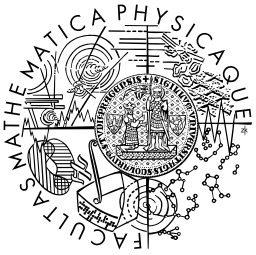


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

DIPLOMOVÁ PRÁCE

Daniel Šipoš

Analýza a vizualizácia správania jazykového modelu GPT-2

Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce: RNDr. David Mareček, Ph.D.

Studijní program: Informatika

Studijní obor: Počítačová grafika a vývoj počítačových her

Praha 2022

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Najväčšia vďaka za dokončenie tejto práce včas patrí môjmu vedúcemu práce, RNDr. Davidovi Marečkovi, Ph.D., ktorý bol naozaj veľmi aktívny a pomáhal vo všetkých možných aj nemožných smeroch.

Podieľal sa na všetkých úsekoch vypracovania tejto práce: od úvodného vysvetľovania GPT-2 a posielania článkov na samoštúdium cez nápady a pripomienky k jednotlivým módom, či rozbehanie programu a technické poznámky k nemu až po veľkú pomoc pri revidovaní textu a ubezpečenie sa, že text práce má štruktúru takú, ako by mal mať.

Prišiel aj s touto veľmi zaujímavou témou, na ktorej ma osobne bavilo pracovať posledné mesiace. Dokonca sa mu podarilo vybaviť mi účet na študentskom clusteri, kde mohli posledné týždne prebiehať generovania a výpočty, ktoré som nakoniec mohol použiť pre analýzu a nasmeroval ma aj na online nástroj MorphoDiTa, ktorý je z dielne našej fakulty a ktorý mi veľmi pomohol pri problémoch s delením textu na vety.

Obrovská vďaka patrí aj mojim textovým kritikom, Katke a Danke, ktoré strávili hodiny svojho drahocenného času čítaním a opravovaním každej čiarky, pomlčky a chýbajúcej uzatváracej zátvorky. Ony mali veľký podiel na tom, že táto práca je napísaná v takej kvalite, v akej je (a za prípadné chyby v tomto poďakovaní nemôžu).

V neposlednom rade patrí veľká vďaka aj mojej rodine, ktorá ma podporovala vo všetkých mojich krokoch, ktoré nakoniec viedli na túto fakultu a k tejto diplomovej práci. Každý z nich mal obrovský vplyv na tom, čo zo mňa nakoniec je a vo finále aj na tejto práci.

Ďakujem.

Název práce: Analýza a vizualizácia správania jazykového modelu GPT-2

Autor: Daniel Šipoš

Ústav: Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce: RNDr. David Mareček, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt: Vizualizácia komplexných modelov neurónových sietí s architektúrou Transformer je vo všeobecnosti náročná úloha, ktorá sa väčšinou rieši vizualizáciou blokov Attention a sledovaním, na ktoré slová sa tento blok zameriava. Modely Transformer ale majú veľké množstvo vrstiev, na každej vrstve majú veľké množstvo hláv Attention a každá hlava môže sledovať rôzne lingvistické znaky. My sme sa preto v tejto práci zamerali na vytvorenie programu, ktorý je určený na prehľadnejšiu vizualizáciu správania jazykového modelu GPT-2. Prišli sme so štyrmi metódami vizualizácie, ktoré skúmajú závislosti generovaných slov od prechádzajúcich slov v texte. Tieto závislosti sledujeme tak, že skúsime prvé slovo v texte vynechať alebo zameniť za podobné slovo a pozorujeme zmenu v pravdepodobnosti generovaného slova. Metódy sme vyskúšali na modele GPT-2 Medium a demonštrujeme, aké výsledky dané metódy vytvorili. Zároveň vyslovujeme hypotézy, ktoré sa pokúšajú objasniť, prečo vyšli výsledky takto.

Klíčová slova: transformer, jazykový model, GPT-2, vizualizace

Title: Analysis and visualization of the GPT-2 language model

Author: Daniel Šipoš

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. David Mareček, Ph.D., Institute of Formal and Applied Linguistics

Abstract: Visualization of deep neural network models with Transformer architecture is generally a very demanding task which is usually solved by visualizing attention blocks and monitoring which words these block focus on. However, Transformer models have many layers and there are multiple attention heads on each layer. Therefore, each head may attend to different linguistic features. In this work, we focus on developing an application that is designed to visualize the behaviour of GPT-2 language models more clearly. We propose four visualization methods that examine the dependencies of generated words on previous words in the text. We monitor these dependencies by removing one of the words in the previously generated text or replacing it with a similar word and then observing changes of the probability of the generated word. We show the results of our methods produced on the GPT-2 Medium model and formulate hypotheses with the aim to explain them.

Keywords: transformer, language model, GPT-2, visualization

Obsah

Úvod

Vo februári 2019 predstavila spoločnosť OpenAI jazykový model patriaci do triedy Transformerov s názvom GPT-2. Tento model má 1,5 miliardy parametrov a je natrénovaný na datasete WebText, ktorý obsahuje viac ako 8 miliónov webových stránok (OpenAI, 2019, Feb 14).

GPT-2 bol podľa nich taký presvedčivý, že sa pokúsili o zodpovedné zverejnenie a samotný model nezverejnili okamžite. Zverejnili iba technickú správu (Radford a kol., 2019) a oveľa menšie modely (zvané *GPT-2 small, medium a large*), na ktorých výskumníci mohli experimentovať. Dôvodom obáv bolo, aby nebol tento model zneužitý, napríklad na generovanie dezinformačných článkov.

Nakoniec, až v novembri 2019, sa v OpenAI po sérii výskumov rozhodli zverejniť aj najväčší model, GPT-2 XL (OpenAI, 2019, Nov 5). Model sa rýchlo stal hitom, pretože dokáže generovať nielen dezinformačné články, ale aj všelijaké druhy literárnych žánrov od príbehov cez básne až po divadelné hry [1] (Vincent, 2019).

Model GPT-2 XL je stále využívaný aj teraz, napriek tomu, že už existuje nový model GPT-3, ktorý je údajne ešte lepší. Tento model má vyše 175 miliárd parametrov (Brown a kol., 2020), čo je vyše stokrát viac ako najväčší model GPT-2. Veľa výskumníkov preto môže považovať už tak dosť veľký model GPT-2 XL za postačujúci pre ich experimenty.

Od vzniku a popularizácie Transformer modelov sa výskumníci pokúšali nájsť spôsob, ako skúmať ich „rozmýšľanie“. Na to, aby zistili, ako model rozmýšľa, skúmali, kam sa pri generovaní výstupu pozerá, na aké slová upriamuje pozornosť (angl. pays *attention*) – sledovali Attention blok modelu. Attention blok pri generovaní výstupu dostane na vstupe slová, ktorým priradzuje váhy podľa toho, aké slová sú preň v momentálnom kontexte dôležité.

Abnar a Zuidema (2020) vo svojom článku premýšľajú nad tým, na aké slová sa jednotlivé hlavy *Attention* zameriavajú a akú týmto slovám prideľujú váhu cez podľa nich jednoduché, ale efektívne metódy. Sami ale priznávajú, že sa pre tieto výpočty robí množstvo zjednodušených predpokladov.

Ďalším problémom, s ktorým sa výskumníci stretli, je, ako prehľadne *ukázať*, na aké slová sa jednotlivé hlavy *attention* zameriavajú. Takouto vizualizáciou sa zaoberal aj Vig (2019) vo svojom článku *A Multiscale Visualization of Attention in the Transformer Model*.

GPT-2 je len jeden z väčšieho počtu jazykových modelov založených na architektúre typu Transformer a v oboch spomínaných článkoch sa výskumníci pokúšali o všeobecné riešenie pre všetky modely architektúry Transformer.

My sa v tejto práci zameriavame len na modely GPT-2. Vytvorili sme aplikáciu, ktorá je schopná po vybraní správneho modelu a nastavení parametrov jednoduchým a prehľadným spôsobom ukázať slová, ktoré majú na generovanie nasledujúceho slova v GPT-2 najväčší vplyv.

Na rozdiel od vyššie spomínaných prác, my sme sa pozreli na tieto problémy z iného uhla pohľadu. Nepozeráme sa na samotné bloky *Attention*, zaujíma nás len výstup samotného modelu (konkrétne výstupné pravdepodobnosti pre jednotlivé

[1] Tímu THEaiTRE sa pomocou GPT-2 podarilo vygenerovať kompletnú divadelnú hru, ktorá mala aj pár predstavení na divadelných doskách (`https://www.theaitre.com/`).

tokeny).

Veríme, že takýto pohľad na dáta je dostatočne jednoduchý pre ľudského analytika a dokáže do skúmania textov vygenerovaných GPT-2 modelom vniesť nové svetlo.

Pre príklad si môžeme predstaviť, že GPT-2 model pri generovaní vygeneroval tento text: *"It wasn't me, it was John who fought the bear. Now there's a group of bears and we're not trying to get rid of them. We're trying to make sure we have wildlife areas like the park."*

Nás by zaujímalo, prečo tento text vygeneroval slovo *„wildlife"* (voľne žijúca zver). Proces generovania tohto textu si môžeme nasimulovať v našom programe.

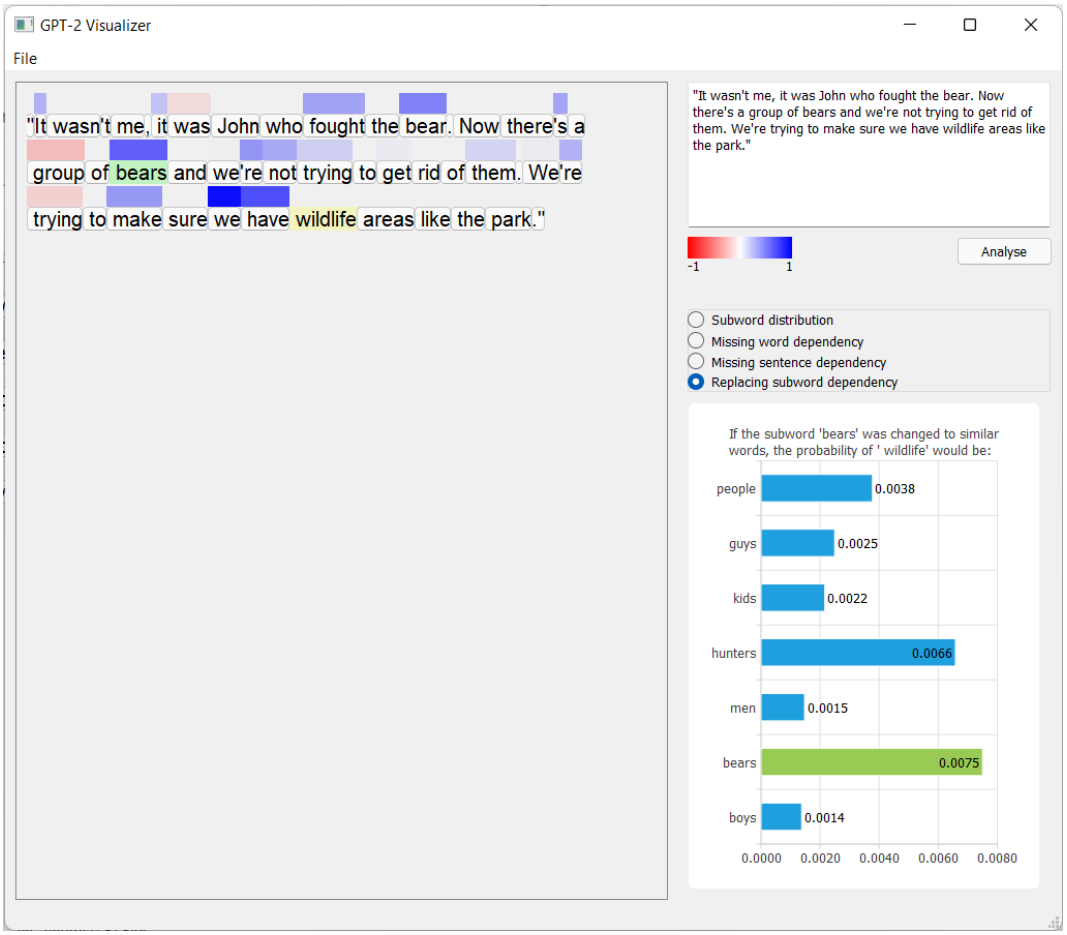
Na obrázku 1 vidíme jeden z našich použitých módov, konkrétne Replacing word dependency (všetky naše módy sú popísané v kapitole Metódy 2). V tomto móde sa pokúšame analyzovať, na akých slovách je slovo „wildlife" závislé (intenzitou farieb nad danými slovami).

Vidíme, že „wildlife" je najviac závislé na predchádzajúcich slovách, ale inak veľkú dôležitosť zohráva aj slovo *„bears"* (medvede). To dáva zmysel, pretože medvede patria do oblastí s voľne žijúcou zverou a v texte sa spomína: „... nechceme sa zbaviť medveďov, snažíme sa mať nejaké oblasti s voľne žijúcou zverou ako tento park.".

Samotné slovo „wildlife" malo pri generovaní pravdepodobnosť 0,75 %, čo nie je veľa a model si takto vylosoval menej pravdepodobné slovo. To je dôvod, prečo sa môže zdať, že závislosť „wildlife" od „bears" nie je až taká silná. Na grafe vpravo dole vidíme tzv. *podobné* slová a akú pravdepodobnosť by malo „wildlife", ak by bolo slovo „bears" nahradené nejakým z týchto slov (podobné slová sú popísané v sekcii 2.4).

V kapitole 1 (Základné pojmy) si predstavíme základnú teóriu za touto prácou, zároveň si podrobnejšie predstavíme články a práce, ktoré sa venovali analýze alebo vizualiácii GPT-2 modelov alebo Transformer modelov vo všeobecnosti. Kapitola 2 (Metódy) nám predstaví naše nové metódy práce s dátami a ako hľadáme závislosti medzi jednotlivými slovami. V kapitole 3 (Popis programu) na vizualizáciu sa povenujeme teórii, hypotézam a samotným algoritmom použitým za jednotlivými časťami vizualizačnej aplikácie.

Úplná dokumentácia pre používateľa, vrátane konkrétnych príkladov a obrázkov sa nachádza v kapitole 4 (Používateľský manuál), ktorá má za cieľ zjednodušiť prácu s aplikáciou. Posledná kapitola 5 s názvom Analýza, je určená na to, aby sme si ukázali niektoré zaujímavé výsledky, ktoré sa nám podarilo dosiahnuť a vyslovili hypotézy, ktoré sa pokúsia vysvetliť tieto konkrétne správania GPT-2 modelov.



Obrázek 1: Text vygenerovaný GPT-2 modelom sledovaný v našom programe. Snažíme sa zistiť, prečo model vygeneroval slovo *„wildlife“* v móde Replacing subword dependency, graf generujeme pre slovo *„bears“*. Bližší popis tejto obrazovky nájdete v 4.2.1.

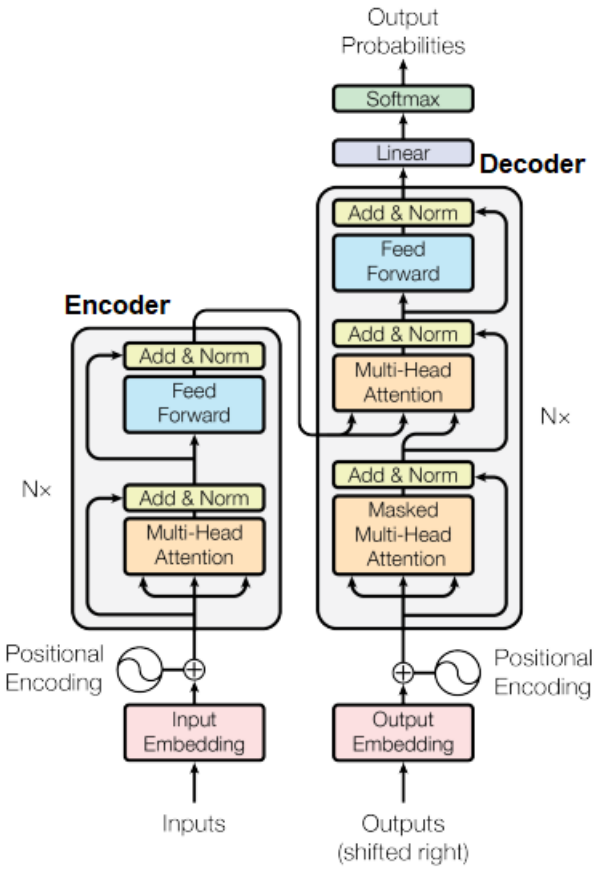
1. Základné pojmy

1.1 Transformer

GPT-2 je *model* neurónovej siete, ktorý je založený na architektúre Transformer.

Transformer je model neurónovej siete, ktorý bol ako prvý predstavený v článku *Attention is All you Need* (Vaswani a kol., 2017). Tento model dosiahol v tom čase najlepšie výsledky na prekladateľských úlohách z angličtiny do francúzštiny (*WMT 2014 English-to-French*) a z angličtiny do nemčiny (*WMT 2014 English-to-German*).

Architektúra tohto modelu je ukázaná na obrázku 1.1. Transformer sa skladá z dvoch hlavných častí, ktoré sa nazývajú *Encoder* (kódovač alebo kódovací blok) a *Decoder* (dekodér).



Obrázek 1.1: Architektúra modelu Transformer (Vaswani a kol., 2017). Navrhované riešenie bolo použiť zásobník $N = 6$ kódovacích a dekódovacích blokov postavených na sebe.

1.1.1 Attention

Hlavnou komponentou v architektúre Transformer je blok zvaný *Attention*. S myšlienkou Attention prišli Bahdanau a kol. (2014) ako k možnému vylepšeniu

sequence-to-sequence modelov určeným na prekladanie textov z jedného jazyka do druhého. Myšlienkou Attention je povoliť modelu vybrať si, na ktoré slová sa bude pri výpočtoch pozerať (dávať na ne pozor – angl. *pay attention*).

V sequence-to-sequence modeloch to pred týmto objavom fungovalo tak, že sa na začiatku spracovalo každé slovo samostatne pomocou Encoder blokov a celá informácia vopred daných rozmerov sa predávala Decoder bloku, ktorý začal postupne generovať slová na výstup. Pre niektoré dlhšie vety a texty potom mohol byť problém zakódovať informácie o všetkých slovách.

Prítomnosť Attention povolila týmto modelom kódovať si každé slovo do nejakej číselnej reprezentácie a následne povoliť Attention bloku vybrať si jednotlivé reprezentácie a z nich informácie, ktoré potrebuje pri výpočtoch.

Attention v transformeroch

Konkrétna implementácia Attention pracuje s troma druhmi vektorov: *query* (požiadavka), *key* (kľúč) a *value* (hodnota). Výstup z bloku sa počíta ako vážený súčet *values*, kde váhy sú určené kompatibilitou odpovedajúceho *key* a *query* (Vaswani a kol., 2017).

Pre každé slovo na vstupe sa najprv vypočíta dvojica *key-value* a pri iterácii výpočtu sa určí *query* ako požiadavka, ktorú má v tomto kroku model.

Query sa potom porovná s každým *key* a ako výsledok tohto porovnania dostane model skóre pre daný *key*. Pretože *query* a *key* sú vektory čísel rovnakej dĺžky, na výpočet skóre sa používa skalárny súčin.

Na tieto hodnoty sa potom aplikuje *softmax* funkcia, ktorá ich naškáluje ako hodnoty 0 – 1 tak, aby spolu dávali súčet rovný 1 a relatívne rozdiely medzi skóre ostali rovnaké. Tieto nové skóre predstavujú váhy, teda ako veľmi sa na danú *value* bude prihliadať pri výpočte výstupu.

Takýto výpočet Attention nazvali v článku ako *Scaled Dot-Product Attention* (škálovaný Attention zo skalárneho súčinu).

Takto by sa dal popísať výpočet jednej Attention hlavy v transformeri. Jeden Attention blok môže obsahovať týchto hláv viac, čo len pridáva všetkým týmto výpočtom a vektorom o dimenziu viac.

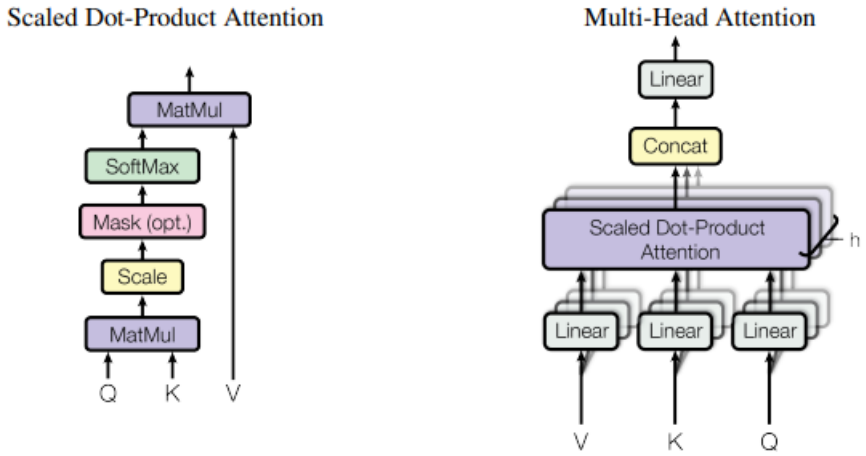
Ilustrácia výpočtu Attention v architektúre Transformer je znázornená na obrázku 1.2.

Self-Attention

Ako sme spomínali na začiatku sekcie, pôvodný Attention vznikol na riešenie prekladateľských úloh. Ten Attention potreboval údaje zo vstupu do modelu a aj z výstupu z modelu, aby vedel správne prekladať (musí vidieť nielen vstupné slová, ale aj aké výstupné slová sú zatiaľ vygenerované). Tento mechanizmus je rozdielny od Self-Attention mechanizmov, ktoré sa používajú v architektúre Transformer.

Self-Attention je mechanizmus Attention, ktorý sa vzťahuje na rôzne pozície jedného reťazca slov tak, aby vypočítal reprezentáciu tohto reťazca (Vaswani a kol., 2017). V prípade architektúry Transformer to znamená, že Self-Attention si pre každé slovo určí dvojicu *key-value* a následne v každej iterácii aj *query*.

Na obrázku 1.1 vidíme 3 rôzne Attention vrstvy: Attention v Encoder bloku, Attention v Decoder bloku dole a Attention v Decoder bloku, ktorý sa pozerá na výstup Encoder bloku. Prvé dva z nich sú Self-Attention, posledný z nich nie je.



Obrázek 1.2: Výpočet Attention v transformeri (Vaswani a kol., 2017). Ľavá časť ukazuje, ako prebieha výpočet Attention z query (Q), key (K) a value (V). Pravá časť načrtáva to, ako funguje attention s väčším množstvom hláv (h).

Dôvodom je to, že sa tento Attention pozerá na vstupné dáta pre Encoder blok a informáciu o tom, na čo sa má zameriavať dostáva zo vstupných dát pre Decoder blok. Táto Attention vrstva sa nazýva Encoder-Decoder Attention (Alammar, 2018).

1.1.2 Encoder

Encoder blok pracuje so vstupnými slovami do siete. Skladá sa z 2 vrstiev, prvou je Self-Attention vrstva a druhou časťou je klasická feed-forward (dopredná) neurónová sieť.

Okolo oboch týchto vrstiev sa nachádzajú reziduálne spoje, ktoré zaručujú, že sa pôvodná informácia vstupujúca do blokov dostane ďalej. Na konci nasleduje normalizačná funkcia s názvom Layer normalization (alebo LayerNorm).

Výstup potom nasleduje do nasledujúceho Encoder bloku, v prípade najvyššieho Encoder bloku je výstup poslaný do všetkých Decoder blokov.

1.1.3 Decoder

Vstupom do Decoder bloku je doterajší výstup modelu. Decoder blok sa skladá z rovnakých dvoch vrstiev ako Encoder blok. Medzi tieto dve vrstvy je navyše vložená nová vrstva, ktorá počíta Attention z výstupu Encoder časti.

Oproti Encoder vrstve nastala ešte jedna zmena. Vstupom do Decoder bloku je vždy vektor fixnej dĺžky vyplnený sprava prázdnymi slovami a prvý Attention blok je upravený tak, aby sa nepozeral na nasledujúce pozície. To je spravené tak, že pred *softmaxom* je skóre týchto pozícií napevno zapísané ako $-\infty$, aby ich *softmax* ohodnotil nulou.

1.1.4 Vstupný embedding

Vstupom do siete nie je priamo slovo alebo celá veta. Samotný text je rozdelený na jednotlivé tokeny (to môžu byť slová, časti slov alebo znaky), tie sú potom

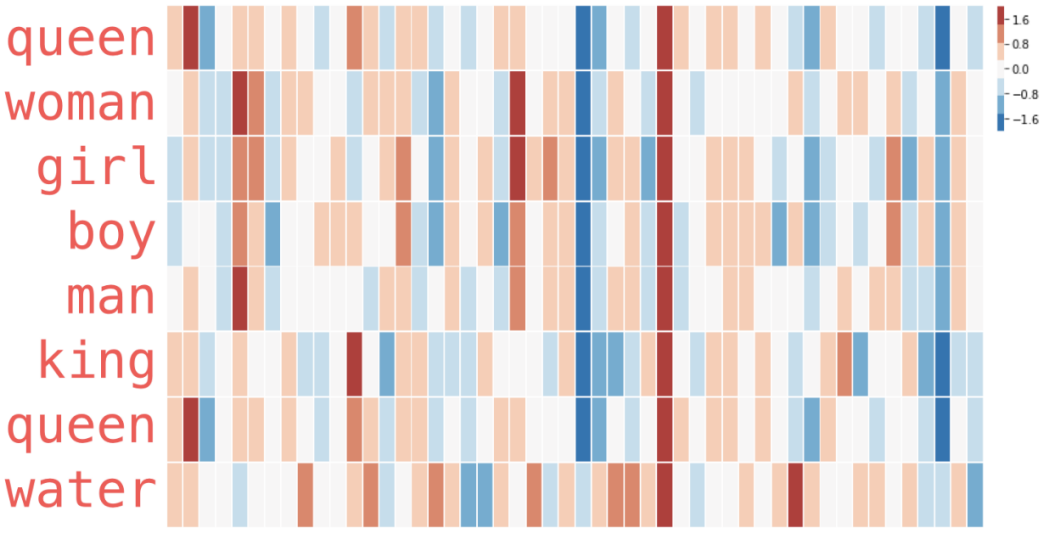
zakódované nejakým prirodzeným číslom (napr. svojím ID z kódovacej tabuľky). Na túto časť slúži kódovací algoritmus bližšie popísaný v sekcii 1.2.3.

Vstupný embedding je taký spôsob reprezentácie slov, ktorý umožňuje slovám s podobným významom mať veľmi podobnú reprezentáciu. Princíp je taký, že každé slovo bude reprezentované vektorom viacerých čísel. Konkrétnu reprezentáciu sa model siete naučí sám pri trénovaní (Alammar, 2019).

Vstupom do modelu ale nie je priamo kód daného tokenu, tento kód ešte prejde *one-hot kódovaním*, ktoré pripraví vstup do siete. One-hot kódovanie pripraví vektor o veľkosti slovníka plný núl a na príslušné miesto napíše jednotku (ak je teda slovník s veľkosťou 6 a chceme zakódovať číslo 3, one-hot nám vráti *[0 0 0 1 0 0]*).

Myšlienkovo je vstupný embedding tabuľka, ktorá každému číslu priradí jeho osobný vektor hodnôt. Technicky, to je matica s rozmermi $s \times d$, kde „s“ je veľkosť kódovacieho slovníka a „d“ je požadovaná veľkosť vstupu (dimenzionalita modelu). V prípade GPT-2 XL modelu to je 50257×1600. Daný token teda bude ohodnotený 1600 rôznymi črtami, ktoré sa model naučil pre dané slovo pri tréningu.

Príklad si môžeme pozrieť na obrázku 1.3. Všetky slová sú podstatné mená a vidíme, že majú veľmi podobnú jednu zložku približne v strede (reprezentovanú tmavočervenou farbou). Slová ako *boy* (chlapec) a *girl* (dievča) majú podobné niektoré úseky, zase slová ako *girl* a *woman* (žena) sú podobné v iných častiach.



