НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп’ютерні системи та мережі»
спеціальності 123 «Комп’ютерна інженерія»
на тему: «Система генерації музики за допомогою рекурентних нейронних мереж»

Виконав : студент __4__ курсу, групи _ІВ-81_
(шифр групи)

_____Ковальков Дмитро Павлович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник ___асистент Кочура Юрій Петрович_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант ___(нормоконтроль) професор, д.т.н., Сімоненко В. П._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2022 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Ковалькова Дмитра Павловича

1. Тема проєкту *Система генерації музики за допомогою рекурентних нейронних мереж,* керівник проєкту Кочура Юрій Петрович, асистент, затверджені наказом по університету

2. Термін здачі студентом закінченого проєкту *11 червня 2022 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Аналіз предметної області та існуючих рішень за темою дипломного проекту.

Розділ 2. Огляд сильних та слабких сторін існуючих підходів.

Розділ 3. Реалізація системи генерації музики за допомогою рекурентних нейронних мереж.

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень) структурна схема системи, функціональна схема (обчислювальний граф моделі), схема алгоритму.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В. П.		

7. Дата видачі завдання «2» травня 2022 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	10.12.2021-18.01.2022	
2.	*Вивчення та аналіз завдання*	02.05.2022-13.05.2022	
3.	*Програмна реалізація системи*	16.05.2022-11.06.2022	
4.	*Оформлення пояснювальної записки*	02.05.2022-11.06.2022	
5.	*Захист програмного продукту*		
6.	*Передзахист*		
7.	*Захист*	*23.06.2022*	

Студент _____ Дмитро КОВАЛЬКОВ
(підпис)

Керівник _____ Юрій КОЧУРА
(підпис)

АНОТАЦІЯ

Метою даної дипломної роботи є створення системи генерації музичних творів за допомогою рекурентних нейронних мереж.

Для реалізації проєкту буде використана мова програмування Python та програмне середовище PyCharm.

ANNOTATION

The purpose of this thesis is to create a system of generation of musical works using recurrent neural networks.

The Python programming language and the PyCharm software environment will be used to implement the project.

Довідки	Формат	Позначення	Найменування	Кількість листів	№ екземпляра	Примітка
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Система генерації музики	3		
			за допомогою рекурентних			
			нейронних мереж			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Система генерації музики	53		
			за допомогою рекурентних			
			нейронних мереж			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Система генерації музики	1		
			за допомогою рекурентних			
			нейронних мереж			
			Структурна схема			
			системи			
	А4	***ІАЛЦ.4672008.005 Д2***	Система генерації музики	1		
			за допомогою рекурентних			
			нейронних мереж			
			Функціональна схема			
			(обчислювальний граф			
			моделі)			
	А4	***ІАЛЦ.467200.006 Д3***	Система генерації музики	1		
			за допомогою рекурентних			
			нейронних мереж			
			Схема алгоритму			

					ІАЛЦ.467200.001 ОА			
Зм		*№ докум.*	*Підп.*	*Дата*				
Розробив		Ковальков Д.П.			*Система генерації музики за допомогою рекурентних нейронних мереж Опис альбому*	Літ.	Аркуш	Аркушів
Перевірив		Кочура Ю.П.					1	1
						НТУУ "КПІ" ФІОТ ІВ-81		

Технічне завдання

до дипломного проєкту

на тему: «*Система генерації музики за допомогою рекурентних нейронних мереж*»

Київ – 2022

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування: «Система генерації музики за допомогою рекурентних нейронних мереж»

Областю застосування цієї системи є проектування акустичного обладнання, аналіз даних, економічне та фінансове прогнозування, оцінка та управління екологічним ризиком, дизайн промислового продукту, оптимізація в числовій математиці, оптимальна робота "закритих" (конфіденційних) інженерних або інших систем, для яких поставлена задача глобальної оптимізації.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського проекту з освітньо-професійної програми “Комп’ютерні системи та мережі”, спеціальність 123 “Комп’ютерна інженерія”, затверджене кафедрою обчислювальної техніки Національного технічного університету України “Київський політехнічний інститут імені Ігоря Сікорського”.

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є створення системи генерації музики з використанням рекурентних нейронних мереж.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до продукту, що розробляється

- Технічна коректність
- Підготовка та навчання моделі
- Створення коректного вихідного файлу

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- Python 3

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми проєкту	10.12.2021-18.01.2022
Вивчення та аналіз завдання	02.05.2022-13.05.2022
Програмна реалізація системи	16.05.2022-11.06.2022
Оформлення пояснювальної записки	02.05.2022-11.06.2022

Пояснювальна записка

До дипломного проєкту

на тему: «*Система генерації музики за допомогою рекурентних нейронних мереж*»

Київ – 2022

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

MIDI – Musical Instrument Digital Interface

RNN – Recurrent Neural Network

CNN - Convolutional Neural Network

VQ-VAE - Vector Quantized Variational AutoEncoder

LSTM - Long Short-Term Memory

GPU - Graphics Processing Unit

API - Application Programming Interface

ANN - Artificial Neural Network

ВСТУП

Музика завжди була важливою складовою нашого життя. З давніх часів люди створювали музику за допомогою різних засобів, підручних матеріалів та для різних потреб і з різними цілями. Музика допомагає людині відволікатися від життєвих турбот , а також налаштовуватися на певний ряд подій. З її допомогою можна розслабитись. Музика за весь час існування не лише приносить задоволення під час прослуховування, а й саме створення музики є чудовим хобі. Тобто вона несе позитивні настрої у всіх своїх аспектах.

Музика допомагає людям у вирішенні певних потреб. Наприклад, задля налаштування на виконання якоїсь роботи, задання певного настрою, допомоги у розслабленні та релаксації.

З часом, з поступовим розвитком різних технологій змінювалась і музика, а також вдосконалювались методи її створення, з'являлись нові інструменти для реалізації цього.

З приходом нових сучасних технологій з'явились нові методи для реалізації різних потреб людини. Ці технології допомогли замінити людину у виконанні деяких завдань, здебільшого простої роботи. Це стосується і музики. До того ж, нові технології не лише спростили створення музики, а й дозволяють створювати програмне забезпечення, яке самостійно може цю музику створювати, тобто навіть виконувати творчу складову, що довгий час вважалось неможливим.

З розвитком різних сфер життя, з'являється все більше напрямків використання музики. Музичний супровід доповнює ті процеси, в яких використовується. В фільмах – задає потрібну атмосферу для тієї чи іншої сцени, події. В іграх – задає гравцю необхідного запалу, викликає заплановані розробником емоції. В театрі музика доповнює виступ, допомагає краще зануритись в події, що відбуваються на сцені, підсилити враження від дійства.

Навіть у магазинах, торгових та громадських центрах музика використовується для виклику у відвідувачів певних емоцій.

Метою цієї дипломної роботи є огляд та аналіз існуючих рішень щодо генерації музики, а також створення власної системи. Для досягнення кінцевого результату буде проведено аналіз існуючих можливостей, а також буде обрано шлях на основі використання рекурентних нейронних мереж.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ ЗА ТЕМОЮ ДИПЛОМНОГО ПРОЕКТУ.

Основним завданням даної дипломної роботи є створення спеціалізованого програмного забезпечення, яке б надало користувачу можливість генерувати музикальні композиції за допомогою штучного інтелекту.

Всі маніпуляції будуть проводитись з файлами, що мають розширення MIDI.

MIDI - це технічний стандарт, що описує протокол зв'язку, цифровий інтерфейс та електричні роз'єми, що з'єднують широкий спектр комп'ютерів, електронних музичних інструментів та відповідних аудіопристроїв для запису, редагування та відтворення музики[1]. Специфікація бере свій початок у документі Universal Synthesizer Interface, опублікованому Четом Вудом та Дейвом Смітом із Sequential Circuits на конференції Audio Engineering Society, що проходила у 1981 році в Нью-Йорку[2].

Також визначено формат файлу, в якому зберігаються та обмінюються дані. Переваги MIDI включають невеликий розмір файлу, простоту модифікації та маніпуляцій, а також широкий вибір електронних інструментів та синтезаторних чи цифрових семпльованих звуків[3]. MIDI запис виконання на клавіатурі може звучати наприклад як фортепіано або інший клавішний інструмент; однак, оскільки MIDI записує повідомлення та інформацію про ноти, а не конкретні звуки, то цей запис може бути змінений на безліч інших звуків, від синтезатору, гітари або семпльованої флейти до повного оркестру. Запис MIDI не є аудіосигналом, як наприклад запис звуку, зроблений за допомогою мікрофона.

Файли з таким розширенням підтримуються майже всім програмним забезпеченням для створення та редагування музики. До того ж їх дуже легко перетворити на звичні для більшості музикантів ноти.

1.1. Історія автоматичного створення музики з використанням нейронних мереж

1.1.1. Перші роботи Льюїса і Тодда

Перші роботи за даною тематикою датуються 1988 роком. Це були роботи Льюїса і Тодда, які запропонували використовувати нейронні мережі для автоматичного створення музики.

З одного боку, Льюїс використовував багатошаровий перцептрон для свого алгоритмічного підходу до композиції під назвою "створення шляхом уточнення"[4]. По суті, він заснований на ідеї використання градієнтів для створення мистецтва.

У машинному навчанні перцептрон – являє собою алгоритм контрольованого навчання бінарних класифікаторів. Бінарний класифікатор - це функція, яка може вирішити, до якого з двох класів належить вхідний сигнал, представлений вектором чисел[5]. Це тип лінійного класифікатора, тобто алгоритм класифікації, який робить свої передбачення на основі лінійної функції, що поєднує набір ваг з вектором вхідних ознак.

З іншого боку, Тодд використовував авторегресійну нейронну мережу (RNN) для послідовного створення музики - принцип, який через стільки років все ще залишається актуальним. Багато людей продовжували використовувати цю ж ідею протягом багатьох років, серед них: Екк та Шмідхубер, які запропонували використовувати LSTM для алгоритмічної композиції. Або якщо розглядати пізніші роботи, модель Wavenet (яка "здатна" генерувати музику) також використовує цей же причинно-наслідковий принцип[5].

Видно, що старі ідеї, які Тодд та Льюїс запровадили у 80-х роках для алгоритмічної композиції, актуальні й сьогодні. Але якщо їхні принципи були вірні, чому вони не досягли успіху? Ну, за словами Льюїса: в ті часи "було важко вираховувати будь-що". У той час, як один сучасний GPU у робочій станції глибокого навчання може мати близько 110 тфлопс теоретичної продуктивності, VAX-11/780 (робоча станція, яку він використав у 1988 році для своєї роботи) мала 0,1 мфлопс. Різниця дійсно є вкрай суттєвою.

1.1.2. Робота Дугласа Екка та Юргена Шмідхубера

Також досить важливою та цікавою є робота 2002 року. Її авторами є Дуглас Екк та Юрген Шмідхубер. У своїй роботі, яка має назву ‘Finding temporal structure in music: blues improvisation with LSTM’ вони намагаються вирішити одну з основних проблем, з якою стикалася (і досі стикається) алгоритмізована музика: відсутність глобальної зв'язності чи структури.

