Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

ВИКОРИСТАННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ СТВОРЕННЯ ЗОБРАЖЕНЬ ГУМОРИСТИЧНОГО ХАРАКТЕРУ

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
доцент, к.ф.-м.н. О. П. Жежерун
(прізвище та ініціали)
_____
(підпис)
“____” _____ 2020 р.

Виконала студентка_____
_____Карлишева А. О.____
(прізвище та ініціали)
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем,

доцент, к.ф.-м.н. О. П. Жежерун

(підпис)

„____”_____2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту__Карлишевій А.О.__ факультету _інформатики_ _____3-го____курсу

Розробити систему нейронних мереж для генерації гумористичних зображень

Вихідні дані:

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

Теоретичні відомості

Розробка системи

Висновки

Список джерел

Дата видачі „___” _____ 2019 р.

Керівник _____ (підпис)

Завдання отримав _____

Календарний план виконання роботи

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання теми курсової роботи		
2.	Пошук та збір тематичних матеріалів		
3.	Ознайомлення з відповідними матеріалами та обдумування структури роботи		
4.	Написання вступу та плану роботи		
5.	Вивчення відповідних матеріалів, що необхідні дня написання розділів роботи		
6.	Вивчення відповідних матеріалів необхідних для створення нейронної мережі		
7.	Безпосередня розробка нейронних мереж		
8.	Збір статистики та аналіз результатів		
8.	Написання розділів курсової роботи		
9.	Коректне оформлення роботи відповідно до вимог написання курсової роботи		
10.	Подання та аналіз попередньої версії роботи з керівником		
11.	Корегування роботи згідно із зауваженнями керівника.		
12.	Захист курсової роботи		

Студентка Карлишева А. О.

Керівник Жежерун

«_____» _____

Зміст

Анотація

Курсова робота присвячена дослідженню способів та результатів використання нейронних мереж для генерації зображень гумористичного характеру.

У першому розділі було досліджено існуючі системи нейронних мереж та проведено аналіз їх роботи.

Другий розділ присвячено створенню системи нейронних мереж для генерації необхідних зображень, тексту, їх комбінації та дослідженню якості результатів.

Ключові слова: **нейронні мережі, рекурентні нейронні мережі, генеративні змагальні мережі, меми**.

Вступ

У сучасному світі технології та культура переплетені дуже тісно. Мільярди людей використовують інтернет щодня, а кількість надісланих повідомлень важко уявити. Обсяг інформації що сприймається за день пересічним користувачем невпинно зростає. У вік «швидкої» інформації, гумор також прискорюється. Якщо раніше популярним видом гумору були, наприклад, анекдоти, що передавалися від людини до людини, з емоціями, інтонаціями та витраченними зусиллями та часом, то в інтернеті гумор сприймається, створюється і передається в картинках. На засвоєння більшості з них вистачає кількох секунд.

Подібні гумористичні картинки називаються «меми». Поєднання зображення та тексту дозволяє точно передати зміст жарту, а варіативність відтінків змісту в залежності від складових вражає. Вони об’єднуються в групи за зображенням, текстом, та контекстом.

За свій життєвий цикл меми можуть отримувати нові відтінки значень та способи використання, зливатися в одне ціле з іншими мемами які можуть не мати жодного стосунку до оригінального жарту, помирати та відроджуватись, доки врешті-решт не набудуть зовсім нового змісту. Деякі з них переходять на інші рівні абстракції - від безпосереднього значення, до іронічного, та навіть пост-іронічного. Інші з них можуть навіть стати абсурдними, та втратити можливість бути зрозумілими без контексту.

Подібні абсурдні меми можуть виглядати так, ніби вони були створені нейронною мережею, що не розуміє зміст і лише відтворює подібне. Але, хоча технології та нейронні мережі активно розвиваються, для створення розважального контенту, зокрема, мемів вони використовуються дуже і дуже нечасто.

У цій роботі буде досліджено можливість створення абсурдного розважального контенту та реакцію споживачів медіа на нього.

