ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Faldina Lénárd**

2022 Hallgató törzskönyvi száma: **T/006180/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Faldina Lénárd**
Törzskönyvi száma: T/006180/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Versgenerálás neurális hálózat segítségével
Poem generation using neural network

Intézményi konzulens: Dr. Kertész Gábor
Külső konzulens:

Beadási határidő: 2021. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és –fejlesztés
specializáció

A feladat

A természetes nyelvfeldolgozás (Natural Language Processing, NLP) a mesterséges intelligencia olyan területe, amely az emberek által használt kommunikációs mód vizsgálatát vette célba. A modern mély tanulási eredmények alapján az elmúlt években nagyszerű megoldások készültek téma, hangulat vagy nyelv azonosítására, klasszifikációra, de a különböző gépi fordító, transzformáló és kivonatoló megoldások is elterjedtek.
A modern NLP megoldások egy családja következtetés helyett szintetikus adatok előállításával foglalkozik, azaz a gyakorlatban a betanítást követően a modell olyan szövegek generálására alkalmas, amely a tanítás során használt adathalmazhoz hasonló.
A dolgozat célja a szakirodalom áttekintését követően egy olyan módszer kidolgozása, amely magyar nyelvű versek generálására alkalmas.

A dolgozatnak tartalmaznia kell:

- a kapcsolódó szakirodalom részletes áttekintését, a kapcsolódó gépi tanulási módszerek bemutatását,
- hasonló problémák, hasonló módszerek bemutatását,
- a feladat pontos specifikációját,
- a megoldás részletes tervét, a módszertan leírását,
- a megoldás implementációjának bemutatását, az eredmények részletes kiértékelését,
- a további kutatási-fejlesztési lehetőségek bemutatását.

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2023. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022.05.12.

Faldina Kénárd
.....
hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
9. sz. melléklet

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Faldina Lénárd

Neptun kód: [redacted]

Tagozat: SZF

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat / Diplomamunka[1] címe magyarul:

Versgenerálás neurális hálózat segítségével

Szakdolgozat / Diplomamunka[2] címe angolul:

Poem generation using neural network

Intézményi konzulens: Dr. Kertész Gábor

Külső konzulens:

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021.03.24	CETLI LEADÁSA, FELADAT KIÍRÁSA	
2.	2021.03.31.	FELADATLAP KIÍRÁS	
3.	2021.05.06.	FORMAI KÖVETELMÉNYEK	
4.	2021.05.14.	ÍRÁSOS ÖSSZEFOGLALÓ TARTALMA	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2021.05.14.

intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Faldina Lénárd ..... [redacted] ..... SZF VAIPALI .....

Telefon: Levelezési cím (pl: lakcím):

[redacted] .....

Szakdolgozat / Diplomamunka[1] címe magyarul:

Versgenerálás neurális hálózat segítségével .....

Szakdolgozat / Diplomamunka[2] címe angolul:

Poem generation using neural network .....

Intézményi konzulens: Külső konzulens:

Dr. Kertész Gábor ....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.28.	SZAKDOLGOZAT 2 KÖVETELMÉNYEI	
2.	2022.04.04.	MEGVALÓSÍTÁS RÉSZLETEZÉSE	
3.	2022.04.25	TESZTELÉS ÁTBESZÉLÉSE	
4.	2022.05.09.	FORMAI KÖVETELMÉNYEK	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 20.....

.....
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

TARTALOMJEGYZÉK

1. BEVEZETÉS

1.1. Szöveg generálás

A Mesterséges Intelligencia egyre szélesebb területen van jelen napjainkban. A nyelvvel is egyre szélesebb körben foglalkozik a mesterséges intelligencia. Az utóbbi években a természetes nyelvek gépi feldolgozása nagy fejlődésnek indult. Utóbbi időkben egyre több helyen találkozunk a természetes nyelv különböző feldolgozásával AI segítségével, gondoljuk csak a különböző beszédfelismerésére vagy mondjuk a szövegértelmezésre. Ennek egy része a természetes nyelv generálásával foglalkozik a mesterséges intelligencia eszközeivel. Ezt a területet nevezzük NLP-nek, azaz Natural Language Processingnek. A természetes nyelvek feldolgozása tulajdonképpen összekötő kapocsként szolgál az emberek és a gépek közti kommunikáció megkönnyítésében. A gép alapvetően képes lenne szavakat felismerni, de mesterséges intelligencia nélkül válaszolni nem lenne képes. Több feladat megoldására is lehet használni például, szöveg generálás, fordítások elvégzése, beszéd felismerés, de akár mondatok, szövegrészletek hangulatát vagy akár összefoglalóját is lehet velük generálni. Tehát összességében az NLP az emberi nyelveknek különböző megoldásait foglalja önmagába, ennek csak nagyon kis része az, amivel a szakdolgozatomban foglalkozni fogok, a szöveg generálás. A szövegek generálása az úgynevezett korpusznyelvészeten alapul, ez igazából annyit foglal magába, hogy a modell nagy mennyiségű rendelkezésre álló adatból következtetéseket von le. Ez alá tartozik még a tartalom kategorizálása, szentimentális elemzések és a szintaktikai elemzések is. Az NLP célja az, hogy a természetes nyelveket kisebb egységekre bontsa le, valamint, hogy megállapítsa a kapcsolatokat, relációkat ezek között az elemek között. Minél több adat (szöveg) áll rendelkezésre, annál könnyebb megtalálni az összefüggéseket az adatok között, vagyis annál pontosabb képet kaphatunk a nyelvről. Felhasználási módjairól el se hinnénk milyen sok van. Mind a hétköznapokban minden pedig az üzleti élet több területén is. Használható akár chatbotok, vagy digitális asszisztensek létrehozásához, de akár gépi fordításhoz is.

Angol nyelven a szöveggenerálás már egészen hihetetlen szinten működik. Képesek olyan szöveget létrehozni, melyről nem lehet egyértelműen eldönteni, hogy egy ember írta vagy pedig egy mesterséges intelligencia. Magyar nyelven ez egy nagyobb feladat. A nyelvünk komplexitása nagyon megnehezíti ezt a feladatot a mérnökök számára. A különböző AI-ok nagyon kedvelt témává váltak az utóbbi években, ezért egyre több kutatás foglalkozik ezzel a témával is. Ennek következményeként nagyon sokat fejlődött a magyar szöveggenerálás. A következőkben egy lehetséges megoldást fogok bemutatni a problémának egy egyedi helyzetére, amely a versgenerálás.

A feladatom célja egy olyan program, amely gépi tanulás segítségével képes már meglévő verseket feldolgozni és ezek szókincsét, illetve nyelviséget megtanulni. Ennek a segítségével pedig egy általunk beírt karakterláncból kiindulva ír nekünk egy művet. Az én esetemben ezek a versek Petőfi Sándor költészetének versei.

2. IRODALOMKUTATÁS

2.1. Használandó módszerek

2.1.1. RNN, visszacsatolt neurális hálózat

Olyan neurális háló, amely ismétlődő jellegű, mivel ugyanazt a funkciót végzi el minden adatbementnél, miközben az aktuális bemenet az előző számításoktól függ. Miután előállította a kimenetet, azt lemásolja és visszatér az ismétlődő hálózatba. Az RNN-ek belső állapotukat felhasználhatják a következő bemenetként. Ez a tulajdonsága teszi lehetővé olyan területeken való felhasználásra, mint például a összekapcsolt kézírás-felismerés vagy a beszédfelismerés. Más hálózatokhoz képest még különbség, hogy az RNN-ben az összes bemenet kapcsolódik egymáshoz. [1]

Aktuális állapot képlete a következőképpen néz ki

$h_t = f(h_{t-1},x_t)$

Aktiválási függvény

$h_t = \tanh(W_{hh}h_{t-1}+W_{xh}x_t)$

W: súly

h: az egyetlen rejtett vektor

W_{hh}:a súly az előző rejtett állapotban

tanh: az aktiválási függvény, amely megvalósítja, hogy az aktiválások a [-1,1] tartományba essenek.

Kimenet

$y_t = W_{hy}h_t$

y_t: kimeneti állapot

Az RNN előnyei

1. Az RNN modellezheti az adatsorokat úgy, hogy feltételezhető, hogy minden minta függ az előzőtől.
2. A visszatérő ideghálózatot a konvolúciós rétegekkel is alkalmazzák, hogy kibővítsék a tényleges pixelszomszédságot.

Az RNN hátrányai

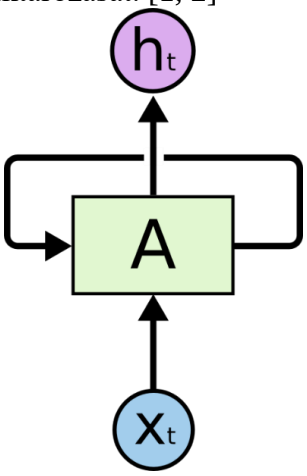
1. Eltűnő és felrobbanó gradiens problémák
2. RNN képzése nagyon nehéz feladat
3. Nem képes feldolgozni nagyon hosszú sorozattokat abban az esetben, ha tanh vagy relu aktiválási függvényt használ [1]

2.1.2. LSTM (Long Short-Term Memory)

Azaz a Hosszú-rövid távú memória. A visszatérő ideghálózatok módosított verziója, amely megkönnyíti a memóriában a múltbéli adatokra való visszaemlékezést. Az RNN eltűnő gradiens problémát ez oldja meg.