Obrázek 1.3: Príklad vstupného slovného embeddingu niekoľkých anglických slov aj s legendou daných farieb (Alammar, 2019). Je to príklad GloVe (Pennington a kol., 2014) vektoru trénovaného na Wikipedii.

1.1.5 Pozičné kódovanie

Pozičné kódovanie, ktoré sa v pôvodnej architektúre ilustrovanej na obrázku 1.1 pripočítava k vstupnému embeddingu, je ďalšia dôležitá časť architekúry Transformer.

Transformer model nepoužíva žiadnu rekurenciu ani konvolúciu, či iné spôsoby, ktorými vie brať do úvahy poradie jednotlivých vstupných tokenov. Bez

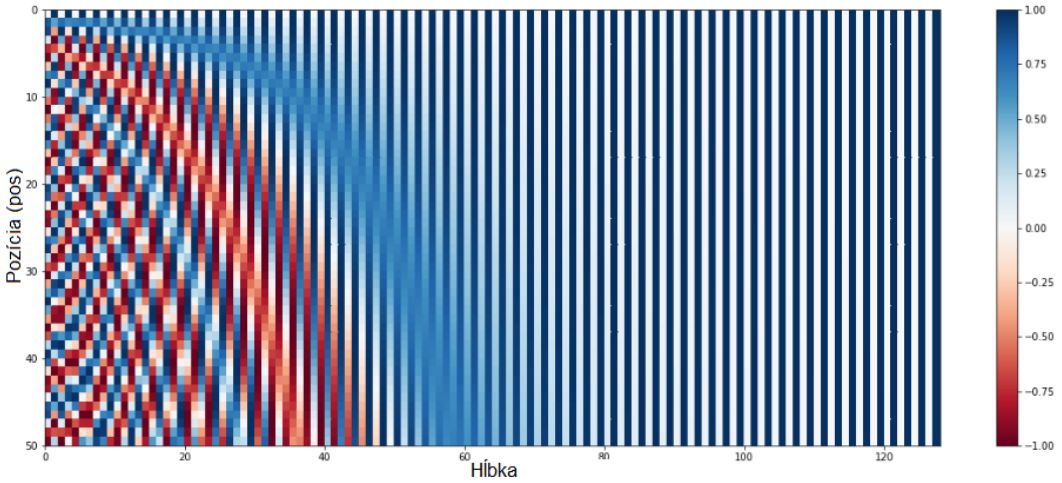
reprezentácie pozície by na poradí slov vo vete nezáviselo. Preto Vaswani a kol. (2017) vymysleli pozičné kódovanie, ktorý vytvára unikátny vektor pre každú pozíciu.

Vzorec používa sínus pre párne a kosínus pre nepárne pozície. Funkcia na výpočet pre slovo na pozícii *pos* pre *i*-tú zložku vyzerá takto (parameter d_{model} predstavuje dimenzionalitu modelu):

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

Na obrázku 1.4 môžeme vidieť, ako vyzerá pozičné kódovanie pre prvých 50 pozícií a $d_{model} = 128$. Táto funkcia je špecificky zvolená tak, aby sa model mohol naučiť pozerať na relatívnu pozíciu, pretože každý pevný posun k je PE_{pos+k} lineárnou funkciou PE_{pos}.



Obrázek 1.4: Pozičné kódovanie pre $d_{model} = 128$. Každý riadok reprezentuje kód pre danú pozíciu (Kazemnejad, 2019).

1.1.6 Výstup

Výstupom z poslednej Decoder vrstvy je jedna plne prepojená vrstva neurónov, ktorej výstupom je *skóre* pre jednotlivé tokeny. Tie sa ďalej dostanú do *softmax* vrstvy, ktorá prepočíta skóre na pravdepodobnosť pri výbere každého jednotlivého tokenu. To je výstupom jednej iterácie architekúry Transformer.

Tento výstup sa presunie naspäť na vstup Decoder vrstvy na generovanie ďalšieho tokenu.

1.2 GPT-2

GPT je skratka pre Generatívny predtrénovaný transformer (v anglickom originál *Generative Pre-trained Transformer*). Za začiatok tohto transformer modelu môžeme považovať rok 2017, kedy sa výskumníci Radford a kol. (2017) pokúšali o vytvorenie rekurentnej neurónovej siete určenej na generovanie recenzií.

Použili na to druh rekurentnej neurónovej siete s názvom mLSTM (*multiplicative long-short term memory*) so 4096 neurónmi. Tento model trénovali učením bez učiteľa (unsupervised learning) na veľkom množstve dát (82 miliónov recenzií produktov na Amazone). Zistili, že pri trénovaní s L1 regularizáciou model používal prekvapivo malý počet neurónov. Po hlbšom skúmaní zistili, že jedna takáto jednotka – neurón vysoko koreluje so sentimentom (citovým zafarbením) daného textu. Nazvali ju *sentiment neuron* (neurón sentimentu). Vďaka tomuto prístupu dosiahli 91,8 % úspešnosť na datasete Stanford Sentiment Treebank (predchádzajúca najlepšia úspešnosť bola 90,2 %).

Po tomto úspechu sa zamerali na to, ako použiť tento *sentiment neuron* na lepšie generovanie textov (Radford a kol., 2018). Použili rovnaký princíp *sentiment neuronu* v spojení s architektúrou Transformer a vznikol prvý GPT model.

GPT-2 model vznikol následne o rok na to s množstvom vylepšení (ako napríklad v oblasti trénovacích dát) a väčšou kapacitou siete (z vyše 117 miliónov na vyše 1,5 miliardy (Shree, 2020)). Jednou zo zmien bola aj maximálna dĺžka generovaného textu, GPT-2 dokáže vygenerovať maximálne 1024 subwordov v jednom generovaní (GPT dokázal takto vygenerovať maximálne 512 subwordov). Táto maximálna dĺžka sa počíta vrátane vstupného textu, pretože (ako bude vysvetlené v 1.2.1) pre GPT-2 je vstup a výstup rovnaké miesto.

1.2.1 Transformer-Decoder

GPT-2 model je založený na architekúre zvanej *Transformer-Decoder*, ktorá z klasickej architektúry Transformer vyškrtla Encoder časť.

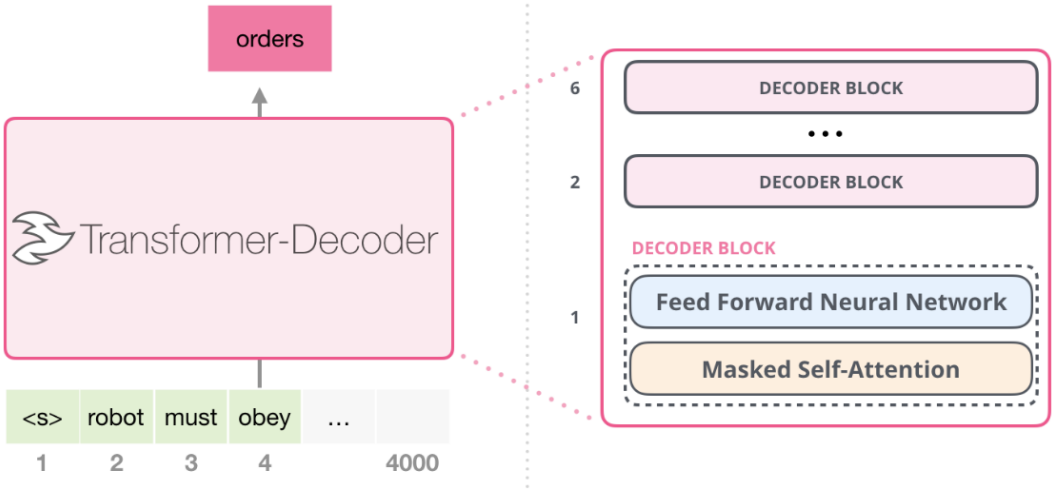
Myšlienkovo to dáva zmysel, pôvodný Transformer model bol vymyslený a trénovaný na prekladateľských úlohách. Na samotný vstup modelu prichádza text v jednom jazyku a výstupom je text v druhom jazyku, samotný vstup prichádza priamo do Encoder blokov a výstup sa generuje v Decoder blokoch za pomoci dát z Encoderu a už vopred vygenerovaných slov, teda slov, ktoré už sú na výstupe. Pre daný kontext prekladania z jazyku do jazyku sú vstupný a aj doterajší výstupný text rovnako dôležité.

V GPT-2 je úlohou predpovedať nasledujúci text pri danom kontexte, teda generovať ďalšie slová. Pri generovaní textu sú slová na vstupe a doteraz vygenerované slová rovnaké, pretože výstup siete sa spätne doplní ako vstup pre nasledujúci cyklus generovania. Tento princíp sa nazýva auto-regresia (angl. auto-regression). Preto sa už v prvej verzii GPT rozhodli Radford a kol. (2018) použiť Transformer-Decoder variantu transformeru.

Transformer-Decoder sa skladá z istého počtu Decoder blokov, podobne ako sú popísané v 1.1.3. Jediným rozdielom je, že druhá Attention vrstva je vynechaná (nakoľko tá pôvodná pracovala s výstupom Encoder vrstiev). Ilustrácia tejto architekúry je zobrazená na obrázku 1.5.

1.2.2 Trénovacie dáta

GPT-2 je vylepšením GPT. GPT-2 bol trénovaný, aby predikoval nasledujúce slovo v texte. Ako trénovacie dáta sa použil v tom čase nový trénovací dataset, ktorý nazvali *WebText* (OpenAI, 2019, Feb 14). Tento dataset je zložený z textu z webových stránok. Kvalitu datasetu zabezpečili tak, že pustili nástroj zvaný



Obrázek 1.5: Ilustrácia *Transformer-Decoder* architektúry, všetky *Decoder* bloky majú rovnaké časti, rozobratý je len prvý z nich (Alammar, 2019, Aug 12).

scraper (určený na prechádzanie webových stránok a vyťahovaní potrebných informácií na sociálnu sieť Reddit a prechádzali len webové adresy, ktoré dostali aspoň 3 karma body).

1.2.3 Kódovanie vstupného textu

Ako je popísané v sekcii 1.3, slová do GPT-2 modelu nejdú v textovej podobe. Text býva zakódovaný do čísel a model sa potom naučí pre každé číslo jeho vstupný embedding.

Ako kódovanie sa používa algoritmus Byte-Pair Encoding (BPE). Tento algoritmus patrí medzi tokenizačné algoritmy založené na podslovách [1] (*subword-based tokenization algorithm*).

Tokenizačné algoritmy môžu byť založené na slovách, znakoch alebo subwordoch. Algoritmy založené na znakoch majú problém pri prenášaní významu, pretože každý zo základných znakov sa vyskytuje vo veľkom počte rozličných slov. Algoritmy založené na slovách majú zase problém v robustnosti. Jednotlivé jazyky majú veľké množstvá slov a nové stále pribúdajú. Môže byť celkom nepraktické pracovať s tak veľkým slovníkom a aj tak môžu niektoré slová chýbať (Khanna, 2021).

Tie založené na subwordoch majú výhodu v tom, že dokážu zakódovať veľké množstvo najčastejších celých slov a najčastejšie časti slov. Myšlienka za tým je, že tieto najčastejšie časti slov môžu obsahovať aspoň nejaké znalosti o danom slove (napríklad v slovenčine prípona „-ejší“, ktorá sa nachádza na konci prídavných mien druhého a tretieho stupňa).

Byte-Pair encoding

Byte-Pair encoding (*BPE*) funguje tak, že si algoritmus preskúma všetky dáta (celý korpus). Za posledný znak každého slova pridá nový znak, ktorý nepatrí do

[1] Síce je slovo „podslovo“ správne, vo zvyšku práce budeme používať anglický termín *subword*.

abecedy (znak pre koniec slova), označme ho „</w>“ (interpunkčné znamienka sú tiež považované za slovo a tiež sa za ne vkladá tento znak). Z každého jednotlivého znaku si spraví záznam do slovníka (aj s jeho početnosťou). Tieto záznamy v slovníku prehlásime za subwordy [2].

Nasleduje iteračný krok, ktorý sa bude vykonávať, kým sa nesplní zastavujúca podmienka, napríklad celkový počet subwordov. Iteračný krok prebieha tak, že sa nájde najfrekventovanejší pár dvoch po sebe idúcich subwordov, ktorý sa v slovníku ešte nenachádza, založí sa do slovníka a upraví sa frekvencia dvoch subwordov tvoriacich tento pár (Khanna, 2021).

Príklad

Ako príklad pre BPE si môžeme predstaviť korpus, ktorý obsahuje len tieto slová a početnosti:

`{"old": 5, "older": 3, "fool": 2}`.

Na konci každého slova pridáme *„</w>“* a spravíme si tabuľku individuálnych znakov. Tá by vyzerala podobne tabuľke 1.1

ID	Subword	Frekvencia
1	</w>	10
2	o	12
3	l	10
4	d	8
5	e	3
6	r	3
7	f	2

Tabulka 1.1: Tabuľka na začiatku algoritmu Byte-Pair encoding. Algoritmus by v tomto momente zakódoval slová nasledovne: „old“ na [2, 3, 4, 1], „older“ na [2, 3, 4, 5, 6, 1], „fool“ na [7, 2, 2, 3, 1].

Najčastejšie používaná dvojica subwordov z nášho korpusu je *„o“* a *„l“*, ktorá sa vyskytuje desaťkrát. Takže si pridáme do tabuľky nový subword *„ol“* s frekvenciou 10, následne pre subwordy *„o“* a *„l“* odčítame 10 výskytov. Tie už máme pokryté novým subwordom. Frekvencia subwordu *„l“* dosiahla nulu, tento subword môžeme z tabuľky odstrániť. Po prvej iterácii by teda tabuľka vyzerala ako tabuľka 1.2.

V ďalšom kroku by sme si vybrali *„ol“* a *„d“* a založili subword *„old“*. Takto by sme využili všetky výskyty *„d“*, mohli by ho z tabuľky zmazať. Nasledovali by konce slov, teda spojenia so znakom „</w>“. Najprv *„old</w>“*, potom *„r</w>“* a následne *„ol</w>“*. Po pár iteráciách by naša tabuľka vyzerala ako tabuľka 1.3. V popise tabuľky sa môžeme pozrieť, ako sa zakóduje náš korpus.

Pôvodné slová obsahujú 38 znakov, na začiatku algoritmu máme pre každý znak samostatný záznam a záznamov máme 7. Po pár iteráciách algoritmu BPE nám ostalo 7 záznamov v tabuľke, náš korpus sme ale zakódovali len na 20 znakov. Celý tento príklad je inšpirovaný príkladom od Khanna (2021).

[2] BPE si text rozkúskuje podľa znakov </w> a k jednotlivému slovu sa považuje aj prípadná medzera pred týmto slovom. Takto si vieme oddeliť subwordy, ktoré sú často na začiatku slova od takých, ktoré sú uprostred

ID	Subword	Frekvencia
1	</w>	10
2	o	2
3	d	8
4	e	3
5	r	3
6	f	2
7	ol	10

Tabulka 1.2: Tabuľka po prvom kroku algoritmu Byte-Pair encoding. Algoritmus by v tomto momente zakódoval slová nasledovne: „old“ na [7, 3, 1], „older“ na [7, 3, 4, 5, 1], „fool“ na [6, 2, 7, 1].

ID	Subword	Frekvencia
1	o	2
2	e	3
3	f	2
4	old	3
5	old</w>	5
6	r</w>	3
7	ol</w>	2

Tabulka 1.3: Tabuľka po pár krokoch algoritmu Byte-Pair encoding. Algoritmus by v tomto momente zakódoval slová nasledovne: „old“ na [5], „older“ na [4, 2, 6], „fool“ na [3, 1, 7].

BPE môže ďalej pokračovať až kým nevytvorí tabuľku, ktorá obsahuje iba samotné slová. Takýto príklad zhruba ukazuje, ako by tabuľka vyzerala, keď je viac slov v texte ako individuálnych znakov, napríklad vyše 400 000 slov a chceme udržať tabuľku na prijateľnej hodnote subwordov (napríklad okolo 50 000).

Kódovanie a dekódovanie

Kódovanie textu prebieha tzv. *„greedy“* (chamtivým) spôsobom. Máme pred sebou nejaký text a vezmeme najdlhší subword, ktorý sa celý vyskytuje od začiatku daného textu, a text nahradíme poradovým číslom v tabuľke subwordov.

Pre príklad si môžeme predstaviť slovné spojenie *„old or fool“* a tabuľku subwordov ukázanú v tabuľke 1.3. Najprv za každé slovo vložíme špeciálny znak konca slova (označme ho ako </w>), čím dostaneme text *„old</w> or</w> fool</w>“*.

Začneme na prvom slove, vidíme, že celé toto slovo sa nachádza v tabuľke, zapíšeme si teda číslo 5 a pokračujeme na slove *„or</w>“* [3], to sa tam samostatne nenachádza, ale nachádza sa tam subword *„o“*, zapíšeme si jeho ID (1) a pokračujeme pre zvyšok slova, to sa tam celé nachádza ako ID 6. Slovo *„fool</w>“* si vieme vyskladať zo subwordov *„f“*, *„o“* a *„ol</w>“*.

Text *„old or fool“* teda vieme zakódovať na sériu znakov *[5, 1, 6, 3, 1, 7]*.

[3] V skutočnosti by sme mali slovo „␣or</w>", pretože sa za slovo považuje aj prípadná medzera pred ním, tú ale v dátach z príkladu kvôli jednoduchosti nemáme.

Dekódovanie textu je následne jednoduché. Máme poradové čísla v tabuľke a jednoducho ich nahrádzame subwordmi, keď uvidíme znak „*</w>*“, vynecháme ho a spravíme vo výslednom texte medzeru[4].

Z predchádzajúceho príkladu textu dostaneme zo zakódovanej série *[5, 1, 6, 3, 1, 7]* priamo slová *„old or fool“*.

1.3 Aplikácie GPT-2

GPT-2 patrí medzi jazykové modely neurónových sietí. To znamená, že (aspoň navonok) to vyzerá, že textu rozumie. Využitia takéhoto modelu sú teda limitované na spracovanie a prácu s textom.

1.3.1 Generovanie textu

Generovanie textu je základné použitie GPT-2. Ako sme spomínali v sekcii 1.2, tieto siete boli historicky vytvorené na generovanie textu, konkrétne krátkych recenzií na produkty kúpené cez internetové obchody.

GPT-2 sa ale dá použiť na oveľa viac ako generovanie recenzií. Na základe vstupného textu (*promptu*) dokáže GPT-2 odhadnúť žáner a prispôsobiť podľa toho aj štýl, ktorým generuje nasledujúci text (OpenAI majú aj online nástroj[5], kde to je možné otestovať).

Branwen (2019) vo svojom blogu popisujú, ako použili *fine-tuning* GPT-2 na generovanie básní.

Rosa a kol. (2021) sa podarilo takýmto spôsobom dokonca vygenerovať celú divadelnú hru, ktorá bola nakoniec aj hraná v divadle. Dramaturg tejto divadelnej hry nechal model generovať pár dialógov a následne sa rozhodol, či a koľko z generovanej časti tam ostane. Takto bola vygenerovaná celá hra a následne mohli nastať ešte ďalšie drobné úpravy. Celkovo v prvej scéne zostalo 91 % všetkých slov presne tak, ako boli vygenerované a zvyšných 9 % slov bolo pridaných, zmenených alebo prehodených.

1.3.2 Chatbot

Pred vzostupom techník na spracovanie prirodzeného jazyka na základe hlbokého učenia trvalo mesiace nastaviť pravidlá a konverzačné témy pre chatbotov. S príchodom modelov ako GPT-2 to je v dnešnej dobe možné v priebehu niekoľkých dní (Saini, 2021).

Jedno z takýchto využití predviedli Wolf (2021). V článku popisujú, ako sa im podarilo postaviť takéhoto chatbota, ktorého demo dokonca zverejnili[6].

[4] V našom prípade pri práci s implementáciou GPT-2 tokenizeru od Hugging Face by sa medzera nachádzala tiež v tabuľke subwordov (pripojená k slovu za ňou, čím vieme, že tento subword môže byť na začiatku slova), takže nie je potrebné tam medzery pridávať (`https://huggingface.co/docs/transformers/model_doc/gpt2#transformers.GPT2Tokenizer`)

[5] `https://app.inferkit.com/demo`

[6] https://convai.huggingface.co/

1.3.3 Preklad

Využitiu GPT-2 na preklad sa venovali Radford a kol. (2019) už v pôvodnom článku o GPT-2. Aj napriek tomu, že pred trénovaním vymazali z WebTexu (trénovaciemu datasetu pre GPT-2) webové stránky, ktoré neboli anglické, dosiahla GPT-2 sieť na prekladových úlohách z angličtiny do francúzštiny celkom zaujímavé výsledky. Pokorili pár *baselines* pre strojový preklad bez učiteľa.

Nerobili žiadny *fine-tuning* na testovanom modeli, aby vedel, že sa jedná o preklad, jednoducho ako vstup do siete zadali pár dvojíc *anglická veta = francúzska veta* a na konci vstupu dali iba anglickú vetu a znak „=“ (*anglická veta =*).

1.3.4 Sumarizácia textu

Podobne ako pri preklade, o sumarizáciu textu pomocou GPT-2 sa pokúšali už Radford a kol. (2019). Tento pokus však nedopadol veľmi úspešne. Model sa často zameriaval na posledné vety a niekedy doplietol nejaké presné skutočnosti, ako napríklad koľko áut bolo účastníkom dopravnej nehody.

O vylepšenie sa pokúšali Ziegler a kol. (2019), ktorý použili techniku *fine-tuningu*, aby model lepšie pripravili na sumarizačné úlohy. Zistili, že model síce dosahuje celkom kvalitné výsledky, ale vo väčšine času (65 % – 98 %) modely len kopírujú celé vety. Hovoria o tom, že kopírovanie textu je v tomto prípade lepšie ako sumáre, ktoré generujú modely s učiteľom (*supervised*), ktoré sú prirodzene a hodnoverne vyzerajúce, ale často klamlivé.

1.4 Vizualizácia transformerov

Transformery sa stali od svojho vzniku celkom populárne nielen vďaka sérii GPT modelov, ale aj vďaka modelu BERT, ktorý zbúral rekordy v rôznych úlohách na spracovanie prirodzeného jazyka (Devlin a kol., 2018). Oba tieto modely (ako aj niektoré väčšie derivácie modelu BERT ako RoBERTa (Liu a kol., 2019)) majú stovky miliónov parametrov, ktoré je možné trénovať (BERT má 340 miliónov parametrov, RoBERTa 355 miliónov a GPT-2 dokonca 1,5 miliardy).

Ani pre odborníkov nemusí byť jednoduché len z číselných dát skúmať, ako tieto modely vo všeobecnosti „premýšľajú“, akým slovám dávajú pri výpočtoch vyššiu váhu ako ostatným. Vizualizácia číselných dát by v tomto mohla pomôcť.

Transformer model sa vnútorne skladá z blokov (Encoder alebo Decoder), v ktorých najzložitejšia vrstva je Attention. Vizualizácie Attention blokov sa začali robiť pomocou techniky zvanej *heatmap* (tepelná mapa) (Rocktäschel a kol., 2015) alebo bipartitných grafov (Strobelt a kol., 2018). Tieto vizualizácie ale pracovali na modeloch, ktoré používali len jeden Attention.

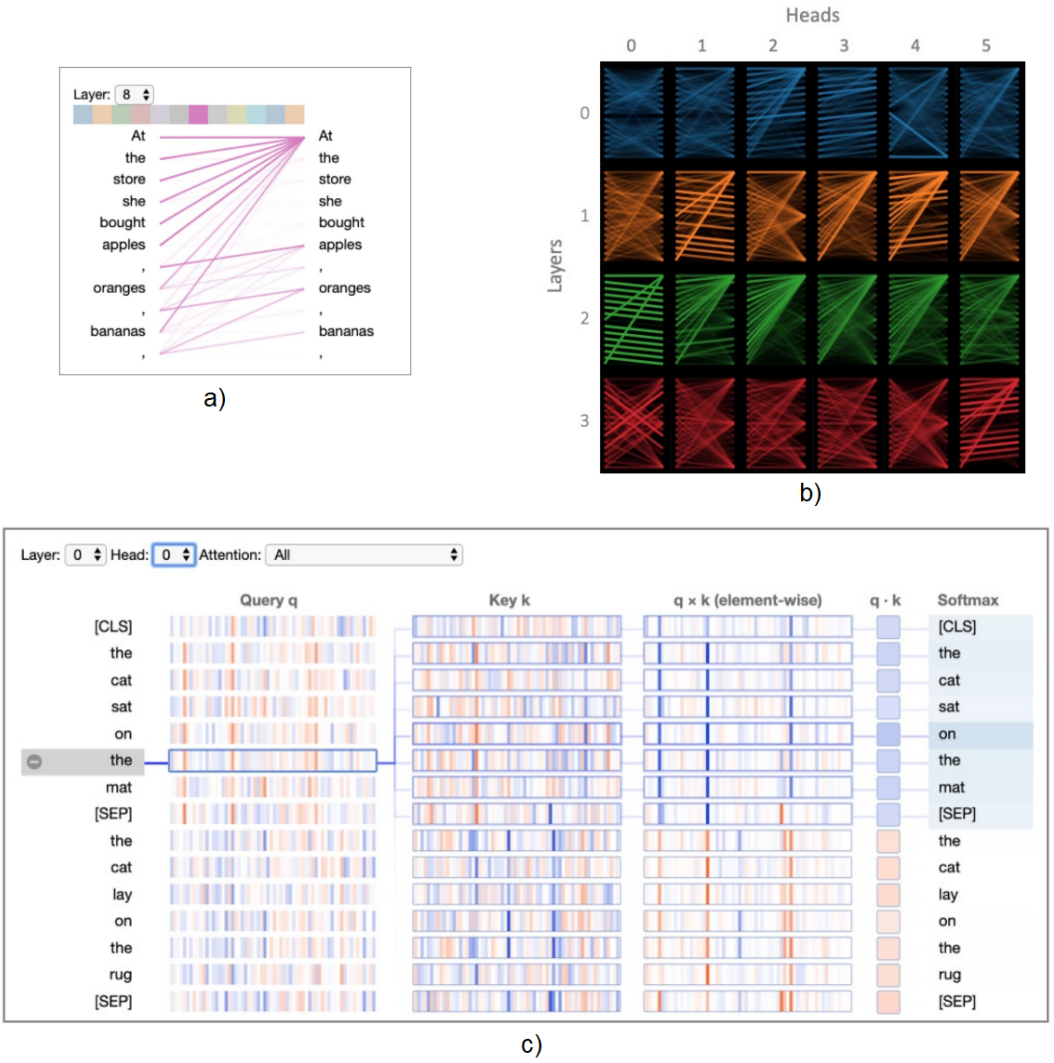
Doteraz asi najlepší nástroj na vizualizáciu transformerov vytvoril Vig (2019). V nástroji sa zameriava na vizualizáciu všetkých Attention hláv cez všetky bloky. Pre malý GPT-2 Small model to je 12 Attention hláv v 12 Decoder blokoch, dokopy teda 144 Attention mechanizmov, ktoré sa vizualizujú.

Nástroj obsahuje 3 rôzne pohľady: *Attention-Head View*, *Model View* a *Neuron View*. Prvé dva ukazujú závislosti medzi slovami vo forme bipartitných grafov,

Neuron View ukazuje konkrétne počítanie Attention v jednej hlave (konkrétne, pre každé slovo ukazuje *query* a *key* a ako vyzerá ich skalárny súčin).

Attention-Head view zobrazuje, akým slovám dáva Attention vyššiu pozornosť, ak sa venuje nejakému slovu. Toto zobrazenie prebieha po jednotlivých blokoch (či už Decoder (GPT-2) alebo Encoder (BERT)), kde je možné vybrať, ktoré Attention hlavy skúmame.

Model View vie zobraziť Attention-Head View pre všetky Attention mechanizmy naraz v podobe mozaiky. Ukážky všetkých pohľadov je možné vidieť na obrázku 1.6.



Obrázek 1.6: Ukážka rôznych vizualizácií, ktoré vytvoril Vig (2019) vo svojej práci: Attention-Head View (a), Model View (b) a Neuron View (c). Model View ukazuje len bloky (vrstvy) 0 – 3 a Attention hlavy 0 – 5.