Основні скорочення

RNN -- Recurrent Neural Networks

LSTM -- Long short-term memory

GPU -- graphics processing unit

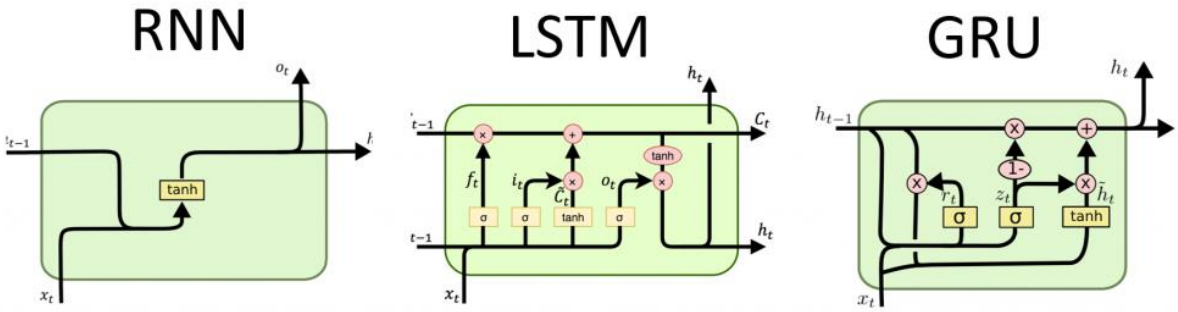
CPU -- central processing unit

GAN -- Generative adversarial network

1. Теоретичні відомості

1.1. Recurrent Neural Networks

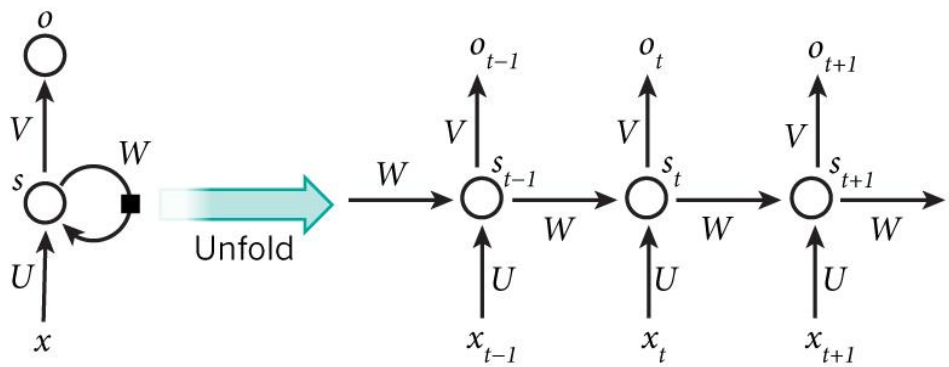
Recurrent Neural Networks, RNN, Рекурентні Нейронні Мережі - клас нейронних мереж, де зв’язки між елементами створюють направлену послідовність. Вони використовуються в основному для генерації послідовних даних, таких як натуральна мова.



(Рисунок 1. Приклади архітектур рекурентних нейронних мереж.)

Рекурентні нейронні мережі називають рекурентними, бо вони виконують одну й ту саму задачу для кожного елементу послідовності, до того ж, результат поточної задачі залежатиме від попередніх обчислень. Теоретично, рекурентні нейронні мережі можуть використовувати інформація послідовностей будь-якої довжини. На практиці ж, зазвичай використовуються вихідні дані лише останніх кількох обчислень.

Рекурентна нейронна мережа розгортається у кілька шарів, причому зазвичай розгортка залежить від кількості слів у послідовності. Тобто при отриманні на вхід речення з n слів використовується n шарів.



(Рисунок 2. Схематична діаграма типової рекурентної нейронної мережі)

На рисунку представленому вище зображено схему роботи та її розгортку рекурентно нейронні мережі, де:

x = (x_1, x_2, …, x_t, …, x_n) -- вхідні дані, що представляють собою послідовність яка може бути опрацьована рекурентно. У даному випадку xt - векторне представлення слова.

t -- поточний крок

s -- При обробці кожного символа, нейронна мережа зберігає прихований внутрішній стан

U -- вхідна вагова матриця

V -- вихідна вагова матриця

W -- вагова рекурсивна матриця, яка використовується у якості параметрів методу, по суті є вихідною матрицею попереднього шару

Операція на кожному кроці рекурентної нейронної мережі може бути визначена як

$$h_t = \sigma(Ux_t + Ws_t - 1)$$

де σ - початкова функція, h_t - вихідні дані

Результат роботи рекурентної нейронної мережі виглядає як:

$$o_t = \sigma\,(Vs_t)$$

Дослідження подібних нейронних мереж також виявили, що рекурентні нейронні мережі показують кращий результат ніж «традиційні» нейронні мережі, наприклад, нейронні мережі відомі як MLP (multilayer perception neural networks)[2]. При цьому, практичне застосування рекурентних нейронних мереж має і власні виклики.

1.1.1. Long short-term memory

Long short-term memory, LSTM, довга короткочасна пам’ять - одна з можливих архітектур рекурентних нейронних мереж, що була створена у 1997 році. Довга короткочасна пам’ять була вперше створена для вирішення проблеми довготривалих залежностей.

Довга короткочасна пам’ять відрізняється від традиційних рекурентних мереж тим, що не переписує збережені значення на кожному кроці.

Ефективність подібних нейронних мереж заключається в збільшенні контролю над результатами і відповідно в збільшенні контролю над помилками та їхнім уникненням, а також за рахунок комірок пам’яті дозволяє вираховувати довготривалі віддалені залежності, наприклад зв’язок між першим та останнім словом у реченні

Нейронна мережа довгої короткочасної пам’яті має вузли довгої короткочасної пам’яті на додачу або замість інших вузлів. Вузли довгої короткочасної пам’яті пам’ятає значення довгих та коротких часових проміжків. Часто вузли довгої короткочасної пам’яті використовуються у блоках з кількох поєднаних вузлів.

Вузол довгої короткочасної пам’яті має вхідний та вихідний вентилі, забувальний вентиль та комірку пам’яті.