Як зазначали самі автори: «Тут ми розглядаємо проблему отримання основних компонентів музичних сигналів, таких як чітко визначена глобальна тимчасова структура у вигляді вкладених періодичностей. Чи можна створити адаптивний пристрій обробки сигналів, який навчиться генерувати нові екземпляри заданого музичного стилю? Оскільки рекурентні нейронні мережі, в принципі, можуть вивчати тимчасову структуру сигналу, вони є хорошими кандидатами для вирішення такого завдання. На жаль, музика, створена стандартними рекурентними нейронними мережами (RNN), часто не має глобальної зв'язності. Причина цього недоліку, мабуть, у тому, що RNN неспроможні відстежувати тимчасово віддалені події, які вказують на глобальну структуру музики. Довга короткочасна пам'ять (LSTM) досягла успіху в аналогічних областях, де інші рекурентні нейронні мережі зазнали невдачі, наприклад, у підрахунку часу та навчанні контекстно-залежних мов.»[5]

Для вирішення цієї проблеми вони запропонували використовувати LSTM, які нібито краще, ніж звичні RNN, навчаються довгим тимчасовим залежностям. У наведеному вище дослідженні вчені довели, що LSTM також є хорошим механізмом для навчання створенню музики. Представлені ними результати експериментів показали, що LSTM успішно навчається формі музики в жанрі блюз і здатна складати нові мелодії в цьому стилі. Примітно, що як тільки мережа знаходить відповідну структуру, вона не відхиляється від неї. Як зазначають самі автори: «LSTM здатна грати блюз із хорошою синхронізацією та правильною структурою доти, доки людина готова слухати.»[6]

Варто також зазначити, що в результаті цих експериментів музика стала одним із перших застосувань LSTM.

До 2009 року більшість робіт було присвячено проблемі алгоритмічного створення музики. В основному, вони намагалися зробити це за допомогою RNN. Проте були певні винятки.

1.1.3. Нейронні мережі для виявлення початку ноти в фортепіанній музиці Матьє Марольта

Ще у 2002 році Марольт та його колеги почали використовували багатошаровий перцептрон, який працював поверх спектрограм для отримання можливості визначення початку нот[7]. Це був перший випадок, коли хтось обробляв музику не в символьному форматі. Саме це започаткувало новітню еру досліджень. Почалася гонка за те, хто першим вирішить будь-яке завдання в режимі наскрізного навчання. Це означає, що потрібно вивчити систему відображення (або функцію), здатну вирішувати задачу безпосередньо з необробленого аудіо, на відміну від її вирішення за допомогою розроблених характеристик (наприклад, спектрограм) або з символічних музичних уявлень (наприклад, MIDI партитур), як це було реалізовано до цього[7].

1.1.4. Система класифікації музичних жанрів Хонглака Лі

З розвитком технологій та плином часу люди почали вирішувати складніші завдання, такі як розпізнавання акордів або тегування музичних аудіофайлів за допомогою класифікаторів глибокого навчання.

Наслідуючи підхід Хінтона, Хонглак Лі та його колеги (створили першу глибоку конволюційну нейронну мережу для класифікації музичних жанрів. Ця робота стала основною і заклала фундамент для цілого покоління дослідників глибокого навчання, які витратили багато зусиль на розробку кращих моделей для розпізнавання високорівневих (семантичних) понять із музичних спектрограм[7].

1.1.5. Переосмислення автоматичного розпізнавання акордів за допомогою згорткових нейронних мереж

Ще одна історично визначна робота, яку варто було б згадати належить Хамфрі та Белло (2012), які в ті дні пропонували використовувати глибокі нейронні мережі для розпізнавання акордів. У їхній статті вони пояснюють дослідникам музичних технологій, що це хороша ідея – вивчати (ієрархічні) уявлення даних.

1.1.6. Наскрізне навчання для аудіо музики

Однак, варто зазначити, що не всі були задоволені використанням моделей на основі спектрограм, що використовував Лі. Приблизно у 2014 році Ділеман та його колеги почали вивчати амбітний напрямок досліджень, який був представлений світові як «End-to-end learning for music audio». У цій роботі вони досліджували ідею прямої обробки хвильових форм для завдання тегування музичних аудіофайлів, що мало певний успіх, оскільки моделі на основі

спектрограм, як і раніше, показували дещо кращі результати при тестуваннях ніж моделі на основі хвильових форм. У той час моделі не тільки не були зрілими, але й навчальні дані були мізерними в порівнянні з тими обсягами даних, до яких зараз мають доступ деякі компанії. Це цілком закономірне явище, адже з плином часу інформації в світі стає все більше. Наприклад, недавнє дослідження, проведене на Pandora Radio, показало, що моделі на основі хвильових форм можуть перевершувати моделі на основі спектрограм за умови наявності достатньої кількості навчальних даних.

Загалом тематика роботи є досить цікавою, як можна бачити з рисунка 1.1, поступово зацікавленість розробниками та дослідниками збільшується. Проаналізувавши наведений графік, можна побачити, що велику кількість статей та праць по даній тематиці було опубліковано досить недавно, переважно за останні п’ять років.

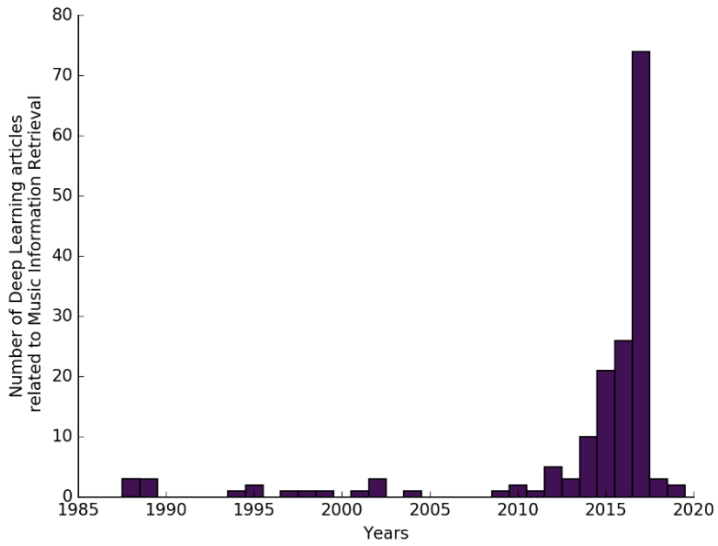


Рисунок 1.1 Кількість публікацій за темою диплому по рокам[8]

1.2. Сучасні засоби для штучної генерації музичних творів

1.2.1. OpenAI’s Jukebox

OpenAI – це дослідницька лабораторія штучного інтелекту, що складається з комерційної корпорації OpenAI LP та її материнської компанії, некомерційної OpenAI Inc. Компанія, що вважається конкурентом DeepMind, проводить дослідження в області штучного інтелекту із заявленою метою просування та розвитку дружнього штучного інтелекту таким чином, щоб це принесло користь всьому людству. Вони мають кілька художніх проектів, зокрема, GPT-3 для літератури, та Jukebox, який спеціалізується на музичних творах[9].

Модель автокодера Jukebox стискає аудіо в дискретний простір, використовуючи підхід на основі квантування, що називається VQ-VAE. Ієрархічні VQ-VAE можуть генерувати короткі інструментальні твори з кількох наборів інструментів, однак вони страждають від руйнування ієрархії через використання послідовних кодерів у поєднанні з авторегресивними декодерами[9]. Спрощений варіант під назвою VQ-VAE-2 дозволяє уникнути цих проблем за рахунок використання тільки кодерів та декодерів прямої дії, і він показує вражаючі результати при генерації високоточних зображень.

Компанія використовує три рівні VQ-VAE, показані нижче, які стискають необроблений звук 44 кГц в 8, 32 і 128 разів. При такому стисканні втрачається більшість деталей звуку, і звук стає набагато більш галасливим у міру зниження рівня. Однак при цьому зберігається важлива інформація про висоту тону, тембр і гучність звуку.

Основна ідея у тому, що вони беруть необроблений звук і кодують його з допомогою згорткових нейронних мереж (CNN). Фактично це спосіб стиснення багатьох змінних у меншу кількість. Їм доводиться це робити, оскільки аудіо має 44000 змінних в секунду, а пісні, як відомо можуть бути досить довгими. Потім

виконується процес генерації, але вже на меншому наборі змінних, після чого виконується зворотний процес під час якого декомпресують у 44000 змінних[9].

Як зазначають автори, для навчання даної моделі, була зібрана база даних з 1,2 мільйона пісень, переважна більшість з яких була на англійській мові. Після цього вони зіставили їх з метаданими та текстами пісень. До метаданих відносяться відомості про: автора, жанр, рік випуску та приналежність до альбому, а також ключові слова, які пов'язані з кожним твором.

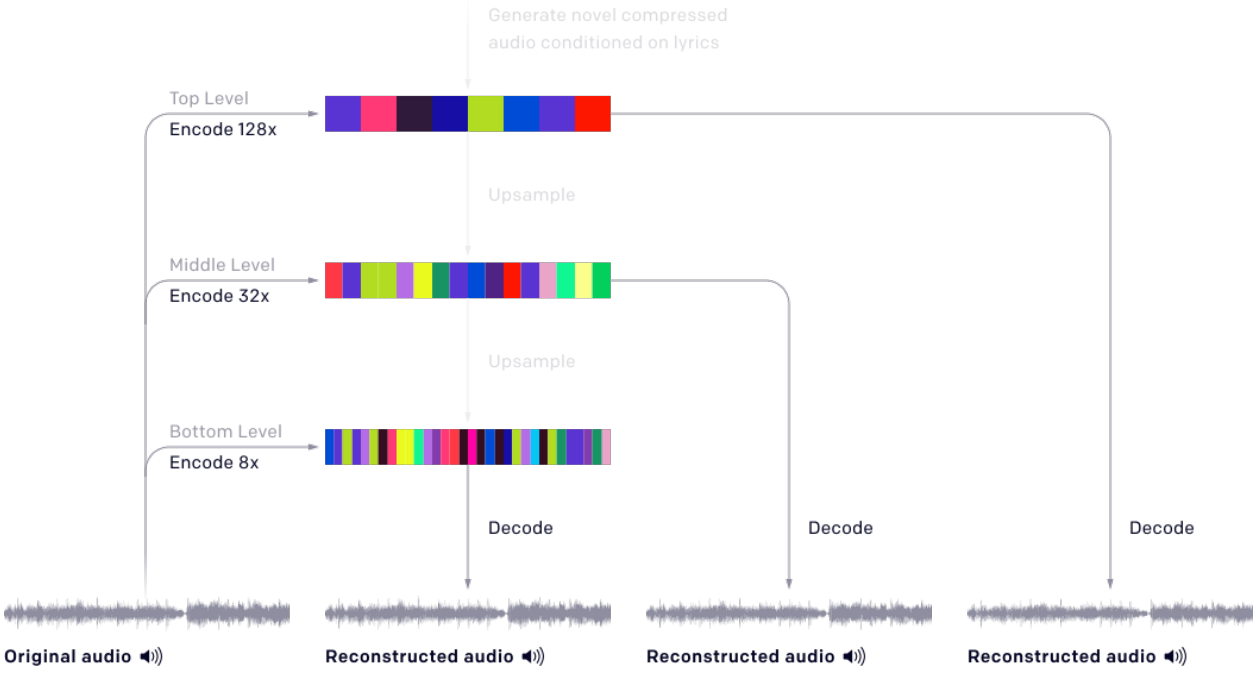


Рисунок 1.2 Процес стиснення даних Jukebox[9]

1.2.2. AIVA

Створена у лютому 2016 року, AIVA спеціалізується на створенні класичної та симфонічної музики[10]. Вона стала першим у світі віртуальним композитором, визнаним музичним суспільством. Навчена на великій кількості вже існуючих творів класичної музики, написаних відомими композиторами, AIVA здатна виявляти закономірності в музиці і на цій основі самостійно складати музичні твори. Алгоритм AIVA заснований на архітектурах глибокого

навчання. З січня 2019 року компанія пропонує комерційний продукт Music Engine, що здатний генерувати короткі композиції у різних стилях[11].