1.5 Nástroj pre morfologickú analýzu

Vo vizualizačnom programe používame v niektorých úsekoch online nástroj pre morfologickú analýzu textu.

Nástroj, ktorý sme zvolili, sa volá MorphoDiTa[7]. Celý názov nástroja je Morphological Dictionary and Tagger (Morfologický slovník a značkovač). Je to open-source nástroj na morfologickú analýzu prirodzených jazykových textov, ktorý dokáže robiť morfologickú analýzu, generovanie, značkovanie a tokenizáciu (rozdelenie vety na jednotlivé slová a interpunkciu).

MorphoDiTa je celkom intuitívny nástroj, ktorý dokáže všetko to, čo v programe potrebujeme. Konkrétne v programe využívame tokenizáciu na delenie textu na jednotlivé vety (viac v 2.3) a značkovanie (angl. *tagging*) na získanie morfologických značiek ku slovám vo vetách (viac v 2.4).

Na komunikáciu s týmto nástrojom využívame REST API nástroja, použitý model je vždy ten najnovší anglický model.

Tým, že používame anglický model, tak aj morfologické značky pre jednotlivé slová (*tagy*) sú anglické. Popis anglických modelov[8] uvádza, že to sú *Part-of-Speech tags* natrénované na korpuse Wall Street Journal (WSJ).

[7] `http://lindat.mff.cuni.cz/services/morphodita/run.php`

[8] `https://lindat.cz/repository/xmlui/handle/11858/00-097C-0000-0023-68D9-0`

2. Metódy vizualizácie

Na rozdiel od iných prác popísaných skôr (v sekcii 1.4), ktoré sa zaoberajú rozoberaním modelov a sledovaním toho, čo presne a s akou váhou sleduje každá *attention* hlava, my sme sa rozhodli pozrieť sa na veci inak, jednoduchšie a graficky prívetivejšie.

Tento vizualizátor sa pozerá na celé počítanie len z hľadiska *prvotných pravdepodobností* jednotlivých subwordov v každom kroku generovania. *Prvotnou pravdepodobnosťou* sa myslí výstup z poslednej vrstvy GPT-2, ktorou je *softmax*. Ten sa stará o to, aby výstupy z predošlej vrstvy, *skóre*, boli naškálované tak, aby mali všetky hodnotu od 0 po 1.

Ak je nastavený parameter modelu *Top k* (parametre sú popísané v kapitole 4.2.2) na hodnotu väčšiu ako 0, tak model vyberie k slov s najvyššími prvotnými pravdepodobnosťami a naškáluje ich znovu tak, aby to spĺňalo vlastnosti pravdepodobnostnej distribúcie. Nazvime to *reálna pravdepodobnosť* pre generovaný subword.

Jednotlivé závislosti generovaných subwordov na predchádzajúcich subwordoch v texte (či už tým textom je vstup programu alebo je vygenerovaný modelom siete externe) sme sa rozhodli sledovať 4 rôznymi spôsobmi, ktoré predstavujú 4 módy vizualizátoru. Sledujeme:

- *subword distribution*, ktorý zaznamená distribúciu 10 najpravdepodobnejších subwordov v každom kroku generovania;
- *Missing word dependency*, ktorý sleduje, ako by sa zmenila pravdepodobnosť vygenerovaného subwordu, ak by nejaký subword v texte chýbal;
- *missing sentence dependency*, ktorý zisťuje zmenu v pravdepodobnosti generovania daného subwordu, ak je z textu vynechaná nejaká konkrétna veta;
- *replacing subword dependency*, ktorý sleduje, ako by sa zmenila pravdepodobnosť pre generovanie daného subwordu, ak by bolo nejaké slovo v texte nahradené podobným slovom, čo sa týka gramatických kategórií.

Cieľom práce je vytvoriť vizualizačný nástroj, ktorý nám pomôže zistiť, ako rozmýšľa model GPT-2, teda prečo model vygeneroval dané slovo a na akých predchádzajúcich slovách to slovo najviac závisí. Vytvorenie týchto módov nám umožňuje sledovať, ako model pracuje so slovami v texte a aký vplyv naň majú dané slová (alebo vety) pri generovaní ďalších tokenov. V tejto kapitole si rozoberieme jednotlivé módy a demonštrujeme ich fungovanie na konkrétnych príkladoch.

2.1 Subword distribution

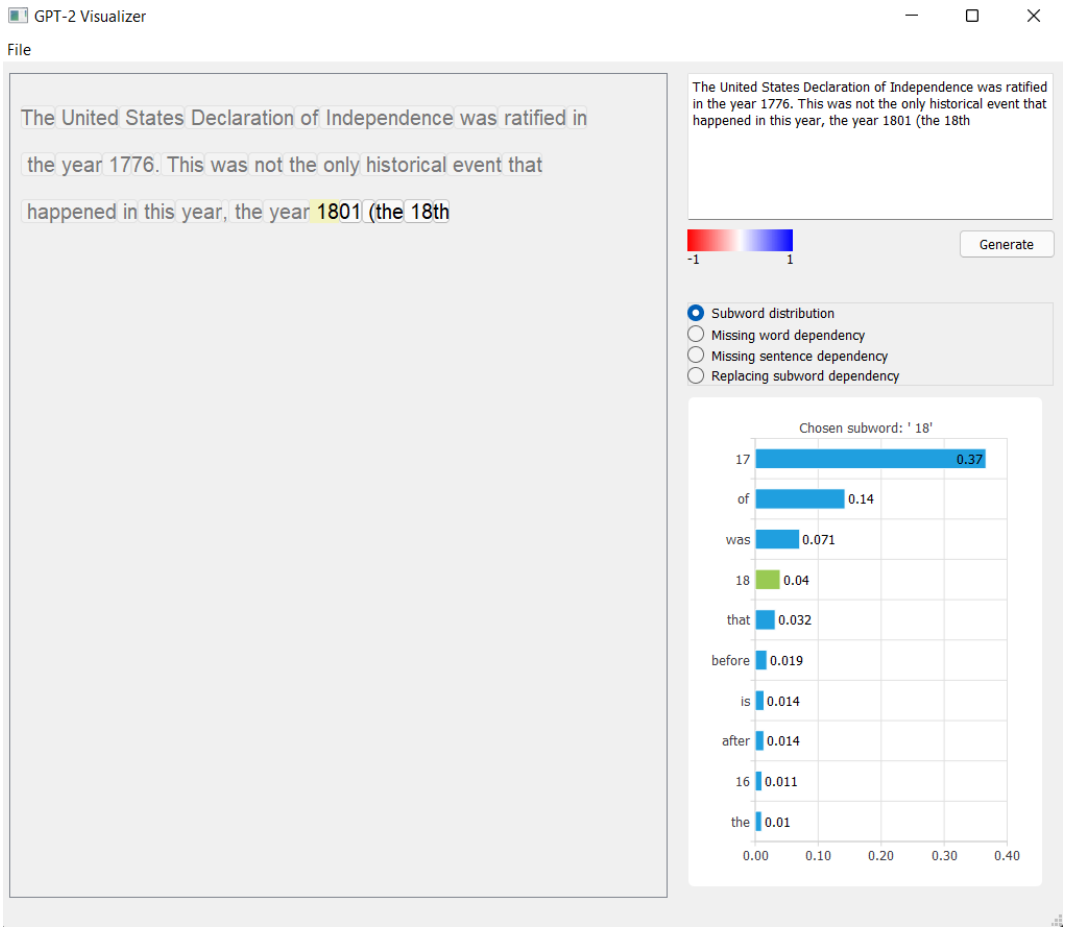
Subword distribution je najjednoduchším z našich módov. Pre každý subword v texte nás zaujíma distribúcia, ktorú mal model pri generovaní daného subwordu. Z tejto distribúcie si uchovávame 10 najpravdepodobnostnejších subwordov (spolu s ich pravdepodobosťami). Ak sa v desiatke najpravdepodobnejších subwordov nenachádzal ten, ktorý sme vygenerovali, tak ho tam pridáme aj s jeho pravdepodobnosťou.

Tento mód predstavuje prvý krok analýzy správania GPT-2 modelu, pretože ukazuje, nad ktorými subwordmi model „rozmýšľa“ v danom kontexte, teda ktorým subwordom zadal najvyššie skóre.

Ak je zvolený *sampling*, ktorý vážene podľa pravdepodobnosti náhodne zvolí subword na výstup, tak zvolený subword je len zhoda okolností. Analýzu by mohlo zaujímať, aká bola pravdepodobnostná distribúcia pri generovaní tohto slova.

Ak sa vygeneruje nejaká nepravdivá informácia v texte, niekedy môže byť na vine samplovanie. Tento mód nám môže odhaliť, akú váhu prikladal model pravdivej (alebo logickej) informácii.

Pre príklad sa môžeme pozrieť na obrázok 2.1. Vstupný text predaný modelu bol priamo navrhnutý tak, aby sa model opäť snažil vygenerovať rok 1776, tak ako prišiel vo vstupnom texte. Tokenizer rozdelil číslo 1776 na vstupe na dva subwordy („␣17“ a „76“). Ale pri generovaní roku sieť vygenerovala rok 1801 (rozdelený na „␣18“ a „01“). Na pravej strane môžeme vidieť pravdepodobnostnú distribúciu pri generovaní subwordu „␣18“ a vidíme, že tento subword dostal pravdepodobnosť okolo 4 %, zatiaľ čo subword „␣17“, ktorý sme očakávali, dostal 37 %.



Obrázek 2.1: Ukážka módu Subword distribution. Táto obrazovka je bližšie popísaná v kapitole 4.2.1. Vstup do siete bol: *„The United States Declaration of Independence was ratified in the year 1776. This was not the only historical event that happened in this year, the year“*. Môžeme vidieť Subword distribution pre prvý generovaný subword – „␣18“.

2.2 Missing word dependency

Missing word dependency sa snaží zisťovať závislosti medzi dvomi subwordmi. Pre jednoduchšie vysvetlenie toho, čo tento mód vlastne robí, si jednotlivé subwordy v dvojici pomenujme. Subword, ktorý sa nachádza v texte skôr nazvime *chýbajúcim* subwordom, ten druhý označme ako *vyhodnocovaný*.

Mód sleduje zmenu v skóre vyhodnocovaného subwordu, ak ten chýbajúci je z pôvodného textu odstránený.

Konkrétne, zoberieme celý vygenerovaný text až po vyhodnocovaný subword a odstránime z neho chýbajúci subword. Tento text potom vložíme ako vstup do daného modelu GPT-2 s nastavenými parametrami a pozrieme sa, akú pravdepodobnosť dostane vyhodnocovaný subword za takýchto okolností.

2.2.1 Motivácia

Tento mód je zaujímavý z dôvodu jednoduchosti a presnosti, avšak má aj svoje nedostatky. V modeli BERT (Devlin a kol., 2018) môžeme vidieť, že neprítomnosť nejakého slova v texte (*maskovanie*) nemá veľký vplyv na pochopenie textu ako celku. Rozhodli sme sa vyskúšať, ako by nám podobná logika vedela v modeloch GPT-2 pomôcť nájsť závislosti medzi slovami.

2.2.2 Počítanie závislosti

Závislosť medzi subwordmi v tomto móde znamená relatívny rozdiel medzi pôvodnou pravdepodobnosťou a *chýbajúcou* pravdepodobnosťou vyhodnocovaného subwordu. Počíta sa takto:

$$-\frac{missing\_probability - original\_probability}{original\_probability}$$

Výsledok tohto vzorca sa ešte upraví na hodnoty v rozmedzí (-1) – 1 (funkciou *numpy.clip()*) a potom sa nastaví farba štítku a aj jej nepriehľadnosť (*opacity*), kde kladné hodnoty dostanú modrú farbu a záporné červenú (viac v 4.4.2).

2.2.3 Podrobný príklad

Pre jednoduchosť predpokladajme, že jeden subword je interpunkcia alebo jedno celé slovo (plus medzera pred ním, ak tam je) a predstavme si, že máme text, ktorý nám GPT-2 model vygeneroval samostatne. Vygenerovaný text by napríklad mohol byť: „*Zleteli orly z Tatry, tiahnu na podolia, ponad vysoké hory, ponad rovné polia*“. Chceme vedieť, ako by sa zmenila pravdepodobnosť slova „␣*rovné*“, ak by slovo „␣*vysoké*“ v texte chýbalo.

Modelu siete by sa na vstup predložil text: „*Zleteli orly z Tatry, tiahnu na podolia, ponad hory, ponad*“ a model by sme požiadali o vygenerovanie jedného ďalšieho subwordu. Sledovali by sme, akú pravdepodobnosť by dostalo slovo „␣*rovné*“.

2.2.4 Nedostatky módu

Za nedostatky módu môžeme považovať to, že veta s vynechaným subwordom môže stratiť pôvodný zmysel, alebo to, že nám môže vzniknúť nezmyselné slovo. Je veľmi jednoduché zadať sieti vstup s celým textom až po vyhodnocovaný subword, kde sa jeden subword vynechá.

Strata pôvodného zmyslu vety

Hlavným nedostatkom tohto módu je, že veta s vynechaným slovom nemusí dávať vždy zmysel. Niekedy môže mať takáto veta dokonca iný význam, hlavne v angličtine, kde jedno slovo môže zastávať viaceré funkcie vo vete. Napríklad v jednom kontexte môže byť dané slovo podstatné meno a v inom sloveso („record“ sa dá preložiť ako rekord, nahrávka, zaznamenať a pod.), zmazanie slovesa by teda mohlo zmeniť prísudok vety.

Ak má veta iný význam, tak, samozrejme, model to dokáže rozpoznať a na prítomnosti *chýbajúceho* subwordu bude závislý takmer celý nasledujúci text (hlavne, ak to mení drasticky nejaký význam vo vete).

Na druhú stranu, ak nová veta význam nemá, model je v prvých slovách zmätený, ale celkom rýchlo sa dokáže z tejto situácie spamätať a pokračovať v logickom generovaní, podľa našich zistení.

Veľký počet subwordov pri generovaní začína na znak medzery (obvykle každé slovo okrem slov na začiatku vety). Algoritmus na rozdeľovanie textu na subwordy je popísaný v 1.2.3. Celkový výstupný text je teda len jednoduché zlepenie jednotlivých subwordov priamo za seba.

Vznik nezmyselného slova

Tento princíp nám prináša ďalší problém pre tento mód. Ak sa za chýbajúci subword vezme subword, po ktorom sa v texte nachádza subword bez medzery na začiatku, subwordy sa začnú spájať do nezmyselných slov. Táto situácia nastáva, ak sa pozeráme na slová zložené z viacerých subwordov (napr. „␣limousine“ sa rozdelí na „␣lim/ous/ine“) alebo skrátené slová v angličtine (*contractions* – napr. „␣I’ve“ sa rozdelí na „␣I/’ve“) a na prvý subword z nich sa pozeráme ako na *chýbajúci* subword.

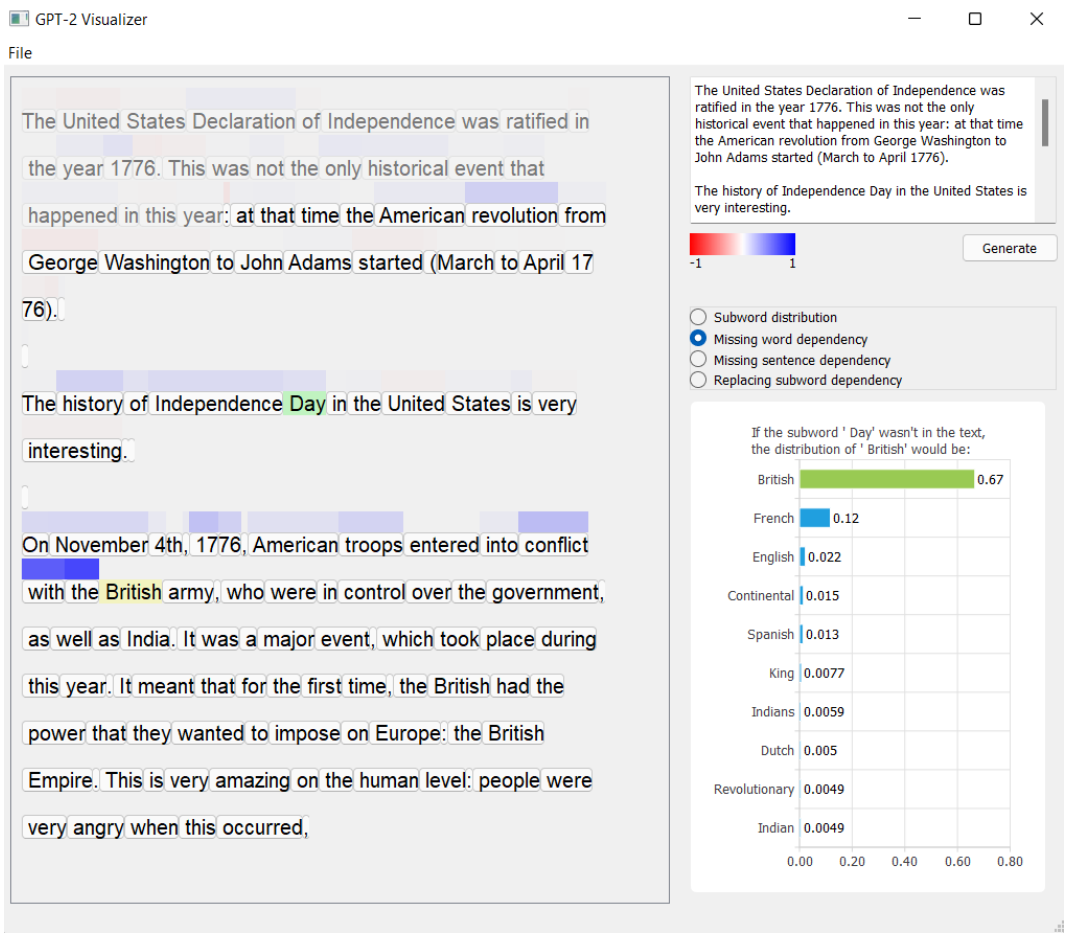
Ak model bude mať text ako *„... the limousine I’ve found...“*, tento text sa na subwordy rozdelí takto: „... the/ lim/ous/ine/ I/’ve/ found...“. Ak sa za chýbajúci subword zvolí *„␣lim“*, text, ktorý sa modelu vloží na vstup bude: *„... theousine I’ve found...“*, čo je nezmyselný text, pretože slovo „theousine“ sa v anglickom jazyku nenachádza.

Naše testy ale ukázali, že model GPT-2 sa z toho dostane celkom rýchlo, pretože pre tento model to je len ako vynechanie (alebo zamaskovanie) jedného slova. Pri ďalšom generovaní sa môže model zamerať aj na iné slová ako to jedno nezmyselné. Aj napriek nezmyselnosti textu to, zdá sa, GPT-2 model úplne nerozhodí a nemá takýto problém módu negatívne následky na ukážku závislostí.

2.2.5 Vizuálny príklad

Príklad spustenia tohto módu v našom vizualizátore je uvedený na obrázku 2.2. Rovnako si môžeme pozrieť graf, ktorý sa generuje na pravej strane aplikácie.

Graf hovorí o tom, ako by vyzerala pravdepodobnostná distribúcia pre subword „␣British“, ak by sa subword „␣Day“ vynechal.



Obrázek 2.2: Ukážka módu Missing word dependency v našom programe. Vstup do siete bol: *„The United States Declaration of Independence was ratified in the year 1776. This was not the only historical event that happened in this year“*. Môžeme vidieť Missing word dependency pre vygenerovaný subword „␣British“. Na grafe vpravo sledujeme, ako by vyzerala pravdepodobnostná distribúcia tohto subwordu, ak by sa subword „␣Day“ vynechal.

Vyhodnocovaný subword je navyše zafarbený na žlto a chýbajúci subword na zeleno, aby sme mohli tieto subwordy vidieť priamo v texte. Graf, ktorý sa zobrazuje na pravej strane dáva zmysel, ak sa porovnáva s grafom, ktorý vygeneruje mód Subword distribution pre vyhodnocovaný subword. Vtedy vidíme, ako sa menia pravdepodobnosti pre jednotlivé subwordy.

2.3 Missing sentence dependency

Missing sentence dependency sa pozerá na to, aký závislý je vygenerovaný subword na nejakej z predošlých viet.

Funguje veľmi podobne ako Missing word dependency (mód popísaný v sekcii 2.2). V tomto móde sledujeme, ako sa mení pravdepodobnostná distribúcia

pre *vyhodnocovaný* subword, ak sa vynechá nejaká konkrétna veta z predchádzajúceho textu. Pozeráme sa teda nielen na pravdepodobnosť pre daný subword zadanú modelom, ale podobne ako pri predchádzajúcich módoch sledujeme pravdepodobnostnú distribúciu 10 najpravdepodobnejších subwordov, ak je *chýbajúca* veta z textu vynechaná plus pravdepodobnosť *vyhodnocovaného* subwordu (ak sa nenachádza v tejto desiatke).

2.3.1 Počítanie závislosti

Závislosť subwordu na predchádzajúcej vete v tomto móde znamená relatívny rozdiel medzi pôvodnou a novou (*chýbajúcou*) pravdepodobnosťou vyhodnocovaného subwordu (ak je daná veta odstránená). Počíta sa takto:

$$-\frac{missing\_probability - original\_probability}{original\_probability}$$

Výsledok tohto vzorca sa ešte upraví na hodnoty v rozmedzí (-1) – 1 (funkciou *numpy.clip()*) a potom sa nastaví farba štítku a aj jej nepriehľadnosť (*opacity*), kde kladné hodnoty dostanú modrú farbu a záporné červenú. Táto farba sa nastavuje pre všetky subwordy patriace danej vete.

2.3.2 Motivácia

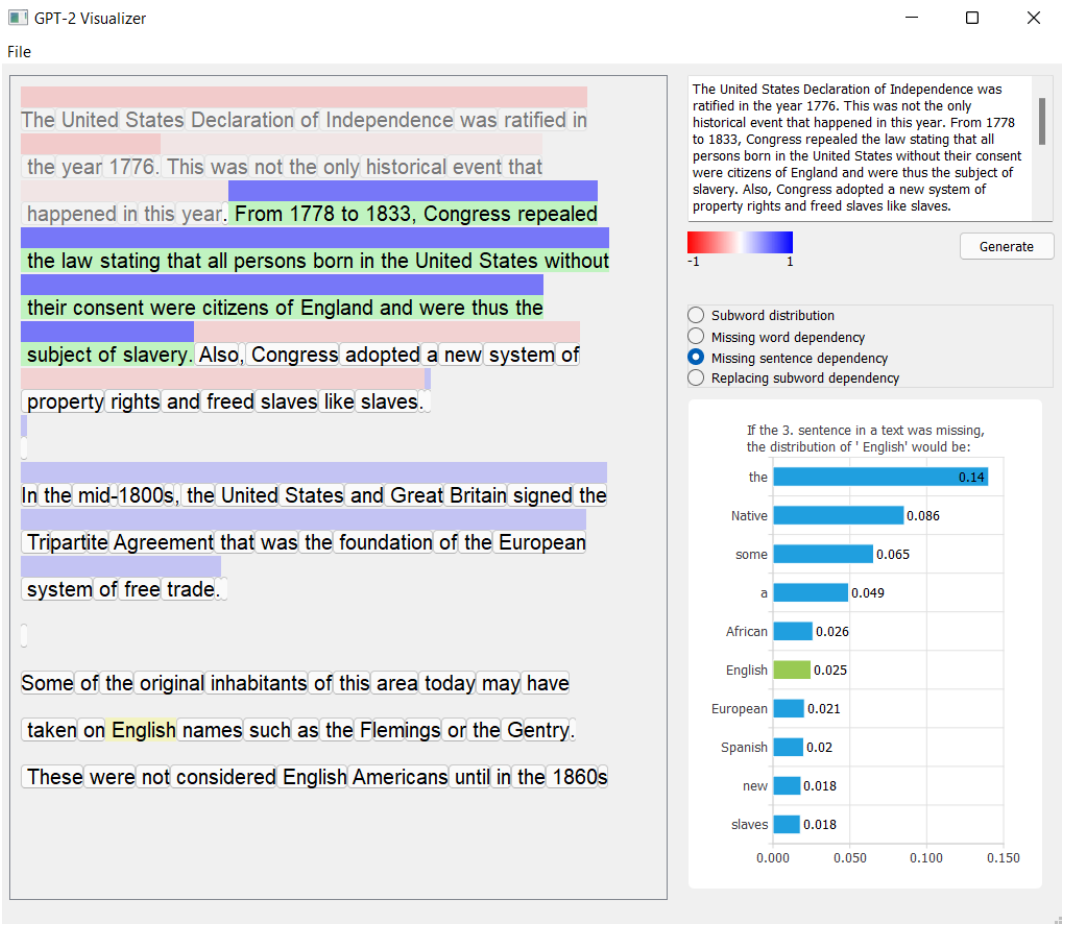
Tento mód sme vymysleli ako jedno z možných riešení nedostatkov *Missing word dependency*. Konkrétne toho, že po vynechaní subwordu nemusí veta dávať zmysel alebo významovo sa môže celá veta zmeniť. Veta je najmenší takýto celok, ktorý sme schopní správne definovať tak, že text bude dávať po gramatickej stránke význam. Samozrejme, aj v tomto prípade môžu nastať situácie, keď vety nebudú dávať zmysel (napríklad, ak sa nasledujúci text odkazuje na dáta z chýbajúcej vety).

Navyše, pre každý vyhodnocovaný subword sa výpočty robia oveľa rýchlejšie, pretože predchádzajúcich viet je vždy menej ako subwordov. Pri dlhších generovaných textoch by tento mód mohol v krátkom čase ukázať, ktoré vety sú dôležité pri generovaní vyhodnocovaného subwordu.

2.3.3 Vizuálny príklad

Na obrázku 2.3 môžeme vidieť príklad spustenia tohto módu vo vizualizátore. Tento text, podobne ako ostatné, ktoré vygeneruje model GPT-2, je nepravdivý, pretože model si iba spája slová a nevie, čo je pravda a čo nie je.

Na grafe vidíme pravdepodobnosť vygenerovaného subwordu „␣English“ (*anglický*), ak by tretia veta, ktorá spomína Anglicko, bola z textu odstránená. Vidíme, že táto pravdepodobnosť je 2,5 % a štítok nad treťou vetou je zafarbený celkom nepriehľadnou farbou. S použitím farebnej legendy na pravej strane pod textovým poľom, môžeme odhadom povedať, že závislosť subwordu od tejto vety je niekde okolo 0.5 (50 %). Pôvodná pravdepodobnosť teda mohla byť približne 5 %.



Obrázek 2.3: Ukážka módu Missing sentence dependency v našom programe. Vstup do siete bol: *„The United States Declaration of Independence was ratified in the year 1776. This was not the only historical event that happened in this year“*. Môžeme vidieť Missing sentence dependency pre subword „␣English“. Graf sa zobrazuje pre 3. vetu, tá je zvýraznená zelenou farbou.

2.3.4 Detekcia viet

Na rozdelenie textu na jednotlivé vety používame task *Tokenize* v online nástroji s názvom *MorphoDiTa*, ktorý je bližšie popísaný v predošlej kapitole 1.5.

2.4 Replacing subword dependency

Replacing subword dependency je najzložitejší mód v programe. Hlavnou myšlienkou módu je zameniť slovo v texte za slovo, ktoré by zohrávalo rovnakú rolu vo vete a potom sledovať dopad tejto zmeny (zmeny v pravdepodobnostiach vyhodnocovaného subwordu). Snažíme sa, aby toto slovo nejak zapadalo do predchádzajúceho textu.

Každé takéto slovo označme ako *podobné* slovo. Slovo, ktoré bolo vygenerované v pôvodnom texte a teraz ho nahrádzame, označíme ako *zamieňané*.

Podobné slová vyberáme z 10 najpravdepodobnejších slov na danú pozíciu v texte (kvôli konzistentnosti so Subword distribution v 2.1). To robíme preto, aby sme zariadili, že to nie sú len ľubovoľné slová, ale aby zapadali do kontextu

celého textu (o čo sa postará GPT-2 model tým, že ich zaradí medzi najlepších 10 v pravdepodobnostnej distribúcii). Ideálne by bolo, ak by tieto podobné slová mali v texte rovnakú rolu (ideálne, ak by to slovo malo rovnaký slovný druh a gramatické kategórie, aby následná veta dávala zmysel).

