КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

ГЕНЕРАТИВНІ НЕЙРОННІ МЕРЕЖІ

Виконав студент 4-го курсу
Микита ОЛІЙНИК _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук _____
Ярослав ЛІНДЕР
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
«___»_____2021р.,
протокол №___
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи 54 сторінки, 33 використаних джерел, 3 рисунки. Ключові слова: АВТОРЕГРЕСИВНІ МОДЕЛІ, ВАРІАЦІЙНІ АВТОКОДУВАЛЬНИКИ, ГЕНЕРАТИВНІ ЗМАГАЛЬНІ МЕРЕЖІ, ГЕНЕРАТИВНІ МОДЕЛІ, НЕЙРОННІ МЕРЕЖІ, МОВНІ МОДЕЛІ.

Об’єктом роботи є найбільш популярні насьогодні види нейронних мереж здатних до генерації складних даних. Предметом роботи є основні види генеративних нейронних мереж та їх застосування для практичних задач.

Метою роботи є теоретичний опис поточних підходів до перетворення нейронних мереж на генеративні моделі та поєднання деяких видів генеративних мереж для розв’язання актуальних проблем генерації даних, зокрема для керування генерацією тексту.

Інструменти розробки: мова програмування Python, фреймворк глибокого навчання PyTorch та бібліотека Transformers.

Результатами роботи є опис теоретичних основ глибокого навчання, трьох основних видів генеративних моделей та мовних моделей та запропонування нового підходу керування генерацією діалогу на основі поєднання двох видів генеративних нейронних мереж. Експеримент з одночасного застосування варіаційного автокодувальника та авторегресивної мережі було відтворено а його результати проаналізовано. Ця робота першою ставить проблему глобального керування генерацією тексту з тренуванням без вчителя. Її результати мають слугувати підґрунтям для подальших досліджень у напрямку керованої генерації тексту.

Сферами застосування генеративних моделей є такі задачі як реставрація, покращення та перетворення зображень, синтез аудіо, генерація тексту, аугментація даних та інші. Мовні моделі для генерації тексту можуть бути застосовані у будь-яких програмних комплексах, користувачем яких є людина. Значення роботи полягає у потенційному надані засобів керування мовними моделями.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

E2E – (від англ. End-to-end) – наскрізна, неперервна; стосовно нейронної мережі зазвичай – та, в якій немає недиференційованих операцій

EM – (від англ. Earth-Mover) – відстань Васерштейн-1

GAN – (від англ. Generative adversarial network) – генеративна змагальна мережа

GRU – (від англ. Gated Recurrent Unit) – вид архітектури RNN

JSD – (від англ. Jensen–Shannon divergence) – розходження Дженсена – Шеннона

KL – (від англ. Kullback–Leibler) – розходження Кульбака — Лейблера

LSTM – (від англ. Long Short-Term Memory) – вид архітектури RNN

MSE – (від англ. mean squared error) – середньоквадратична помилка

RNN – (від англ. Recurrent Neural Network) – рекурентна нейронна мережа

Seq2Seq – (від англ. Sequence-to-sequence) – вид моделей машинного навчання, вхідними та вихідними даними яких є послідовності

VAE – (від англ. Variational Autoencoder) – варіаційний автокодувальник

WGAN – (від англ. Wasserstein GAN) – GAN Васерштейна

ВСТУП

Оцінка сучасного стану об’єкта дослідження. З стрімким ростом доступних даних та обчислювальних потужностей в останнє десятиріччя, глибоке навчання також отримало бурхливий розвиток. Нейронні мережі зайняли важливу роль в усіх сферах машинного навчання та штучного інтелекту, замінив собою класичні статистичні підходи, які все ще є актуальними при малих доступних ресурсах, проте як правило поступаються підходом глибокого навчання на проблемах великого масштабу. Генеративні моделі – не виключення. Зараз в усіх передових алгоритмах побудови генеративних моделей – варіаційний автокодувальник [1], авторегресія, генеративні змагальні мережі [2] – нейронні мережі є основним компонентом, звідси й загальна назва цих моделей – генеративні нейронні мережі. Вони стрімко розвиваються, проте, як і все глибоке навчання у цілому, генеративні нейронні мережі все ще мають багато нерозв’язаних проблем та питань залишених без відповіді.

Мовні моделі, що зазвичай є представниками авторегресивних генеративних мереж, за останні роки як жодні інші зазнали вибухового розвитку, особливо після заміни старих шарів типу RNN таких як LSTM [3] та GRU [4] на механізм уваги (attention) що було втілено в архітектурі Трансформер [5]. Всі ці архітектури вдало вирішували окремі проблеми попередників – такі як наприклад коротка довжина контексту сприйняття – та ставили нові рекорди у низках текстових і не тільки задач. Проте, розвиток архітектур не змінив загальних рис притаманних авторегресивним моделям, таких як покрокова генерація одного токену за іншим.

Актуальність роботи та підстави для її виконання. Один з можливих напрямків досліджень в сфері генеративних нейронних мереж, якому було приділено недостатньо уваги – це пошук нових підходів для перетворення нейронних мереж на генеративні моделі, в тому числі й завдяки комбінації існуючих підходів задля одночасного поєднання їх найкращих особливостей.

Нерозкритою темою залишається й глобальне керування генерацією мовних моделей, на відміну від лише локального керування під час кожної окремої ітерації генерації. Ця робота присвячена обом цим темам.

Мета й завдання роботи. Мета цієї роботи полягає у розробці засобу глобального керування генерацією мовних моделей за допомогою поєднання авторегресивних генеративних нейронних мереж з варіаційним автокодувальником. Це є одним з перших застосувань ідеї варіаційного автокодувальника до мовних моделей для розв'язання практичних проблем та першим підходом до розв'язання практичної проблеми глобального керування генерацією мовних моделей, зокрема Трансформерів. Поставленими завданнями є аналіз та систематизація поточних здобутків стосовно генеративних нейронних мереж, дослідження та опис мовних моделей та зокрема архітектури Трансформер, запропонування поєднання двох видів генеративних мереж для побудови глобально керованої діалогової моделі, втілення експерименту з застосування варіаційного автокодувальника до архітектури Трансформер для розв'язання поставленої задачі, аналіз результатів експерименту та окреслення можливих подальших напрямків для досліджень.

Об'єкт, методи й засоби дослідження або розроблення. Об'єктом дослідження виступають генеративні нейронні мережі та їх поєднання для побудови керованої мовної моделі.

Об'єкт розробки – втілення експерименту з застосування варіаційного автокодувальника до архітектури Трансформер для побудови глобально керованої діалогової моделі. Основою середовища стала операційна система з відкритим вихідним кодом Ubuntu 20.04 на ядрі Linux 5.4 та систему контейнеризації Docker 20.10.2, прикладний програмний інтерфейс для обчислень загального призначення на графічних процесорах CUDA 10.0, бібліотека примітивів глибокого навчання cuDNN 7 та мова загального призначення Python 3.8. У якості фреймворку глибокого навчання було обрано PyTorch 1.4.0, у якості основи імплементації архітектур Трансформерів

було використано бібліотеку Transformers [6]. Основним інструментом розробки була обрана інтегрована середа розробки PyCharm від JetBrains з відкритим вихідним кодом.

Можливі сфери застосування. Генеративні нейронні мережі можуть знайти і знаходять використання в будь-яких програмних комплексах вихідними або проміжними даними є штучно згенеровані дані складної структури. Конкретними прикладами таких даних є фото, відео, аудіо, текст та багато інших; конкретними прикладами використань генеративних нейронних мереж є фотообробка, синтез голосу, діалогові системи та багато інших.

Мовні моделі можуть бути використані як частини інтерфейсу будь-яких програмних комплексів, користувачами яких є людина, таких чат-боти, цифрові асистенти та інших. Діалогові системи можуть бути застосовані для покращення психічного стану користувачів віртуальних співбесідників або для розвитку дітей що взаємодіють з розумними ляльками, тощо.

Повний спектр можливих сфер та конкретних сценаріїв застосувань генеративних нейронних мереж та мовних моделей зокрема ще досі повністю не відкрито, але зрозуміло що можливості великі.

Ця робота може бути використана як підґрунтя для подальших досліджень та розробок у напрямку поєднання різних видів генеративних моделей та побудови глобально керованих мовних моделей. Результати безпосередньо цієї роботи відкривають шлях до глобального впливу на генерацію авторегресивних моделей зовнішніми агентами - іншими моделями глибокого навчання, програмними комплексами або людьми. Такий вплив є засобом поєднання глобального контролю над вихідними даними ззовні моделі з локальним контролем структурою даних самою моделлю. Прикладом такого корисного поєднання у діалоговий системі може бути генерація відповіді з конкретною емоцією чи настроєм, де настрій визначається зовнішнім агентом, а конкретне формулювання відповіді вже самою нейронною мережею.

РОЗДІЛ 1. НЕЙРОННІ МЕРЕЖІ

1.1 Основні поняття

Нейронна мережа – це обчислювальна параметрична функція виду:

$$y = F_\theta(x),$$

де x – вхідні дані, y – вихідні дані, θ – параметри. Вхідні та вихідні являють собою тензори (у тому числі часто скаляри, вектори та матриці) або кортежі тензорів.

Нейронна мережа складається з архітектури та значень параметрів.

Архітектура нейронної мережі – це послідовність перетворень вхідних даних у вихідні. Параметри нейронної мережі – це конкретні значення що беруть участь у перетвореннях всередині нейронної мережі та оптимізуються під час тренування.

Тренування нейронної мережі – це процес пошуку оптимальних за заданою функцією втрат параметрів для даної архітектури нейронної мережі:

$$\theta' = \underset{\theta}{argmin}\ J(F_\theta(d_x), d),$$

де J – функція втрат, θ' – знайдені значення параметрів, а d – зовнішні дані, на яких зазвичай й обчислюється значення функції втрат.

Гіперпараметри нейронної мережі – це параметри її архітектури. Вони на відміну від звичайних параметрів під час тренування не оптимізуються і повинні бути задані заздалегідь.

1.2 Найпростіші архітектури нейронних мереж

Одним з найпростішим прикладом нейронної мережі є лінійна функція

$$y = wx + b,$$

де w та b – параметри, вага та зсув відповідно.

Формально кажучи, це є одношарова нейронна мережа з лінійною функцією активації, вхідною та вихідною розмірностями 1 та двома

параметрами. Якщо розширити вхідну розмірність, замінивши скаляри x та w векторами, отримаємо модель лінійної регресії. Проте ця модель все ще є лінійною і для більшості практичних задач недостатньо складною.

Можна ускладнити лінійну модель, поєднавши декілька лінійних моделей в одну, так щоб вихідні дані моделі на першому рівні були вхідними даними для лінійної регресії другого рівня.

$$y_1 = W_1 x + b_1,$$
$$y_2 = w_2 y_1 + b_2,$$

або

$$y_2 = w_2(W_1 x + b_1) + b_2,$$

де W_1 – матриця ваг моделі першого рівня, b_1 – вектор зсуву моделі першого рівня, w_2 – вектор ваг моделі другого рівня, b_2 – скаляр зсуву моделі другого рівня.

Не важко побачити, що така комбінація лінійних моделей все ще є лінійною моделлю. Справді, якщо зробити заміни $w' = w_2 W_1$ та $b' = w_2 b_1 + b_2$ отримаємо $y_2 = w'x + b'$. Тому щоб досягти ускладнення моделі, до проміжного значення y_1 можна застосувати нелінійну функцію активації σ:

$$z_1 = W_1 x + b_1,$$
$$y_1 = \sigma(z_1),$$

У такому вигляді отримана модель має право називатися перцептроном, що уперше було описано Френком Розенблатом у 1958 році [7]. Кожен елемент $y_1^i = \sigma(W_1^i x + b_1^i)$ вектора y_1 та y_2 називаються штучними нейронами.

За термінологією Розенблата, цей перцептрон є перцептроном з одним прихованим шаром через наявність лише одного вектору y_1 – шару асоціативних елементів типу А. В сучасному глибокому навчанні таку мережу доцільно називати двошаровою через наявність двох лінійних параметричних операцій – шарів, але єдиної точки зору на те що вважати шаром нема, а самі поняття нейронів та шарів є дуже не чіткими та використовуються рідко через різноманітність архітектур та складність виокремлення в них таких складових.