AIVA багато в чому нагадує OpenAI Jukebox. Проте їх підходи розрізняються у базовій структурі даних, на основі яких вони генерують. музику. AIVA працює з так званим MIDI-файлом. Різниця тут у тому, що OpenAI у попередньому прикладі взяв MIDI-файл та створив аудіозапис. Але AIVA створює, враховуючи MIDI-файл, аналогічну, але зовсім нову та іншу композицію.

Їхній підхід полягає в тому, що користувач може завантажити пісню у форматі MIDI. Сервіс використовуватиме цей файл як основу, щоб зумовити свій процес генерації. Іншими словами, ця пісня впливатиме на створювану в процесі композицію.

1.2.3. Amper Music

Цілком інший підхід використовує Amper Music. Він використовує так звані дескриптори.

Дескриптори – це музичні алгоритми, що грають певний стиль музики. Один дескриптор може бути орієнтованим для гри панкового нью-йоркського року, а інший може досягти успіху в документальній класичній композиції[12].

Під час створення ми можемо редагувати дві змінні. Перша – це довжина пісні. А друга - набір певних слів та словосполучень, що словесно описують дескриптор. Даний функціонал представлений на рисунку 1.3. Після цього вам пропонується вибрати набір інструментів, які будуть взяті за основу. Після генерації музичного твору у нас є можливість додатково редагувати його. Всі створені треки умовно діляться на три частини. Це початок, основна частина і кінець. Ми можемо за допомогою скролів масштабувати ці елементи.

В результаті користувач має можливість отримати музикальний твір досить високої якості. А після певного редагування, яке можна проводити одразу після генерації, все стає ще краще.

Основним недоліком даного сервісу є неможливість завантажити готові файли в розширенні .midi, яке найчастіше використовується для професійної роботи з музикою.

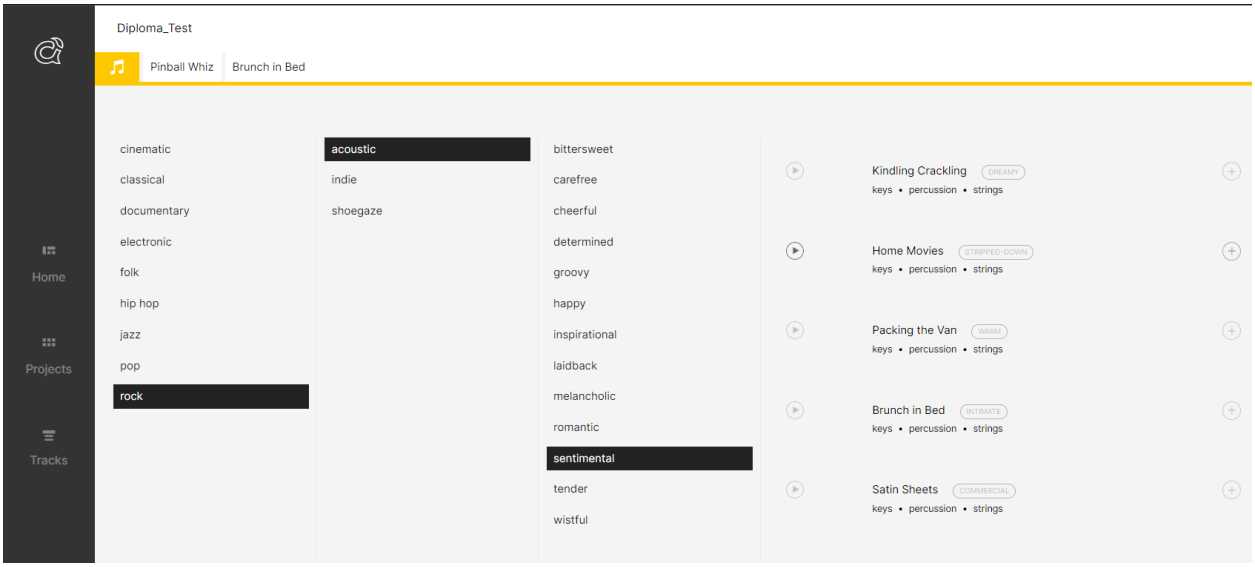


Рисунок 1.3 Вибір слів для опису дескриптора

ВИСНОВОК ДО РОЗДІЛУ 1

За результатами проведених пошуків та аналізу можна стверджувати, що цей напрям є досить цікавим. За останні 35 років стає все більше статей та публікацій за темою роботи. Отже, можна стверджувати, що тема стає більш популярною і постійно розвивається.

РОЗДІЛ 2
ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ

Штучний інтелект - це високотехнологічна область, яка потребує найвищого рівня знань. Тому що створення продуктів, які можуть мислити та діяти як люди, не просте завдання. Величезним кроком до того щоб розпочати роботу є обрання мови програмування.

2.1. Огляд існуючих мов програмування

2.1.1. Мова програмування Java

Мова програмування Java від компанії Oracle є однією з найвідоміших мов програмування. Java – це високорівнева об'єктно-орієнтована мова програмування загального призначення. Протягом багатьох років вона адаптувалася до останніх інновацій та технологічних досягнень. Те саме справедливо і для технологій штучного інтелекту.

Для роботи зі штучним інтелектом, мова Java забезпечує простоту використання та налагодження, а також спрощує масштабні проекти. Існування технології віртуальних машин є найбільшою перевагою використання мови програмування Java. Віртуальна машина Java спрощує процес реалізації, заощаджуючи ваш час та енергію від компіляції програми знову і знову. Ця технологія допомагає розробникам створювати єдину версію програми, яку можна запускати на інших платформах на базі Java[13].

Розробники також можуть працювати над графікою та інтерфейсами, роблячи їх привабливішими за допомогою стандартного набору інструментів

Standard Widget Toolkit. В цілому, Java допомагає підтримувати, переносити та робити безпечними програми штучного інтелекту[14].

2.1.2. Мова програмування JavaSript

Так само, як і Java, JavaScript також добре підходить для роботи зі штучним інтелектом. Однак переважно він використовується для розробки більш безпечних та динамічних веб-сайтів.

Як і Java, JavaScript також має велике співтовариство розробників, які підтримують процес розробки. Завдяки таким бібліотекам, як jQuery, React.js та Underscore.js, розробка стає більш швидкою та ефективною. Найбільш відомими та розповсюдженими бібліотеками для роботи зі штучним інтелектом на JavaScript є: Keras.js, TensorFlow.js, Neuro.js, ML5.js та Brain.js[15].

Ще однією перевагою машинного навчання JavaScript є легка інтеграція з мобільними додатками. Вже існує багатий набір кросплатформенних інструментів розробки мобільних додатків на JavaScript, таких як Cordova та Ionic.

Ці інструменти стали дуже популярними, оскільки дозволяють один раз написати код та розгорнути його для пристроїв iOS та Android. Щоб зробити код сумісним у різних операційних системах, кросплатформні інструменти розробки запускають "webview", об'єкт браузера, який може виконувати код JavaScript і може бути вбудований у програму цільової операційної системи. Ці об'єкти браузера підтримують бібліотеки машинного навчання JavaScript.

Винятком є React Native, популярне кросплатформенне середовище розробки мобільних додатків, яке не покладається на "webview" для запуску додатків. Однак, враховуючи популярність мобільних програм машинного навчання, Google випустив спеціальну версію TensorFlow.js для React Native[15].

Якщо ви написали свій мобільний додаток у нативному коді (тобто без використання додаткових бібліотек) і хочете інтегрувати в нього код машинного

навчання на JavaScript, ви можете додати до своєї програми власний вбудований об'єкт браузера (наприклад, WKWebView в iOS).

Існують інші бібліотеки машинного навчання для мобільних додатків, такі як TensorFlow Lite і Core ML. Однак вони вимагають нативного кодування на мобільній платформі, для якої ви розробляєте свою програму. Машинне навчання JavaScript, з іншого боку, дуже універсальне. Якщо ви вже реалізували версію програми машинного навчання для браузера, ви можете легко перенести її на мобільний додаток практично без змін.

Варто відзначити, що машинне навчання JavaScript на стороні сервера також розвивається. Ви можете запускати бібліотеки машинного навчання JavaScript на Node.js, серверному движку програм JavaScript. TensorFlow.js має спеціальну версію, яка підходить для серверів під керуванням Node.js. Код JavaScript, який ви використовуєте для взаємодії з TensorFlow.js, такий же, як і для програм, що працюють у браузері. Але у фоновому режимі бібліотека використовує спеціальне обладнання сервера, щоб прискорити навчання та висновки. Машинне навчання на Node.js - досить новий напрямок, але воно швидко розвивається, оскільки зростає інтерес до додавання можливостей машинного навчання до веб- та мобільних додатків. У міру зростання спільноти фахівців з машинного навчання на JavaScript та подальшого розвитку інструментів, Node.js може стати найкращим варіантом для багатьох веб-розробників, які хочуть додати машинне навчання та штучний інтелект у свій набір навичок.[16]

Використовуючи JavaScript, ви можете забезпечити безпеку, високу продуктивність та скоротити час розробки.

2.1.3. Мова програмування Lisp

Мова програмування Lisp - одна з найстаріших мов, що використовуються для розробки штучного інтелекту. Вона була розроблена у 1960-х роках і завжди

була адаптованою та інтелектуальною мовою. Якщо ваш проект вимагає модифікації коду, вирішення проблем, швидкого створення прототипів або динамічної розробки, Lisp - це те, що вам потрібно[15].

Common Lisp відома своєю надзвичайною гнучкістю, чудовою підтримкою об'єктно-орієнтованого програмування та можливістю швидкого створення прототипів. Вона також має надзвичайно потужну систему макросів, яка дозволяє адаптувати мову до ваших програм, і гнучке середовище виконання, яке дозволяє модифікувати і налагоджувати запущені програми (відмінно підходить для розробки на стороні сервера і довго працюючого критично важливого програмного забезпечення). Це мультипарадигмальна мова програмування, що надає можливість підібрати підхід та парадигму відповідно до галузі застосування[17].

Деякі успішні проекти, створені на Lisp, – Routinic, Grammarly та DART. Хоча вона має свої недоліки, Lisp, як і раніше, є перспективною мовою програмування для розробки штучного інтелекту.

Синтаксис Lisp є досить специфічним, а ще мова не має бібліотек. Крім того, для роботи з нею потрібні певні спеціальні конфігурації програмного забезпечення та комп'ютера.

Lisp - одна з найстаріших мов програмування, яка за фактом є предком кількох інших мов. На час свого виникнення це був фундаментальний прорив, який сприяв тому, що штучний інтелект став функціональним інструментом для машинного навчання. Проте сьогодні ця мова не часто використовується для штучного інтелекту. Спільнота віддає належне цієї технології, тому що це витоки штучного інтелекту. Але реальність така, що зараз випереджають інші мови програмування[16].

2.1.4. Мова програмування R

Мова програмування R - одна із найбільш підходящих варіантів для проектів, де потрібні статистичні обчислення. Вона підтримує такі бібліотеки, як MXNet, TensorFlow, Keras. Ця мова використовується у багатьох галузях, таких як освіта, фінанси, телекомунікації, фармацевтика тощо. Саме цією мовою користуються технологічні гіганти, такі як Microsoft, Google, Facebook, а також компанії, як Uber, Airbnb та інші.