To sa snažíme docieliť tak, že vygenerujeme všetky možné vety, kde zamieňaný subword nahradíme *podobným* slovom a všetky tieto vety vložíme do online nástroja s názvom *MorphoDiTa* (popísanom v 1.5). Na tomto nástroji zapneme task *Tag*, ktorý nám ku každému slovu vráti jeho morfologickú značku (*tag*). Za podobné slová následne považujeme len tie slová, ktoré dostanú rovnakú značku ako *zamieňané* slovo.

Výpočty tentokrát nebudú prebiehať na všetkých dvojiciach subwordov. Rozhodli sme sa, že vytvoríme ďalšie reštrikcie pre podobné slová. Dôvodom, prečo sme tak urobili, je hlavne rýchlosť výpočtov a ich celkový počet. Predpokladáme, že pre niektoré subwordy nemá zmysel robiť výpočty, nakoľko by pravdepodobne nezohrávali rolu pri generovaní nasledujúcich subwordov zo sémantického hľadiska.

2.4.1 Počítanie závislosti

Závislosť medzi subwordmi v tomto móde znamená relatívny rozdiel medzi pôvodnou pravdepodobnosťou vyhodnocovaného subwordu a priemernou pravdepodobnosťou vyhodnocovaného subwordu cez podobné slová. Počíta sa takto:

$$-\frac{avg - original\_probability}{original\_probability},$$

kde

$$avg = \frac{\sum_{i=1}^{s} similar\_probability_i}{s},$$

kde s je počet podobných slov

Výsledok tohto vzorca sa ešte upraví na hodnoty v rozmedzí (-1) – 1 (funkciou *numpy.clip()*) a potom sa nastaví farba štítku a aj jej nepriehľadnosť (*opacity*), kde kladné hodnoty dostanú modrú farbu a záporné červenú.

2.4.2 Reštrikcie pre zamieňaný subword

Výpočty pre zámenu subwordov sa nezakladajú, ak 1 subword nepredstavuje celé slovo (s možnou medzerou pred slovom). Dôvodom je, že sa snažíme subword zameniť za iný subword, ktorý zapadá do predchádzajúceho kontextu. Zamieňanie subwordov za iné nemusí dávať vždy zmysel.

Pre príklad si vieme predstaviť generovanie vlastných mien. Jedno z fiktívnych miest, ktoré sme použili pre generovanie textov, bolo *„Gransbury Woods“*. Tokenizer toto meno rozdelí na Gr/ans/bury. Pri zamieňaní by sme mohli zameniť prvý subword (uprostred vety by to bolo „␣Gr“), ale mohli by sa tam vyskytnúť aj iné slová, ako napríklad „␣woods“, „␣sea“ alebo „␣forest“, čím by sme po zámene dostali celkom nezmyselné slová ako „woodsansbury Woods“.

Ďalšie výpočty, ktoré preskakujeme, nastávajú, keď *zamieňaný subword* zohráva vo vete rolu neplnovýznamového slova alebo číslovky. Tieto slová nemusí byť vždy rozumné nahrádzať inými slovami s rovnakým slovným druhom hlavne vtedy, keď generujeme v anglickom jazyku. Príkladom môžu byť „prepositional

verbs“ (predložkové slovesá, ako napríklad „give up“ alebo „agree with“). V takýchto slovách nie je vždy možné nahradiť predložku inou tak, aby mala veta významovo zmysel.

Číslovky síce význam ako taký nesú, ale otázka je, aký majú vplyv na vetu a generovanie textu, ak by sme často len nahradili jedno číslo iným. Nejaký vplyv môže mať napríklad storočie (17. a 18. storočie – 17th and 18th century), v tomto prípade sa ale číslovka skladá z dvoch subwordov („17“ a „th“) a tým pádom sa nekvalifikuje medzi možné podobné ani zamieňané slová (pretože slovo sa skladá z dvoch subwordov).

Dôsledkom toho je, že v tomto móde jeden *zamieňaný* subword je vždy jedno slovo a pojmy subword a slovo majú v kontexte tohto módu rovnaký význam.

2.4.3 Podrobný príklad

Pre tento príklad predpokladajme, že každé slovo a interpunkcia v texte predstavujú jeden subword a medzery patria k slovám za nimi. Predstavme si, že by model vygeneroval text: „*Zleteli orly z Tatry, tiahnu na podolia, ponad vysoké hory, ponad rovné polia*“. My by sme chceli vedieť, ako ovplyvňuje slovo „␣*orly*“ slovo „␣*podolia*“.

Pozrieme sa na to, akým slovám dal model najvyššie pravdepodobnosti pri generovaní slova „␣orly“. Môžu to byť slová rôznych slovných druhov, hlavne pri tak krátkom vstupnom texte. Zistíme, že najvyššie pravdepodobnosti dostali slová „␣orly“, „␣havrany“, „␣sýkorky“, „␣čierne“ a „␣sme“.

Vygenerujeme vety s týmito slovami („Zleteli orly z Tatry...“, „Zleteli havrany z Tatry...“, „Zleteli sýkorky z Tatry...“, „Zleteli čierne z Tatry...“, „Zleteli sme z Tatry...“) a pozrieme sa na ich *part-of-speech tag* vygenerovaný online nástrojom MorphoDiTa (popísanom v kapitole 1.5).

Zistíme, že rovnakú morfologickú značku v texte ako „␣orly“ vrátila MorphoDiTa aj pre vety so slovami „␣havrany“ a „␣sýkorky“.

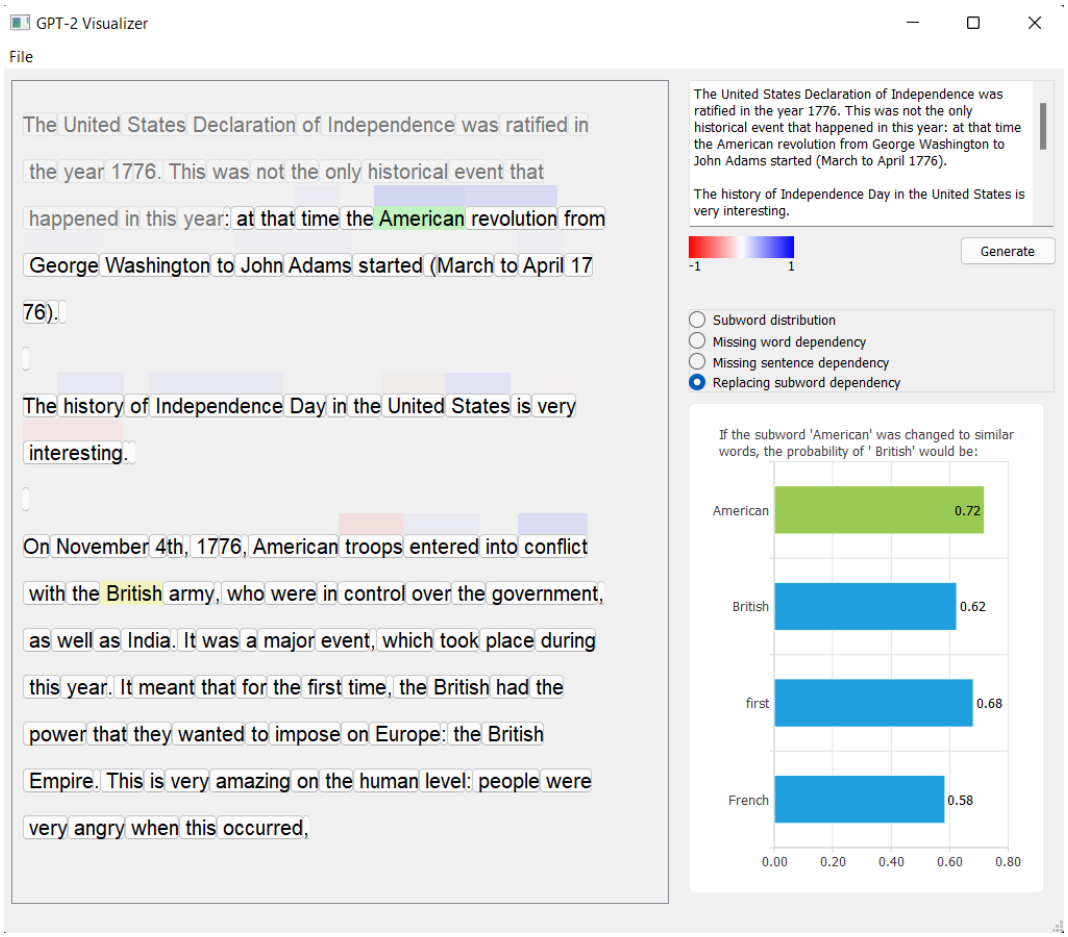
Modelu siete GPT-2, ktorý má rovnaké parametre ako ten, ktorý vygeneroval celý text, teda predložíme vstup „*Zleteli havrany z Tatry, tiahnu na*“ a budeme sledovať, akú pravdepodobnosť priradí slovu „␣podolia“. Potom to isté spravíme pre vstup „*Zleteli sýkorky z Tatry, tiahnu na*“.

Nakoniec si ešte vypočítame priemernú pravdepodobnosť slova „␣podolia“ cez všetky *podobné* slová.[1]

2.4.4 Vizuálny príklad

Príklad spustenia tohto módu v našom vizualizátore je uvedený na obrázku 2.4. Na grafe vpravo môžeme vidieť, aké ostatné podobné slová sa vygenerovali pre subword „␣American“ (zvýraznený zelenou farbou). Pravdepodobnosť pre vygenerovanie subwordu „␣British“ (zvýrazneného žltou farbou) je síce najväčšia, ale pre ďalšie podobné slová stále dosahuje vyše 50 % pravdepodobnosť.

[1] Môžeme očakávať, že samotný druh vtáctva nebude mať nejaký veľký efekt na destináciu tiahnutia, takže v tomto prípade by sme asi nemohli očakávať nejaký zaujímavý výsledok. Príklad je určený hlavne na ukážku rôznych slovných druhov, ktoré môžu byť v danom kontexte vygenerované a ako z nich filtrujeme podobné slová.



Obrázek 2.4: Ukážka módu Replacing subword dependency v našom programe. Vstup do siete bol: *„The United States Declaration of Independence was ratified in the year 1776. This was not the only historical event that happened in this year, the year“*, môžeme vidieť Replacing subword dependency pre vygenerovaný subword „␣British“. Na grafe vpravo sledujeme, ako sa mení pravdepodobnosť vygenerovania tohto subwordu, ak sa subword „␣American“ zamení za *podobné slová*.

3. Popis programu na vizualizáciu

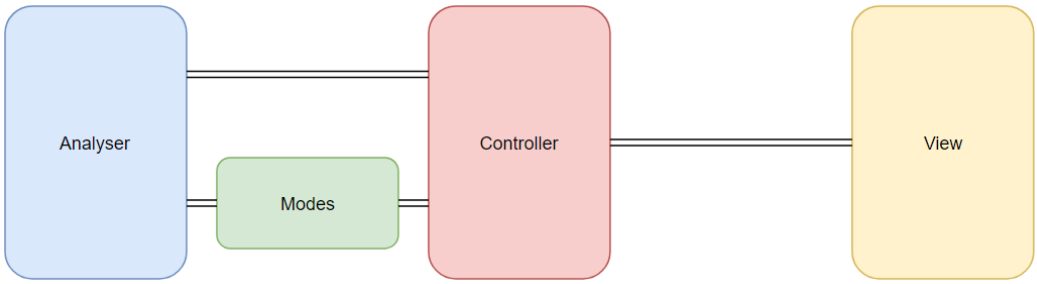
3.1 Celková architektúra programu

Z dizajnovej stránky je celá vizualizačná aplikácia založená na miernej obmene návrhového vzoru *MVC* (Model-View-Controller). Koncept MVC sa často využíva ako architektúra webových aplikácií (Leff a Rayfield, 2001). Takáto aplikácia sa delí na 3 logické časti:

- Model, ktorý udržiava dáta a logiku programu;
- Pohľad (v texte ďalej budeme používať originálne pomenovanie *View*), ktorý zobrazuje informácie;
- Radič (v texte ďalej už len ako *Controller*), ktorý vyhodnocuje a spracováva vstup používateľa.

Pri dizajne aplikácie sme sa inšpirovali týmto konceptom a tri hlavné komponenty aplikácie sme nazvali Analyzér (pre konzistenciu anglického názvoslovia budeme ďalej používať výraz *Analyser*), View a Controller.

Analyser sme sa snažili navrhnúť čo najjednoduchšie, s čo možno najvšeobecnejším API a logiku vložili do Controller komponenty. Controller dostane používateľský vstup, ktorý vyhodnotí a jednotlivými požiadavkami si (aj s pomocou módov) vyskladá od Analyser komponenty informácie, ktoré predá View na zobrazenie.



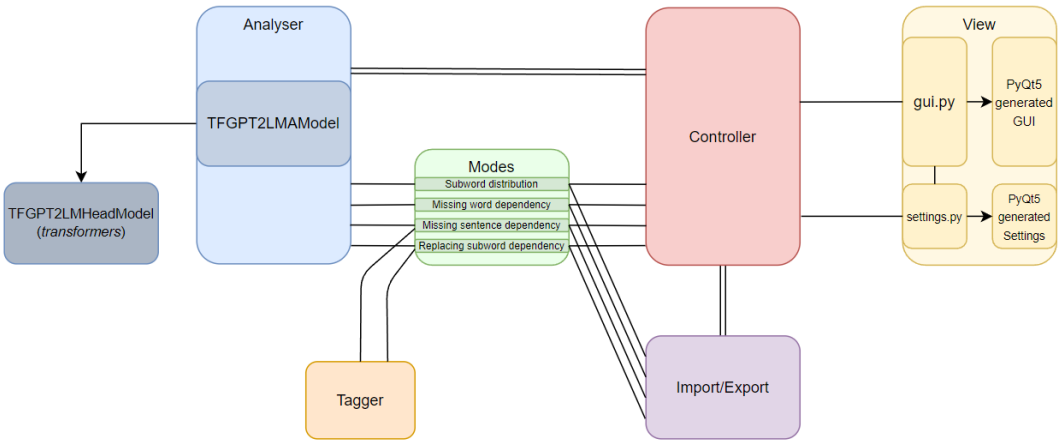
Obrázek 3.1: Hlavná myšlienka architektúry

Naším počiatočným cieľom bolo mať čo najviac oddelené všetky 3 hlavné komponenty programu, aby mohli byť v prípade potreby čo možno najjednoduchšie zameniteľné. Túto zameniteľnosť komponent dokonca aj využívame, pretože sme v rámci práce vytvorili aplikáciu bez používateľského rozhrania, nazvanú *Controller NoGUI*, ktorá je určená iba na spúšťanie výpočtov na výpočetných centrách a clustroch. Jej jediná funkčnosť je, že vygeneruje alebo analyzuje zadaný text na vstupe s použitím daných parametrov a spustí export do súboru, ktorý nakoniec pôjde jednoducho zobraziť pomocou funkcie Import. Viac o tejto funkcionalite je v kapitole 4 (Používateľský manuál).

Controller NoGUI teda dokáže úplne zameniť Controller komponentu s využitím všetkých módov, ale bez využívania View komponenty.

Jednotlivých komponent je v programe ešte viac. Máme komponentu Tagger určenú na REST API komunikáciu s nástrojom MorphoDiTa a komponentu ImportExport, ktorá je oddelená od Controller komponenty a využíva jednotlivé módy na Import z logu do vizualizátora a export do logu.

Celková architektúra aj s jednotlivými podčasťami komponent sa nachádza na obrázku 3.2.



Obrázek 3.2: Celková architektúra programu (obrázok vo vyššom rozlíšení je súčasťou Prílohy A.1). Dvojité čiary znázorňujú komunikáciu medzi hlavnými komponentami, jednoduchá čiara ukazuje komunikáciu konkrétnej podčasti komponenty s inou komponentou alebo podčasťou, šípky znázorňujú dedičnosť a ukazujú vždy smerom k rodičovi.

Celý zdrojový kód je písaný v jazyku *Python*, niektoré automaticky generované časti používateľského rozhrania vytvorené v programe *Qt Designer* majú iné prípony. Aktualizovaný zdrojový kód sa nachádza na adrese `https://gitlab.mff.cuni.cz/siposd/gpt2-visualizer` a je prístupný verejnosti.

3.2 Analyser

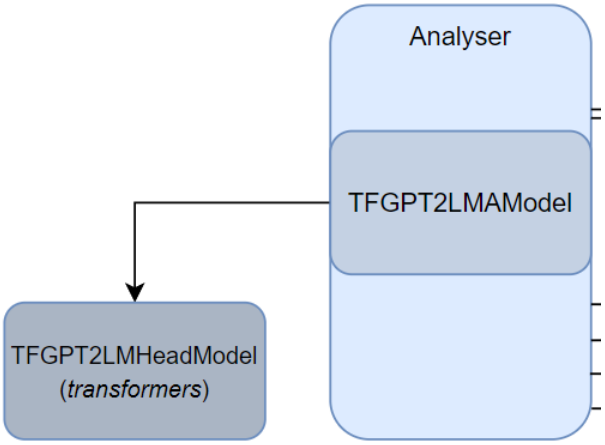
Analyser je komponenta, ktorá obaľuje daný GPT-2 model a sprístupňuje API pre jednotlivé požiadavky od modelu (ako napríklad generovanie výstupného textu pre vstupný text, analýza daného textu, ...).

Analyser pracuje s použitím knižnice *tensorflow*, čo znamená, že podporuje len modely vytvorené v tejto knižnici[1].

Na začiatku si táto komponenta vytvorí tokenizer a model na základe vstupného parametra pri volaní programu. Tokenizer aj model sa vytvoria zavolaním statickej metódy *from_pretrained()*, ktorá ako parameter berie buď názov modelu z dostupných modelov v API od Hugging Face[2] alebo cestu k adresáru, kde sa nachádza tokenizer aj model (viac v 4.1).

[1]Otvorenie modelov vytvorených s pomocou knižnice *pytorch* je možné miernou úpravou zdrojového kódu v súbore *analyser.py*, kde v metóde *setup()* sa do oboch volaných metód pridá parameter *from_pt=True*. Toto správanie nie je úplne odladené a otestované, preto nie je ani podporované.

[2]https://huggingface.co/models



Obrázek 3.3: Bližší pohľad na komponentu Analyser. Šípka značí dedenie vlastností a je nasmerovaná k rodičovi.

Trieda tokenizerov, ktorá sa vytvára, je klasická *GPT2Tokenizer* trieda. Model sa ale vytvára ako inštancia triedy *TFGPT2LMAModel*. Túto triedu sme vytvorili v programe ako jednoduché rozšírenie triedy *TFGPT2LMHeadModel* z open-source zdrojového kódu knižnice *transformers* od Hugging Face. Jediný rozdiel oproti pôvodnej triede je v metóde *_generate_no_beam_search()*. Zmena v tejto metóde oproti pôvodnej nastala v troch riadkoch, kde sa pozeráme, či nám nebol predaný argument s názvom *wanted_output* do *kwargs*, a ak áno, tak prepisujeme vygenerovaný token príslušným tokenom z *wanted_output*.

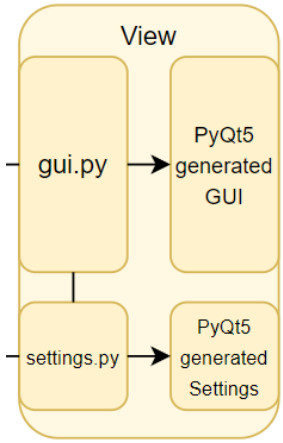
Tento princíp využívame hlavne pri analyzovaní textu (pri generovaní to nepotrebujeme). V analyzovaní textu chceme, aby model zistil presné hodnoty skóre a pravdepodobnosti pre jednotlivé tokeny, ale následne chceme, aby na výstup vygeneroval token, ktorý sa nám pri výpočtoch zíde (napríklad presne ten, ktorý vygeneroval náhodne nejaký z modelov GPT-2 predtým a my chceme tento text analyzovať).

3.3 View

Komponenta *View* sa stará o zobrazovanie jednotlivých údajov na obrazovke. Máme dve obrazovky v našom programe: *Vizualizátor* (alias hlavná obrazovka programu) a *Settings* (obrazovka na nastavenie parametrov programu). Viac o používaní a vzhľade obrazoviek je napísané v ďalšej kapitole, presnejšie v sekcii 4.2.

Dizajn oboch obrazoviek je navrhnutý v aplikácii *Qt Designer*, ktorá úzko spolupracuje s knižnicou *PyQt5* používanou v programe. Tieto obrazovky vytvorili a vygenerovali súbory *gui_generated.py* a *settings_generated.py*, ktoré ostali bez zmeny po generovaní.

Zmeny a nejaké doplnenia (ako napríklad obsluhovanie signálov alebo nastavenie predvolených hodnôt) sú vytvorené v triedach *GUI* a *Settings*, ktoré dedia od automaticky vygenerovaných tried vytvorených vo vyššie spomínaných súboroch.



Obrázek 3.4: Bližší pohľad na komponentu View. Šípka značí dedenie vlastností a je nasmerovaná k rodičovi. Jednoduchá čiara značí komunikáciu medzi jednotlivými podčasťami.

3.3.1 Obrazovka vizualizátora

Obrazovka vizualizátora (v triede GUI) obsluhuje všetky vstupy, ktoré zadáva používateľ (napr. písanie textu, klikanie na tlačidlá, klikanie na hlavné menu ...). Navyše k tomu obsahuje táto obrazovka potrebné API na správne zobrazovanie jednotlivých súčastí používateľského rozhrania. Máme tu metódy na zobrazovanie výstupného textu rozdeleného do subwordov, zobrazovanie farebných štítkov nad subwordmi, kreslenie grafov aj nastavovanie primárnej a alternatívnej farby jednotlivých subwordov.

3.3.2 Obrazovka Settings

Obrazovka *Settings* dostáva momentálne nastavenia parametrov pre model GPT-2, ktoré zobrazí ako predvolené hodnoty pri spustení obrazovky. Ak spraví používateľ nejaké zmeny a potvrdí ich, táto obrazovka ich zaznamená a informuje o nich hlavnú Controller komponentu.

3.4 Controller

Controller je najzákladnejšia časť programu. Spája *View* komponentu s *Analyserom* a určuje, čo sa bude vykonávať a ako.

Ako je podrobne popísané v nasledujúcej kapitole v sekcii 4.3, funkčnosť programu sa delí na 2 časti: získanie generovaného textu a samotná analýza závislostí. O obe tieto časti sa stará táto komponenta.

Na obrázku 3.1 môžeme vidieť hlavnú myšlienku celkovej architekúry programu. Z komponenty Controller do Analysera vedú 2 cesty, jedna je priama, to je získanie generovaného textu a jeho spracovanie (o to sa stará nastavenie hlavného módu – *Generate* alebo *Analyse* vo View). Druhá cesta vedie cez komponentu *Modes*, ktorá robí analýzu závislostí.

Smerom View → Controller sa v tejto komponente nachádzajú metódy na obsluhovanie jednotlivých tlačidiel a funkcionalít na obrazovke, konkrétne:

- kliknutie na tlačidlo Generate/Analyse,
- zmena aktuálneho módu,
- zmena parametrov pre generovanie výstupu modelu GPT-2 cez obrazovku Settings,
- klikanie na subwordy a farebné štítky nad nimi,
- Import a Export voľby.

Smerom Controller → View obsahuje táto komponenta metódy na:

- vyplnenie vstupno-výstupného poľa výstupným textom z komponenty Analyser,
- vytváranie tlačidiel subwordov na obrazovke na základe výstupu z komponenty Analyser,
- vytváranie farebných štítkov s farbou predpočítanou v aktuálnom móde,
- kreslenie grafov na základe výpočtu v aktuálnom móde,
- vyfarbovanie tlačidiel subwordov na žlto (primárna farba) alebo na zeleno (alternatívna farba) tak, ako určí aktuálny mód,
- resetovanie hlavnej obrazovky (napríklad pri novom generovaní na základe vstupu).

Táto komponenta sa taktiež stará aj o viacvláknovosť, aj keď v tejto verzii funguje aplikácia zatiaľ na maximálne dvoch vláknach súčasne. Výpočty módov fungujú v inom vlákne ako v tom, ktoré sa stará o responzivitu hlavnej obrazovky. Je zakázané z používateľského pohľadu vytvárať viac výpočtov naraz.

V ďalšej verzii aplikácie je cieľom sparalelizovať výpočty ešte viac, aby mali ešte vyšší potenciál byť rýchlejšie (minimálne pri volaní exportu, kde je potrebné vypočítať závislosti medzi všetkými dvojicami subwordov vo všetkých módoch).

Controller navyše obsahuje cache (triedy *Cache*), ktorý si ukladá výpočty jednotlivých módov, aby nebolo potrebné robiť rovnaký výpočet viackrát.

3.5 Módy

Módy sú komponenty programu, ktoré sa starajú o analýzu správania daného GPT-2 modelu. Celý princíp módov je nastavený tak, aby bolo jednoduché nejaký mód odobrať alebo pridať nový.

Každý mód dedí od triedy *Mode*, ktorá definuje nejaké základné správanie každého módu, napríklad metódy na kliknutie na subword alebo farebný štítok (*button_click()*, resp. *label_click()*), metódu na exportovanie, inicializačnú metódu *___init___()* a pod.

Taktiež máme triedu *ModesManager*, ktorá priraďuje jednotlivým módom ich rádio voľbu na používateľskom rozhraní a pridáva ich do zoznamu módov pre Controller.

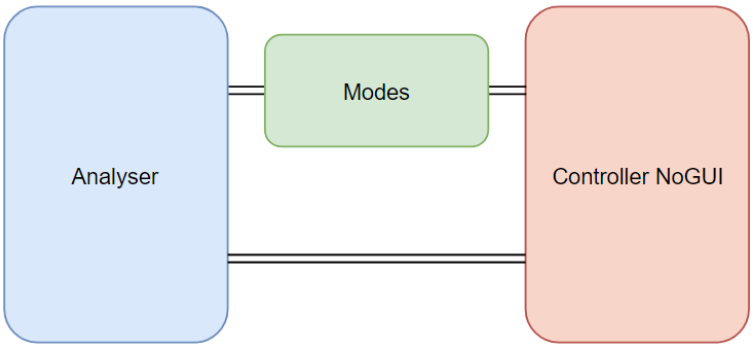
Ak by sme chceli vytvoriť nový mód, tak potrebujeme len vytvoriť novú triedu, ktorá bude dediť od triedy *Mode* a tento mód potom potrebujeme zaregistrovať v *ModesManager* (navyše by bolo ideálne spraviť úpravy na obrazovke, napr. vytvoriť novú rádio možnosť špeciálne pre tento mód). Nakoniec je ešte potrebné správne vytvoriť import tak, aby sa nám pri importe naplnila pamäť dátami rovnako ako pri používaní programu alebo pri exporte.

Existujúce módy máme štyri, každá má inú myšlienku a funkciu pri analýze, bližšie popísané sú v predchádzajúcej kapitole 2.

3.6 Controller NoGUI

Táto komponenta je špeciálne vytvorená pre predpočítanie exportu na výpočetnom clusteri alebo iných externých miestach, ktoré nepodporujú otváranie používateľského rozhrania. *Controller NoGUI* simuluje prácu *Controller* komponenty pri zavolanom exporte, ako vstup požaduje okrem GPT-2 modelu aj nastavenie parametrov (konfiguráciu) pre tento model a vstup pre model. Viac o spúšťaní tejto časti programu je v 4.5.

Controller NoGUI načíta vstupnú konfiguráciu a vstup pre GPT-2 a po prepočítaní vstupu *Analyser* komponentou zavolá Export na jednotlivých módoch.



Obrázek 3.5: Architektúra pre aplikáciu Controller NoGUI

3.7 Tagger

Tagger komponenta je zložená z dvoch tried: *TextTokenizer* a *MorphoditaTagger*. Hlavnou úlohou tejto komponenty je REST API volanie MorphoDiTy na potrebnú úlohu (viac o tomto nástroji v 1.5).