Основною відмінністю перцептрона Розенблата від сучасних штучних нейронних мереж є область значень елементів. В Розенблата всі вони були з множини (-1; 0; 1), що вплинуло й на вибір функції активації – вона була порогова, й на алгоритм тренування – метод корекції помилки. В сучасних нейронних мережах в якості області значення ваг, зсувів та активацій використовується множина дійсних чисел $\mathbb{R}$, а алгоритми тренування вимагають диференційованості мережі. Так як порогова функція має похідну 0 в усіх точках окрім однієї, вона є невдалим вибором. Найбільш популярними функціями активації зараз є:

- Логістична, сігмоїда: $\sigma(x) = \frac{1}{1+ e^{-x}}$
- Гіперболічний тангенс: $\sigma(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
- ReLU (rectified linear unit) [8]: $\sigma(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$
- Leaky ReLU (дірява ReLU) [9]: $\sigma(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases}, a \ll 1$

Як правило, вибір функції активації рідко суттєво впливає на результати.

1.3 Тренування нейронних мереж

1.3.1 Обчислення градієнту

Зараз, під нейронною мережею, часто мають на увазі параметричну функцію в якої до умови на обчислюваність додається умова на диференційованість. Так є через основний метод тренування штучних нейронних мереж – метод зворотного поширення помилки [10][11], що в своїй суті є градієнтним методом оптимізації.

Для обраної функції втрат J та даної архітектури F_θ з параметрами θ що складається з N шарів з проміжними активаціями $y_{1..N}$, метод зворотного поширення помилки обчислює часткові похідні для J відносно всіх параметрів:

$$\frac{\partial J}{\partial \theta_i},$$

де $i \in [1,\ N]$ – номер шару. Тут і далі для спрощення запису вважаємо всі активації y_i і відповідно z_i та параметри θ_i (w_i чи b_i) скалярами. У випадку більших розмірностей обчислення аналогічні.

Обчислюються ці похідні за допомогою ланцюгового правила диференціювання складної функції:

$$\frac{\partial J}{\partial \theta_i} = \frac{\partial J}{\partial y_i}\frac{\partial y_i}{\partial \theta_i},$$

$$\frac{\partial J}{\partial y_i} = \frac{\partial J}{\partial y_{i+1}}\frac{\partial y_{i+1}}{\partial y_i} = \frac{\partial J}{\partial y_N}\frac{\partial y_N}{\partial y_{N-1}} \dots \frac{\partial y_{i+2}}{\partial y_{i+1}}\frac{\partial y_{i+1}}{\partial y_i}$$

Можна побачити, що для обчислення похідної відносно активації шару зручно використовувати похідну з наступного шару. Саме тому зазвичай для уникнення повторних операцій обчислення похідних починається з останнього шару і прямує до першого – з цього і випливає назва методу, на відміну від прямого проходу, що відбувається під час обчислень значень активацій шарів.

Похідні $\frac{\partial y_i}{\partial \theta_i}$ та $\frac{\partial y_{i+1}}{\partial y_i}$ можна розписати далі врахувавши наприклад функції активації:

$$\frac{\partial y_i}{\partial \theta_i} = \frac{\partial y_i}{\partial z_i}\frac{\partial z_i}{\partial \theta_i},$$

$$\frac{\partial y_{i+1}}{\partial y_i} = \frac{\partial y_{i+1}}{\partial z_{i+1}}\frac{\partial z_{i+1}}{\partial y_i},$$

$$\frac{\partial y_i}{\partial z_i} = \frac{\partial \sigma(z_i)}{\partial z_i} = \sigma'(z_i)$$

Просте обчислення похідної є основною вимогою до функцій активацій, після нелінійності. Так наприклад для найбільш популярних функцій активацій:

- Логістична, сігмоїда: $\sigma'(x) = \sigma(x)(1 - \sigma(x))$
- Гіперболічний тангенс: $\sigma'(x) = 1 - \sigma(x)^2$
- ReLU (rectified linear unit): $\sigma'(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$

- Leaky ReLU (дірява ReLU): $\sigma'(x) = \begin{cases} \alpha, & x < 0 \\ 1, & x \geq 0 \end{cases}, a \ll 1$

У випадку лінійного шару $z_{i+1} = w_i y_i + b_i$ як було приведено вище, похідні приймають вигляд:

$$\frac{\partial z_{i+1}}{\partial y_i} = w_i,$$

$$\frac{\partial z_{i+1}}{\partial w_i} = y_i,$$

$$\frac{\partial z_{i+1}}{\partial b_i} = 1$$

У випадку інших шарів похідні обчислюються так само, розписом за ланцюговим правилом до найпростіших операцій, похідні яких відомі. Такий універсальний підхід до визначення нейронних мереж (прямого проходу) та їх тренування (зворотного проходу) є по-перше дуже гнучким та дозволяє працювати з величезним простором можливих архітектур, а по-друге дозволяє використовувати автоматичне диференціювання. Сьогодні всі популярні фреймворки глибокого навчання (такі як наприклад PyTorch [12] та Tensorflow [13]) дозволяють задавати лише архітектуру прямого проходу нейронної мережі, автоматично беручи на себе всі обчислення похідних під час зворотного проходу.

Маючи обчислені похідні $\frac{\partial J}{\partial \theta_i}$, можемо складемо з них вектор градієнту $\nabla_\theta J$ – вектор, який вказує напрямок найскорішого зростання функції втрат при поточних значеннях параметрів.

1.3.2 Градієнтний спуск

Тренування нейронної мережі полягає у ітеративній оптимізації функції втрат, при якій кожен крок робиться у напрямку, пропорційному протилежному значення градієнту. Цей алгоритм має назву градієнтного спуску, і один його крок оновлення параметрів можна записати як:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta),$$

де J – функція втрат, θ – вектор параметрів, η – крок градієнтного спуску. Початкові значення параметрів є невеликими випадковими числами.

Градієнтний спуск є найпростішим з методів градієнтної оптимізації. І хоча методи другого та більших порядків не є застосовними на практиці до великих нейронних мереж через обчислювальну складність, існує низка чисельних алгоритмів першого порядку, що на практиці виявляються кращими за звичайний градієнтний спуск.

Найпростішим розвитком градієнтного спуску є градієнтний спуск з імпульсом (momentum) [14]. Його ідея полягає у тому щоб використовувати для оновлення значень параметрів не значення градієнту безпосередньо, а згладжені значення градієнтів за допомогою експоненційного згладжування. Інтуїтивно, це зменшує коливання градієнтного спуску за напрямками з великою варіативністю, та більше спрямовує його у напрямку постійного спадання значення функції втрат, що в кінці кінців дає пришвидшений збіг алгоритму.

$$m_t = \gamma m_{t-1} + \eta g_t$$
$$\theta_{t+1} = \theta_t - m_t$$

де g_t – вектор градієнту обчисленого на кроці t, γ – параметр алгоритму, що зазвичай покладається рівним 0.9.

Прикладом подальшого розвитку алгоритмів градієнтного спуску є Adam (від англ. Adaptive Momentum) [15]. Як градієнтний спуск з імпульсом, Adam підраховує експоненційне згладжене градієнтів. Окрім цього, Adam підраховує і експоненційне згладжене квадратів градієнтів, як в алгоритмах Adadelta [16] та RMSprop, що використовується для обчислення значення кроку індивідуального для кожного параметру.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2,$$

де β_1 та β_2 – параметри алгоритму, значення яких були запропоновані авторами як 0.9 та 0.999 відповідно. Так як початкові значення середніх m_t та v_t покладаються рівними нулю, робиться корекція їх значень:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Нарешті, оновлення параметрів відбувається за правилом:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon}\hat{m}_t$$

Авторами Adam значення β_1, β_2 та ε за замовчуванням покладаються рівними 0.9, 0.999 та 10^{-8} відповідно. Через свою високу ефективність на практиці, надалі в цій роботі у якості алгоритму градієнтної оптимізації буде використовуватися Adam.

1.3.3 Функції втрат

В машинному навчанні часто виділяють такі основні класи задач як навчання з учителем, навчання без учителя, навчання з підкріпленням та інші. В контексті глибокого навчання, відмінність між цими методами полягає у тому як формується «сигнал помилки» J, відносно якого обчислюються градієнт для тренування моделі. У навчанні з вчителем, найбільш основному класі задач на який тісно спираються інші підходи, J є функцією втрат визначеною на тренувальному наборі d що складається з пар вхідних та цільових вихідних $(x_i, y_i), i = 1..n$, де n – розмір тренувального набору даних. Головними вимогами до функції втрат є обчислюваність та диференційованість. Отже, нейронні мережі також можуть бути функціями втрат при тренуванні інших мереж.

Двома основними задачами навчання з підкріпленням є регресія (y_i – дійсні числа або вектори) та класифікація (y_i – категорійна змінна).

Найбільш поширеною функцією втрат в задачі регресії є середня квадратична похибка (mean squared error, MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

де $\hat{y}_i = F_\theta(x_i)$, тобто $\hat{y}_i$ – дані на виході нейронної мережі F_θ при вхідних даних x_i. Інтуїтивно можна побачити, що мінімізація MSE наближує вихід моделі $\hat{y}_i$ до цільових значень y_i.

У задачі класифікації найбільш часто використаною метрикою є перехресна ентропія, яка у загальному вигляді записується як:

$$H(p, q) = E_p(-\log q)$$

Для випадку коли розподіли p та q є дискретними:

$$H(p, q) = -\sum_x p(x) \log q(x)$$

Під час тренування нейронної мережі для розв'язання задачі багатокласової класифікації, у якості розподілу $p(x)$ є цільові класи в унітарному кодуванні (у вигляді вектору з нулів та однієї одиниці у позиції з номером що дорівнює класу), а $q(x)$ – це поточна оцінка нейронною мережею ймовірнісного розподілу, отримана з чисельного вектору на кінці нейронної мережі z за допомогою спеціальної функції активації – softmax, яка задається наступним чином:

$$\sigma\colon \mathbb{R}^M \to [0;\ 1]^M,\ \ \sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{M} e^{z_k}},$$

де M – кількість класів у розподілі, розмірність вихідного вектору. Вектор z в цьому випадку має назву логітів.

Не важко переконатись що всі елементи вектору на виході softmax лежать у межах від 0 до 1 а їх сума дорівнює 1, що робить softmax зручною для обчислення апроксимації категорійного розподілу.

У випадку $M = 2$, категорійний розподіл стає розподілом Бернуллі і отримуємо задачу бінарної класифікації. Так як ймовірність одного класу

однозначно визначається ймовірністю іншого (їх сума дорівнює 1), замість вектору розмірності 2 достатньо використовувати одне дійсне число, а замість softmax на кінці мережі достатньо сігмоїди.

Функція втрат приймає вигляд:

$$H = -\sum_{i=1}^{n} y_i \log \widehat{y_i} + (1 - y_i)\log (1 - \widehat{y_i})$$

Інтуїтивно можна побачити, що мінімізація бінарної перехресної ентропії збільшує $\widehat{y_i}$ на виході мережі коли цільовий клас y_i дорівнює 1 та зменшує у протилежному випадку, тим самим наближаючи два розподіли.

Перехресна ентропія тісно пов'язана з інформаційною ентропією Шеннона та розходженням Кульбака – Лейблера (KL):

$$H(p, q) = H(p) + D_{KL}(p||q)$$

Розходження Кульбака-Лейблера, у свою чергу, також може бути функцією втрат у задачі класифікації. Для дискретних розподілів ймовірностей воно має вигляд:

$$D_{KL}(p||q) = \sum_{x} p(x) \log \frac{p(x)}{q(x)}$$

1.3.4 Стохастичний градієнтний спуск

До цього моменту градієнтний спуск - обчислення функції втрат, градієнтів на ній та оновлення параметрів описувалось так що враховувався повністю увесь набір даних. Такий підхід дає найкращі оцінки функції втрат та градієнтів, проте має суттєвий практичний недолік – лише один крок градієнтного спуску вимагає прямого та зворотного проходу через нейронну мережу всього набору даних, що з урахуванням розмірів сучасних нейронних мереж та наборів даних є дуже витратним за часом.