R включає пакети, створені користувачем, такі як графічні пристрої, інструменти, можливості імпорту/експорту, статистичні методи. Завдяки вбудованій підтримці графіки та моделювання даних, мова дозволяє розробникам без особливих проблем працювати над сучасними методами глибокого навчання[18].

2.1.5. Мова програмування C++

Розроблена в 1983 Б'ярном Струструпом, C++ є найшвидшою мовою програмування, що ідеально підходить для проектів штучного інтелекту, котрі вимагають багато часу. Вона використовується при написанні програм, в тих випадках коли важливі продуктивність та правильне використання ресурсів. Вона також дає простір широкого використання алгоритмів і статистичних методів штучного інтелекту, і навіть підтримує повторне використання програм розробки[13].

C++ використовується для ресурсомістких додатків, штучного інтелекту в іграх та програмування роботів, а також для швидкого виконання проектів завдяки високому рівню продуктивності та ефективності.

Відомим користувачем машинного навчання та штучного інтелекту є Google. Google використовує C++ у різних областях оптимізації пошукових систем, особливо Google Chrome.

2.1.6. Мова програмування Python

Ще одна мова програмування – Python, зарекомендував себе як мову програмування в якій знайдуться засоби та технології майже для будь-якого напрямку розробки. Однією з причин, чому він є хорошим вибором, є його простота. Враховуючи, що розробка штучного інтелекту є досить складною задачею, буде набагато краще, якщо використовувана мова програмування буде простою для розуміння та реалізації. Синтаксис для програмування на Python може бути легко вивчений будь-ким, хто цікавиться програмуванням. Те саме стосується і реалізації алгоритмів цієї мови.

Незалежно від галузі або розміру проекту, Python – хороший вибір: як для написання невеликих скриптів, так і для створення та підтримки великих проектів[19].

Python – це універсальна мова, яка підтримує різні стилі програмування. Вони включають об'єктно-орієнтоване, функціональне і процедурне програмування. Мова має багато бібліотек, які підтримують штучний інтелект. Одним з них є Pybrain, який використовується для машинного навчання в Python. Ще одна важлива бібліотека – Numpy, яка призначена для виконання складних обчислень.

Є багато причин чому розвиток штучного інтелекту на цій мові так пришвидшився за останні роки. До цих причин належать:

- Великий вибір бібліотек - одна з основних причин, через яку Python є найпопулярнішою мовою програмування, що використовується для розробки штучного інтелекту. Бібліотека - це модуль або група модулів, опублікованих різними джерелами, які включають попередньо написану частину коду, котра дозволяє користувачам досягти деякої функціональності або виконати різні дії. Бібліотеки Python надають елементи базового рівня, тому розробникам не потрібно щоразу кодувати

їх із самого початку. Існує безліч форумів та підручників з Python, до яких можна звернутися за допомогою[20].

- Низький початковий бар'єр дозволяє фахівцям швидко освоїти Python і почати використовувати його для розробки, не витрачаючи надто багато зусиль на вивчення мови. Мова програмування Python схожа на повсякденну англійську мову, і це полегшує процес навчання. Її простий синтаксис дозволяє комфортно працювати із складними системами, забезпечуючи чіткі зв'язки між елементами системи[20].
- Python - мова з відкритим кодом, а це означає, що для програмістів, починаючи з початківців і закінчуючи професіоналами, відкрито багато ресурсів. Велика кількість документації з Python доступна в Інтернеті, а також у спільнотах та форумах Python, де програмісти та розробники обговорюють помилки, вирішують проблеми та допомагають один одному[20].

Завдяки всім цим та багатьом іншим особливостям, Python став однією з найкращих мов для розробки штучного інтелекту.

2.2. Огляд бібліотек машинного навчання та штучного інтелекту на мові програмування Python

Бібліотека - це колекція заздалегідь скомпільованого коду, створеного для виконання конкретних, чітко визначених операцій у програмі. Ці частини коду зазвичай розробляються для загальних операцій, які залишаються незмінними в різних програмах[21].

Крім попередньо скомпільованого коду, бібліотеки також складаються з конфігураційних даних, шаблонів, документації, класів, значень тощо. Ця колекція зібраних воєдино ресурсів робить програмування зручним і більш простим для розробника, оскільки він може не створювати схожі функції, які

служать одній і тій же цілі знову і знову[21]. Бібліотеки надзвичайно корисні для пришвидшення досягнення успіху з розробки на Python та на багатьох інших мовах програмування.

2.2.1. Бібліотека Numpy

Це досить відома та популярна бібліотека Python, яка використовується для роботи з багатовимірними даними та складними математичними функціями над цими даними. Бібліотека Numpy підвищує швидкість обчислення математичних виразів та виконання складних функцій, що працюють із масивами[21].

2.2.2. Бібліотека Pandas

Pandas – це відома бібліотека Python, яка зазвичай використовується для концепцій машинного навчання. По суті, це бібліотека аналізу даних, яка аналізує та маніпулює даними. Pandas полегшує розробникам роботу із структурованими багатовимірними даними та концепціями тимчасових рядів та дозволяє отримувати ефективні результати[21].

2.2.3. Бібліотека Matplotlib

Це бібліотека візуалізації даних, що використовується для побудови графіків та діаграм. Сама бібліотека є розширенням SciPy та обробляє складні моделі даних Pandas, а також структури даних NumPy. Matplotlib пропонує такі функції, як Basemap, інструменти GTK, Cartopy, Mplot4d і т.д., які допомагають

у побудові графіків зображень, 3D-графіків, контурних графіків та багато іншого[21].

2.2.4. Бібліотека SciPy

Це бібліотека Python, яка бере свій початок від NumPy. SciPy використовується службами розробки Python для виконання технічних та наукових обчислень на великих наборах даних. У бібліотеці є модулі оптимізації масивів та лінійної алгебри, які допомагають у науковому аналізі та проектуванні[21].

2.2.5. Бібліотека Scikit-learn

Це потужна бібліотека Python, яка спочатку була створена для моделювання даних та побудови алгоритмів машинного навчання. Вона має простий та послідовний інтерфейс, винятково зручний для користувача, що полегшує її використання та обмін даними[21].

2.2.6. Бібліотека TensorFlow

Це бібліотека машинного навчання з відкритим вихідним кодом, яка використовується для досягнення даних та виробничих цілей. Бібліотека пропонується Google і може бути використана для спрощення побудови моделей машинного навчання. TensorFlow пропонує гнучку структуру та архітектуру, що дозволяє їй працювати на різних обчислювальних платформах. Однак вона має

власний блок обробки тензорів (TPU), за допомогою якого можна отримати кращі результати[21].

2.2.7. Бібліотека Keras

Це бібліотека нейронних мереж Python з відкритим вихідним кодом, що пропонується як розширення бібліотеки TensorFlow. Keras призначена для побудови та оцінки нейронних мереж у рамках моделей машинного навчання та глибокого навчання. Бібліотека є гнучкою, модульною та розширюваною, в неї можна інтегрувати цілі, оптимізатори, шари та функції активації[21].

2.2.8. Бібліотека Theano

Це широко поширена бібліотека Python, яка використовується декількома компаніями-розробниками Python для оцінки складних математичних виразів. Вона спеціально розроблена для технології машинного навчання та дозволяє ефективно оптимізувати і оцінювати матричні обчислення. Theaono може бути інтегрована з NumPy для збільшення швидкості обчислень до 140 відсотків[21].

2.2.9. Бібліотека PyTorch

Це дуже популярна бібліотека Python, яка готова до виробництва, яка зазвичай використовується в концепціях машинного навчання. Бібліотека підтримує GPU-прискорення та забезпечує оптимізацію продуктивності глибоких нейронних мереж. PyTorch в основному використовується для

підвищення продуктивності фреймворків глибокого навчання та підтримується великою спільнотою Python[21].

2.3. Огляд варіацій нейронних мереж

2.3.1. Штучна нейронна мережа

Штучна нейронна мережа, є групою з кількох перцептронів в кожному шарі. ANN також відома як нейронна мережа з прямою передачею даних, оскільки вхідні дані обробляються тільки в прямому напрямку[22].

Як можна побачити з рисунку 2.1, ANN складається з 3 шарів - вхідного, прихованого та вихідного. Вхідний шар приймає вхідні дані, прихований шар обробляє вхідні дані, а вихідний шар видає результат.

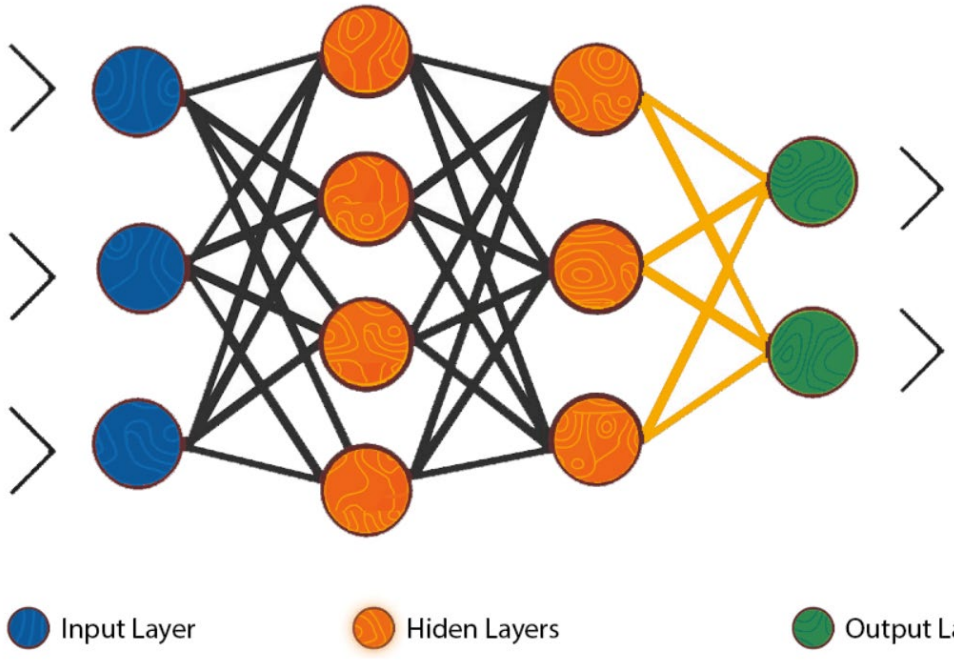


Рисунок 2.1 Штучна нейронна мережа[22]

Штучна нейронна мережа здатна навчатися будь-якій нелінійній функції. Тому ці мережі відомі як універсальні апроксиматори функцій.

Однією з основних причин універсальної апроксимації є функція активації. Функції активації надають мережі нелінійних властивостей. Це допомагає мережі вивчати будь-які складні відносини між входом та виходом[22].

2.3.2. Згорткова нейронна мережа

Згорткові нейронні мережі (CNN) зараз дуже часто використовуються для глибокого навчання. Ці моделі CNN зустрічаються в різних програмах і областях, і вони особливо поширені в проектах обробки зображень і відео.

Будівельними блоками CNN є фільтри, так звані ядра. Ядра використовуються для отримання відповідних характеристик з вхідних даних за допомогою операції згортки. В результаті згортки зображення з фільтрами виходить карта ознак, приклад такої продемонстровано на рисунку 2.2[22].

