Хмельницький національний університет
Факультет програмування
та комп'ютерних і телекомунікаційних систем
Кафедра інженерії програмного забезпечення

ДИПЛОМНА РОБОТА

Технологія розробки програмної системи для генерування музичних творів
Назва теми

із використанням штучного інтелекту

Рівень вищої освіти Другий (магістерський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДРІПЗ.150155.01.01.ПЗ

Виконав студент 2 курсу група ІПЗм-19-1 _____ І. Р.Бабич
Підпис Ініціали, прізвище

Керівник канд. техн. наук, доцент _____ О. М.Яшина
Підпис Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент _____ Г. І.Радельчук
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

7 грудня 2020 р.

Хмельницький 2020

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Програмування та комп'ютерних і телекомунікаційних систем
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Другий (магістерський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
02.09.2020 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ)

Бабичу Івану Руслановичу
Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи) Технологія розробки програмної системи для генерування музичних творів із використанням штучного інтелекту

Керівник проєкту (роботи) Яшина Оксана Миколаївна, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.09.2020 р. № 118
2. Строк подання студентом проєкту (роботи) на кафедру 01.12.2020 р.
3. Вихідні дані до проєкту (роботи) Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз відомих моделей, методів та засобів
2. Моделі та методи для вирішення задачі
3. Проектування програмного забезпечення для вирішення проблеми
4. Реалізація програмного забезпечення для вирішення проблеми
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди)

6. Консультанти розділів дипломного проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання « 01 » вересня 2020 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітки
1 Ознайомлення з предметною областю; формулювання мети та задач дослідження, визначення об'єкта та предмета дослідження; визначення структури дипломної роботи	01.09 – 07.09.2020	
2 Робота над розділом 1 дипломної роботи – вивчення літературних джерел; аналіз відомих моделей та засобів за темою роботи; висновки до розділу та постановка задачі	08.09 – 25.09.2020	
3 Робота над розділом 2 дипломної роботи – розробка моделей та методів вирішення поставленої задачі; висновки до розділу	26.09 – 10.10.2020	
4 Робота над науковими статтями	11.10 – 20.10.2020	
5 Робота над розділом 3 дипломної роботи – розробка алгоритмів та технологій, проектування ПЗ для вирішення поставленої задачі; висновки до розділу	11.10 – 26.10.2020	
6 Робота над розділом 4 дипломної роботи – програмна реалізація спроектованих рішень, результати експериментів, їх аналіз; висновки до розділу	27.10 – 15.11.2020	
7 Узгодження постановки задачі, отриманих результатів та висновків; написання вступу, загальних висновків, оформлення джерел посилання та додатків; оформлення пояснювальної записки та графічних матеріалів згідно вимог стандартів	16.11 – 30.11.2020	
8 Попередній захист дипломної роботи	Листопад (згідно графіка)	
9 Перевірка роботи на наявність плагіату; нормконтроль; брошурування пояснювальної записки; підготовка супровідних документів	01.12 – 04.12.2020	
10 Підготовка до захисту дипломної роботи	05.12 – 08.12.2020	

Студент _____ І. Р. Бабич
Підпис Ініціали, прізвище

Керівник проєкту (роботи) _____ О. М. Яшина
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема дипломної роботи: Технологія розробки програмної системи для генерування музичних творів із використанням штучного інтелекту.

Автор роботи: Бабич Іван Русланович.

Керівник роботи: Яшина Оксана Миколаївна.

Пояснювальна записка: 124 с., 30 рис., 4 табл., 4 дод., 15 джерел.

ШТУЧНИЙ ІНТЕЛЕКТ, ГЛИБИННЕ НАВЧАННЯ, МУЗИКА, PYTHON, TENSORFLOW.

Об'єктом дослідження є програмні системи генерування музичних творів із використанням штучного інтелекту.

Предметом дослідження є моделі та методи створення та вдосконаленння систем генерування музичних творів із використанням штучного інтелекту.

Метою дипломної роботи є створення моделі створення музики на основі вивчення штучного інтелекту теорії музики та аналізу музичних творів та удосконалення методу врахування введення даних користувача для генерування музичних творів із використанням штучного інтелекту.

Впровадження розробленої моделі, удосконаленого методу та програмного засобу дає можливість покращити процес генерування гармонійних поліфонічних музичних творів з врахуванням введення даних користувача.

Практична значимість отриманих результатів полягає в тому, що запропоновані метод та модель повинні надати більш широку можливість в налаштуванні системи тренування й генерування музичного твору, щоб користувач міг одержати контроль не тільки над більшим дотриманням теорії музики (краща структурованість твору, краща гармонійність мелодії, тощо), але й над більш оригінальною відтворюваністю мелодій.

4.12.2020

(дата)

(підпис)

ABSTRACT

Topic: AI-based Technology of Developing a Software System for Music Art Pieces Generation.

Author of work: Babich Ivan Ruslanovich.

Head of work: Yashyna Oksana Mykolayivna.

Explanatory note: 124 p., 30 pic., 4 tab., 4 appx., 15 sources.

ARTIFICIAL INTELLIGENCE, DEEP LEARNING, MUSIC, PYTHON, TENSORFLOW.

The object of this research are the software systems for generating musical works using artificial intelligence.

The subject of the research are the models and methods of creating and improving systems for generating musical works using artificial intelligence.

The purpose of this graduate work is to create a model of creating music based on the study of artificial intelligence of music theory and analysis of musical works and improve the method of taking into account the input of user data for generating musical works using artificial intelligence.

The implementation of the developed model, improved method and software makes it possible to improve the process of generating harmonious polyphonic music with the consideration of user input.

The practical significance of the obtained results is that the proposed method and model should provide a wider opportunity to configure the system of training and generating a musical work, so that the user can gain control not only over greater adherence to music theory (better structured work, better melody harmony, etc.), but also over a more original reproducibility of melodies.

4.12.2020

(date)

(signature)

ЗМІСТ

ВСТУП

В наш час при виконанні рутинних справ та турбот, з кожним днем невпинно зростає кількість інформаційних та психологічних навантажень на організм людини. Так як всі вони вимагають максимальної напруги функціональних систем, людині необхідні певні фактори, що оптимізують стан організму, для подальшої його адаптації. Одним з таких факторів є музика.

Музика є значним чинником в житті людини, так як вона здатна поліпшити самопочуття й активність людини, зменшити внутрішню напругу в її організмі.

Протягом всього часу існування людства, йому вдавалось відтворювати музичні твори за допомогою голосу, музичних інструментів, а в сучасних умовах – за допомогою програмного забезпечення генерування музичних творів.

Проте, в умовах технологічного та інформаційного розвитку в суспільстві, сучасні технології можуть вирішувати такі ж самі проблеми, як і людина, в тому числі й композицію та відтворювання музичних творів. Штучний інтелект є однією з галузей, що здатна вирішити таку проблему.

Штучний інтелект, як і машинне навчання в цілому, постійно розвиваються і тепер вони регулярно використовуються для таких задач, як класифікація (ідентифікація по зображенню), передбачення (погоди), а також найбільш сучасніші завдання, такі як переклад. Але все більшою сферою застосування методів глибинного навчання є генерування контенту. Вміст може бути різного роду: зображення, текст, музика тощо.

Проблема генерування музичних творів із використанням штучного інтелекту є доволі актуальною, оскільки аналіз зарубіжної науково-технічної літератури і патентного пошуку за темою дослідження показав, що багато науковців при розробці вирішували велику кількість задач, повязанні з генеруванням музичних творів, такі як:

– генерування музичного твору ex nihilo або на основі мінімальної сідової інформації (наприклад, початкова нота або високорівневий опис);

– варіація тривалості музичного твору;

– варіація контенту музичного твору;

– мелодійно-гармонійна взаємодія в музичному творі;

– структура музичного твору.

Однак, загального рішення усіх проблем не існує, так як стратегії та підходи до генерування музичних творів залежать від уявлення самих науковців та інженерів до створення музики. Крім того, немає жодних гарантій того, що поєднання різноманітних архітектур та стратегій дасть надійну та точну систему.

Аналіз також показав, що більшість з розглянутих існуючих методів генерування музичних творів не мають достатньої інтерактивності. Існують також і винятки, тому на їх основі можна здійснити вдосконалення інтерактивності програмної системи з користувачем.

Саме тому, в даній дипломній роботі окреслено задачі, що є найменш дослідженими зі сторони науковців та інженерів:

– оригінальність музичного твору – згенерований музичний твір з частин файлу інших музичних творів може містити низький творчий фактор і це є проблемою не тільки мистецького характеру, а й економічного, оскільки порушує питання авторських прав;

– інтерактивність користувача – більшість сучасних систем генерує занадто механічну музику, без наявності людського типу творчості. Певна інтерактивність із користувачем (людьми) може не тільки допомогти в плані виконання музичних завдань для якісного твору (композиція, контрапункт, гармонізація, аналіз, аранжування тощо), але й зміна користувачем певних параметрів надає варіативність звучання мелодії.

Визначені проблеми є пов’язаними між собою, коли інтерактивність користувача може вирішити питання оригінальності твору шляхом кооперації людини з програмною системою. Тому для користувача необхідно розробити систему, яка буде якомога гнучкішою в налаштуванні змін генерування музичного твору.

Дане дослідження є актуальним для підтримки музикантів (здійснення композиції, аналізу, гармонізації, аранжування, продюсування, міксінгу тощо) та

може підвищити ефективність створення музичних творів із співпрацею зі сторони програмної системи та музиканта.

Метою дипломної роботи є створення моделі та удосконалення методу врахування введення даних користувача для генерування музичних творів із використанням штучного інтелекту.

Об’єктом дослідження є програмні системи генерування музичних творів із використанням штучного інтелекту.

Предметом дослідження є моделі та методи створення та вдосконаленння систем генерування музичних творів із використанням штучного інтелекту.

Можна виокремити наступні завдання:

– здійснити аналіз існуючих підходів генерування музичних творів;

– розробити модель вивчення штучного інтелекту теорії музики та аналізу музичних творів;

– вдосконалити метод обробки музичних треків шляхом включення введення більше користувацьких даних;

– розробити програмну реалізацію системи генерування музичних творів із використанням штучного інтелекту;

– провести тестування отриманих результатів.

Для досягнення мети використані наступні методи:

– аналіз (вивчення існуючих моделей та методів для більшого розуміння поставленої проблеми);

– опис (характеристика існуючих моделей та методів, що розглядались протягом цієї роботи),

– порівняння (вибір найоптимальніших підходів до вирішення проблеми цієї роботи),

– абстрагування (зведення процесів роботи системи в математичні об’єкти);

– моделювання (абстрагування функціональних процесів та самої системи);

– тестування (перевірка на ефективність отриманих результатів).

Наукова новизна отриманих результатів: розроблена модель створення музики на основі вивчення штучного інтелекту теорії музики та аналізу сучасних музичних творів, а також удосконалено метод обробки музичних треків із врахуванням введення даних користувача.

Отримані результати можуть бути застосовані при спрощеному генеруванні музичних творів для поліпшення психологічної напруги людини без застосування зайвих людських та інших ресурсів, що принесе практичну користь у шоу-бізнесі, та й в музичному бізнесі в цілому.

За темою дипломної роботи опублікована одна стаття у іноземному науковому виданні [1] та тези доповіді до наукової конференції [2].

1 АНАЛІЗ ВІДОМИХ МОДЕЛЕЙ, МЕТОДІВ ТА ЗАСОБІВ

1.1 Аналіз моделей у галузі генерування музичних творів із використанням штучного інтелекту

Було здійснено аналіз усіх можливих моделей нейронних мереж, які б могли автоматизувати процес генерування музики та вирішити проблему цього проекту.

Після здійснення пошуку джерел, які вирішували схожу проблему, до уваги приходять такі типи мереж, які підходять до вирішення проблематики цієї роботи:

– приховані марковські моделі;

– моделі глибинного навчання.

Було здійснено детальний аналіз кожного з типу моделей для того, щоб здійснити вибір найоптимальнішого варіанту.

До глибинного навчання приховані марковські моделі були стандартом для генеративних моделей аудіо/музики, оскільки процеси, що стоять за марковськими моделями, дуже підходять для композиції музичних творів [3]. Модель Маркова необов’язково «тренується» так само, як модель глибинного навчання або будь-яка модель з галузі машинного навчання. У той час як моделі машинного навчання, є «засновані на моделі», модель Маркова є більш «заснованою на екземплярах».

Було виділено переваги й недоліки використання марковських моделей згідно робіт [4]. Переваги марковських моделей:

– простота в концептуальності;

– проста реалізація та простий алгоритм навчання, оскільки модель є таблицею ймовірностей переходу;

– тренування на кількох прикладах;

– марковські моделі – це операційні моделі (автомати), на яких може бути прикріплений певний контроль над генерацією.

Недоліки марковських моделей є такі:

– їхній перший порядок (тобто, враховуючи лише попередній стан) не охоплює довгострокові тимчасові структури;

– їхній n-ий порядок (враховуючи n попередніх станів) можливий, але він потребує розмір «вибухаючої» навчальної підготовки та призводить до плагіату;

– погане узагальнення.

Причиною використання глибинного навчання для створення музичного контенту є його загальність. На відміну від рукотворних моделей, таких як система генерації музики, що базується на граматиці [5] або система, що базується на правилах [6], глибинне навчання може бути агностичним, оскільки воно вивчає модель із довільного корпусу музики. Як заявляли Ф'єбрінк та Карамійо [7], особливості глибинного навчання можуть бути такими:

– воно може зробити генерування музичного твору більш можливим, коли бажаний додаток є занадто складним, щоб його можна було описати аналітичними рецептурами або ручним проектуванням грубої сили;

– алгоритми тренування глибинних мереж часто менш крихкі, ніж розроблені вручну набори правил, і вивчені правила узагальнюють до нових контекстів, в яких вхідні дані можуть змінюватися.

Більше того, на відміну від тих же моделей, що базуються на правилах або граматиці, глибинне навчання ефективно обробляє неструктуровані дані, з яких його ієрархія шарів витягуватиме подання вищого рівня, адаптовані до завдання.

Переваги глибинних нейронних мереж:

– фіксація різних типів відносин, контекстів та закономірностей;

– можливість тренування з довгостроковими залежностями та залежностями високого порядку;

– краще узагальнення за допомогою розподілених подань.

Недоліки глибинних нейронних мереж:

– хоча й наявна простота в концептуальності, але оптимізовані реалізації поточних архітектур глибинних мереж можуть бути складними і потребують значного налаштування;

– необхідність у наданні великої кількості наборів даних, щоб мережа могла мати можливість добре тренуватися;

– відсутність прямого управління, яке могло б підключатись (так як дані мережі є генеративними моделями з розподіленим поданням).

Оскільки реалізації глибинного навчання вже цілком розвинені і доступна велика кількість прикладів, вони на сьогоднішній день досить популярні на відміну від застарілих марковських моделей.

Щоб зробити остаточний вибір, було виконано тестування двох типів моделей згідно швидкодії. Відповідно до публікації [8], був проведений експеримент, який передбачав створення моделей Маркова для генерації музики, а потім створення інших моделей, що використовують алгоритми глибинного навчання. Ці моделі проходили тренування на основі наборів даних з MIDI-файлів фортепіанної музики різних композиторів, таких як Бах, Бетховен, Шопен, Моцарт тощо.

В результаті експерименту, зроблено висновок, що мережі глибинного навчання виконують набагато точніше (38%), ніж приховані марковські моделі (33%).

Хоча й тести, що були дані для прихованих марковських моделей, мали відносно слабку унікальність мелодії, вони показали непогані результати точності структури музичного твору в класах тестуючих моделей (рисунок 1.1).

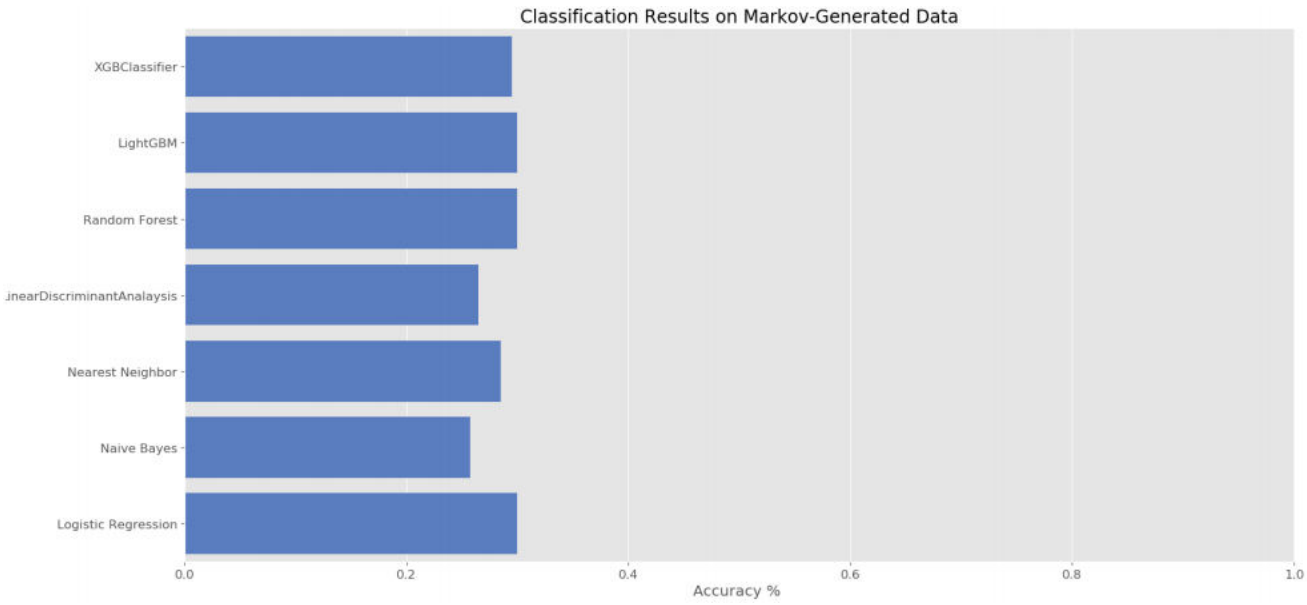


Рисунок 1.1 – Результати роботи алгоритмів на основі даних прихованих марковських моделей

Тести, що виконувались для моделей глибинного навчання, дали більш конкретну та цілеспрямовану унікальність мелодії, що призвело до вищого загального показника точності структури музичного твору (рисунок 1.2).

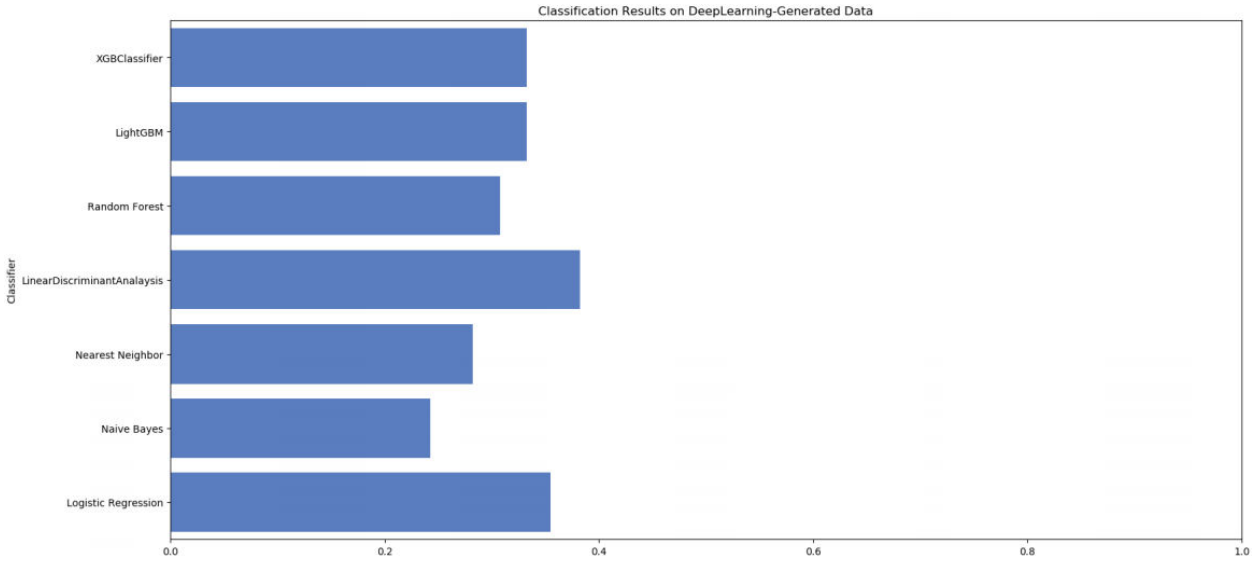


Рисунок 1.2 – Результати роботи алгоритмів на основі даних мереж глибинного навчання

Після того, як було обрано необхідний тип мереж для вирішення проблеми дипломної роботи, було проаналізовано моделі глибинного навчання та їх архітектури, щоб визначити, які саме з них найоптимальніше підходять для розроблення власної моделі створення музики на основі вивчення штучного інтелекту теорії музики та аналізу музичних творів.

В основу опису та аналізу включено особливості їхнього тренування нейронної мережі, так як це відіграє велику роль для запланованого в даній роботі, як тренування для якісного гармонізованого музичного твору та можливість втручання користувача для введення своїх даних.

Так як існує велика кількість архітектур глибинного навчання, було здійснено аналіз усіх основних видів архітектур, що можуть бути використані для генерування музичних творів та вирішення поставлених проблем.

Було виділено такі основні типи архітектур:

– нейронні мережі прямого поширення;

– автокодувальник;

– обмежена машина Больцмана;

– рекурентні нейронні мережі.

Нейронна мережа прямого поширення (багатошарова нейронна мережа) є сукупністю послідовних шарів основних будівельних блоків:

– перший рівень, що складається з вхідних вузлів, є вхідним шаром;

– останній рівень, що складається з вихідних вузлів, є вихідним шаром;

– будь-який шар між вхідним та вихідним шаром є прихованим шаром.

Поєднання прихованого шару та нелінійної функції активації робить нейронну мережу універсальним апроксиматором, здатним подолати обмеження лінійної сепарації.

У випадку практичних ілюстрацій архітектур нейронних мереж, щоб спростити фігури, вузли зміщення дуже рідко ілюструються. З подібною метою одиниці суми та одиниці функції активації також майже завжди опускаються, що призводить до більш абстрактного вигляду, такого як на рисунку 1.3.

На етапі тренування обчислення похідних стає дещо складнішим ніж для базового будівельного блоку (без прихованого шару).

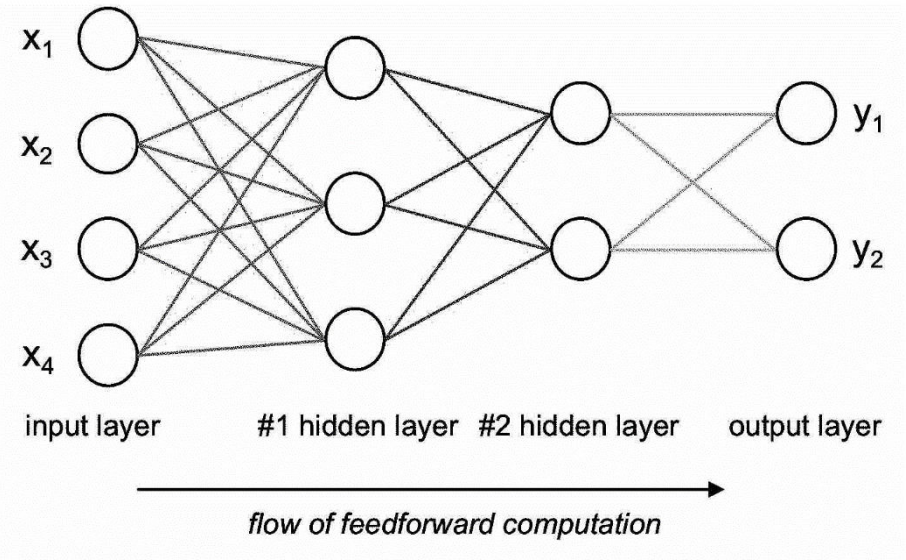


Рисунок 1.3 – Приклад спрощеної мережі прямого поширення

Метод зворотного поширення помилки є стандартним методом оцінки похідних (градієнтів) для нейронної мережі прямого поширення. Він базується на принципі ланцюгового правила [9], щоб оцінити внесок кожного вагового коефіцієнта в остаточну помилку прогнозування, тобто вартість.