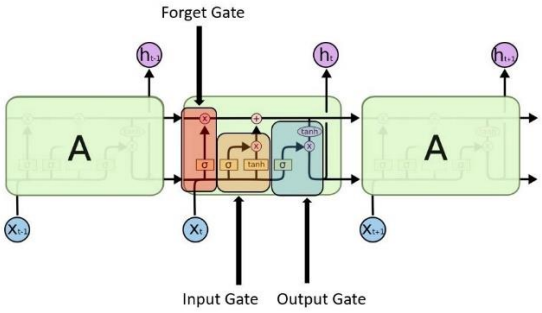
Azért van erre szükség, mert van olyan eset amikor több összefüggésre van szükségünk. Ezek az összefüggések és információk között egy nagy rés keletkezik. Ezek a rések

okozhatják a hosszú távú függőségi problémákat, az LSTM-eket kifejezetten úgy tervezték, hogy elkerüljék a hosszú távú függőségi problémát.
Főként az ismétlődő neurális hálózatokban használják. A hálózatban hurkok vannak ezzel lehetővé teszi az információ elraktározását. [1, 2]



1. ábra LSTM egy cellája[1]

Alapvető felépítése a celláknak:



2. ábra LSTM Cellák felépítése[2]

Az LSTM-en belül 3 kapu található,

1. Bemeneti kapu:

   Felfedezi, hogy melyek azok a bementi értékek, amelyek használni kell a memória módosításához. 2 részből áll, először szigmoid függvény segítségével eldönti, hogy mely adatokat kell tovább adni ennek az értéke 0,1 lehet. Ezután a tanh függvény eldönti, hogy milyen súllyal kell az adott értéket számolni [-1,1] intervallumon.

2. Felejtő kapu:

[1] https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e
[2] http://home.mit.bme.hu/~engedy/NN/NN-RNN-LSTM-ESN.pdf)

Megkeresi azokat a részleteket, amelyeket el kell távolítani az adott blokkból szigmoid függvény segítségével. Ez függ az előző állapottól és a tartalom bemenetétől. Visszatérési értékként 0 vagy 1 értéket veheti fel, ahol a 0 a felejtést jelenti, az 1 pedig azt, hogy tartsuk meg.

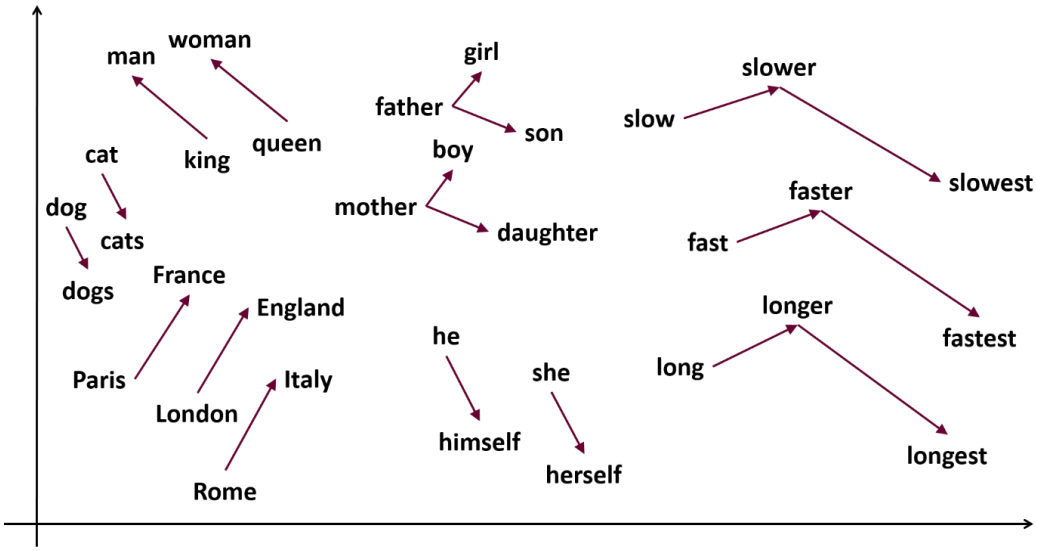
3. Kimeneti kapu:

   A bemenet és a memória blokk segítségével számolja ki a kimenetet. A szigmoid függvény visszaad egy értéket, amely 0 és 1 lehet. A tanh függvény pedig visszaad egy fontossági értéket [-1,1] tartományban. [2]

Ez a módszer arra lesz nagyon hasznos, hogy a függéseket és összefüggéseket, amiket egyszer már felépítünk azt hosszú ideig elraktározza a memória. Ezáltal minél több összefüggés kerül bele, annál pontosabbá válik majd a program. Erre a módszerre azért is lesz szükség, mivel ez a leghatékonyabb módszer az idősor modellezésre, osztályozására.

2.1.3. Word2Vec algoritmus

A Word2Vec alapja az eloszlási hipotézis. Képes rögzíteni egy szó összefüggését egy dokumentumban, szemantikai és szintaktikai hasonlóságot és kapcsolatot más szavakkal. Megpróbálunk egy adott szót előre jelezni annak kontextusa (CBOW) alapján, vagy egy adott szó alapján előre jelezzük a környező környezetét (Skip-Gram). A módszerrel először is felírunk egy szót számokkal egy tömbben. Ezek a számok fognak rámutatni a mi szavunkra. Ezek után, ha az adott szóhoz találunk egy szinonimát vagy valamilyen kifejezést, ami közel áll hozzá akkor a számaik nem lesznek egymástól messze, megjelenítésnél ez nagyon látszódik. [3]



3. ábra Szóvektorok ábrázolása

Így van lehetőségünk a kapcsolatokat jelölni a szavak és fogalmak között. Alapvető működéséhez tartozik, hogy egy rejtett réteg van benne. Kiválasztunk egy adott szót a mondat

közepéről és megpróbálja megjósolni, hogy a szó körül milyen valószínűséggel vannak azok a szavak. Például, ha az adott szó a „bolt”, akkor a „vásárol” szónak sokkal nagyobb valószínűséget tulajdonít, mint például a „park” szónak. A modell lényegében szópárokat fog elmenteni ebből is jön az egyik legnagyobb hátránya, hogy hajlamos hatalmas adatbázist felépíteni. Így nagy szövegeknél lehetséges, hogy lassú futási időt eredményez és nagy tár helyet igényel. Ha kettő szó hasonlít akkor kimenetként nagyon hasonló eredményeket kapunk vissza. A Word2Vec egyetlen rejtett réteget tartalmaz, fontos, hogy ebben a rejtett rétegben annyi neuron van ahány szóból áll a tanításhoz használt szókincs. [3]

A hasonlóságra használható képlet, ahol a két objektum A és B, ezt a matematikában koszinusz hasonlóságnak nevezzük.

$$sim(A, B) = cos(\emptyset) = \frac{A \cdot B}{\|A\|\|B\|}$$

Tomas Mikolov módszere számomra talán az egyik legfontosabb a projektem közben. Ennek a módszernek a segítségével szemantikai teret fogok tudni összeállítani a versek szavaiból, amiket kinyerek a magyarlánc nevű program segítségével a szövegből. Így lehetséges lesz az, hogy a szavak „asszociáljanak” egymásra és így lesz lehetőségem összegyűjteni az összeillő szavakat, ezekből szókapcsolatokat tudok generálni, majd a végén a teljes szöveg is összeállhat belőle.

Word2Vec módszernek két fő létrehozási módja van a Skip-Gram és Continous Bag of Words.

2.1.4. Continous Bag of Word módszer

A CBOW a szópárok közötti kapcsolatok megtanulására szolgál. Ebben a tanulási módszerben a kontextust több szó is képviseli egy adott célszóhoz. Például, „macska” és a „fa” összefüggő szavak, „felmászott” pedig a célszó.

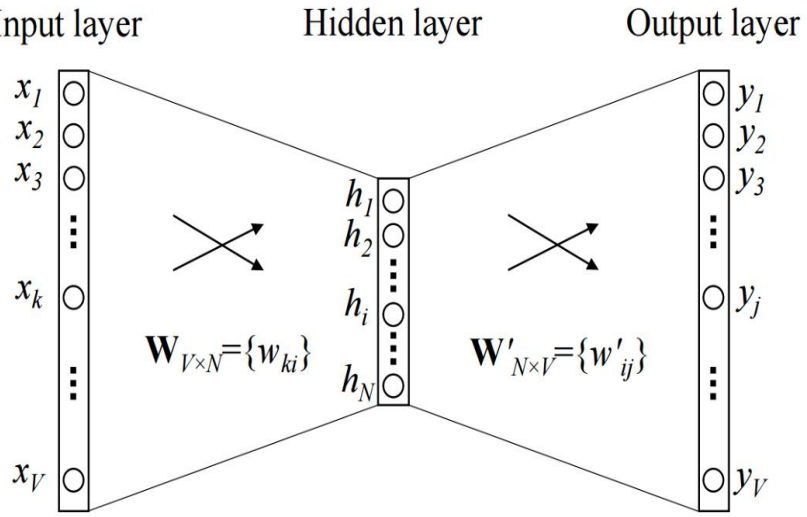


Figure 1: A simple CBOW model with only one word in the context

4. ábra CBOW model

2.1.5. Skip Gram model

Ez a módszer úgy működik, hogy beírjuk a cél szót a hálózatba, tehát megfordítjuk a cél- és a kontextusszavak használatát. A modell a valószínűségi eloszlást adja ki. Így nem egy adott szót ad ki, hanem hogy mely szavak illenének bele a szövegkörnyezetbe. Ez a modell hurokba veszi az egyes mondatok szavait, vagy megpróbálja a jelenlegi szót használni a szomszédok (a kontextus) megjóslására. [4]

