Dokumentaci nástroje na vytvoření umělé inteligence

Contents

[úvod 3](#_Toc146112256)

[Moduly 4](#_Toc146112257)

[Vlastní funkce 4](#_Toc146112258)

[Úprava textu 4](#_Toc146112259)

[init(name) 4](#_Toc146112260)

[consec() 4](#_Toc146112261)

[vec() 5](#_Toc146112262)

[ivec() 5](#_Toc146112263)

[transform() 5](#_Toc146112264)

[number\_words() 5](#_Toc146112265)

[Data(n: int) 5](#_Toc146112266)

[split(l) 5](#_Toc146112267)

[Prepare\_Text(name:"name of the file", k) 5](#_Toc146112268)

[drown(k,name) 5](#_Toc146112269)

[Matematické funkce 6](#_Toc146112270)

[ReLu(x) 6](#_Toc146112271)

[softmax(x) 6](#_Toc146112272)

[sigmoid(x) 6](#_Toc146112273)

[one\_hot\_encode(num) 6](#_Toc146112274)

[sigmoidPrime(X) 6](#_Toc146112275)

[dev\_Relu(x) 6](#_Toc146112276)

[normalize\_vector(v) 6](#_Toc146112277)

[Neuronová sít 7](#_Toc146112278)

[class Net 7](#_Toc146112279)

[\_\_init\_\_(self,first,second,alpha) 7](#_Toc146112280)

[forward(self,x) 7](#_Toc146112281)

[cycle(self, x, Y) 7](#_Toc146112282)

[correct(self, dW1, dB1, dW2, dB2, dW3, dB3, Arda = True) 7](#_Toc146112283)

[load(self,name) 8](#_Toc146112284)

[save(self,name) 8](#_Toc146112285)

[Backprop(batch\_size,epoch,name, Autograd = True) 8](#_Toc146112286)

[translate(name,name2,name3) 8](#_Toc146112287)

[cosine\_similarity(first, second) 8](#_Toc146112288)

[cosine\_similarity\_word(first, second, Embedding\_vector) 8](#_Toc146112289)

[similliar(word,n, Embedding\_vector) 9](#_Toc146112290)

[search(word,name, Embedding\_vector) 9](#_Toc146112291)

[training\_function(Batch\_size, epochs, name,k) 9](#_Toc146112292)

[Jak program používat 9](#_Toc146112293)

[Příprava textu 9](#_Toc146112294)

[Trénování modelu 9](#_Toc146112295)

[Pracování s natrénovaným modelem 9](#_Toc146112296)

[Závěr 10](#_Toc146112297)

[Vlastní testování a tipy, jak trénovat síť 10](#_Toc146112298)

[Zdroje 10](#_Toc146112299)

# úvod

Tento program je nástroj na vytvoření vlastní neurální sítě, která dokáže zpracovat libovolný text uživatele a pomocí NLP sítě vytvořit slovník embedding vektorů. A to pomocí techniky, která se pomocí následujících slov dokáže ujasnit konotaci slova a naučením neurální sítě předpokládat další slovo také naučí síť chápat slova pomocí vektorů. Tyto vektory pak může uživatel použít pro hledání slov v textu které jsou jinak psané, ale mají podobný význam. Cílem tohoto programu bylo se naučit něco praktického o neurálních sítích a backpropagaci, vytvořit program který umí hledat slova dle konotace a vytvořit nástroj který je jednodušše přístupný a pomocí kterého si kdokoliv může vyzkoušet natrénovat vlastní text na vlastní databázi.