Необхідно звернути увагу, що в найпоширенішому випадку функція витрат нейронної мережі прямого поширення не є опуклою, що означає, що може бути кілька локальних мінімумів.

Градієнтний спуск, як і інші більш складні евристичні методи оптимізації, не гарантує досягнення глобального оптимуму. Але на практиці розумна конфігурація моделі (зокрема, її гіперпараметри) та налагоджена евристика оптимізації, така як стохастичний градієнтний спуск, призведуть до точних рішень.

Автокодувальник – це нейронна мережа з одним прихованим шаром і з додатковим обмеженням: кількість вихідних вузлів дорівнює кількості вхідних вузлів. Вихідний рівень фактично відображає вхідний рівень. Це показано на рисунку 1.4 з його особливим симетричним аспектом форми діаболо (або пісочного годинника).

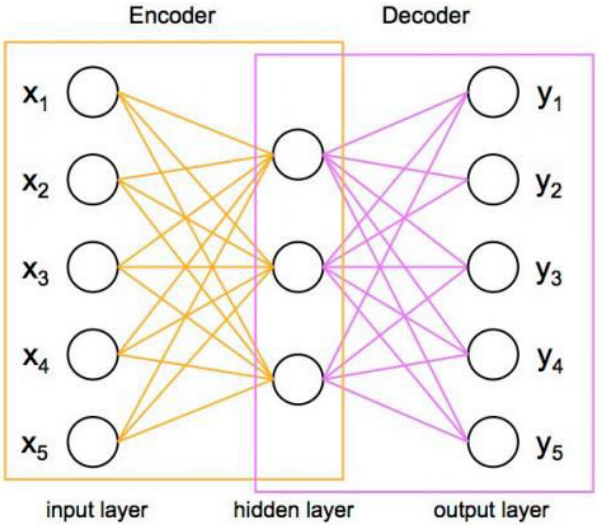


Рисунок 1.4 – Архітектура автокодувальника

Тренування автокодувальника представляє випадок безконтрольного навчання, оскільки приклади не містять жодної додаткової позначкової інформації (ефективне значення або клас, який передбачається). Але справа у

тому, що це реалізується із застосуванням звичайних контрольованих методик навчання, представляючи вихідні дані, рівні вхідним. На практиці автокодувальник намагається вивчити функцію ідентифікації. Оскільки прихований шар зазвичай має менше вузлів, ніж вхідний, компонент кодера (показаний жовтим на рисунку 1.4) повинен стискати інформацію, тоді як декодер (показаний фіолетовим) повинен якомога точніше реконструювати початкову інформацію. Це змушує автокодувальника виявляти значущі (розрізнювальні) функції для кодування корисної інформації у вузли прихованого шару (їх також називають прихованими змінними). Тому автокодувальники можуть використовуватися для автоматичного вилучення високорівневих функцій [10]. Сукупність видобутих функцій часто називають вбудовуванням. Натренувавшись, для вилучення функцій із введення потрібно просто передати вхідні дані та зібрати активації прихованого рівня (значення прихованих змінних).

Іншим цікавим використанням автокодувальників є високоякісний контроль генерації вмісту. Приховані змінні автокодувальника складають компактне зображення загальних рис вивчених прикладів. Екземпляром цих прихованих змінних та декодуванням вбудовування ми можемо створити новий музичний вміст, що відповідає значенням прихованих змінних.

Обмежена машина Больцмана (RBM) є генеративною стохастичною мережею, яка може вивчити розподіл ймовірностей за своїм набором входів. Її назва походить від того, що вона є обмеженою формою загальної машини Больцмана [11], названої на честь розподілу Больцмана в статистичній механіці, яка використовується в якості функції вибірки. Архітектурні обмеження RBM (рисунок 1.5) полягають у тому, що:

– він організований по шарах, як і для мережі прямого поширення або автокодувальника, а точніше два шари – видимий шар (аналог вхідного й вихідного шару автокодувальника) та прихований шар (аналог прихованого шару автокодувальника);

– як і для стандартної нейронної мережі, не може бути зв’язків між вузлами в межах одного рівня.

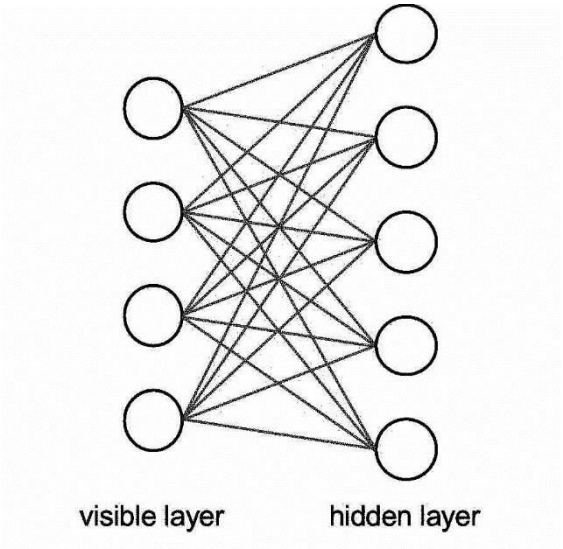


Рисунок 1.5 – Архітектура обмеженої машини Больцмана

RBM має певну схожість по ціллям з автокодувальником. Однак є кілька важливих відмінностей:

– RBM не має вихідних даних – вхідні дані також виконують роль вихідних;

– RBM є стохастичним (і, отже, не детермінованим, на відміну від мережі прямого поширення або автокодувальника);

– RBM тренується під навчанням без учителя, з певним алгоритмом (з назвою контрастивна дивергенція), тоді як автокодувальник навчається із застосуванням стандартного методу навчання з учителем, з тими ж даними, що і вхідні та вихідні дані;

– значення, якими маніпулюють, є логічними значеннями.

RBM – це архітектура, присвячена для навчальних розподілів. Більше того, він може ефективно навчатися лише на кількох прикладах. Для систем генерування музичних творів це цікаво для вивчення та створення акордів, оскільки комбінаторний характер можливих нот, що утворюють акорд, є великим, а кількість прикладів зазвичай невелика.

Навчання RBM має деяку схожість з навчанням автокодувальника з тією практичною різницею, що через відсутність частини декодувальника RBM буде чергувати два етапи:

– крок вперед – кодувати вхідний (видимий шар) у прихований шар, роблячи прогнози щодо активації вузла прихованого шару;

– крок назад – декодувати/реконструювати вхід (видимий шар), роблячи прогнози щодо активації вузла видимого шару.

Тут процес реконструкції є прикладом породжувального навчання (а не розрізнювального навчання, як для автокодувальників, яке базується на регресії).

Після завершення навчальної фази на етапі генерування можна взяти семпл з моделі шляхом випадкової ініціалізації вектора видимого шару *v* (дотримуючись стандартного рівномірного розподілу) та запуску вибірки до конвергенції. З цією метою приховані вузли та видимі вузли по черзі оновлюються.

На практиці конвергенція досягається тоді, коли енергія стабілізується. Енергія конфігурації (пари видимого та прихованого шарів) виражається у формулі (1.1):

$$E(v, h) = -a^T v - b^T h - v^T W h \qquad (1.1)$$

де v і h – вектори стовпців, що представляють видимий та прихований шари;

W – матриця ваг, пов'язана із зв'язками між видимими та прихованими вузлами;

a і b – вектори стовпців, що представляють ваги зміщення для видимих та прихованих вузлів, причому a^T та b^T є їх відповідними транспозиціями у вектори рядків

v^T – транспонування v у векторний рядок.

Насправді існують три можливості для природи змінних RBM (одиниці, видимі чи приховані шари):

– логічні або Бернуллі – це є випадок стандартного RBM, в яких одиниці (видимі та приховані) є логічними, з розподілом Бернуллі;

– мультинулі – розширення з одиницями мультинулі, тобто з більш ніж двома можливими дискретними значеннями;

– безперервний – інше розширення з безперервними одиницями, що приймає довільні дійсні значення (зазвичай в межах [0,1]).

Рекурентна нейронна мережа (RNN) є нейронною мережою прямого поширення, розширена повторюваними сполученнями для вивчення серії предметів (наприклад, мелодії як послідовності нот). Вхідні дані RNN – це елемент x_t послідовності, де t являє собою індекс або час, а очікуваний результат – наступний елемент x_{t+1}. Іншими словами, RNN буде натренований на передбачання наступного елементу послідовності.

Для цього вихід прихованого шару повертається до себе як додатковий вхід (із специфічною відповідною матрицею ваги). Таким чином, RNN може тренуватися не тільки на основі поточного елемента, але і на основі свого попереднього власного стану, а отже, рекурсивно, на цілої попередньої послідовності. Отже, RNN може вивчати послідовності, зокрема тимчасові послідовності, як у випадку з музичним змістом.

Приклад RNN (з двома прихованими шарами) показаний на рисунку 1.6.

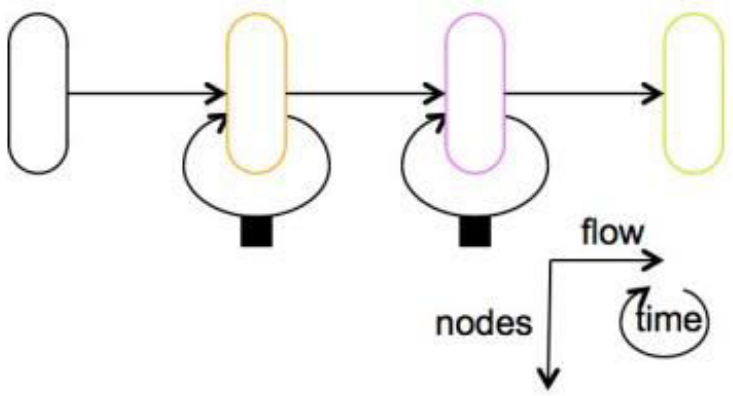


Рисунок 1.6 – Архітектура рекурентних нейронних мереж (складені)

Повторні зв’язки сигналізуються суцільним квадратом, щоб відрізнити їх від стандартних зв’язків. Версія візуального подання, що розгорнулася, наведена на рисунку 1.7 з новою діагональною віссю, що представляє часовий крок, для того, щоб проілюструвати значення попереднього кроку кожного шару (тоншим і світлішим кольором).

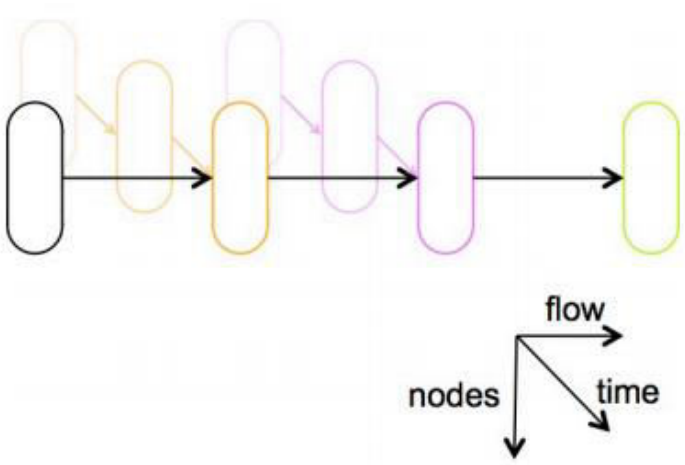


Рисунок 1.7 – Архітектура рекурентних нейронних мереж (розгорнуті)

Як і для стандартних з’єднань (показано жовтими суцільними лініями), періодичні зв’язки (показані фіолетовими пунктирними лініями) повністю з’єднують (із заданою матрицею ваги) всі вузли, що відповідають вузлам попереднього кроку, до вузлів, що відповідають поточному кроку, як показано на рисунку 1.8.

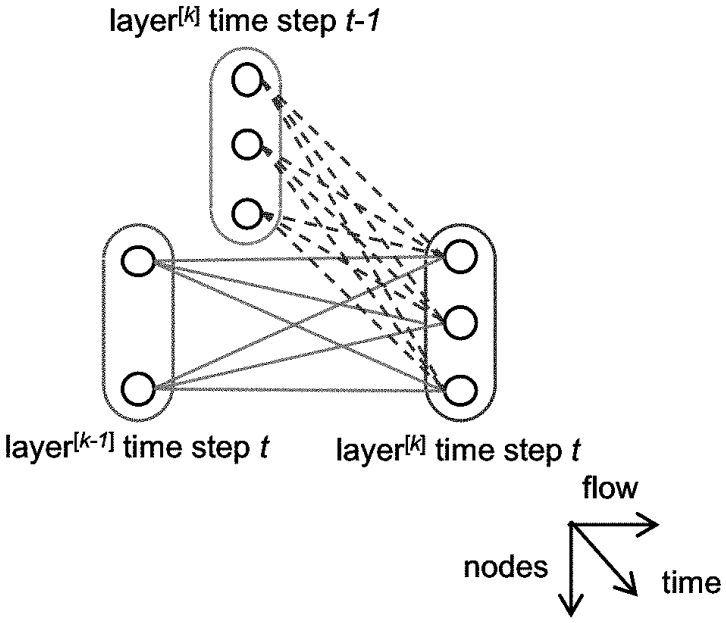


Рисунок 1.8 – Стандартні зв’язки проти повторних (розгорнуті)

RNN може тренувати розподіл ймовірностей за послідовністю, навчившись передбачати наступний елемент на кроці часу *t* у послідовності як умовний

розподіл ймовірностей $P(s_t|s_t - 1, ..., s_1)$, також позначений як $P(s_t|s_{<t})$, тобто розподіл ймовірностей $P(s_t)$ з урахуванням усіх попередніх елементів, що генерують s_1, s_2, ..., s_{t-1}. Таким чином, рекурентні мережі є кращими у вивченні послідовностей, і тому регулярно використовуються для природної обробки тексту та для генерації музики.

Рекурентна мережа не тренується точно так само, як мережа прямого поширення. Ідея полягає в тому, щоб представити приклад елемента послідовності (наприклад, ноту в мелодії) як вхідний x_t і наступний елемент послідовності (наступну ноту) x_{t+1} як вихідний y_t. Це навчить рекурентну мережу прогнозувати наступний елемент послідовності.

На практиці RNN рідко тренується як елемент за елементом, але з послідовністю як вхідним сигналом і такою самою послідовністю, зміщеною вліво на один крок або елемент, як вихід (рисунок 1.9). Тому рекурентна мережа буде тренуватись, щоб передбачати наступний елемент для всіх успішних елементів послідовності.

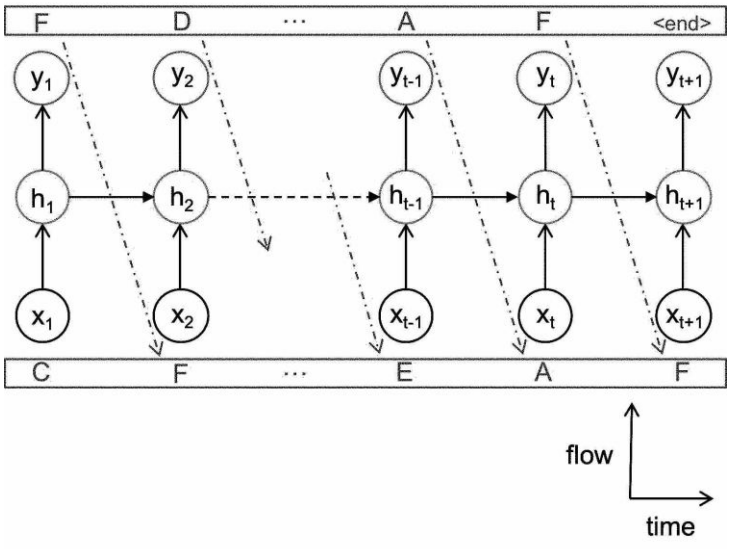


Рисунок 1.9 – Навчання рекурентної нейронної мережі

Алгоритм зворотного поширення помилки для обчислення градієнтів нейронних мереж прямого поширення, був розширений до алгоритму зворотного поширення помилки через час (BPTT) для рекурентних мереж. Інтуїція полягає в

розгортанні RNN у часі та розгляді впорядкованої послідовності пар вводу – виводу, але з кожною розгорнутою копією мережі, що використовує однакові параметри, а потім у застосуванні стандартного алгоритму зворотного поширення помилки [12].

Зазвичай рекурентні мережі мають вихідний рівень, ідентичний його вхідному шару, оскільки рекурентна мережа передбачає наступний елемент, який буде ітеративно використовуватися як наступний вхід рекурсивним способом для створення послідовності.

Навчання рекурентної мережі зазвичай розглядається як випадок контрольованого навчання, оскільки для кожного предмета наступний предмет подається як очікуваний прогноз, хоча це не додаткова позначувальна інформація (ефективне значення або клас, який передбачається), а лише періодична інформація про наступний елемент (суттєво присутня в послідовності).

Рекурентні нейронні мережі зазвичай страждають від тренувальної проблеми, спричиненої труднощами оціночних градієнтів, оскільки при зворотному поширенні помилки через часове повернення призводить до повторюваних множень і, таким чином, може призвести до надмірного посилення або мінімізації ефектів. Ця проблема була адресована та вирішена за допомогою архітектури довгої короткочасної пам’яті (LSTM). Оскільки рішення було досить ефективним, LSTM став фактичним стандартом для рекурентних мереж.

Ідея LSTM полягає в захисті інформації в комірках пам’яті в межах «блоку», захищеному від стандартного потоку даних рекурентної мережі. Рішення про запис, читання та забуття (стирання) значень комірок у «блоці» виконуються шляхом відкривання або закриття «вентилів» і виражаються на чітко вираженому контрольному рівні (метарівні), одночасно вивчаючись під час тренувального процесу. Отже, кожні «вентилі» модулюються ваговим параметром, і, отже, є придатними для зворотного поширення помилки та стандартного тренувального процесу. Іншими словами, кожен блок LSTM дізнається, як підтримувати свою пам’ять як функцію введення, щоб мінімізувати втрати.

Концептуальний вигляд комірки мережі довгої короткочасної пам’яті показаний на рисунку 1.10.

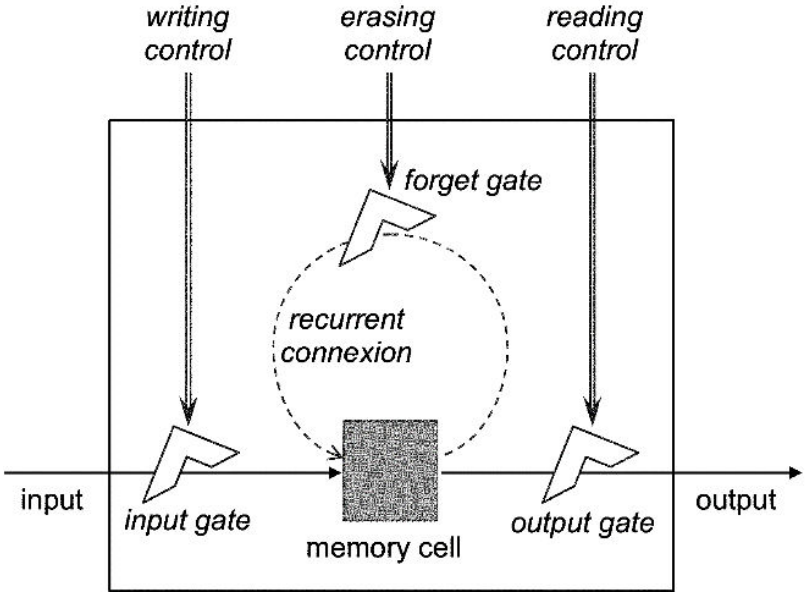


Рисунок 1.10 – Архітектура LSTM (концептуальна)

1.2 Аналіз методів у галузі генерування музичних творів із використанням штучного інтелекту

Далі для того, щоб вдосконалити метод обробки музичних треків для врахування введення даних користувача, необхідно детально проаналізувати низку методів, та як вони застосовуються до поданих в попередньому розділі архітектур, і обрати з них найоптимальніший для вирішення проблеми цієї роботи.

В кінці підрозділу методи та архітектури будуть порівняні між собою в табличному вигляді і ті, які вирішують найбільшу кількість проблем (в тому числі й ті, що необхідні для цієї роботи), будуть обрані для модифікації або буде обрано створити власну модель або метод.

Варто зрозуміти, що загального рішення не існує, і тому необхідно підбирати метод та модель щодо своїх критеріїв на певне уявлення реалізації системи генерування музичних творів.

У нашому випадку штучний інтелект повинен виконувати такі властивості:

– інваріантність у часі, мережа може генерувати необмежено довгі музичні твори, тому вона повинна бути однаковою для кожного часового кроку;

– інваріантність у нотах, музичний твір можна вільно транспонувати вгору-вниз, і він залишається принципово незмінним, структура нейронної мережі буде майже однаковою для кожної ноти;

– поліфонічна мелодія, відтворення більше одної ноти за крок у часі;

– втручання користувача у введення даних для зміни генерування музичного твору.

Найбільш прямим та простим методом генерування музичного твору є використання задачі передбачення або класифікації нейронної мережі для створення музики.

Розглянемо наступну мету: для цієї мелодії ми хочемо згенерувати супровід, наприклад, контрапункт. Ми розглянемо набір даних, кожен з яких є парою (мелодія, мелодія контрапункту).

Потім ми тренуємо архітектуру нейронних мереж прямого поширення в контрольованому типі навчання на цьому наборі даних. Після навчання ми можемо вибрати довільну мелодію та направити її в архітектуру, щоб створити відповідний супровід контрапункту у стилі набору даних. Виробництво завершується в один крок подальшої обробки.

Такий метод генерації музики можна назвати одноетапним прямопоширювальним методом. Прикладом є система символічного створення музичного супроводу MiniBach Chorale.

Для цього прикладу розглянемо наступну мету: створити контрапунктний супровід до цієї мелодії для сопранового голосу за допомогою трьох відповідних партій, що відповідають альта, тенору та басу. В якості корпусу ми використаємо набір поліфонічних хоралів Дж. С. Баха. Оскільки необхідно, щоб ця перша вступна система була простою, ми розглядаємо лише чотири міри довгих уривків з корпусу. Набір даних побудований шляхом вилучення всіх можливих уривків із довжиною чотири такти з оригінальних 352 хорів, також транспонованих у всі можливі ключі. Після тренування на цьому наборі даних система може бути

використана для формування трьох голосів контрапункту, що відповідають довільній мелодії довжиною чотири такти, що надається як вхід. Це захоплює практичні твори Дж. С. Баха, який обрав різні мелодії для сопрано і склав три додаткові голосові мелодії (для альта, тенору та басу) у контрапунктній манері.

По-перше, потрібно вирішити вхідні та вихідні подання. Було представлено чотири такти музики 4/4. І вхідні, і вихідні подання є символічними, типу піаніно, з одним гарячим кодуванням для кожного голосу, тобто мульти – унітарним кодуванням для вихідного подання. Три перші голоси (сопрано, альт і тенор) мають 20 можливих нот плюс додатковий маркер для кодування утримання, тоді як останній голос (бас) має 27 можливих нот плюс символ утримання. Квантування часу (значення кроку часу) встановлюється на шістнадцятій ноті, яка є мінімальною тривалістю ноти, що використовується в корпусі. Вхідне подання має розмір 21 можливої ноти × 16 часових кроків × чотири міри, тобто 21 × 16 × 4 = 1344, тоді як вихідне подання має розмір (21 + 21 + 28) × 16 × 4 = 4480.

Архітектура мережі зворотнього зв’язку показана на рисунку 1.11. Вхідний рівень має 1344 вузли, а вихідний рівень – 4480. Є один прихований шар із 200 одиниць. Функцією нелінійної активації, яка використовується для прихованого шару, є ReLU. Функція активації вихідного рівня сигмовидна, а функція витрат – двійковою перехресною ентропією (випадок багатокласової одинарної мітки).