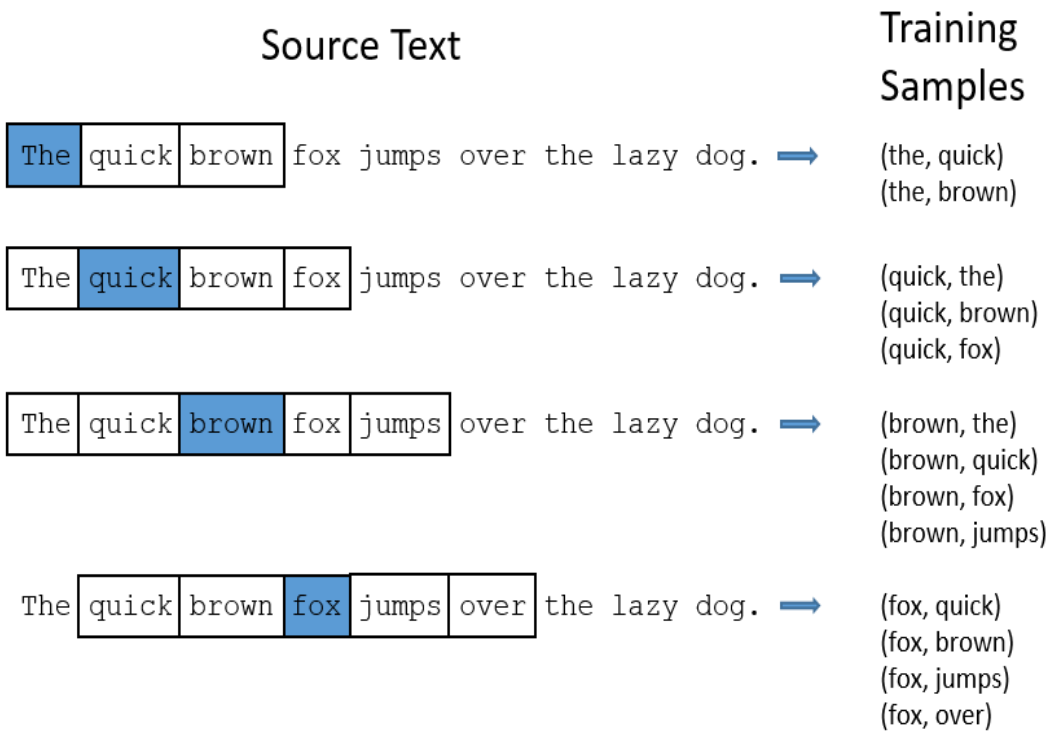
Képletek:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

(1) Ebben az esetben a C az oktatási környezet mérete, w_t:az ablak középső szavak
A c több képzési példát is tud eredményezni, így nagyobb pontosságot tud eredményezni az edzésidő. [5]

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^T v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_{w_O}}^T v_{w_I}\right)}$$

(2) Ebben az esetben SoftMax függvényt használunk. v_w és a v'_w a kimeneti illetve bementi w vektor ábrázolása,. W: a szókincsben, szótárban szereplő szavak száma Ez a formula nem praktikus ugyanis a számítási költsége log p (w_O | w_I) a W-től függ ami gyakran nagy szám lehet. [5]



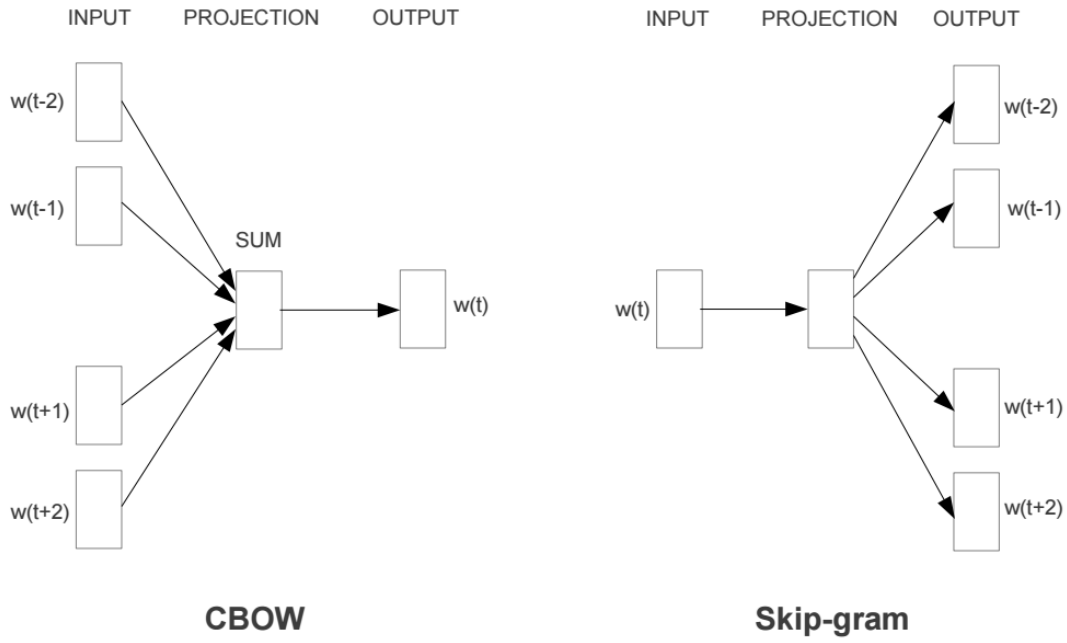
5. ábra Ablakos módszer példa[3]

Ezen a képen azt láthatjuk, hogy a módszer a szövegen létrehoz egy ablakot, amelynek a mérete jelen esetben 5. A középső szó szövegkörnyezetét vizsgáljuk meg és összepárosítsuk az ablakon belül lévő többi szóval. Lényegében szópárokat hozunk létre. Ezeket a szópárokat felhasználva lehet megnézni és megjósolni, hogy egy adott szóhoz milyen szavak állnak általában közel a szövegkörnyezetekben.

[3] https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b

A két módszer felépítés béli különbsége

A CBOW esetén több bemenet van és egy összegzés segítségével kiválasztja azt az értéket, adatot, amit a legvalószínűbbnek talál, a mellette lévő szavakból ez az ábrán t-vel van jelölve.
A Skip-Gram bemenetnek egy értéket kap és kimenetnek több megoldást ad. Azokat, amelyeknek a legnagyobb a valószínűsége, hogy az adott bementi szó mellett találjuk egy szövegkörnyezetben.



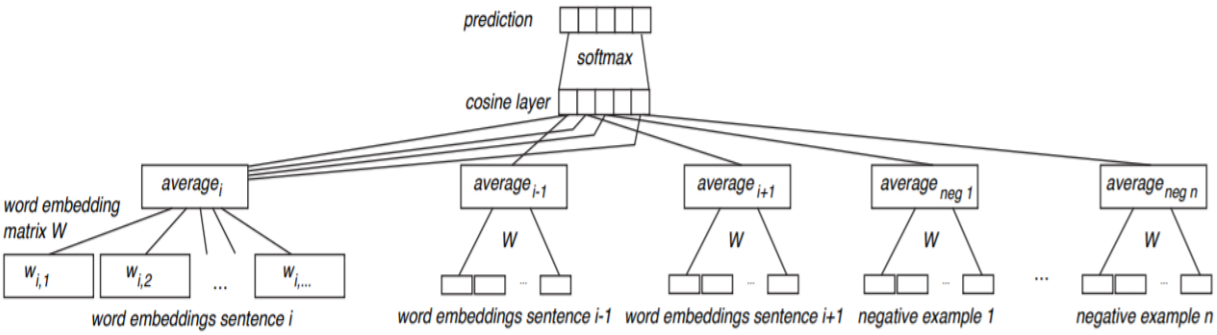
6. ábra CBOW - Skip-gram összehasonlítása

2.1.6. Sziámi CBOW

Ennek a modellnek a lényege, hogy egy neurális hálózat, amelynek a célja a hatékony és kiváló minőségű mondatbeágyazásokat becsüljön meg. Mégis mit jelent a hatékonyság? A hatékonyság azt jelenti ebben az esetben, hogy a jelentésükben közel álló mondatok hasonlóak legyenek egymáshoz és a szemantikailag különböző mondatok eltérőek legyenek. A Word2Vec módszer erre a problémára nem hatékony, szemben a sziámi CBOW-al. Két mondatra (s_i és s_j) definiálunk egy P valószínűséget, ami megmutatja a szemantikai hasonlóságot számosítva a két mondat között. A P valószínűséget a következőképpen számoljuk ki a SoftMax függvény használatával: [6]

$$p\emptyset(s_i, s_j) = \frac{e^{\cos(s_i^{\emptyset}, s_j^{\emptyset})}}{\sum_{si \in S} e^{\cos(s_i^{\emptyset}, s_j^{\emptyset})}}$$

Teszteléséhez felállítunk több tesztet pozitív és negatív példákkal is. Egy teszt két mondatból áll. Kiszámítjuk hozzájuk a szóbeágyazásokat. Azok a szavak, amelyek hiányoznak a szókészletünkből ezentúl elhanyagoljuk. A két mondat koszinuszos hasonlósága adja meg a végső hasonlósági számot. [6]



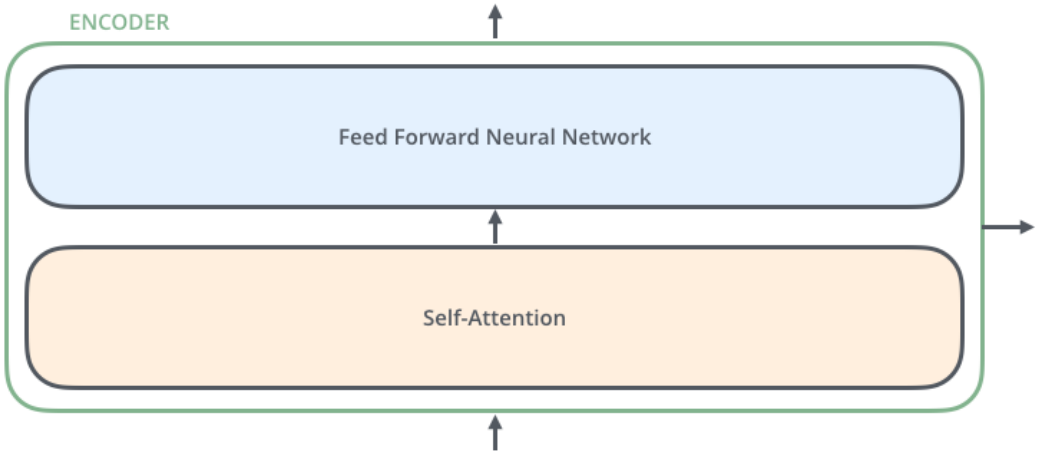
7. ábra Sziámi CBOW felépítése [4]

2.1.7. Transzformátorok

A transzformátorok a verses forma, illetve a szöveg releváns részeire figyel. Ennek a segítségével értelemes mondatokat és szókapcsolatokat lehet létrehozni, illetve segít a verses forma kivitelezésében. Ezen felül még gyorsítja is a fordítást. Ez egy plusz réteg a