TextTokenizer je trieda, ktorá volá *Tokenize* task na daný text, ktorý sa triede predá. Výstupom je delenie textu na jednotlivé vety. Trieda si prečíta výstup od MorphoDiTy (v notácii *JSON*) a prevedie ho na pole textových reťazcov, v ktorom je každá veta jedna položka.

Táto trieda je volaná módom *Missing sentence dependency* a požiadaná o jednorázové rozdelenie výstupu modelu na jednotlivé vety.

Trieda s názvom *MorphoditaTagger* volá *Tag* task v MorphoDiTe na chcený text. MorphoDiTa vráti zoznam slov rozdelených do viet a ku každému slovu jeho *part-to-speech tag*. MorphoditaTagger je používaná v móde Replacing subword

dependency, kde ju používame na označkovanie možných *podobných* slov. Viac o tomto použití v 2.4.

3.8 Import/Export

Import/Export je komponentou programu, ktorá sa stará o správny import zo súboru a aj export do súboru. Export len vytvára správne pomenovanú zložku, kde sa bude daný export zapisovať.

Na začiatku exportu je vždy celý výstupný text, nasledovaný prázdnym riadkom a pod ním reťazcom *<end of generated text>*. Potom nasleduje ďalší prázdny riadok a po ňom sú vypísané zakódované subwordy z tohto výstupného textu a nasledujú jednotlivé módy svojím názvom a exportom.

O túto časť sa starajú jednotlivé módy samostatne pomocou metódy *export()*, pretože export v našom programe zahŕňa najprv vypočítanie všetkých dosiaľ neexistujúcich dát do cache a až následne vypísanie kompletných dát pre daný mód do súboru na export.

Import spätne začína tak, že si prezrie exportovaný súbor a nájde reťazec *<end of generated text>*. Tento reťazec bol vybraný tak, aby mal model GPT-2 čo najnižšiu pravdepodobnosť toto kľúčové slovo vygenerovať samostatne. Pri importe nás zaujíma hlavne tá časť súboru, kde sa nachádzajú jednotlivé kódy subwordov. Tie sa načítajú a použije sa *GPT2Tokenizer* na ich dekódovanie.

V súbore potom nasledujú exporty jednotlivých módov. Každý mód má odlišnú štruktúru pre export, pretože každý mód funguje inak. Preto je v tejto komponente import pre každý mód napísaný ručne. Pri importe si aplikácia ukladá všetky prečítané dáta do pamäte (cache), aby bolo následne možné vo vizualizátore všetko zobrazovať aj bez výpočtu.

Začiatok súboru, ktorý vygeneroval export a je následne možné importovať (všetky parametre sú pri tomto príklade nastavené na predvolené a použitý je model *gpt2-medium*) je možné vidieť na obrázku 3.6.

```
It was a foggy morning in the Gransbury woods when Dory
   started seeing strange things. At first, it seemed that
    Dory wasn't quite at the end of the rainbow: she'd
   seen something moving about on the other side of the
   road, so she wondered if it was Dory again. But then,
   what would that look like?

As she approached closer, she realized she could make out
   a dark cloud hovering above the grass and bushes. She
   glanced toward where Dory

<end of generated text>

[1026 373 257 19558 1360 3329 287 262 1902 504 10711 16479
    618 360 652 2067 4379 6283 1243 13 1629 717 11 340
   3947 326 360 652 2492 470 2407 379 262 886 286 262
   27223 25 673 1549 1775 1223 3867 546 319 262 584 1735
   286 262 2975 11 523 673 14028 611 340 373 360 652 757
   13 887 788 11 644 561 326 804 588 30 198 198 1722 673
   10448 5699 11 673 6939 673 714 787 503 257 3223 6279
   33627 2029 262 8701 290 37413 13 1375 27846 3812 810
   360 652]


___SUBWORD DISTRIBUTION___
1026;0;[1026];[1]
373;2;[338|318|373|468|1595|561|481|460|2492|2331];
 [0.5623|0.1421|0.1266|0.0271|0.0118|0.0114|0.0093|
 0.0079|0.0065|0.0052]
257;0;[257|262|407|281|655|635|588|691|530|477];
 [0.1356|0.0543|0.0328|0.0291|0.0198|0.0181|0.0166|
 0.0148|0.0121|0.012]
19558;10;[845|922|1049|1263|890|4950|1310|2495|1643|
 1256|19558];[0.032|0.0258|0.0253|0.0154|0.0129|0.0126|
 0.0121|0.0119|0.0115|0.011|9.433637e-05]
...
```

Obrázek 3.6: Príklad *export logu*. Na začiatku sa nachádza celý výstupný text, nasledovaný prázdnym riadkom a pod ním reťazcom *<end of generated text>*. Nasleduje výpis kódov pre jednotlivé zakódované subwordy z tohto výstupu a potom jednotlivé módy s názvom a ich vlastným exportom. Pre Subword distribution ide o kód vygenerovaného subwordu, poradie tohto subwordu, zoznam kódov 10 najpravdepodobnejších subwordov (+ ten vygenerovaný, ak nie je v tejto desiatke) a nakoniec pravdepodobnosti týchto subwordov.

4. Používateľský manuál

Program je určený na vizualizáciu správania jazykového modelu GPT-2 na vygenerovanom texte v *anglickom* jazyku. Konkrétne, viacerými spôsobmi sledujeme to, ako sa menia pravdepodobnosti pre práve generovaný subword na základe predchádzajúcich subwordov (prostredníctvom módov).

Text môže byť vygenerovaný priamo po zadaní výzvy (angl. *prompt*) vybraným modelom GPT-2 (hlavný mód – Generate) alebo môžeme do textového poľa zadať už vopred vygenerovaný text (hlavný mód – Analyse). Navyše je možné zavolať import na už spočítané dáta (viac v sekcii Import 4.3.3).

4.1 Spustenie

Pre spustenie je potrebné mať zariadenie, ktoré spĺňa požiadavky (tie sú viac popísané na konci kapitoly v sekcii 4.6 (Požiadavky)). Ak takéto zariadenie máte aj so správnou verziou jazyku Python (odporúčame verzie 3.8–3.9), inštalovanie jednotlivých balíčkov je jednoduché, pretože všetky (aj so správnymi verziami) sú popísané v súbore *requirements.txt*. Pre samotnú inštaláciu je potom teda už len potrebné do príkazového riadku napísať:

```
pip install -r requirements.txt
```

Vizualizátor je následne možné spustiť jednoducho spustením súboru *controller.py* ako skriptu v jazyku Python. Spustenie cez príkazový riadok by mohlo vyzerať nasledovne:

```
python controller.py
```

4.1.1 Zoznámenie sa s programom

Pre prvé zoznámenie sa s programom odporúčame zapnúť aplikáciu a vyskúšať si import. V zložke *logs* máme už predpočítané niektoré celé exporty.

Prácu s už predpočítanými dátami odporúčame preto, lebo vizualizátor bude okamžite reagovať na všetky vstupy používateľa (pri generovaní a analýze bude kvôli dodatočným výpočtom odozva dlhšia).

Program sa zapne, ako je už popísané na začiatku sekcie, zapnutím súboru *controller.py* v jazyku Python. Ak sa nachádzame v danej zložke, tak príkaz by mal vyzerať nasledovne:

```
python controller.py
```

Pre ďalší postup odporúčame nasledovať návod v sekcii 4.3.3.

4.1.2 Parametre pre vizualizátor

Pri spustení je možné zadať jeden parameter, a tým je špecifický GPT-2 model. Ten je možné uviesť cestou k súboru, kde sa daný model nachádza, alebo názvom modelu v knižnici *transformers*. Názvy modelov pre GPT-2 sú (v zátvorke sa nachádza počet hláv attention) (Hugging Face, 2022):

- gpt2 (12),
- gpt2-medium (24),
- gpt2-large (36),
- gpt2-xl (48).

Ak sa program zavolá bez parametra, predvolený model je *gpt2-medium*. Pre iný model je spustenie cez príkazový riadok nasledovné:

```
python controller.py gpt2
```

Podmienky pre spustenie vlastného modelu

V programe podporujeme aj volanie vlastného modelu. To sa líši od volania podľa názvu jedného z modelov, ktoré majú *huggingface* vo svojom API (popísané vyššie).

Príklad spustenia programu s vlastným modelom:

```
python controller.py cesta/k/adresáru/s/modelom
```

Cesta k adresáru môže byť aj relatívna. V tomto adresári sa musia nachádzať súbory:

- tokenizer.json,
- config.json,
- vocab.json,
- merges.txt a
- tf_model.h5 (samotný model, ale čítateľný pre knižnicu *Tensorflow*, iné modely nebudú fungovať, program hľadá presne tento názov).

4.2 Popis obrazoviek

Program obsahuje 2 obrazovky – hlavnú obrazovku vizualizátoru a obrazovku na nastavenie parametrov (tzv. *Settings* obrazovka). Počiatočná obrazovka pri zapnutí programu je hlavná obrazovka.

4.2.1 Hlavná obrazovka vizualizátoru

Táto obrazovka sa zobrazí po načítaní modelu GPT-2 zo vstupu a až keď je všetko pripravené pre výpočty. Obrazovku je možné vidieť na obrázku 4.1. Najdôležitejšie časti sú očíslované.

Očíslované časti sú:

1. Hlavné menu (momentálne obsahuje len možnosť *File*).
2. „Subword sekcia“: Zobrazuje text, ktorý je výstupom modelu GPT-2 rozdelený do príslušných subwordov.

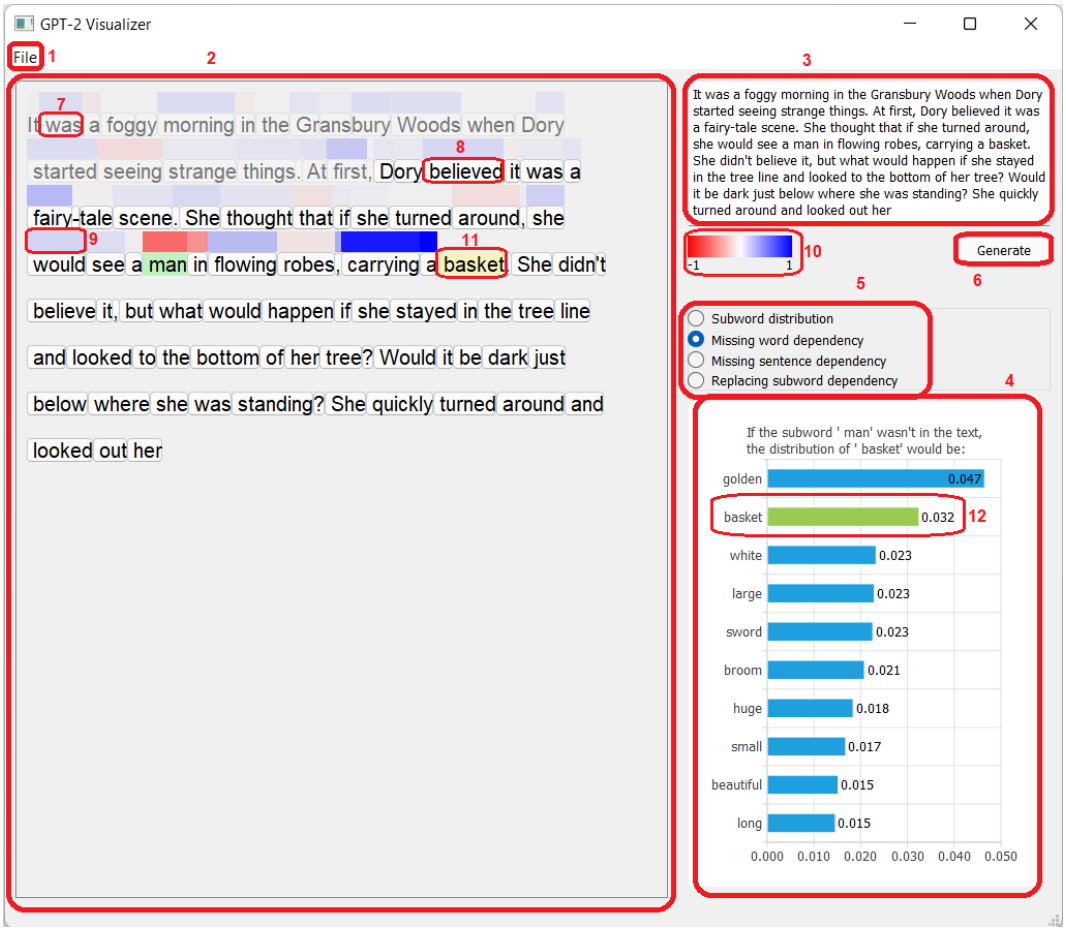
3. Vstupno-výstupné pole: Do tohto poľa sa zadáva text, ktorý je vstupom do modelu a po výpočtoch sa tam objaví výstupný text modelu.
4. Sekcia pre grafy: V tejto sekcii sa zobrazí graf generovaný príslušným módom.
5. Sekcia pre zvolenie módu: Obsahuje prepínacie tlačidlo, ktorým vieme zvoliť mód.
6. Tlačidlo Generate/Analyse: Toto tlačidlo spustí generovanie alebo analýzu podľa toho, aký hlavný mód generovania je zvolený.
7. Neaktívny subword: Na tento subword nie je možné kliknúť, neprebehli preň výpočty (nastáva len ak sa text generoval cez Generate, tieto subwordy sú vstupom pre tento mód).
8. Aktívny subword: Na tento subword je možné kliknúť.
9. Farebný štítok: Tento štítok ukazuje podľa legendy (10) závislosť *vyhodnocovaného* subwordu (11) na subworde pod štítkom.
10. Legenda pre farebné štítky: Legenda ukazuje farebný prechod pre jednotlivé závislosti, záporná závislosť značí, že subword ovplyvňuje generovanie vyhodnocovného subwordu (11) negatívne, kladná závislosť ukazuje pozitívne ovplyňovanie pravdepodobnosti vyhodnocovaného subwordu. Výpočet týchto farieb je rozdielny pre jednotlivé módy a tento výpočet je viac popísaný v kapitole 2
11. Vyhodnocovaný subword.
12. Pravdepodobnosť vyhodnocovaného subwordu v danom móde je v grafe označená inou (zelenou) farbou.

Hlavné menu

Hlavné menu má v čase písania práce len jednu voľbu, a to je *File* menu. Tá má štyri sekcie a spolu päť možností (tie je možné vidieť na obrázku 4.2). Tri z týchto možností majú priradenú klávesovú skratku z hlavnej obrazovky.

Možnosti sú:

1. Settings: otvorí Settings obrazovku (klávesová skratka $Ctrl + Shift + S$),
2. Generate: prepne hlavný mód generovania programu na Generate (klávesová skratka $Ctrl + G$),
3. Analyse: prepne hlavný mód generovania programu na Analyse ($Ctrl + Shift + G$),
4. Export: zapne exportovanie dát pre vygenerovaný text na hlavnej obrazovke (ak tam nie je žiaden vygenerovaný text, nič sa nevykoná),
5. Import: spustí dialógové okno pre výber súboru, ktorý sa naimportuje do programu (ak to nie je súbor v správnom formáte, vypíše hlášku do príkazového riadku).



Obrázek 4.1: Hlavná obrazovka vizualizátora

4.2.2 Nastavenie parametrov

Parametre predané vybranému modelu GPT-2 sa dajú nastaviť na obrazovke *Settings*. Z hlavnej obrazovky sa tam dostaneme tak, že vyberieme v menu na hlavnej obrazovke $File \rightarrow Settings$ alebo na OS Windows stlačíme klávesovú skratku $Ctrl + Shift + S$.

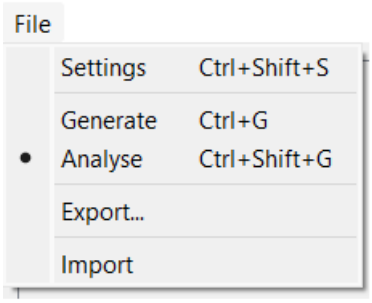
Názvy týchto parametrov sú podobné názvom parametrov, ktoré sa modelu GPT-2 predávajú, s miernymi vizuálnymi zmenami. Prvé písmeno je kapitalizované a znak „_“ je nahradený medzerou, teda napríklad parameter s názvom „max_length“ je prepísaný na „Max length“.

V hlavnom móde Generate sa predávajú všetky parametre, v hlavnom móde Analyse sa parameter „Max length“ ignoruje.

Na obrázku 4.3 môžeme vidieť, ako táto obrazovka vyzerá.

Popis parametrov a ich význam

- *Max length*: Maximálna dĺžka sekvencie na generovanie.
- *Temperature*: Hodnota použitá na škálovanie pravdepodobností ďalšieho tokenu. Nižšia hodnota znamená menej náhodnosti pri generovaní, model teda bude viac deterministický a pri výbere bude ešte viac preferovať slová s vyššou pravdepodobnosťou (Isozaki, 2019).



Obrázek 4.2: *File menu* na hlavnej obrazovke

- *Top p*: Ak je táto hodnota nastavená na číslo menšie ako 1, tak do generovania sa vyberie maximálny počet najpravdepodobnejších tokenov, ktoré spolu dávajú menšiu pravdepodobnosť, ako táto hodnota (ak je Top p = 1, nevykonáva sa žiadny takýto výpočet, hodnota väčšia ako 1 nedáva zmysel, pretože to je súčet pravdepodobností).

  Napríklad, ak by sme mali Top p = 0,95 a pravdepodobnosti pre generovanie v slovníku by boli 0,6, 0,3 a 0,1; do generovania by sa zvolili len prvé 2 slová (pretože $0{,}6 + 0{,}3 < 0{,}95$, ale $0{,}6 + 0{,}3 + 0{,}1 > 0{,}95$).

- *Repetition penalty*: Parameter pre penalizáciu opakovane vygenerovaných tokenov. Ak je hodnota rovná 1, žiadna penalizácia sa neaplikuje. Vyššia hodnota penalizuje opakovane generované tokeny. Táto penalizácia je namodelovaná podľa článku *Ctrl: A conditional transformer language model for controllable generation* (Keskar a kol., 2019).

  Tento článok navrhuje presný spôsob, akým sa penalizujú nedávno vygenerované tokeny a prichádza aj s presným vzorcom, ktorý upravuje pravdepodobnosť generovania takýchto tokenov.

- *Top k*: Počet tokenov s najvyššou pravdepodobnosťou, ktoré si model nechá na tzv. „top-k-filtering“. Určuje počet slov s najvyššou pravdepodobnosťou, z ktorých bude model vyberať ďalší token (subword).

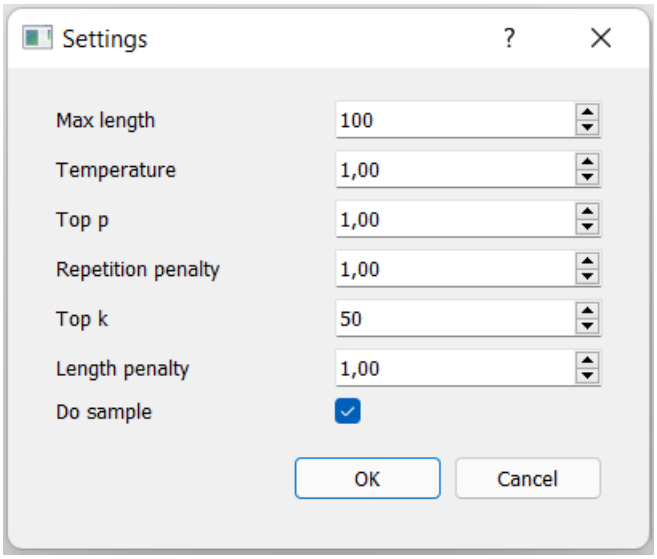
- *Length penalty*: Exponenciálna penalizácia na dĺžku generovanej sekvencie. Ak je hodnota menšia ako 1, model sa snaží generovať kratšie sekvencie. Ak je hodnota vyššia ako 1, model sa snaží generovať dlhšie sekvencie.

- *Do sample*: Ak je parameter (začiarkavacie políčko) zvolený, vykoná sa sampling (náhodný výber vážený podľa pravdepodobnosti), inak sa len deterministicky vyberie token s najvyššou pravdepodobnosťou.

Popisy týchto parametrov sú preložené z dokumentácie v zdrojovom kóde metódy `generate` z knižnice *transformers*, ktorú naše modely GPT-2 používajú[1] (Hugging Face, 2022).

GPT-2 model vo vnútri pracuje s tokenmi, ktoré tokenizer preloží na jednotlivé subwordy.

[1]Presný zdrojový kód s popisom metódy *generate()* je možné nájsť na `https://github.com/huggingface/transformers/blob/3e8d17e66d223e681a08272351368ab077560d1d/src/transformers/generation_tf_utils.py#L402`



Obrázek 4.3: Obrazovka pre nastavenie parametrov (*Settings*)

4.3 Použitie

Funkčnosť programu sa delí na 2 časti: získanie generovaného textu a samotná analýza závislostí. Získať generovaný text môžeme 3 spôsobmi:

- text môžeme vygenerovať pomocou vizualizátora (viac v sekcii 4.3.1),
- môžeme zadať už vygenerovaný text z externých zdrojov (napríklad, ak nám text generuje iná aplikácia alebo online GPT-2 generátor) (viac v 4.3.2),
- môžeme použiť import nejakého súboru, ktorý bol exportovaný naším programom, teda všetky informácie boli vopred predpočítané (viac v 4.3.3).

Následne začína druhá časť, ktorou je analýza závislostí. Tá je bez ohľadu na generovaný text stále rovnaká.

4.3.1 Generovanie textu zo vstupu

Tento spôsob získania generovaného textu je asi najjednoduchší, pretože sa dá vykonať priamo v v našej aplikácii. Najprv potrebujeme prepnúť predvolený hlavný mód z Analyse na Generate, to vieme spraviť kliknutím do File menu na hornej lište aplikácie a zvolením Generate. Tejto činnosti je na operačných systémoch Windows priradená klávesová skratka $Ctrl + G$.

Následne potrebujeme vybrať správne parametre generovania. Na to si potrebujeme otvoriť obrazovku Nástroje (*Settings*, popísaná v sekcii 4.2.2), to je možné urobiť vybraním File → Settings na hornej lište (alebo na OS Windows klávesovou skratkou $Ctrl + Shift + S$). Pri generovaní sa nás týkajú všetky parametre (bližšie popísané v 4.2.2).

Nakoniec ostáva zadať vpravo hore do textového poľa vstup pre GPT-2 model. Dôležité je, aby bol tento text kratší ako parameter *Max Length* nastavený na obrazovke *Settings*. Vstup prechádza cez tokenizer (popísaný v sekcii o kódovaní textu – 1.2.3), ktorý rozdelí vstup na jednotlivé subwordy. Ak zadávame vstup v

angličtine, tak počet subwordov je približne rovný počtu slov a interpunkčných znamienok na vstupe, niektoré slová (hlavne vlastné mená), môžu byť rozdelené do viacerých subwordov. Obvykle pri klasickej *Max Length* nastavenej na hodnotu okolo 100 a viac postačuje zadať na vstup jednu-dve kratšie vety a začiatok ďalšej (ideálne menej ako 30 slov, chceme, aby to vygenerovalo aj nejaký text, nie len pár slov).

Výpočet sa následne začne kliknutím na tlačidlo Generate pod textovým poľom. Kým výpočet prebieha, tlačidlo Generate je zošednuté.

Príklad použitia

1. Spustíme *controller.py*,
2. v hlavnom menu (na hornej lište) vyberieme File → Settings,
3. zmeníme *Max Length* na 125, *Top p* na 0,85 a *Repetition penalty* na 1,05,
4. potvrdíme kliknutím na tlačidlo Ok,
5. v hlavnom menu vyberieme File → Generate,
6. do textového poľa vpravo hore zadáme text: *The highest-paid Norwegian limousine driver of all time,*
7. spustíme kliknutím na Generate.

4.3.2 Analýza vopred vygenerovaného textu

Tento spôsob sa hodí hlavne vtedy, ak máme vygenerovaný text z iných zdrojov a zaujímalo by nás, prečo sa takto vygeneroval.

Analýza je predvolený hlavný mód, takže nie je potrebné ho nastavovať. Ak sme ale predtým používali Generate mód, tak Analyse je možné nastaviť podobne, buď File → Analyse, alebo klávesovou skratkou $Ctrl+Shift+G$ na OS Windows.

Následne je potrebné nastaviť parametre, tie sa nastavujú podobne ako pri generovaní textu, File → Settings alebo $Ctrl + Shift + S$ (pre OS Windows). V tomto okne nás nezaujíma prvý parameter *Max Length*, pretože ten sa vždy nastaví podľa textu na vstupe. Je ale veľmi dôležité nastaviť ostatné parametre presne tak, ako boli nastavené v čase generovania. Obrazovka *Settings* ukazuje všetky parametre tak, ako sú predvolené v knižnici *transformers* pre generovanie textu, takže ak hodnotu pre nejaký z parametrov nepoznáme, s veľkou pravdepodobnosťou ide o predvolenú hodnotu.

Nakoniec ostáva zadať vopred vygenerovaný text do textového poľa vpravo hore. Je potrebné si byť istý, že tento text vygeneroval daný model GPT-2 počas jedného generovania, pretože GPT-2 vie vygenerovať maximálne 1024 subwordov, čo je maximum, ktorý náš model dokáže zobrať, dlhšie texty program nevezme.

Výpočet sa začne kliknutím na tlačidlo Analyse pod textovým poľom. Ak tam to tlačidlo nie je, ale je tam tlačidlo s nápisom Generate, tak pravdepodobne nastala chyba v prvom kroku a je potrebné nastaviť hlavný mód na Analyse.

Výpočet je v tomto prípade len simulácia behu GPT-2, kde sa modelu zadá prvý subword textu, následne sa uloží skóre, ale namiesto vybrania náhodného

alebo najlepšieho subwordu sa tam vnúti ďalší subword v texte. Týmto dosiahneme to, že vieme, nad akými subwordmi model „rozmýšľal“ pri generovaní pôvodného textu, ale vždy zvolíme subword, ktorý zvolil pôvodný model pri generovaní. To znamená, že tento spôsob je deterministický a na rovnakom vstupe vždy vráti rovnaký výsledok, ak sa nastavia všetky parametre správne.

Príklad použitia

V zložke *logs* sa nachádzajú niektoré už vopred vygenerované export logy. Každý log začína celým vygenerovaným textom, takže pre vyskúšanie tohto spôsobu zobrazenia dát nie je potrebné generovať externým spôsobom vlastný text.

Najjednoduchší spôsob použitia vyzerá nasledovne:

1. spustíme *controller.py*,
2. otvoríme si zložku *logs* a súbor *generated_log.txt*, z ktorého si vykopírujeme všetky riadky zo začiatku až po riadok s textom *<end of generated text>* a súbor zavrieme,
3. vo vizualizátore nakopírujeme tento text do textového poľa vpravo hore,
4. klikneme na Analyse.

Poznámka: V tomto prípade môžeme vynechať krok o nastavovaní parametrov, pretože export log v súbore *generated_log.txt* je vygenerovaný s predvoleným nastavením parametrov.

4.3.3 Import

Import je tretí zo spôsobov, ako sa dá získať generovaný text do vizualizátoru a k analýze. Tento spôsob vyžaduje export log vygenerovaný týmto programom (viac o exporte v 4.3.5). Výhodou tohto spôsobu zobrazenia dát je celková rýchlosť, pretože dáta už sú vypočítané a my ich len zobrazujeme. To znamená, že dáta sa do pár sekúnd zobrazia a všetky kalkulácie sa už len načítavajú.

Importovanie export logu do aplikácie urobíme jednoducho zvolením cez File → Import na hornej lište. Otvorí sa nám dialógové okno prieskumníka, ktoré od nás očakáva cestu k export logu, tzn. je potrebné nájsť a vybrať konkrétny log, ktorý chceme importovať. Po zvolení súboru sa začne tento súbor importovať.

Príklad použitia

V zložke *logs* sa nachádzajú niektoré už exportované logy, ktoré si vieme naimportovať a vyskúšať, ako fungujú. Súbory *analysed_log.txt* a *generated_log.txt* sú vyexportované z hlavných módov Analyse, resp. Generate. Tieto súbory sú priamo vygenerované naším programom, a to tak, že najprv sa zadal vstup pre generovanie textu, prebehol export a následne sa na vygenerovanom texte zavolala analýza textu a export. Tieto súbory by preto mali byť veľmi podobné a je možné na nich demonštrovať rozdiely medzi generovaním a analýzou bez toho, aby sme museli robiť na počítači nejaké výpočty.