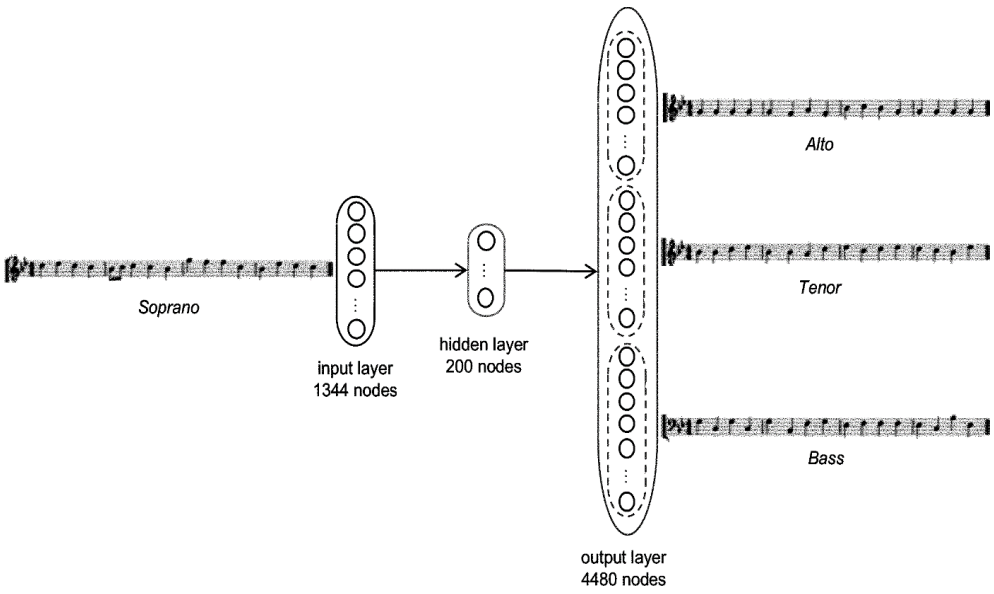


Рисунок 1.11 – Архітектура MiniBach

Детальна інформація про архітектуру та кодування показана на рисунку 1.12. Вона показує кодування послідовних зрізів часу музики у послідовні унітарні вектори, безпосередньо зіставлені з вхідними вузлами (змінними).

На рисунку кожен зафарбований чорним квадратом векторний елемент, а також кожен відповідний зафарбований чорним квадратом елемент вхідного вузла ілюструють конкретне кодування (індекс одного гарячого вектора) певного зрізу часу ноти залежно від його фактичного кроку (або затримки у випадку більш тривалого примітка, показана в дужці).

Подвійний процес відбувається на виході. Кожен елемент вихідного вузла ілюструє обрану ноту (ту, що має найбільшу ймовірність), що веде до відповідного одноразового індексу, що в кінцевому підсумку веде до послідовності нот для кожного голосу контрапункту.

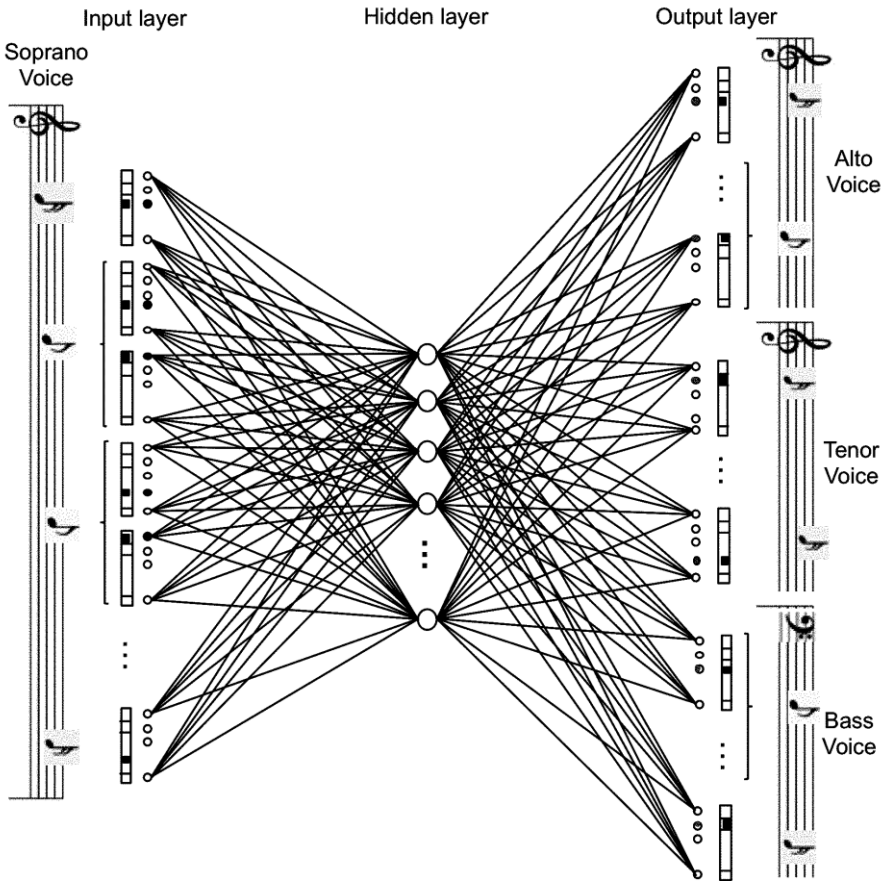


Рисунок 1.12 – Кодування MiniBach

Система представляється як One – hot×64×(1 + 3), що означає кодування з одним вхідним + трьома вихідними голосами, кожен з 64-ма (для чотирьох тактів по 16 часових кроків кожним) унітарним кодуванням нот. Архітектура представляється у вигляді двошарової архітектури прямого поширення (один прихований шар).

Система MiniBach добре генерує супровід (контрапункт, що складається з трьох різних мелодій), що відповідає вхідній мелодії. Це приклад контрольованого навчання, оскільки приклади тренування включають як вхід (мелодія), так і відповідний результат (супровід).

Тепер припустимо, що наша мета – генерувати мелодію самостійно, а не як супровід якоїсь вхідної мелодії, – базуючись на стилі, засвоєному з корпусу мелодій. Стандартна архітектура прямого поширення та супутній йому одноетапний прямопоширювальний метод, така як ті, що використовуються в MiniBach, не підходять для такої мети.

Щоб налаштувати створення нового музичного контенту з мінімальної сідової інформації, або ex nihilo, застосуємо новий метод з використанням архітектури автокодувальника.

Як було вже пояснено, на етапі навчання автокодувальник спеціалізує свій прихований шар на детекторі ознак, що характеризують тип вивченої музики та її варіації. Потім ці функції можна використовувати як вхідний інтерфейс для параметризації генерації музичного контенту. Ідея полягає в тому, щоб:

– вибрати сід як вектор значень, що відповідає одиницям прихованого шару;

– вставити його в прихований шар;

– прямо передавати його через декодер.

Цей метод, який можна назвати декодерним прямопоширювальним, може створити новий музичний твір, що відповідає особливостям, у тому ж форматі, що і тренувальні приклади.

Для того, щоб мати мінімальний і високорівневий вектор функцій, часто використовується один з типів автокоувальнику – складений автокоувальник. Потім сід вставляють у вузький прихований шар складеного автокодувальника і

подають по ланцюжку декодувальників. Тому проста сідова інформація може генерувати довільно довгий, хоча і фіксований, музичний твір.

Прикладом такого методу є символічна система генерації музики DeepHear Ragtime Melody та система генерації deepAutoController Cappoффа та Кейсі.

Для DeepHear системи використовується корпус – 600 тактів регтайм – музики Скотта Джопліна, розділений на чотири сегменти довгими частинами. Представлення – це фортепіанний рулон із мульти – унітарним кодуванням. Квантування (часовий крок) – це шістнадцята нота, таким чином, подання включає 4 × 16 = 64 часові кроки (позначені як One – hot×64). Кількість вузлів введення становить близько 5000, що забезпечує словниковий запас приблизно 80 можливих значень нот. Архітектура показана на рисунку 1.13 і являє собою чотирьохшаровий накопичуваний автокодувальник (позначений як Autoencoder) із зменшуваною кількістю прихованих одиниць, до 16 одиниць.

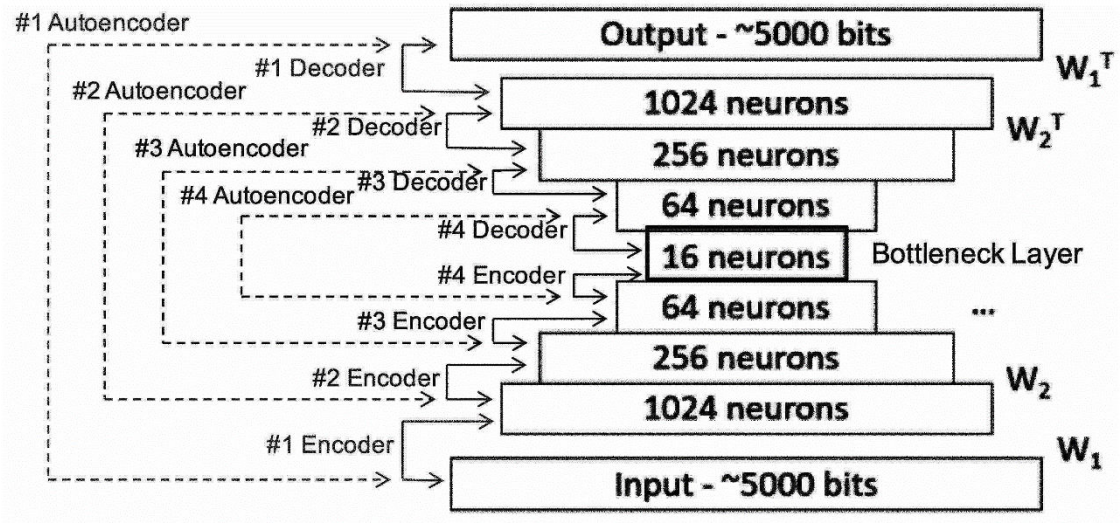


Рисунок 1.13 – Складена архітектура автокодувальника DeepHear

Після етапу попередньої підготовки проводиться заключна підготовка, причому кожен наданий приклад використовується як вхідний матеріал, так і як вихідний результат, у формі самоконтрольованого навчання.

Генерація виконується шляхом введення випадкових даних як сіди в 16 вузьких прихованих шарів, а потім шляхом перенаправлення їх у ланцюжок

декодерів для отримання вихідних даних (у тому ж форматі довжиною у чотири міри, як у прикладах тренування).

У роботі [13] Sun зауважує, що система виробляє певну кількість плагіату. Деяка сформована музика майже копіюється з корпусу. Було також заявлено, що це пов’язано з невеликим розміром вузького прихованого місця (лише 16 вузлів). Було виміряно подібність (визначається як відсоток нот у створеному фрагменті, які також є в одній із навчальних частин) і виявлено, що в середньому воно становить 59,6%, що справді є досить високим, хоча це не заважає більшості згенерованих частин композицій до звучання по різному.

Система deepAutoController подібна до DeepHear, оскільки вона також використовує складений автокодувальник. Але представлення є у вигляді звуку, а точніше спектру, породженому перетворенням Фур’є, для більш детальної інформації. Набір даних складається з 8000 пісень 10 музичних жанрів, що приводить до 70000 кадрів перетворень Фур’є за величиною. Усі дані нормуються до діапазону [0,1]. Використана функція витрат є середньоквадратичною помилкою. Архітектура є двохшаровим складеним автокодувальником, вузький прихований шар має 256 одиниць, а вхідні і вихідні шари мають 1000 вузлів. Автори прикладу повідомляють, що збільшення кількості прихованих одиниць не покращує продуктивність моделі.

Система також забезпечує користувальницький інтерфейс, проаналізований у для інтерактивного управління генерацією, наприклад, вибір цього входу (який потрібно вставити на вузькому прихованому шарові), генерування випадкового введення та управління (масштабуванням або приглушенням) активацією блоку.

Інша стратегія розширення музичного контенту заснована на семплінгу. Семплінг – це дія генерації елемента (вибірки) із стохастичної моделі згідно розподілу ймовірностей.

Основним питанням семплінгу є забезпечення відповідності згенерованих семплів заданому розподілу. Основна ідея полягає в тому, щоб сформувати послідовність семплінгових значень таким чином, щоб, у міру того, як

генерується все більше і більше семплінгових значень, розподіл значень наближався до цільового розподілу.

Таким чином, значення самплінгу виробляються ітеративно, причому розподіл наступної вибірки залежить лише від поточного значення вибірки. Кожна послідовна вибірка генерується за допомогою стратегії генерування та тестування, тобто шляхом генерації потенційного кандидата, прийняття або відхилення її (на основі визначеної щільності ймовірності) та, її регенерації.

Запропоновано різні приклади самплінгу: алгоритм Метрополіса – Гастінгса, вибірку Гіббса (GS), блокування вибірки Гіббса тощо.

Що стосується музичного змісту, ми можемо розглянути два різні рівні розподілу ймовірностей (і вибірки):

– рівень предмета або вертикальний вимір – на рівні складеного музичного елементу, наприклад акорду; у цьому випадку розподіл стосується відносин між компонентами акорду, тобто описує ймовірність нот виникати разом;

– рівень послідовності або горизонтальний вимір – на рівні послідовності предметів, наприклад, мелодії, що складається з послідовних нот; у цьому випадку розподіл стосується послідовності нот, тобто описує ймовірність появи конкретної ноти після цієї ноти.

Архітектура RBM, як правило, використовується для моделювання вертикального розміру, тобто яких нот слід відтворювати разом. Вона присвячена вивченню дистрибутивів і може ефективно навчатись на кількох прикладах. Це особливо цікаво для вивчення та генерації акордів, оскільки комбінаторний характер можливих нот, що утворюють акорд, великий, а кількість прикладів зазвичай невелика.

Архітектура RNN часто використовується для горизонтального виміру, тобто яка нота, ймовірно, буде відтворена після цієї ноти.

Варто підкреслити, що семплінг також може бути доданий для посилення мінливості контенту, яке допомогає в таких системах де одне й ті ж вхідні дані завжди дають однаковий вихідні дані. Припущення полягає в тому, що вихідне подання мелодії кодується одноразово. Іншими словами, вихідне подання має тип

піаніно, вихідний рівень активації softmax, а генерація моделюється як завдання класифікації. На рисунку 1.14, де P ($x_t = C|x_{<t}$) представляє умовну ймовірність того, щоб елемент (нота) x_t на кроці t була C, враховуючи попередні елементи $x_{<t}$ (дотепер створена мелодія).

Стандартна детермінована стратегія полягає у виборі класу (ноти) з найбільшою ймовірністю, тобто $argmax_{xt}$ $P(x_t|x_{<t})$. Тоді ми можемо легко переключитися на недетерміновану стратегію, відібравши результати, які відповідають (за допомогою функції softmax) розподілу ймовірностей між можливими нотами. Здійснюючи самплінг ноти, що слідує за генерованим розподіленям, ми вводимо стохастичність у процесі і, отже, мінливість у генерації.

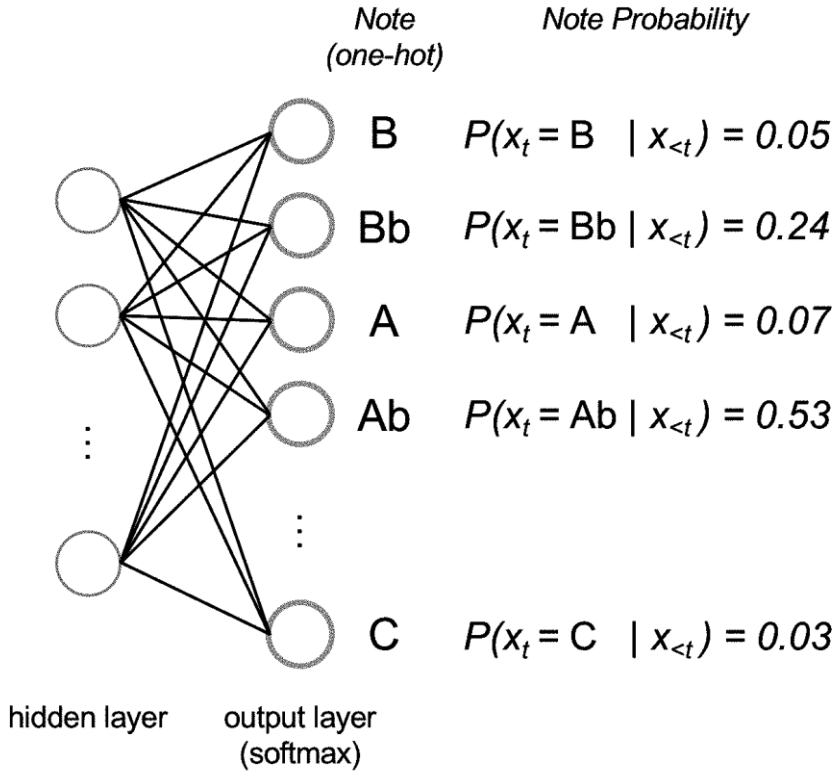


Рисунок 1.14 – Самплінг виводу softmax

Такий тип мінливості використовувався в системі CONCERT, розробленою Моцером у 1994 році.

Важливим обмеженням одноетапного прямопоширювального методу та декодерного прямопоширювального методу є те, що тривалість музики, яка

генерується (точніше кількість кроків або мір), є фіксованою. Це фактично фіксується архітектурою, а саме кількістю вузлів вихідного рівня. Щоб створити довший (або коротший) музичний твір, потрібно переналаштувати архітектуру та відповідне її представлення.

Стандартним рішенням цього обмеження є використання рекурентної нейронної мережі (RNN). Типовим використанням є:

– вибрати першу інформацію про сід (наприклад, першу ноту мелодії);

– направити його в рекурентну мережу, щоб створити наступний елемент (наприклад, наступну ноту);

– використовувати цей наступний елемент як наступний вхід для створення наступного наступного елемента;

– повторювати цей процес ітеративно, доки не буде створена послідовність (наприклад, нот, тобто мелодії) потрібної довжини.

Таким чином, можна помітити ітераційний аспект генерації, оброблений елемент за елементом. Тому можна назвати цей метод ітераційним методом зворотного кроку в часі, скороченого як ітераційна стратегія прямого поширення. Насправді часто також присутня рекурсія – вихід поточного виходу як наступний вхід. Однак є кілька рідкісних винятків, наприклад, у архітектурах Sequential та в архітектурах BLSTM, де є ітерація, але немає рекурсії. Прикладом є блюзова акордова послідовність, розроблена Еком і Шмідхубером.

В одному з їхніх експериментів мета полягає в тому, щоб навчитися і генерувати послідовності акордів. Формат представлення – фортепіанний рулон із двома типами послідовностей: мелодією та акордами, хоча акорди представлені як ноти. Мелодійний діапазон, а також акордовий словниковий запас сильно обмежений, оскільки корпус складається з 12 мір довжиною блюзу і виготовлений вручну (мелодії та акорди). 13 можливих нот поширюються від середнього C (C_4) до тенору C (C_5). 12 можливих акордів поширюються від C до B.

Використовується одноразове кодування. Квантування часу (часовий крок) встановлюється на восьмій ноті, половині мінімальної тривалості ноти, що

використовується в корпусі, що становить чверть ноти. З музикою довжиною 12 тактів це дорівнює 96 крокам у часі.

Архітектура цього першого експерименту: вхідний шар з 12 вузлами (що відповідає одноразовому кодуванню 12-лекторного словника), прихований шар із чотирма блоками LSTM, що містять по дві комірки та шар з вихідними даними з 12 вузлами (ідентично до вхідного шару).

Генерація виконується поданням сідового акорду (представленого нотою) та ітераційним перенаправленням мережі, виробляючи прогноз наступного акорду часового кроку, використовуючи його як наступний вхід і так далі, поки не буде сформована послідовність акордів. Архітектура та ітеративне покоління проілюстровано на рисунку 1.15.

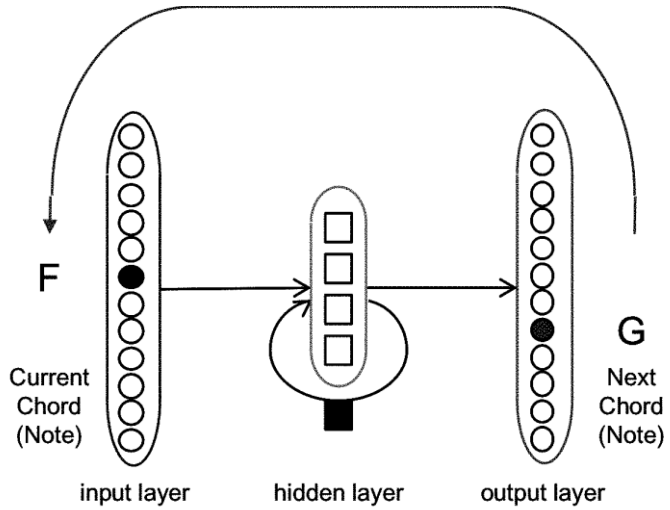


Рисунок 1.15 – Архітектура покоління блюзових акордів

Нарешті, щоб обрати найоптимальніший метод та модель, необхідно порівняти їх між собою щодо критерію їхньої ефективності у використаня до подолання специфічних проблем тощо.

Все що було аналізовано текстом вище, було організовано у таблицях для підсумування щодо архітектур моделей, методів, проблем та прикладів систем, які використовували дані моделі та методи.

Для кожної таблиці, яка аналізує кожну систему (рядок) щодо можливих типів (стовпців) для цього виміру, виникнення «X» при перетині рядка (системи)

та стовпця (типу) означає що ця система відповідає типу для цього виміру (наприклад, слідує певній грані представлення, базується на якомусь типі архітектури, виконує певні проблеми тощо).

Виникнення «X» також буде появлятись у вигляді «X^n», щоб показати кількість випадків виникнення цього типу.

Результати були проілюстровані у таблицях 1.1, 1.2 та 1.3.

Таблиця 1.1 – Порівняння систем щодо архітектур та методів

	Архітектура				Метод			
	ПП	АК	RBM	RNN	ОПП	ДП	Сампл.	ІСПП
Система								
MiniBach	X^2				X			
deepAutoController		X^2				X		
DeepHear		X^4				X		
RBM			X				X	
Blues				X				X
CONCERT				X			X	X
Celtic				X^3			X	X

Таблиця 1.2 – Порівняння архітектур та методів щодо проблем

	Проблема		
	Ex Nihilo	Варіація довжини	Мінливість контенту
Архітектура			
Пряме поширення	X		
Автокодувальник	X		X
Обмежена машина Больцмана	X		X
Рекурентна нейронна мережа	X	X	X
Метод			
Одноетапна прямопоширювальна			
Декодерна прямопоширювальна	X		
Семплінг	X		X
Ітеративна прямопоширювальна	X	X	

Таблиця 1.3 – Порівняння методів щодо архітектур

	Архітектура			
	ПП	АК	RBM	RNN
Метод				
Одноетапна прямопоширювальна	X			
Декодерна прямопоширювальна		X		
Семплінг	X	X	X	X
Ітеративна прямопоширювальна	X			X

Створене порівняння у вигляді кореляційних таблиць дозволяє зробити висновки, що для найбільшого вирішування проблем підійдуть рекурентна нейронна мережа та мережа довгої короткочасної пам’яті. Також включається той факт, що модель повинна здатність згадувати минулі деталі та розуміти основну підструктуру, щоб створити цілісний твір у відповідності з музичною структурою. В цьому випадку допомагають обрані архітектури, так як обоє успішно фіксують закономірності, що виникають з часом.

Було вирішено розробити власну модель, так як генерування музичного твору одною архітектурою недостатньо при багатовимірності поліфонічної музики, оскільки системи з рекурентними нейронними мережи інваріантні лише в часі, але не по нотам.

Серед розглянутих методів, було вирішити вдосконалити метод врахування веедних даних системи deepAutoController декодерного прямопоширювального методу, так як дані приймалися частково та вони приймалися в плані простих наборів даних.

1.3 Постановка задачі

Після огляду предметної області та здійснення детального аналізу було визначено, що проблеми, вирішувані в даній дипломній роботі є й досі актуальними, так як людське суспільство й надалі потребує музики для поліпшення внутрішньої психологічної напруги в організмі.