Вхідний вентиль i_t визначає важливість вхідних даних x_t та чи необхідно змінити стан комірки пам’яті:

$$i_t = \sigma\left(U_i x_t + W_i s_{t-1} + b_i\right)$$

Забувальний вентиль f_t визначає, чи пам’ятати чи забути попередній стан:

$$f_t = \sigma\left(U_f x_t + W_f s_{t-1} + b_f\right)$$

Тангенсна функція τ_t грає роль функції активації:

$$\tau_t = tan\left(U_t x_t + W_\tau s_{t-1} + b_\tau\right)$$

Значення комірки пам’яті отримується за формулою:

$$c_t = i_t \circ \tau_t + f_t \circ c_{t-1}$$

Вихідний фільтр o_t визначає, яку частину комірки пам’яті буде передано наступному вузлу:

$$o_t = \sigma\ (U_o x_t + \ W_o s_t + b_o)$$

Стан s_t:

$$s_t = o_t \circ \tan(c_t)$$

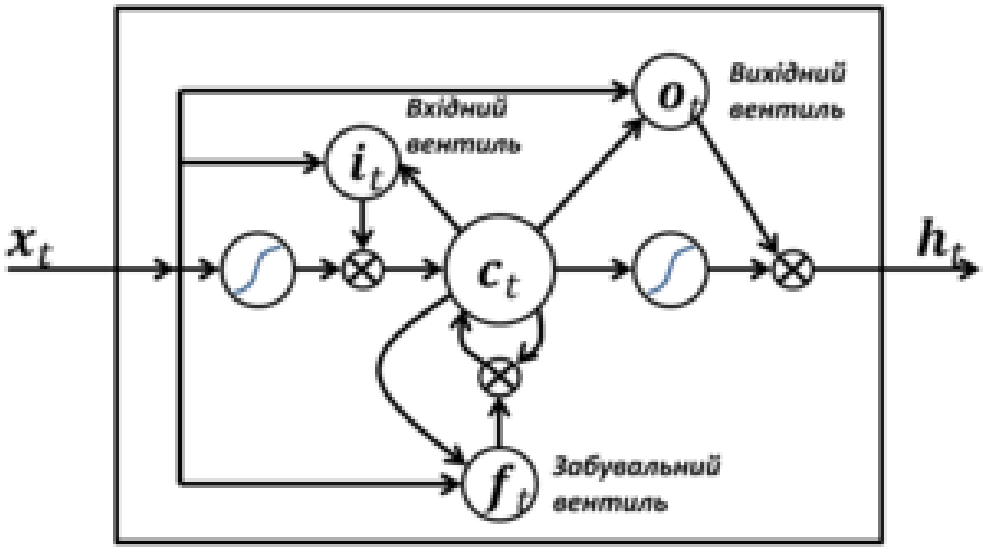
° - добуток Адамара;

σ – логістична функція активації вентилів $\sigma(x) = (1 + \ e^{-x})^{-1}$

b – параметр упередження;

W, U - матриці ваг

Нульові s_0 та c_0 ініціалізуються перед проходженням першого вузла.