Program má nástroje k zpracování textu několika způsoby a vytvoří několik separátních .JSON a .TXT souborů, kde budou uloženy různé soubory, které budou při programu průběžně používány. Dále obsahuje classu Net, která inicializuje síť připravenou na vlastní učení. Pro tento program byly vybrány dvě Hidden layers. Tento počet je založen na standartu používaným spousty programátory a firmami, míň a nuance a nelinearity by se ztratili, přičemž by zároveň bylo těžší získat embedding vektory. Více vrstev a riskujeme moc pomalý výpočet a moc velká data. Zároveň bychom museli podstatně zvětšit minimální počet slov použitý v input Textu. Dále tato síť využívá tří nelinearit a to sigmoidu, ReLu a softmaxu, funkce, které budou popsané později. Velikost jednotlivých vrstev je založena na matematických propočtech a testování, pro menší data sety, které dokáže zpracovat lepší počítač, je doporučené používat okolo sta až dvou set neuronů pro Embedding vrstvu a okolo 400 až 500 pro první vrstvu. Výpočet je tak svižný a zároveň neztrácíme nuance které bychom ztratili při větších data setech.

Samotná class Net obsahuje spoustu funkcí které budou popsané později, ale je dobré zmínit, že také obsahuje metodu automatické úpravy gradientu, která měří jako často jsou jednotlivé parametry měřeny a dle toho jsou upravovány jednotlivé neurony více nebo méně. Tato metoda je velice výpočetně náročná, ale vyhýbá se možným problémům jako je přetrénování, nebo naopak pod trénování jednotlivých vrstev.

Dále program obsahuje funkce, které pracují s vlastní sítí. Tyto funkce buď používají již natrénovaných matic nebo jednotlivé operace zmíněny v úvod. Všechny funkce jsou v kódu vysvětleny, ale později se objeví i zde, a to navíc s popisem proč funkce byla designovaná tak jak byla.

# Moduly

Program používá modly JSON, random, numpy, cupy a re, kde re je používanou pouze pro kontrolu vlastního kódu.

1. Jako první se uživatel musí rozhodnout, jestli bude používat modul numpy nebo cupy, program je dělaný pro syntaxy obou modulů, takže je na uživateli aby si vybral, co bude používat. Je vysoce doporučeno uživateli používat cupy místo numpy, cupy používá technologie CUDA, které je třeba mít nainstalované na RTX NVIDEA kartách. Cupy zrychluje výpočty čtyřiceti násobně a většina operací které naopak zpomaluje, nejsou často využívaná. Numpy je slouží pouze pro uživatele bez karty s Tensor jádry nebo pro uživatele s výkonnější CPU.
2. Modul Random je využit pro inicializaci.
3. Modul JSON je použit pro jednodušší ukládání souborů a pro jejich jednoduchou manipulaci.

# Vlastní funkce

## Úprava textu

Pro natrénování libovolné klasické standratní neurální sítě je třeba velká, silná a dobře navržená databáse a to obvzlášť pro program klasifikující jazyk. Pro využití u klasického NLP je třeba umění zpracovat libovolný korpus textu, proto u modelů nevyužívající moderní tokenizaci je třeba vybrat korpus který obsahují kýžená slova často a v různých kontextech. Tento model není zamýšlen pro obecnou NLP klasifikaci ale pro užší využití jako jsou knihy nebo skripta, proto lze obecně vypustit slova která jsou použitá pouze jednou. Pro úspešný embedding proces bylo při testování ukázáno že slova by měly být použita aspoň 40 krát aby měl program představu o konotaci slova

### init(name)

Funkce init je první funkce, která zpracuje vámi zvolený text. Na vstupu za „name“ stačí napsat název vašeho souboru v uvozovkách a program vytvoří soubor init.json, která obsahuje list relevantních slov které použijeme k učení. K tomu používáme list Stop Words, které smažou veškeré slova, která nemají přesnou podstatu, která by šla obecně popsat, například slova jako „I“, „nothing“ nebo třeba „you“. Nás zajímají pouze slova kontextová, takže celý text nejdříve projdeme, unifikujeme do malých písmen, zbavíme se veškerých znaků, co nejsou písmena a poté se zbavíme slov, které nebudeme potřebovat. Poté se nám do vybrané složky uloží složka init.json se kterou dále pracujeme.