Дослідження предметної області дало зрозуміти, що об’єктом для дослідження цієї роботи слугують відповідні системи, які розглядають схожі проблеми в рамках області, а предметом дослідження – самі функціональні властивості системи в плані методів та моделей, що намагаються вирішити досліджувану проблему.

Отже, на основі нових рішень та вдосконалень, було сформульовано мету проекту – створення моделі та удосконалення методу врахування введення даних користувача для генерування музичних творів із використанням штучного інтелекту. Поставлена мета досягається розв’язанням таких основних задач:

– здійснити аналіз існуючих підходів генерування музичних творів;

– розробити модель вивчення штучного інтелекту теорії музики та аналізу музичних творів;

– вдосконалити метод обробки музичних треків шляхом включення введення більше користувацьких даних;

– розробити програмну реалізацію системи генерування музичних творів із використанням штучного інтелекту;

– провести тестування отриманих результатів.

За функціональністю система має вирішувати такі задачі, як:

– створення моделі генерування музичного твору;

– тренування моделі генерування музичного твору з включеністю оптимізування та обчислення функції витрат;

– генерування самого гармонійного поліфонічного музичного твору.

1.4 Висновки

В розділі було здійснено деталізований аналіз літературних джерел та систем предметної області для генерування музичних творів із використанням штучного інтелекту та аналізу музичних творів, щоб визначити стан проблеми дипломної роботи.

В підрозділі 1.1 проведено аналіз моделей, які вирішують схожу проблему. Було прийнято рішення розробити власну модель шляхом складання архітектур рекурентних нейронних мереж та мереж довгої короткочасної пам’яті для інваріантності в часі та в нотах, а також для отримання поліфонічної мелодії.

В підрозділі 1.2 проведено аналіз методів, які використовують обрані моделі для вирішення схожої проблеми. Було вирішено удосконалити метод в системі deepAutoController, надавши можливість більшого введення даних та використання більш комплексного набору даних.

В підрозділі 1.3 здійснено постановку задачі, а також визначено об’єкт, предмет, мету та задачі дослідження.

2 МОДЕЛІ ТА МЕТОДИ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ

2.1 Розроблення моделі створення музики на основі вивчення штучного інтелекту теорії музики та аналізу музичних творів

Здійснений аналіз предметної області проблеми дипломної роботи у вигляді дослідження та оцінювання методів і моделей, показав, що найоптимальнішим варіантом буде розробка власної моделі для створення музики на основі вивчення штучного інтелекту теорії музики та аналізу музичних творів, а також вдосконалення методу обробки музичних треків із врахуванням введення нових даних користувачем.

Для найоптимальнішого вирішення проблеми необхідно спроектувати власну модель, що буде тренувати мережу програмного додатку для цієї роботи.

В основі моделі тренувального процесу системи генерування музичних творів заплановані наступні особливості:

– здатність згадувати минулі деталі даних музичного твору, щоб їх можна було спроектувати в майбутньому;

– наявність інваріантності в часі;

– наявність інваріантності по нотам;

– здатність генерування поліфонічного музичного твору – відтворення більше одної ноти на один крок у часі;

– здатність розуміння музичної структури;

– узгодження творів із музичними правилами.

Як і було узгоджено з попереднього розділу, обрані рекурентні нейронні мережі і, зокрема, мережі довгої короткочасної пам’яті, успішно фіксують закономірності, що виникають з часом. Щоб зафіксувати складність музичної структури у порівнянні з гармонійною та мелодійною структурою, ноти на кожному проміжку часу слід моделювати як спільний розподіл ймовірностей. Простіше кажучи, повинно бути розуміння музичного розміру або кількості нот у такті. Крім того, архітектура, заснована на рекурентній нейронній мережі,

дозволяє системі необмежено генерувати ноти, однакові для кожного часового кроку, роблячи пісні інваріантними в часі.

Щоб врахувати октави та круговість висоти, потрібен більший контекст. Дотримуючись концепції однієї із розглянутих архітектур згорткової нейронної мережі, рішення полягає у використанні ядра або вікна нот та ковзання цього ядра або обертання навколо нот. Це дозволяє моделі скористатися перевагами «місцевих просторових кореляцій» між заходами та укріплення стійкості до природних перетворень. Музично кажучи, ноти можна транспортувати вгору-вниз, залишаючись принципово однаковими. Щоб врахувати цю круговість висоти, мережа повинна бути приблизно однаковою для кожної ноти. Крім того, дана концепція відтворює ідею поліфонії, коли вона може одночасно відтворювати кілька нот, враховуючи вибір зв'язних акордів. Нарешті, вони вирішують інваріантність по нотам, так як рекурентні мережі лише інваріантні в часі.

Розроблена модель для дипломної роботи використовує двошарову архітектуру водночас з мережею довгої короткочасної пам'яті та з рекурентною мережею, що використовує ядро для транспорування нот.

Варто підкреслити додатково, що таке складання рекурентних мереж має проблему в створенні акордів – вихідні дані кожної ноти повністю незалежні від вихідних даних кожної ноти. Тут можна спробувати використати комбінацію мереж RNN-RBM: нехай перша частина нашої мережі стосується часу, а друга частина створює гарні акорди. Але RBM дає єдиний умовний розподіл пакету вихідних даних, що несумісно з використанням однієї мережі на ноту.

Для вирішення цієї проблеми було запозичено концепцію «двоосьової» конфігурації. Для одної моделі є дано дві осі (та одна псевдовісь) – вісь часу і вісь ноти (та псевдо вісь напрямку обчислення). Кожен рекурентний рівень перетворює входи на виходи і посилає періодичні з'єднання по одній з цих осей.

Перші два шари мають зв'язки через етапи часу, але незалежні від нот. З іншого боку, останні два шари мають зв'язок між нотами, але незалежні між часовими кроками. Це дозволяє мати водночас паттернізацію як у просторі часу, так і в просторі нот, не жертвуючи інваріантністю.

Отже, щоб захопити гармонійну та мелодійну структури між нотами, модель буде використовувати двошарову архітектуру водночас з мережею довгої короткочасної пам’яті та з рекурентною мережею, де одна мережа довгої короткочасної пам’яті розташована на осі часу, а інша – на осі нот.

Було спроектовано процес тренування моделі для мережі дипломної роботи.

Нотні осі, які мережа довгої короткочасної пам’яті отримує в якості вхідного сигналу, є конкатенацією остаточних вихідних даних нотних осей LSTM для попередньої нотної партитури та активації останнього шару часової осі LSTM для конкретної ноти. Вихідні дані остаточних активацій нотних осей LSTM потім подаються в шар softmax для конвертування на їхню ймовірність у вигляді формули (2.1):

$$p^{(n,t)}(v_n = 1 | v_{<n}) \tag{2.1}$$

Витрата відповідає похибці перехресної ентропії передбачень порівняно з відтвореною нотою на кожному часовому кроці. Отже, кожна нота має часову складову від часової осі LSTM. Це дозволяє зрозуміти часові співвідношення для конкретної ноти та змоделювати спільний розподіл нот на конкретному етапі часу. Приєднуючи інформацію з LSTM, орієнтованої на часовий компонент, та LSTM, сфокусовану на компоненті ноти, зв’язки всередині та між нотами фіксуються для кожного кроку часу. Використовуючи цей підхід для кожної ноти послідовно, можна дізнатися повний умовний розподіл для кожного часового кроку. Крім того, ще однією ключовою функціональною частиною є вбудовування в модель вікно, яке буде ковзати по послідовностям нот. Ця архітектура дозволяє моделі засвоїти гармонійну та мелодійну структуру нот, що враховує круговість висоти звуку.

Часова вісь LSTM залежить від обраних нот, а не від конкретного виходу шарів нотних осей. Обґрунтування полягає в тому, що всі ноти на всіх етапах часу відомі, тому тренування можна пришвидшити. Приріст часу надходить від обробки вхідних даних, а потім подання попередньо оброблених вхідних даних через часові осі LSTM паралельно для всіх нот. Далі шар нотних осей LSTM

обчислює ймовірності за всі часові кроки. Це забезпечує значну швидкість використання графічного процесора для паралельних обчислень.

Коли вивчено розподіл ймовірностей, вибірка з цього розподілу пропонує спосіб генерувати нові послідовності. Послідовності не відомі заздалегідь. Мережа повинна проектувати один крок у майбутньому за один раз. Вхідні дані для кожного кроку часу використовуються для просування шарів часової осі LSTM по одному кроку за раз, щоб скласти ноту в наступний період. Спочатку потрібно взяти зразок під час створення розподілу. Кожна нота взята з розподілу Бернуллі. Потім це отримане значення використовується для введення до наступної ноти. Цей процес повторюється для всіх нот, після чого модель переходить до наступного часового кроку.

Під час тренування було використано більший та різноманітний набір даних з різними нотними та структурними моделями. Метою тут було піддати модель під час навчання найрізноманітнішим моделям, щоб заохотити якомога більше різноманітності у випуску. Формат MIDI-файлу дозволяє використовувати часову позицію в музиці. Компонент часу був важливою особливістю для вбудовування в набір даних, щоб модель могла вивчати закономірності з часом відносно різних послідовностей нот. До векторів нот, що подаються в модель, було додано додатковий вимір: двійковий файл, 0 або 1, щоб вказати, чи була нота артикульована чи витримана на певному етапі часу. Наприклад, перший крок часу для відтворення ноти представлено як 11. Збереження попередньої ноти представлено як 10, а пауза – як 00. Цей доданий вимір дозволяє моделі відтворювати одну і ту ж ноту кілька разів поспіль. З точки зору введення вимір артикуляції або біт обробляється заздалегідь. Ця обробка виконується паралельно відтворювальному виміру, який разом подається на часову вісь LSTM. З точки зору вихідних даних, нотна вісь LSTM дає ймовірність відтворення ноти та ймовірність артикуляції тієї самої ноти. При обчисленні функції витрат карається неправильна артикуляція відтвореної ноти. Вихідні дані артикуляції для нот, які не слід відтворювати, ігноруються. Якщо нота не відтворена, нема сенсу покарання за артикуляцію.

Необхідно надати функціональні модулі для моделі створення музики у вигляді математичних об’єктів.

Для здійснення тренування мережі дипломної роботи, її необхідно спроектувати в більш організовану структуру даних. Модуль генерування матриці стану ноти дозволяє згенерувати матрицю (вектор ознак для одної точки даних із пакету) із випадкового сегмента з восьми тактів із випадково вибраного MIDI-файлу за формулою (2.2):

$$\sum_{i=1}^{L_{batch}} \begin{bmatrix} [p,a]_N^{(1)} \cdots [p,a]_N^{(T)} \\ \vdots \\ [p,a]_1^{(1)} \cdots [p,a]_1^{(T)} \end{bmatrix} \tag{2.2}$$

де N – кількість нот на кожному кроці часу;

T – кількість кроків часу в векторі;

p – бінарне значення стану “відтворювання”;

a – бінарне значення стану “артикуляції”;

L_{batch} – чотирьохвимірний тензор матриці стану ноти.

Даний вектор даних у цій нейромережі називається «матрицею стану ноти». Це представляє стан «відтворення» та «артикуляції» кожної ноти в діапазоні MIDI-значень та для кожного часового кроку через визначений проміжок часу (тобто вісім мір з 16 кроками часу на такт). Модель бере за вхідні дані один пакет цих векторів даних характеристик, єдиний чотирьохвимірний тензор, який в спроектованій моделі буде називатись «Note_State_Batch».

Далі, отриманий вектор йде на обробку до модуля ядра вхідних даних, який як і було вищесказано, грає роль одновимірного згорткового шару та транспонує ноти по нотній осі для генерування поліфонічного музичного твору, що є інваріантним по нотам. Для кожної ноти на кожному кроці часу до кожного пакету модуль генерує розширений 80-елементний вектор, що складається з:

– номера MIDI – ноти (2.3);

$$[n]_{1x1} \tag{2.3}$$

– одного унітарного вектора класу висоти ноти (2.4);

$$[0 \dots 1 \dots 0]_{12x1} \tag{2.4}$$

– вікна відтворення значень чи артикуляції відносно «n»-ої ноти (ефективний згорнутий аспект ядра моделі) (2.5);

$$\left[p_{n-12}^{(t)} a_{n-12}^{(t)} \dots p_{n}^{(t)} a_{n}^{(t)} \dots p_{n+12}^{(t)} a_{n+12}^{(t)}\right]_{50x1} \tag{2.5}$$

– вектор суми всіх відтворених нот у кожному класі висоти звуку (2.6);

$$\left[n\left(P_{pitchclass}\right)_{(n)}^{(t)} \dots n\left(P_{pitchclass}\right)_{(n+11)}^{(t)}\right]_{12x1} \tag{2.6}$$

де $n(P_{pitchclass})$ – кількість класів висоти звуку.

– бінарний вектор, що представляє позицію 16 ноти в межах такту (2.7).

$$[LSB \dots MSB]_{4x1} \tag{2.7}$$

де *LSB* – найменш значний такт;

MSB – найбільш значний такт.

Загально, даний функціональний модуль можна описати формулою (2.8):

$$\sum_{i=1}^{L_{batch}} \sum_{n=1}^{N} \sum_{t=1}^{T} [p, a]_{N}^{(1)} \tag{2.8}$$

де $[p, a]^{(1)}{}_{N} \rightarrow [N_{num}\ P_{pitchclass}\ P_{prev\text{-}vic}\ P_{context}\ P_{0}]_{80x1}$;

P_0 – вектор нулів, коли не враховується артикуляція.

Отриманий розширений вектор буде оброблятись модулем часових осей LSTM, які проходять вздовж часових осей на довжину виміру часового пакету для кожної ноти паралельно з прив’язаними вагами. Даний функціональний модуль фіксує послідовні патерни музичного твору і, у поєднанні з модулем ядра вхідних даних, зберігає інваріантість трансляції завдяки вікну введення відносних нот та прив’язаним LSTM вагам для всіх нот. Завдяки цим зв’язаним вагам, обчислення можуть виконуватися паралельно між нотами та між семплами матриці стану ноти у вигляді окремих ефективних пакетів. Єдиний необхідний послідовний аспект – вздовж осі часу. Виконується довільна кількість каскадних комірок LSTM і після кожної комірки застосовується маска виключення.

Даний модуль буде працювати за такою формулою(2.9):

$$\sum_{t=1}^{T} h^{(t)}_{(N-t-layer)_n} \tag{2.9}$$

де $h^{(t)}_n$ – приховані стани моделі.

Нарешті, необхідно обробити отримані дані через модуль нотних осей LSTM, де замість послідовного проходження по часової осі, цей етап проходить послідовно по нотній осі. Даний модуль надає можливість запустити нотну LSTM мережу, щоб генерувати ймовірність вихідних нот для кожної ноти на кроці часу та для кожного пакету. Даний процес відображається формулою (2.10):

$$\sum_{n=1}^{N} h_{(num-t-layer)_n} \tag{2.10}$$

Крім того, цей модуль включає в себе локальний зворотний зв’язок згенерованих семплів до їхніх вхідних даних. Після кожного нотного кроку, комірка LSTM виробляє пару логітів, що представляють зворотну сигмоїду ймовірності генерування відтворення або артикуляції для цієї ноти (2.11):

$$Wh_{(num-n-layer)_n} + b = \begin{bmatrix} y_n - play \\ y_n - artic \end{bmatrix} \tag{2.11}$$

Далі, з розподілу Бернуллі витягується семпл гри та артикуляції. Якщо семпл відтворення має значення «0», що означає «не відтворено», артикуляційний семпл також примусово доходить до «0», щоб уникнути генерації будь-яких значень, відсутніх у вхідних даних. Згенерована дискретизована пара в ноті n-1, об'єднана з вхідними даними часового LSTM на ноті n, повертається назад на вхід нотатурного LSTM для кроку n. Цей зворотний зв'язок створює умовну ймовірність для кожної ноти на основі фактичних значень, створених для нижчих нот. Це допомагає запобігти відтворенню дисонансних одночасних нот (2.12):

$$Sample[\sigma(y_n) = Prob(note_n = 1)] = \begin{bmatrix} playgen_n \\ articgen_n \end{bmatrix} \quad (2.12)$$

Остаточними вихідними тензорами цього модуля є пакет логітів та відповідні згенеровані семпли, які будуть використовуються для тренування та генерації музики, відповідно.

Після вибудовування пакету логітів до елементів пакету стану нот, що відповідають одному кроку часу в майбутньому, необхідно обчислити всі витрати за допомогою модулю функції витрат.

Завдання модуля полягає у обчисленні перехресної ентропії між згенерованими даними пакету логітів та даними пакету станів нот у вигляді згенерованих семплів у вигляді формули (2.13):

$$-N_{state} ln\big(\sigma(y)\big) + \Big(-(1 - N_{state}) ln\big(1 - \sigma(y)\big)\Big) \quad (2.13)$$

де N_{state} – номер стану ноти;

y – логіти, що представляють зворотну сигмоїду ймовірності генерування відтворення або артикуляції ноти;

σ – ваговий коефіцієнт.

Отримані витрати необхідно мінімізувати для якісного генерування музичних творів.

Дотримуючись наукової публікації Мун [14] як рекомендований орієнтир, до кожного шару LSTM було застосовано випадання 0,75. Обраним оптимізатором було ADADELTA [15]. Вибрана швидкість навчання – 1,0. Моделі будуть оцінюватись у двох вимірах за формулою:

$$\Delta_\theta = f\left(\nabla_\theta Loss, \Delta_{\theta-prev}, \gamma, \dots\right) \quad (2.14)$$

Отже, на основі моделей двонаправлених конфігурацій, була розроблена власна модель для тренувального процесу мережі, яка аналізує MIDI-файли тренувальних музичних творів та на їх основі генерує музичні твори, узгоджені з музичними правилами, згідно структури твору, гармонізації мелодії, тощо.

Порівняно з розглянутими вище системами, що використовували одинарні моделі (такі, як система MiniBach, наприклад), вони відтворюють тільки одну ноту за крок часу, тобто воні здатні генерувати лише монодичні музичні твори, коли дана модель здатна відтворювати поліфонічні твори за допомогою згортково-подібного ядра, який може транспонувати ноти.

Також в порівняні з системою блюзової акордової послідовності, розглянутої також вище, дана модель має контроль над варіативністю довжини відтворювання згенерованого музичного твору і також має контроль над артикуляцією ноти.

Нарешті, порівнюючи дану модель з іншими зразками, такими як моделі двонаправленої конфігурації, (паралельні мережі LSTM-NADE), можна побачити, що в останньої моделі є недолік в порівнянні з моделлю дипломної роботи у використанні двостороннього ядра зведення вихідних даних нот. Воно здатно фіксувати деякі найважливіші взаємозв’язки між нотами, які вони здатні проаналізувати з тренувальних файлів, але воно також заважає мережі вивчати будь-які інші точні залежності в файлах через лімітовану природу двостороннього ядра.

При аналізу адекватності розробленої моделі, більшості серйозних аномалій не було знайдено і можна дійти до висновків, що вона виконує усі необхідні функціональності для генерування музичних творів.

Дана модель має на меті удосконалене розуміння музичної структури та чітке передбачення нотних даних із наданням можливості генерування поліфонічної музики, що узгоджується із музичними правилами.

За допомогою цієї моделі можна здійснити тренування та валідацію моделі в чисельній формі. Використання потім натренованої моделі необхідне для створення згенерованої музичної композиції.

Однак через використання рекурентних нейронних мереж, модель страждає від надмірного повторення однієї і тієї ж ноти або створює послідовності нот, які не мають глобальної зв’язної структури. Тому робота може звучати звивисто або без загальної закономірності.

Вирішенням цієї проблеми є впровадження вдосконаленого методу обробки музичних треків, при яких користувач може зробити музичний твір менш повторним шляхом зміни параметрів під час генерування.

2.2 Вдосконалення методу обробки музичних треків із врахуванням введення даних користувача

На думку автора, для генерування якісного музичного твору з унікальним звучанням, необхідна певна інтерактивність користувача з програмною системою, так як у системи відсутня особливість людського творчого процесу, який в основному протікає неусвідомлено.

Серед усіх розглянутих методів, які було проаналізовано в попередньому розділі, найбільш оптмальним варіантом серед частково інтерактивних інкрементальних систем, заснованих на архітектурах глибинного навчання себе показав метод системи deepAutoController. Дана система забезпечує інтерактивне управління генерацією музичних творів такими шляхами:

– вибір заданих вхідних даних;

– генерування випадкових вхідних даних в стек декодера;

– контроль (масштабуванням або приглушенням) активації заданих прихованих одиниць.

Але в цьому методі є недолік у плані малої кількості вповаджених гіперпараметрів та в застосуванні спрощенної структури даних – при аналізі було визначено, що оригінальна модель застосовувала багатовимірний вектор для зберігання нотних даних MIDI-файлів.

Для покращення методу обробки даних, було вроваджено більш комплексну структуру даних, яка приймає більшу кількість гіперпараметрів.

Покращений метод має на меті використання вектору даних у нейронній мережі цього проекту, що називається «матрицею стану ноти», яку показано у формулі (2.15):

$$NSM = \begin{bmatrix} [p,a]_N^{(1)} \cdots [p,a]_N^{(T)} \\ \vdots \\ [p,a]_1^{(1)} \cdots [p,a]_1^{(T)} \end{bmatrix} \tag{2.15}$$

де N – кількість нот на кожному кроці часу;

T – кількість кроків часу в векторі;

p – бінарне значення стану “відтворювання”;

a – бінарне значення стану “артикуляції”.

Вона представляє стан «відтворення» та «артикуляції» кожної ноти в діапазоні MIDI – значень та для кожного часового кроку через визначений проміжок часу (тобто вісім мір з 16 кроками часу на такт).

Оптимізація методу при впровадженні матриці стану ноти полягає в тому, що вдосконалений метод обробки музичних даних має більш розвинену гнучкість введення даних, а також більшу загальність у встановлені та зміні різних гіперпраметрів, таких як:

– кількість шарів;

– розмір прихованої одиниці;
– довжина послідовності;
– проміжки часу;
– розмір пакетів;
– швидкість навчання.

Також даний метод є параметризованим, тому користувачі, на відміну від оригінального методу deepAutoController, можуть також встановити довжину партитури нот, поданих у LSTM нотних осей, і тривалість кроків часу, поданих у LSTM часових осей. Розмір партитури та тривалість часових кроків є важливими особливостями, оскільки музика сильно варіюється залежно від жанру та виконавця. Даний метод дозволить адаптувати її до конкретних потреб користувача.

Також код, який буде написаний для цього методу, має мінімізувати використання циклів «for», щоб збільшити швидкість тренування нейронної мережі за обчислювальний час.

2.3 Висновки

В розділі було здійснено розробку моделі та вдосконалення методу, щоб отримані результати використати для проектування системи дипломної роботи.

В підрозділі 2.1 розроблено математичну модель створення музичних творів на основі вивчення штучного інтелекту теорії музики та аналізу музичних творів. Також здісненено планування згідно структури моделі, а також детально пояснено логіку її тренування мережі та генерування музичних творів. Функціональність моделі було розбито на модулі, які виконують свою роботу на використанні формул (1)-(8), було обґрунтовано роботу самих модулів. Після того, було здійснено порівняння отриманої моделі з вже існуючими, які вирішували схожу проблему та було проведено аналіз адекватності розробленої моделі для підвердження відсутності аномалій.

В підрозділі 2.2 оптимізовано метод обробки музичних треків із врахуванням введення даних користувача системи deepAutoController. Здійснено власне дослідження щодо принципу інтерактивності методу та описано зміни, які мають покращити унікальність музичного твору за рахунок втручання користувача. Були описані переваги запропонованих змін.

3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИРІШЕННЯ ПРОБЛЕМИ

3.1 Проектування архітектури програмної системи

Розроблена математична модель створення музичних творів на основі вивчення штучного інтелекту теорії музики та аналізу музичних творів, дозволяє здійснити проектуваня рішень для програмної системи дипломної роботи.

Як і в розробленої в попередньому розділі моделі, спроектована архітектура тренувального процесу буде містити весь функціонал розроблених модулів. Всього їх буде шість штук:

– модуль генерування пакету стану нот;
– модуль ядра вхідних даних;
– модуль часових осей LSTM;
– модуль нотних осей LSTM;
– модуль функції витрат;
– модуль генерування музичного твору.