Модифікацією градієнтного спуску є стохастичний градієнтний спуск, що за один крок розглядає не весь набір даних а лише один екземпляр. Так досягається мінімальна кількість обчислень необхідних для одного кроку,

проте з’являються проблеми того що обчислені градієнти є не точними, вони далекі від напрямку найшвидшого зростання функції і в них багато шуму, через що оптимізація значно сповільнюється або повністю зупиняється.

Компромісним рішенням є міні-пакетний градієнтний спуск (mini-batch gradient descent). У ньому, на кожному кроці розглядається невелика випадкова підвибірка (міні-пакет) з набору даних. Регулюючи розмір цієї підвибірки як додатковий гіперпараметр тренування, можна досягти невеликої кількості обчислень необхідних для одного кроку, зберігши швидкість оптимізації у перерахунку на один крок.

РОЗДІЛ 2. ГЕНЕРАТИВНІ НЕЙРОННІ МЕРЕЖІ

2.1 Автокодувальники

2.1.1 Звичайні автокодувальники

Автокодувальник – це алгоритм навчання без вчителя; нейронна мережа, що на виході відтворює свої вхідні дані. В автокодувальнику зазвичай виділяють дві складові – кодувальник та декодувальник, які поєднанні між собою кодом z, латентною змінною, що трактується як навчене представлення даних. Кодувальник перетворює вхідні дані на латентний код, декодувальник перетворює латентний код на вхідні дані. Так як латентний код є вектором дійсних чисел, вся система може вважатися єдиною нейронною мережею і тренуватися E2E з функцією втрат, наприклад, будь-якою метрикою у вхідному (вихідному) просторі. Так, якщо автокодувальник відтворює вектори дійсних чисел, у якості функції втрат зручною може бути MSE:

$$F_\theta \colon \mathbb{R}^M \to \mathbb{R}^L$$

$$G_\omega \colon \mathbb{R}^L \to \mathbb{R}^M$$

$$J = \frac{1}{n}\sum_{i=1}^{n}(x_i - (G_\omega \circ F_\theta)(x_i))^2,$$

де F_θ та G_ω – кодувальник та декодувальник відповідно.

Зазвичай вважається що латентна розмірність L менша за розмірність вхідних та вихідних даних M. У цьому випадку, навіть найпростіший автокодувальник має практичне значення, адже у невеликому латентному векторі в ході навчання з'являється стиснене представлення даних. Такий принцип примусового пропуску інформації в нейронній мережі через занадто маленький вектор для виділення в цьому векторі високорівневих ознак називається шийкою пляшки (bottleneck). Завдяки ньому, автокодувальник може бути використано для навчання ознак та представлень, зниження розмірності. Наприклад, у найпростішому випадку якщо F_θ та G_ω є лінійними функціями, автокодувальник стає еквівалентним методу головних компонент.

2.1.2 Знешумлювальні автокодувальники

Розвитком звичайного автокодувальника є знешумлювальні автокодувальники. Ідея знешумлювального автокодувальника полягає у тому щоб подавати на вхід кодувальника не оригінальні дані, а частково спотворені. Тоді, в ході навчання автокодувальник навчиться перетворювати зашумлені вхідні дані на чисті – знешумлювати їх. Цей підхід базується на ідеї про те, що латентна змінна вбирає в себе достатньо високорівневі характеристики вхідних даних, які не залежать від часткових спотворень. Знешумлювальні автокодувальники можуть бути використано для покращення зображень, наприклад, для розв'язання задачі підвищення роздільної здатності.

Ідея часткового спотворення вхідних даних також лежить в основі аугментації даних – низки підходів, що дозволяє штучно збільшити кількість тренувальних даних, що може діяти як регуляризація та в результаті чого стає можливим отримання в результаті тренування більш стійких моделей.

2.1.3 Варіаційні автокодувальники

Описані до цього автокодувальники не є генеративними мережами. Справді, використовувати декодувальник G_ω звичайного автокодувальника як генератор не вдасться, адже не зрозуміло який латентний код z подавати на вхід декодувальника. Єдине чинне джерело кодів z – це кодувальник F_θ, отриманий під час тренування автокодувальника $(G_\omega \circ F_\theta)$. Але кодувальник не є стохастичною моделлю і лише детерміновано обчислює латентний код за вхідними даними. Також не можна згенерувати z напряму з деякого відомого простого розподілу, адже справжній розподіл латентних векторів в нейронних мережах є невідомим і як правило складним.

Опишемо тепер як перетворити автокодувальник на генеративну модель.

Сформулюємо задачу генерації даних $\{x\}$ як задачу апроксимації невідомого розподілу $p(x)$.

Генеративна (породжувальна) модель – це модель, яка вирішує задачу генерації даних, тобто апроксимує $p(x)$. Як правило, апроксимація полягає в оцінці параметрів розподілу. Для практичних застосувань важливо те, що генеративна модель розв'язує задачі оцінки густини розподілу та генерації вибірки з розподілу.

Інший клас моделей – дискримінативні (розрізнювальні, або умовні) – апроксимують умовний розподіл $p(y|x)$. Якщо апроксимація полягає в обчисленні параметрів розподілу випадкової величини y за даним значенням x, така модель може бути побудована за допомогою нейронної мережі за означенням нейронної мережі даним на початку як функції y від x. Тобто, нейронні мережі в простому варіанті є дискримінативними моделями і питання полягає у тому як їх перетворити на моделі генеративні.

Розпишемо розподіл $p(x)$ як:

$$p(x) = p(x|z)p(z),$$

де $p(z)$ – деякий відомий розподіл. Тоді задачу генерації даних можна представити як задачу апроксимації умовного розподілу $p(x|z)$. Це можна

зробити за допомогою дискримінативної моделі, нейронної мережі G_ω, вхідними даними якої є випадкова змінна z, а вихідними – параметри розподілу x:

$$p(x|z) \approx G_\omega(x|z),$$

Назвемо G_ω генератором. Тоді маючи натренований генератор, можна згенерувати з відомого розподілу змінну z, подати її на генератор і на виході отримати параметри невідомого розподілу змінної x.

У контексті автокодувальників, генератор є декодувальником. Відмінність звичайного автокодувальника від варіаційного полягає у тому що компоненти першого у якості вихідних значень мають змінні, а не параметри розподілів випадкових змінних.

Аналогічно заміні декодувальника звичайного автокодувальника на генератор варіаційного автокодувальника, представимо й кодувальник у вигляді $p(z|x)$. Формула Байєса поєднує ці розподіли: $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$, але так як $p(x) = \int p(x|z)p(z)dz$, обчислити $p(z|x)$ за цим виразом практично не можливо.

Для обчислення $p(z|x)$ скористаємось варіаційним виведенням (variational inference) – знов апроксимуємо розподіл:

$$p(z|x) \approx F_\theta(z|x),$$

Для того щоб натренувати параметри θ так щоб F_θ дійсно було гарною апроксимацією, будемо мінімізовувати розходження Кульбака-Лейблера між цими двома розподілами, тобто:

$$\theta = \underset{\theta}{\operatorname{argmin}} D_{KL}\big(\, F_\theta(z|x) \,||\, p(z|x)\big) =$$

$$= \underset{\theta}{\operatorname{argmin}} E_{z\sim\, F_\theta(z|x)} \log \frac{F_\theta(z|x)}{p(z|x)} =$$

$$= \underset{\theta}{\operatorname{argmin}} \big(E_{z\sim F_\theta(z|x)} \log F_\theta(z|x) - E_{z\sim\, F_\theta(z|x)} \log p(z|x)\,\big) =$$

$$= \underset{\theta}{\operatorname{argmin}} \big(E_{z\sim\, F_\theta(z|x)} \log F_\theta(z|x) - \; E_{z\sim\, F_\theta(z|x)} \log p(x,z) + \log p(x)\,\big) =$$

$$= \underset{\theta}{\operatorname{argmin}}\big(E_{z\sim F_\theta(z|x)} \log F_\theta(z|x) - E_{z\sim F_\theta(z|x)} \log G_\omega(x|z)$$

$$- E_{z\sim F_\theta(z|x)} \log p(z)\big) =$$

$$= \underset{\theta}{\operatorname{argmax}}\big(E_{z\sim F_\theta(z|x)} \log G_\omega(x|z) - D_{KL}(F_\theta(z|x) \,||\, p(z))\big)$$

Вираз $\log p(x)$ має назву evidence, а останній вираз $E_{z\sim F_\theta(z|x)} \log G_\omega(x|z) - D_{KL}(F_\theta(z|x) \,||\, p(z))$ має назву ELBO - evidence lower bound. Така назва походить через те, що $\log p(x) = ELBO + D_{KL}\big(F_\theta(z|x) \,||\, p(z|x)\big)$, а розходження Кульбака-Лейблера D_{KL} завжди невід'ємне, тобто ELBO дійсно є нижньою межею значення $\log p(x)$.

Першим доданком ELBO є функція логарифмічної правдоподібності (log-likelihood), що максимізується в методі максимальної правдоподібності, і її мінімізація аналогічна тому як мінімізується похибка реконструкції в звичайних автокодувальниках.

Другий доданок є розходженням між розподілом обчисленим кодувальником та деяким відомим розподілом латентних кодів. Так як ці коди уявляють собою вектори дійсних чисел, зазвичай доцільно у якості відомого розподілу обирати багатовимірний стандартний нормальний розподіл. Параметрами цього розподілу є вектор середніх значень (з елементами що дорівнюють нулю) і матриця коваріацій. Для того щоб спростити обчислення та зменшити кількість параметрів, матриця коваріацій покладається діагональною (з елементами що дорівнюють 1).

Варіаційні автокодувальники мають й просте інтуїтивне пояснення як розвиток звичайних автокодувальників. Якщо відкинути різницю на кінці – відмінність між функціями втрат та правдоподібності – різниця усередині полягає у тому, що латентна змінна замінена латентним розподілом випадкових змін, на який покладено регуляризацію у вигляді KL-розходження, яка штрафує кодувальник за генерацію латентних кодів з розподілом далеким від стандартного нормального. Саме ця регуляризація і є

тим, що привносить структуру у латентний простір і дозволяє генератору – декодувальнику працювати з випадковими векторами.

Тренується варіаційний автокодувальник як звичайна нейронна мережа з функцією втрат:

$$J = -ELBO = -\sum_{x} F_\theta(z|x) \log G_\omega(x|z) + D_{KL}(F_\theta(z|x) \,||\, p(z))$$

Слід зазначити, що за рахунок того, що будь-який нормальний розподіл можна представити у вигляді лінійного перетворення стандартного нормального розподілу – $\mathrm{N}(\mu, \sigma^2) = \mu + \sigma * \mathrm{N}(0, 1)$, і за рахунок того, що кодувальник видає ці параметри розподілу – μ та σ, тренування можливе у режимі E2E. Вибірку латентних кодів у міні-пакеті потрібно обирати не з $\mathrm{N}(\mu, \sigma^2)$ а з $\mathrm{N}(0, 1)$ і потім помножувати на σ і додавати μ, обчислених кодувальником. Такий трюк дозволяє уникати потреби робити обернений прохід через недиференційовану операцію обрання вибірки з розподілу і дістав назву трюком репараметризації (reparametrization trick) [1].

2.2 Генеративні змагальні мережі

Знов сформулюємо задачу генерації даних $\{x\}$ як задачу апроксимації невідомого розподілу $p(x)$. На цей раз розпишемо:

$$p(x) = g(p(z)),$$

де $p(z)$ – деякий відомий розподіл. Тоді задачу генерації даних можна представити як задачу апроксимації функції трансформації розподілу $p(z)$ на $p(x)$. Це можна зробити за допомогою нейронної мережі G_ω, вхідними даними якої є випадкова змінна z, а вихідними – x:

$$g(z) \approx G_\omega(z),$$

Назвемо G_ω генератором. Тоді маючи натренований генератор щоб синтезувати випадкову змінну x з невідомого розподілу, достатньо синтезувати відому змінну z та подати її на генератор.