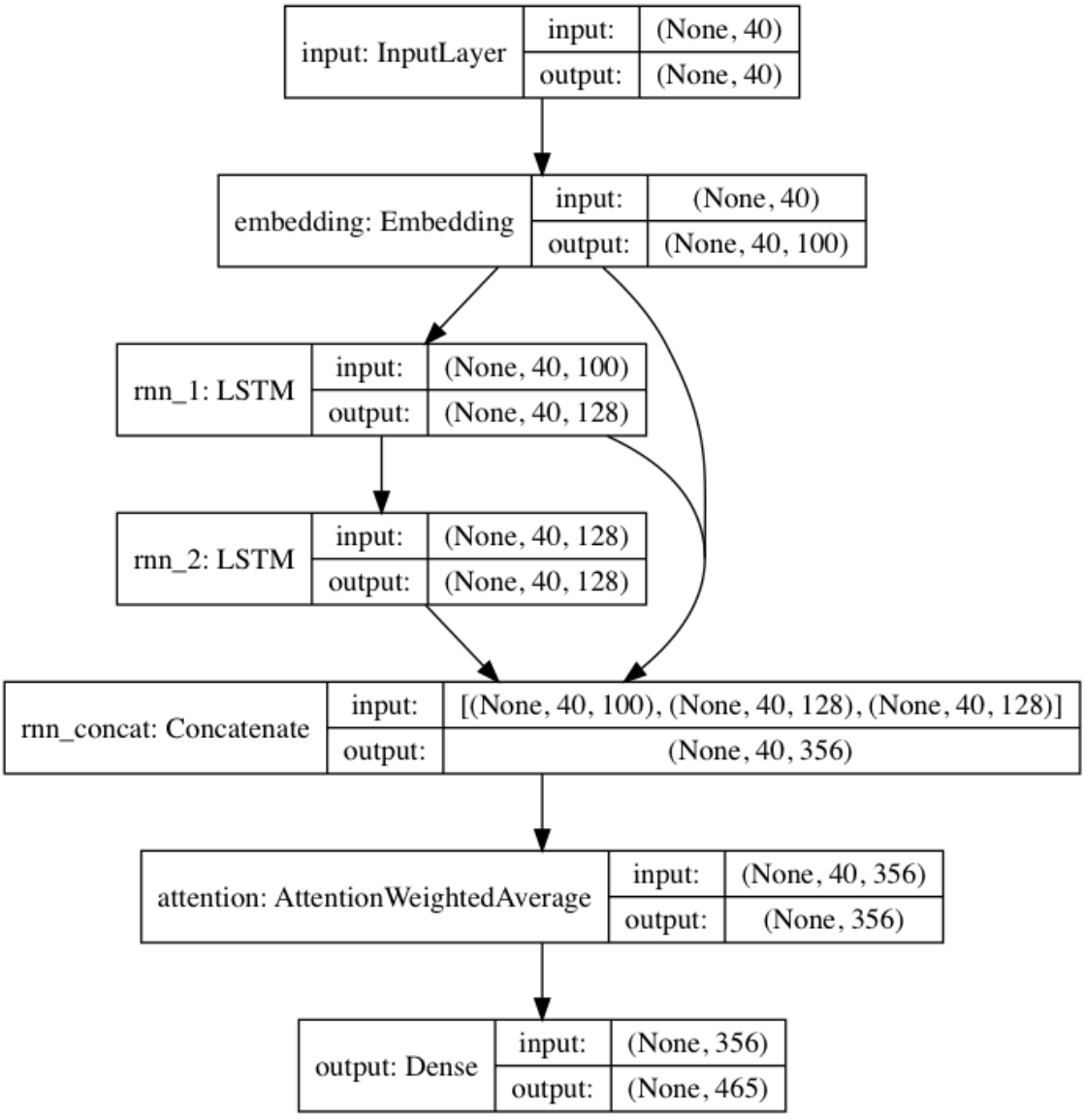


(Рисунок 3. Схема роботи довгої короткочасної пам’яті.)

1.1.2. TextGenRNN

textgenrnn -- python-модуль на основі Keras та Tensorflow для генерація текстових стрічок.

Для моделі за замовчанням, модуль використовує вхідні дані у вигляді стрічки довжиною до 40 символів. Перший шар нейронної мережі переводить кожний символ у 100-вимірний вектор і передає ці дані до рекурентного шару довгої короткочасної пам’яті. Вихідні дані цього шару далі передаються до ще одного рекурентного шару і проходять повторну обробку. Усі три шари далі передаються до attention шару для визначення і надання ваги для найважливіших тимчасових показників та усереднити їх.



(Рисунок 4. Схема роботи textgenrnn [8])

Вихідні дані виглядають як пари з символу та ймовірностей того, що даний символ йде наступним після інших символів (до 394), включаючи літери верхнього та нижнього регістрів, пунктуацію та навіть емодзі.

Також textgenrnn використовує бібліотеку NVIDIA CUDA Deep Neural Network library, що дозволяє тренувати нейронну мережу на GPU. Це дозволяє значно зменшити час необхідний для тренування. При цьому, якщо відстуні необхідні драйвера чи відеокарта не відповідає необхідним вимогам, нейронна мережа тренується на CPU.

Генерація результатів на вже натренованій нейронній мережі завжди відбувається з використанням CPU.

1.2. Generative adversarial networks

Generative adversarial networks, GAN, генеративні змагальні мережі - клас алгоритмів нейронних мереж, що використовуються для створення фотореалістичних зображень та які реалізовані за допомогою систем з двух нейронних мереж, які змагаються між собою.

Одна з двох мереж у системі грає роль генератора, а інша грає роль дискримінатора. Відповідно, дискримінаційна мережа розрізняє реальні дані від даних згенерованих генератором, а метою тренування є збільшення помилок дискримінатора. При цьому, дискримінатор продовжує тренуватися разом з генератором, тому чим кращі зображення створює генеративна нейронна мережа, тим більша точність дискримінативної мережі.

Принцип роботи подібних нейронних мереж оснований на мінімаксі - правилі прийняття рішень з теоріх ігор. Суть мінімаксу полягає в мінізмізації можливих втрат у випадку найгіршого сценарію.

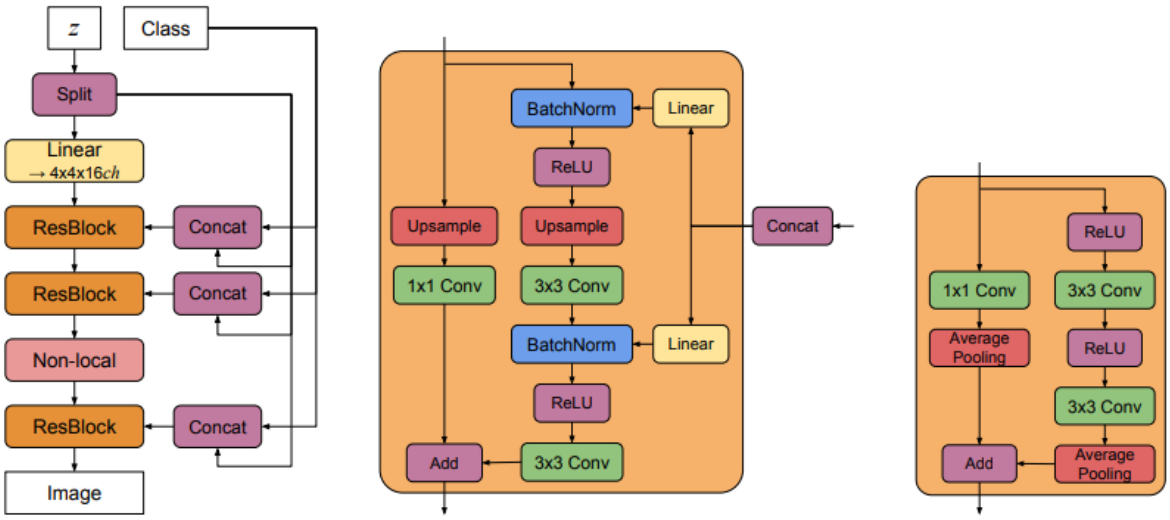
Дискримінатор (D) та генератор (G) грають у гру мінімаксу за формулою

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

(Рисунок 5. Формула мінімаксу для генеративної змагальної мережі [3])