Загальна структура проектованої реалізації архітектури моделі розділена на дві основні задачі: тренування та валідація моделі чисельно, а потім використання натренованої моделі для створення нових MIDI-файлів для якісного оцінювання.

Обидві функції використовують однаковий граф моделі в різних контекстах: тренувальний процес, показано на розробленій моделі у рисунку 3.1, працює за таким алгоримом:

– ітеративно вводить у модель Note_State_Batch;
– запускає модель через усі відповідні кроки часу та ноти;
– виводить тензор відповідного логіту або зворотну сигмоподібну ймовірність того, що дана нота буде відтворюватись на кроці часу для власного відтворювання або артикуляції.

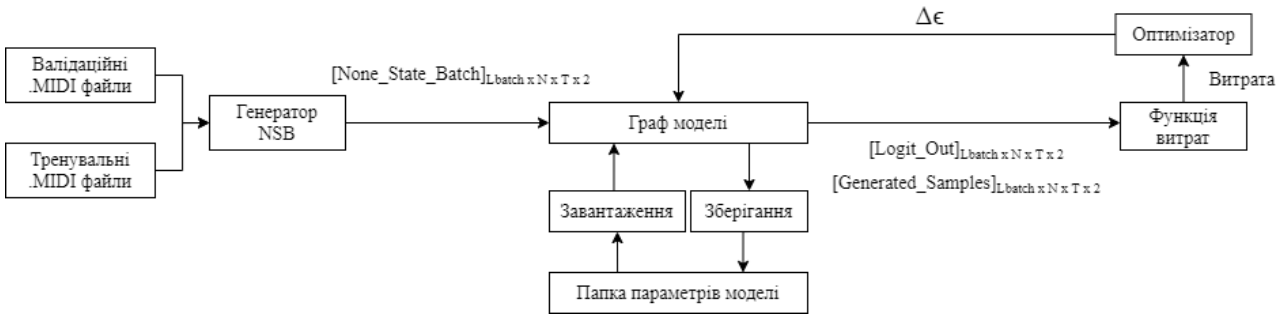


Рисунок 3.1 – Діаграма тренувального процесу

Детально опишем кожний з спроектованих модулів.

Функція модулю генерування пакету стану нот відповідно полягає у конверсії введених користувачем трнувальних файлів, валідаційних файлів та параметрів (бажана кількість MIDI-файлів, кількість часових кроків через семпл, кількість пакетів тощо.) у пакет матриць стану ноти – чотирьохвимірний тензор розмірністю L_{batch} x N x T.

Спроектована архітектура для цього модуля показана на рисунку 3.2.

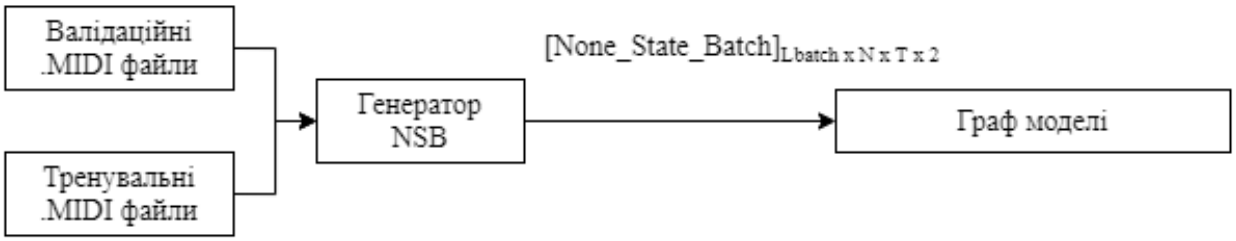


Рисунок 3.2 – Діаграма процесу генерування пакету стану нот

Модуль графу моделі буде виконувати наступні задачі:

– транспонування нот для створення структури нового музичного твору;

– обробка музичного твору по часовій осі;

– обробка музичного твору по нотній осі.

Так як здійснюється багато функцій за один модуль, було вирішено провести декомпозицію вищого рівню графу моделі для того, щоб здійснити деталізацію опису кожного з компонентів цього модуля.

Декомпозиція модуля показана на рисунку 3.3.

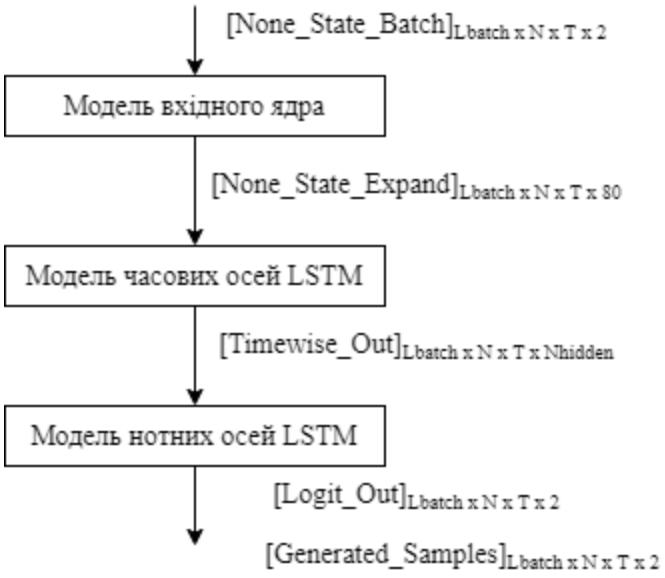


Рисунок 3.3 – Декомпозиція графу моделі

Детально опишемо кожний з отриманих компонентів модуля.

Компонент вхідного ядра для кожної ноти на кожному часовому кроці на кожний пакет генерує 80-елементний розширений вектор з переформованими MIDI-нотами.

Псевдокод роботи модуля вхідного ядра показаний на рисунку 3.4.

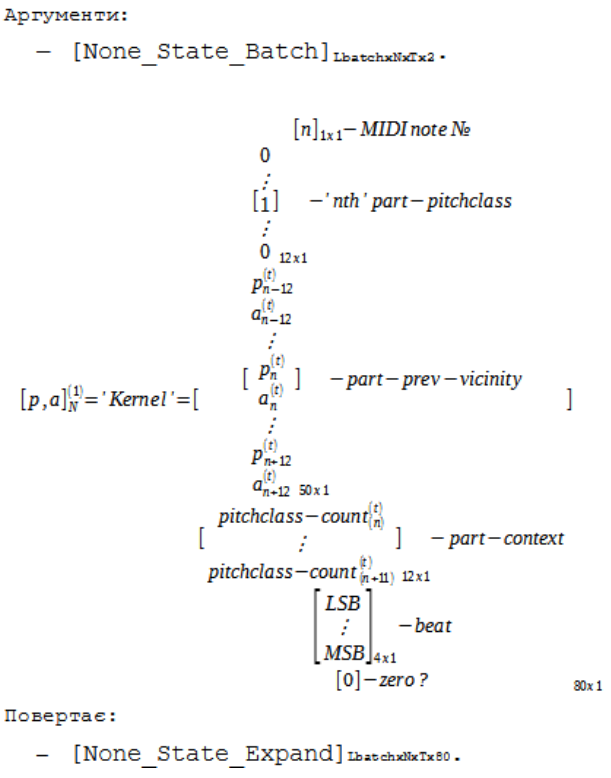


Рисунок 3.4 – Псевдо-код для вхідного ядра

Наступний компонент декомпозиції графу ядра є компонентом обробки часових осей LSTM. Даний компонент запускає паралельні LSTM мережі розмірністю N x L_{batch} через часову вісь разом з приєднаними вагами.

Псевдокод для компоненту обробки часових осей LSTM зображений на рисунку 3.5.

Аргументи:

- $[\text{None\_State\_Expand}]_{L_{batch}xNxTxL_{filt}}$. Where $L_{filt} = 80$

$t = 1:T$

$h_{(1)n}^{(t)} = \text{LSTM}(h_{(1)n}^{(t-1)}, \text{None\_State\_Expand}_n^{(t)})$

$h_{(2)n}^{(t)} = \text{LSTM}(h_{(2)n}^{(t-1)}, h_{(1)n}^{(t)})$

$\vdots$

$\text{timewise\_out} = h_{(\text{num-t-layer})n}^{(t)} = \text{LSTM}(h_{(\text{num-t-layer})n}^{(t-1)}, h_{(\text{num-t-layer})n}^{(t)})$

Повертає:

- $[\text{timewise\_out}]_{L_{batch}xNxTxN_{hnum\text{-}t\text{-}layer}}$.

Рисунок 3.5 – Псевдо-код для часових осей LSTM

Заключним компонентом є компонент обробки нотних осей LSTM.

Даний компонент запускає нотні LSTM мережі для генерування ймовірності вихідних нот для кожної ноти на кожному кроці часу для кожного пакету та під кінець генерує матрицю ймовірностей відтворювальної та артикуляційної ноти n на кроці часу t з урахуванням музичних входів до кроку часу t-1.

Даний підмодуль може бути потенційно одно або багатошаровою стадією в графі моделі, подібно підмодулю обробки часових осей LSTM, також із виключенням після кожного шару.

Псевдокод для компоненту обробки нотних осей LSTM зображений на рисунку 3.6.

Аргументи:

– $[\text{timewise\_out}]_{L_{batch} \times N \times T \times N_{num\text{-}t\text{-}layer}}$.

For n = 1 to N:

$\text{cell\_input}_n = \begin{bmatrix} timewise-out_n \\ note-gen_{n-1} \end{bmatrix}$

$h_{(1)n} = \text{LSTM}(h_{(1)(n-1)}, \text{cell\_input}_n)$

$h_{(2)n} = \text{LSTM}(h_{(2)(n-1)}, h_{(1)n})$

⋮

$h_{(\text{num-t-layer})n} = \text{LSTM}(h_{(\text{num-t-layer})(n-1)}, h_{(\text{num-t-layer-1})n})$

$\text{Logits}_n = Wh_{(\text{num-t-layer})n} + b = \begin{bmatrix} Logit_n - play \\ Logit_n - artic \end{bmatrix}$

$\text{note\_gen}_n = \text{Sample}[\sigma(\text{Logits}_n) = \text{Prob}(\text{note\_n}=1)] = \begin{bmatrix} play-gen_n \\ artic-gen_n \end{bmatrix}$

if $(\text{play\_gen}_n = 0)$ then $\text{artic\_gen}_n = 0$

Повертає:

– $[\text{Logits}]_{L_{batch} \times N \times T \times 2}$.

– $[\text{note\_gen}]_{L_{batch} \times N \times T \times 2}$.

Рисунок 3.6 – Псевдо-код для нотних осей LSTM

Модуль функції витрат обчислює всі можливі витрати у плані перехресної ентропії між пакетом логітів та пакетом стану нот. Отримані витрати проходять подальшу мінімізацію за допомогою оптимізатора.

Псевдокод функції витрат показаний на рисунку 3.7.

Аргументи:

– $\text{Logits}]_{L_{batch} \times N \times T \times 2}$ (inverse sigmoid of Probability that play/articulate = 1;

– $\text{Labels}(t\text{-}1) = [\text{Note\_State\_Batch}]_{L_{batch} \times N \times T \times 2}$.

cross_entropy = sigmoid_cross_entropy_with_logits(logits=logits. labels=Labels) = - note_state ln(σ(logits)) + - (1 - note_state) In (1 - σ((logits)) = - note_state ln(Probability=1)+ -(1 - note_state) In (Probability=0)

$Loss = \frac{1}{TNL\,L_{batch}} \sum_{b=1}^{L_{batch}} \sum_{t=1}^{T} \sum_{n=1}^{N} cross-entropy$

$\log-likelihood\ at\ 1\ time\ step = -\frac{1}{TL_{batch}} \sum_{b=1}^{L_{batch}} \sum_{t=1}^{T} \sum_{n=1}^{N} cross-entropy$

$\text{Log-likelihood at 1 time step} = -\frac{1}{TL_{batch}} \sum_{b=1}^{L_{batch}} \sum_{t=1}^{T} \sum_{n=1}^{N} cross-entropy$

Повертає:

– Loss (scalar).

Рисунок 3.7 – Псевдокод для обчислення витрат

Під час задачі генерації музики, представленого на рисунку 3.8, модуль ітеративно виконується через один крок часу, кожен раз повертаючи генеровані семпли (Generated_Samples) як вхідні дані Note_State_Batch для наступного кроку часу. Ці семпли накопичуються, і це створює тензор генерованих семплів у вигляді довільної довжини часу Note_State_Batch. Потім згенеровані семпли перетворюються у MIDI – файли за допомогою функцій подальшої обробки для якісної оцінки.

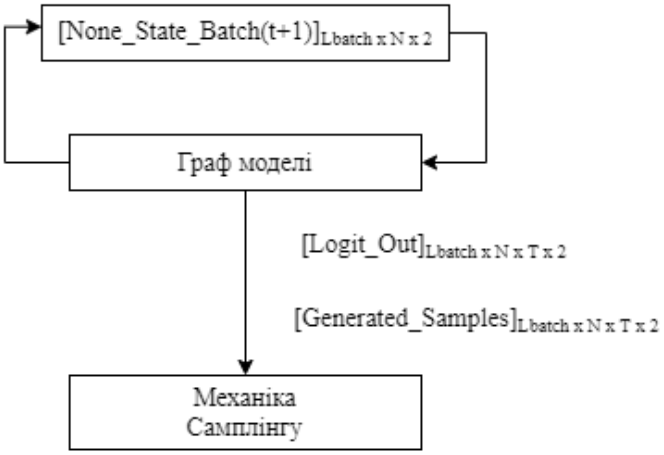


Рисунок 3.8 – Діаграма генерації музичної композиції

Повна спроектована архітектура моделі тренувального процесу представлена на рисунку А.1 (додаток А).

3.2 Розроблення вимог до програмного забезпечення для вирішення задачі

Тепер необхідно обрати набір технологій, які необхідні для програмної розробки самого дипломного проекту.

Необхідно обрати мову форматування, якою музичний твір буде висловлюватися для інтерпретації комп’ютером. Найоптимальнішим варінтом є використання MIDI – технічний стандарт, що описує протокол, цифровий інтерфейс та роз’єми для взаємодії між різними електронними музичними інструментами, програмним забезпеченням та пристроями[16]. MIDI передає

повідомлення про події, які вказують дані про ефективність нот у реальному часі, а також дані управління.

Цей формат зменшує абстрактні музичні твори високого рівня до машиночитаних даних у дуже компактному форматі (пісня, що зберігається у файлі MP3 розміром 4 МБ, складатиме лише кілька сотень кілобайт в MIDI), що буде ефективною в ролі вхідних даних у датасеті для здійснювання навчання нейронної мережі, так як буде більша швидкодія.

З цієї мови було включено два повідомлення для наших проблем:

а) Note on вказує на позначення відтворення ноти. Це містить:

1) номер каналу, який вказує на інструмент або доріжку, заданий цілим числом у межах набору {0,1, ..., 15};

2) номер ноти MIDI, який вказує висоту ноти, задану цілим числом у межах набору {0,1, ..., 127};

3) швидкість, яка вказує, наскільки голосно звучить нота, задана цілим числом у межах набору {0,1, ..., 127}.

Прикладом такого повідомлення є «Note on, 0, 60, 50», що означає «на каналі 1, почне відтворювати середнє С зі швидкістю 50»;

б) Note off вказує на те, що нота закінчується. У цій ситуації швидкість вказує на те, як швидко відпускається нота. Приклад «Note off, 0, 60, 20», що означає «На каналі 1 буде припинено відтворення середнього С зі швидкістю 20».

Кожна подія ноти фактично вбудована в фрагмент доріжки, структуру даних, що містить значення дельта-часу, яке визначає інформацію про час та саму подію. Значення дельта-часу може представляти:

– відносний метричний час – кількість цокань з самого початку. Посилання, назване поділом і визначене в заголовку файлу, визначає кількість цокань на квартальну ноту;

– абсолютний час – корисний для реальних вистав.

Приклад уривку з файлу MIDI (перетвореного в читабельний файл ASCII) та відповідного йому балу наведено на рисунку 3.9. Ділення встановлено на 384,

тобто 384 цокання на квартальну ноту (що відповідає 96 цоканням для шістнадцятої ноти).

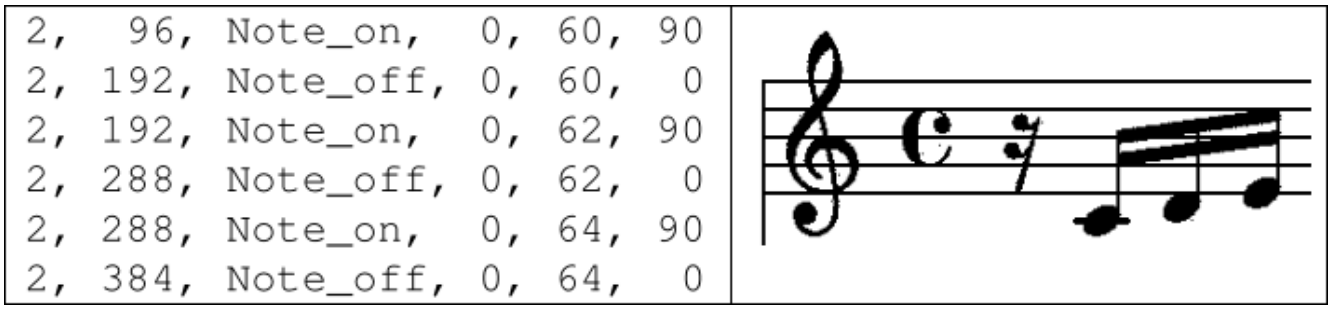


Рисунок 3.9 – Витяг із файлу MIDI (ліва частина) та партитура, що відповідає уривку MIDI

Було здійснено пошук високорівневих мов, які можуть дозволити вирішити проблему дипломного проекту.

Після здійснення огляду всіх можливих варіантів, було обрано найоптимальніші – Python, Java, C++, R та LISP.

Було детально розглянуто кожен з можливих варіантів.

Python є інтерпретованою, об’єктно-орієнтованою мовою програмування високого рівня з динамічною семантикою.

Однією з головних причин переваги в Python для використання у побудуванні штучного інтелекту складається у широкому наборі бібліотек, які надають користувачам інструментарій базового рівня, тому їм не потрібно кодувати їх з самого початку. Машинне навчання вимагає постійної обробки даних, а бібліотеки Python дозволяють отримувати доступ, обробляти та трансформувати дані.

Ось деякі з найпоширеніших бібліотек, які можна використовувати для машинного навчання та штучного інтелекту:

– tensorflow призначений для роботи з глибинним навчанням шляхом налаштування, навчання та використання штучних нейронних мереж з масивними наборами даних;

– numpy є розширенням для Python, яке надає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою високорівневих математичних функцій для операцій з масивами;

– python MIDI є бібліотекою, яка надає засоби для обробки, послідовності, запису та відтворення даних MIDI;

– ще важливою бібліотекою для створення генератора музичних композицій є Mingus – інструментарій Python, який використовується для комп'ютерної музикології. Це дозволяє навчати мережу основ теорії музики та генерувати приклади випадкових музикальних композицій.

Python є простою та ефективною мовою у використанні при обробці великого обсягу даних та у використанні складних алгоритмів, яка має зручний для користувача синтаксис, що ефективно під час написання коду.

Також Python є гнучким у зміні стилю програмування (ООП, скриптування) і у його роботі з іншими мовами програмування.

Наступна варіант під назвою Java, основана на основі класів, яка розроблена для забезпечення якомога меншої залежності від реалізації. Це мова програмування загального призначення, призначена для того, щоб розробники додатків могли писати один раз та запускати де завгодно.

Java так само має значний вибір бібліотек для роботи з штучним інтелектом, такі як DeepLearning4J та TensorFlow, що правда їх менше, ніж у Python.

Java також корисна своєю портативністю на інші машини, що підтримують технологію Java Virtual Machine, простотою налагодження програми та інше.

Ця мова програмування має свою музичну бібліотеку jMusic, яка призначена для надання підтримки структури музичних даних, їхніх модифікацій та введення/виведення в різні формати файлів. Ця бібліотека може відображати ноти як партитуру.

Мова програмування C++ є найшвидшою у виконанні скомпільованого коду серед усіх розглянутих варіантів, особливо якщо враховувати той факт, що машинне навчання є трудомісткою задачею в плані пам'ятних ресурсів.

C++ також має в наявності музичну бібліотеку CFugue, який дає можливість відтворювати музичні ноти безпосередньо з програм C/C++, по суті схожа функціональністю з jMusic.

Варіант R є мовою програмування спеціально призначеною для статистичних розрахунків, аналізу та візуалізації даних.

Оскільки R вважається вузькоспеціалізованою мовою, так як в основному вона використовується для статистики, мова є менш гнучкою та загальною в порівнянні з Python.

Також R має гірший синтаксис у порівнянні з мовою Python.

Найбільш гідним показником мови R є природна реалізація та підтримка матричної арифметики та пов’язаних з ними структур даних, таких як вектори та матриці, що необхідно для реалізації моделі дипломного проекту. У цьому відношенні це нарівні з Matlab та Octave, і подібне впровадження в Python зазвичай включає лише пакет numpy, який є менш організованим та функціональним на відміну від приведених вище прикладів.

Відповідно до своєї спеціалізації, R має безліч потужних бібліотек, необхідних не тільки для статистики, а й для машинного навчання.

Останнім розглянутим варіантом є LISP. Не дивлячись на те, що LISP є найстаршою мовою серед усіх розглянутих варіантів, вона є доволі ефективною та гнучкою мовою для вирішування індуктивних задач та машинного навчання.

Найбільші особливості, які необхідні для розробки штучного інтелекту в LISP, є наступними:

– швидке створення прототипів;
– динамічне створення об’єкта;
– автоматичний збір сміття;
– підтримка символічних виразів.

Незважаючи на гнучкість для машинного навчання, найбільшим недоліком у LISP є відсутність підтримки відомих бібліотек машинного навчання.

Порівнявши всі надані технології, було підсумовано, що найбільш оптимальнішою технологією для вирішення проблеми цієї роботи є Python, так як

дана мова програмування є загальною при роботі з штучним інтелектом, маючи повний набір інструментів для його продукування.

Для здійснення порівняння Python з іншими мовами, було використано реалізацію формули Лейбніца для π. Роботи програм були емульовані за допомогою контейнера Docker та скрипта bash, який запускає роботу програм різних мов. Результати показані на рисунку 3.10.

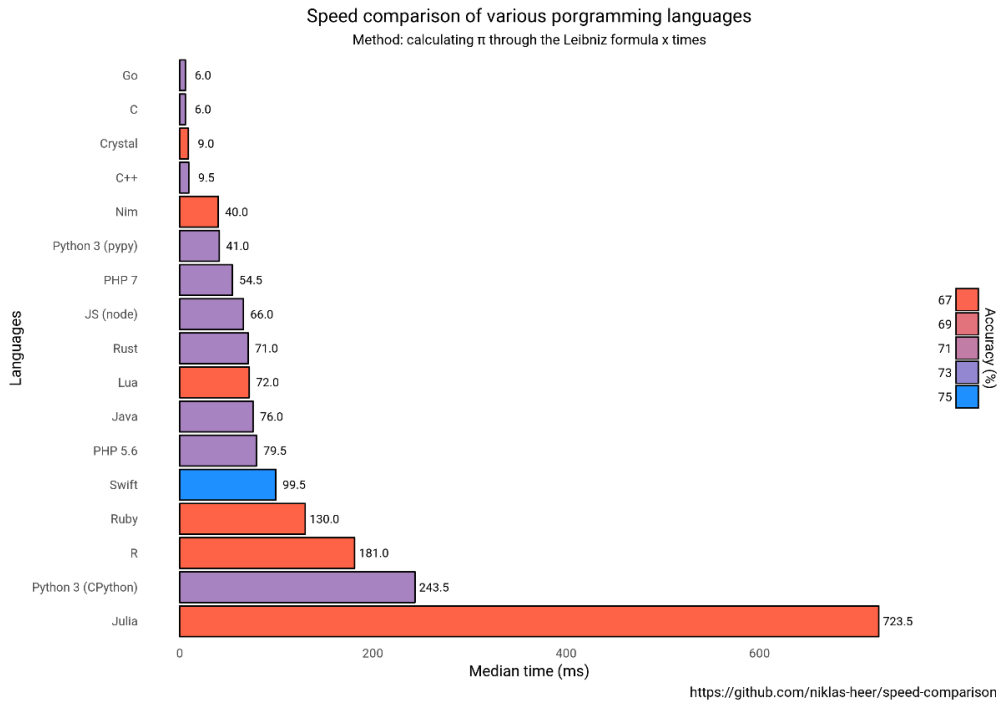


Рисунок 3.10 – Порівняння швидкості різних мов обчисленням π формулою Лейбніца

Необхідно обрати інтегроване середовище розробки, яке підтримує Python.