Ключовою теоретичною відмінністю генератора генеративних змагальних мереж від генератора варіаційних автокодувальників є те, що в генеративних змагальних мережах генератору не є необхідним обчислення на своєму виході параметрів невідомого розподілу $p(x|z)$ – достатньо лише обчислення одного прикладу x, що відповідає поданому z.

Розглянемо також ймовірність того, що приклад $\hat{x}$ було обрано з розподілу $p(x)$. Цю ймовірність можна апроксимувати нейронною мережею D_θ:

$$p(\hat{x} \sim p(x)) \approx D_\theta(\hat{x})$$

Назвемо D_θ дискримінатором. Його вхідними даними є x, а вихідними – ймовірність того, що приклад справді належить до $\{x\}$. Тоді система з генератору G_ω та дискримінатору D_θ називається генеративними змагальними мережами [2]. Особливістю цієї системи є те, що обидві нейронні мережі можуть бути натренованими одночасно та E2E у манері гри з нульовою сумою.

Сформулюємо задачу тренування дискримінатора як таку щоб надавати найближчі до одиниці ймовірності справжнім прикладам з набору даних $\{x_{1..n}\}$ та найближчі до нуля – прикладам, згенерованих генератором. Задачею генератора, відповідно, буде отримання якомога ближчої до 1 оцінки дискримінатора згенерованих прикладів. Задачу тренування V системи генеративних змагальних мереж можна подати у вигляді:

$$\min_{G_\omega} \max_{D_\theta} V(D_\theta, G_\omega) = E_{x\sim p(x)} \log\big(D_\theta(x)\big) - E_{z\sim p(z)} \log\big(D_\theta\big(G_\omega(z)\big)\big)$$

На практиці оптимізація V відбувається для G_ω та для D_θ почергово.

2.2.1 Генеративні змагальні мережі Васерштейна

Позначимо $C(G_\omega)$ критерій V за умови оптимальності дискримінатора D_θ:

$$C(G_\omega) = \max_{D_\theta} V(G_\omega, D_\theta)$$

Можна показати, що мінімізація критерію C відповідає мінімізації розходження Дженсена – Шеннона (Jensen–Shannon divergence JSD), між розподілами даних $p(x)$ та генератора $p_g(x)$ [2]:

$$\mathrm{C}(G_\omega) = -\log(4) + 2 * JSD(p||p_g)$$

Порівняємо тепер розходження Дженсена – Шеннона з EM відстанню (Earth-Mover distance) або відстанню Васерштейна-1:

$$W(p, p_g) = \inf_{\gamma \in \Pi(p, p_g)} E_{(x,y)\sim\gamma}||x - y||, \quad (1)$$

де $\Pi(p, p_g)$ – множина всіх спільних розподілів, чиї відособлені розподіли дорівнюють відповідно p та p_g.

Інтуїтивно, $\gamma(x, y)$ є «кількістю маси», що потрібно перенести з точки x у точку y для того щоб перетворити розподіл p на p_g, або іншими словами, транспортним планом. Тоді EM відстань відповідно є ціною оптимального транспортного плану.

Можна показати, що EM відстань є значно кращою за розходження Дженсена – Шеннона як функція втрат градієнтної оптимізації в силу своєї слабкості да гладкості [17]. Ідея генеративних змагальних мереж Васерштейна полягає у оптимізації EM відстані замість JSD. Але залишається питання обчислювання EM відстані, адже обчислення за формулою (1) є дуже складними. На щастя, двоїстість Канторовича – Рубенштейна дозволяє переписати EM відстань у вигляді [17]:

$$\mathrm{W}(p, p_g) = \sup_{||f||_L \leq 1} E_{x\sim p} f(x) - E_{x\sim p_g} f(x),$$

де $||f||_L \leq 1$ – множина всіх відображень з $\{x\}$ в $\mathbb{R}$, що задовольняють умові Ліпшиця з константою 1.

Отже, для того щоб обчислити EM відстань, можна спробувати апроксимувати f нейронною мережею F_θ, і знайти її параметри θ розв'язавши наступну задачу оптимізації:

$$\max_\theta E_{x\sim p} F_\theta(x) - E_{z\sim p(z)} F_\theta(G_\omega(z)), \quad (2)$$

$$f(x) \approx F_\theta(x)$$

Згадаємо, що задача генератора постає у мінімізації ЕМ відстані двох розподілів. Знов приходимо до змагального тренування, при якому одна й та ж функція (2) максимізується дискримінатором та мінімізується генератором. Ця система й має назву генеративних змагальних мереж Васерштейна.

І хоча F_θ вже і не є дискримінатором, заміна JSD на ЕМ відстань на практиці значно полегшує тренування GAN, роблячи його більш стабільним та зменшуючи кількість гіперпараметрів. На відміну від звичайних GAN, де генератор та дискримінатор повинні балансувати під час тренування, дискримінатор WGAN насправді апроксимує відстань Васерштейна, а отже може бути натренованим до збіжності.

Не розв'язаним питанням залишається задоволення умови Ліпшиця функцією F_θ. В оригінальній роботі по WGAN автори пропонують обрізати ваги мережі щоб ті лежали у деякому замкненому інтервалі. В подальших роботах було запропоновано більш доцільних підхід, WGAN-GP [18]. Він полягає у додаванні до оптимізуємої функції (2) середньо квадратичного відхилення норми градієнтів моделі від 1. Модифікована функція втрат дискримінатора тоді має вигляд:

$$J(F_\theta) = E_{z\sim p(z)}F_\theta\big(G_\omega(z)\big) - E_{x\sim p}F_\theta(x) + \lambda E_{\hat{x}\sim p}\big(\big|\big|\nabla_{\hat{x}}F_\theta(\hat{x})\big|\big| - 1\big)^2$$

2.3 Авторегресивні моделі

Авторегресивні моделі служать для моделювання часових рядів у яких кожне наступне значення змінної залежить від попередніх. Зазвичай, ця залежність моделювалась лінійно, проте у контексті глибокого навчання моделювання відбувається за допомогою глибокої нейронної мережі, набагато більш складної та виразної моделі.

Окрім моделювання часових рядів, авторегресивні нейронні мережі можуть бути вдало використані для моделювання розподілів будь-яких складних змінних, що розкладаються на впорядковану послідовність більш простих змінних, у якій кожна наступна змінна залежить від попередніх.

Прикладом такого складного розподілу може слугувати текст, якій розкладається на слова, які зазвичай пишуться у послідовності зліва направо і розподіл яких можна змоделювати простим категорійним розподілом над словником. Іншим прикладом може слугувати аудіо, яке так само як і текст на слова, розкладається на кадри спектрограми [19] або навіть значення амплітуди [20].

З ймовірнісної точки зору, розподіл складної змінної $p(x)$ розкладається на умовні розподіли наступним чином:

$$p(x) = \prod_{t=1}^{L} p(x_t|x_1, \dots, x_{t-1}) = \prod_{t=1}^{L} p(x_t|x_{<t}) ,$$

де L – довжина послідовності. Кожен умовний розподіл апроксимується однією й тією ж авторегресивною нейронною мережею:

$$p(x_t|x_{<t}) \approx F_\theta(x_t|x_{<t})$$

Тренування авторегресивних нейронних мереж зазвичай відбувається у режимі так званого teacher forcing. За нього, на вхід моделі подається одночасно всі послідовність $x_{1..L-1}$ а на виході моделі очікується та сама послідовність зсунута вліво на один елемент - $x_{2..L}$. Кінцева функція втрат усереднюється по значенням функцій втрат між всіма парами виходів моделі та наступних елементів послідовності:

$$J = \frac{1}{L-1} \sum_{t=2}^{L} J'\left(F_\theta(x_t|x_{<t}), x_t\right),$$

де J' - функція втрат між апроксимацією моделлю розподілу $p(x_t|x_{<t})$ та елементом послідовності з номером t. Якщо елементи послідовності дискретні, ця функція втрат зазвичай є перехресною ентропією.

Генерація вибірки з апроксимованого нейронною мережею розподілу є ітеративним процесом. На кожному кроці поточна послідовність $x_{<t}$ подається на вхід нейронної мережі F_θ, яка обчислює параметри розподілу $p(x_t|x_{<t})$. З цього розподілу обирається новий елемент послідовності $x_{<t}$, він додається до поточної послідовності і робиться перехід на наступну ітерацію. Генерація

зупиняється за сигналом окремого класифікатора зупинки (або за досягненням максимальної довжини), що як правило реалізується як окрема голівка на кінці тієї ж самої мережі-генератора F_θ. Якщо змінна x_t є дискретною, така голівка заміняється окремою категорією, і зупинка відбувається якщо на черговій ітерації було згенеровано значення що дорівнює саме цій категорії зупинки. У випадку тексту, це називається токеном зупинки.

Можна побачити, що однією з вимог до авторегресивної нейронної мережі є можливість працювати з послідовностями різної довжини. Існують окремі спеціалізовані архітектури (розглянуті далі), що дозволяють це робити для однієї послідовності. Проте для ефективного тренування за допомогою міні-пакетного градієнтного спуску та для швидкої генерації одразу декількох послідовностей одночасно потрібна здатність працювати з декількома послідовностями різної довжини одночасно. Це досягається приведенням послідовностей до однієї довжини – набивкою (padding) за допомогою спеціальних значень у послідовності, що ігноруються моделлю на вході та не враховуються при обчисленні функції втрат. Реалізується це за допомогою механізму маскування – занулення функції втрат та вхідних даних, а у випадку дискретних змінних x_t ще й за допомогою додавання окремої категорії набивки, яка у текстовому випадку має назву токену набивки.

РОЗДІЛ 3. МОВНІ МОДЕЛІ

3.1 Загальні положення

Мовна модель – це текстова генеративна модель. Так як текст уявляє собою послідовність слів – дискретних змінних розподіл яких просто описується категорійним – під мовними моделями часто мається на увазі авторегресивна генеративна модель, що працює з текстом. В контексті глибокого навчання під мовною моделлю часто мається на увазі нейронна

мережа що апроксимує розподіл $p(x_t|x_{<t})$, де $x_{1..L}$ – послідовність токенів. Токени – це одиниці тексту, такі як слова, літери чи морфеми. Визначення словника токенів залежить від токенизатора та підходу до токенізації тексту.

Для того щоб токен подати на нейронну мережу у вигляді вектору дійсних чисел, найпростішим підходом може слугувати унітарне кодування, проте воно не є ефективним за великого розміру словника. Перевага надається матриці ембеддінгів (embedding), в якій кожному можливому токену з словника відповідає один стовпчик – ембеддінг – порівняно невеликого розміру. Ця матриця складається з дійсних чисел і тренується градієнтним спуском разом з іншими параметрами моделі, тож вважається компонентом нейронної мережі.

Більшість з сказаного про авторегресивні моделі в попередньому розділі виконується й для мовних моделей. Окрім спеціальних токенів зупинки послідовності ($< eos >$, end-of-sequence) та токену набивки ($< pad >$, padding) ще є важливим токен початку послідовності ($< bos >$, beginning-of-sequence). Він використовується як перший елемент послідовності $x_{1..L}$ для того щоб почати генерацію тексту з нуля.

Є декілька різних підходів до обрання наступного токену x_{t+1} з розподілу $p(x_{t+1}|x_{1..t})$. Найпростіші з них – жадібне обрання токену з найбільшою ймовірністю чи обрання випадкового токену з категорійного розподілу з вектором ймовірностей $p(x_{t+1}|x_{1..t})$ – іноді приводять до занадто загальних та простих виразів [21]. Часто використовується так званий променевий пошук, на кожній ітерації якого підтримується не одна найймовірніша послідовність, а набір з K найбільш ймовірних. Проблемою застосування променевого пошуку до великих мовних моделей є збільшення затрат за часом пропорційно до ширини променю K.