Рисунок 2.2 Результат роботи згорткової нейронної мережі[22]

В даний час CNN є найбільш підходящою моделлю для вирішення всіх проблем, пов'язаних із зображеннями. За точністю вони перевершують конкурентів. Основна перевага CNN у порівнянні з попередниками полягає в

тому, що вони автоматично визначають важливі особливості без контролю з боку людини. Наприклад, за наявності безлічі фотографій кішок та собак система самостійно визначає відмітні ознаки для кожного класу[23].

CNN також ефективна з обчислювальної точки зору. Вона використовує спеціальні операції згортки та об'єднання, а також виконує спільне використання параметрів. Це дозволяє моделям CNN працювати на будь-якому пристрої[23].

2.3.3. Рекурентні нейронні мережі

Рекурентна нейронна мережа (RNN) - це особливий тип штучної нейронної мережі, пристосований для роботи з даними часових рядів або даними, що включають послідовності. Прості нейронні мережі з прямою передачею призначені тільки для даних, які не залежать один від одного. Однак якщо ми маємо дані у послідовності, в якій одна точка даних залежить від попередньої, нам необхідно модифікувати нейронну мережу, щоб врахувати залежність між цими точками даних. У RNN існує концепція "пам'яті", яка допомагає їм зберігати стан або інформацію попередніх входів для генерації наступного виходу послідовності[24].

Важливим плюсом рекуркнтних нейронних мереж є те, що RNN враховує послідовну інформацію, що є у вхідних даних, тобто залежності, роблячи прогнози.

2.3.4. Мережі з довготривалою пам’яттю

Мережі з довготривалою пам'яттю (LSTM)- це особливий вид RNN, здатних до навчання довгострокових залежностей. Вони дуже добре працюють на великій кількості завдань і в даний час широко використовуються.

LSTM розроблені для того, щоб уникнути проблеми довгострокової залежності. Запам'ятовування інформації протягом тривалих періодів часу - це практично їхня поведінка за умовчаннямна відміну від звичайних рекурентних мереж.

Всі рекурентні нейронні мережі мають вигляд ланцюжка модулів нейронної мережі, що повторюються. У стандартних RNN цей повторюваний модуль матиме дуже просту структуру. Така структура продемонстрована на рисунку 2.3.

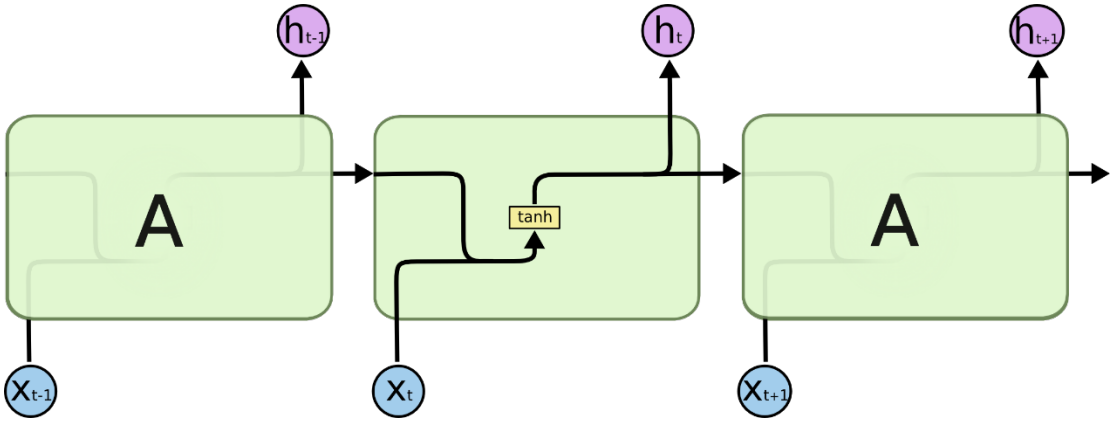


Рисунок 2.3 Структура рекурентної нейронної мережі[25]

LSTM також мають цю структуру, схожу на ланцюжок, але модуль, що повторюється, реалізований інакше. Замість одного шару нейронної мережі тут чотири, що взаємодіють особливим чином[25]. Приклад структури LSTM можна побачити на рисунку 2.4.

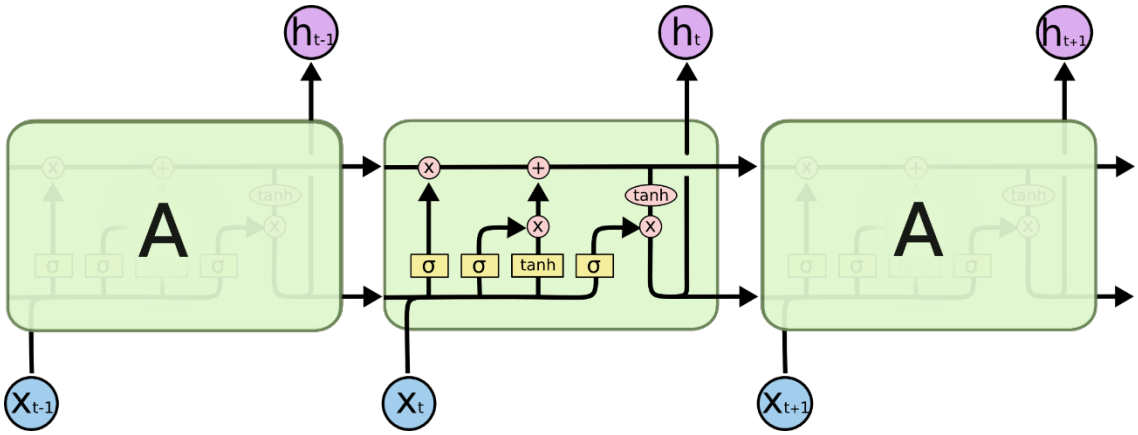


Рисунок 2.4 Структура мережі з довготривалою пам’яттю[25]

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі цього проєкту було наведено та розглянуто різні мови програмування. Були наведені сильні та слабкі сторони. Також додаткова розглянуто бібліотеки для мови програмування Python та різновиди нейронних мереж.

В результаті було обрано необхідні технології для подальшої роботи над реалізацією поставленого завдання.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СИСТЕМИ ГЕНЕРАЦІЇ МУЗИКИ ЗА ДОПОМОГОЮ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ.

3.1. Створення та навчання моделі

За результатами аналізу з другого розділу, були визначені технології для розробки.

Keras – високорівнева бібліотека для нейронних мереж, що спрощує взаємодію з Tensorflow. Він був розроблений для можливості швидкого проведення експериментів.

Music21 - це інструментарій Python, що використовується для комп'ютерного музикознавства. Він дозволяє опрацьовувати основи теорії музики, генерувати та аналізувати музичні приклади. Крім того, він дозволяє нам створювати об'єкти нот та акордів, щоб ми могли легко створювати власні MIDI-файли.

Проект складається з двох основних частин, а саме з тренування моделі та власне генерації вихідного файлу.

Звичайно, весь процес починається з опрацювання вхідних даних. У цьому розділі ми розглянемо, як працювати з вхідними даними, та як приготувати їх для використання у LSTM-моделі.

Для нашої мети підходить будь-який набір MIDI файлів, що складається з одного інструменту. Першим кроком до реалізації є вивчення даних, з якими ми працюватимемо.

Нижче можна побачити уривок із MIDI-файлу, який був прочитаний за допомогою Music21:

```
<music21.note.Note G>
<music21.chord.Chord C4 E3>
<music21.chord.Chord C4 E3>
<music21.note.Note C>
<music21.chord.Chord B-2 F3>
<music21.note.Note E>
<music21.note.Note G>
<music21.note.Note F>
<music21.chord.Chord B-2 F3>
<music21.note.Note F>
<music21.chord.Chord C4 E3>
<music21.note.Note E>
<music21.chord.Chord B-2 F3>
<music21.note.Note D>
<music21.chord.Chord C4 E3>
<music21.note.Note F>
<music21.chord.Chord C4 E3>
```

Рисунок 3.1 MIDI-файл прочитаний за допомогою Music21

Як можна побачити з рисунку 3.1, бібліотека Music21 розділяє аудіофайл на частини, котрі в свою чергу поділяються на два типи об'єктів: акорди і ноти. Об'єкти нот (Note) містять інформацію про висоту тону, октаву та зміщення ноти.

- Висота визначає частоти звуку, або наскільки він високий чи низький, і позначається буквами [A, B, C, D, E, F, G], де A позначає найвищий звук, а G в свою чергу навпаки найнижчий.
- Октава вказує, яка саме частина інструменту (набір клавіш) використовується.
- Зміщення ж означає розташування самої ноти у творі.

А об'єкти акордів (Chord) – це спеціальний об’єкт в якому міститься інформація про ті ноти, які програються в один момент часу.

Зрозуміло, що для генерації музики, наша рекурентна нейронна мережа повинна коректно передбачати, наступну ноту чи акорд. Для цього спочатку потрібно виокремити всі можливі варіанти нот та акордів з переліку файлів для навчання. Це означає, що наш масив пророцтв повинен містити всі ноти та акорди, які зустрічаються в нашому навчальному наборі.

Далі потрібно розібратися, де буде розміщуватися нота. При прослуховуванні музики, можна помітити, що ноти мають різні інтервали між ними. Може бути певна послідовність нот, а потім період відпочинку, коли жодна нота не грається.

Нижче, на рисунку 3.2 представлений ще один уривок із MIDI-файлу, прочитаного за допомогою Music21. Цього разу позаду об'єкта видно його зміщення. Це дозволяє побачити інтервал між кожною нотою та акордом.

Як видно з цього уривка та більшої частини набору даних, найбільш поширений інтервал між нотами – 0,5. Тому ми можемо спростити дані і модель, ігноруючи можливих зміни у списку. Це не дуже вплине на кінцеву мелодію музики, що генеруються мережею.

Тепер, коли ми вивчили дані та визначили, що як вхідні та вихідні дані нашої LSTM-мережі ми хочемо використовувати ноти та акорди, настав час підготувати дані для мережі.

```
<music21.note.Note C> 43.0
<music21.chord.Chord F4 A2> 43.0
<music21.note.Note A> 43.5
<music21.chord.Chord F4 A2> 43.5
<music21.note.Note E> 44.0
<music21.chord.Chord F4 A2> 44.0
<music21.chord.Chord F4 A2> 44.5
<music21.note.Note E-> 45.0
<music21.chord.Chord C3 A3> 45.0
<music21.chord.Chord F3 B1> 45.5
<music21.chord.Chord C3 A3> 46.0
<music21.chord.Chord F3 B1> 46.5
<music21.chord.Chord E3 A3> 47.0
<music21.chord.Chord E3 A3> 47.5
<music21.chord.Chord E3 A3> 48.0
<music21.chord.Chord E3 A3> 48.5
<music21.chord.Chord C3 A3> 49.0
<music21.chord.Chord F3 B1> 49.5
```

Рисунок 3.2 MIDI-файл прочитаний за допомогою Music21 (зі зміщенням)

Спочатку ми завантажимо дані до масиву, як показано в наведеному нижче фрагменті коду на рисунку 3.3:

Ми починаємо із завантаження кожного файлу та його подальшого опрацювання Music21 за допомогою функції converter.parse(file). В результаті, ми отримуємо список усіх нот та акордів у файлі. Ми додаємо висоту тону кожної ноти, використовуючи її рядкову нотацію, оскільки найбільш значущі частини ноти можна відтворити за допомогою рядкової нотації висоти тону. І ми додаємо кожен акорд, кодуючи ідентифікатор кожної ноти в акорді в один рядок, між собою ноти будуть відокремлюватися крапкою.