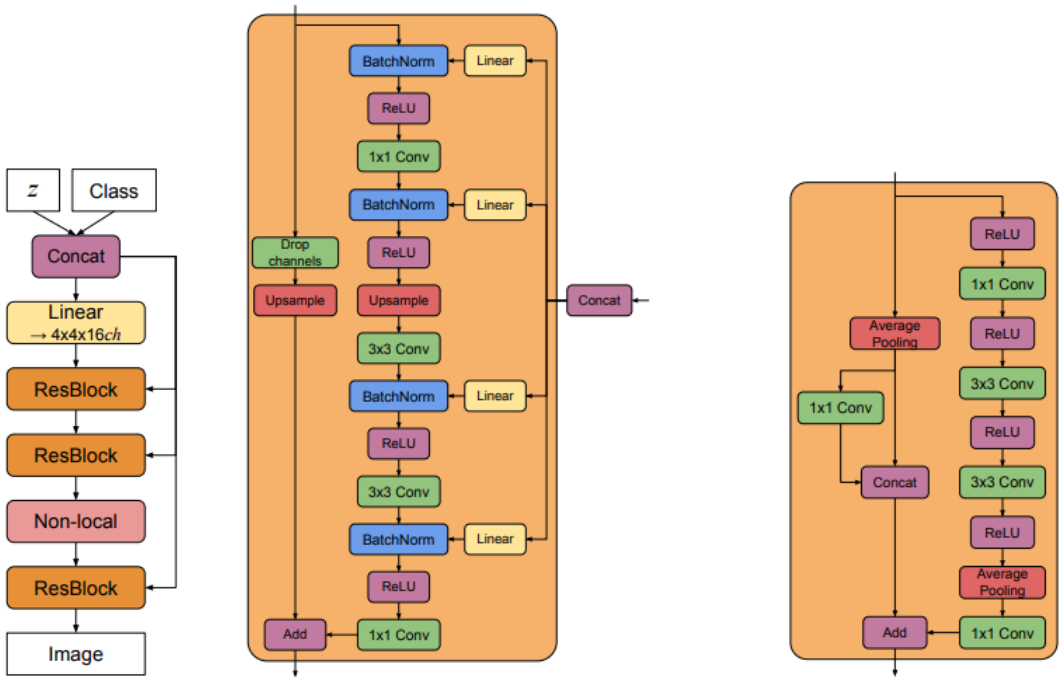
У даній науковій роботі було використано BigGAN-deep розроблену та натреновану Tensorflow. BigGAN відзначається тим, що різниця між точністю та різноманіттям зменшується, тобто більша точність не означає менше різноманіття, і навпаки, більше різноманіття не означає меншу точність.

BigGAN-deep відрізняється від BigGAN глибиною та конфігурацією блоків.



(Рисунок 6. Схема архітектури BigGAN [6])

Основою BigGAN моделі є архтектура ResNet GAN з деякими змінами.



(Рисунок 7. Схема роботи BigGAN-deep)

2. Розробка системи

2.1. Генерація зображень

Для генерації зображень було використано уже натреновану нейронну мережу BigGAN від TensorFlow.

TensorFlow -- це опен-сорс платформа розроблена Google для машинного навчання, що надає широкий вибір інструментів, бібліотек та ресурсів.

У якості датасету використана збірка зображень від image-net.org у категорії 269 grey wolf. Даний датасет містить у собі близько тисячі зображень сірих вовків.

У процесі написання коду було отримано значну кількість помилок навіть на найпростіших етапах імпортування необхідних модулів. Виявилось, що останні версії Tensorflow хоча і підтримують необхідний функціонал, але через внутрішні зміни цей функціонал фактично неможливо використати. В основному, проблеми виникали через зміни у внутрішній архітектурі модуля, підтримку яких інші модулі ще не додали. Необхідні для роботи інших модулів класи виявлялися збережні за новою адресою всередині модуля, через що їх використання було неможливе та довелося підібрати оптимальні версії модулів вручну.

Процес генерації зображень можна розділити на кілька етапів. Перший обов’язковий етап це імпорт та сетап необхідних модулів.

```
1  import tensorflow.compat.v1 as tf
2  import io
3  import IPython.display
4  import numpy as np
5  import PIL.Image
6  from scipy.stats import truncnorm
7  import tensorflow_hub as hub
8
9  tf.disable_v2_behavior()
```

(Рисунок 8. Імпорт необхідних модулів)

Наступний етапом є завантаження готового BigGAN або BigGAN Deep модулю з TensorFlow Hub. Тут же вказується і розмірність згенерованих зображень.Серед доступних розмірів: 128x128 пікселів, 256x256 пікселів та 512x512 пікселів.

```
module_path = 'https://tfhub.dev/deepmind/biggan-deep-512/1'  # 512x512 BigGAN-deep
tf.reset_default_graph()
print('Loading BigGAN module from:', module_path)
module = hub.Module(module_path)
inputs = {k: tf.placeholder(v.dtype, v.get_shape().as_list(), k)
          for k, v in module.get_input_info_dict().items()}
output = module(inputs)

print()
print('Inputs:\n', '\n'.join(
    '  {}: {}'.format(*kv) for kv in inputs.items()))
print()
print('Output:', output)
```

(Рисунок 9. Завантаження моделі)

Було ініціалізовано також кілька функцій необхідних для генерації зображень, встановлені необхідні значення кількості згенерованих зображень, шуму та усічення.

```
initializer = tf.global_variables_initializer()
sess = tf.Session()
sess.run(initializer)

num_samples = 20
truncation = 0.4
noise_seed = 0

category = 269 # timber wolf, grey wolf, gray wolf, Canis lupus

z = truncated_z_sample(num_samples, truncation, noise_seed)
y = category

ims = sample(sess, z, y, truncation=truncation)
imshow(imgrid(ims, cols=min(num_samples, 5)))
```

(Рисунок 10. Код необхідний для ініціалізація.)