Був здійснений наступний пошук найоптимальнішого середовища розробки, серед якого були знайдені такі варіанти – Pycharm, Visual Studio Code, Spyder.

Pycharm є інтегрованим середовищем розробки специфічно орієнтованою для Python.

Основними компонентами, які необхідні для дипломного проекту, є:

– вбудований відлагоджувач Python та графічний модуль тестового модуля (які будуть необхідні для тестування програмного проекту);

– підтримка наукового стеку бібліотек, таких як Numpy, Matplotlib тощо.

– інші наукові інструменти, такі як інтегрований блокнот IPython, інтерактивна консоль Python та Anaconda.

– різна інтелектуальна допомога в кодуванні у вигляді навігації коду, рефакторингу, та інших особливостей Pycharm, які підвищать ефективність написання організованого коду для проекту.

Visual Studio Code є легковажною кросплатформерною версією Visual Studio, яка підтримує безліч мов для розробки, в основному, веб-технологій та додатків, в тому числі й Python.

Як і Pycharm, Visual Studio Code має відлагоджувач та рефакторинг для відповідного тестування програми та ефективного написання коду.

Visual Studio Code також має велику підтримку плагінів, які можуть бути ефективними для повноцінного написання Python-додатку.

Ще одна особливість Visual Studio Code що він має підтримку до створення файлів типу Jupyter Notebook – файл проекту з відкритим кодом Jupyter, який дозволяє легко поєднувати текст Markdown та виконуваний вихідний код Python на одному полотні, який називається блокнотом (notebook).

За допомогою такого файлу можна обмінюватися кодом Python, що буде містити живий код, рівняння, візуалізації та текст розповіді, що може бути корисним для тестування.

Spyder є ще одним легковажним кросплатформерним середовищем розробки, орієнтованим на спеціалізацію до науки про дані.

Як і двоє інших варіантів, Spyder має необхідний набір інструментарію для повноцінного інтегрованого середовища розробки, такого як відлагоджувач, рефакторинг, навігація коду, автозаповнення та інші.

Так само, як і Pycharm, Spyder має вбудовані наукові бібліотеки, які необхідні для дипломного проекту, такі як Numpy.

Оскільки середовище орієнтоване на науку про дані, Spyder має в наявності вбудований дисплей для графіки, виготовленої за допомогою Matplotlib, який візуалізує математичні об’єкти.

Таким чином, під час аналізу можливих варіантів, найбільш оптимізованим середовищем розробки було обрано Pycharm, так як воно є найбільш функціональним середовищем, необхідним для написання коду й має низку вбудованих біблоітек (а також можливість завантижити більше бібліотек в один проект), необхідних для програмної реалізації моделі дипломного проекту.

Для проекту буде використана безкоштовна версія цього середовища – Pycharm Commuinty останнього року.

Нарешті, для тренування моделі дипломної роботи необхідно вибрати датасети, або набори даних.

Їхній вибір має бути фундаментальним для гарної генерації музики. Набір даних повинен мати достатній розмір (тобто містити достатню кількість прикладів), щоб гарантувати точне навчання та майже однорідні категорії, щоб композиція вийшла цікавою та узагальненою, адже компроміс між розміром набору даних та його узгодженістю є однією з головних проблем при побудові глибинних генеративних моделей.

Було обрано декілька публічних різноманітних наборів даних та бібліотек, з якими можливо поекспериментувати та побачити які з них можуть надати можливість згенерувати найцікавішу мелодію:

– база даних MIDI класичних фортепіанних творів (Бетховен, Моцарт, Чопін та інші);

– набір символічної музики від Walder, величезний набір очищених та попередньо оброблених MIDI-файлів;

– набір даних Yamaha e-Piano Competition, в якому доступно від учасників записи MIDI спектаклів.

В результаті під час процесу програмної реалізації прийнятих рішень будуть використовуватись такі технології:

– мова форматування MIDI;

– мова програмування Python;

– інтегроване середовище розробки PyCharm Community Edition;

– бібліотека для машинного навчання TensorFlow;

– бібліотека для високорівневих математичних функцій NumPy;
– набір датасетів для тренування моделі.

3.3 Висновки

У розділі було розроблено архітектурний дизайн програмного забезпечення. Зокрема, проведено детальний опис кожного із спроектованих модулів. Кожний з функціональних модулів був смодельований у псевдокоди для простішого уявлення розроблюваного коду програмної реалізації дипломної роботи.

Також було здійснено аналіз усіх оптимальних варіантів щодо програмних засобів та вибір технологій для здійснення програмної реалізації цього дипломного проекту.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИРІШЕННЯ ПРОБЛЕМИ

4.1 Детальна реалізація

Коли тепер отримано всі необхідні проектуючі компоненти, можна переходити до програмної реалізації дипломного проекту.

Програмний додаток складається з декількох відокремлених модулів, які, взаємодіючи один з одним, дозволяють в реальному часі генерувати поліфонічний музичний твір.

Є такі модулі в програмному додатку:

– модуль графа моделі;

– основний керуючий модуль.

Обидва модулів функціонують незалежно від інших, і взаємодія відбувається тільки через керуючий модуль. Розглянемо ці модулі детальніше.

Модуль графа моделі ModelGraph є програмною реалізацією моделі тренувального процесу, розробленої в попередньому розділі. Він відіграє роль модульної структури, яка забезпечує можливість об’єднання декількох подмодулей, які працюють разом.

До складу модуля графа моделі входять наступні підмодулі:

– модуль вхідного ядра;

– модуль часових осей LSTM;

– модуль нотних осей LSTM;

– модуль втрат.

Розглянемо кожен з підмодулів детальніше.

Підмодуль вхідного ядра Input_Kernel, як і в модельній версії, бере з тензору None_State_Batch пару кожної ноти n та кожного кроку часу t та конвертує у розширений 80-елементовий вектор None_State_Expand, створюючи розмірність для часової та нотної інваріантності.

Згідно псевдокоду, була розроблена програмна реалізація цього підмодулю. Аргументами виступають масив input_data [batch_size x num_notes x

num_timesteps x 2], вхідні дані якого представляють те, що на попередньому кроці часу того, що ми намагаємося передбачити, цілі числа нижньої та вищої ноти Midi_low та Midi_high, та ціле число time_init, що вказує, де починається компонент ритму для пакету.

Підмодуль буде повертати сам None_State_Expand у вигляді масиву [batch_size x num_notes x num_timesteps x 80].

Код вхідного ядра наведено нижче:

```
def Input_Kernel(input_data, Midi_low, Midi_high, time_init):

# Захоплення розмірів input_data (batch_size та num_timesteps - змінна довжина)
    batch_size = tf.shape(input_data)[0]
    num_notes = input_data.get_shape()[1].value
    num_timesteps = tf.shape(input_data)[2]

    #Номер MIDI ноти (лише функція індексу нот)
    Midi_indices = tf.squeeze(tf.range(start=Midi_low, limit = Midi_high+1,
delta=1))
    x_Midi = tf.ones((batch_size, num_timesteps, 1,
num_notes))*tf.cast(Midi_indices, dtype=tf.float32)
    x_Midi = tf.transpose(x_Midi, perm=[0,3,1,2]) # shape = batch_size, num_notes,
num_timesteps, 1

    # part_pitchclass (лише функція індексу нот)
    Midi_pitchclasses = tf.squeeze(x_Midi % 12, axis=3)
    x_pitch_class = tf.one_hot(tf.cast(Midi_pitchclasses, dtype=tf.uint8),
depth=12)

    # part_prev_vicinity
    input_flatten = tf.transpose(input_data, perm=[0,2,1,3])
    input_flatten = tf.reshape(input_flatten, [batch_size*num_timesteps, num_notes,
2]) # channel for play and channel for articulate
    input_flatten_p = tf.slice(input_flatten, [0,0,0],size=[-1, -1, 1])
    input_flatten_a = tf.slice(input_flatten, [0,0,1],size=[-1, -1, 1])

    # зворотне ядро ідентичності
    filt_vicinity = tf.expand_dims(tf.eye(25), axis=1)
```

```
    #одновимірний згорнутий фільтр для кожного відтворювання нот та артикулярних
масивів
    vicinity_p = tf.nn.conv1d(input_flatten_p, filt_vicinity, stride=1,
padding='SAME')
    vicinity_a = tf.nn.conv1d(input_flatten_a, filt_vicinity, stride=1,
padding='SAME')

    #з'єднати назад і переставити стеки так, щоб відтворювально-артикульовані
цифри чергувалися
    vicinity = tf.stack([vicinity_p, vicinity_a], axis=3)
    vicinity = tf.unstack(vicinity, axis=2)
    vicinity = tf.concat(vicinity, axis=2)

    #переформувати за основними розмірами, потім поміняти осі
    x_vicinity = tf.reshape(vicinity, shape=[batch_size, num_timesteps, num_notes,
50])
    x_vicinity = tf.transpose(x_vicinity, perm=[0,2,1,3])

    #part_prev_context
    input_flatten_p_bool = tf.minimum(input_flatten_p,1)
    # 1 якщо відтворюється нота, 0 якщо не відтворюється.

    #ядро
    filt_context = tf.expand_dims(tf.tile(tf.eye(12), multiples=[(num_notes //
12)*2,1]), axis=1)

    context = tf.nn.conv1d(input_flatten_p_bool, filt_context, stride=1,
padding='SAME')
    x_context = tf.reshape(context, shape=[batch_size, num_timesteps, num_notes,
12])
    x_context = tf.transpose(x_context, perm=[0,2,1,3])

    #ритм (лише функція індексу осі часу плюс значення time_init)
    Time_indices = tf.range(time_init, num_timesteps + time_init)
    x_Time = tf.reshape(tf.tile(Time_indices, multiples=[batch_size*num_notes]),
shape=[batch_size, num_notes, num_timesteps,1])
    x_beat = tf.cast(tf.concat([x_Time%2, x_Time//2%2, x_Time//4%2, x_Time//8%2],
axis=-1), dtype=tf.float32)

    #нуль
    x_zero = tf.zeros([batch_size, num_notes, num_timesteps,1])

    #Фіналізований вектор
```

```
    Note_State_Expand = tf.concat([x_Midi, x_pitch_class, x_vicinity, x_context,
x_beat, x_zero], axis=-1)

    return Note_State_Expand
```

Підмодуль часових осей LSTM буде запускати ряд комірок LSTM за часовою віссю. Кожна нота та семпл в пакеті будуть виконуватися паралельно над вектором Note_State_Expand для кожної ноти паралельно з прив'язаними вагами.

Аргументами виступають масив input_data – тензор з розмірами [batch_size x num_notes x num_timesteps x input_size], та state_init – список тюльп LSTM ([batch_size * num_notes x num_units [layer]], [batch_size * num_notes x num_units [layer]]).

Підмодуль буде повертати тензор з розмірами [batch_size*num_notes x num_timesteps x num_units_final] та список тюльп LSTM ([batch_size*num_notes x num_units[layer]], [batch_size*num_notes x num_units[layer]]).

Код обробки часових осей LSTM наведено нижче:

```
      def LSTM_TimeWise_Training_Layer(input_data, state_init,
output_keep_prob=1.0):

      # batch_size і num_timesteps мають змінну довжину
    batch_size = tf.shape(input_data)[0]
    num_notes = input_data.get_shape()[1].value
    num_timesteps = tf.shape(input_data)[2]
    input_size = input_data.get_shape()[3].value

    num_layers = len(state_init)

    # Спресувати вхідні дані
    input_flatten = tf.reshape(input_data, shape=[batch_size*num_notes,
num_timesteps, input_size])

    # генерувати список комірок довжини, заданої початковим станом
    cell_list=[]
    num_states=[]
    for h in range(num_layers):
        num_states.append(state_init[h][0].get_shape()[1].value)
```

```
        lstm_cell = BasicLSTMCell(num_units=num_states[h], forget_bias=1.0,
state_is_tuple=True,activation=math_ops.tanh, reuse=None)
        lstm_cell = DropoutWrapper(lstm_cell, output_keep_prob=output_keep_prob)
        cell_list.append(lstm_cell)

    #Створити екземпляр багатошарової часової клітки
    multi_lstm_cell = tf.contrib.rnn.MultiRNNCell(cell_list, state_is_tuple=True)

    #Перебіг через часові кроки LSTM та генерування послідовності вихідних даних
за часом
    output_flat, state_out = tf.nn.dynamic_rnn(cell=multi_lstm_cell,
inputs=input_flatten, initial_state=state_init, dtype=tf.float32)

    output = tf.reshape(output_flat, shape=[batch_size, num_notes, num_timesteps,
num_states[-1]])

    return output, state_out
```

Підмодуль нотних осей LSTM LSTM_NoteWise_Layer буде запускати ряд комірок LSTM від низької ноти до високої ноти, щоб генерувати ймовірність вихідних нот для кожної ноти на кожному кроці часу для кожного пакету.

Аргументами виступають масив input_data [batch_size x num_notes x num_timestepsxsize_input], список тюльп LSTM state_init ([batch_size*num_time_steps x num_units[layer]], та output_keep_prob — плаваюче значення від 0 до 1, вказуючи утримання шару виключення.

Підмодуль буде повертати y = логіт (ймовірність = 1) з розмірами batch_size x num_notes x num_timesteps x 2 та note_gen з розмірами batch_size x num_notes x num_timesteps x 2.

Код обробки нотних осей LSTM наведено нижче:

```
    def LSTM_NoteWise_Layer(input_data, state_init, output_keep_prob=1.0):
    # batch_size and num_timesteps мають змінну довжину
    batch_size = tf.shape(input_data)[0]
    num_notes = input_data.get_shape()[1].value
    num_timesteps = tf.shape(input_data)[2]
    input_size = input_data.get_shape()[3].value
```

```
    num_layers = len(state_init)

    # Змінна форми введення
    # Розмірності batch_size та num_timesteps вхідних даних вирівнюються, щоб
трактувати їх як єдиний "пакетний" розмір для комірки LSTM
    notewise_in = tf.transpose(input_data, perm=[0,2,1,3])
    notewise_in = tf.reshape(notewise_in, shape=[batch_size*num_timesteps,
num_notes, input_size])

    # генерування список комірок LSTM з довжиною, заданою початковим станом
    # кожен шар має маску виключення
    cell_list=[]
    num_states=[]
    for h in range(num_layers):
        num_states.append(state_init[h][0].get_shape()[1].value)
        lstm_cell = BasicLSTMCell(num_units=num_states[h], forget_bias=1.0,
state_is_tuple=True,activation=math_ops.tanh, reuse=None)
        lstm_cell = DropoutWrapper(lstm_cell, output_keep_prob=output_keep_prob)
        cell_list.append(lstm_cell)

    #Застосування багатошарової клітинки часової осі
    multi_lstm_cell = tf.contrib.rnn.MultiRNNCell(cell_list, state_is_tuple=True)

    h_state = state_init
    p_gen_n= tf.zeros([batch_size*num_timesteps, 1])
    a_gen_n= tf.zeros([batch_size*num_timesteps, 1])

    y_list=[]
    note_gen_list=[]

    # Цикл по нотам для нотної LSTM, щоб отримати P(va(n)|va(<n))
    for n in range(num_notes):
        #об'єднати попередньо відібрані ноти відігравально-артикульованої
комбінації з виведенням за часом
        # здійснити зворотній зв'язок компонентів "відтворення", так і
"артикуляції" (артикулярний компонент - маскована версія)
        cell_inputs = tf.concat([notewise_in[:,n,:], tf.cast(p_gen_n, tf.float32),
tf.cast(a_gen_n, tf.float32)], axis=-1)

        h_final_out, h_state = multi_lstm_cell(inputs=cell_inputs, state=h_state)
```

```
        # Повністю підключений шар для створення 2 виходів, які відповідають
[logit (p = 1), logit (a = 1)],зберігаючи його у формі logit для зручності функцій
Tensorflow (logit = зворотний сигмоїд = відображення 1: 1 з імовірністю)
        y_n = tf.layers.dense(inputs=h_final_out, units=2, activation=None)

        # вибірка відтворення ноти/артикуляції з використанням ймовірності події =
сигмоїд(y_n)
        note_gen_n = tf.distributions.Bernoulli(logits=y_n).sample()

        # обнулити всі артикуляції, коли одна і та ж нота на одному і тому ж
часовому кроці того ж пакету не відтворюється
        p_gen_n = tf.slice(note_gen_n, [0,0], [-1,1])
        a_gen_n = tf.slice(note_gen_n, [0,1], [-1,1])
        a_gen_n = tf.multiply(p_gen_n, a_gen_n) # якщо дана нота не відтворена (=
0), автоматично встановити артикуляцію на нуль.
        note_gen_n = tf.concat([p_gen_n, a_gen_n], axis=1)

        # Переформувати 1-у розмірність назад у розмірність пакетів та часових
кроків
        y_n_unflat = tf.reshape(y_n, shape=[batch_size, num_timesteps, 2])
        note_gen_n_unflat = tf.reshape(note_gen_n, shape=[batch_size,
num_timesteps, 2])
        # Приєднання до нотного списку
        y_list.append(y_n_unflat)
        note_gen_list.append(note_gen_n_unflat)

    # Перетворити вихідний список на тензор
    y_out = tf.stack(y_list, axis=1)
    note_gen_out = tf.stack(note_gen_list, axis=1)

    return y_out, note_gen_out
```

Завдання підмодуля функції втрати Loss_Function полягає у тому, щоб Logits_Output на кроці часу *t* передбачав Note_State_Batch на кроці часу *t+1*. Записи в Logits_Output на кроці часу t були створені записами в Note_State_Batch на кроці часу *t*. Для того, щоб правильно вирівняти тензори для обчислення функції втрат, останній зріз часу Logits_Output та перший зріз Note_State_Batch видаляються.

Аргументами виступають масив Note_State_Batch з формою [batch_size x num_notes x num_timesteps x 2], масив y_out, який є пакет логітів (batch_size x num_notes x num_timesteps x 2].

Підмодуль буде повертати саму втрату у вигляді скалярного дійсного числа та логарифмічну правдоподібність.

Код функції втрат наведено нижче:

```
def Loss_Function(Note_State_Batch, y_out):

    batch_size = tf.shape(y_out)[0]
    num_notes = y_out.get_shape()[1].value
    num_timesteps = tf.shape(y_out)[2]

    # Вирівняти y_out із наступними вхідними даними note_state на наступному етапі
    y_align = tf.slice(y_out, [0,0,0,0],[batch_size, num_notes, num_timesteps-1,
2])
    Note_State_Batch_align = tf.slice(Note_State_Batch, [0,0,1,0],[batch_size,
num_notes, num_timesteps-1, 2])

    # обчислення логарифмічної правдоподібності
    cross_entropy = tf.nn.sigmoid_cross_entropy_with_logits(logits=y_align,
labels=Note_State_Batch_align)

    # якщо нота не відтворена, замаскувати термін втрати для артикуляції
    cross_entropy_p = cross_entropy[:,:,:,0]
    cross_entropy_a = cross_entropy[:,:,:,1] * Note_State_Batch_align[:,:,:,0]
    cross_entropy = tf.stack([cross_entropy_p, cross_entropy_a], axis=-1)

    # обчислення функції витрат
    Loss = tf.reduce_mean(cross_entropy) * 2 # negative log-likelihood of batch
(factor of 2 for both play and articulate)

    # обчислення логарифмічної правдоподібності нот за один часовий крок
    Log_likelihood = -Loss*num_notes

    return Loss, Log_likelihood
```

Наступний модуль Main є основним керуючим та представляє собою клас, що ініціалізує всі інші модулі та контролює весь процес роботи генератора музики. У його завдання входять:

– ініціалізація всіх модулів;
– забезпечення взаємодії між різними модулями;
– передача вхідних та вихідних даних;
– контроль над процесом генерації в цілому.

При запусканні цього модуля спочатку відбувається ініціалізація всіх підмодулів та необхідних Python бібліотек. Далі зчитується поточна робоча папка, де є специфікована бібліотека MIDI-файлів. В нашому випадку це збірник класичних фортепіанних творів, таких як Бетховен, Моцарт та інші (рисунок 4.1).

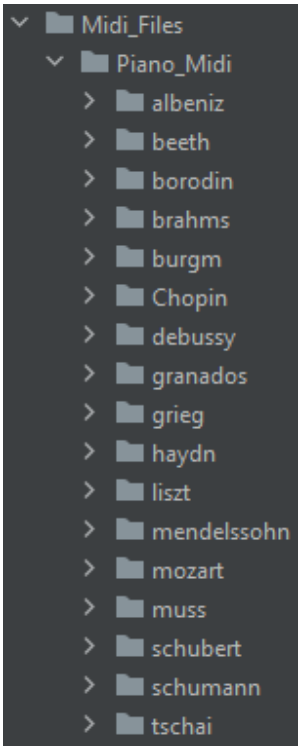


Рисунок 4.1 – Папка MIDI-файлів класичних фортепіанних творів

Під час цього етапу користувач може ввести список папок для вилучення MIDI-файлів; кількість максимуму часових кроків, необхідних для збереження кожного MIDI-файлу та кількість частин музичних творів, які потрібно відкласти для незалежної перевірки під час навчання.