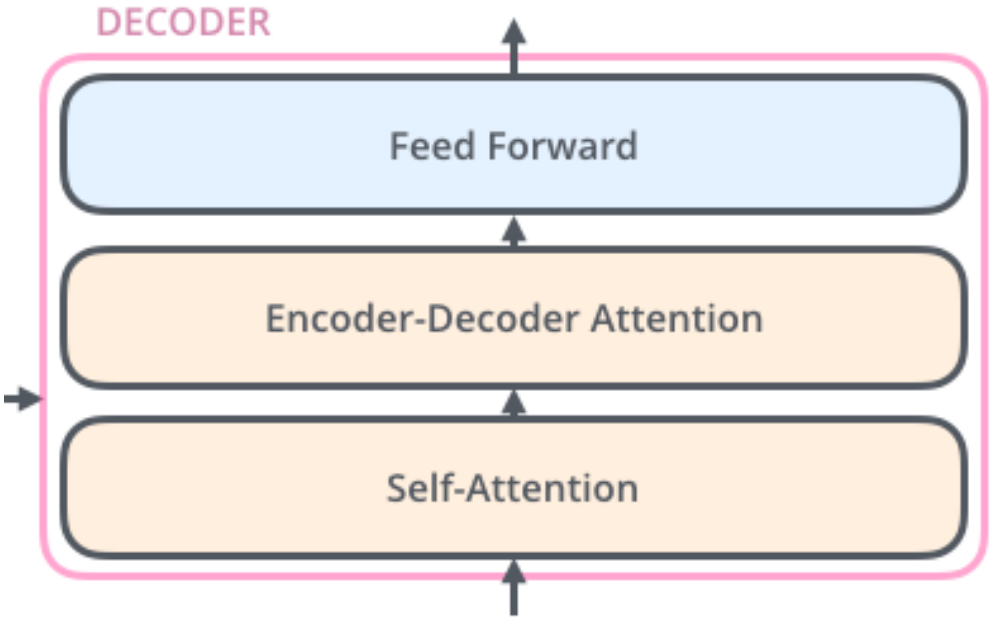
[4] https://arxiv.org/pdf/1606.04640.pdf

programban. Felépítésében több egymás után csatolt encoderből és ugyanennyi hasonlóan felépített decoderből áll, minden kódoló hasonlít egymásra és azonos architektúrával rendelkezik. Minden encoder réteg még két rétegből áll egy Self-Attention , illetve egy Feed Forward Neural Network rétegből.



8. ábra Transzformátor encoder

A szóvektorok először Self-Attention rétegen haladnák át, mindegyik a saját útján. Ezek között összefüggések vannak.



9. ábra Transzformátor decoder

A decoder rétegek felépítése a következő, az első réteg, amelybe az adat beérkezik egy úgynevezett Self-Attention réteg ebben az a decoder kiszámolja az Self-Attention értékét mátrixok segítségével. Ezután az adat tovább halad egy Encoder-Decoder Attention rétegbe, ahol további számításokat végez a modell és végül pedig egy Feed forward rétegen keresztül kikerül a decoderből. [7]

2.1.8. Transzformátor és LSTM/RNN összehasonlítása

A transzformátorok, LSTM és az RNN is jól használható, viszont mégsem mindegy melyiket választjuk. Külön-külön már kifejtettem a módszerekről szükséges legfontosabb tudnivalókat, de mégis melyik milyen esetben használható jól és miért?
A transzformátorokat azzal a céllal vezették be hogy párhuzamosítható legyen a számítás és ezzel csökkenteni tudják a tanítási időt, illetve az volt a cél, hogy csökkentsék a teljesítmény csökkenését a hosszú távú függőségek következtében. Ebből is adódnak a főbb jellemzői. Nem szekvenciális, nem szóról szóra, hanem mondatok egészét képes feldolgozni. Self-attention, ez egy új egység a transzformátorokban, ezt a mondatokban megtalálható szavak közötti hasonlóság számításra használjuk. A pozíció beágyazás pedig egy másik újítás az LSTM és RNN-hez képest. Ez az ismétlődések elkerülésére hasznos, Az alap ötlete az, hogy előre rögzített és betanult súlyokat használjunk, amelyek információkat tárolnak különböző tokenek formájában a mondaton belül. Mivel a transzformátor nem szekvenciális ezért nem jelennek meg benne hosszú távú függőségi problémák, ezért a transzformátorok nagyon jól használhatóak nagy adatmennyiségek, illetve nagyobb kontextusú szövegek generálására és feldolgozására. Mivel mondatok egészét dolgozzák fel, nem csak szavak összefüggését, ezért nem áll fenn az információ vesztés esélye. Multihead-attention és a tokenek pedig együtt lehetővé teszik, hogy a szavak közötti kapcsolatokat el tudjuk tárolni és ezeket később felhasználni. Szöveg generálásra jelenleg az egyik, ha nem a legjobb transzformátor a GPT-2.

A rekurrens neurális hálókat és az LSTM-et egy csoportba veszem mivel alapvető tulajdonságaiban azonosak. Mind a kettő szekvenciális feldolgozással működik, a mondatokat szóról szóra kell feldolgozni. Múltbeli rejtett állapotokon keresztül őrzi meg a múltbeli információkat. Markov tulajdonságot követik, azaz minden állapot csak a korábban látott állapottól függ. Mivel ez a kettő módszer szekvenciális, ezért nem lehet párhuzamosan feldolgozni őket. egy adott szó kódolásához szükségem van minden esetben az előző szó rejtett állapotaira ezért azt ki kell számolni előtte. Ez egy elég nagy hátránya az LSTM és RNN modellnek. A második tulajdonság az jelenti, hogy az információk ebben a modellben megmaradnak, de a gond, az, hogy agy szó kódolása csak a következő szónak a kódolására fog elég erősen kihatni és egy idő után ez teljesen elenyésző mérték lesz, ez miatt nagyobb kontextusokra nem jól alkalmazható. LSTM-ben ez a tulajdonság bővítve lett, de az sem teljesen jó megoldás. Lehet ezt a problémát enyhíteni amennyiben a kódolást a mondat két végéről is elvégezzük így az mondat eleje és vége is nagy súllyal fog megjelenni. [8]

Összességében azt lehet mondani, hogy a transzformátor az, ami jobb a másik kettő említett modellnél, mert elkerüli a rekurziót a mondatok egészének feldolgozásával és a szavak közötti kapcsolatok megtanulásával, köszönhetően a multihead-attention és tokenek használatának. Viszont a transzformátoroknál nagyon fontos a megfelelő paraméterezés, mert csak a rögzített mondathosszig fog megfelelően működni. RNN-ket hosszú sorozatokon lehet célszerű használni, mert jelentősen rövidebb futási időt eredményeznek a transzformátorokkal szemben. [8]

2.1.9. BPE tokenizáló

A Byte-level BPE, azaz Byte-level Byte-Pair-Encoding, egy 2015-ben bevezetett univerzális tokenizáló, ami lehetővé teszi, azt, hogy nyelvtől függetlenül tudjuk megtalálni és kialakítani a szöveggeneráláshoz szükséges tokenjeinket. A BPE tokenizáló adja az alapját a GPT modelleknek.

A BPE használ egy pre-tokenizert, ami a lehető legegyszerűbb is lehet, például szavak szétválasztás, de bonyolultabbat is lehet alkalmazni, az úgynevezett szabály alapú tokenizálást is. A pretokenizálás után minden a korpuszban megtalálható összes szóra és szimbólumra létrehoz egy elfordulási számot. Ez addig fut amíg azt paraméterben megadjuk. Ez után kigyűjti a gyakori szimbólumokat egymás mellet és ezeket ötvözi, így azok a szimbólumok innentől kezdve egynek számítanak. Gyakorlatban ez a következőképpen fog kinézni:

("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

Ezek az alapszavak, amelyeken elvégezzük a tokenizálást. Kezdetben az összes betűt tartalmazza a tömbünk. A számok a szavak mellett az előfordulásukat jelentik. Ezek után mivel az „u” és „g” karakterek többször is egymás mellett helyezkednek el és tegyük fel, hogy a szövegkörnyezetben ezek a legfrekventáltibbak, ezért ezeket egyesítjük, innentől kezdve a következőképpen néz ki a szótárunk:

("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)

Következő leggyakoribb szókapcsolatok az „un” és a „hug”, ezért mint az előzőekben, itt is elvégezzük az egyesítésüket. és ezt folytatjuk tovább. Végeredményében így fog kinézni a tokenizált szótárunk:

("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)

A tokenizálás során előfordulhat speciális token is, amelyet az „<unk>” tokennel fogunk ellátni, ez az jelenti, hogy az adott szókapcsolat nem található meg a szótárunkban, azaz ismeretlenként jelöljük meg. [9]

2.2. Hasonló problémák bemutatása

2.2.1. Paul Mooney megoldása

RNN segítségével egy adott előadó stílusában ír verset Python programozási nyelvel. A bemenete az előadó dalszövegei. Markov lánc felhasználásával kiszámolja annak a valószínűségét, hogy egy szó milyen valószínűséggel követ egy másikat (szó1→szó2). Az LSTM-et pedig a generált szöveg következő sorainak kiválasztására használja, több szempont figyelembevétele alapján. Például, a szótagszámok és a rímrendszer felépítését figyelembe véve. Ezek a módszerek segítségével fel lehet építeni ritmikus és logikus mondatokból álló dalszövegeket és verseket is. Először is analizálni kell a bemeneti szöveget, ami ebben az esetbene egy .txt fájl. A szavak gyakoriságát kigyűjti, a rímeket pedig csak az utolsó két betűre veszi figyelembe és ezeket is elraktározza. Később ezek segítségével építi fel a szöveget. Több algoritmust is találtam ehhez a megoldáshoz, némelyik alapnak jó lesz az én projektemhez, mert én is egy adott költőtő stílusában szeretnék új szöveget írni, miközben nekem is figyelnem kell a rímekre és a dallamosságra egyaránt. [10]