Konkrétny príklad použitia vyzerá nasledovne:

1. spustíme *controller.py*,
2. v hlavnom menu (na hornej lište) vyberieme File → Import,
3. v dialógovom menu otvoríme zložku *logs*,
4. vyberieme z nej ľubovoľný súbor (napríklad *analysed_log.txt*) a klikneme na tlačidlo Otvoriť.

4.3.4 Analýza závislostí

Analýza závislostí prebieha rovnako bez ohľadu na to, ako získame generovaný text. Táto časť začína, keď sa na ľavej strane obrazovky zobrazí generovaný text vo forme tlačidiel, na ktoré je možné kliknúť. Napravo hore, v poli pre vstup, sa objaví vygenerovaný text na kopírovanie.

Najprv si zvolíme mód, v ktorom chceme analyzovať. Módy je možné zvoliť na pravej strane pod poľom určeným na vstup. Mód sa aktivuje kliknutím na jeden z klikateľných subwordov, následne nastáva výpočet analýzy pre vybraný subword vo vybranom móde. Tento výpočet nastáva vždy len raz pre každú dvojicu subword-mód, následne sa uloží do cache, odkiaľ sa budú vypočítané informácie v prípade potreby vyťahovať.

Každý mód má dve možné činnosti: buď zobrazí farebné štítky nad niektorými z predchádzajúcich subwordov, alebo zobrazí v pravej dolnej časti graf. Ak mód zobrazí farebné štítky nad subwordmi, na tie je možné kliknúť a prevedie sa ďalší výpočet.

Pozor, ak sa následne vyberie iný mód, tak sa všetky farebné štítky deaktivujú a nebude možné na nich kliknúť. Pre informácie z tohto výpočtu je opäť potrebné zvoliť pôvodný mód a vybrať pôvodne počítané slovo.

Príklad použitia

1. Spustíme *controller.py*,
2. v hlavnom menu (na hornej lište) vyberieme File → Import,
3. v dialógovom menu otvoríme zložku *logs*,
4. vyberieme z nej súbor *generated_log.txt* a klikneme na tlačidlo Otvoriť,
5. vpravo pod textovým poľom vyberieme *Missing sentence dependency*,
6. na ľavej strane, kde sú tlačidlá, klikneme na tlačidlo so slovom *hovering* v predposlednom riadku (nad jednotlivými vetami vyššie by sa mali zobraziť farebné štítky a slovo *hovering* by sa malo prefarbiť na žlto),
7. klikneme na modrý štítok nad prvou vetou (vpravo dole by sa mal zobraziť graf a celá prvá veta by sa mala prefarbiť na zeleno),
8. teraz zvolíme na pravej strane mód *Subword dependency* a klikneme na slovo *glanced* na konci predposledného riadku (malo by sa zvýrazniť slovo *glanced* a na pravej strane by sa mal zobraziť graf ukazujúci 10 najpravdepodobnejších slov z pravdepodobnostnej distribúcie pri generovaní tohto subwordu, spolu s tým, akú pravdepodobnosť malo vygenerovanie slova *glanced*).

Poznámka: Popis toho, čo znamenajú farebné štítky v ktorom móde a čo ukazujú jednotlivé grafy je rozpísaný v sekcii 4.4 (Popis jednotlivých módov). Môže sa to medzi jednotlivými módmi líšiť.

4.3.5 Export

Ďalšou, separátnou funkcionalitou programu je export. Export prebieha v dvoch krokoch. Najprv sa zadá výpočet údajov z každého módu a následne sa všetky dáta pre každé slovo plus údaje o výstupe modelu zapíšu do súboru.

Módy si všetky výpočty prevedené vo vizualizátore ukladajú do vlastnej časti cache. Pri zavolaní exportu sa všetky nevyplnené dáta z cache vyplnia (napríklad dáta pre subword, na ktorý sme neklikli). To je dôvod, prečo sa po úspešnom exporte už nezakladajú žiadne výpočty a program zobrazuje všetky údaje hneď.

Príklad použitia

1. Spustíme *controller.py*,
2. v hlavnom menu (na hornej lište) vyberieme File → Generate,
3. do textového poľa vpravo hore zadáme text: *The highest-paid Norwegian limousine driver of all time,*
4. spustíme kliknutím na Generate,
5. počkáme, kým sa vygeneruje text a zobrazí sa v ľavej časti vizualizátora,
6. v hlavnom menu vyberieme File → Export, čím zapneme exportovanie.

Predvolené nastavenie generovania textu je na 100 subwordov, export počíta všetky dáta, ktoré sme schopní získať pre každý z módov. To by znamenalo vyše 10 000 separátnych cyklov modelu (za cyklus modelu považujeme generovanie ďalšieho slova; experimentálne sme zistili, že to je tá časovo najnáročnejšia fáza), čo na menej výkonných počítačoch môže trvať aj vyše hodiny.

Preto majú export funkcie v jednotlivých módoch celkom dôkladný výpis súčasného stavu exportu na príkazový riadok, takže by sme mali byť stále schopní určiť, v akom štádiu sa export približne nachádza.

Formát exportovaného súboru

- Na začiatku súboru je vždy výstupný text z modelu, presne tak, ako sa mu predal, resp. ako si vygeneroval.
- Výstup vždy končí prázdnym riadkom a textom *<end of generated text>*.
- Nasleduje presné rozloženie textu na subwordy, ktoré sú predstavované svojím indexom v tokenizeri oddelené medzerou a celé obalené do hranatých zátvoriek (napr. *[1026 373 257 19558 1360 3329]*).
- Potom prichádzajú exporty jednotlivých módov, každý mód začína kľúčovým reťazcom. Módy sú v poradí:

1. Subword distribution,
2. Missing word dependency,
3. Missing sentence dependency a
4. Replacing subword dependency.

Poznámka: Výstupný text z modelu môže obsahovať úplne ľubovoľnú kombináciu všelijakých znakov v kódovaní Unicode, takže vždy existuje nejaká šanca, že model pri generovaní vygeneruje prázdny riadok a za ním text *<end of generated text>*, čím by sa znemožnil správny import takéhoto súboru. Pre minimalizáciu tejto pravdepodobnosti sme vybrali práve reťazec *<end of generated text>*, v bežnom modeli by sa tento reťazec rozdelil do šiestich subwordov, čo je v našom prostredí celkom veľa. Hlavne, ak uvážime, že modely kvôli rýchlosti pri náhodnom generovaní pomocou samplingu, vyberajú nasledujúce slovo len z *Top K* slov s najvyšším skóre (Top K je predvolene nastavené na 50). Vo väčšine prípadov sa subword < nevyskytuje takto vysoko.

Navyše sme sa snažili vymyslieť text, ktorý pre nás dáva zmysel, ale model by mohol preferovať v mohutnej väčšine prípadov iné subwordy, napríklad *</endoftext/>*, čo je značka pre token, ktorý sa používa pri učení GPT-2 tokenizera a značí, že ďalej už text nepokračuje. Reťazec *</endoftext/>* sa nachádza celkom často na webe (hlavne, keď sa spomína GPT-2), čo znamená, že sa určite nachádzal aj v nejakých testovacích dátach pre tieto modely (raz nám ho pri testovaní jeden z modelov dokonca vygeneroval v textovej podobe). Takže je predpoklad, že aj keď sa vygeneruje znak <, tak distribúcia pre nasledujúci subword bude s vysokou pravdepodobnosťou obsahovať tento súbor alebo ďalšie výrazy, modelu známe z testovania.

Ukážkové logy sa nachádzajú v zložke *logs* a všetky majú formát bližšie popísaný v sekcii 3.8. Formát je potrebné precízne dodržať kvôli importu, každá zmena môže export log poškodiť natoľko, že ho nebude možné importovať do programu.

4.4 Popis jednotlivých módov

Módy sú spôsoby, ktorými môžeme sledovať závislosti generovania jedného subwordu na inom subworde, ktoré sa v texte už nachádza. Každý mód má naimplementované svoje správanie po kliknutí na tlačidlo so subwordom (primárna funkcia) a po kliknutí na nejaký zo zafarbených obdĺžnikových štítkov nad subwordmi (sekundárna funkcia).

4.4.1 Subword distribution

Subword distribution je ten najjednoduchší z módov v projekte. Tento mód si uchováva subword a jeho pravdepodobnosť pre 10 najpravdepodobnejších subwordov na danej pozícii v texte. Ak sa vygenerovaný subword nenachádza v tejto desiatke, tak sa pridá na koniec aj so svojou pravdepodobnosťou, takže nakoniec sa tam nachádza 11 párov údajov. Pravdepodobnosť subwordov je daná skóre, ktoré subword dostal po poslednej softmax vrstve modelu, zaokrúhlená na 4 desatinné miesta (ak to takto obsahuje prvú platnú číslicu, teda pravdepodobnosť je väčšia

ako 0,0001). Pri generovaní (ak je zapnutý *sampling*) náhodne vyberie jedno zo subwordov vážene podľa pravdepodobnosti, ktorá im bola priradená[2].

Kliknutie na subword

Po kliknutí na subword sa na pravej strane zobrazí graf, ktorý ukazuje 10 najpravdepodobnejších subwordov a pravdepodobnosť vygenerovaného subwordu (ak sa toto slovo nenachádza medzi desiatimi najpravdepodobnejšími) pre danú pozíciu v texte.

Kliknutie na farebný štítok

Tento mód nevyfarbuje štítky nad subwordmi, takže táto funkcionalita nemá zmysel.

4.4.2 Missing word dependency

V *Missing word dependency* móde sledujeme, ako sa mení rozmýšľanie modelu v závislosti od prítomnosti jednotlivých subwordov. Pozorujeme teda, ako sa zmení pravdepodobnosť vygenerovaného subwordu (a aj pravdepodobnostná distribúcia pre daný subword), ak nejaký z predchádzajúcich subwordov z textu vynecháme.

V tomto móde si uchovávame údaje pre každú dvojicu subwordov v texte, taktiež si pre každý subword uložíme aj jeho pôvodnú pravdepodobnosť.

Údaje, ktoré si pre každú takúto dvojicu slov ukladáme sú pravdepodobnosť vyhodnocovaného subwordu a distribúcia 10 najpravdepodobnejších subwordov a vygenerovaného subwordu (podobne ako pri Subword distribution (v sekcii 4.4.1 vyššie).

Motivácia, prečo sme zvolili tento mód, je vysvetlená v kapitole 2.2.

Kliknutie na subword

Po kliknutí na subword sa spustí výpočet všetkých možných viet, kde je tento subword v úlohe *vyhodnocovaného subwordu*. Vygenerujú sa teda vstupy do modelu, kde si rolou *chýbajúceho* subwordu prejde každý subword nachádzajúci sa v texte pred tým, na ktorý sme klikli.

Pre každú takúto dvojicu si zistíme pravdepodobnosť vyhodnocovaného subwordu, ak ten chýbajúci bol zo vstupu do modelu vynechaný. Túto pravdepodobnosť porovnáme s pôvodnou pravdepodobnosťou vyhodnocovaného subwordu pri generovaní pôvodného textu.

Rozdiel týchto dvoch pravdepodobností potom použijeme na vytvorenie farebného štítku nad *chýbajúcim* subwordom. Ak je tento rozdiel kladný, zafarbí

[2]V praxi to funguje ešte o trošku zložitejšie, modelu sa dáva parameter *Top K*, ktorý určuje koľko najvyššie hodnotených subwordov prechádza do samplingu, kde sa im prepočíta pravdepodobnosť tak, aby dokopy tieto subwordy mali pravdepodobnosť 100 % a potom sa vykoná sampling (náhodné vybranie jedného subwordu vážené podľa pravdepodobnosti). Robí sa to kvôli celkovej rýchlosti samplingu. Náš program ale používa tú prvotne vygenerovanú pravdepodobnosť.

sa štítok na červeno. Ak je rozdiel záporný, zafarbí sa na modro. Nepriehľadnosť farby (angl. *opacity*) je určená veľkosťou rozdielu. Čím väčší rozdiel, tým nepriehľadnejšia farba.

V rámci výpočtu sa vyhodnocovaný subword prefarbí na žlto.

Kliknutie na farebný štítok

Farebný štítok nad subwordom ukazuje, ako sa zmenila pôvodná pravdepodobnosť *vyhodnocovaného* subwordu (zafarbeného na žlto), ak by sa *chýbajúci* subword (pod daným farebným štítkom) v generovanom texte nenachádzal.

Kliknutím na takýto farebný štítok sa spustí funkcia, ktorá nám na pravej strane ukáže graf podobný grafu zo Subword distribution módu. Ukáže nám 10 najpravdepodobnejších subwordov a vyhodnocovaný subword aj s ich pravdepodobnosťami, keď *chýbajúci* subword je vynechaný.

Hlavnou myšlienkou je ukázať, nad ktorými subwordmi model GPT-2 uvažuje a akú váhu im prikladá, ak je konkrétny subword vynechaný.

V rámci výpočtu sa *chýbajúci* subword z dát v grafe zafarbí na zeleno.

4.4.3 Missing sentence dependency

Missing sentence dependency funguje veľmi podobne ako mód Missing word dependency. Jediný rozdiel je, že namiesto slov vynechávame celé vety.

Mód teda sleduje zmenu v skóre vyhodnocovaného subwordu, ak by niektorá z predchádzajúcich viet (označme ju ako *chýbajúca* veta) na vstupe do modelu chýbala.

Pri behu módu si uchovávame údaje o dvojici veta-subword, kde veta sa vždy nachádza v texte pred subwordom. Údaje, ktoré si uchovávame, sú tiež podobné minulým sekciám: pravdepodobnosť vyhodnocovaného subwordu a distribúcia 10 najpravdepodobnejších subwordov a vygenerovaného subwordu.

Motivácia za zvolením tohto módu je popísaná v sekcii 2.3.

Kliknutie na subword

Po kliknutí na subword sa spustí výpočet všetkých možných platných dvojíc veta-subword pre daný subword. Výpočet neprebieha, ak je vyhodnocované slovo v prvej vete alebo ako prvé slovo druhej vety.

Pre každú takúto dvojicu si zistíme pravdepodobnosť *vyhodnocovaného* subwordu, ak odstránime zo vstupu do modelu *chýbajúcu* vetu. Rozdiel tejto pravdepodobnosti a pôvodnej pravdepodobnosti *vyhodnocovaného* subwordu potom použijeme na vytvorenie farebného štítku nad všetkými subwordmi patriacimi do *chýbajúcej* vety.

Ak je tento rozdiel kladný, štítok sa zafarbí na červeno, ak je záporný, zafarbí sa na modro. Nepriehľadnosť je opäť určená veľkosťou rozdielu týchto pravdepodobností. Čím väčší rozdiel, tým nepriehľadnejšia farba.

Na konci výpočtu sa vyhodnocovaný subword zafarbí na žlto.

Kliknutie na farebný štítok

Kliknutím na farebný štítok sa spustí funkcia, ktorá na pravej strane zobrazí graf s podobnou logikou, ako ten v Missing word dependency (odkaz na farebný štítok). Ukáže sa 10 najpravdepodobnejších subwordov a vyhodnocovaný subword aj s ich pravdepodobnosťami, ak *chýbajúca* veta chýba.

Hlavnou myšlienkou je ukázať, nad ktorými subwordmi model GPT-2 premýšľa, ak by sa v texte neukázala celá jedna veta (ale text pred aj po nej ostal nedotknutý).

Na konci výpočtu sa celá *chýbajúca* veta z dát v grafe zafarbí na zeleno.

4.4.4 Replacing subword dependency

Replacing subword dependency je najzložitejší mód v programe. Slová na vstupe do modelu nahrádzame inými slovami, ktoré zohrávajú podobnú rolu vo vete (majú podobný *part-of-speech tag* (1.5)) a následne sledujeme zmenu v pravdepodobnosti nasledujúcich subwordov. Tieto slová označujeme ako *podobné* slová (danému *zamieňanému* slovu).

Motivácia za vytvorením tohto módu, aké slová sa volia a ako určujeme, akým slovným druhom vo vete dané slovo je, je popísaná v 2.4.

Podobne ako v Missing word dependency si uchovávame údaje o dvojiciach subwordov. Subword nachádzajúci sa v texte skôr označme za zamieňaný subword, ten druhý nazvime vyhodnocovaným subwordom.

Konkrétne údaje, ktoré si uchovávame, sú *podobné subwordy*, ktoré sme zamenili za *zamieňaný* subword a ich efekt na *vyhodnocovaný* subword, teda akú pravdepodobnosť má vyhodnocovaný subword, ak sa v texte namiesto zamieňaného subwordu nachádza iný. Taktiež si ešte uložíme pôvodnú pravdepodobnosť vyhodnocovaného subwordu a priemernú pravdepodobnosť vyhodnocovaného subwordu cez vstupy obsahujúce nové subwordy, ktoré sme pri zamieňaní použili.

Kliknutie na subword

Po kliknutí na subword sa spustí výpočet pre všetky možné vstupy, kde sa tento subword nachádza v úlohe *vyhodnocovaného* subwordu. Pre každý subword nachádzajúci sa vo vygenerovanom texte skôr sa skontroluje, či je daný subword platným *zamieňaným* subwordom podľa reštrikcií (popísaných v 2.4.2). Ak nie je, tak sa výpočet preskočí, inak pokračuje ďalej.

Po výpočte dostaneme zoznam *podobných* slov, ktorými sme nahradili zamieňaný subword, spolu aj s pravdepodobnosťou vyhodnocovaného subwordu pre každú takúto zámenu.

Vypočítame si priemernú pravdepodobnosť cez *podobné* slová a tú porovnáme s pôvodnou pravdepodobnosťou vyhodnocovaného slova pri pôvodnom generovaní (so zamieňaným slovom).

Ak je rozdiel týchto dvoch pravdepodobností kladný, štítok nad zamieňaným subwordom zafarbíme na červeno, v opačnom prípade na modro. Nepriehľadnosť farby (*opacity*) je určená veľkosťou rozdielu, čím väčší rozdiel, tým sýtejšia (nepriehľadnejšia) je farba.

V rámci výpočtu sa vyhodnocovaný subword prefarbí na žlto.

Kliknutie na farebný štítok

Farebný štítok nad subwordom ukazuje, ako by sa v priemere zmenila pôvodná pravdepodobnosť *vyhodnocovaného* subwordu, ak sa *zamieňaný* subword zamenil za niektoré z *podobných* slov.

Kliknutím na takýto farebný štítok sa spustí funkcia, ktorá v pravej dolnej časti obrazovky zobrazí horizontálny stĺpcový graf. Tento graf ukazuje všetky *podobné* slová a pravdepodobnosť *vyhodnocovaného* subwordu s danými slovami, keď nahradili *zamieňaný subword*.

Hlavnou myšlienkou grafu je ukázať, ako sa mení myslenie daného modelu GPT-2 nad konktrétnym slovom v závislosti od toho, aké slovo je použité v texte pred tým.

V rámci výpočtu sa *zamieňaný* subword zafarbí na zeleno.

4.5 Počítanie exportu bez užívateľského rozhrania

Program taktiež obsahuje ďalší spustiteľný súbor v jazyku Python s názvom *controller_nogui.py*. Tento súbor je vytvorený špeciálne na vytváranie export logov bez vizualizácie. Je určený pre väčšie modely *(gpt2-large, gpt2-xl)* a pre počítanie na vzdialených prostrediach (servery, výpočetné clustre a pod.).

Cieľom súboru je oddeliť výpočet náročný na zdroje od samotnej vizualizácie pomocou export logu, ktorý je po vygenerovaní možné importovať do vizualizéru (odkaz na import).

Súbor funguje nasledovne:

1. načíta konfiguráciu a vstup, zistí hlavný mód, v ktorom má prebiehať výpočet (Generate/Analysis) a pripraví model GPT-2,
2. spustí hlavný mód s danými parametrami,
3. na výsledku spustí export, ktorý uloží do zložky *logs*.

4.5.1 Použitie

Na vstupe sú očakávané dva parametre, tretí je voliteľný:

1. vstupný konfiguračný súbor obsahujúci parametre pre model GPT-2,
2. vstupný text,
3. (voliteľný) model GPT-2.

Signatúra súboru je teda nasledovná:

```
python controller_nogui.py path/config.txt path/input.txt
[modelname|modelpath]
```

Príklady použitia

V zložke *configs* sú pripravené nejaké základné konfiguračné súbory pre analýzu (posledné písmeno je *a*) aj generovanie (posledné písmeno je *g*). Rovnako sú v zložke *prompts* pripravené vstupy do GPT-2 modelu, ich názvoslovie je podobné tomu pri konfiguráciách. Oba tieto druhy súborov sú len príklady a je možné ich použiť ako šablónu pre vytvorenie vlastných konfigurácií a vstupov pre modely GPT-2.

Konkrétne príklady použitia sú nasledovné:

```
> python controller_nogui.py configs/configa.txt
prompts/prompta1.txt
> python controller_nogui.py configs/configg.txt
prompts/promptg1.txt gpt2
```

4.6 Požiadavky

Program je vypracovaný z väčšej časti v jazyku Python, dôležité použité knižnice sú:

- *NumPy* a *TensorFlow*, použité na matematické výpočty;
- *PyQt5*, použitá na vytvorenie a prácu s obrazovkami;
- *requests*, použitá na REST API volania;
- *transformers* od *Hugging Face* (Hugging Face, 2022), ktorá obsahuje jazykové modely GPT-2 a implementovaný tokenizer použité v tomto projekte.

Obrazovky boli vytvorené v programe Qt Designer.

Program by mal byť spustiteľný na všetkých zariadeniach, ktoré dokážu nainštalovať všetky potrebné knižnice, ktoré sú popísané v súbore *requirements.txt*. Aj napriek tomu odporúčame použiť Python v major verziách 3.8 alebo 3.9.

Program je vyvinutý a plne otestovaný na operačnom systéme Windows, ktorý je odporúčaný pre spustenie programu. Testy prebiehali aj na OS Linux a aj napriek tomu, že všetky testy uspeli, neodporúčame program spúšťať na tomto operačnom systéme, nakoľko program nebolo možné plne otestovať v tomto prostredí.

Z hardvérového hľadiska musíme počítať s tým, že do RAM je potrebné uložiť výpočty počas behu programu a podľa nastavenia *tensorflow* sa musí celý model neurónovej siete načítať buď do RAM, alebo na grafickú kartu.

Základné modely z knižnice *transformers* majú veľkosť od 1 GB (gpt2) až do 5,8 GB (gpt2-xl). Samotné výpočty by ani pri najväčšom možnom výpočte nemali presahovať 400 MB. Dôvodom je, že GPT-2 vie vygenerovať maximálne 1024 subwordov (*podslov*) počas jedného generovania a vstup do modelu sa počíta do tohto počtu (viac v sekcii 1.2).

Pre plnú funkcionalitu teda odporúčame použiť zariadenie s aspoň 8 GB RAM alebo 4 GB RAM + 8 GB GPU.

5. Analýza

5.1 Popis analýzy

Analýzu, ktorú sme vykonávali na našej vizualizačnej aplikácii, môžeme nazvať základnou analýzou, kde viac ako samotné modely GPT-2 sme skúmali metódy predstavené v kapitole 2 (Metódy) – módy našej aplikácie. Aj napriek tomu vieme vytvoriť pár hypotéz aj o GPT-2.

Pri analýze sme použili 43 vygenerovaných textov na modeli GPT-2 Medium. Tieto texty sme vygenerovali zo štyroch rôznych uvádzacích textov pre model (angl. *prompt*[1]), pri ktorých bola snaha prinútiť model, aby generoval rôzne druhy literárnych žánrov.

5.1.1 Parametre

GPT-2 Medium dostal vždy rovnaké parametre pre generovanie. Všetky až na dve boli nastavené na predvolené hodnoty, výnimku tvorila *Max length* (maximálna dĺžka generovania, po ktorej má model prestať), ktorú sme nastavili na *150* a *Top p* (obmedzuje náhodný výber tokenov), ktoré sme nastavili na *0,95*. Všetky parametre (aj s predvolenými hodnotami) sú bližšie popísané v kapitole 4.2.2.

Maximálnu dĺžku generovania bolo potrebné nastaviť, pretože inak by model mohol generovať text dlhý až 1024 subwordov, čo by samo o sebe trvalo vyše 6,5-krát dlhšie. Tri zo štyroch našich módov robia výpočty kvadraticky (závislosť sa vypočítava medzi všetkými dvojicami slov), tieto výpočty by trvali vyše 20-krát dlhšie.

Parameter *Top p* ovplyvňuje náhodný výber tokenov nasledovne: do *samplingu* (váženého náhodného výberu) vyberá len najpravdepodobnejšie tokeny s celkovým súčtom nižším ako Top p. Tento parameter sme nastavili preto, aby sampling neznižoval kvalitu generovaného textu. Od tohto je aj predvolene nastavený parameter *Top k* na 50 – tento parameter určuje, koľko najpravdepodobnejších tokenov sa vyberá do samplingu. *Top p* sme nastavovali pre prípady, keď si je model celkom istý svojím generovaním a najpravdepodobnejšiemu tokenu priradí pravdepodobnosť vyše 95 %. Táto situácia môže nastať napríklad pri generovaní porovnaní (napr. *less than* – menší ako), v angličtine nie je v takomto kontexte pre slovo „than“ veľa synonymných slov. Ak by sme *Top p* nenastavili, tak by existovala malá pravdepodobnosť, že sa vygeneruje iné slovo ako „than“, napríklad „then“ (v preklade „potom“). To je častá chyba v angličtine a je veľká pravdepodobnosť, že model sa pri učení stretol aj s touto kombináciou.

Aj takáto malá pravdepodobnosť môže mať potom dopad na všetky nasledujúce generovania a model potom môže pôsobiť až príliš náhodne.

5.1.2 Prompty

Modelu GPT-2 Medium sme predkladali jeden zo štyroch osobne vymyslených promptov a z nich sme vygenerovali 43 textov, ktoré sme použili pre túto analýzu.

[1]Vo zvyšku textu budeme používať anglický výraz *prompt*.

Všetky tieto texty sa nachádzajú v zložke *calculated_logs*. Súbory majú príponu, s číslom promptu (napr. -_01.txt).

Prompt 01

```
It was a foggy morning in the Gransbury Woods when Dory
    started seeing strange things. At first, it
```

```
It was a foggy morning in the Gransbury Woods when Dory started seeing
strange things. At first,
```

Tento prompt je základným promptom, ktorý pre analýzu používame. Je to prompt, ktoré sme použili na vygenerovanie *export logov* pre import v zložke *logs* – *generated_log.txt* a *analysed_log.txt*. Zároveň sa tento prompt (a text vygenerovaný s ním) nachádza aj v zložke *prompts* ako prompty, ktoré je priamo možné predať programu Controller NoGUI (popísanom v 4.5) na priame generovanie *export logov* bez užívateľského rozhrania.

O modele GPT-2 Medium sme sa dozvedeli, že takýto vstup mu stačí na to, aby vyvodil, že má generovať príbeh. Príbehy boli dejovo rôzne, mierne sa zvyknú líšiť aj v literárnom žánri – niektoré texty sa zameriavajú na monštrá a fiktívne postavy, iné sa zas vydali strašidelnou cestou a hovoria o smrti a kostiach visiacich zo stromu.

Tento prompt je zložený z 23 subwordov, takže novo vygenerovaný text má 127 subwordov, čo niekedy ani nepokrylo úvodnú časť príbehu. Už z tejto časti je možné odvodiť, aký literárny žáner sa model snaží generovať.

Ako príklad si môžeme ukázať tento text:

```
It was a foggy morning in the Gransbury Woods when Dory
    started seeing strange things. At first, it was the
    eerie sound of cars driving by. But over the next 3
    weeks, Dory and her family began to notice strange
    things occurring at night. A very familiar sight they
    observed was that mysterious lights which looked like
    those seen on the side of the road. This would persist
    for weeks. Within the month, the first sighting was of
    a white owl hovering in the trees and a
```

Na tomto texte vidíme, že vygenerovaný text má jasné črty príbehu. Ide síce len o úvod, ale už v ňom sme vnorení to temnej atmosféry slovami ako *eerie sound* (desivý zvuk), *mysterious lights* (záhadné svetlá) alebo *strange things* (zvláštne veci). Slovné spojenie *strange things* je priamo skopírované z promptu.