### consec()

Funkce je použitelná pouze pokud již existuje init.json, tato funkce udělá list slov dle toho kdy jaké slovo bylo poprvé využito a následovně je pak zařazeno do listu. Tuto funkci potřebujeme pro pozdější funkce a vytvoření slovníku. Vytvoří složku consec.json

### vec()

### ivec()

Funkce vec a ivec použijí složky consec.json k tomu, aby vytvořili dva slovníky, jeden který přiřazuje číslo ke slovu (vec) a druhou která udělá opak. Tyto složky později použijeme pro překládání jednotlivých slov pro síť a zpátky pro uživatele. Vytvoří příslušné složky vec a ivec.

### transform()

Funkce transform pomocí složky vec přeloží init.json do pouhých čísel, které později použijeme pro přiřazení vektorů. Vytvoří složku corpus.json

### number\_words()

Tato funkce slouží čistě pro zjednodušení používání programu, uživatel se může dozvědět kolik slov a kolik různých slov jeho data set má a dál s touto informací může nakládat.

### Data(n: int)

Tato funkce na vstupu bere číslo n, které určuje, na kolik slov se máme koukat pro určení konotace zkoumaného slova, velké modely jako chatGPT nebo Bard používají vždy celé věty, ale pro obecnost a zjednodušení je standardem používat okolo 5 až 7 slov, toto číslo je již na uvážení uživatele.

Tato funkce vytvoří slovník, kde každé slovo má přiřazené všechny své možné konotace, s tímto slovníkem můžeme později dál pracovat při backpropagaci. Vytvoří složku Data.Json.

### split(l)

Funkce split vezme složku Data.json a potom pomocí vstupu l vytvoří training data a test data v poměru l. Kde l je číslo mezi 0 a 1 kde 1 znamená že chceme, aby celý náš text byl training (nedoporučeno) a 0 že chceme celý náš data set jako test. Tato funkce po sléze vytvoří dva náhodně zamíchané listy (pro zbavení se co nejvíce přetrénovaní a zobecnění celého procesu), které jsou poté již používané při tréninku. Na výstupu se vytvoří dle svých jmen složky Train\_Data.json a Test\_Data.json

### Prepare\_Text(name:"name of the file", k)

Tato funkce je pro uživatele nejdůležitější, na vstupu chce uživatel zadat jméno textu na kterém chce síť trénovat a číslo k, které určuje kolik slov kontextu uživatel potřebuje. Potom funkce použije téměř všechny funkce dříve zmíněné k automatické přípravě programu. Zároveň v sobě má metody na určení chyby uživatele. Na výstupu vytvoří všechny dříve zmíněné složky.

### drown(k,name)

Funkce drown je použita čistě kvůli praktickým důvodům, jelikož není přístup k výkonným serverům a i za vuyžití CUDA není python nejrychlejší musí se nějak optimalizovat velikost vektorů které budou použity. Funkce drown odstraní všechny slova která se objeví méně než k krát. Funkce je designovaná na složky init.json, což by měl být parametr použit v name.

## Matematické funkce

### ReLu(x)

Jednoduchá funkce, která není lineární, často používaná při strojovém učení. Kdyby strojové učení neobsahovalo nelinearity, byla by celá síť jen jedna velká lineární kombinace, tudíž bychom pouze dělali regresi a žádné nelinearity, které používáme pro koncept chápání, by se neobjevili. X může být číslo anebo array a vrátí nám příslušnou hodnotu funkce ReLu.

### softmax(x)

Softmax je funkce která nám určí pravděpodobnost výsledku, je také nelineární, ale je hlavně využita kvůli metodě Cross Entropy, kterou používáme při backpropagaci. Ta je používaná primárně při učení strojů, kde je jen pouze jedná správná odpověď, v našem případě slovo které se snažím předpokládat pomocí konotace. Tato entropie je použita kvůli jednoduchosti matematické operace a kvůli jejímu otestování a přijetí širší AI komunitou. Na vstupu musí být array a na výstupu je opět array.

### sigmoid(x)

Další nelineární funkce, použita pro normalizaci dat a pro numerickou stabilitu celého programu. Na vstupu je array a na výstupu příslušný upravený array.