В цій роботі буде розглядатися так званий ядерний синтез (nucleus sampling) [21]. Двома основними його компонентами є відбір найбільш ймовірних токенів з сумарною ймовірністю менше параметру p (або просто k найбільш ймовірних токенів) та синтез з температурою, за яким значення

логітів діляться на значення параметру температури t. Отриманий на кінці мережі вектор логітів пропускається через softmax, після чого обирається токену з отриманого категорійного розподілу. На практиці це дозволяє досягти більшої різноманітності згенерованих виразів.

3.2 Рекурентні нейронні мережі

Розглянемо конкретні архітектури що добре придатні для реалізації авторегресивної нейронної мережі. Такі архітектури зазвичай відносяться до класу Seq2Seq і типовим прикладом архітектури з цього класу є рекурентні нейронні мережі (RNN).

Основним компонентом RNN є клітина (cell). Обробка даних RNN постає у ітеративному застосуванні однієї клітини. На відміну від найпростішого перцептрона, який має один вхідний вектор та один вихідний, між якими проходять лінійні перетворення та застосування функції активації, в клітині RNN ще вводиться поняття внутрішнього стану (hidden state). На кожній ітерації, його значення є результатом лінійних перетворень поточного вхідного вектору та попереднього внутрішнього стану з застосуванням функції активації. Поточний вихідний вектор є лінійним перетворенням поточного внутрішнього стану з застосуванням функції активації. Обчислення, що проходять ітерації t застосування клітини RNN, можна записати наступним чином:

$$h_t = \sigma_h(W_h x_t + U_t h_{t-1} + b_h)$$
$$y_t = \sigma_y\big(W_y h_t + b_y\big),$$

де x_t – вхідний вектор, y_t – вихідний вектор, h_t – вектор внутрішнього стану, σ – функція активації, W, U та b – матриці та вектор – параметри клітини RNN.

Архітектурною проблемою примітивних RNN є те що на кожній ітерації t вся інформація з попередніх ітерації доступна до клітини лише через єдиний закодований вектор внутрішнього стану h_t. Це призводить до того, що RNN

не здатна сприймати контекст великої довжини, а також до проблем під час тренування – так званих вибухаючих та затухаючих градієнтів.

Частково розв'язати зазначені проблеми була призвана LSTM (long short-term memory) архітектура RNN [3]. Її відмінністю від примітивних RNN є поява так званих воріт (gates) – векторів, якими клітина керує тим, яка інформація буде включена у поточних вихідних векторах, а яка буде запозичена з попередніх. Ці вектори також є лінійними проекціями вхідних векторів із застосуванням логістичної функції активації. Формально:

$$f_t = \sigma_g\left(W_f x_t + U_f h_{t-1} + b_f\right)$$
$$i_t = \sigma_g(W_{ih} x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$
$$\bar{c}_t = \sigma_h(W_c x_t + U_c h_{t-1} + b_c)$$
$$c_t = f_t c_{t-1} + i_t \bar{c}_t$$
$$h_t = o_t \sigma_h(c_t),$$

де x_t – вхідний вектор, h_t – вектор внутрішнього стану, σ_g – сигмоїда, σ_h та σ_c - гіперболічний тангенс, f_t – ворота забуття, i_t – ворота оновлення, o_t – ворота виходу, $\bar{c}_t$ – вектор активацій клітини, c_t – вектор стану клітини.

Подальшим розвитком LSTM стали GRU (gated recurrent unit) [4] . Відмінністю є те, що в GRU використовується на одні ворота менше – відсутні ворота виходу. Це дозволяє спростити модель без практичної втрати в виразності, що як наслідок може привести до покращення результатів та зменшення споживаних обчислювальних ресурсів.

3.3 Архітектура Трансформер

RNN є однією з найдавніших Seq2Seq архітектур, і не дивно що існують набагато більш складні та виразні моделі. Прикладом такої є архітектура Трансформер [5], моделі з якою в останні роки провели революцію в сфері обробки природної мови.

Основним компонентом Трансформерів є модуль уваги (attention) [22]. Його можна розглянути як розширення архітектури RNN.

Ключовою проблемою RNN є скритий вектор h, на який покладається складна задача послідовно накопичувати всю інформацію про контекст та раніше згенеровані токени. На відміну від RNN, клітина модулю уваги на ітерації t має доступ не тільки до попереднього скритого вектору h_{t-1}, а до всіх попередніх скритих векторів $\{h_{1..t-1}\}$. Для того щоб отримати вектор активації клітини s_t на ітерації t, всі попередні скриті вектори $\{h_{1..t-1}\}$ зважуються коефіцієнтами, кожен з яких обчислюється невеликим перцептроном a на підставі двох скритих векторів, один з яких є поточним вектором стану клітини c_t. Після цього вектор активації клітини s_t стає на місце h_t в RNN. Формально:

$$a_t = softmax(\{a(c_t, h_1), \dots, a(c_t, h_{t-1})\})$$

$$s_t = \sum_{i=1}^{t-1} a_{ti} h_i$$

І хоча модуль уваги одразу встановив нові рекорди у задачі машинного перекладу, спочатку він використовувався лише в тандемі із звичайною RNN. В архітектурі Трансформер було повністю позбавлено від RNN, а модуль уваги еволюціонував у модуль самоуваги (self-attention).

Модуль самоуваги приймає на вхід послідовність векторів $\{x_{1..t}\}$ та віддає на виході послідовність векторів $\{y_{1..t}\}$. Для кожного вектора x_t обчислюються три проекції q_t, k_t та v_t. Кожен вектор y_t є лінійною комбінацією векторів v_t з коефіцієнтами що визначаються за векторами q та k. Формально:

$$q_t = x_t W_q$$

$$k_t = x_t W_k$$

$$v_t = x_t W_v$$

$$a_t = softmax(\{q_t k_1^T, \dots, q_t k_{t-1}^T\})$$

$$y_t = \sum_{i=1}^{t-1} a_{ti} v_i,$$

де W_q, W_k та W_v – матриці параметрів, а всі вектори вважаються векторами-рядками.

Іншою особливістю модуля уваги що використовується у Трансформерах є наявність декількох голівок самоуваги. Це означає те що замість одного модуля уваги паралельно на одній і тій же самій послідовності векторів обчислюються одразу декілька, вихідні вектори яких проектуються в єдиний спільний вихідний вектор. Це дозволяє модулю мати декілька коефіцієнтів уваги a_t і звертати свою увагу одразу у декілька точок.

Описавши модуль самоуваги, перейдемо безпосередньо до архітектури Трансформера. Вона складається з двох частин: кодувальника та декодера.

Кодувальник складається з декількох однакових блоків кодувальника, розташованих один за одним. Кожен блок складається з двох підблоків - модуля уваги та невеликого перцептрона, що застосується до кожного вихідного вектору кодувальника. Паралельно з кожним підблоком проведено залишкове з'єднання (residual connection) [23] та застосовано нормалізацію шару (layer normalization) [24]. Результатом роботи кодувальника є послідовність векторів такої ж самої довжини як і послідовність вхідна.

Говорячи про кодувальник, варто звернути увагу на одну проблему модуля уваги: на відміну від рекурентних мереж, у модуль уваги ніяк не закладено розуміння послідовності токенів, він у деякому сенсі є інваріантним відносно порядку своїх вхідних даних. Це може бути проблемою для кодування тексту як послідовності токенів. Щоб уникнути цього, у вхідній послідовності векторів кодувальника до векторів ембеддінгів токенів додаються так звані позиційні ембеддінги. Вони або обчислюються на етапі передобробки, або навчаються під час тренування моделі як її параметр для кожної позиції.

Декодер Трансформеру за архітектурою є схожим з кодувальником. Головною його відмінністю є поява у кожному блоці третього підблоку – другого модуля самоуваги, вхідними послідовностями якого є як і вихідна послідовність з попереднього модуля самоуваги, так і вихідні вектори кодувальника.

У глобальному розумінні, Трансформер є Seq2Seq архітектурою, кодувальник якого виділяє з контексту ознаки високого рівня, а декодер є мовною моделлю, яка на кожному етапі генерації приймає на вхід ознаки з кодувальника та вже згенеровану частину тексту. Варто зазначити, що для того щоб зберегти можливість використання декодера у авторегресивному режимі, то через позиційну індиферентність модуля уваги під час тренування потрібно застосувати маскування до модуля самоуваги – покласти коефіцієнти уваги на вектори після поточного рівними нулю так, щоб модуль уваги не «підглядав наперед».

3.4 Приклади Трансформерів

Бум розвитку глибокого навчання стався у 2012 році з виходом роботи по AlexNet – нейронній мережі, яка значно випереджала всі інші підходи класифікації зображень ImageNet [25]. У той же час було помічено що для широкого спектру задач машинного зору дуже ефективною є техніка претренування нейронної мережі на ImageNet. Це пояснюється тим що знання які отримує мережа після тренування на класифікації зображень часто є схожими на знання необхідні для розв’язання інших задач машинного зору.

Упродовж довгого часу були пошуки схожої на ImageNet задачі для претренування мовних моделей. Прорив стався з приходом GPT-2 [26]. Тоді було помічено що при претренуванні мовної моделі на величезному корпусі тексту, що складається з мільйонів веб-сторінок, відбувається так званий переніс знань (transfer knowledge) з моделювання мови на інші задачі обробки природньої мови, як переклад, відповіді на запитання та узагальнення.

Важливим фактором відіграв і розмір моделей – кількість параметрів перевищувала 1.5 мільярди. Останні наробки в цьому напрямку включають в себе нейронні мережі – GPT-3 – з кількістю параметрів 175 мільярдів [27].

Архітектурно GPT-2 є мовною моделлю і являє собою декодер Трансформера. Іншим прикладом перетренованої на натуральній мові моделлю, що гарно демонструє свої властивості з переносу знань, є BERT [28]. На відміну від GPT, BERT являє собою кодувальник Трансформера, не може бути використаним як мовна модель напряму і погано підходить для генерації природної мови. Проте, він краще здатен до задач розуміння природної мови, таких як класифікація тексту.

РОЗДІЛ 4. ПОЄДНАННЯ ГЕНЕРАТИВНИХ МОДЕЛЕЙ ДЛЯ КЕРОВАНОЇ ГЕНЕРАЦІЇ ДІАЛОГУ

4.1 Генерація діалогу

Сформулюємо задачу генерації діалогу наступним чином: за заданою історією діалогу між двома співрозмовниками, згенерувати наступний вислів – відповідь у цьому діалозі. В цьому розділі будемо розглядати найпростіший випадок коли довжина історії дорівнює 1, тобто контекст який подається на початку генерації на модель складається лише з одного попереднього виразу співрозмовника. Довжина історії є гіперпараметром і суттєво на міркування та реалізацію не впливає, проте її зменшення дозволяє зекономити обчислювальні ресурси, пришвидшивши тренування та вкластись в обмеження на пам’ять.

Мовну модель в такому діалоговому сценарії можна уявити як $F_\theta(x_t|x_{<t}, x')$, де x' - попередній вираз від співрозмовника з історії діалогу, а x – нова відповідь, що генерується.

Форматом даних з яким працює діалогова модель є послідовність токенів, яка починається з токену початку послідовності $< bos >$. За ним йдуть токен першого співрозмовника $< speaker1 >$ та токени його вислову, потім токен другого співрозмовника $< speaker2 >$ та токени його вислову. Закінчується послідовність токенів токеном кінця послідовності $< eos >$. Тоді маючи мовну модель натреновану на такому форматі даних, для генерації діалогової відповіді буде потрібно подати послідовність токенів у цьому форматі до токену $< speaker2 >$ включно і генерувати моделлю нові токени доки не буде згенеровано токен $< eos >$ (або досягнуто максимальної довжини послідовності).

Окрім ембеддінгів токенів та позиційних ембеддінгів, до вхідного вектору мовної моделі також додаються ембеддінги типів токенів як в TransferTransfo [29]. Ці вектори допомагають модулю уваги розрізняти між токенами що відповідають вислову співбесідника та токенами згенерованими моделлю.

4.2 Постановка проблеми

Генеративні Трансформери такі як GPT-2 досягають передових результатів на майже усіх задачах генерації тексту, у тому числі й на задачі генерації діалогів. Не дивлячись на високу якість згенерованого тексту, досі можливості з керування згенерованим текстом були дуже обмеженими.

