7 | 3 | 2 | false |

*Graf č.1:*

*Obrázek č.1:*

1.0

1.2

2.0

0.5

0.1

1.5

*Tabulka č.2:*

|  |  |
| --- | --- |
| **Příkazy** | |
| run | r |
| learn | l |
| plot | p |
| get prop | gp |
| reload | r |
| update data | uds |
| help | h |
| prep | pr |
| reset plot | rp |

(Pro více informací použijte příkaz help)