Prompt 02

```
"I didn't mean it to be like this. It happened so fast. I
    told him. It has to stop. Finally, he
```

Tento prompt sme vytvorili ako krátky prompt zložený z viacerých viet. Hlavným dôvodom bolo vyskúšať mód *Missing sentence dependency*, pretože ten pracuje s celými vetami. Cieľom bolo vytvoriť prompt, ktorý bude obsahovať väčšie množstvo krátkych viet s domnienkou, že sa model možno pridá a budeme môcť skúmať, ako ďaleko nastávajú závislosti.

Tento prompt je zložený z 28 subwordov, ktoré tvoria štyri celé vety a piatu uvádzajú. Tento prompt sme nechali vygenerovať 11-krát pre potreby analýzy.

Výsledné generovania neukazujú na jeden konkrétny literárny žáner. Zo štyroch nami testovaných promptov boli výsledky v tomto najviac rôznorodé. Niekedy to znelo ako článok z novín, inokedy model vytvoril príbeh.

Problémov s týmto promptom môže byť viacero. Prvé vety nám v podstate nehovoria nič, pretože sa odkazujú na zámeno „it“ (to), ktoré v danom kontexte môže znamenať čokoľvek a model GPT-2 si veľmi rád náhodne domyslí pri generovaní, čo „to“ znamená. Výsledky to potvrdzujú, málokedy vidíme vyššiu závislosť na prvých vetách, pretože text ako taký dáva zmysel aj bez nich.

Možno by model bez posledného slova v prompte generoval zaujímavejšie texty, pretože sa tam to slovo možno až tak nehodí.

Ako príklad jedného generovania modelu s týmto promptom si môžeme ukázať tento text:

```
"I didn't mean it to be like this. It happened so fast. I
    told him. It has to stop. Finally, he gave in and
    accepted his fate."

It is an incredibly moving and touching story. I could go
    on forever about what he is trying to do. I don't need
    to. What I do need to know is, if you love someone and
    you want your life to improve, then stop hurting them.

As this situation has made me feel, especially about my
    parents, I realize just how valuable family members are
    . Just as one of my cousins passed away earlier this
    year. My mother, my sister and my brothers are the ones
     who always comfort me when things are going poorly and
     keep me focused
```

Prvé, čo si môžeme všimnúť na tomto texte je, že úvodné úvodzovky si model na konci prvého odseku sám zatvoril. To môže znamenať, že si pamätá, že ich je potrebné uzavrieť. Párkrát počas generovania sa nám stalo, že model tieto úvodzovky neuzavrel a pokračoval ďalej akoby ich uzavrel.

Interpunkčné znamienka sa zhlukujú do subwordov, čo znamená, že „.“ a „.″“ sú dva rozdielne subwordy a dostanú dve rozdielne pravdepodobnosti, ale niekedy tento výber až tak neovplyvní výber ďalšieho subwordu. Môžeme polemizovať, že takáto chyba sa môže ľuďom stať celkom často a v trénovacích dátach to mohol model uvidieť a zistil, že to nie je až taký problém, hlavne, ak bude v ďalšom písaní pokračovať v novom riadku (teda oddelí citovanú časť a ďalšie pokračovanie).

Môžeme si všimnúť, že model v tomto texte vygeneroval prázdny riadok, teda dva znaky konca riadku za sebou. Takto to je počas celej analýzy. Ak sa vygeneruje znak konca riadku, tak ďalší nasleduje vo vyše 99 % takmer pre všetky prípady (výnimkou je Prompt 04, aj to len pre niektoré prípady).

Ďalšou vecou, ktorú si môžeme všimnúť na príklade, je žáner, v ktorom bol text vygenerovaný. Tento príklad vyzerá ako koniec nejakého článku, kde novinár robí rozhovor s niekým blízkym osoby, o ktorej sa hovorí a ktorej príbeh je *„incredibly moving and touching“* (neuveriteľne dojemný).

Tento export log je možné si nájsť a importovať ho do vizualizátora. Nachádza sa v zložke *calculated_logs* pod názvom *log20221504152851_02.txt*

Prompt 03

```
The highest-paid Norwegian limousine driver of all time,
```

Týmto promptom sme chceli otestovať znalosť slov zložených z viacerých subwordov. Takmer všetky texty vygenerované modelom GPT-2 Medium s týmto vstupom sú bulvárne novinové články, v ktorých sa spomínajú drahé veci alebo peniaze. To sa viaže na pojem „highest-paid“ (najviac zarábajúci), aj napriek tomu, že sa tento pojem skladá až z troch subwordov (rozdelených na „␣highest/-/paid“).

Tieto subwordy ale predstavujú celé slová v angličtine a znamenajú to isté. Na testovanie znalosti (alebo odvoditeľnosti) slov zložených z viacerých subwordov sa v prompte nachádza slovo „limousine“ (limuzína), ktorú kódovanie rozdelilo na lim/ous/ine. Domnievame sa, že model aj napriek tomu rozumie tomuto slovu, čo môžeme demonštrovať týmto vygenerovaným textom:

```
The highest-paid Norwegian limousine driver of all time,
    who earned about $1 million last year, has a $6 million
     annual salary. He was on an exclusive limousine tour
    in Asia as part of a private jet tour.

"In the late 1990s there was talk about the world's best
    limousine driver. I came out of this journey with a lot
     of money - at least one million dollars," says Gudrun
    Johansen.

"Some people were interested because they could have a
    nice car for a car trip from Norway to China and back.

"Some people could do it and some people couldn't do it,
    but most of them took it."

She says she has also
```

V príklade sa spomína *„nice car“* (pekné auto) ako synonymum pre limuzínu. Limuzína sa taktiež spomína v poslednej vete prvého odseku: *He was on an exclusive limousine tour in Asia as part of a private jet tour.* (Bol na exkluzívnom turné limuzínou v Ázii v rámci turné súkromným tryskáčom.).

Taktiež si môžeme všimnúť, že model vygeneroval meno *Gudrun Johansen*. Generovanie tohto mena nebolo príliš pravdepodobné. Na subwordy sa delí takto: „ G/ud/run/ Joh/ansen“ a jednotlivé subwordy, ktoré boli vygenerované, mali nasledujúce pravdepodobnosti: 0,45 %, 24 %, 7,4 %, 0,7 % a 44 % (celý export log je možné otvoriť vo vizualizátore, nachádza sa v zložke *calculated_logs* s názvom *log20220305010357_03.txt*).

Podarilo sa nám nájsť jednu stránku na Wikipedii s týmto menom, ktorá je iba v dánčine a obsahuje len pár odsekov. Je možné, že si model toto meno zapamätal a vygeneroval alebo k nemu prišiel náhodne. Pri niektorých generovaniach sme

sa pokúšali vygenerované mená vyhľadať na webe a párkrát sa nám stalo, že sme nedokázali nájsť nič, takže nie všetky mená, ktoré model vygeneruje, patria niekomu známemu.

Prompt 04

```
Poor old lady, she swallowed a fly.
I don't know why she swallowed a fly.
Poor old lady, I think she'll die.

Poor old lady, she swallowed a spider.
It squirmed and wriggled and turned inside her.
She swallowed the spider to catch the fly.
I don't know why she swallowed a fly.
Poor old lady, I think she'll die.
```

Tento celkom dlhý prompt[2] sme chceli použiť na to, aby sme zistili, či vie model rozoznať, že ide o poéziu a má sa pokúsiť vygenerovať rým.

Prompt je dlhý 82 subwordov, takže sa vygenerovalo len 68 ďalších, čo približne postačovalo na ďalšiu strofu básne pri generovaniach.

Nanešťastie, výsledky ukázali, že model GPT-2 Medium to na tomto prompte nerozoznal. V prompte sa opakujú vety a zdá sa, že model považoval to za najdôležitejší znak promptu, v ktorom má pokračovať.

Typický text pre tento prompt vyzeral nejak takto:

```
Poor old lady, she swallowed a fly.
I don't know why she swallowed a fly.
Poor old lady, I think she'll die.

Poor old lady, she swallowed a spider.
It squirmed and wriggled and turned inside her.
She swallowed the spider to catch the fly.
I don't know why she swallowed a fly.
Poor old lady, I think she'll die.

Poor old lady, she swallowed a worm.

It swallowed the worm to catch the fly.

It could not get past the spider or it would die.

Poor old lady, I think she swallowed a worm.

That's pretty good, though. She should have eaten more
```

Na tomto vygenerovanom texte si môžeme hneď všimnúť, že štruktúra básne je úplne iná, stále to namiesto jedného znaku pre nový riadok vygenerovalo dva po sebe, takže medzi každým veršom je voľný riadok.

[2]Prompt aj neskôr vygenerovaný text sú zámerne odsadené viac do stredu, aby viac vyznela štruktúra básne a báseň nesplývala.

Pri dôkladnejšej analýze môžeme sledovať, že model vygeneroval na začiatku strofy správnu štruktúru a slovo „worm“ (červík), čo celkom pasuje k predchádzajúcemu textu.

Prvých pár veršov sa model snaží držať štruktúru, dokonca vygeneroval aj samostatne zaujímavú vetu, ale v poslednej vete sa to zlomí. Dovtedy sme *„poor old lady“* (úbohú starú pani) ľutovali a hovoríme, ako si myslíme, že zomrie (*„think she'll die“*) a nakoniec napíšeme, že to je fajn a mala jesť viac.

Tento text je možné nájsť v zložke *calculated_logs* s názvom *log202227041048-50_04.txt* a je možné ho importovať do vizualizátora.

5.2 Zistenia

V analýze jednotlivých módov sa pozeráme na priemerné výsledky cez všetky vygenerované texty. V závislostných módoch pozorujeme, ako ďaleko siahajú *priame závislosti* od generovaného subwordu. Závislosti, ktoré v jednotlivých módoch sledujeme, sú priame závislosti, t.j. sledujeme, ako generovaný subword závisí priamo na prítomnosti toho-ktorého subwordu v texte.

Pri týchto módoch nevieme sledovať *nepriame závislosti* (ako veľmi závisí generovaný subword od subwordov predošlých, ktoré závisia na chýbajúcom/zamieňanom subworde), pretože vždy v texte meníme alebo vynechávame len jeden subword. Ten už však mohol ovplyvniť generovanie subwordov pred subwordom, na ktorý sa zameriavame.

Na začiatok si definujme dva pojmy:

- *Pozitívna závislosť* – prítomnosť subwordu (resp. vety) v texte *pozitívne* ovplyvňuje pravdepodobnosť na generovanie daného subwordu. Ak nie je daný subword (resp. veta) prítomný (chýba alebo je nahradený), pravdepodobnosť sa znižuje.

- *Negatívna závislosť* – prítomnosť subwordu (resp. vety) v texte *negatívne* ovplyvňuje pravdepodobnosť na generovanie daného subwordu. Ak nie je daný subword (resp. veta) prítomný, pravdepodobnosť sa zvyšuje.

5.2.1 Subword distribution

Tento mód sleduje pôvodné pravdepodobnosti pri generovaní subwordov. Analýzou výsledkov tohto módu vieme určiť to, že pri generovaní sa v priemere vygeneroval subword na pozícii *3,75* z najpravdepodobnejších 50, ktoré sa dostali do samplingu, kde boli vyberané vážene podľa ich pravdepodobnosti.

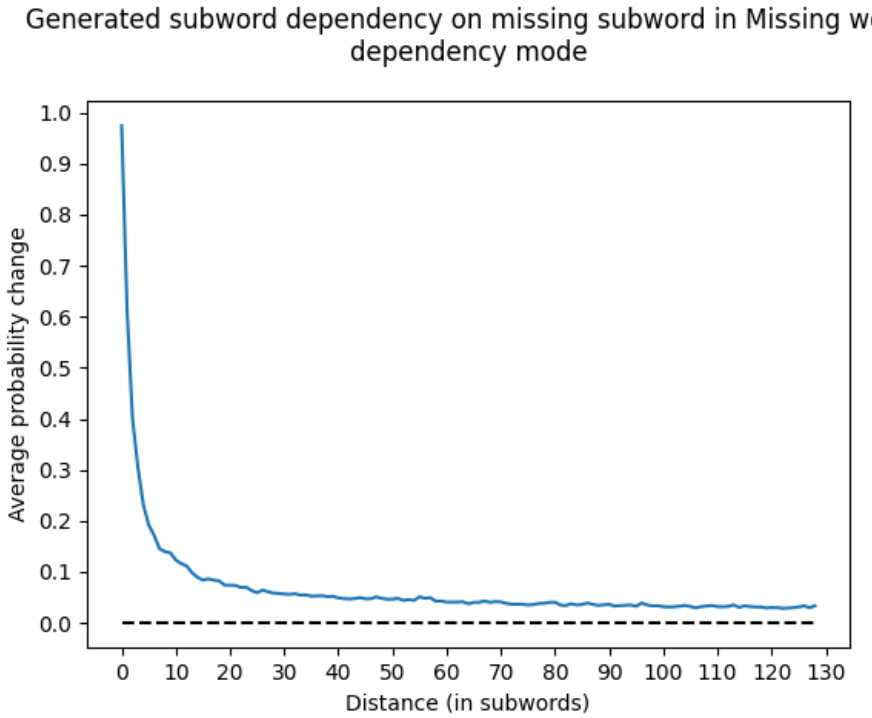
Vygenerovaný subword sa medzi 10 najpravdepodobnejšími subwordmi, ktoré sa pre zobrazenie v grafe ukladali, nachádzal priemerne v 83,34 % (zhruba v 5/6) prípadov. Priemerná primárna pravdepodobnosť (pravdepodobnosť určená skóre na výstupe siete ešte pred samplovaním) vygenerovaného subwordu bola 31,83 %.

Priemerná primárna pravdepodobnosť najpravdepodobnejšieho slova bola zaokrúhlene 39,73 %.

5.2.2 Missing word dependency

V Missing word dependency sledujeme závislosti medzi subwordmi tak, že ten, čo sa nachádza skôr v texte vynecháme a pozrieme sa, aký to malo vplyv na pravdepodobnosť pri generovaní toho druhého (viac v 2.2).

Pri závislostiach nás zaujíma, ako ďaleko sa model pozerá. V našom prípade teda, ako sa vyvíja závislosť generovaného subwordu na chýbajúcom subworde v závislosti od ich vzdialenosti. Presne tieto údaje sú zobrazené na obrázku 5.1, ktorý ukazuje zmenu pravdepodobnosti (závislosť) vzhľadom na vzdialenosti medzi jednotlivými subwordmi. Vzdialenosť meriame počtom subwordov medzi danými slovami.

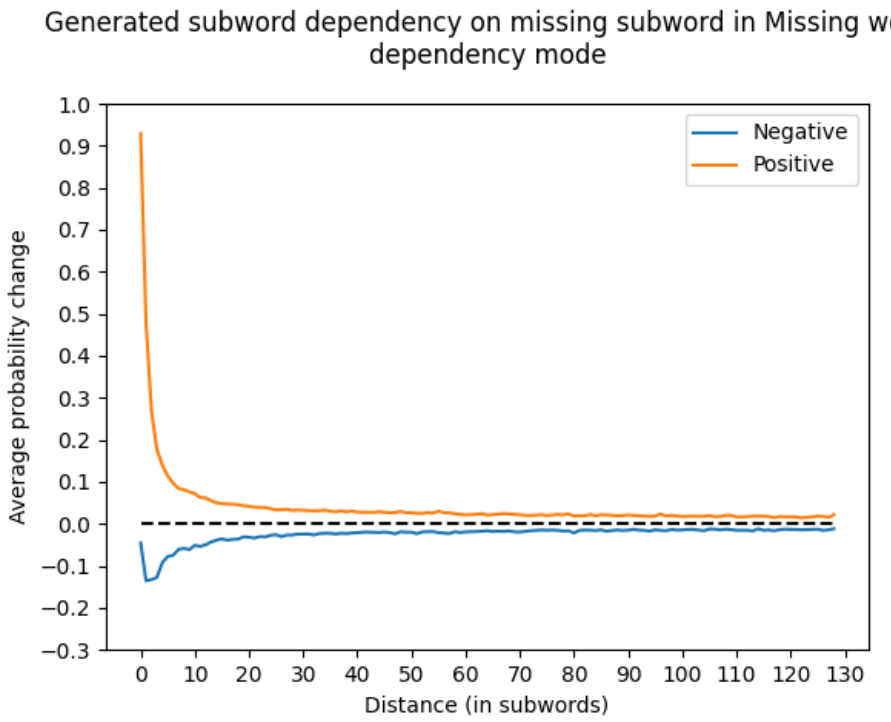


Obrázek 5.1: Priemerná závislosť (v absolútnej hodnote) generovaného subwordu na predchádzajúcom subworde vzhľadom na vzdialenosť medzi nimi (koľko subwordov sa medzi nimi nachádza). Táto závislosť je vypočítaná v móde Missing word dependency.

Závislosť môže byť pozitívna aj negatívna. Ukážku toho, ako sa vyvíja pozitívna a negatívna závislosť vzhľadom na vzdialenosť, môžeme vidieť na obrázku 5.4.

Na oboch týchto obrázkoch môžeme vidieť, že závislosť je najsilnejšia hneď na začiatku a postupne prudko klesá, ale aj pri vzdialenosti vyše 120 subwordov, majú slová závislosť v priemere okolo 5 %.

Negatívna závislosť rýchlo vystúpala zhruba na 14 % (-14 % pretože rozdiel v pravdepodobnosti je záporný), ale nakoniec klesla a v každom bode bola nižšia ako pozitívna závislosť. Vo všetkých vygenerovaných dátach nebolo veľmi časté, že nastane negatívna závislosť hneď na začiatku, pretože to by znamenalo, že vygenerovaný subword mal nižšiu pravdepodobnosť v pôvodnom texte, čo znížilo pravdepodobnosť, že sa daný subword vôbec vygeneruje.



Obrázek 5.2: Priemerná závislosť generovaného subwordu na predchádzajúcom subworde vzhľadom na vzdialenosť medzi nimi s oddelenými zložkami pre pozitívne a negatívne závislosti. Táto závislosť je vypočítaná v móde Missing word dependency.

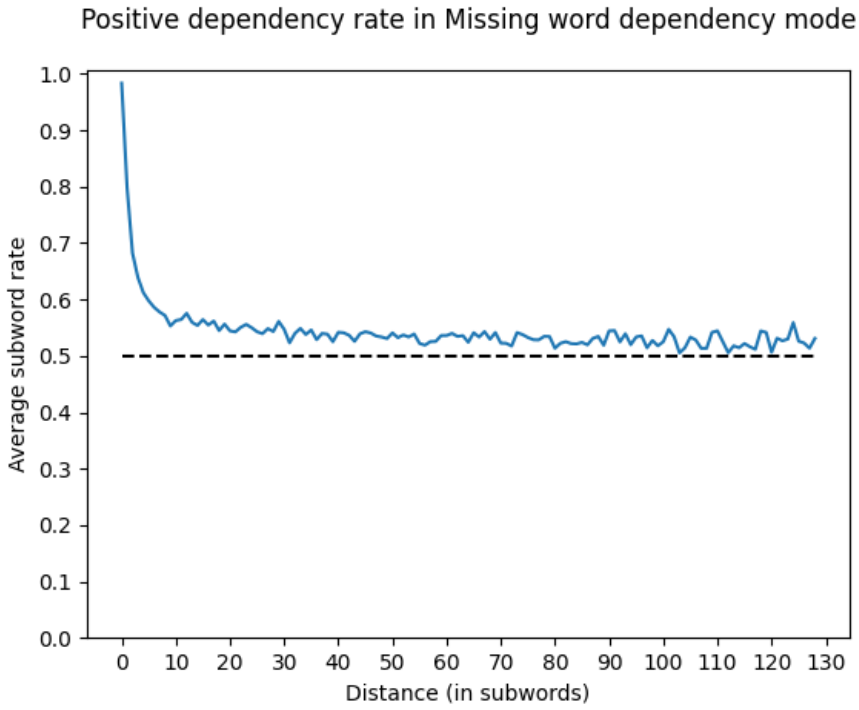
Naša domnienka je, že pri malých vzdialenostiach sa negatívna závislosť prejavuje tak, že *chýbajúci* subword v pôvodnom texte veľmi zvýšil pravdepodobnosť iného subwordu, napríklad začiatok nejakého ustáleného slovného spojenia. Bez tohto subwordu tu model príležitosť na ustálené slovné spojenie nevidí, takže je vyššia šanca pre vygenerované slovo.

Pomer pozitívnych a negatívnych závislostí naznačuje obrázok 5.3. Na obrázku je zobrazený podiel pozitívnych závislostí zo všetkých závislostí (teda dvojíc všetkých slov). Vidíme, že priamo predchádzajúce slovo má pozitívnu závislosť na generované slovo vo vyše 98 % prípadov, ale celkom rýchlo táto závislosť klesá na hodnoty tesne nad 50 %.

Tieto údaje sú úzko späté s údajmi v obrázku 5.2, kde ukazujeme vývoj negatívnej závislosti vzhľadom na vzdialenosť. Negatívna závislosť nie je bežná v generovanom texte z toho dôvodu, že znižuje pravdepodobnosť generovaného subwordu pri prvotnom generovaní. To znamená, že menší počet takýchto subwordov je predsa len vygenerovaných. Pozitívna závislosť naopak povzbudzuje model, aby vybral daný subword.

Tieto hypotézy ale dávajú zmysel len pre kratšie vzdialenosti, na obrázku 5.1 sme videli, že priemerná závislosť klesá so stúpajúcou vzdialenosťou. V záverečnej časti grafu 5.3 vidíme, že pomer pozitívnych a negatívnych závislostí zhruba od vzdialenosti 90 subwordov vyššie osciluje výrazne viac ako predtým.

To môže byť spôsobené tým, že samotné závislosti sú príliš malé ($< 1\,\%$), čo vo výsledky nepredstavuje veľký rozdiel. ä Preto sme sa rozhodli pozrieť na to, aký je podiel silných pozitívnych a silných negatívnych závislostí (závislosti vyššie



Obrázek 5.3: Podiel pozitívnych závislostí generovaného subwordu na predchádzajúcich subwordoch vzhľadom na ich vzdialenosť. Táto závislosť je vypočítaná v móde Missing word dependency.

ako 5 %). Tento údaj je zobrazený na obrázku 5.4.

Vidíme, že pre pozitívne aj negatívne závislosti sa to aj pre vzdialenosti vyššie ako 120 subwordov pohybuje okolo 10 %. Z tohto grafu by sme mohli usúdiť, že aj napriek vzdialenosti sa model pri generovaní necháva ovplyvňovať aj subwordmi vzdialenými viac ako 120 subwordov.

5.2.3 Missing sentence dependency

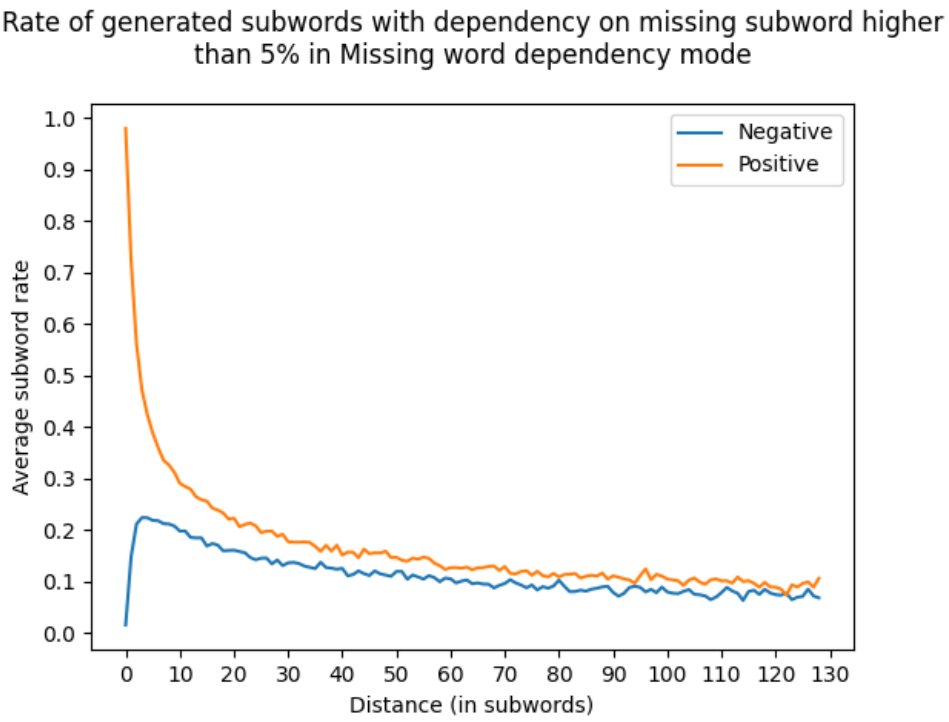
Missing sentence dependency sleduje závislosti generovaného subwordu na predchádzajúcich vetách tak, že sa pozrie na to, ako by sa zmenila pravdepodobnosť generovaného subwordu, ak by sa daná veta vôbec v texte nevyskytovala.

V Missing sentence dependency sme sa najprv pokúsili sledovať rovnaké údaje ako v predchádzajúcej sekcii, konkrétne ako závisí generovaný subword na predchádzajúcich vetách vzhľadom na vzdialenosť od nich.

Tieto údaje nám pomohli vygenerovať grafy pre obrázky 5.5 a 5.6. Tieto 2 grafy majú rôzne výkyvy a ťažko sa z nich niečo vyvodzuje. Vidíme jednoznačnú závislosť na predchádzajúcej vete. Závislosť pre vzdialenejšie vety má tendenciu postupne klesať.

Na grafe 5.6 môžeme vidieť, že pozitívna závislosť na prvej vete prevažuje tú negatívnu v priemere zhruba o 4 % a potom sa rozdiel pozitívnej a negatívnej závislosti pohybuje okolo 0.

Preto sme sa na vety pokúsili pozrieť z iného uhla pohľadu a sledovali sme, či je priemerná závislosť generovaného subwordu na predchádzajúcich vetách závislá na pozícii daného slova vo vete. Výsledný graf je ukázaný na obrázku 5.7.



Obrázek 5.4: Podiel silných závislostí (vyšších ako 5 %) generovaného subwordu na predchádzajúcich subwordoch vzhľadom na ich vzdialenosť. Táto závislosť je vypočítaná v móde Missing word dependency.

Opäť ide o celkom oscilujúce dáta. Aj napriek tomu vidíme, že s vyššou pozíciou vygenerovaného subwordu sa znižuje závislosť na predchádzajúcich vetách.

To môže byť spôsobené tým, že text sa viac zameriava na subwordy vo vlastnej vete, ako na tú predchádzajúcu.

5.2.4 Replacing subword dependency

Replacing subword dependency sleduje závislosti medzi generovaným subwordom a *zamieňaným* subwordom tak, že sa snaží zamieňaný subword nahradiť za *podobné* slovo (viac v 2.4).

Nie každý subword je možné nahradiť, nedáva to vždy úplne zmysel. Z tohto dôvodu subwordy filtrujeme tak, aby nám ostali vždy len také, ktoré predstavujú jedno slovo, navyše toto slovo musí byť plnovýznamové (viac v 2.4.2).