2.2.2. Kínai vers írás neurális hálózattal

Először is kiválasztottak 2000 versnek a négysoros versszakait. A tanítási fázisban szegmentálták a szavakat, majd pedig számítottak úgynevezett TextRank-ot minden szóhoz. A legmagasabb TextRank-al rendelkező szó lesz a kulcsszó egy adott sorban. Később ezekkel a kulcsszavakkal tanítják az ismétlődő neurális hálózatot. A kulcsszavak kigyűjtése után egy táblázat felépítése következik, melynek a felépítése: az első oszlop, a sor adott kulcsszava, a második oszlop, az előző sor a versszakban, a harmadik oszlop pedig az aktuális sor. Ezek az adatok alapján lehet megtanítani a hálózatot. Ebből a módszerből számomra hasznos a sorok közötti összefüggések kigyűjtése és a sorok kulcsszavainak megkeresése. [11]

2.2.3. Versek generálása visszacsatolt neurális hálózattal

Egy olyan neurális hálózatot mutat be, amely képes verseket generálni. Ez a vers generálásból egy egyszerűbb példát mutat be. Első lépésként újra kell értelmezni a versírás problémáját, ami azt jelenti, hogy ez egy előrejelzési probléma. A verseknek témája van és ezért azt szeretnénk megjósolni, hogy mit írna az adott témáról a költő. Második lépésként redukálni kell ezt a nagy problémát például, mindig csak a következő betűt próbáljuk meg megjósolni. Ehhez egy olyan neurális hálóra van szükség, amely tetszőleges számú karaktert képes beolvasni, erre egy jól használható módszer az RNN. Ebben a megoldásban fent áll a hiba, veszteség, ami azt jelenti, hogy a várható és a tényleges előrejelzés között lesz különbség. Viszont a hálózat ezen a hibán minden futással tud javítani. Ha ez megvan akkor visszatérhetünk a nagy, eredeti problémára, a teljes vers generálására. Bemenetként kap egy tárgyat a hálózat futásakor ezeknek a karaktereknek a kimenetét figyelmen kívül hagyjuk, kivéve az utolsó karaktert, onnan kezdjük el felépíteni. Vizuálisan úgy lehet elképzelni, hogy a bemenet tehát a bal oldal a vers tárgy, tehát egy tárgyvektor, a jobb oldal pedig a dekóder, ami a verset fogja generálni. Ezt a módszert használják gépi fordító rendszerekben is. [12]

2.2.4. Szöveg generálás LSTM segítségével Python-ban Keras használatával

Ebben a projektben karakter alapú szöveggenerálást mutatják be Keras segítségével. Lewis Caroll, Alice Csodaországban című könyvet vették alapul. A program először beolvassa majd normalizálja a szöveget, ezután kiválogatja az egyedi karaktereket és azt elmenti egy tömbben. Az így kapott szöveget felosztja 100 hosszú karakterláncokra és meghatározza a következő karaktert. Ez az az ablakos módszer, amelyet én is használok a projektem során. A tanítóadatbázist ezekből a karakterláncokból építi fel, olyan módon, hogy minden karakterhez a fentebb említett kiválogatott karakter tömbjével megegyező hosszúságú 0 és 1 értéket tartalmazó vektort hoz létre. Azon a helyiértéken lesz egyes, ahol a betű megegyezik a kiválogatott karakterek tömbjében lévő betűvel. A model LSTM réteget tartalmaz, illetve softmax aktivációs függvényt használ, amely az ilyen felépítésű tanító adatbázishoz tökéletes. A generálás egy előre megadott seed alapján történik és ebből generálja le a neurális hálózat a szöveget betűről betűre. Számomra ez a projekt azért hasznos meg az adatbázist én is hasonló ablakos módszerrel építem fel, de a különböző karaktereket, amelyek az én projektemben szavak, nem tudom kigyűjteni egy tömbbe, mert hatalmas lenne a memória igénye a programnak. Ez a probléma miatt megfelelő nekem a szóvektorok használata. [13]

2.2.5. LSTM használata árak megtippeléséhez

Ebben a projektben árakat tippelnek meg az eddigi adatok használatával. Az adatokat beolvassák, normalizálják, majd pedig vektorokká alakítják. Ezek a vektorok alapján tippel a program. Ez a projekt különbözik az általam választott projekttől, de mégis hasznos volt számomra mivel vektor alapú predikciót használ, amit én is alkalmazok a szóvektorok miatt. Ezen kívül itt ismerkedtem az 'mse' loss függvénnyel illetve az 'adam' optimizáló algoritmussal is melyek hasznosak az én projektem szempontjából is. [14]

3. MEGVALÓSÍTÁS TERVE

Petőfi Sándor versei alapján a tervem egy olyan program, ami az ő stílusában képes írni egy művet. A programom alap adatai tehát Petőfi versei. A magyar nyelv feldolgozása egy nehéz feladat az informatikában, mivel a nyelvtan az anyanyelvünkben bonyolult és nagyon összetett. Meg kell találni a szavak közötti kapcsolatokat, hogy milyen valószínűséggel állnak bizonyos szavak egymás mellet a szövegkörnyezetbe.

Egyetemi tanulmányaim alatt, több projektet is csináltam különböző tantárgyak keretin belül ebben a témában. Ezek alapján úgy gondolom, hogy a legjobb a megoldás egy transzformátor használata lesz azon belül is GPT-2 modell, mivel jelenleg talán ez a legjobb modell az informatikában különböző NLP és szöveg generálást végrehajtó feladatokhoz. Azért is gondolom ezt, mert a magyar már előre generált modellek, corpusok Wikipedián keresztül lettek tanítva így a régies szavak nem találhatóak meg benne, viszont nagyon sok felesleges szót tartalmaz. Tokenizálókból is sajnos hasonló a helyzet. A már elkészített tokenizálók se működnek megfelelően ilyen szövegen.

Programozási nyelvnek a python-t választottam, mert a Keras, illetve NumPy könyvtárak segítségével egyszerűbben megoldható a neurális hálózatok létrehozása. Nagy segítségemre lesz még egy a githubon is megtalálható transzformátorokat tartalmazó könyvtár a Huggingface/transformers könyvtára. (https://github.com/huggingface/transformers)

Első lépésként nagyon sok adatot kell gyűjteni. Ezek, mint említettem Petőfi versei lesznek, jelenleg csak egy évben írt összes verse lesz. Ezeket egy külön mappában szeretném eltárolni, így, ha a jövőben bővíteni szeretném a corpust, elegendő lesz a txt file-t bemásolni ebbe a mappába.

Alapvetően 2 osztályt tervezek, egyik egy tokenizáló osztály lenne. Ezen belül fogom megadni a tokenizálót, a speciális tokeneket, tanítani, majd elmenteni a modellt, hogy ne kelljen minden alaklommal lefuttatni, ugyanis nagyon nagy számítás igénye van egy ilyen megoldásnak.
A másik osztály sokkal több dologért lesz felelős. Itt történik meg a szövegbeolvasás, a tanítás, modell létrehozása, tanítása lehetőség szerint videókártyán, paraméterek megadása és hangolása, optimalizáló és loss függvény meghatározása és a végén természetesen a szöveg generálása.

4. MEGVALÓSÍTÁS

A megvalósítás során több különböző megoldással is próbálkoztam, mire megtaláltam azt az egyet, amely használható a feladathoz, a transzformátorokat. Alapvetően négy részre lehet bontani a feladat megoldását. Ezeket kettő különböző .py file-ba szerveztem ki. Először az adatgyűjtést kellett elvégeznem, ehhez az online mindenki számár elérhető Petőfi összes költeményét használtam. Ezek után jöhetett a modell építése és tanítása. Itt fontos volt az, hogy GPT-2 modellel szeretném építeni a neurális hálózatot, ezért a tokenizálásnál BPE tokeneket, azaz Byte-Pair Encoder Tokenizert használtam. Ez a BPE az, amit a GPT modellek használnak. A transzformátor könyvtár beépített függvényeivel tudtam ezt megcsinálni.

4.1.Tokenizálás

A tokenizáláshoz használt paramétereim a következők voltak. Először, mint említettem BPE tokenizálót használtam, normalizáláshoz pedig szekvenciális NFKC normalizálót használtam. Ez Unicode Normalizzation Form C. Működését a felhasznált módszerek részben részleteztem. A decoder és a pre_tokenizer pedig Byte Level volt. Ezek maguktól értetődőek voltak, mivel Byte level tokenizálót használtam. A tanításhoz a szavak számát személyre szabtam, itt 55000 volt a megfelelő paraméter ennyi kapcsolatot épített fel a tanítás. Speciális tokenjeim pedig a következők voltak. Összesen öt darabot használtam:

- „eos_token” - end of sequence token melyet a szekvenciák kezdetének jelölésére használ a modell,
- „bos_token” -begin of sequence token melyet a szekvenciák kezdeténél használ a modell,
- „unk_token” - unknown token, melyet a modell számára ismeretlen értékeknél használ,
- „pad_token” - padding token, melyet a szekvenciák darabolásához használ a modell
- „mask_token” - Ez egy speciális token a nyelv modellezésére

Természetesen a modellt el is mentettem, hogy ne kelljen minden egyes futtatáskor a számítógépnek ezeket a számításokat elvégezni, ezzel mind időt és memóriát spóroltam. Kimenetként 2 filet kapok. Egy szöveges filet „merges.txt” néven, itt vannak eltárolva a kapcsolatok a szavakban, ahogy fentebb a felhasznált módszerekben részleteztem. A másik epdig egy json file, „vocab.json”, Ebben a file-ban vannak elmentve a szavak, illetve szórészletek és hozzájuk rendelve a ID-k 1-től 55000-ig, természetesen nem csak a szavak, hanem írásjelek és a fentebb említett speciális tokenek kapnak itt ID-kat. Ez lesz az a file, amely később segíteni fogja a predikciót. A létrehozott szakat tartalmazó file következőképpen néz ki:

```
Ġcsillag atlan
Ġcsillag jai
Ġcsillag odat
```

4.2.A GPT modell

Ezek után következhetett a GPT modell létrehozása és tanítása. A programot videókártyán futtatom, hogy gyorsabb legyen. Egy külön mappában van elmentve a verseket tartalmazó txt file, azért választottam ezt a megoldást, mert így bővíteni is lehet ezt és nem csak egy txt file-ra működik, például mellé rakhatnám Arany János műveit is, ezzel érdeksebbé téve a generált szöveget. Mielőtt elkezdődik a GPT modell generálása először a fentebb említett tokenizáló függvények futnak le egy külön python fileba. Az onnan kapott eredmények lesznek a GPT-2 modellnek a Tokenizálója ugyanazokkal a speciális tokenekkel hozzáadva, amiket fentebb említettem (eos_toke, bos_token, unk_token, pad_token, mask_token).
A modellnek ezután a konfigurációját kell beállítani, itt megadni a bos_token_id-t és az eos_token_id-t, illetve a könyvtár méretét, ami már a fentebb említett 55000. Ezt configot természetesen hozzá kell rendelni a modellhez, ami Egy TFGPT2LMHeadModel(config)-ként jelenik meg a kódban. Itt lett létrehozva a GPT-2 modell.