Таке кодування дозволить легко декодувати вихідні дані, згенеровані мережею, у правильні ноти та акорди.

Тепер, коли ми помістили всі ноти та акорди до послідовного списку, ми можемо створити послідовності, які будуть служити входом нашої мережі.

```
from music21 import converter, instrument, note, chord

notes = []

for file in glob.glob("midi_songs/*.mid"):
    midi = converter.parse(file)
    notes_to_parse = None

    parts = instrument.partitionByInstrument(midi)

    if parts:
        notes_to_parse = parts.parts[0].recurse()
    else:
        notes_to_parse = midi.flat.notes

    for element in notes_to_parse:
        if isinstance(element, note.Note):
            notes.append(str(element.pitch))
        elif isinstance(element, chord.Chord):
            notes.append('.'.join(str(n) for n in element.normalOrder))
```

Рисунок 3.3 Фрагмент коду опрацювання вхідних даних

У прикладі коду ми задали довжину кожної послідовності в 100 нот/акордів. Це означає, що для передбачення наступної ноти в послідовності мережа має попередні 100 нот, які допомагають зробити прогноз.

Останнім кроком у підготовці даних для мережі є нормалізація вхідних даних та односкладове кодування вихідних даних.

```
sequence_length = 100


pitchnames = sorted(set(item for item in notes))


note_to_int = dict((note, number) for number, note in enumerate(pitchnames))

network_input = []
network_output = []


for i in range(0, len(notes) - sequence_length, 1):
    sequence_in = notes[i:i + sequence_length]
    sequence_out = notes[i + sequence_length]
    network_input.append([note_to_int[char] for char in sequence_in])
    network_output.append(note_to_int[sequence_out])

n_patterns = len(network_input)


network_input = numpy.reshape(network_input, (n_patterns, sequence_length, 1))

network_input = network_input / float(n_vocab)

network_output = np_utils.to_categorical(network_output)
```

Рисунок 3.4 Фрагмент коду передбачення наступної ноти

Нарешті ми переходимо до проектування архітектури моделі. У нашій моделі ми використовуємо чотири різні типи шарів:

- Шари LSTM - це шар рекурентної нейронної мережі, який приймає на вхід послідовність і може повертати або послідовності (return_sequences = True), або матрицю.
- Випадаючі (Dropout) шари - це техніка регуляризації, яка полягає в установці частки вхідних одиниць в 0 при кожному оновленні під час навчання, щоб запобігти перепідгонці. Частка визначається параметром, що використовується у шарі[26].
- Щільні (Dense) шари - це так звані пов'язані шари нейронної мережі, в яких кожен вхідний вузол з'єднаний з вихідним вузлом[27].
- Активізаційний (Activation) шар визначає, яку функцію використовуватиме нейронна мережа для генерації або обчислення вихідного коду.

Тепер, коли у нас є деяка інформація про різні шари, які ми будемо використовувати, потрібно додати їх до моделі мережі. На рисунку 3.5 продемонстровано використання всіх шарів, які наявні в системі.

```
model = Sequential()
model.add(LSTM(
    256,
    input_shape=(network_input.shape[1], network_input.shape[2]),
    return_sequences=True
))
model.add(Dropout(0.3))
model.add(LSTM(512, return_sequences=True))
model.add(Dropout(0.3))
model.add(LSTM(256))
model.add(Dense(256))
model.add(Dropout(0.3))
model.add(Dense(n_vocab))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
```

Рисунок 3.5 Фрагмент коду реалізації архітектури моделі

Для кожного шару LSTM, Dense, а також Activation перший параметр – це кількість вузлів, які повинен мати шар. Для шару Dropout перший параметр – це частка вхідних одиниць, які мають бути відкинуті під час навчання.

Для першого шару, яким в нашому випадку є LSTM, ми маємо надати унікальний параметр під назвою input_shape. Ціль цього параметра - повідомити мережі форму даних, на яких вона буде навчатися.

Останній шар завжди повинен містити таку ж кількість вузлів, скільки різних виходів має наша система. Це гарантує, що вихідні дані мережі відповідатимуть нашим класам.

Загалом буде використовуватися мережа, що складається з трьох шарів LSTM, трьох шарів Dropout, двох шарів Dense та одного активаційного шару.

Для розрахунку втрат на кожній ітерації навчання ми використовуватимемо категоріальну перехресну ентропію, оскільки кожен із наших виходів належить лише одному класу, а у нас понад два класи для роботи. А для оптимізації нашої мережі ми будемо використовувати оптимізатор RMSprop, оскільки він є дуже гарним вибором саме для рекурентних нейронних мереж[28].

Після того як ми визначили архітектуру нашої мережі, настав час розпочати навчання. Для навчання мережі використовується функція model.fit() у Keras. Перший параметр – це список вхідних послідовностей, який ми підготували раніше, а другий – список відповідних їм виходів.

Щоб переконатися, що ми можемо зупинити навчання в будь-який час без втрати всієї роботи, потрібно використовувати контрольні точки. Контрольні точки моделі дозволяють нам зберігати ваги вузлів мережі в файл після кожної ітерації. На рисунку 3.6 продемонстровано реалізацію контрольних точок.

```
checkpoint = ModelCheckpoint(
    filepath, monitor='loss',
    verbose=0,
    save_best_only=True,
    mode='min'
)
callbacks_list = [checkpoint]
```

Рисунок 3.6 Програмна реалізація контрольних точок

Це дозволяє зупинити роботу нейронної мережі, в будь-який час, не переймаючись втратою даних. В іншому випадку нам довелося б чекати, поки мережа пройде всі ітерації, перш ніж ми отримали можливість зберегти дані у файл.

3.2. Генерація результату

Щоб використовувати нейронну мережу для генерування музики, необхідно привести її до того ж стану, що й раніше. При генерації буде використовуватися аналогічна модель системи з тими ж шарами. За винятком того, що замість навчання мережі ми завантажимо в модель дані, отримані в результаті навчання.

Оскільки в нашому розпорядженні є повний список нотних послідовностей, ми виберемо випадковий індекс у списку як початкову точку. Це дозволяє отримувати різні результати генерації при кожному запуску.

Тут також необхідно створити функцію відображення для декодування вихідних даних мережі. Ця функція перетворюватиме числові дані в категорійні, в нашому випадку - ноти.

Для кожної ноти, яку ми хочемо згенерувати, ми повинні надіслати до мережі послідовність. Перша послідовність, яку ми відправляємо - це послідовність нот з початковим індексом. Для кожної наступної послідовності, яку ми використовуємо як вхідні дані, ми видаляємо першу ноту з послідовності і вставляємо вихід попередньої ітерації в кінець послідовності. Даний принцип продемонстрований на рисунку 3.7.

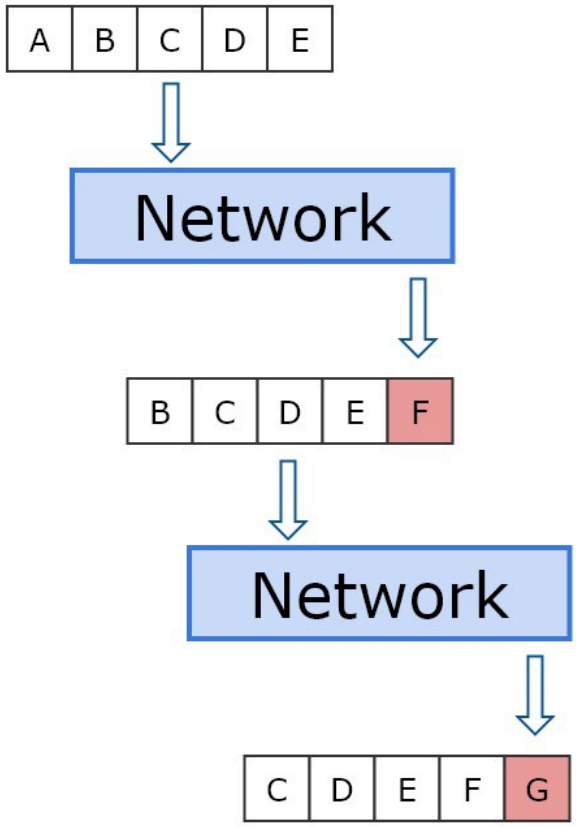


Рисунок 3.7 Принцип генерації нот починаючи з першої послідовності[29]

Щоб визначити найбільш імовірне передбачення вихідних даних мережі, ми отримуємо індекс найбільшого значення.

Потім ми збираємо усі виходи мережі в один масив.

Тепер, коли у нас є всі закодовані уявлення нот та акордів у масиві, ми можемо приступити до їх декодування та створення масиву об'єктів Note та Chord.

Якщо шаблон є акордом, ми маємо розбити рядок на масив нот. Потім ми перебираємо рядкову виставу кожної ноти і створюємо об'єкт Note для кожної з них. Потім ми можемо створити об'єкт Chord, який містить кожну з цих нот.

Якщо патерн є нотою, ми створюємо об'єкт Note, використовуючи рядкове уявлення висоти тону, що міститься в патерні.

Наприкінці кожної ітерації ми збільшуємо зсув на 0,5 і додаємо створений об'єкт до списку. Принцип такої реалізації продемонстровано на рисунку 3.8.

```
offset = 0
output_notes = []



for pattern in prediction_output:

    if ('.' in pattern) or pattern.isdigit():
        notes_in_chord = pattern.split('.')
        notes = []
        for current_note in notes_in_chord:
            new_note = note.Note(int(current_note))
            new_note.storedInstrument = instrument.Piano()
            notes.append(new_note)
        new_chord = chord.Chord(notes)
        new_chord.offset = offset
        output_notes.append(new_chord)

    else:
        new_note = note.Note(pattern)
        new_note.offset = offset
        new_note.storedInstrument = instrument.Piano()
        output_notes.append(new_note)


    offset += 0.5
```

Рисунок 3.8 Програмна реалізація додавання нот та зсуву

3.3. Демонстрація виконання та аналіз згенерованого результату

Після запуску програми на тренування, ми можемо бачити в консолі прогрес виконання. Це проілюстровано на рисунку 3.9.

```
Epoch 1/200
65/65 [==============================] - 207s 13s/step - loss: 4.9783
Epoch 2/200
65/65 [==============================] - 196s 13s/step - loss: 4.4859
Epoch 3/200
65/65 [==============================] - 203s 14s/step - loss: 4.1884
Epoch 4/200
65/65 [==============================] - 230s 15s/step - loss: 5.2579
Epoch 5/200
65/65 [==============================] - 256s 16s/step - loss: 5.1536
Epoch 6/200
65/65 [==============================] - 217s 15s/step - loss: 5.0019
Epoch 7/200
65/65 [==============================] - 224s 15s/step - loss: 4.7319
Epoch 8/200
 5/65 [=========>....................] - ETA: 2:18 - loss: 4.7026
```

Рисунок 3.9 Відображення процесу тренування в консолі

Внаслідок реалізації системи контрольних точок, можна перервати виконання в будь-який момент.

Після закінчення етапу тренування ми можемо перейти до другої частини і зайнятися генерацією кінцевого результату.

Після повного виконання програми, в директорії з’являється MIDI-файл. Його можна одразу прослухати, або за допомогою спеціальної програми (наприклад MuseScore) переглянути, як вихідний файл виглядає на нотному листі. На рисунку 3.9 продемонстровано нотне уявлення музичного файлу, який був згенерований за допомогою мережі LSTM. З першого погляду видно, що у ньому наявна певна структура.