### one\_hot\_encode(num)

Funkce, která je používaná pro jasné určení vektorů jednotlivých čísel. Chceme, aby se naše síť trénovala bez jakýchkoliv iniciálních závislostech. Tudíž vektor každého slova musí být úplně stejně rozlišný od všech ostatních slov. K tomu použijeme metodu one hot encode, což vytvoří vektory velikost našeho slovníku a ke každému slovu přiřadí hypotetický vektor nul s jednou jedničkou značící naše slovo. Na vstupu je číslo, které chceme kódovat a na výstupu vektor.

### sigmoidPrime(X)

Je derivací sigmoidu, je použita pro účely backpropagace. Na vstupu je vektor, který chceme derivovat a na výstupu jeho derivace.

### dev\_Relu(x)

Derivace ReLu, na vstupu je array a na výstupu derivovaný array.

### normalize\_vector(v)

Normalizace vektoru L2, použito pro standardizaci knihovny a pro numerickou stabilitu při backpropagaci. Na vstupu je array a na výstupu jeho normalizovaná verze.

## Neuronová sít

### class Net

Vlastní síť umělé inteligence, program v sobě má řádku kódu, který vytváří objekt „net“ se kterým se dál pracuje. Vlastní síť je jen soubor matic a čísel které se mezi sebou nějakým způsobem násobí. Obsahuje funkce ukládání, načítaní, propagace, backpropagace, gradientu, úpravu matic a samozřejmě inicializace.

### \_\_init\_\_(self,first,second,alpha)

Vlastní síť musit mít parametry first, second a aplha, first je číslo, které určuje počet neuronů, které budou použity v první vrstvě a second nápodobně. Alpha je rychlost učení které bude síť podstupovat. Již byly zmíněny doporučené hodnoty, ale pro ujasnění, first by nemělo být cokoliv nad 8 pokud chcete, aby se síť něco naučila do měsíce a pod jakékoliv číslo pod 300 riskuje ztráty nuance, a tedy jakékoliv využitelnosti. Druhá vrstva by měla mít od 100 do 200 neuronů, a to záleží na velikosti data setu. Aplha je čistě k zvážení uživatele, ale měla by být v rozmezí od 1 do 0.000001.

Vlastní inicializace potom vytvoří matice neuronů, Biasu a gradientu, a to pomocí Xavierovi inicializace a He inicializace. Xavierova se používá pro inicializaci vrstev a Biasu využívající Sigmoid zatímco He je používána pro inicializaci ReLu vrstev a Biasu. Tyto náhodné inicializace se používají pro lepší výkon sítě a pro ideálnější hledání minima ztrátové funkce používané při backpropagaci. Existují spoustu metod inicializace, ale tyto metody byly v minulosti nejlépe fungujícími při NLP a cross entropy projektech.

Bias je pouhé poupravení výsledků z násobení matic, tak aby nefigurovalo v sítí pouze násobení ale i jemnější sčítání.

Další komponent je sledování gradientů, tyto matice sledují jako často jsou parametry při učení měněny a dle toho upravují jako moc jsou měněny. Dosahujeme tak normalizace a stabilizace, které by měly vést k lepším výsledkům. Možnost Automatického gradientu lze vypnout, jelikož je velice náročná a o to víc náročná pro CPU orientovanou numpy.

### forward(self,x)

Funkce sítě, která vezme vektor x a propočítá propagaci. Funkce často používaná při backpropagaci.

### cycle(self, x, Y)

Funkce cycle vypočítává gradient neboli jak se mají parametry změnit abychom maximalizovali zlepšení funkce. K tomu používáme takzvanou ztrátovou funkci, která vypočítává nejlepší změny pro naši síť pomocí chain rule a pomocí derivací.

### correct(self, dW1, dB1, dW2, dB2, dW3, dB3, Arda = True)

Funkce correct vezme gradient a opraví paramtery, pokud je Arda nastavená na True, bude používaný automatický gradient, který byl již výše zmíněn. Pouze odečteme hodnoty vypočítáné funkcí cycle.

### load(self,name)

### save(self,name)

Toto jsou funkce které ukládají a načítají síť pro příští použítí. Ukládáme je jako tensory v JSON souborech, kde „name“ znamená pod jakým jménem chceme tensor uložit a jaký tensor chceme načíst.