Ezek után következik a tanító adatbázis tokenizálása. Itt is ugyanazt a file-t használom, ahol az összes verset tárolom, ennek igazából annyi az oka, hogy mindenképpen csak ezeket az adott szavakat szerettem volna eltárolni és nem akartam, hogy olyan szavak is megjelenjenek a generált szövegben, amelyeket akkoriban Petőfi nem használt. Annyival alakítottam át, hogy ne teljesen ugyanaz legyen, hogy megkavartam a szekvenciákat, ezzel kicsit javítva a tanítás minőségén, annak ellenére, hogy ugyanazt az adatbázist használtam. A már részletezett ablakos módszert alkalmaztam a tanító adatbázis felépítéséhez.

Optimalizálónak egy egyéni paraméterekkel felszerelt Adam optimalizálót alkalmaztam, melynek értékeinek a következő paramétereket adtam meg:

- learing_rate=3e-5
- epsilon=1e-08
- clipnom=1.0

Loss függvénykent egy beépített függvényt használtam, amely a SparseCategoricalCrossentropy volt.

Metric pedig ismét egy beépített volt a loss, SparseCategoricalCrossentropy-hez kapcsolódó „accuracy”.

A modell ezekkel az értékekkel lett létrehozva. Tanítását 10 epochon keresztül futtattam, mely során a loss függvény 6,5-ről 3,4-re csökkent az accuracy pedig szépen egyenletes növekedett. A tanítás időtartama videokártyán futtatva körülbelül 3-4 órát vett igénybe.

```
Model: "tfgpt2lm_head_model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 transformer (TFGPT2MainLaye  multiple                 128082432
 r)

=================================================================
```

10. ábra GPT-2 model

A modellt elmentettem a program egy almappájában, hogy ne legyen szükséges újra és újra lefuttatni a hosszú tanítást. Ezt az elmentett modellt és config file-t kiolvasom és következhet az utolsó része a megvalósításnak, ami a modell alapján történő szöveg generálás.

4.3.Szöveg generálás

Következhet a talán legizgalmasabb része a szöveg generálása, több lehetséges módját is megvalósítottam a kódban, hogy ezeket később össze lehessen hasonlítani, ezeket fogom a következőben részletezni. Minden generálásnak az alapja egy kezdeti szöveg, melyet mi magunk adunk meg. Ezt a szöveget kódoljuk a tokenizáló segítségével, természetesen ugyanazzal, melyet ez előtt is használtunk és a return_tensors paraméterét be kell állítani „tf" -re.

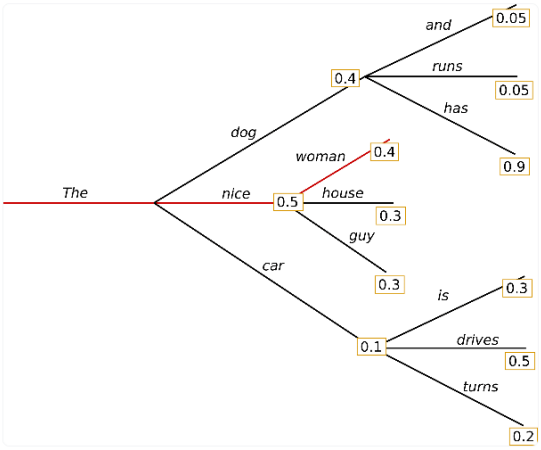
4.3.1. Greedy search

A legegyszerűbb szöveg generálási módszer. Működése egy egyszerű valószínűség számítson alapul:

$$argmax_w P(w|w_{1:t-1})$$

11. ábra Greedy search képlet

Minden alkalommal a következő szót válassza ki a modell, minden alkalommal azt, amelynél a legnagyobb a valószínűsége, hogy az adott szó után az következik.



12. ábra Greedy search fa

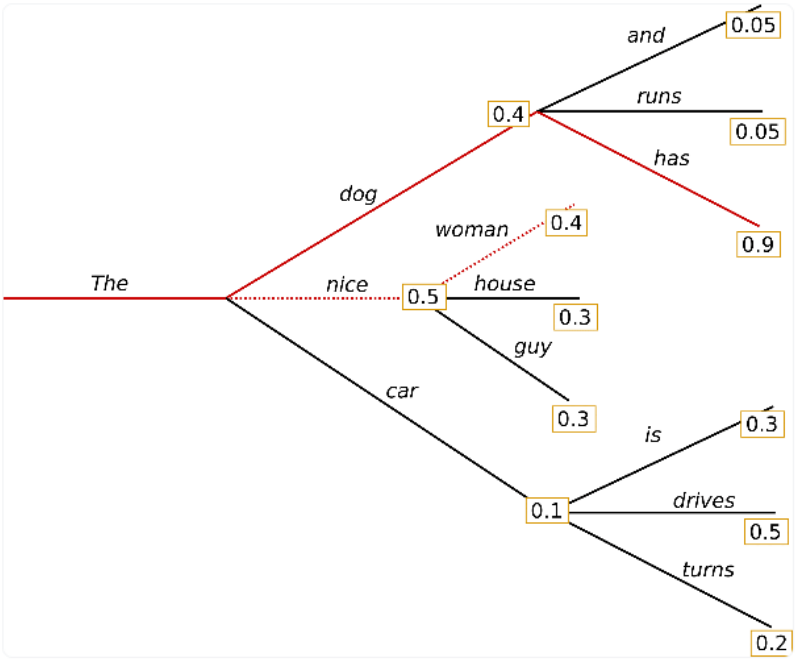
A módszerrel a legnagyobb probléma az, hogy a generált szöveg nagyon gyorsan elkezdi ismételni önmagát! Ezt a problémát a következő generálási módszerekben ki lehet javítani paraméterek megadásával. Kódban a következő módon implementáltam. Generálásához nincs szükség sok paraméterre, mindössze a létrehozott szekvencia hosszát kell megadni. [15]

```
greedy_output = model.generate(input_ids, max_length=50)
```

13. ábra Greedy search kód

4.3.2. Beam search

A beam search csökkenti annak valószínűségét, hogy minden alkalommal a legnagyobb valószínűségű szót válasszuk ki, ezáltal lehetővé teszi azt, hogy sokkal kreatívabb és egyedibb szöveget generáljunk. Több lehetséges megoldást is nyomon követ. Például, ha a num_beams=2 paraméterrel a következőképpen fog kinézni a fa:



14. ábra Beam search fa

Mint látszik is itt egyszerre 2 lehetséges megoldás van: „The dog has” melynek a valószínűsége 0,36 és a „The nice woman” melynek valószínűsége 0,2. A beam search minden esetben jobb megoldást ad mint a greedy search, de ez sem garantálja nekünk a jó megoldást minden esetben. Továbbra is találhatunk benne ismétlődéseket, erre a problémára a következő megoldást lehet beleírni a paramétere közé: no_repeat_ngram_size=2. Ez nagyon sokat segít mivel nem engedi az adott szót a megadott számnál többször használni a generálás során. Fontos azt megjegyezni, hogy óvatosan kell bánni ezzel a paraméterrel, mert okozhat gondokat a generálás során.

Nekem a megoldás során a 2 volt az a szám, amellyel a legjobb megoldásokat kaptam. Ezen felül a beam searchben lehetőségünk van több megoldást is generálni egy időben, amennyiben a num_return_sequences-t bevezetjük. Itt fontos azt megjegyezni, hogy num_return_sequences <= num_beams relációt minden esetben be kell tartani! Az implementálás során több paraméterre is szükség van. Megadjuk a szöveg hosszát, a következő részben részletezett paramétereket is lehet alkalmazni a beam searchben. [15]

```
beam_output = model.generate(
    input_ids,
    max_length=100,
    num_beams=5,
    top_p=0.6,
    temperature=0.7,
    no_repeat_ngram_size=2,
    early_stopping=True,
    num_return_sequences=5
)
```

15. ábra Beam search kód

4.3.3. Sampling

A sampling egy másik alap generáló módszer. A sampling lényege, hogy valószínűséget számolunk a következő képlettel

$$w_t \sim P(w|w_{1:t-1})$$

:

16. ábra Sampling képlet

Minden alkalommal több szót is figyelembe veszünk, abban különbözik a greedy search-től, hogy nem csak a legutolsó szót vesszük figyelembe.

17. ábra Sampling működése

Itt az első generált szó a „car”, a P(„The”) szóból generálódik, ez hasonló, mint a greedy search, de a második generált szavunk a „drives” már különbözni fog. Itt már a P („The”, „car”) valószínűségéből ered. Mint az ábrából is látszódik, nem mindig a legnagyobb valószínűséget válasszuk. Ennek a problémának a megoldása megoldható egy paraméter bevezetésével, ami a „temperature”. Ennek az értéke [0,1] között mozoghat. Megadja az a minimum valószínűséget, hogy a szót használjuk. Ha az előző példánál bevezetjük a „temperature = 0.7” -et, akkor a következőképpen változik a generált szöveg

.