Люди, які знаються на музиці і вміють читати нотну грамоту, можуть помітити, що на листі подекуди розкидані дивні ноти, які не зовсім підходять в конкретний момент. Це результат того, що нейронна мережа не може створювати

ідеальні мелодії. Проте навіть в такому вигляді, мелодія сприймається цілісною та завершеною.

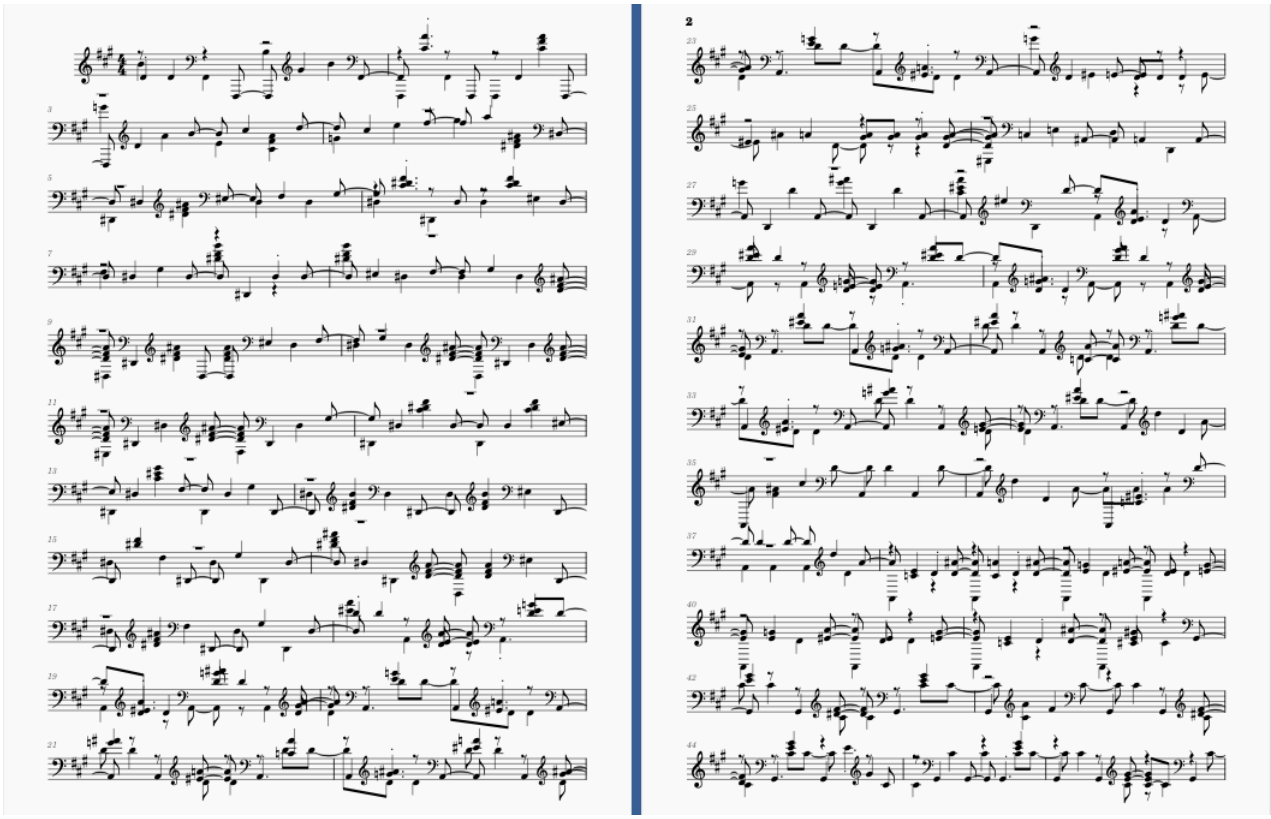


Рисунок 3.9 Файл згенерований нейронною мережею при перегляді за допомогою MuseScore

Звичайно, та реалізація, яка існує на даний момент, не підтримує різну тривалість нот, а також різні зсуви між ними. Для цього можна було б додати більшу кількість класів для різної тривалості та додати класи відпочинку, які б встановлювали період відпочинку між нотами.

Для досягнення кращих результатів при додаванні великої кількості класів також довелося б збільшити глибину мережі LSTM, а це в свою чергу вимагає значно потужнішої обчислювальної системи.

Також, було б добре додати фази початку та кінця. На даний момент, мережа, не має різниці між своїми частинами, тобто мережа не розрізняє моменти, коли одна частина закінчується, а інша починається. Такі доповнення

дозволять мережі генерувати фрагмент від початку до кінця, внаслідок чого музика буде відчуватися більш завершеною, а не закінчуватися раптово.

ВИСНОВКИ ДО РОЗДІЛУ 3

В цьому розділі було розглянуто основні етапи розробки системи генерації музики за допомогою рекурентних нейронних мереж. Було описано процес створення програми, а також архітектури моделі нейронної системи.

ВИСНОВКИ

За час виконання роботи був проведений аналіз предметної області, а також огляд існуючих рішень за темою дипломної роботи. Був проведений огляд існуючих технологій розробки.

Внаслідок проведеного аналізу було розроблено систему генерації музики за допомогою рекурентних нейронних мереж. Було детально розглянуто реалізацію системи та наведені приклади результатів виконання.

Внаслідок виконання роботи було розроблено систему, яка відповідає технічному завданню поставленому на дипломний проєкт.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Swift, Andrew. (May 1997), "A brief Introduction to MIDI", SURPRISE, Imperial College of Science Technology and Medicine, archived from the original on 30 August 2012, retrieved 22 August 2012
2. MIDI History:Chapter 6-MIDI Is Born 1980–1983. [Електронний ресурс] – Режим доступу до ресурсу: www.midi.org (дата звернення: 18.01.2020)
3. "What is MIDI?". Archived from the original on 16 June 2016. Retrieved 31 August 2016.
4. Lewis, "Creation by refinement: a creativity paradigm for gradient descent learning networks," IEEE 1988 International Conference on Neural Networks, 1988, pp. 229-233 vol.2, doi: 10.1109/ICNN.1988.23933.
5. Eck, D. and Schmidhuber, J. 2002. Learning the Long-Term Structure of the Blues. Proceedings of the 2002 International Conference on Artificial Neural Networks (ICANN).
6. Freund, Y.; Schapire, R. E. (1999). "Large margin classification using the perceptron algorithm" (PDF). Machine Learning. 37 (3): 277–296. doi:10.1023/A:1007662407062. S2CID 5885617.
7. Towards Data Science. Neural Networks For Music: A Journey Through Its History [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/neural-networks-for-music-a-journey-through-its-history-91f93c3459fb (дата звернення: 06.06.2022)
8. GitHub. Deep Learning for Music (DL4M) [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/ybayle/awesome-deep-learning-music (дата звернення: 07.06.2022)
9. OpenAI. Jukebox [Електронний ресурс] – Режим доступу до ресурсу: https://openai.com/blog/jukebox/ (дата звернення: 07.06.2022)
10. Silicon Luxembourg. AIVA: The Artificial Intelligence Composing Classical Music [Електронний ресурс] – Режим доступу до ресурсу:

https://www.siliconluxembourg.lu/aiva-the-artificial-intelligence-composing-classical-music/ (дата звернення: 07.06.2022)

11. Wikipedia. OpenAI [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/OpenAI (дата звернення: 07.06.2022)
12. Amper Music. Musical Algorithms [Електронний ресурс] – Режим доступу до ресурсу: https://ampermusic.zendesk.com/hc/en-us/articles/360023279774-Musical-Algorithms (дата звернення: 09.06.2022)
13. HashDork. 9 лучших языков программирования, используемых в ИИ. [Електронний ресурс] – Режим доступу до ресурсу: https://hashdork.com/ru/programming-languages-used-in-ai/ (дата звернення: 09.06.2022)
14. YOH. 10 BEST PROGRAMMING LANGUAGES FOR AI DEVELOPMENT [Електронний ресурс] – Режим доступу до ресурсу: https://www.yoh.com/blog/10-best-programming-languages-for-ai-development (дата звернення: 09.06.2022)
15. LogRocket. Best JavaScript machine learning libraries in 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://blog.logrocket.com/best-javascript-machine-learning-libraries-in-2021/ (дата звернення: 09.06.2022)
16. Venturebeat. 4 reasons to learn machine learning with JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://venturebeat.com/2021/04/23/4-reasons-to-learn-machine-learning-with-javascript/ (дата звернення: 09.06.2022)
17. Офіційний сайт Common Lisp. Welcome to Common-Lisp.net [Електронний ресурс] – Режим доступу до ресурсу: https://common-lisp.net/ (дата звернення: 09.06.2022)
18. Офіційний сайт R. What is R? [Електронний ресурс] – Режим доступу до ресурсу: https://www.r-project.org/about.html (дата звернення: 09.06.2022)
19. ZFORT БЛОГ. Лучший Язык Для Программирования Ии: Руководство Пользователя [Електронний ресурс] – Режим доступу до ресурсу:

https://www.zfort.com.ua/blog/luchshii-yazyk-dlya-programmirovaniya-ii-rukovodstvo-polzovatelya (дата звернення: 09.06.2022)

20. Towards Data Science. Why I Think Python is Perfect for Machine Learning and Artificial Intelligence [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/8-reasons-why-python-is-good-for-artificial-intelligence-and-machine-learning-4a23f6bed2e6 (дата звернення: 09.06.2022)
21. CitrusBug. Best Python Library For AI And ML [Latest] [Електронний ресурс] – Режим доступу до ресурсу: https://citrusbug.com/blog/python-libraries-for-machine-learning (дата звернення: 10.06.2022)
22. Analytics Vidhya. CNN vs. RNN vs. ANN – Analyzing 3 Types of Neural Networks in Deep Learning [Електронний ресурс] – Режим доступу до ресурсу: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/#h2_6 (дата звернення: 10.06.2022)
23. Towards Data Science. Applied Deep Learning - Part 4: Convolutional Neural Networks [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2#:~:text=The%20main%20advantage%20of%20CNN,CNN%20is%20also%20computationally%20efficient (дата звернення: 10.06.2022)
24. Machine Learning Mastery. An Introduction To Recurrent Neural Networks And The Math That Powers Them [Електронний ресурс] – Режим доступу до ресурсу: https://machinelearningmastery.com/an-introduction-to-recurrent-neural-networks-and-the-math-that-powers-them/ (дата звернення: 10.06.2022)
25. Colah's blog. Understanding LSTM Networks [Електронний ресурс] – Режим доступу до ресурсу: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (дата звернення: 10.06.2022)

26. Machine Learning Mastery. A Gentle Introduction to Dropout for Regularizing Deep Neural Networks [Електронний ресурс] – Режим доступу до ресурсу: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/ (дата звернення: 09.06.2022)
27. Analytics India Magazine. A Complete Understanding of Dense Layers in Neural Networks [Електронний ресурс] – Режим доступу до ресурсу: https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/#:~:text=one%20by%20one.-,What%20is%20a%20Dense%20Layer%3F,in%20artificial%20neural%20network%20networks (дата звернення: 10.06.2022)
28. Офіційний сайт Keras. Optimizers [Електронний ресурс] – Режим доступу до ресурсу: https://keras.io/api/optimizers/ (дата звернення: 10.06.2022)
29. Towards Data Science. How to Generate Music using a LSTM Neural Network in Keras. [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5 (дата звернення: 10.06.2022)