Маючи натреновану мовну модель що апроксимує розподіл $p(x_t|x_{<t})$ є різні способи обрання наступного токену з цього розподілу, як вже розглянутий ядерний синтез що перед вибіркою з категорійного розподілу модифікує його. Такі штучні модифікації розподілу й лежать в основі методів локального керування генерацію тексту мовною моделлю. За цих підходів, на кожній ітерації генерації окремо модифікується ймовірнісний розподіл за наступними токенами. Таким чином можна наприклад зменшувати до нуля ймовірності небажаних токенів або комбінації токенів (N-грам), тим самим

блокуючи їх. Можна навпаки трохи збільшувати ймовірності окремих ключових слів, тим чином спрямовуючи генерації в потрібному напрямку. Очевидним плюсом таких підходів є простота реалізації, проте нажаль зазвичай за таких штучних модифікацій отриманих розподілів страждає загальна якість згенерованого тексту.

В протилежність методам локального керування можна виділити методи глобального керування що впливають на отриманий розподіл окремого токену не постфактум, а змінюють обчислення в мережі ще до останнього шару, впливаючи тим самим на кінцеві розподіли більш природньо. Способом реалізації глобального керування є апроксимація умовного розподілу $p(x|a)$ замість звичайного $p(x)$, де a – додаткова інформація що описує x. Ця інформація може бути наприклад у вигляді категорії емоції чи теми вислову, або дійсного числа що описує позитивність, сентиментальність та інші риси вислову. Як і раніше, $p(x|a)$ апроксимується авторегресивною мовною моделлю F_θ, на вхід якої тепер ще подається додаткова інформація a:

$$p(x_t|x_{<t}, x', a) \approx F_\theta(x_t|x_{<t}, x', a)$$

Такий підхід зазвичай вимагає на етапі тренування моделі розміченого набору даних з пар текстів x та додаткової інформації a про них. Такі розмічені дані зазвичай складно отримати, і ця проблема є ще більш гострою у діалоговому сценарії, адже розмітки потребує кожен вислів в діалозі, а сучасні Трансформери є великими мережами і для оптимальних результатів потребують великої кількості тренувальних даних. Через цю проблему бажаними є підходи навчання без вчителя.

4.3 Опис підходу

Нарешті опишемо повну запропоновану систему для глобального керування генерацією діалогу без вчителя – додаткової розмітки даних.

Система є поєднанням двох типів генеративних моделей – авторегресивних та варіаційних автокодувальників. Від авторегресивних

моделей система має мовну модель – генератор тексту G_ω. Від варіаційних автокодувальників система має окрім генератора (декодувальника) ще й кодувальник F_θ, який приєднано до декодувальника за допомогою регуляризованого латентного простору z, як зазвичай і робиться в варіаційних автокодувальниках.

Як і раніше, для глобального керування на вхід мовної моделі подається додаткова інформація, проте замість a з розміченого набору даних ця інформація буде у вигляді латентного коду z. Тоді генератор має вигляд $G_\omega(x_t|x_{<t}, x', z)$. Кодувальник як і раніше має вигляд $F_\theta(z|x)$, тобто він обчислює параметри розподілу латентного коду z лише за останнім, другим виразом з історії x. Слід зазначити що ця система вже строго кажучи не є автокодувальником, адже вхідна послідовність токенів на кодувальник x не дорівнює послідовності токенів з якою працює декодувальник і яка складається з конкатенації x та попереднього виразу x'.

Така структура системи мотивована тим, що до виразу-відповіді x застосовується принцип шийки пляшки у вигляді латентного коду z. Справді, якщо замаскувати значення функції втрат генератора на елементах x' (покласти їх рівними нулю), тим самим вимагаючи від нього лише вміння генерувати x, генератор буде вчитись генерувати x за всіма поданими на нього даними – попереднім виразом x' та латентним кодом z. Оскільки z обчислюється кодувальником на основі x, яку потім і потрібно відновити генератором, в ході тренування в z буде накопичуватись якомога більше інформації про x. А оскільки до простору z застосовується KL-регуляризація, як зазвичай у VAE, і якщо розмірність z покласти достатньо малою, простір z буде містити структуровану високорівневу інформацію про x, таку як, можливо, емоційний окрас та іншу. Наявність додаткового вхідного вектора у генераторі з такого простору і відкриває можливості з глобально керованої генерації тексту.

У якості архітектури для генератора та кодувальника оберемо класичні Трансформери для генерації та класифікації – GPT-2 та BERT. Обидві моделі візьмемо в найменших стандартних варіантах з 117 мільйонами та 110 мільйонами відповідно (12 шарів, 12 голівок уваги, внутрішня розмірність 768). Такі розміри є оптимальними по складності та вимогам до обчислювальних ресурсів. Так як вхідні дані обох моделей зазвичай є текстовими, відкритим залишається питання щодо того як поєднати ці дві архітектури за допомогою чисельного вектору z.

4.4 Порівняння з іншими роботами

Хоча ця робота і є першою що поєднує VAE та Трансформери для керування генерацією діалогу, вона не є першою що поєднує VAE та Трансформери взагалі. Однією з перших в цьому є модель Optimus [30], роботу з якої було опубліковано наприкінці 2020 року. І хоча мотивація та застосування між двома роботами відрізняються, в Optimus так само є два Трансформери – GPT-2 та BERT, що поєднано у VAE за допомогою латентного коду z. Отже, деякі деталі запропонованого підходу можуть бути взяті звідти.

Так, для поєднання генератора та кодувальника в Optimus запропоновано дві інтеграційні схеми. В першій, лінійно трансформований вектор z додається до ембеддінгу кожного вхідного токену генератора. В другій, лінійно трансформований вектор z додається як додатковий x_0 на кожному модулі самоуваги генератора. Таким чином кожен з шарів уваги здатен напряму звертати свою увагу на вектор z. Автори Optimus зазначають, що поєднання двох схем дає на практиці найкращі результати.

Іншим корисним прийомом в Optimus є спосіб подолання проблеми KL зникнення (vanishing). Ця проблема є частою в варіаційних автокодувальниках і вона полягає у тому що під час тренування KL-компонента функції втрат стає нульовою – кодувальник генерує розподіл z що повністю з стандартним

нормальним і декодувальник стає повністю ігнорувати латентний код z. Для подолання цього автори Optimus пропонують починати тренування моделі без KL-розходження у функції втрат, що еквівалентно звичайному, не варіаційному, автокодувальнику. Так, без регуляризації на простір z, полегшується корисне використання z генератором на початкових етапах тренування. Вже потім, з середини тренування, починається введення KL-компоненти функції втрат з коефіцієнтом що лінійно зростає від 0 до 1 на 75% тренування. Останні 25% тренування відповідають повноцінному тренуванню варіаційного автокодувальника з двома компонентами у функції втрат.

Ці два прийому разом з початковими вагами кодувальника та генератора претренованими на великому корпусі тексту з англомовної Вікіпедії було взяті такими самими як в Optimus. Ключовими відмінностями Optimus від цієї роботи є те, що Optimus являє собою повноцінний автокодувальник, в якому вхідні дані кодувальника та дані з якими працює генератор є однаковими і окрім z генератор ніякої вхідної інформації немає. Через це z має зберігати всю інформацію про вхідні та вихідні дані та мати велику розмірність. В той же час, ця робота скоріше є розширенням класичної діалогової моделі TransferTransfo, в якій на генератор подається окрім історії діалогу в спеціальному форматі ще й вектор z з кодувальника, розмірність якого покладається якомога меншою для того щоб він містив лише високорівневу інформацію про вихідний вираз-відповідь, для досягання керованої генерації діалогу.

4.5 Постановка експерименту

Опишемо деталі реалізації експерименту з тренування генеративних нейронних мереж для керованої генерації діалогу за описаним підходом.

У якості тренувальної вибірки даних візьмемо комбінацію з трьох відкритих англомовних діалогових наборів даних – EmpatheticDialogues [31], Persona-Chat [32] та DailyDialog [33]. Їх сумарний розмір складає майже 48

тисяч діалогів та майже 349 тисяч окремих діалогових виразів. І хоча в цих наборах даних окрім діалогів є ще додаткові розмітки (такі як персони в Persona-Chat), для експерименту візьмемо лише безпосередньо діалоги як послідовності виразів без будь-якої додаткової інформації – так як описаний підхід є навчанням без вчителя і вимагає лише нерозмічених даних. Також відведемо 10% цього набору даних для тестової підвибірки – вона буде потрібна для перевірки моделі на відсутність перенавчання – на те, що вона не почала запам’ятовувати тренувальну вибірку цілком, а буде добре узагальнюватись на прикладах які раніше не бачила.

Шляхом підбору упродовж багатьох тестових запусків експерименту, знайдемо значення для гіперпараметрів. Так, оптимальною розмірністю латентного вектора z було обрано 8. Інтуїтивно це є недостатньо великим значенням для повного кодування виразу-відповіді, проте достатньо великим для високорівневого опису загальних характеристик виразу. Значення деяких інших гіперпараметрів склали: 2 для кількості тренувальних епох, 8 для розміру міні-пакету та $5*10^{-5}$ для початкового значення кроку градієнтного спуску, що з ходом тренування лінійно зменшувався до нуля. Максимальну довжину послідовності покладемо рівною 128 токенів, притому приберемо обмеження на довжину історії у кількості виразів – на контекст генератора будуть подаватися всі виразі з історії діалогу, з яких будуть обиратися лише 128 останніх токенів.

У якості середи для програмної реалізації було обрано операційну систему з відкритим вихідним кодом Ubuntu 20.04 на ядрі Linux 5.4 та систему контейнеризації Docker 20.10.2, прикладний програмний інтерфейс для обчислень загального призначення на графічних процесорах CUDA 10.0, бібліотека примітивів глибокого навчання cuDNN 7 та мова загального призначення Python 3.8. У якості фреймворку глибокого навчання було обрано PyTorch 1.4.0, у якості основи реалізації архітектур Трансформерів було використано бібліотеку Transformers [6]. Основним інструментом розробки була обрана інтегрована середа розробки PyCharm від JetBrains з

відкритим вихідним кодом. Також було використано систему контроля версій Git.

Всі запуски проводились на ПК з відеокартою NVIDIA GeForce RTX 2070 SUPER з 8 ГБ відео пам’яті. Одночасне тренування двох Трансформерів, GPT-2 та BERT, за такого обмеження по пам’яті та за обраних гіперпараметрів можливе лише з розміром міні-пакету 4. Для того щоб досягти ефективного розміру міні- пакету 8, було використано так звану акумуляцію градієнтів. Цей підхід полягає у тому, що оновлення параметрів відбувається лише після кожних n кроків з невеликим міні-пакетом m, упродовж яких градієнти рахуються, накопичуються в буфері через операцію суми але не використовуються для оновлення одразу. Це дозволяє досягти ефективного розміру міні-пакету $m * n$. В даному випадку, $m = 4, n = 2$.

4.6 Результати експерименту

Спочатку продемонструємо показники з одного з запусків експерименту у Tensorboard. За віссю абсцис на всіх графіках – номер кроку оптимізації з початку тренування. Одна епоха дорівнювала приблизно 14 тисячам кроків.