### Backprop(batch\_size,epoch,name, Autograd = True)

Tato funkce reprezentuje vlastní učení sítě. Využívá téměř všech vlastností Classy Net, matematické funkce a text generovaný funkcí split. Jednotlivě projíždí všechny testovací prvky z trénovacích dat, spojené vektory ze všech konotačních slov je vstupní vektor pro funkci forward a výsledek propagace poté porovnáme pomocí Cross entropie s reálným slovem. Tento proces je opakován dle čísla batch\_size a výsledky jsou poté zprůměrované a odečtené od našich parametrů. Celý tento proces probíhá dle určeného čísla epoch. Poté co síť dotrénuje je uložena pod jménem “name”. Autograd nastavuje zapnutí automatické úpravy gradientu o kterém již bylo hovořeno u Class.

### translate(name,name2,name3)

Další důležitou funkcí je funkce translate, tato funkce vezme poslední vrstvu neboli v našem případě vrstvu embedding, jelikož je napojená na softmax, a přeloží každý její řádek do korespondujícího vektoru. Poté je vytvořena složka kde každé slovo má přiřazen svůj vlastní normalizovaný embedding vektor. Embedding vektory jsou vektory které určují význam slova podle jednotlivých dimenzí vektoru, například první dimenze může být spojena s tím, jestli je popisovaná věc živá nebo ne, a naopak druhá dimenze by mohla popisovat, jestli je věc spíš kulatá nebo hranatá. Bohužel není takto lehce rozpoznatelné, co každá dimenze dělá, ale při dobře vytrénované síti dokážeme dělat s vektory dokonce jednoduchou aritmetiku. Jako například král-muž=královna. Dále pomocí embedding vektorů můžeme zjišťovat podobnost slova pomocí jejich skalárního součinu, čím víc se blíží číslo k 1 tím podobnější si slova jsou, čím blíž jsou k -1 tím rozdílnější ve složkách jsou. Takto později hledáme podobná slova v textu.

Funkce obsahuje 3 vstupy, name požaduje jméno sítě, name2 požaduje jméno překladače, v našem případě ivec, a name3 je jméno pod kterým si přejeme síť uložit.

### cosine\_similarity(first, second)

Tato funkce provede skalární součin dvou vektorů, vlastní funkce je používaná pouze u jiných funkcí, jelikož není praktické vypisovat každý vector separátně. First je první vector a second je vector druhý. Na výstupu je float.

### cosine\_similarity\_word(first, second, Embedding\_vector)

Využívá předchozí funkci a vlastnost „load“ sítě. Na vstupu jsou dvě různá slova v uvozovkách a příslušný slovník který chcete použít. Funkce si slova přeloží a na vstupu vypočte float podobnosti obou slov.

### similliar(word,n, Embedding\_vector)

Funkce má na vstupu slovo které chcete porovnávat, n, což signifikuje počet podobných slov a knihovnu kterou chcete použít. Za použití předchozích funkcí je vypočtena podobnost všech vektorů, která je potom seřazena od nejmenšího po největší

### search(word,name, Embedding\_vector)

Tato funkce hledá slova nejpodobnější slovu word v textu name, za použití slovníku Embedding vector. Použita je funkce cosine simillarity.

### training\_function(Batch\_size, epochs, name,k)

Tato funkce spojuje funkci backpropagace, drown a translate. Na vstupu je velikost tréninkových úseků stejně jako u backpropagace, počet jejich epochů, jméno pod kterým by se měl slovník uložit a číslo k které značí minimální počet instancí slova aby bylo slovo použito.