ДОДАТОК 1

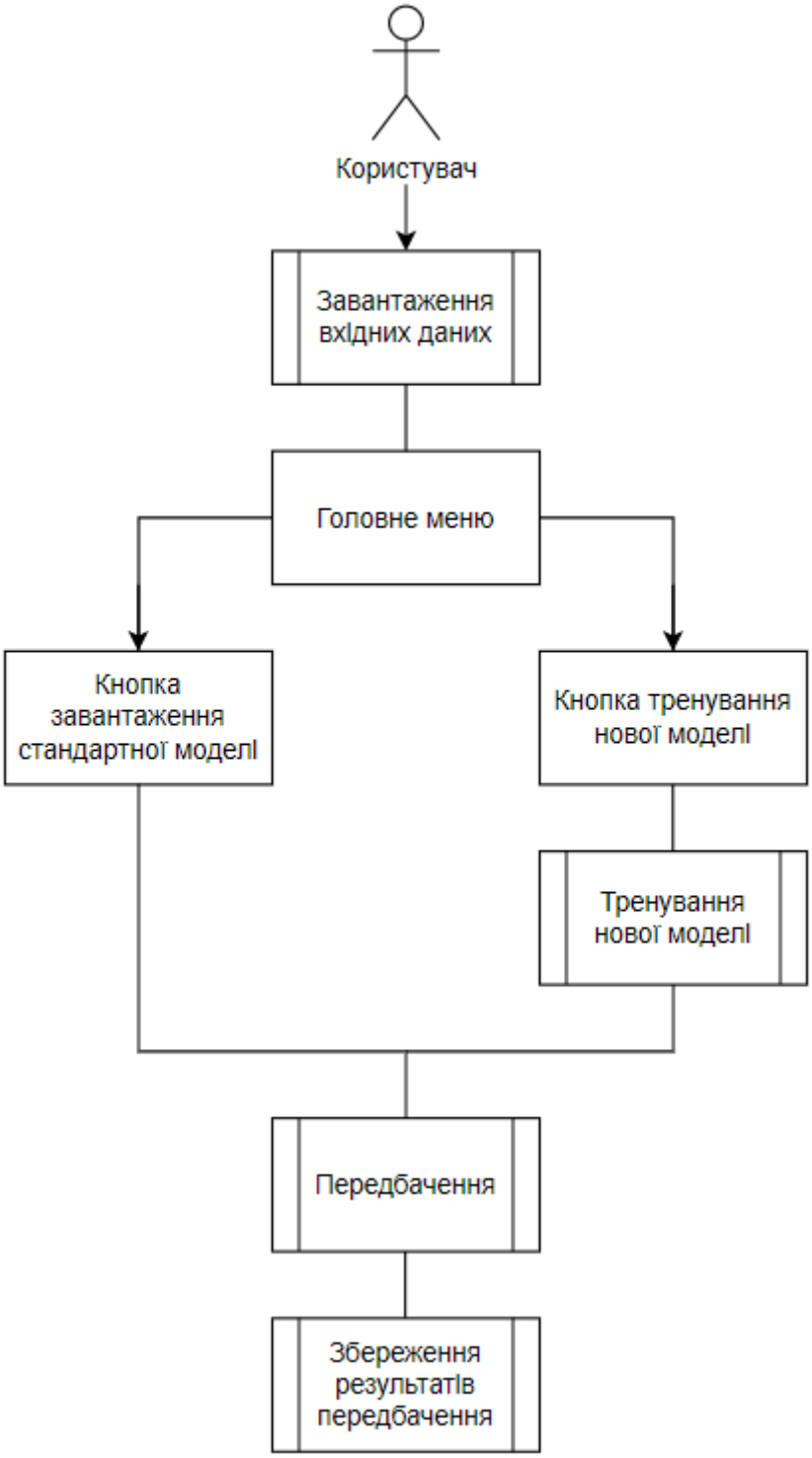
Система генерації музики за допомогою рекурентних нейронних мереж

Структурна схема програмного забезпечення

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.004 Д1	
		№ докум.	Підпис	Дата		
Розробив		Ковальков Д.П.			Система генерації музики за допомогою рекурентних нейронних мереж Структурна схема програмного забезпечення	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Кочура Ю.П.				
Н. Контр.		Сімоненко В. П.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-81
Затвердив						

ДОДАТОК 2

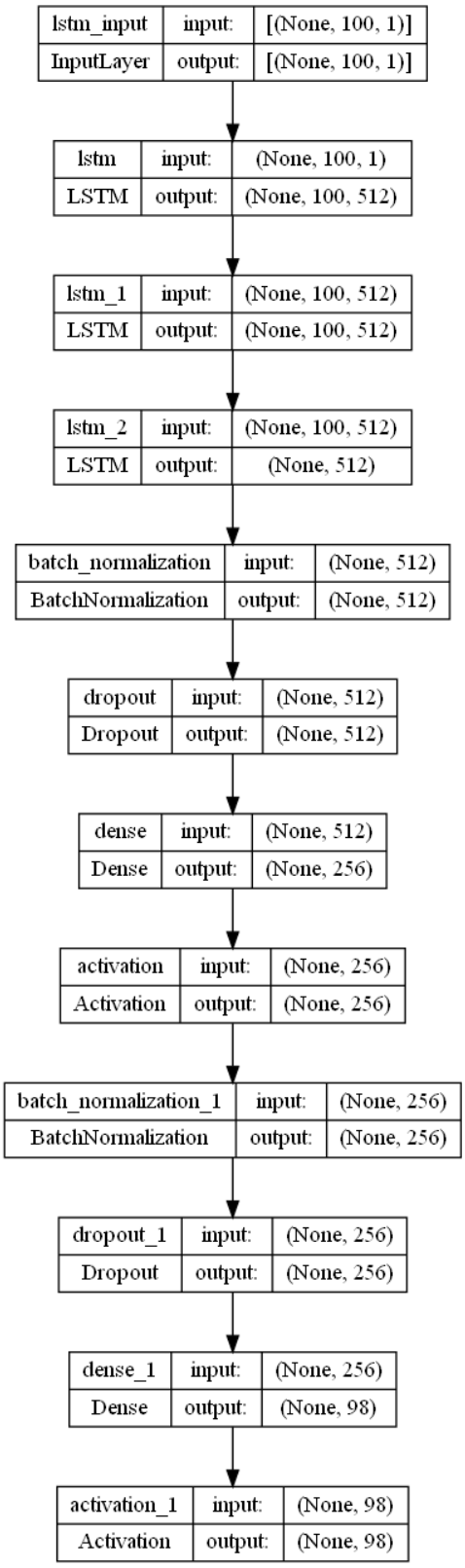
Система генерації музики за допомогою рекурентних нейронних мереж

Функціональна схема (обчислювальний граф моделі)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.005 Д2	
		№ докум.	*Підпис*	*Дата*		
Розробив		*Ковальков Д.П.*			*Система генерації музики за допомогою рекурентних нейронних мереж*	*Літ.* / *Аркуш* / *Аркушів*: *1* / *1*
Перевірив		*Кочура Ю.П.*				
Н. Контр.		*Сімоненко В. П.*			*Обчислювальний граф моделі*	*НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-81*
Затвердив						

ДОДАТОК 3

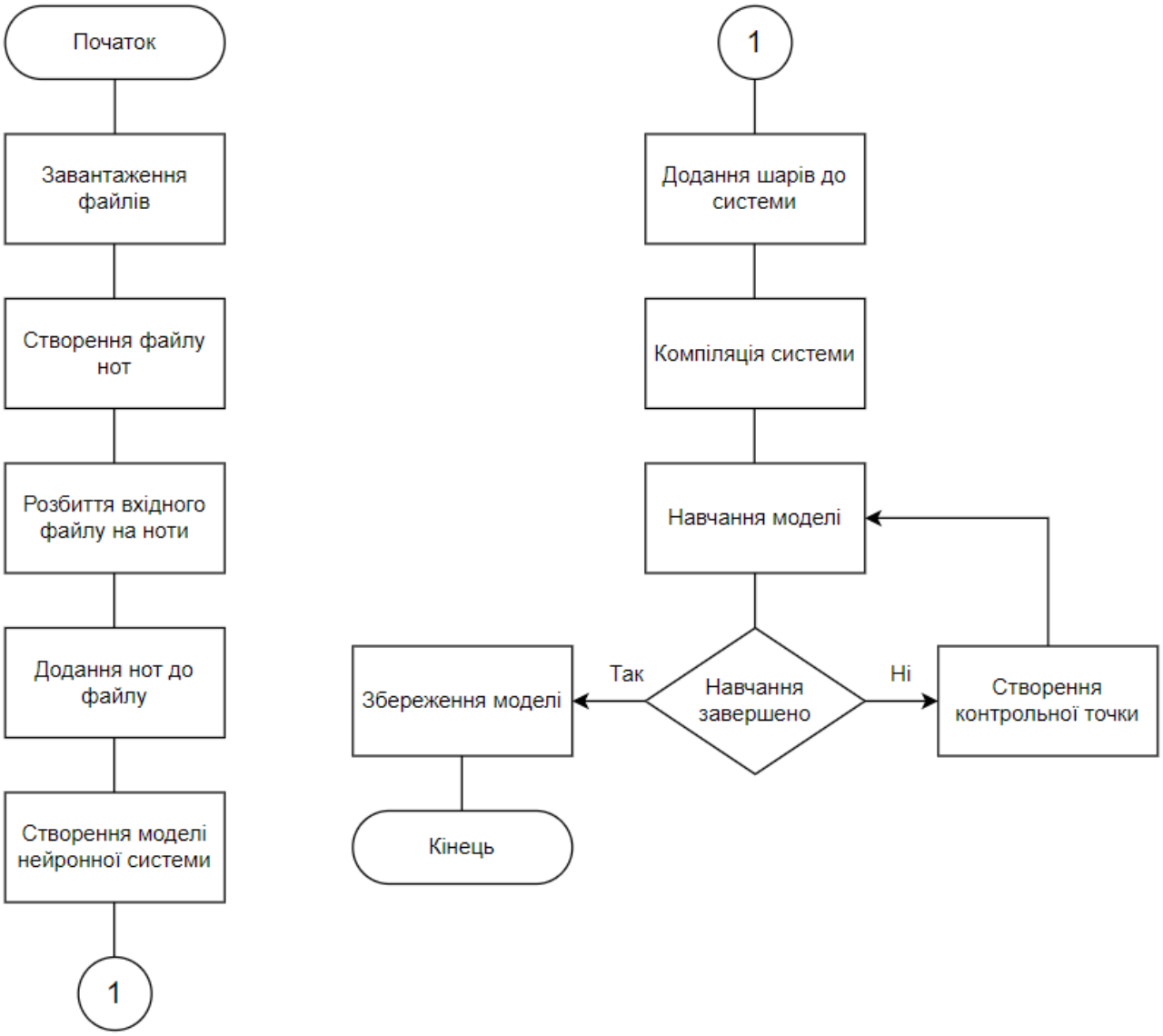
Система генерації музики за допомогою рекурентних нейронних мереж

Схема алгоритму

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.006 Д3	
		№ докум.	*Підпис*	*Дата*		
Розробив		*Ковальков Д.П.*			*Система генерації музики за допомогою рекурентних нейронних мереж*	*Літ.* / *Аркуш* / *Аркушів*
Перевірив		*Кочура Ю.П.*				*1* / *1*
Н. Контр.		*Сімоненко В. П.*			*Схема алгоритму*	*НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-81*
Затвердив						