Далі модуль будує вхідне ядро, часову LSTM, нотну LSTM, функцію втрат та оптимізатор для відтворення моделі тренувального процесу та обробки MIDI-файлів до генерації власного музичного твору. Під час цього етапу користувач може ввести приховані розміри для часових та нотних етапів LSTM (список для кожного) як num_t_units та num_n_units.

```
# Побудова графу моделі:
tf.reset_default_graph()
print('Початок процесу будування графу.')
#Взяття кількості нот із вибірки
num_notes = sample_state.shape[1]

# Заповнювачі графічного вводу
Note_State_Batch = tf.placeholder(dtype=tf.float32, shape=[None, num_notes, None,
2])
time_init = tf.placeholder(dtype=tf.int32, shape=())

# Генерування розширеного тензору з пакету матриць стану нот
Note_State_Expand = Input_Kernel(Note_State_Batch, Midi_low=24, Midi_high=101,
time_init=time_init)

print('Форма Note_State_Expand = ', Note_State_Expand.get_shape())

# Часовий LSTM Time Wise граф тренування
num_t_units=[200, 200]
output_keep_prob = tf.placeholder(dtype=tf.float32, shape=())

# Генерування заповнювачу початкового стану (при t = 0)
timewise_state=[]
for i in range(len(num_t_units)):
    timewise_c=tf.placeholder(dtype=tf.float32, shape=[None, num_t_units[i]])
#None = batch_size * num_notes
    timewise_h=tf.placeholder(dtype=tf.float32, shape=[None, num_t_units[i]])
    timewise_state.append(LSTMStateTuple(timewise_h, timewise_c))

timewise_state=tuple(timewise_state)

timewise_out, timewise_state_out =
LSTM_TimeWise_Training_Layer(input_data=Note_State_Expand,
state_init=timewise_state, output_keep_prob=output_keep_prob)

print('Форма часового виводу даних = ', timewise_out.get_shape())

# Нотний LSTM граф

num_n_units = [100, 100]

# Генерування заповнювачу початкового стану (при n = 0)
notewise_state=[]
for i in range(len(num_n_units)):
    notewise_c=tf.placeholder(dtype=tf.float32, shape=[None, num_n_units[i]])
#None = batch_size * num_timesteps
    notewise_h=tf.placeholder(dtype=tf.float32, shape=[None, num_n_units[i]])
    notewise_state.append(LSTMStateTuple(notewise_h, notewise_c))

notewise_state=tuple(notewise_state)
```

```
y_out, note_gen_out = LSTM_NoteWise_Layer(timewise_out, state_init=notewise_state,
output_keep_prob=output_keep_prob)

p_out = tf.sigmoid(y_out)
print('Форма y_out = ', y_out.get_shape())
print('Форма генерованих вибірок = ', note_gen_out.get_shape())

# Функція витрат та оптимізатор

loss, log_likelihood = Loss_Function(Note_State_Batch, y_out)
optimizer = tf.train.AdadeltaOptimizer(learning_rate = 1).minimize(loss)
print('Граф побудовано')
```

Після попередньої обробки MIDI-файлу проходить етап тренування моделі, де проводиться певна кількість пакетів до тренування і на кожні 100 пакетів запускається пакет валідації та реєструє валідаційні втрати на додаток до тренувальних втрат. Тут користувач може вказати кількість епох; їхній розмір (кількість матриць стану нот); кількість часових кроків, на яких потрібно тренуватися (має бути менше максимальної кількості часових кроків); назва моделі; назва файлу, де зберігається модель та коефіцієнт випадання – keep_prob.

```
# Тренування моделі

start_time = time.time()
N_epochs = 50000
loss_hist=[]
loss_valid_hist=[]
restore_model_name = 'Long_Train'
save_model_name = 'Long_Train_256'
batch_size = 5
num_timesteps = 256
keep_prob=.5

# Зберігання моделі
Output_Directory = Working_Directory + "/Output/" + save_model_name
directory = os.path.dirname(Output_Directory)

try:
    print('Створення нової папки призначення')
    os.mkdir(directory)
except:
    print('Папка призначення вже існує')

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    saver = tf.train.Saver()

    # спроба відновити попередньо треновану
    if restore_model_name is not None:
        Load_Directory = Working_Directory + "/Output/" + restore_model_name

        print("Завантаження моделі з: {}".format(restore_model_name))
        saver.restore(sess, Load_Directory + '/{}'.format(restore_model_name))
```

```
    # Початкові стани
    timewise_state_val=[]
    for i in range(len(num_t_units)):
        c_t = np.zeros((batch_size*num_notes, num_t_units[i])) #start every batch
with zero state in LSTM time cells
        h_t = np.zeros((batch_size*num_notes, num_t_units[i]))
        timewise_state_val.append(LSTMStateTuple(h_t, c_t))

    notewise_state_val=[]
    for i in range(len(num_n_units)):
        c_n = np.zeros((batch_size*num_timesteps, num_n_units[i])) #start every
batch with zero state in LSTM time cells
        h_n = np.zeros((batch_size*num_timesteps, num_n_units[i]))
        notewise_state_val.append(LSTMStateTuple(h_n, c_n))

    # Тренувальний цикл
    for epoch in range(N_epochs+1):

        # Генерування випадкового файлу тренувальних даних
        if (epoch % 100 == 0):
            print('Отримання пакету даних')
            _, batch_input_state = multi_training.getPieceBatch(training_pieces,
batch_size, num_timesteps)
            batch_input_state = np.array(batch_input_state)
            batch_input_state = np.swapaxes(batch_input_state, axis1=1, axis2=2)

        # Запуск сесії
        feed_dict = {Note_State_Batch: batch_input_state, timewise_state:
timewise_state_val, notewise_state: notewise_state_val, time_init: 0,
output_keep_prob: keep_prob}
        loss_run, log_likelihood_run, _, note_gen_out_run = sess.run([loss,
log_likelihood, optimizer, note_gen_out], feed_dict=feed_dict)

        # Періодичне зберігання моделі та історії втрат
        if (epoch % 1000 == 0) & (epoch > 0):
            save_path = saver.save(sess, Output_Directory +
'/{}'.format(save_model_name))
            print("Модель збережена в файлі: %s" % save_path)
            np.save(Output_Directory + "/ training_loss.npy", loss_hist)
            np.save(Output_Directory + "/ valid_loss.npy", loss_valid)

        # Регулярне обчислення витрат валідації та зберігання тренувальних та
валідаційних витрат
        if (epoch % 100) == 0 & (epoch > 0):
            # Обчислення валідаційних витрат
            _, batch_input_state_valid =
multi_training.getPieceBatch(validation_pieces, batch_size, num_timesteps)
            batch_input_state_valid = np.array(batch_input_state_valid)
            batch_input_state_valid = np.swapaxes(batch_input_state_valid, axis1=1,
axis2=2)
            feed_dict = {Note_State_Batch: batch_input_state_valid, timewise_state:
timewise_state_val, notewise_state: notewise_state_val, time_init: 0,
output_keep_prob: keep_prob}
            loss_valid, log_likelihood_valid = sess.run([loss, log_likelihood],
feed_dict=feed_dict)

            print('епоха = ', epoch, ' / ', N_epochs, ':')
            print('Втрата тренування = ', loss_run, '; Логарифмічна
правдоподібность тренування = ', log_likelihood_run)
            print('Втрата валідації = ', loss_valid, '; Логарифмічна
правдоподібность валідації = ', log_likelihood_valid)

            loss_hist.append(loss_run)
```

```
            loss_valid_hist.append(loss_valid)

        # Періодичне генерування вибірок музичних творів

end_time = time.time()

print('Час тренування = ', end_time - start_time, ' секунд')

# Будування графіків історії втрат
plt.plot(loss_hist, label="Training Loss")
plt.plot(loss_valid_hist, label="Validation Loss")
plt.legend()
plt.show
```

Далі проходить генерування фінального тесту та логарифмічна правдоподібность й нарешті проводиться саме генерування музичного файлу з вектору початкової ноти, що оброблявся на протязі роботи програми. Тут користувач може вказати довжину пісні в 16-нотних кроках; розмір пакету створеної музики та коефіцієнт випадання – keep_prob.

```
# Будування графіків історії втрат
# Генерація музики
T_gen=64*16
batch_gen_size=10
keep_prob=.5
music_model_name = save_model_name

#Завантаження моделі
Load_Directory = Working_Directory + "/Output/" + music_model_name
directory = os.path.dirname(Load_Directory)


try:
    os.mkdir(directory)
except:
    print('Папка вже існує.')

# Початок роботи з початкової Note_State_Batch з розміром 't' = 1
notes_gen_initial = np.zeros((batch_gen_size, num_notes, 1,2))

# Початкові стани
notes_gen = notes_gen_initial

timewise_state_val=[]
for i in range(len(num_t_units)):
    c = np.zeros((batch_gen_size*num_notes, num_t_units[i])) #перший крок часу з
нульовим станом у комірках часового LSTM
    h = np.zeros((batch_gen_size*num_notes, num_t_units[i]))
    timewise_state_val.append(LSTMStateTuple(h, c))

notewise_state_val=[]
for i in range(len(num_n_units)):
    c = np.zeros((batch_gen_size*1, num_n_units[i]))
    h = np.zeros((batch_gen_size*1, num_n_units[i]))
    notewise_state_val.append(LSTMStateTuple(h, c))
```

```
notes_gen_arr=[]

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    saver = tf.train.Saver()

    print("Завантаження моделі з: {}".format(music_model_name))
    saver.restore(sess, Load_Directory + '/{}'.format(music_model_name))

    for t in range(T_gen):
        feed_dict = {Note_State_Batch: notes_gen, timewise_state:
timewise_state_val, notewise_state: notewise_state_val, time_init: t % 16,
output_keep_prob: keep_prob}
        timewise_state_val, notes_gen = np.squeeze(sess.run([timewise_state_out,
note_gen_out], feed_dict = feed_dict), axis=2)
        notes_gen_arr.append(np.squeeze(notes_gen))

        if t % 50 == 0:
            print('Крок часу = ', t)

# Зберігання згенерованих нот у .MIDI-файлі
notes_gen_out = np.stack(notes_gen_arr, axis=2)
notes_gen_out = np.swapaxes(notes_gen_out, axis1=1, axis2=2)

for iter in range(batch_gen_size):
    file = Load_Directory + "/Epoch " + str(epoch) + " Music_Dropout" + str(iter)
midi_out=midi_musical_matrix.noteStateMatrixToMidi(notes_gen_out[iter,:,:,:],
name=file)

print('Midi файл було збережено.')
```

Усі моделі, графіки втрат та сформовані MIDI-файли автоматично зберігаються у названої користувачем підпапці, що в папці Output, яка зарезервована для всіх створених файлів.

Повний код програмного забезпечення наведений в додатку Б.

4.2 Тестування програмного забезпечення

Для перевірки на швидкодію генерування музичних творів штучним інтелектом дипломного проекту, необхідно провести тестування.

Для кількісного аналізу у принципі показу ефективності логарифмічної правдоподібності було порівняно декілька моделей двонаправленої конфігурації з моделю дипломної роботи, отриманою тренуванням протягом часу, де:

– «Випадково» показує ефективність вибору відтворення кожної ноти з 50% ймовірністю;

– TP-LSTM-NADE: паралельно пов’язана модель LSTM-NADE, для якої було використано два шари LSTM з 200 вузлами в кожному та 100 прихованих юнітів у модифікованому шарі NADE;

– BALSTM: Двоосьовий LSTM з віконним і двостороннім входом, для якої було використано два шари LSTM у напрямку осі часу з 200 вузлами кожен і два шари LSTM у напрямку осі ноти зі 100 вузлами кожен.

Усі моделі було протестовано на наборі даних Piano-Midi.de. Два значення представляють найкращі та середні показники у 100 випробуваннях, масштабовані на 88/78, щоб нормалізувати кількість нот.

Результати було організовано у таблицю 4.1.

Таблиця 4.1 – Порівняння систем щодо архітектур та стратегій

Модель	Логарифмічна правдоподібность	Натреновано годин
Випадково	-61	--
TP-LSTM-NADE	-5.44, -5.49	24 - 48
BALSTM	-4.90, -5.00	24 - 48
Модель дипломного проекту	-5.16, -6.59	1

Результат, отриманий цією моделлю, є незначним покращенням порівняно з попередньою роботою оригінального автора. Продуктивність моделі сильно постраждала через менший вплив часу навчання.

У своєму блозі, Джонсон відмітив, що модель було натреновано протягом 22-24 годин, щоб зафіксувати якість його музичних вибірок, тоді як навчання цієї моделі займало лише одну годину.

Більшу проникливість щодо тренування моделі можна отримати, переглянувши графіки тренування, зроблені за допомогою візуалізатора

Tensorboard. Це видно на рисунку 4.2, що втрати поступово зменшуються, коли ми рухаємось до більшої кількості епох (ітерацій).

Спостерігається, що у вихідній точці витрати становлять приблизно 0,0286, але, рухаючись далі, функція витрат швидко зближується, а потім стає стійкою і повільною до кінця і зменшується приблизно до 0,002098 за 200-у епоху. На 120-й епосі спостерігається величезне збільшення витрат, що є відхиленням від цього процесу. Витрати сягають приблизно до 0,0718.

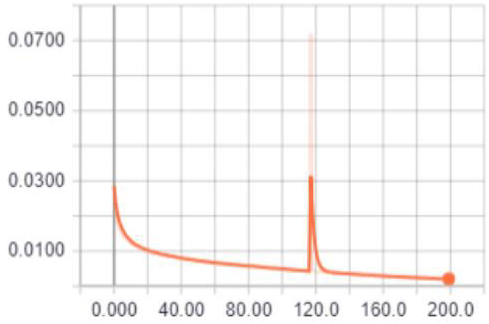


Рисунок 4.2 – Графік кількості витрат протягом 200 ітерацій на Piano Midi файлах

Рис. 4.3 описує точність моделі для всіх 200 епох. Помічено, що точність на початковому рівні становить 71%, вона лінійно зростає і з 20-ї по 25-у епоху досягає 90%. Після 25-ї епохи точність зростає з меншим градієнтом і стабільно. Незважаючи на викиди на 120-му кроці, точність не сильно впливає, явно визначаючи стабільність цієї моделі. На останню епоху, точність, що прогнозувалася за моделлю становить 97,23%.

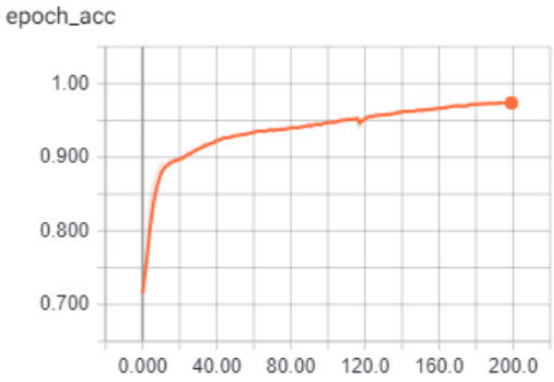


Рисунок 4.3 – Графік точності навчання протягом 200 ітерацій на Piano Midi файлах

Модель приймає введення “Матриці стану нот” у вигляді пакету, що використовується для налаштування вагових коефіцієнтів та упереджень у моделі. Здійснивши огляд на витрати пакету, показані на рисунку 4.4 та точність пакету на рисунку 4.5, спостерігається, що модель змінювалась приблизно 1 460 000 разів.

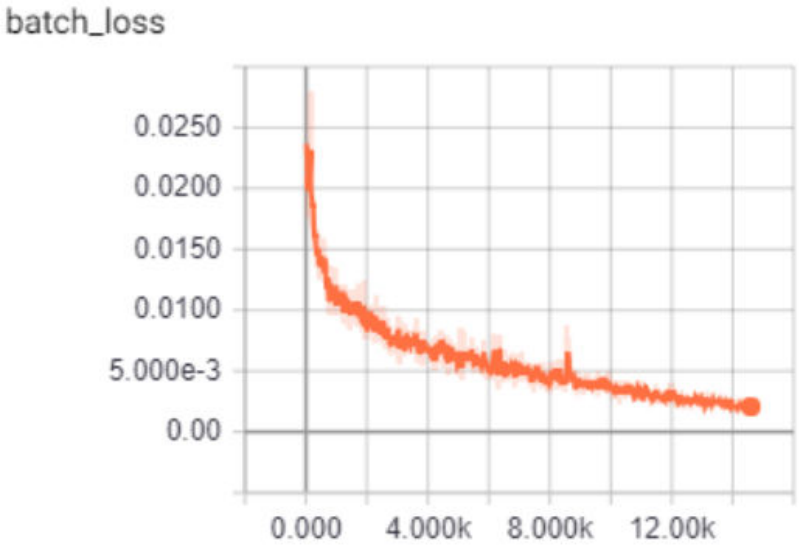


Рисунок 4.4 – Графік витрат пакетів під час тренування моделі

На цих рисунках видні скачки, що вказує відбирання нових пакетів. На них також спостерігається, що прогнозована ними точність зміни не надто велика, але у випадку витрат, прогнозованих для пакету, різниця є трохи більшою, ніж очікувалося. Величина витрат, прогнозована на рисунку 4.4 для останнього пакету є 0,00196, а точність на рисунку 4.5 – 97,27%.

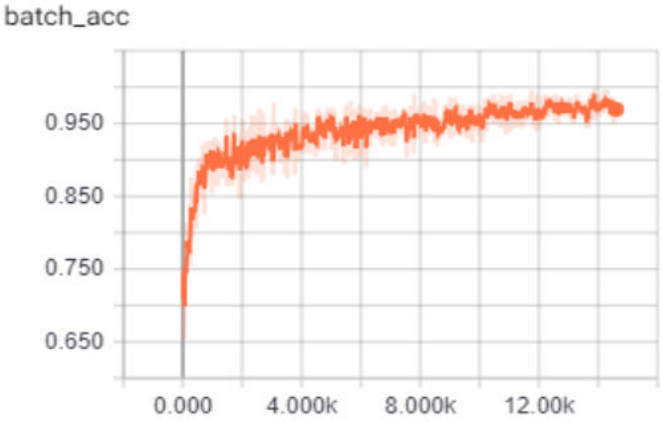


Рисунок 4.5– Графік точності пакетів під час тренування моделі

Було відмічено, що у статті Джонсона повідомляється про оптимізацію з використанням RMSprop, тоді як в цьому проекті використовувався Adadelta, що могло вплинути на швидкодію моделі дипломної роботи також.

Здійснюючи якісний аналіз, можна зрозуміти, що вибірки, згенеровані цією моделлю, виробляють непоганий ритм і добре створюють невеликі відрізки нот з мелодією, простою гармонією, а в деяких випадках навіть контрапунктом. Ці відрізки нот є поліфонічними та надають відчуття узгодженості.

Є недоліки в згенерованих творах, таких як недостатня кількість пауз та хаотична структура, але на це впливає чинник кількості часу, на який модель здійснює свій тренувальний процес. Різниця в структурованості музики, створеною моделлю, що натренована протягом 30 хвилин та моделлю, що натренована протягом двох годин є великою.

В основному, з отриманими результатами було зроблено висновок, що жодних аномалій у програмному забезпеченні не знайдено.

4.3 Висновки

Отже, для реалізації моделі тренувального процесу та методу обробки музичних даних, спроектованих в попередньому розділі, було розроблено програмний продукт генерування музичних творів із використанням штучного інтелекту, що дозволяє створювати гармонійну поліфонічну музику з втручанням користувача для підвищення організації та оригінальності мелодії.

Створений програмний продукт здатний за справедливу кількість часу згенерувати музичні твори, що може використовувати минулу інформацію для обробки її в майбутній твір (що допомогло в об’єднанні шарів рекурентних нейронних мереж), та має чітке розуміння музичної структури в плані обєднання частин існуючих музичних творів у новий згуртовано. Дана особливість сходиться з нашим завданням до розроблення моделі на основі вивчення теорії музики та аналізу музичних творів.

Продукт також має більшу кількість гіперпараметрів, що користувач може змінити для генерування музичних творів, на яких він має своє чітке уявлення, на відміну від оригінального методу Джонсона, що задовольняє критерій розробки методу обробки музичних треків із врахуванням введення даних користувача.

Порівняння показників якості розробленого ПЗ з аналогами інших моделей показало, що для цієї моделі необхідно провести більше експериментів у плані збільшення кількості часу тренування та зміни певних технологій, таких як оптимізатори або набори даних, щоб побачити чи приведуть дані зміни до покращення структури музичних творів, яка буде їхня якість з іншими музичними жанрами тощо.

Покращення програмного продукту є перспективним, так як надалі існує можливість вирішити оптимально більше проблем, пов’язаних із генеруванням музики та зробити звучання музичних творів більше натуральним і схожим до творів, створених людиною.

ВИСНОВКИ

На основі результаті виконаних теоретичних та практичних досліджень можна зробити наступні висновки.

У першому розділі проведено аналіз теоретичних джерел та формулювання постановки задачі. Під час аналізу та опису предметної області дипломної роботи було сформульовано мету та завдання. Огляд існуючих архітектур дав можливість звузити область до глибинних технологій, так як вони є доволі сучасним рішенням проблеми дипломного проекту.

У другому розділі проведено порівняльну оцінку на основі аналізу існуючих моделей та методів, що показала варіативність проблем, з якими стикалися при розробці системи генерування музичних творів й кожен підхід пропонував власне вирішення проблеми. Було розроблено модель створення музики на основі вивчення штучного інтелекту теорії музики та аналізу музичних творів, а також вдосконалено метод обробки музичних треків із врахуванням введення даних користувача.

У третьому розділі спроектовано архітектуру для тренувального процесу штучного інтелекту на основі «двоосьової» конфігурації, що відображає тренування та валідацію нейронної мережі в чисельній формі. За допомогою такої мережі здійснюється тренування на створення поліфонічних творів з гармонійно-мелодійною структурою між нотами та інваріантністю в часі та нотах для створення різноманітності в мелодії.

У четвертому розділі здійснено програмну реалізацію дипломної роботи. Детально описані модулі системи, що відтворюють розроблювану модель й метод, описаний у 3-му розділі. Розроблений програмний продукт може використовуватись при створенні оригінальних авторських музичних композицій,. Також було здійснено тестування, яке показало кращі результати порівняно з іншими моделями.

Отже, результатом дипломної роботи є технологія генерування музики з використанням штучного інтелекту. Запропоновані вдосконалення дають

можливість збільшити інтерактивність між людиною та програмним забезпеченням під час написання музичних творів, за рахунок підвищення написання музики зі сторони програмного забезпечення, а також генерування оригінальної, структурованої музики зі сторони людини.

Отримані результати представляються перспективними для подальшого дослідження, наприклад витрата більшого часу тренування моделі, а також для експерименту у підставленні різних технологій для порівняння якості генерування музичних творів з іншими чинниками.

За результатами дослідження, опублікована одна наукова стаття [1] та тези доповіді на науковій конференеції [2].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бабич І. Р. Підходи до розробки штучного інтелекту та їхній вплив на автоматизацію написання музики [Електронний ресурс] / І. Р. Бабич, О. М. Яшина // Modern scientific researches. – 2020. – № 12-01. – С. 35–39. – Режим доступа: https://www.modscires.pro/index.php/msr/issue/view/msr12-01 (дата звернення: 29.06.2020). – Назва з екрана.

2. Бабич І. Р. Модель тренувального процесу та метод обробки музичних даних програмної системи генерування музичних творів із використанням штучного інтелекту / І.Р. Бабич, О.М. Яшина // Міжнародний науково-технічний журнал «ВОТТП». – Хмельницький, 2020. – №1. – С. 9-17.

3. Pachet F. Beyond the cybernetic jam fantasy: The Continuator [Text] / Francois Pachet // Computer Graphics and Applications – 2004. – №4. – P. 31-35.

4. Mozer M. C. Neural network composition by prediction: Exploring the benefits of psychophysical constraints and multiscale processing / Michael C. Mozer // Connection Science – 1994. – №6. – P. 247-280.

5. Steedman M. A generative grammar for Jazz chord sequences / Mark Steedman // Music Perception – 1984. – №2. – P.52-77.

6. Ebcioglu K. An expert system for harmonizing four-part chorales / Kemal Ebcioglu // Computer Music Journal – 1988. – №12. – P.43-51.

7. Fiebrink R. The machine learning algorithm as creative musical tool [Електронний ресурс] / Rebecca Fiebrink, Baptiste Caramiaux // arXiv. – Режим доступу: https://arxiv.org/abs/1611.00379 (дата звернення: 01.11.2016). – Назва з екрана.

8. Cruz J. Deep Learning vs Markov Model in Music Generation [Електронний ресурс] / Jeffrey Cruz, David Paul Benjamin // DigitalCommons@Pace. – Режим доступу: https://digitalcommons.pace.edu/cgi/viewcontent.cgiarticle=1228&context=honorscollege_theses (дата звернення: 01.05.2019) – Назва з екрана.

9. Rumelhart D. E. Learning representations by back-propagating errors / David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams // Nature – 1986. – №323. – P.533-536.

10. Quoc V. Le. Building high-level features using large scale unsupervised learning [Електронний ресурс] / Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean, Andrew Y. Ng // International Conference on Machine Learning, Edinburgh. – 2012. – №29 – Режим доступу: https://arxiv.org/pdf/1112.6209.pdf (дата звернення: 12.07.2012) – Назва з екрана.

11. Hinton G. E. Learning and relearning in Boltzmann machines / Geoffrey E. Hinton, Terrence J. Sejnowski // Explorations in the Microstructure of Cognition – 1986. – №1. – P. 282–317.

12. Goodfellow I. Deep Learning [Електронний ресурс] / Ian Goodfellow, Yoshua Bengio, Aaron Courville // MIT Press – Режим доступу: https://mitpress.mit.edu/books/deep-learning (дата звернення: 18.11.2016) – Назва з екрана.

13. Sun F. Composing and harmonizing music with neural networks [Електронний ресурс] / Felix Sun // Github – Режим доступу: https://fephsun.github.io/2015/09/01/neural-music.html (дата звернення: 21.12.2017 р.). – Назва з екрана.

14. Moon T. RnnDrop: a novel dropout for RNNs in ASR [Електронний ресурс] / Moon T., Choi H., Lee H., Song I. // Automatic Speech Recognition and Understanding – 2015. – Режим доступу: https://ieeexplore.ieee.org/abstract/document/7404775 (дата звернення 11.02.2016). – Назва з екрана.

15. arXiv.org [Електронний ресурс] : [архів з відкритим доступом]. – Електронні дані (1 789 904 записів). – США : Корнельський університет, 2020. – Режим доступу: https://arxiv.org/pdf/1212.5701.pdf (дата звернення 22.12.2012). – Назва з екрана.