В залежності від налаштованих параметрів були отримані як досить реалістичні зображення вовків, так і не дуже реалістичні чи навіть зовсім некоректні зображення.

Значення шуму відповідало за реалістичність зображень, а значення усічення відповідало за різноманітність.

Було згенеровано набір зображень з різними параметрами. Загалом було згенеровано близько сотні зображень, з яких було відкладено для використання близько десяти.

(Рисунок 11. Приклад нереалістичного згенерованого зображення.)

(Рисунок 12. Приклад реалістичного згенерованого зображення.)

(Рисунок 13. Приклад частково реалістичного згенерованого зображення.)

2.2. Генерація тексту

2.2.1. Збір та препроцессинг датасету

Досить значна частина роботи полягала у зборі текстового датасету та його корекції.

Оскільки не було знайдено жодного існуючого датасету ні російською, ні українською, що відповідав би критеріям, датасет збирався вручну зі значної кількості інтернет-джерел.

- випадкові україномовні джерела в інтернеті, що зазвичай містили лише кілька десятків цитат за раз

- випадкові російськомовні джерела, що зазвичай містили сотні цитат

В залежності від розміру джерела збирався або вручну, або автоматичними парсерами на основі збережених HTML-файлів джерел.

Оскільки російськомовний контент було знайти набагато легше, основу масу датасету становить саме він - у перекладі українською.

Отже, наступним етапом після збору тексту, був його переклад. Для зменшення обсягу витраченого часу, було використано Google-перекладач. Через ліміт у п’ять тисяч символів за переклад, було витрачено близько години тільки на переклад тексту.

Наступним етапом став етап корекції та фільтрації. Кожна вхідна стрічка була перевірена та скорегована вручну, частина була видалена через некоректність висловлювання або втрату змісту у процесі перекладу. Орієнтовно, на цей етап було витрачено близько п’ятнадцяти годин.

Загалом було зібрано 4036 (чотири тисячі тридцять шість) рядків тексту, при рекомендованих для використаної бібліотеки 2000-5000 рядків.

2.2.2. Генерація тексту

Для генерації тексту було використано уже існуючу Python-біблотеку TextGenRNN на основі Tensorflow.

Зібраний вручну датасет було розміщено в текстовому файлі, причому зазвичай кожне речення було розміщенно в новому рядку, оскільки textgenrnn опрацьовує датасет саме окремими рядками. Виключенням стали випадки, коли речення були скомбіновані таким чином, що їхнє розділення на окремі рядки спричиняло втрату будь-якого сенсу.

Генерація відбувалася у кілька етапів, кожний з яких містив у собі кілька епох нейронної мережі, зазвичай 5. Це відбувалося для того, щоб мати можливість зберігати проміжний результат на випадок перетренування нейронної мережі.

Проміжні результати зберігалися у файли з розширенням .hdf5 та містили у собі вагові матриці.

Загалом для досягнення оптимального результату тренування нейронної мережі відбулося впродовж 17 епох. Завдяки особливостям використаної бібліотеки, було можливість бачити проміжні результати навчання після завершення кожної епохи.

Результати генерації тексту після перших кількох епох виглядали як набори випадкових символів.

Після приблизно десяти епох, набори символів почали за виглядом нагадувати українську мову - можна було побачити короткі існуючі слова, на кшталт «не», «та», «або». Довші слова нагадували мову лише за звучанням та типовим поєднанням звуків.

Після п’ятнадцятої епохи результат генерації нагадував мову ще більше. Зустрічалися окремі слова та граматично правильні поєднання слів. Однак, повні речення все ще виглядали як набори слів та не містили жодного сенсу.

Після сімнадцятої епохи нейронна мережа почала генерувати речення подібні до натуральної мови, за вийнятком досить частих граматичних

помилок. На цьому тренування нейронної мережі було припинено, через те, що результат обрахунку внутрішньої функції втрат перестав зменшуватися, зупинившись приблизно на 1.4, а потім навіть почав збільшуватися, а отже, були досягнуті найкращі можливі результати на зібраному датасеті.

```
from textgenrnn import textgenrnn
textgen = textgenrnn()

textgen.train_from_file('data/citatyukr.txt', num_epochs=5)
textgen.save('trained_memegen.hdf5')
textgen.generate(5)
```

(Рисунок 14. Код для створення та тренування нейронної мережі)

Механізм створення об’єкту textgenrnn зі збереженого файлу незначним чином відрізняється від створення об’єкту з порожньою нейронною мережею. Необхідно також додатково вказати файл конфігурації та словника, а при тренуванні вказати чи тренується нова модель чи ні.