18. ábra Sampling temperature paraméterrel

Ismét elkezdjük kiszámolni a „The” szó utáni valószínűséget, de mivel itt már nem lehet a „car” szót választani a valószínűsége miatt, ezért a „nice” szót válasszuk ki. Ez után pedig a „house” -t. Kódban ezekkel a paraméterekkel alkalmaztam. [15]

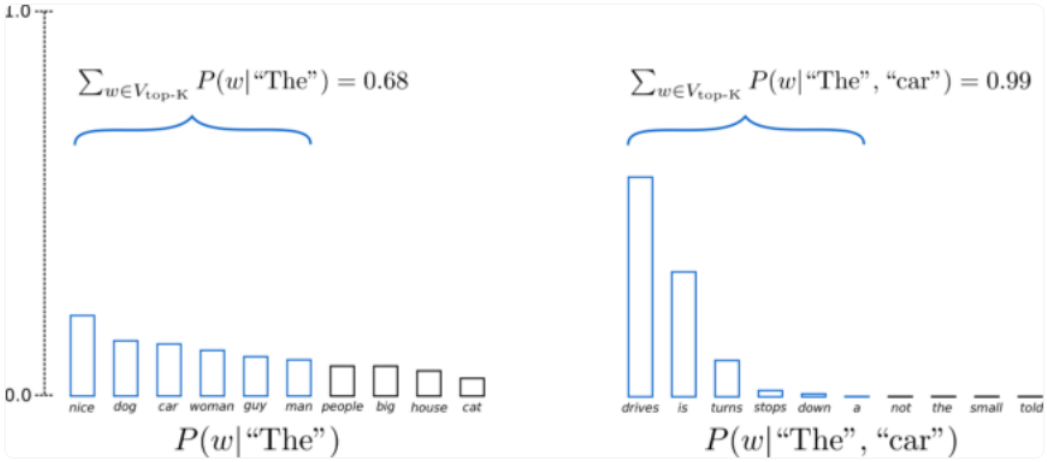
```
sample_outputs = model.generate(
    input_ids,
    do_sample=True,
    temperature=0.7,
    max_length=100,
    top_p=0.92,
    top_k=50,
    num_return_sequences=5
)
```

19. ábra Sampling kód

Paraméterek között megtalálható még a top_k és top_p, amiről nem esett szó. Mind a kettő módszer. A két paraméter két különböző módot ad arra, hogy generálásnál csak a legjobb szavakat vegyük figyelembe.

4.3.4. top_k paraméter, top_k sampling

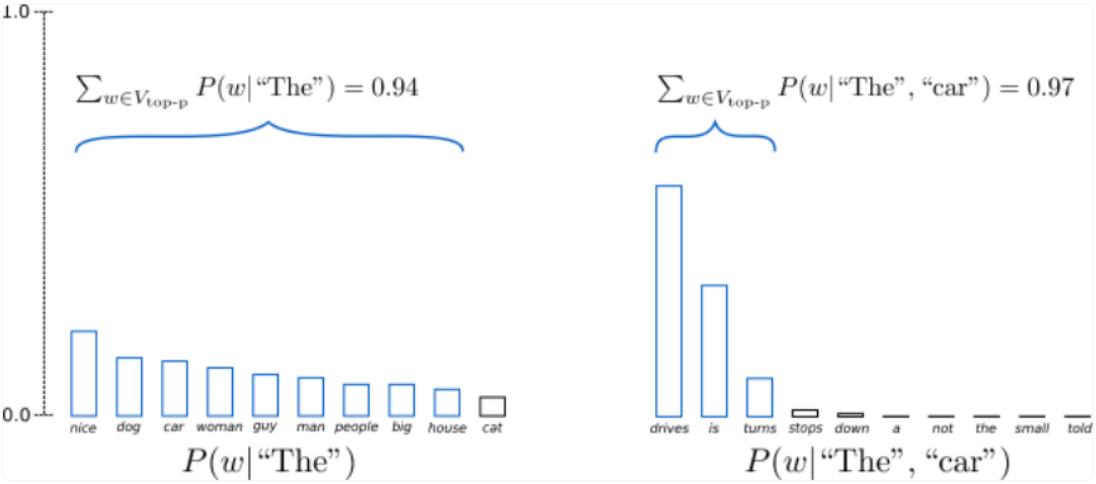
Arra használható, hogy a generálás során csak adott k darab számú szónak a valószínűségét vegyük figyelembe, ezzel kiszűrve a nagyon kis valószínűségű szavak, ezáltal a szövegkörnyezetbe nem beleillő szavak kiszűrését. Abban az esetben, ha például 6-ra állítjuk ezt a számot a következőképpen fog kinézni a szavak kiválasztása. [15]



20. ábra top_k sampling

4.3.5. top_p paraméter, top_p sampling

A top_p ugyanúgy használható a szavak ki szűrésére, mint a top_k. Annyi a különbség a kettő módszer között, hogy a top_p nem egy előre megadott fix darabszámnyi szót választ ki, hanem valószínűséget vizsgál és csak egy bizonyos össz valószínűségig veszi figyelembe a szavakat. Mint a lenti ábrán is látszódik ezzel sokkal dinamikusabbá válik a generálás, mivel nagyban befolyásolja a legvalószínűbb szavak valószínűsége a kimenetet.



21. ábra top_p sampling

Az első esetben jól látszódik, hogy a 0.94-es össz valószínűség 9 szóból adódik, de a második generálásnál, csupán 3 szó adja ki a 0.97-es valószínűséget. Természetesen a top_k és top_p módszer együttesen is alkalmazható, ami meglepően jó eredményeket tud adni. [15]

4.3.6. Beépített generate függvény

Ez az utolsó generáló módszer, amit kipróbáltam. A transzformátorok könyvtárában létezik egy beépített generálási lehetőség, amelyet egy pipeline létrehozása után tudsz alkalmazni. Igazából a már fentebb említett generálási módszereket szedi össze egy függvény alá, melyben különböző paraméterek segítségével tudod meghívni azt amelyiket szeretnéd.

```
generator = pipeline('text-generation', tokenizer=tokenizer, model=model)
print(generator("Világtörténet! mily csodálatos könyv",
                max_length=100,
                top_k=50,
                top_p=0.92,
                no_repeat_ngram_size=2,
                temperature=0.7)[0]['generated_text'])
```

22. ábra Generate függvény

Több pipeline létrehozása is lehetséges, de nekem a „text-generation-re” volt szükségem.

Generálások után pedig a console-ra kiírtattam a különböző megoldásokat, így már össze lehetett őket hasonlítani és lehet találni közöttük nagyon szép generált szöveg részleteket. Tapasztalataim alapján azt mondanám, hogy a legsikeresebb generáló módszer az én feladatomban a top_k sampling és top_p sampling generálás együttes használata volt.

5. TESZTELÉS

5.1.Beam output

Paramétereim a generálás során a következőek voltak:

```
input_ids,
max_length = 100,
num_beams = 5,
top_p=0.6,
temperature = 0.7,
no_repeat_ngram_size=2,
early_stopping=True,
num_return_sequences=5
```

A következő generált szöveget kaptam eredményül:

Világtörténet! mily csodálatos könyv!

Hadd-e, hogy nem, nem a sors, de nem az isten,

Amelyet e nagy, egy a nagy

Keblűű s egy-egy szép

Árömed s az ég alatt nem is, mint két-két

Re vagy nem legyen, és a lány vagy egy sugár is is az öröm

Égleted vagy, s ez is a szív, mely egy, amely nem

A Beam search algoritmussal létrehozott megoldás. Látszik, hogy nem a legszebb, nagyon sok benne az ismétlés, nem lehetett teljesen kiszűrni az ismétlődéseket. A szövegkörnyezetet megtalálja, hasonló szavakat használ, mint a többi megoldásban. Verses formánk pedig nagyon szépen létrejött. A szöveg értelmetlen, nem a legmegfelelőbb generálási módszer a mi estünkben.

5.2.Greedy Output

Paraméterem a generálás során mindössze egy hossz volt, többre nem volt szükség::

```
max_length=50
```

Világtörténet! mily csodálatos könyv!

A MAGYAR A MAGYAR A MAGYAR A MAGYAR

A MAGYARTASZ A MAGYAR KERT A MAGYAR...

MITT A MAGYAR A MAGYAR A MAGYAR A MAGYAR A MAGYAR A MAGYAR A MAGYAR A MAGYAR

A Greedy search által generált megoldás látható, hogy nem jó, nagyon hamar elkezdi ismételni önmagát a program. A részletezésénél erre ki is tértem, hogy ez a legnagyobb gond ezzel a generálási móddal. Verses formája sem igazán van meg. Nem alkalmas a feladathoz.

5.3.Sample output

Paramétereim a sample search során a következők voltak:

```
input_ids,
do_sample=True,
temperature = 0.7,
max_length=100,
top_p=0.92,
top_k=50,
num_return_sequences=5
```

Világtörténet! mily csodálatos könyv!
Ez a szív, az emlékezet.
Reng az isten s most a halál fénye
Túlelt jár a nap; a tavasz, a szél akkor:
„A szabadság világnak... oh magyar vagyok!
Ittha nagy, a magyar, nagy vagy nem, ha nem szeretsz és mi sem,
Sír  *Az ördög itt nem vagyok!”*
Ha ne tanúltad bömbölnek készül.
Endadjatok úgy, s nem,
Nem a kő többé

A sample search által generált kimenet, talán az összes közül a legjobb. Paraméterezése során a top_k és top_p samplig közös használatát találtam a legjobbnak a tesztelések során. Ezen felül a temperature paramétert is beállítottam a kreatív generálás miatt. A szavak kapcsolata megvan. A verse formát gyönyörűen generálta és a vers generálás során talán az egyik legfontosabb szempont a rímek is megvannak. A sample search az a módszer, amely a legalkalmasabb a feladatom megoldásához. Sajnos a modell csak eddig generálta, mert a hosszát én szabtam meg paraméterként.

5.4.Generate függvény output

Paramétereim a sample search során a következők voltak:

```
max_length=100,
top_k=50,
top_p=0.92,
no_repeat_ngram_size=2,
temperature = 0.7
```

Világtörténet! mily csodálatos könyv,

S az isten, egy kis kis szerelem.

Amily szép, nem a nagy, oh nagy a világ, az a magyar

Magig a szív, oly nagy az ég, a lány vagy egy a

Hadd legyen, hogy nem is, aki nem, ha nem lesz egy nagy van, mint a két kebel

Két, te, és nagy lesz a föld, ki én nem leszek van egy élet

Paraméterezésénél hasonló értékeket használtam, mint a sample search esetén, mivel ahogy említettem így adja a legszebb megoldásokat. A transzformátor könyvtárba beépített generate függvény eredménye egész jó. Szavak elképzelhetőek azonos szövegkörnyezetben, Verses formával itt sincs gond, De a szöveg sajnos nem teljesen értelmes.