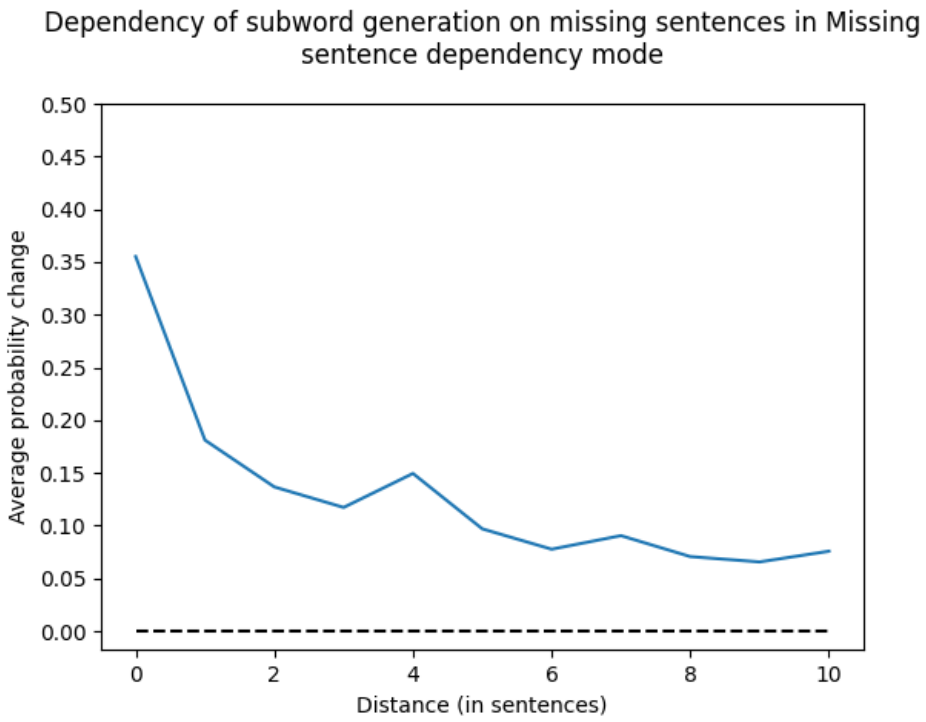
To znamená, že celkový počet výsledkov, z ktorých sme vytvárali graf, je menší ako počet výsledkov pre *Missing word dependency*.

Výsledky tohto módu sme sa pokúsili porovnať s výsledkami módu Missing word dependency. To je dôvod, prečo sme vygenerovali grafy sledujúce presne rovnaké udalosti.

Na obrázku 5.8 vidíme priemernú závislosť generovaného subwordu na predchádzajúcom subworde vzhľadom na vzdialenosť medzi nimi.

Graf má úplne rovnaké črty, ako graf 5.1, ktorý ukazuje rovnaké údaje pre mód Missing word dependency.

Môžeme vidieť, že závislosť na prvých pár subwordoch má nižšie hodnoty. To sa dá vysvetliť tým, že ak tam ten priamo predošlý subword nie je, model sa ho



Obrázek 5.5: Priemerná závislosť subwordu od predchádzajúcich viet vzhľadom na ich vzdialenosť (koľko celých viet sa nachádza medzi subwordom a sledovanou vetou). Táto závislosť je vypočítaná v móde Missing sentence dependency.

tam snaží vygenerovať. Na druhú stranu, ak sa tam nejaký subword nachádza, tak model rozmýšľa nad celkovým kontextom a čo by sa tam za daného kontextu hodilo. V niektorých situáciách model rozmýšľa nad podobnými a inokedy nad inými subwordmi.

Na obrázku 5.9 môžeme vidieť podobné dáta ako v grafe 5.2, pozitívne a negatívne zložky sú oddelené. Od vzdialenosti približne 25 subwordov vidíme, že grafy majú takmer rovnaké hodnoty. Niekde sa dokonca zdá, že negatívna závislosť prevláda nad tou pozitívnou.

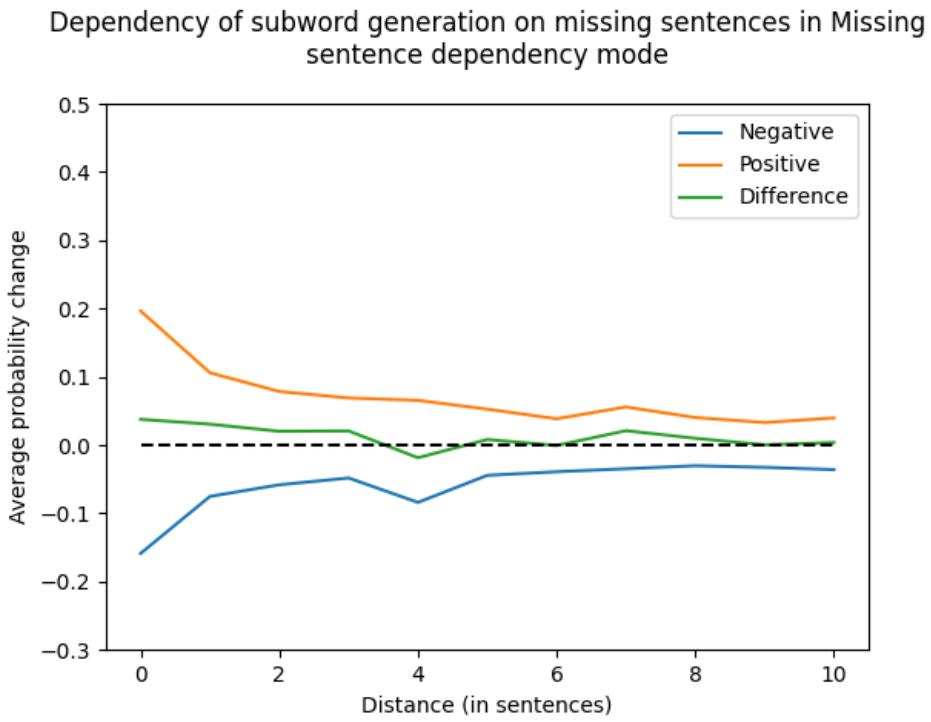
To môžeme opäť vysvetliť tým, že negatívna závislosť v našich výpočtoch môže byť neobmedzená a pozitívna závislosť je obmedzená číslom 1.

V tomto móde sú oveľa častejšie negatívne závislosti ako v móde Missing word dependency – v ňom sme ich vysvetľovali tak, že prítomnosť daného subwordu znižuje pravdepodobnosť pre generovaný subword. Negatívne závislosti v Replacing subword dependency majú mierne odlišné teoretické vysvetlenie. Negatívna závislosť v Replacing subword dependency nastáva, ak *podobné* slová ešte viac dvíhajú pravdepodobnosť generovaného subwordu.

Pre vzdialenosti do približne 25 subwordov prevláda pozitívna závislosť, takže vidíme, že *zamieňané* slovo pomáha pri generovaní toho vygenerovaného.

Na obrázku 5.10 je graf, ktorý ukazuje podiel pozitívnych závislostí generovaného subwordu na predchádzajúcom subworde vzhľadom na ich vzdialenosti.

Vidíme, že počet pozitívnych závislostí stále poväčšine prevažuje nad počtom negatívnych závislostí, aj keď tentokrát existujú pozície, pre ktoré je počet negatívnych závislostí vyšší. Pre vzdialenosti menšie ako 10 subwordov je počet pozitívnych závislostí celkom jednoznačne vyšší, potom sa graf v podstate ustáli

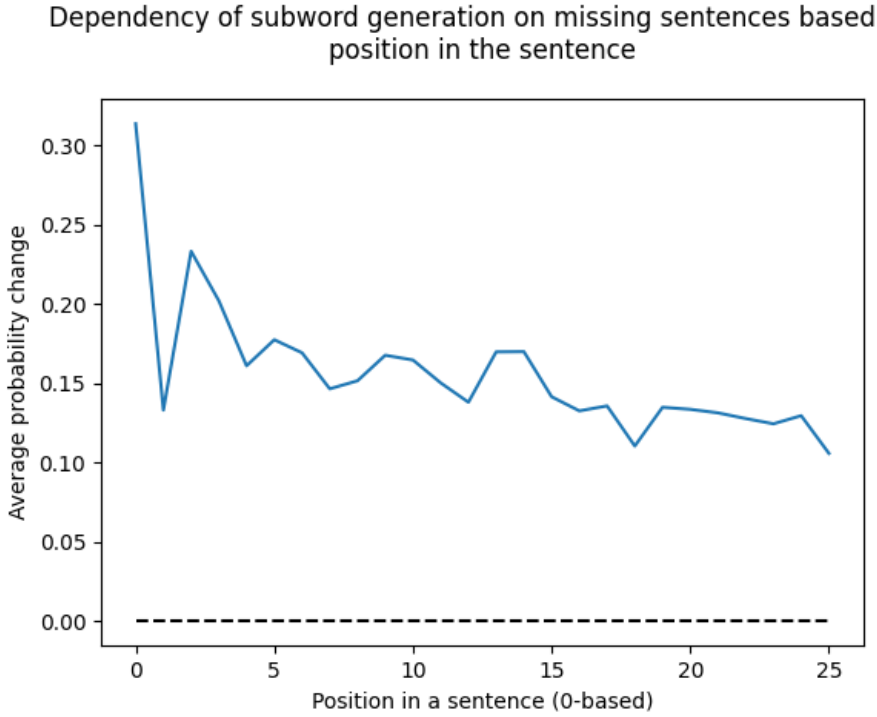


Obrázek 5.6: Priemerná závislosť subwordu od predchádzajúcich viet vzhľadom na ich vzdialenosť (v počte viet), rozlíšená je pozitívna a negatívna závislosť a aj ich rozdiel. Táto závislosť je vypočítaná v móde Missing sentence dependency.

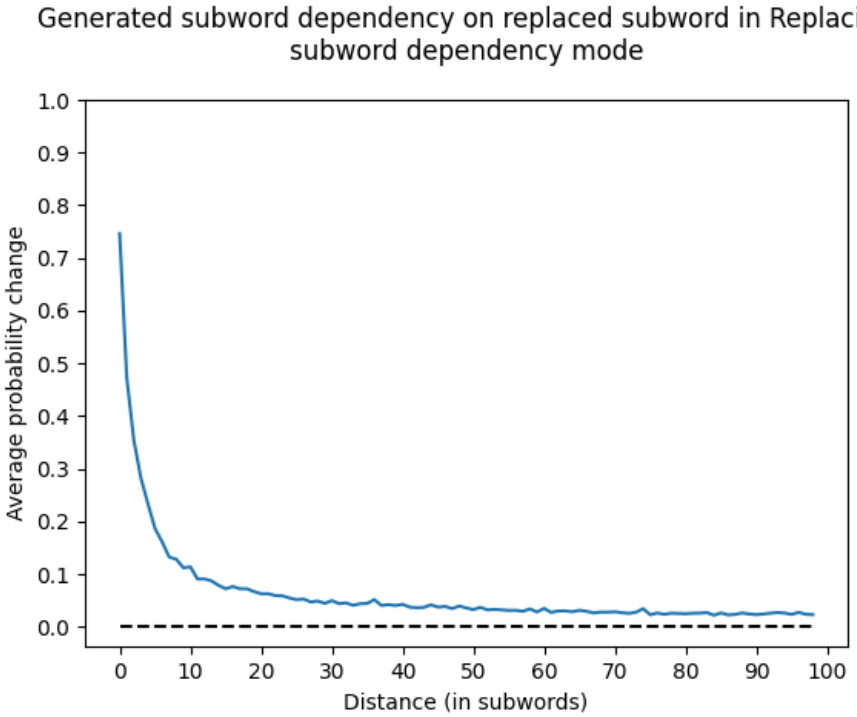
okolo 50 %.

Tieto pozitívne aj negatívne závislosti môžu byť minimálne a nemusia mať veľký vplyv na vygenerovaný subword. Na obrázku 5.11 ukazujeme podiel silných závislostí vzhľadom na ich vzdialenosť.

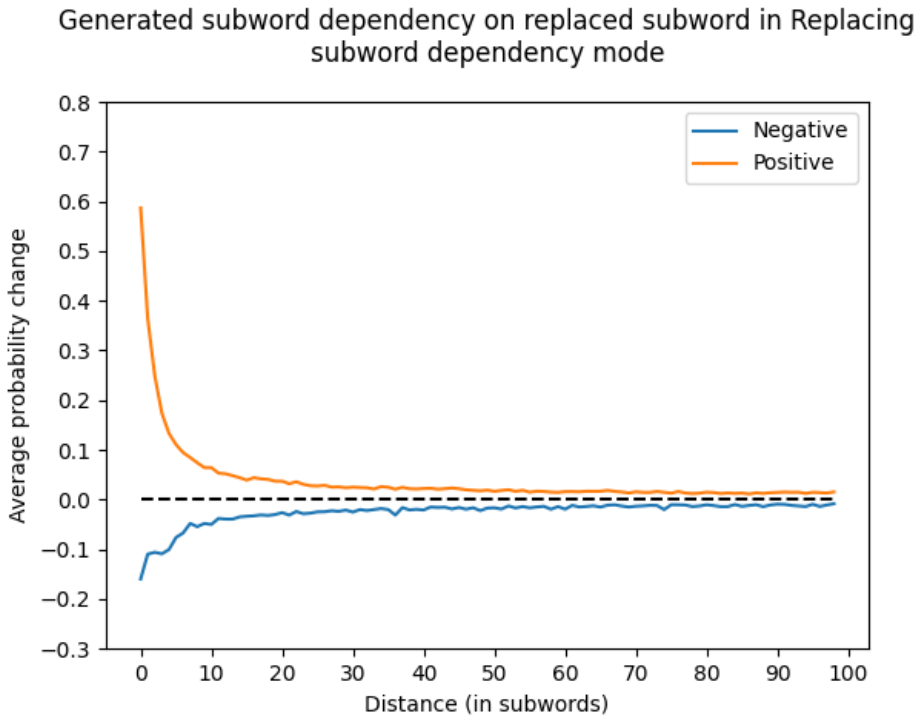
Opäť vidíme, že pre vzdialenosti do 20 subwordov prevažuje pozitívna závislosť o pár percentuálnych bodov nad negatívnou.



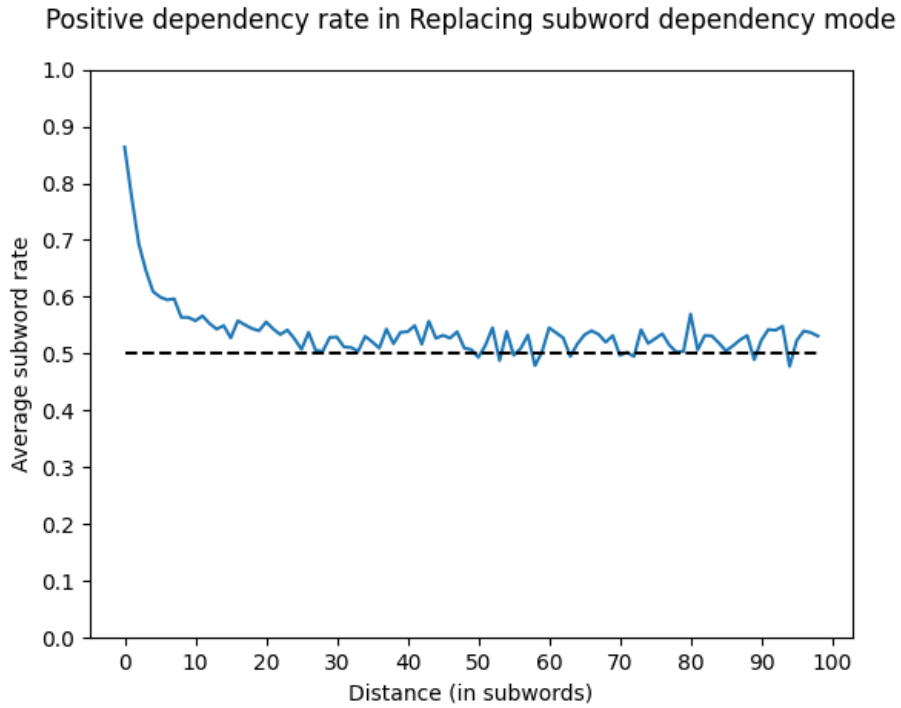
Obrázek 5.7: Priemerná závislosť generovaného subwordu od prechádzajúcich viet vzhľadom na pozíciu generovaného subwordu vo vete. Táto závislosť je vypočítaná v móde Missing sentence dependency.



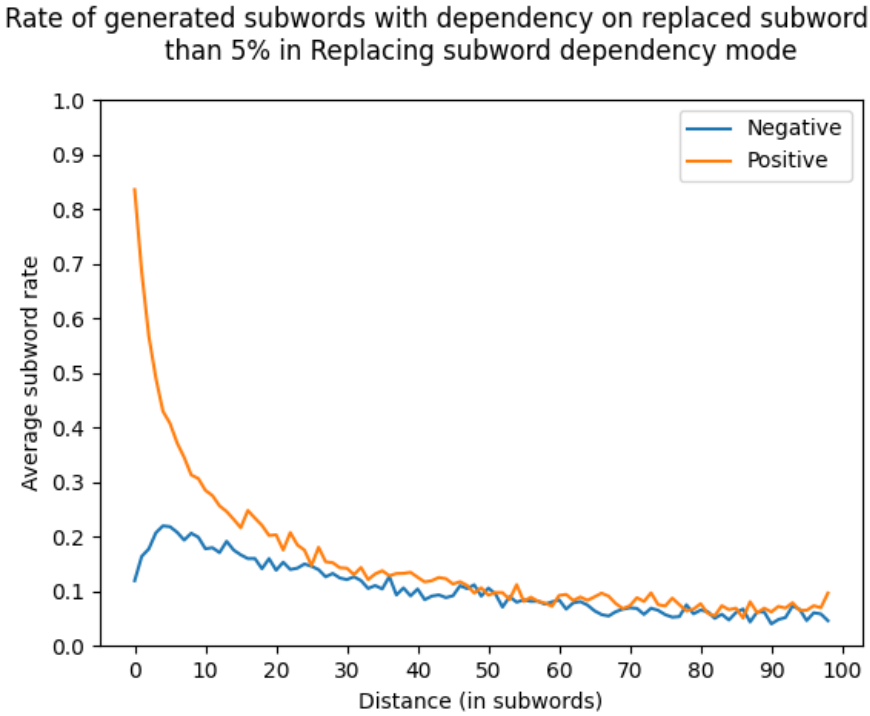
Obrázek 5.8: Priemerná závislosť (v absolútnej hodnote) generovaného subwordu na predchádzajúcom subworde vzhľadom na vzdialenosť medzi nimi (koľko subwordov sa medzi nimi nachádza). Táto závislosť je vypočítaná v móde Replacing word dependency.



Obrázek 5.9: Priemerná závislosť generovaného subwordu na predchádzajúcom subworde vzhľadom na vzdialenosť medzi nimi s oddelenými zložkami pre pozitívne a negatívne závislosti. Táto závislosť je vypočítaná v móde Replacing subword dependency.



Obrázek 5.10: Podiel pozitívnych závislostí generovaného subwordu na predchádzajúcich subwordoch vzhľadom na ich vzdialenosť. Táto závislosť je vypočítaná v móde Replacing subword dependency.



Obrázek 5.11: Podiel silných závislostí (vyšších ako 5 %) generovaného subwordu na predchádzajúcich subwordoch vzhľadom na ich vzdialenosť. Táto závislosť je vypočítaná v móde Replacing subword dependency.

Záver

V tejto práci sme predstavili aplikáciu určenú na vizualizáciu závislostí vygenerovaných slov v modeloch hlbokých neurónových sietí architektúry Transformer s názvom GPT-2. Aplikácia podporuje načítavanie vlastných, ako aj predvolených modelov GPT-2.

Predstavili sme aj program, ktorý umožňuje všetky závislosti predpočítať bez užívateľského rozhrania, ktorý je ideálny pre výpočty na výpočetných clusteroch.

Spolu s aplikáciou sme predstavili nové metódy, ktoré sledujú závislosti medzi vygenerovanými slovami. Tieto metódy sledujú závislosti medzi dvojicou slov (závislosť slova na slove v texte pred ním) tak, že to prvé slovo z textu vynecháme alebo nahradíme a pozeráme sa na to, ako sa zmenila pravdepodobnosť vygenerovaného slova. Predstavili sme aj metódu, ktorá skúma závislosti vygenerovaných slov na predchádzajúcich vetách. Pri každej metóde sme si ukázali, ako vyzerá, keď sa používa v aplikácii aj s príkladmi údajov, ktoré zobrazuje.

V analytickej časti sme následne skúmali, či sú tieto metódy vhodné na analyzovanie správania modelov GPT-2 a aj to, čo nám tieto modely vedia povedať o modele GPT-2 Medium, na ktorom prebiehala analýza. Analýza prebiehala na 43 rôznych textoch, ktoré boli vygenerované týmto modelom zo 4 rôznych vstupov pre model.

Dozvedeli sme sa, že model GPT-2 Medium berie pri generovaní najväčší ohľad na najbližších prvých pár slov, ale aj napriek tomu svoju úlohu zohrávajú aj vzdialené slová. Zistili sme, že v módoch Missing word dependency aj Replacing subword dependency platí, že na vyše 10 % subwordoch vzdialených 120 – 130 subwordov má generované slovo silnú závislosť (5 % a viac).

Pomocou módu Missing sentence dependency sme zistili, že vygenerované slová majú tendenciu závisieť na predchádzajúcich vetách viac, ak sa nachádzajú na začiatku vety.

Veríme, že táto práca poslúži ako vhodný nástroj na výskumy správania jazykového modelu GPT-2 a že naše modely inšpirujú ďalšie výskumy na jazykových modeloch architektúry Transformer.

Budúca práca

Analýza modelov GPT-2 a ich správania je vo všeobecnosti rozsahovo veľmi veľká téma, ktorá by vyžadovala milióny vygenerovaných textov rôznych dĺžok, rôznych nastavení parametrov a rôznych modelov GPT-2. Ďalší výskum by sa mohol zaoberať aj skúmaním modelu GPT-2 na promptoch, ktoré by určovali literárny žáner generovaného textu a jednotlivé výsledky skúmal medzižánrovo.

Mohlo by mať zmysel vymyslieť podobné vizualizačné aplikácie aj pre iné druhy Transformerov, či už to je model BERT alebo aj menšie jazykové modely, kde by jednotlivé výpočty mohli trvať kratšie ako pre GPT-2.

Seznam použité literatury

Abnar, S. a Zuidema, W. (2020). Quantifying attention flow in transformers. *arXiv preprint arXiv:2005.00928*.

Alammar, J. (2018). The Illustrated Transformer. `https://jalammar.github.io/illustrated-transformer/`. [Online; accessed 15-April-2022].

Alammar, J. (2019). The Illustrated Word2vec. `https://jalammar.github.io/illustrated-word2vec/`. [Online; accessed 15-April-2022].

Alammar, J. (2019, Aug 12). The Illustrated GPT-2 (Visualizing Transformer Language Models). `https://jalammar.github.io/illustrated-gpt2/`. [Online; accessed 15-April-2022].

Bahdanau, D., Cho, K. a Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Branwen, G. (2019). GPT-2 Neural Network Poetry. `https://www.gwern.net/GPT-2`. [Online; accessed 22-April-2022].

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. a kol. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, **33**, 1877–1901.

Devlin, J., Chang, M.-W., Lee, K. a Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Hugging Face (2022). OpenAI GPT2. `https://huggingface.co/docs/transformers/model_doc/gpt2`. [Online; accessed 30-March-2022].

Isozaki, I. (2019). Understanding the GPT-2 Source Code Part 1. `https://medium.com/analytics-vidhya/understanding-the-gpt-2-source-code-part-1-4481328ee10b`. [Online; accessed 9-April-2022].

Kazemnejad, A. (2019). Transformer architecture: The positional encoding. `https://kazemnejad.com/blog/transformer_architecture_positional_encoding/`.

Keskar, N. S., McCann, B., Varshney, L. R., Xiong, C. a Socher, R. (2019). Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

Khanna, C. (2021). Byte-Pair Encoding: Subword-based tokenization algorithm. `https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0`. [Online; accessed 15-April-2022].

Leff, A. a Rayfield, J. T. (2001). Web-application development using the model/view/controller design pattern. In *Proceedings fifth IEEE international enterprise distributed object computing conference*, pages 118–127. IEEE.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L. a Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

OpenAI (2019, Feb 14). Better Language Models and Their Implications. `https://openai.com/blog/better-language-models/`. [Online; accessed 10-April-2022].

OpenAI (2019, Nov 5). GPT-2: 1.5B Release. `https://openai.com/blog/gpt-2-1-5b-release/`. [Online; accessed 10-April-2022].

Pennington, J., Socher, R. a Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Radford, A., Jozefowicz, R. a Sutskever, I. (2017). Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*.

Radford, A., Narasimhan, K., Salimans, T. a Sutskever, I. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I. a kol. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, **1**(8), 9.

Rocktäschel, T., Grefenstette, E., Hermann, K. M., Kočiskỳ, T. a Blunsom, P. (2015). Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664*.

Rosa, R., Musil, T., Dušek, O., Jurko, D., Schmidtová, P., Mareček, D., Bojar, O., Kocmi, T., Hrbek, D., Košt'ák, D. a kol. (2021). THEaiTRE 1.0: Interactive generation of theatre play scripts. *arXiv preprint arXiv:2102.08892*.

Saini, M. (2021). Practical Applications of Open AI's GPT-2 Deep Learning Model. `https://medium.com/the-research-nest/practical-applications-of-open-ais-gpt-2-deep-learning-model-14701f18a432`. [Online; accessed 22-April-2022].

Shree, P. (2020). The Journey of Open AI GPT models. `https://medium.com/walmartglobaltech/the-journey-of-open-ai-gpt-models-32d95b7b7fb2`. [Online; accessed 26-April-2022].

Strobelt, H., Gehrmann, S., Behrisch, M., Perer, A., Pfister, H. a Rush, A. M. (2018). Seq2seq-vis: A visual debugging tool for sequence-to-sequence models. *IEEE transactions on visualization and computer graphics*, **25**(1), 353–363.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. a Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, **30**.

Vig, J. (2019). A multiscale visualization of attention in the transformer model. *arXiv preprint arXiv:1906.05714*.

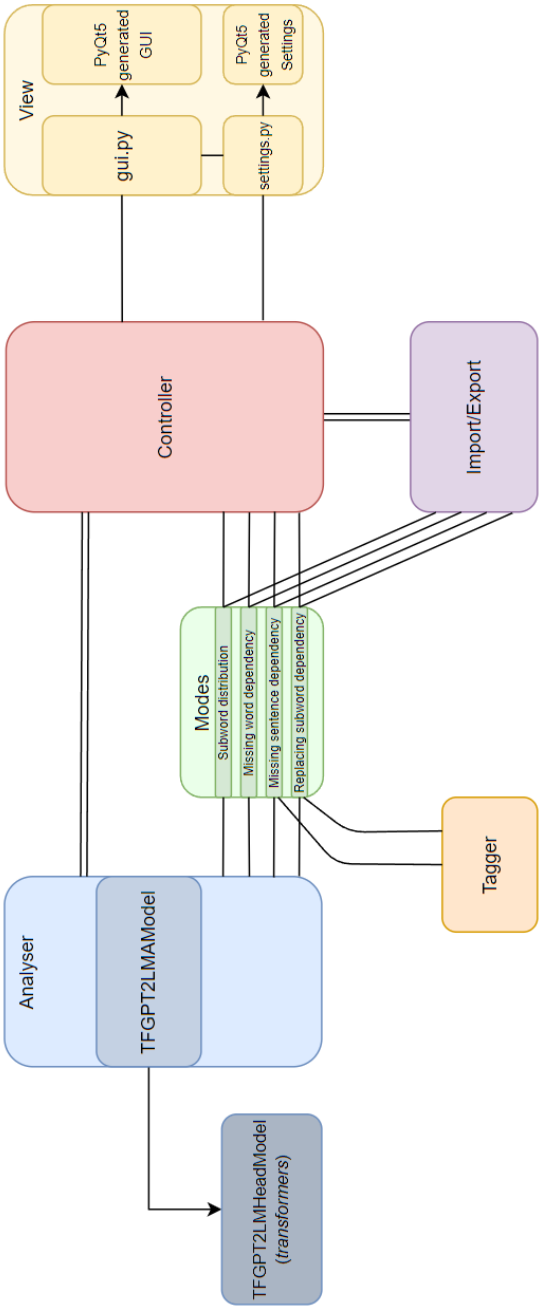
Vincent, J. (2019). OpenAI has published the text-generating AI it said was too dangerous to share. `https://www.theverge.com/2019/11/7/20953040/openai-text-generation-ai-gpt-2-full-model-release-1-5b-parameters`. [Online; accessed 10-April-2022].

Wolf, T. (2021). How to build a State-of-the-Art Conversational AI with Transfer Learning. `https://medium.com/huggingface/how-to-build-a-state-of-the-art-conversational-ai-with-transfer-learning-2d818ac26313`. [Online; accessed 22-April-2022].

Ziegler, D. M., Stiennon, N., Wu, J., Brown, T. B., Radford, A., Amodei, D., Christiano, P. a Irving, G. (2019). Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*.

A. Přílohy

A.1 První příloha



Obrázek A.1: Celková architektúra programu otočená na dĺžku pre lepšiu prehľadnosť