За умовчанням використовується температура значенням 0,5.

```
from textgenrnn import textgenrnn
textgen = textgenrnn(
    config_path="./textgenrnn_config.json",
    weights_path="trained/trained_memegen.hdf5",
    vocab_path="./textgenrnn_vocab.json")
textgen.train_from_file('data/citatyukr.txt',new_model=False,num_epochs=3)
textgen.save("trained/trained_memegen_2.hdf5")
textgen.generate(10)
```

(Рисунок 15. Код для продовження тренування моделі)

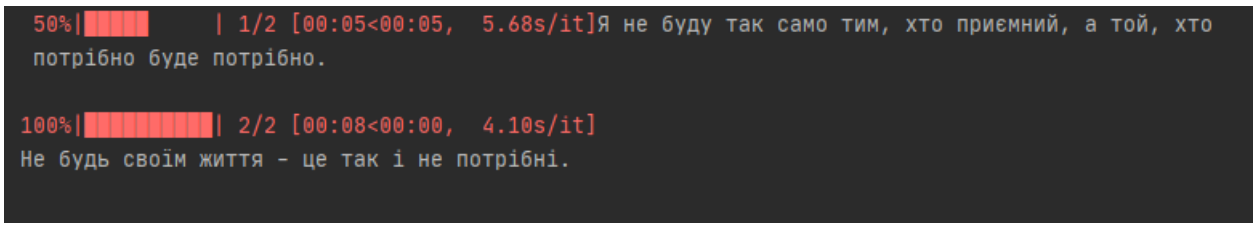
На рисунку нижче представлено приклад роботи нейронної мережі з заданою температурою 1 та 0,1. Як можна побачити, з високою температурою було згенеровано неіснуючі слова, що свідчить про високу ймовірність помилок, тоді як з низькою температурою результат виглядає набагато краще.

```
 50%|█████     | 1/2 [00:05<00:05,  5.91s/it]По-пвід, які приксе: На кайду різність, ініше ,
 обрагу її правди з особований так, як тріїм!

Подарую дуже бідсяний, як запрошла в будь-який бачу чекати буде мене.

100%|██████████| 2/2 [00:09<00:00,  4.80s/it]
```

(Рисунок 16. Приклад генерації тексту з температурою 1.)



(Рисунок 17. Приклад генерації тексту з температурою 0,1)

2.3. Поєднання зображення та тексту

На цьому етапі відбувався контроль якості результату роботи нейронних мереж.

Спочатку створювався об’єкт уже натренованої нейронної мережі і генерувалася певна кількість текстів із заданою температурою та максимальною довжиною стрічки. Температура відповідає за «креативність» згенерованих результатів, де 0 означає повну відстутність «креативності», найменшу кількість помилок та максимальну наближеність згенерованого результату до оригінального датасету, а 1 означає найбільшу «креативність» та найбільшу кількість помилок.

Також було встановлене обмеження у вигляді максимальної кількості символів, для покращення розміщення тексту на зображенні.

```
from PIL import Image, ImageFont, ImageDraw
from textgenrnn import textgenrnn
import textwrap

textgen = textgenrnn(config_path="./textgenrnn_config.json",
                     weights_path="trained/trained_memegen.hdf5"
                     , vocab_path="./textgenrnn_vocab.json"

                     )
n = 10
generated = textgen.generate(n, return_as_list=True, temperature=0.35, max_gen_length=80)
```

(Рисунок 18. Код необхідний для генерації тексту.)

Наступним етапом відбираються картинки згенеровані другою нейронною мережею, де основним показником для відбору є суб’єктивне сприйняття зображення та його оригінальність.

Подальшим кроком є накладення згенерованого тексту на одне й те саме зображення за допомогою python-модулю PIL (Pillow). Були підібрані оптимальний шрифт, колір та колір контуру для кращої читабельності тексту.

```
for i in range(n):
    title_font = ImageFont.truetype('data/lobster.ttf', 25)
    title_text = textwrap.wrap(generated[i], width=43)
    text = ""
    for line in title_text:
        text = text + line + "\n"
    my_image = Image.open("images/best/5_02.png").convert('RGB')
    image_editable = ImageDraw.Draw(my_image)
    fill_color = (255, 255, 255)
    stroke_color = (0, 0, 0)
    image_editable.multiline_text((15,435), text, fill=fill_color, stroke_width=3, stroke_fill=stroke_color, font=title_font)
    my_image.save("res/result"+str(i)+".jpg")
```

(Рисунок 19. Накладення тексту на зображення.)