6. EREDMÉNYEK ÉRTÉKELÉSE

Az eredmények értékelése egy nehezebb pont volt a projekt során, mivel hogyan tudnánk egy ilyen szövegnek, versnek a jóságát meghatározni. Ehhez alapvetően magamnak állítottam fel a kritériumokat, mint például: szavaknak legyen közük egymáshoz, legyen rímes, legyen a generált szövegnek verses formája. Ezeket végül nagyon sok tesztelés, futtatás és paraméterek pontosítása után sikerült elérnem. A következő megoldásokat kaptam az elmúlt időkben:

```
---------------------------------------------------------------------------------------------------------
Világtörténet! mily csodálatos könyv!
Ez a szív, az emlékezet.
Reng az isten s most a halál fénye
Túlelt jár a nap; a tavasz, a szél akkor:
„A szabadság világnak... oh magyar vagyok!
Ittha nagy, a magyar, nagy vagy nem, ha nem szeretsz és mi sem,
Sír Az ördög itt nem vagyok!”
Ha ne tanúltad bömbölnek készül.
Endadjatok úgy, s nem,
Nem a kő többé
---------------------------------------------------------------------------------------------------------

Ereszt éjenként ezüst guzsalyán,
Bolyott ott a nap arany utcáiban,
Hogy más, a kemény rokonink megkoszorúzni,
S a csendes bús kocsiján
Bennem e gondolat s ez az én,
De nem is úgy, ha ne halálbért
---------------------------------------------------------------------------------------------------------

S én rajtok hidegen, gúnyosan kacagok.
S ha a szív, egy kis szerelem
Keblűűű s a lány,
Magas s az isten remeke

---------------------------------------------------------------------------------------------------------
Én írogatni kezdtem,
Minden már a szél;
Nem vagyok, ha nem tudom, hogy volna
Az én!
Hogyha volna tudom.
S ha
Szeretem nekem, mi nem
```

```
Holon
Kebzértildon és bagóz?
--------------------------------------------------------------------
```

A tesztelés során kapott jobb eredményeket értékeljük az fentebb leírt szempontok alapján. Első sorban a futási időt jegyezném meg. Ez a hosszú közel 3 órás futási idő elfogadható ezekkel a paraméterekkel, kezdetben ennél sokkal nagyobb volt az időigénye a programnak. Paraméterek pontosításával és a modell videókártyán való tanításával ezt csökkenteni lehetett. Ennél tovább csökkenteni nem sikerült a számítógépem határai miatt. Fontos volt számomra, hogy ne a gyorsaság legyen az elő számú cél, sokkal inkább a minőség, erre az egész munkám során nagy figyelmet szenteltem. Természetesen addig a szintig amíg nem rontott az eredményen folyamatosan javítottam.

Az első kritériumom, amely a szavak szövegkörnyezetbe való illeszkedése, minden versnél minden alkalommal tökéletesen megfelel. Ahogy olvassuk a verseket látható, hogy az egymáshoz közel lévő szavak hasonlóak. Például: könyv-emlékezet, nap-tavasz-szél, isten-ördög képek az első versben. Az arany-ezüst kapcsolat a másodikban. Hideg-gúnyos, szerelem-szív-lány is megtalálható bennük, ezért úgy gondolom, hogy ennek a kritériumnak megfelel a generálás.

A sor elválasztások, azaz verses forma megléte látható, hogy megvan. Tördelve vannak a sorok, Vannak mondat végi írásjelek, vesszők. Talán rímek terén nem tökéletes a modell. Látható, hogy nem tartja be a rímek formáit, ennek ellenére sok rím megtalálható benne és a dallamossága is megmarad a verseknek. Nem monotonok lettek a generált szövegek. Másik nagyobb hiányossága, az, hogy néhány esetben értelmetlen szavakat generál. A fentebb olvasható versekben is található ilyen, például Kebzértildon szó. Ilyen nem volt a bemeneti adatok között. Minden bizonnyal a tokenizálásnál talált egy ilyen szókapcsolatot, ami lehetséges, hogy a normalizálás miatt történt.

7. ÖSSZEFOGLALÁS

Összességében ez a versgenerálás magyar nyelven nagyon izgalmas volt. Nagyon sok lehetőség van benne, megmutatta azt, hogy a mi nehéz nyelvünket is tudja kezelni egy jól felépített neurális hálózat. Úgy érzem és egyben remélem is, hogy a jövőben még nagyon sok hasonló megoldás fog még napvilágot látni és talán az én szakdolgozatomat is fogják alapjául venni! Nagyon szívesen olvasnék még továbbfejlesztett változatait versgeneráló algoritmusoknak.

Tovább fejlesztési lehetőségnek mindenképpen megszeretném említeni 2022. eleji konferencián bemutatott ELTE-verskorpusz-t[5]. Nagyon sok lehetőséget foglal magában, mivel az elmúlt pár évszázad költőinek verseit tartalmazza, tokenizálva, ritmikálva. Ezzel sokat lehet még fejleszteni a ritmikusságon és a rímeken, azon, hogy még versesebb és ember közelibb legyen a generált vers. Ezen kívül lehetőségnek látom a modell GPT-3 és hamarosan GPT-4-re való konvertálását, amely még tovább tudná javítani a generált versek minőségét. Továbbá lehetne implementálni és tanítani, sokkal nagyobb adatbázisokon, például több költőnek verseit is magába foglalhatná. Úgy érzem, ez a feladat még csak a jéghegy csúcsa és még többet lehet kihozni belőle.

Felhasználást azt talán elsőre nehéz megfogalmazni. Én igazából ezt az algoritmust, amikor verset generálunk én a szórakoztató iparban tudnám elképzelni, azon belül is mesterséges digitális világban, mint például a Metaverzum. Az ott található karaktereket sokkal ember közelibbé és dinamikusabbá lehet tenni. Mégis a valódi haszna igazából elméleti, az, hogy nincsenek határok a neurális hálózatok fejlődésében NLP terén. Létre lehet hozni olyan robotokat, algoritmusokat, amivel kommunikálni lehet és én úgy érzem a jövő ebbe az irányba megy. Remélem még az én életemben fogok látni jelentősebb eredményeket.

[5] GitHub - ELTE-DH/poetry-corpus: Corpus of Hungarian poems in TEI XML with machine annotation

8. SUMMARY

Overall, poem generation and NLP in Hungarian language is very exciting. There is a lot of opportunity in it, and it shows that a good neural network can handle our difficult language too. I think, and I hope, that in the future a lot of similar results will appears and maybe this thesis will be the base of some of them. I would like to read improved poem generation algorithms in the future.

I would like to mention one possible improvement. In the beginning of 2022 was a conference where the Elte-poem corpus[6] has been shown. There is a lot of opportunity in that It contains the last some century's poets all poems, tokenized and rhythmically. With this corpus, it is possible to improve rhymes, and you can generate more melodic poems. Addition, another opportunity is to convert the model to GPT-3 and after the release to GPT-4, it will improve the quality of poem generation. Furthermore, it would be great to train the model on a bigger database, for example with more poems of poets. I think this project is a tip of the iceberg and there is a lot of possibility to generate better texts.

It is really hard to explain use cases. I think this algorithm could be using entertainment industry specifically in virtual worlds like Metaverse. Characters in this world could be more real and more dynamic. Though the real benefit is that there is no limits in NLP and in text generation. It is possible to create robots, algorithms, that can communicate and I think this will be the future. I hope I will see major results in my life.

[6] GitHub - ELTE-DH/poetry-corpus: Corpus of Hungarian poems in TEI XML with machine annotation

9. IRODALOMJEGYZÉK

[1] A. Mittal, „Understanding RNN and LSTM,” 12 október 2019. [Online]. Available: https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e.

[2] S. Hochreite és J. Schmidhuber, „Long Short-Term Memory,” 1997..

[3] T. Mikolov, „Distributed Representations of Words and Phrases,” 2013.

[4] M. Chablani, „towards data science,” 14 június 2017. [Online]. Available: https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b.

[5] T. M.-I. S.-K. C.-G. C.-J. Dean, „Distributed Representations of Words and Phrases,” 2013.

[6] T. K.-A. B. M. d. Rijke, „Siamese CBOW: Optimizing Word Embeddings for Sentence Representations,” Berlin, 2016..

[7] G. Giacaglia, „How Transformers Work,” 11 március 2019.. [Online]. Available: https://towardsdatascience.com/transformers-141e32e69591.

[8] „Why does the transformer do better than RNN and LSTM in long-range context dependencies?,” 4. április 2020. [Online]. Available: https://ai.stackexchange.com/questions/20075/why-does-the-transformer-do-better-than-rnn-and-lstm-in-long-range-context-depen.

[9] „Hugging Face,” 3. május 2022.. [Online]. Available: https://huggingface.co/docs/transformers/main/tokenizer_summary#byte-pair-encoding.

[10] P. Mooney, „Kaggle,” 22. május 2020. [Online]. Available: https://www.kaggle.com/paultimothymooney/poetry-generator-rnn-markov.

[11] Z. W.-W. H.-H. W.-. H. W.-W. L.-H. W.-E. Chen, „https://arxiv.org/pdf/1610.09889.pdf,” Peking, 2016.

[12] D. Krivitski, „Generation of poems with a recurrent neural network,” 25 május 2018. [Online]. Available: https://medium.com/@DenisKrivitski/generation-of-poems-with-a-recurrent-neural-network-128a5f62b483.

[13] J. Brownlee, „Machine Learning Mastery,” 4. augusztus 2016. [Online]. Available: https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/.

[14] S. Maitra, „towards data science,” 13 október 2019. [Online]. Available: https://towardsdatascience.com/recurrent-neural-network-to-predict-multivariate-commodity-prices-8a8202afd853.

[15] „Hugging Face,” 3. május 2022.. [Online]. Available: https://huggingface.co/blog/how-to-generate.