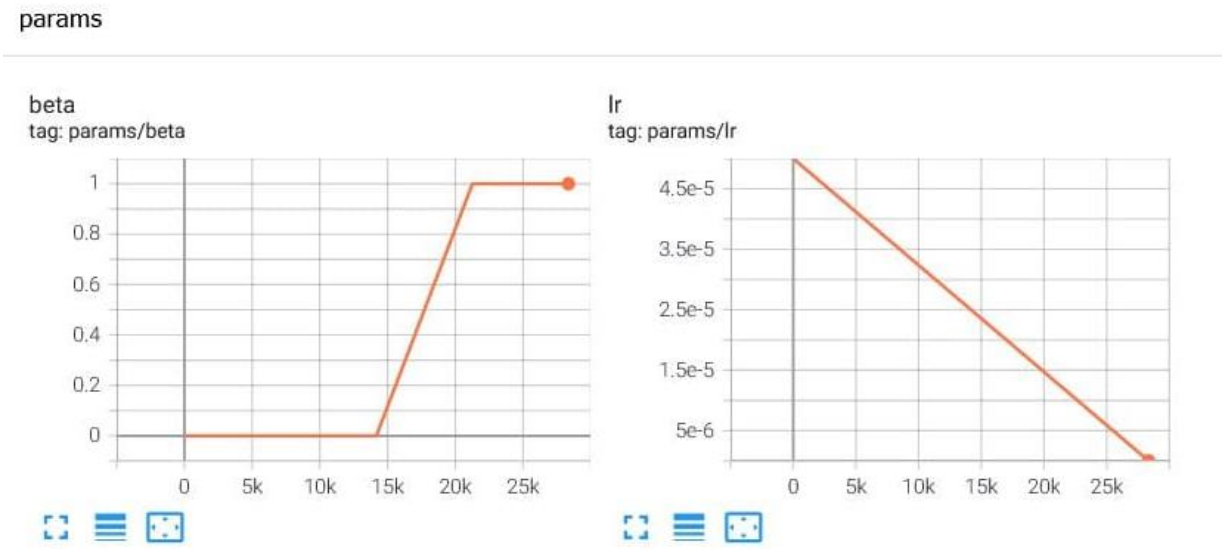

Рисунок 1. Значення гіперпараметрів тренування, що змінювались з часом. Зліва направо: коефіцієнт при KL-компоненті функції втрат та кроку градієнтного спуску.

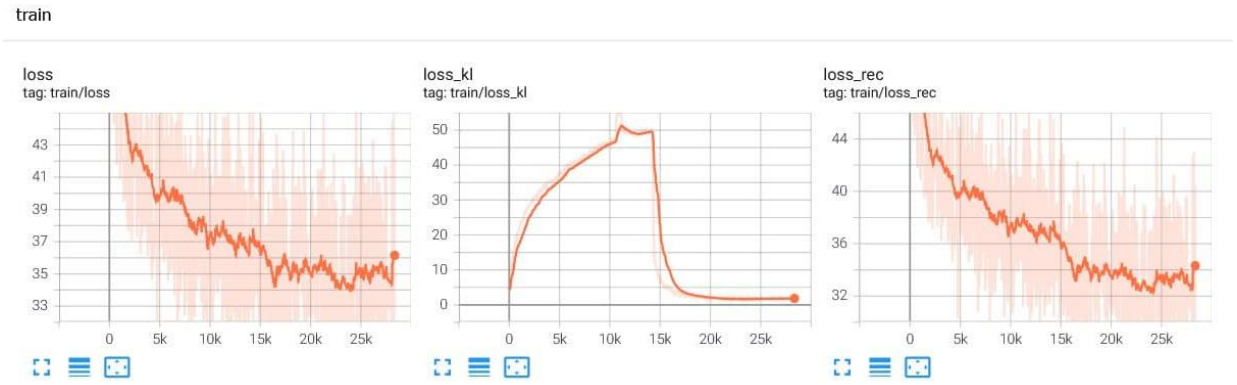

Рисунок 2. Значення функцій втрат на тренувальній вибірці. Зліва направо: загальне значення функції втрат, KL-компонента функції втрат та помилка реконструкції.

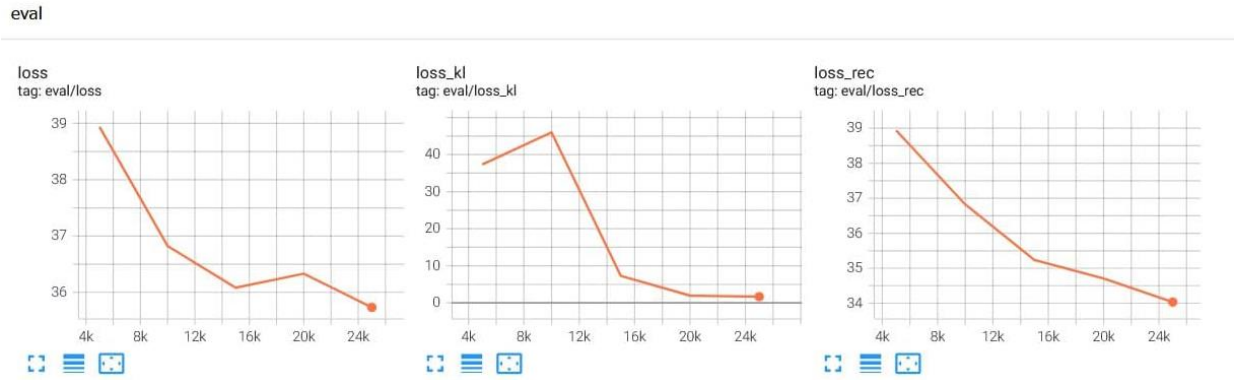

Рисунок 3. Значення функцій втрат на тестовій вибірці.

На графіках можна побачити як з ходом тренування поступово зменшувалось значення помилки реконструкції діалогової відповіді, що свідчить проте що модель дійсно вчилась генерувати текст. Поки у KL-компоненти функції втрат був нульовий коефіцієнт, її значення поступово зростало, що свідчило про віддалення розподілу значень z від нормального, про появу складної структури в просторі латентної змінної. Проте як тільки додавалось до функції втрат KL-розходження з ненульовим коефіцієнтом, його значення різко падало до дуже малих, що говорить про швидку появу простої структури в латентному просторі, про те що z має розподіл близький до нормального. Також можна побачити що на тестовій вибірці значення функції втрат є близькими до тренувальних, що свідчить про відсутність перенавчання нейронної мережі.

Також наглядно продемонструємо можливості моделі. Для цього запропонуємо два режими. В обох режимах зафіксуємо історію діалогу і будемо працювати лише з останнім виразом.

В першому режимі оцінимо якість роботи кодувальника через порівняння векторів z отриманих з різних виразів-відповідей. Для цього, задамо набір з виразів-прикладів та обчислимо для кожного вектор z. Також, згрупуємо цей набір на піднабори зі схожими виразами та обчислимо для кожного піднабору його вектор z' як середнє векторів z виразів що входять до цього набору. Потім, для декількох тестових виразів обчислимо їх вектори z і знайдемо вирази-приклади та піднабори з найближчими векторами. Для міри близькості двох векторів будемо використовувати косинус подібності.

Таблиця 1. Набір виразів-прикладів

Номер піднабору	Вираз-приклад
1	Hello!
1	Hello, how are you doing?
1	Hi! Nice to see you!
1	Hello, my name is Kyle.
1	Hello, what's up?
2	Please don't do that.
2	No! Stop!
2	You better stop.
3	This is amazing!
3	That's the best thing I saw today.
3	Thanks, I love it!
4	I don't like it
4	This is terrible
4	It's the worst!
5	I don't want to tell you
5	I don't remember
5	Can't say right now
6	I don't want to talk to you anymore
6	Leave me alone
6	Wish we never met

Таблиця 2. Тестові вирази та найближчі до них приклади

Тестовий вираз	Найближчі приклади		Найближчий піднабір	
	Вираз	Відстань	Піднабір	Відстань
Hi!	Hello!	0.989	1 (Hello!, …)	0.912
	Leave me alone	0.873	2 (Please don't do that, …)	0.867
	No! Stop!	0.873	3 (This is amazing!...)	0.785
That’s bad.	This is terrible	0.833	4 (I don't like it…)	0.866
	I don’t like it	0.700	2 (Please don't do that, …)	0.665
	This is amazing!	0.698	3 (This is amazing!...)	0.660
This idea is the best!	This is amazing!	0.824	3 (This is amazing!...)	0.883
	That’s the best thing I saw today.	0.796	4 (I don't like it…)	0.626
	It’s the worst!	0.782	1 (Hello!, …)	0.528
I will not answer that.	I don't want to tell you	0.934	6 (I don't want to talk to you...)	0.935
	Please don't do that.	0.925	5 (I don't want to tell you...)	0.934
	I don't remember	0.856	2 (Please don't do that....)	0.896
Stay away from me!	Leave me alone	0.867	2 (Please don't do that....)	0.767
	You better stop.	0.836	6 (I don't want to talk to you...)	0.716
	Hello!	0.826	1 (Hello!...)	0.660

Як можемо бачити, в усіх тестових виразів найбільш близькі у просторі z прикладі та піднабіри дійсно є найбільш на них схожими, що говорить про те що в просторі z дійсно зберігається високорівнева інформація о виразах, їх «суть». Проте є й нелогічні випадки. Так, наприклад приклад «This is amazing!» є ближчим до тестового виразу «That’s bad.» ніж приклад «It's the worst!». Це відображає проблему підходу: через підхід без вчителя, модель іноді не розуміє яку саме «суть» їх зберігати в z. Хоча людям й зрозуміло, що вираз з різко позитивною реакцією «This is amazing!» повинен бути якомога далі в латентному просторі від виразів з різко негативними реакціями «That’s bad.», «It's the worst!», модель, схоже, всі виразі з різкими реакціями (як позитивними, так і негативними) кодує у близькі вектори z.

В другому режимі оцінимо якість роботи генератора через оцінку принципової схожості згенерованих виразів на вирази, за якими було обчислено вектор z. Для цього, задамо набір тестових виразів, обчислимо для

кожного вектори z та обчислимо їх середній вектор z'. Після цього подамо історію діалогу та вектор z' на генератор та згенеруємо декілька виразів-відповідей.

Таблиця 3. Історія діалогу

Співрозмовник	Вираз
Система	What do you think about hiking?
Користувач	I love it! I think I'm going to do it tomorrow

Таблиця 4. Набори тестових виразів та згенеровані відповіді

Набір тестових виразів	Згенеровані відповіді
Hello! Hello, how are you doing? Hi! Nice to see you! Hello, my name is Kyle. Hello, what's up?	Cool. Will it be Friday? Hope so! Where you going OH cool! How long? Well done. Where was it? Well done! What day Awesome! Can you hike? Good luck! May I see? Cool! When did you do it? Good idea! Where? Really? Congratulations on it!
Please don't do that. No! Stop! You better stop.	Hopefully you won't disappoint. I can't believe it. You haven't! You said you don't. I'm sorry! Ok. I heard it! I hope no one sees. I hope you're safe I can't do that! I hope you aren't!
This is amazing! That's the best thing I saw today. Thanks, I love it!	It's the best! Being outside is great! Yeah it is awesome! That sounds amazing! I bet it is beautiful!

	That sounds fun!! Oh I love it! Thats great, opens eyes. It's going to be amazing! IT is surreal!
I don't like it This is terrible It's the worst!	You sound kinda nervous Yeah rock climbing sucks That is super sad tho It does stink. I think it's dangerous But hiking is boring I completely hate it. It is nothing special It is never easy It isn't so nice!
I don't want to tell you I don't remember Can't say right now	I don't want to You haven't done that I don't know how! I didn't expect that I can't decide I can't tell you I can't decide if I want I don't think you'll be able I haven't heard what it is You don't need it

Як можемо бачити, для кожного набору тестових виразів генеруються в цілому схожі відповіді. Так, для першого набору що складається з виразів привітання згенеровані відповіді мають позитивний настрій, а для другого, навпаки – негативний. Для останнього набору, наприклад, що складається з виразів що виражають невпевненість, згенеровані відповіді так само є в цілому невпевненими. Також слід зазначити, що більшість виразів добре виглядають саме в контексті діалогу.

І хоча в цілому якість згенерованих виразів на перший погляд є достатньою і залежність від поданого вектору z є виразною, все одно є багато

випадків коли хоча б одна з цих двох характеристик не виконується. Так, для прикладу, у другому наборі згенеровані вирази «You haven't!» та «I hope you aren't!» не є граматично коректними в контексті діалогу, а в виразах «Hopefully you won't disappoint.» та «Ok. I heard it!» не явного впливу виразів з тестового набору. Це відображає другу проблему підходу: як часто буває в навчанні що переслідує одразу дві цілі, часто бувають випадки коли хоча б одна з цих цілей не досягається. В даному випадку, такими цілями є генерація виразу в якому була би добре врахована подана історія діалогу та в той же час генерація виразу в якому були б добре врахований поданий вектор z.