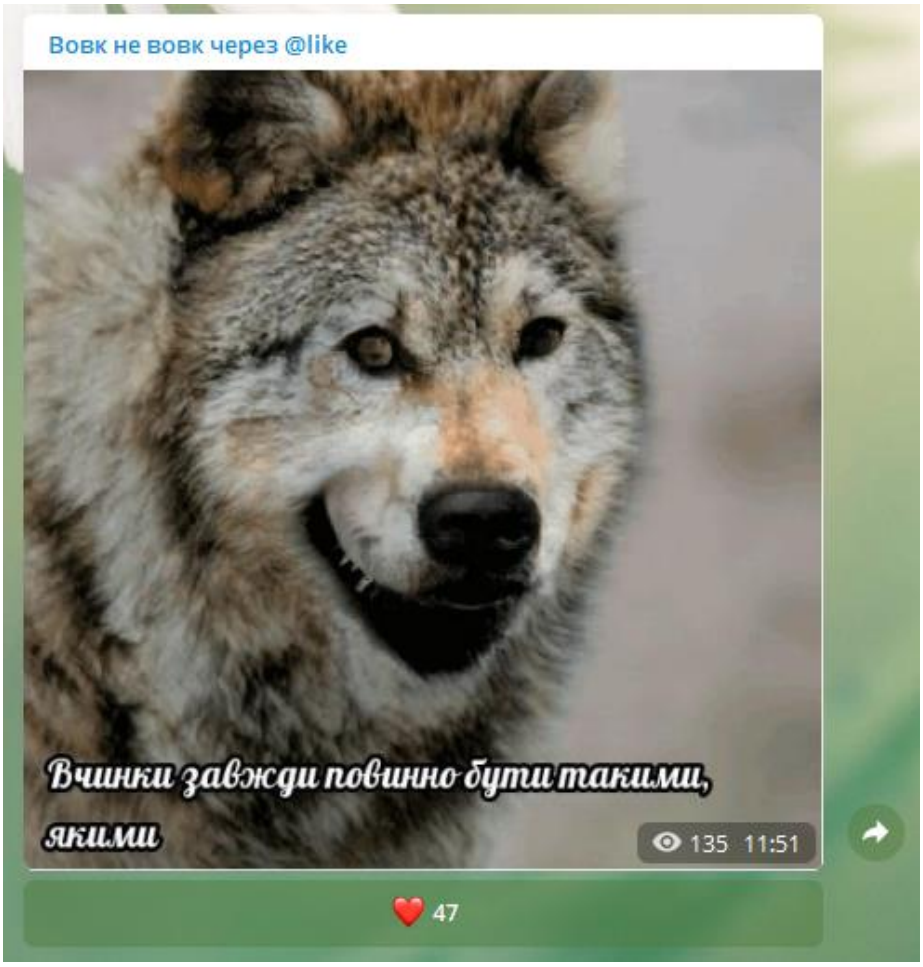
Останнім кроком є перегляд фінальних згенерованих зображень, відбір найвдаліших варіантів та їх представлення перед аудиторією.

2.4. Оцінка результатів

Для об’єктивної оцінки результатів роботи нейронних мереж було вирішено викласти результати в публічний доступ. Засобом розповсюдження медіа було обрано месенджер Telegram, що також має деякий функціонал соціальних мереж. Основною причиною вибору даного месенджеру стала його популярність серед «цільової аудиторії» подібних гумористичних зображень (молоді).

Було створено канал t.me/vovknevovk, посилання на який було розповсюджене можливими безкоштовними засобами. Наприклад, розсилкою повідомлень в чати та публікаціями в інших соціальних мережах з закликом переглянути записи у каналі та допомогти у розповсюдженні інформації.

За допомогою телегам-бота LikeBot (t.me/like) були створені публікації, що дозволяли користувачам поставити «лайк».



(Рисунок 20. Приклад фінального зображення.)

Загалом, за обмежений час розповсюдження інформації про канал та публікації згенерованих зображень, було отримано досить позитивні відгуки.

За період тривалістю дві доби, канал здобув ____ зацікавлених підписників, найпопулярніший запис отримав _____ лайків, а найбільша кількість переглядів запису становила _____.

(Рисунок - скріншоти остаточної статистики)

Окрім того, перше місце за популярністю, а отже й за вдалістю, розділили два зображення.

(Рисунок 21. Найпопулярніші зображення.)

Висновки

Було досліджено існуючі нейронні мережі та їх архітектури, такі як RNN, LSTM, GAN, BigGAN. Також було створено власну систему з рекурентної та генеративної змагальної нейронної мережі що генерують меми, зображення гумористичного характеру. Загальна оцінка випадковим чином обраних респондентів показала, що було отримано результат високої якості. Зображення були оцінені як досить смішні, або, як мінімум, цікаві для загальної аудиторії.

Основним викликом став збір та препроцессинг необхідного датасету для тренування рекурентної мережі. За умов подальшого розвитку проекту ймовірне збільшення текстового датасету та датасету зображень для покращення якості результатів.

Джерела

1 Forecasting of the Prevalence of Dementia Using the LSTM Neural Network in Taiwan, Stephanie Yang, Hsueh-Chih Chen, Chih-Hsien Wu, Meng-Ni Wu, Cheng-Hong Yan, 2021. [Електронний ресурс] Режим доступу до ресурсу: https://www.mdpi.com/2227-7390/9/5/488/html

2 A Critical Review of Recurrent Neural Networks for Sequence Learning, Zachary C. Lipton, John Berkowitz, Charles Elkan, 2015. [Електронний ресурс] Режим доступу до ресурсу: https://arxiv.org/abs/1506.00019

3 Generative Adversarial Networks, Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, 2014. [Електронний ресурс] Режим доступу до ресурсу: https://arxiv.org/abs/1406.2661

4 Long Short-Term Memory, Sepp Hochreiter, Jurged Schmidhuber, 1997. [Електронний ресурс] Режим доступу до ресурсу: https://direct.mit.edu/neco/article/9/8/1735/6109/Long-Short-Term-Memory

5 Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio, 2014. [Електронний ресурс] Режим доступу до ресурсу: https://arxiv.org/pdf/1412.3555.pdf

6 Large scale GAN training for high fidelity natural image synthesis, Andrew Brock, Jeff Donahue, Karen Simonyan, 2019. [Електронний ресурс] Режим доступу до ресурсу: https://arxiv.org/abs/1809.11096

7 Використання нейронних мереж для визначення схожості текстів українською мовою, Брус Андрій, 2020. [Електронний ресурс] Режим доступу до ресурсу: http://ekmair.ukma.edu.ua/handle/123456789/18605

8 Textgenrnn. [Електронний ресурс] Режим доступу до ресурсу: https://github.com/minimaxir/textgenrnn