## Jak program používat

### Příprava textu

Nejprve je důležité připravit text, to je prakticky jednoduché a teoreticky těžší, nejprve je třeba zadat do přiloženého textového souboru rozsáhlý anglický text na kterém se bude umělá inteligence učit. Prakticky pouze stačí zadat anglický text který je jakkoliv formátovaný do textového souboru „text.txt“ a poté s ním dál v programu pracovat. Teoreticky je třeba rozhodnout jaký text, kolik textu a v jakém uspořádání bude text vložen. To je proces experimentalizace, ale doporučeno je dost jednorodý text který má alespoň 300000 slov,

Dále stačí pouze použít funkci Prepare\_Text, která je popsaná dříve

### Trénování modelu

Dále je třeba připravit síť, v programu na řádku 350 je připraven řádek na upravení vámi zvolené sítě a jejich parametrů. To je opět předmět experimentalizace, ale jak bude popsáno později, nejlepší výsledky jsou dosaženy pokud embedding vektor je pod velikost 300 a pokud je rychlost učení opravdu malá se zaplou Ardou. Poté stačí použít funkci Backrprop dle manuálu.

### Pracování s natrénovaným modelem

Dále už je můžete model testovat a používat dle popsaných funkcí. Pokud chcete prohledávat text pomocí funkce search, musíte nejdřív do složky s programem vložit textový soubor který bude zmíněný text prohledávat.

## Závěr

Ačkoliv je program optimalizován pro co nejmenší škálu využití, modely stále často vykazují příznaky nedotrénovasti kvůli nedostatku dat, i tak se nejlepší pokusy v některých případech trefují, což je kvůli pravděpodobnosti podobné události považováno za veliký úspěch. Existuje spoustu způsobů, jak model vylepšit, ale tato práce bez hluboké strukturové změny se dá považovat za vysoce optimalizovanou, jelikož využívá pokročilé metody inicializace, ortogonalizace, normalizace, optimalizace, auto gradientu, sebe opravování a spoustu dalších malých opatření jako numerická stabilita. Dále by se tato práce dala pokračovat do práce bakalářské za například předvedení tokenizace, zlepšení na transformery a zlepšení jazyka.

## Vlastní testování a tipy, jak trénovat síť

Při programování kódu bylo samozřejmě vyzkoušeno spoustu různých nastavení a vyšlo spoustu různých výsledků, některé otřesné a některé skoro uspokojující. Při vytváření textu je dobré vzít text obsahující podobná slova a podobné konotace, nejlepší výsledky ze začátku byly dokázány za použití pouze knihy 1984, kde model dokázal předpovědět správné slovo asi v 5% případů a následně dokázal rozeznat že slova „dobrý“ a „špatný“ byly často používány v jiných kontextech. Pro hlubší trénování je ale třeba použít velikost textu alespoň 10x větší než je právě tato kniha. Přičemž by při nejlepším měl zmiňovat podobná slova a to od podobného autora.

Vlastní rychlost učení je na uvážení uživatele, při testování bylo použito spoustu různých velikostí textu a k něm je třeba využít různé rychlosti učení, ARDA tuto potřebu snižuje, ale stále je dobré si nejdříve zjistit co funguje a co ne.

Velikost neurální sítě byla odhadnuta jako dobrá při velikostech 400 a 300, ale pro menší texty, kde výsledky byly nejlepší, je doporučené používat menší čísla a to až 100 100, při čtení odborných textů bylo zjištěno že lepší je mít první vrstvu o něco větší pro zachování detailů které by jinak byly redundantní.

Samozřejmě výsledky nejsou vynikající, ale předpovězení dalšího slova nebo pochopení konotace se několikrát podařilo a to i přes otřesnou optimalizaci pythonu, což se dá považovat při takto malém projektu jako úspěch.

## Zdroje

<http://neelnanda.io/transformer-tutorial>

<https://www.youtube.com/watch?v=hQwFeIupNP0>

<https://prateekvjoshi.com/2016/03/29/understanding-xavier-initialization-in-deep-neural-networks/>

[A Gentle Introduction to Cross-Entropy for Machine Learning - MachineLearningMastery.com](https://machinelearningmastery.com/cross-entropy-for-machine-learning/?__cf_chl_tk=JJHTJdmVI3YhN.ijC20_ZCKsm4kJGoPhGKHD31USj1M-1695140181-0-gaNycGzNDLs)

[Natural Language Processing (NLP) [A Complete Guide] (deeplearning.ai)](https://www.deeplearning.ai/resources/natural-language-processing/)