Нарешті, покажемо роботу моделі в другому режимі з одним виразом в тестовому наборі, тобто тепер вектор z просто рахується з одного виразу та напряму подається на генератор, разом з історією.

Таблиця 5. Тестові вирази та згенеровані відповіді

Тестовий вираз	Згенерована відповідь
I heard that hiking is bad for your health	I am the pride of your heart! I can either imagine hiking sick with cancer. If you hike unhealthy you will become sick. It seems your health suffers from it. That was unexpected for you parents. You seem like an insecure beginner! A friend told me that the best way is hiking. Stay out of harm's way. Life disappoints Cheers for your dedication, that matters With no hiking would you be annoying!
What will you do if something happens along the way?	What are you thinking at the moment about it? What do you think is the next step for you to take? What does it take to get there, what are you thinking of hiking What did you think about taking it to the lake? What do you think is going to happen next if you finish hiking? What do you think will happen if you hike ahh? What do you think of when you get back from the trail? What do you think you are going to do after this hike?

	What do you think will happen during the trip next time? How long would you like to hike it if you didn't mind?
I don't really like hiking but I would love to go with you	I don't mind a good day when I'm not hiking. I've never done it that much but it makes me think about life. I know it could be a bit more expensive but I am sure of it. I do too but I think I can be a bit of a hiker myself Yeah I don't really like it so I want to hiking in a different part of the country I've been thinking about more, but never done it. I don't really do it everyday though. I feel like living it out I've always wanted to are pretty cool at least, but I may be lucky I've been doing it for a good reason, but it's not good for me I know, but there is a lot of work to do right now
I like dancing more. Would you like to join me?	That's a must for me. Make a good trip tomorrow? You must love it. Do you hike often? It's good to have good weather. Where do you hike? I do too! It is a fun experience for you. All I do is enjoy the freedom. How about you? Yeah. I enjoy hunting on that specific date. I love it too! Do you play sports? I will do everything. Peak climbing for fun! It can be fun. Have a go do it on your own. That sounds fun. You should try out some hiking trails

Як можемо бачити, знов простежується деяка залежність між виразами на яких обчислювався вектор *z* та виразами які було згенеровано з використанням цього вектору. Так, у першому прикладі, половина згенерованих виразів щось згадують пов’язане зі здоров’ям, а у другому всі вирази є питаннями що починаються з «What», так само як і тестовий вираз. В третьому та четвертому прикладах згенеровані вирази складаються з двох частин: в третьому ці частини часто розділені «but», а в четвертому дві частини є окремими реченнями – знову, так само як і тестові вирази. Проте все ще як і якість згенерованих виразів в контексті діалогу, так і залежність між поданим виразом та згенерованим, залишають бажати набагато більшого.

У підсумках з результатів експерименту з тренування генеративних нейронних мереж для керованої генерації діалогу можна сказати що отримана модель дійсно показує багатообіцяючі результати, генеруючи структурно коректні діалогові відповіді з залежністю від поданого на вхід вектору z.

Проте, нажаль, присутня низка проблем. Деякі згенеровані вирази не є одночасно і коректними в контексті діалогу, і прямо залежними від поданого вектору. Також, якщо залежність від вектору z і спостерігається, вона не завжди є такою як очікувалось, через підхід з навчанням без вчителя. Іноді високорівнева інформація що зберігається в z описує не емоцію чи настрій виразу, наприклад, а структурні особливості, як перше слово чи наявність двох речень. Іншою фундаментальною проблемою притаманною для всіх автокодувальників є складність роботи з даними різного розміру, в даному випадку з виразами різної довжини, адже дані різних розмірів всі кодуються в один вектор фіксованої довжини. Прояв цієї проблеми можна побачити порівнявши рівень впливу тестових виразів на згенеровані відповіді у таблицях 4 та 5.

ВИСНОВКИ

В даній роботі було викладено теоретичні основи глибокого навчання та генеративних мереж, зокрема трьох основних типів – варіаційних автокодувальників, генеративних змагальних мереж та авторегресивних нейронних мереж. Окрему увагу було приділено опису прикладу авторегресивних моделей – мовним моделям, з архітектурою заснованою на рекурентних нейронних мережах та на Трансформерах. Було поставлено задачу генерації діалогу та раніше не розв'язану проблему глобально керованої генерації діалогу. Було запропоновано новий підхід з застосування поєднання двох типів генеративних нейронних мереж – варіаційних автокодувальників та авторегресивних – для розв'язання поставленої проблеми. Підхід запропонований в цій роботі було порівняно з іншою новою та схожою роботою. Експеримент з втілення запропонованого підходу реалізовано. Результати експерименту було відображено у вигляді графіків і таблиць та проаналізовано. На підставі аналізу результатів підкреслено основні проблеми пов'язані з запропонованим підходом. Ця робота є першою що пропонує підхід глобального керування генерацією тексту без вчителя з використанням передових нейронних мереж – Трансформерів. Розв'язання зазначених проблем підходу є підставою для подальших досліджень в цьому напрямку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

[1]. Doersch, Carl. Tutorial on variational autoencoders. / Doersch, Carl. – 2016. – 23 с. – (Препринт / arXiv:1606.05908).

[2]. Goodfellow, Ian J. Generative adversarial networks. / Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, та Yoshua Bengio. – 2014. – 9 с. – (Препринт / arXiv:1406.2661).

[3]. Hochreiter, Sepp. Long short-term memory. / Hochreiter, Sepp, Jürgen Schmidhuber. – 1997. – 32 с. – (Neural computation 9, no. 8: 1735-1780).

[4]. Cho, Kyunghyun. Learning phrase representations using RNN encoder-decoder for statistical machine translation. / Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, та Yoshua Bengio. – 2014. – 11 с. – (Препринт / arXiv:1406.1078).

[5]. Vaswani, Ashish. Attention is all you need. / Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, та Illia Polosukhin. – 2017. – 11 с. – (Препринт / arXiv:1706.03762).

[6]. Wolf, Thomas. HuggingFace's Transformers: State-of-the-art natural language processing. / Wolf, Thomas, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac et al. – 2019. – 8 с. – (Препринт / arXiv:1910.03771).

[7]. Rosenblatt, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. / Rosenblatt, Frank. – 1958. – 23 с. – (Psychological review 65, no. 6: 386).

[8]. Nair, Vinod. Rectified linear units improve restricted boltzmann machines. / Nair, Vinod, та Geoffrey E. Hinton. – 2010. – 8 с. – (конф. Icml’2010)

[9]. Maas, Andrew L. Rectifier nonlinearities improve neural network acoustic models. / Maas, Andrew L., Awni Y. Hannun, та Andrew Y. Ng. – 2013. – 6 с. – (In Proc. icml, vol. 30, no. 1, p. 3. 2013).

[10]. Rumelhart, David E. Learning representations by back-propagating errors. / Rumelhart, David E., Geoffrey E. Hinton, та Ronald J. Williams. – 1986. – (nature 323, no. 6088 (1986): 533-536).

[11]. Deep learning. Vol. 1 / Goodfellow, Ian, Yoshua Bengio, Aaron Courville, та Yoshua Bengio. – Cambridge: MIT press, 2016 – 802 с.

[12]. Paszke, Adam. Pytorch: An imperative style, high-performance deep learning library. / Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen та ін. – 2019. – 12 с. – (Препринт / arXiv:1912.01703).

[13]. Abadi, Martín. Tensorflow: A system for large-scale machine learning. / Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin та ін. – 2016. – 18 с. – (In 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), с. 265-283).

[14]. Qian, Ning. On the momentum term in gradient descent learning algorithms. / Qian, Ning. – 1999. – 14 с. – (Neural networks 12, no. 1 (1999): с. 145-151).

[15]. Kingma, Diederik P. Adam: A method for stochastic optimization. / Kingma, Diederik P., та Jimmy Ba. – 2014. – 15 с. – (Препринт / arXiv:1412.6980).

[16]. Zeiler, Matthew D. Adadelta: an adaptive learning rate method. / Zeiler, Matthew D. – 2012. – 6 с. (arXiv preprint arXiv:1212.5701).

[17]. Arjovsky, Martin. Wasserstein generative adversarial networks. / Arjovsky, Martin, Soumith Chintala, та Léon Bottou. – 2017. – 10 с. – (In International conference on machine learning, pp. 214-223. PMLR).

[18]. Gulrajani, Ishaan. Improved training of wasserstein gans. / Gulrajani, Ishaan, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin та Aaron Courville – 2017. – 20 с. – (Препринт / arXiv:1704.00028).

[19]. Shen, Jonathan. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. / Shen, Jonathan, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen та ін. – 2018. – 5 с. – (In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4779-4783. IEEE, 2018).

[20]. Oord, Aaron van den. Wavenet: A generative model for raw audio. / Oord, Aaron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, та Koray Kavukcuoglu – 2016. – 15 с. – (Препринт / arXiv:1609.03499).

[21]. Holtzman, Ari. The curious case of neural text degeneration. / Holtzman, Ari, Jan Buys, Li Du, Maxwell Forbes, та Yejin Choi. – 2019. – 16 с. – (Препринт / arXiv:1904.09751).

[22]. Bahdanau, Dzmitry. Neural machine translation by jointly learning to align and translate. / Bahdanau, Dzmitry, Kyunghyun Cho, та Yoshua Bengio. – 2014. – 15 с. – (Препринт / arXiv:1409.0473).

[23]. He, Kaiming. Deep residual learning for image recognition. / He, Kaiming, Xiangyu Zhang, Shaoqing Ren, та Jian Sun. – 2016. – 9 с. – (In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016).

[24]. Ba, Jimmy Lei. Layer normalization. / Ba, Jimmy Lei, Jamie Ryan Kiros, та Geoffrey E. Hinton. – 2016 – 14 с. – (Препринт / arXiv:1607.06450).

[25]. Krizhevsky, Alex. Imagenet classification with deep convolutional neural networks. / Krizhevsky, Alex, Ilya Sutskever, та Geoffrey E. Hinton. – 2012. – 9 с. – (Advances in neural information processing systems 25 (2012): 1097-1105).

[26]. Radford, Alec. Language models are unsupervised multitask learners. / Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, та Ilya Sutskever. – 2019. – 24 с. – (OpenAI blog 1, no. 8 (2019): 9).

[27]. Brown, Tom B. Language models are few-shot learners. / Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan та ін. – 2020. – 25 с. – (Препринт / arXiv:2005.14165).

[28]. Devlin, Jacob. Bert: Pre-training of deep bidirectional transformers for language understanding. / Devlin, Jacob, Ming-Wei Chang, Kenton Lee, та Kristina Toutanova. – 2018. – 16 с. – (Препринт/ arXiv:1810.04805).

[29]. Wolf, Thomas. Transfertransfo: A transfer learning approach for neural network based conversational agents. / Wolf, Thomas, Victor Sanh, Julien Chaumond, та Clement Delangue. – 2019. – 6 с. – (Препринт / arXiv:1901.08149).

[30]. Li, Chunyuan. Optimus: Organizing sentences via pre-trained modeling of a latent space. / Li, Chunyuan, Xiang Gao, Yuan Li, Baolin Peng, Xiujun Li, Yizhe Zhang, та Jianfeng Gao. – 2020. – 22 с. – (Препринт / arXiv:2004.04092).

[31]. Rashkin, Hannah. Towards empathetic open-domain conversation models: A new benchmark and dataset. / Rashkin, Hannah, Eric Michael Smith, Margaret Li, та Y-Lan Boureau. – 2018. – 12 с. – (Препринт / arXiv:1811.00207).

[32]. Zhang, Saizheng. Personalizing dialogue agents: I have a dog, do you have pets too?. / Zhang, Saizheng, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, та Jason Weston. – 2018. – 16 с. – (Препринт / arXiv:1801.07243).

[33]. Li, Yanran. Dailydialog: A manually labelled multi-turn dialogue dataset. / Li, Yanran, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, та Shuzi Niu. – 2017. – 10 с. – (Препринт / arXiv:1710.03957).