ДОДАТОК А

(обов’язковий)

ДІАГРАМА МОДЕЛІ ТРЕНУВАЛЬНОГО ПРОЦЕСУ

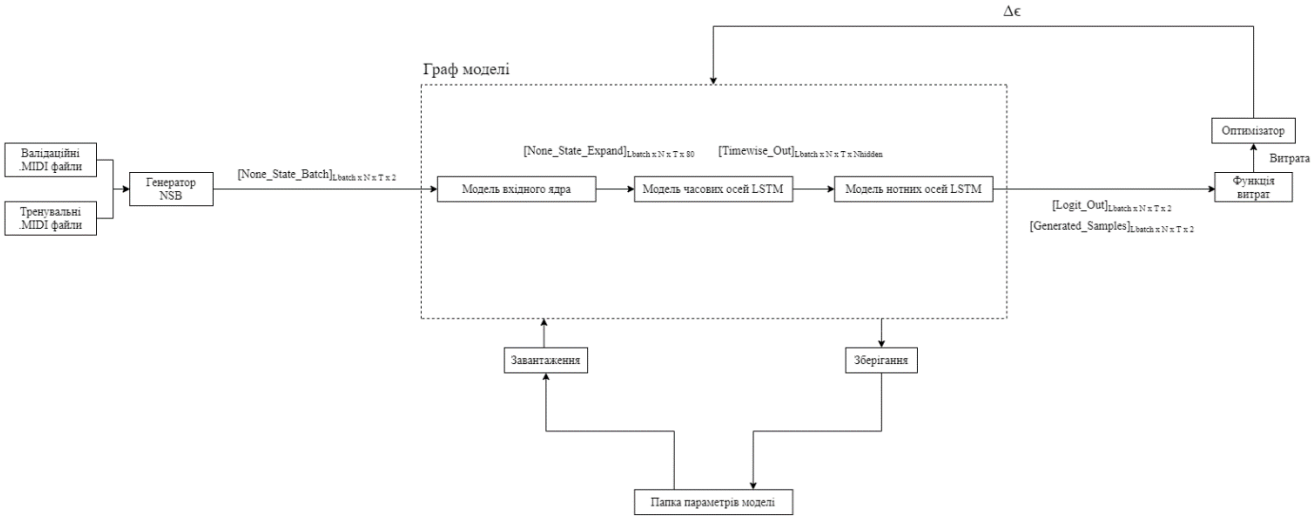


Рисунок А.1 – Повна діаграма моделі тренувального процесу

ДОДАТОК Б

(обов'язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

Б.1 Програмний код модуля Main:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import time
import os
import random
import midi_musical_matrix
import Training
from tensorflow.contrib.rnn import LSTMStateTuple
from ModelGraph import Input_Kernel, LSTM_TimeWise_Training_Layer,
LSTM_NoteWise_Layer, Loss_Function

Working_Directory = os.getcwd()
Music_Directory = Working_Directory + "/Midi_Files/Piano_Midi/"
print(Working_Directory)
Midi_Directories = ["albeniz", "beeth", "borodin", "brahms", "burgm", "Chopin",
"debussy", "granados", "grieg", "haydn", "liszt", "mendelssohn", "mozart", "muss",
"schubert", "schumann", "tschai"]
max_time_steps = 256 # only files atleast this many 16th note steps are saved
num_validation_pieces=10

training_pieces={}
for f in range(len(Midi_Directories)):
    Training_Midi_Folder = Music_Directory + Midi_Directories[f]
    training_pieces = {**training_pieces,
**Training.loadPieces(Training_Midi_Folder, max_time_steps)}

validation_pieces={}
for v in range(num_validation_pieces):
    index = random.choice(list(training_pieces.keys()))
    validation_pieces[index] = training_pieces.pop(index)

print('')
print('Кількість тренувальних даних = ', len(training_pieces))
print('Кількість валідаційних даних = ', len(validation_pieces))

practice_batch_size = 15
practice_num_timesteps = 128

_, sample_state = Training.getPieceBatch(training_pieces, practice_batch_size,
practice_num_timesteps)
sample_state = np.array(sample_state)
sample_state = np.swapaxes(sample_state, axis1=1, axis2=2)
print('Вибірка пакету стану вхідних даних: shape = ', sample_state.shape)

tf.reset_default_graph()
print('Початок процесу будування графу.')
num_notes = sample_state.shape[1]
```

```
Note_State_Batch = tf.placeholder(dtype=tf.float32, shape=[None, num_notes, None,
2])
time_init = tf.placeholder(dtype=tf.int32, shape=())

Note_State_Expand = Input_Kernel(Note_State_Batch, Midi_low=24, Midi_high=101,
time_init=time_init)

print('Форма Note_State_Expand = ', Note_State_Expand.get_shape())

num_t_units=[200, 200]
output_keep_prob = tf.placeholder(dtype=tf.float32, shape=())

timewise_state=[]
for i in range(len(num_t_units)):
    timewise_c=tf.placeholder(dtype=tf.float32, shape=[None, num_t_units[i]])
    timewise_h=tf.placeholder(dtype=tf.float32, shape=[None, num_t_units[i]])
    timewise_state.append(LSTMStateTuple(timewise_h, timewise_c))

timewise_state=tuple(timewise_state)


timewise_out, timewise_state_out =
LSTM_TimeWise_Training_Layer(input_data=Note_State_Expand,
state_init=timewise_state, output_keep_prob=output_keep_prob)

print('Форма часового виводу даних = ', timewise_out.get_shape())

num_n_units = [100, 100]

notewise_state=[]
for i in range(len(num_n_units)):
    notewise_c=tf.placeholder(dtype=tf.float32, shape=[None, num_n_units[i]])
    notewise_h=tf.placeholder(dtype=tf.float32, shape=[None, num_n_units[i]])
    notewise_state.append(LSTMStateTuple(notewise_h, notewise_c))

notewise_state=tuple(notewise_state)

y_out, note_gen_out = LSTM_NoteWise_Layer(timewise_out, state_init=notewise_state,
output_keep_prob=output_keep_prob)

p_out = tf.sigmoid(y_out)
print('Форма y_out = ', y_out.get_shape())
print('Форма генерованих вибірок = ', note_gen_out.get_shape())

loss, log_likelihood = Loss_Function(Note_State_Batch, y_out)
optimizer = tf.train.AdadeltaOptimizer(learning_rate = 1).minimize(loss)
print('Граф побудовано')

start_time = time.time()
N_epochs = 50000
loss_hist=[]
loss_valid_hist=[]
restore_model_name = 'Long_Train'
save_model_name = 'Long_Train_256'
batch_size = 5
num_timesteps = 256
keep_prob=.5

Output_Directory = Working_Directory + "/Output/" + save_model_name
directory = os.path.dirname(Output_Directory)

try:
    print('Створення нової папки призначення')
    os.mkdir(directory)
```

```
except:
    print('Папка призначення вже існує')

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    saver = tf.train.Saver()

    if restore_model_name is not None:
        Load_Directory = Working_Directory + "/Output/" + restore_model_name

        print("Завантаження моделі з: {}".format(restore_model_name))
        saver.restore(sess, Load_Directory + '/{}'.format(restore_model_name))

    timewise_state_val=[]
    for i in range(len(num_t_units)):
        c_t = np.zeros((batch_size*num_notes, num_t_units[i])) #start every batch
with zero state in LSTM time cells
        h_t = np.zeros((batch_size*num_notes, num_t_units[i]))
        timewise_state_val.append(LSTMStateTuple(h_t, c_t))

    notewise_state_val=[]
    for i in range(len(num_n_units)):
        c_n = np.zeros((batch_size*num_timesteps, num_n_units[i])) #start every
batch with zero state in LSTM time cells
        h_n = np.zeros((batch_size*num_timesteps, num_n_units[i]))
        notewise_state_val.append(LSTMStateTuple(h_n, c_n))

    for epoch in range(N_epochs+1):

        if (epoch % 100 == 0):
            print('Отримання пакету даних')
            _, batch_input_state = Training.getPieceBatch(training_pieces,
batch_size, num_timesteps) # not using their 'convolution' filter
            batch_input_state = np.array(batch_input_state)
            batch_input_state = np.swapaxes(batch_input_state, axis1=1, axis2=2)

        feed_dict = {Note_State_Batch: batch_input_state, timewise_state:
timewise_state_val, notewise_state: notewise_state_val, time_init: 0,
output_keep_prob: keep_prob}
        loss_run, log_likelihood_run, _, note_gen_out_run = sess.run([loss,
log_likelihood, optimizer, note_gen_out], feed_dict=feed_dict)

        if (epoch % 1000 == 0) & (epoch > 0):
            save_path = saver.save(sess, Output_Directory +
'/{}'.format(save_model_name))
            print("Модель збережена в файлі: %s" % save_path)
            np.save(Output_Directory + "/ training_loss.npy", loss_hist)
            np.save(Output_Directory + "/ valid_loss.npy", loss_valid)

        if (epoch % 100) == 0 & (epoch > 0):
            _, batch_input_state_valid = Training.getPieceBatch(validation_pieces,
batch_size, num_timesteps)
            batch_input_state_valid = np.array(batch_input_state_valid)
            batch_input_state_valid = np.swapaxes(batch_input_state_valid, axis1=1,
axis2=2)
            feed_dict = {Note_State_Batch: batch_input_state_valid, timewise_state:
timewise_state_val, notewise_state: notewise_state_val, time_init: 0,
output_keep_prob: keep_prob}
            loss_valid, log_likelihood_valid = sess.run([loss, log_likelihood],
feed_dict=feed_dict)

            print('епоха = ', epoch, ' / ', N_epochs, ':')
            print('Втрата тренування = ', loss_run, '; Логарифмічна
правдоподібность тренування = ', log_likelihood_run)
```

```
            print('Втрата валідації = ', loss_valid, '; Логарифмічна
правдоподібность валідації = ', log_likelihood_valid)

            loss_hist.append(loss_run)
            loss_valid_hist.append(loss_valid)
end_time = time.time()

print('Час тренування = ', end_time - start_time, ' секунд')

plt.plot(loss_hist, label="Training Loss")
plt.plot(loss_valid_hist, label="Validation Loss")
plt.legend()
plt.show

keep_prob=1
training_loss_ave=[]
validation_loss_ave=[]

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    if restore_model_name is not None:
        print("Завантаження моделі з: {}".format(save_model_name))
        saver.restore(sess, Load_Directory + '/{}'.format(save_model_name))

    timewise_state_val=[]
    for i in range(len(num_t_units)):
        c_t = np.zeros((batch_size*num_notes, num_t_units[i]))
        h_t = np.zeros((batch_size*num_notes, num_t_units[i]))
        timewise_state_val.append(LSTMStateTuple(h_t, c_t))

    notewise_state_val=[]
    for i in range(len(num_n_units)):
        c_n = np.zeros((batch_size*num_timesteps, num_n_units[i]))
        h_n = np.zeros((batch_size*num_timesteps, num_n_units[i]))
        notewise_state_val.append(LSTMStateTuple(h_n, c_n))

    for p in range(10):
        _, batch_input_state_test = Training.getPieceBatch(training_pieces,
batch_size, num_timesteps)
        batch_input_state_test = np.array(batch_input_state_test)
        batch_input_state_test = np.swapaxes(batch_input_state_test, axis1=1,
axis2=2)

        feed_dict = {Note_State_Batch: batch_input_state_test, timewise_state:
timewise_state_val, notewise_state: notewise_state_val, time_init: 0,
output_keep_prob: 1}
        loss_run, log_likelihood_run, note_gen_out_run = sess.run([loss,
log_likelihood, note_gen_out], feed_dict=feed_dict)
        training_loss_ave.append(-78*loss_run)

        _, batch_input_state_valid = Training.getPieceBatch(validation_pieces,
batch_size, num_timesteps)
        batch_input_state_valid = np.array(batch_input_state_valid)
        batch_input_state_valid = np.swapaxes(batch_input_state_valid, axis1=1,
axis2=2)

        feed_dict = {Note_State_Batch: batch_input_state_valid, timewise_state:
timewise_state_val, notewise_state: notewise_state_val, time_init: 0,
output_keep_prob: 1}
        loss_run, log_likelihood_run, note_gen_out_run = sess.run([loss,
log_likelihood, note_gen_out], feed_dict=feed_dict)
        validation_loss_ave.append(-78*loss_run)
```

```
        print(p)

plt.plot(training_loss_ave, label="Training Log likelihood")
plt.plot(validation_loss_ave, label="Validation Log likelihood")
plt.legend()
plt.show

T_gen=64*16
batch_gen_size=10
keep_prob=.5
music_model_name = save_model_name

Load_Directory = Working_Directory + "/Output/" + music_model_name
directory = os.path.dirname(Load_Directory)


try:
    os.mkdir(directory)
except:
    print('Папка вже існує.')

notes_gen_initial = np.zeros((batch_gen_size, num_notes, 1,2))

notes_gen = notes_gen_initial

timewise_state_val=[]
for i in range(len(num_t_units)):
    c = np.zeros((batch_gen_size*num_notes, num_t_units[i]))
    h = np.zeros((batch_gen_size*num_notes, num_t_units[i]))
    timewise_state_val.append(LSTMStateTuple(h, c))

notewise_state_val=[]
for i in range(len(num_n_units)):
    c = np.zeros((batch_gen_size*1, num_n_units[i]))
    h = np.zeros((batch_gen_size*1, num_n_units[i]))
    notewise_state_val.append(LSTMStateTuple(h, c))

notes_gen_arr=[]

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    saver = tf.train.Saver()

    print("Завантаження моделі з: {}".format(music_model_name))
    saver.restore(sess, Load_Directory + '/{}'.format(music_model_name))

    for t in range(T_gen):
        feed_dict = {Note_State_Batch: notes_gen, timewise_state:
timewise_state_val, notewise_state: notewise_state_val, time_init: t % 16,
output_keep_prob: keep_prob}
        timewise_state_val, notes_gen = np.squeeze(sess.run([timewise_state_out,
note_gen_out], feed_dict = feed_dict), axis=2)
        notes_gen_arr.append(np.squeeze(notes_gen))

        if t % 50 == 0:
            print('Крок часу = ', t)

notes_gen_out = np.stack(notes_gen_arr, axis=2)
notes_gen_out = np.swapaxes(notes_gen_out, axis1=1, axis2=2)

for iter in range(batch_gen_size):
    file = Load_Directory + "/Epoch " + str(epoch) + " Music_Dropout" + str(iter)
    midi_out = midi_musical_matrix.noteStateMatrixToMidi(notes_gen_out[iter,:,:,:],
name=file)
```

```
print('Midi файл було збережено.')
```

Б.2 Програмний код модуля ModelGraph:

```
import tensorflow as tf
from tensorflow.python.ops import math_ops
from tensorflow.contrib.rnn import BasicLSTMCell
from tensorflow.contrib.rnn import DropoutWrapper

def Input_Kernel(input_data, Midi_low, Midi_high, time_init):

    batch_size = tf.shape(input_data)[0]
    num_notes = input_data.get_shape()[1].value
    num_timesteps = tf.shape(input_data)[2]

    Midi_indices = tf.squeeze(tf.range(start=Midi_low, limit = Midi_high+1,
delta=1))
    x_Midi = tf.ones((batch_size, num_timesteps, 1,
num_notes))*tf.cast(Midi_indices, dtype=tf.float32)
    x_Midi = tf.transpose(x_Midi, perm=[0,3,1,2]) # shape = batch_size, num_notes,
num_timesteps, 1

    Midi_pitchclasses = tf.squeeze(x_Midi % 12, axis=3)
    x_pitch_class = tf.one_hot(tf.cast(Midi_pitchclasses, dtype=tf.uint8),
depth=12)

    input_flatten = tf.transpose(input_data, perm=[0,2,1,3])
    input_flatten = tf.reshape(input_flatten, [batch_size*num_timesteps, num_notes,
2]) # channel for play and channel for articulate
    input_flatten_p = tf.slice(input_flatten, [0,0,0],size=[-1, -1, 1])
    input_flatten_a = tf.slice(input_flatten, [0,0,1],size=[-1, -1, 1])

    filt_vicinity = tf.expand_dims(tf.eye(25), axis=1)

    vicinity_p = tf.nn.conv1d(input_flatten_p, filt_vicinity, stride=1,
padding='SAME')
    vicinity_a = tf.nn.conv1d(input_flatten_a, filt_vicinity, stride=1,
padding='SAME')

    vicinity = tf.stack([vicinity_p, vicinity_a], axis=3)
    vicinity = tf.unstack(vicinity, axis=2)
    vicinity = tf.concat(vicinity, axis=2)

    x_vicinity = tf.reshape(vicinity, shape=[batch_size, num_timesteps, num_notes,
50])
    x_vicinity = tf.transpose(x_vicinity, perm=[0,2,1,3])

    input_flatten_p_bool = tf.minimum(input_flatten_p,1)

    filt_context = tf.expand_dims(tf.tile(tf.eye(12), multiples=[(num_notes //
12)*2,1]), axis=1)

    context = tf.nn.conv1d(input_flatten_p_bool, filt_context, stride=1,
padding='SAME')
    x_context = tf.reshape(context, shape=[batch_size, num_timesteps, num_notes,
12])
    x_context = tf.transpose(x_context, perm=[0,2,1,3])

    Time_indices = tf.range(time_init, num_timesteps + time_init)
```

```
    x_Time = tf.reshape(tf.tile(Time_indices, multiples=[batch_size*num_notes]),
shape=[batch_size, num_notes, num_timesteps,1])
    x_beat = tf.cast(tf.concat([x_Time%2, x_Time//2%2, x_Time//4%2, x_Time//8%2],
axis=-1), dtype=tf.float32)

    x_zero = tf.zeros([batch_size, num_notes, num_timesteps,1])

    Note_State_Expand = tf.concat([x_Midi, x_pitch_class, x_vicinity, x_context,
x_beat, x_zero], axis=-1)


    return Note_State_Expand

def LSTM_TimeWise_Training_Layer(input_data, state_init, output_keep_prob=1.0):

    batch_size = tf.shape(input_data)[0]
    num_notes = input_data.get_shape()[1].value
    num_timesteps = tf.shape(input_data)[2]
    input_size = input_data.get_shape()[3].value

    num_layers = len(state_init)

    input_flatten = tf.reshape(input_data, shape=[batch_size*num_notes,
num_timesteps, input_size])

    cell_list=[]
    num_states=[]
    for h in range(num_layers):
        num_states.append(state_init[h][0].get_shape()[1].value)
        lstm_cell = BasicLSTMCell(num_units=num_states[h], forget_bias=1.0,
state_is_tuple=True,activation=math_ops.tanh, reuse=None)
        lstm_cell = DropoutWrapper(lstm_cell, output_keep_prob=output_keep_prob)
        cell_list.append(lstm_cell)

    multi_lstm_cell = tf.contrib.rnn.MultiRNNCell(cell_list, state_is_tuple=True)

    output_flat, state_out = tf.nn.dynamic_rnn(cell=multi_lstm_cell,
inputs=input_flatten, initial_state=state_init, dtype=tf.float32)

    output = tf.reshape(output_flat, shape=[batch_size, num_notes, num_timesteps,
num_states[-1]])

    return output, state_out

def LSTM_NoteWise_Layer(input_data, state_init, output_keep_prob=1.0):

    batch_size = tf.shape(input_data)[0]
    num_notes = input_data.get_shape()[1].value
    num_timesteps = tf.shape(input_data)[2]
    input_size = input_data.get_shape()[3].value

    num_layers = len(state_init)

    notewise_in = tf.transpose(input_data, perm=[0,2,1,3])
    notewise_in = tf.reshape(notewise_in, shape=[batch_size*num_timesteps,
num_notes, input_size])

    cell_list=[]
    num_states=[]
    for h in range(num_layers):
        num_states.append(state_init[h][0].get_shape()[1].value)
        lstm_cell = BasicLSTMCell(num_units=num_states[h], forget_bias=1.0,
state_is_tuple=True,activation=math_ops.tanh, reuse=None)
        lstm_cell = DropoutWrapper(lstm_cell, output_keep_prob=output_keep_prob)
```

```
        cell_list.append(lstm_cell)

    multi_lstm_cell = tf.contrib.rnn.MultiRNNCell(cell_list, state_is_tuple=True)

    h_state = state_init
    p_gen_n= tf.zeros([batch_size*num_timesteps, 1])
    a_gen_n= tf.zeros([batch_size*num_timesteps, 1])

    y_list=[]
    note_gen_list=[]

    for n in range(num_notes):
        cell_inputs = tf.concat([notewise_in[:,n,:], tf.cast(p_gen_n, tf.float32),
tf.cast(a_gen_n, tf.float32)], axis=-1)

        h_final_out, h_state = multi_lstm_cell(inputs=cell_inputs, state=h_state)

        y_n = tf.layers.dense(inputs=h_final_out, units=2, activation=None)

        note_gen_n = tf.distributions.Bernoulli(logits=y_n).sample()

        p_gen_n = tf.slice(note_gen_n, [0,0], [-1,1])
        a_gen_n = tf.slice(note_gen_n, [0,1], [-1,1])
        a_gen_n = tf.multiply(p_gen_n, a_gen_n)
        note_gen_n = tf.concat([p_gen_n, a_gen_n], axis=1) # concatenate

        y_n_unflat = tf.reshape(y_n, shape=[batch_size, num_timesteps, 2])
        note_gen_n_unflat = tf.reshape(note_gen_n, shape=[batch_size,
num_timesteps, 2])

        y_list.append(y_n_unflat)
        note_gen_list.append(note_gen_n_unflat)

    y_out = tf.stack(y_list, axis=1)
    note_gen_out = tf.stack(note_gen_list, axis=1)

    return y_out, note_gen_out

def Loss_Function(Note_State_Batch, y_out):

    batch_size = tf.shape(y_out)[0]
    num_notes = y_out.get_shape()[1].value
    num_timesteps = tf.shape(y_out)[2]

    y_align = tf.slice(y_out, [0,0,0,0],[batch_size, num_notes, num_timesteps-1,
2])
    Note_State_Batch_align = tf.slice(Note_State_Batch, [0,0,1,0],[batch_size,
num_notes, num_timesteps-1, 2])

    cross_entropy = tf.nn.sigmoid_cross_entropy_with_logits(logits=y_align,
labels=Note_State_Batch_align)

    cross_entropy_p = cross_entropy[:,:,:,0]
    cross_entropy_a = cross_entropy[:,:,:,1] * Note_State_Batch_align[:,:,:,0]
    cross_entropy = tf.stack([cross_entropy_p, cross_entropy_a], axis=-1)

    Loss = tf.reduce_mean(cross_entropy) * 2

    Log_likelihood = -Loss*num_notes

    return Loss, Log_likelihood
```

Б.3 Програмний код модуля Training:

```
import os, random
from midi_musical_matrix import *
from data import *
import pickle
import numpy
import signal

division_len = 16

def loadPieces(dirpath, max_time_steps):
    pieces = {}

    for fname in os.listdir(dirpath):
        if fname[-4:] not in ('.mid','.MID'):
            continue

        name = fname[:-4]

        try:
            outMatrix = midiToNoteStateMatrix(os.path.join(dirpath, fname))
        except:
            print('Пропущено пошкоджений файл = ', name)
            outMatrix=[]

        if len(outMatrix) < max_time_steps:
            continue

        pieces[name] = outMatrix
        print ("Завантажено {}".format(name))
    return pieces

def getPieceSegment(pieces, num_time_steps):
    piece_output = random.choice(list(pieces.values()))
    start = random.randrange(0,len(piece_output)-num_time_steps,division_len)

    seg_out = piece_output[start:start+num_time_steps]
    seg_in = noteStateMatrixToInputForm(seg_out)

    return seg_in, seg_out

def getPieceBatch(pieces, batch_size, num_time_steps):
    i,o = zip(*[getPieceSegment(pieces, num_time_steps) for _ in
range(batch_size)])
    return np.array(i), np.array(o)

def trainPiece(model,pieces,epochs,start=0):
    stopflag = [False]
    def signal_handler(signame, sf):
        stopflag[0] = True
    old_handler = signal.signal(signal.SIGINT, signal_handler)
    for i in range(start,start+epochs):
        if stopflag[0]:
            break
        error = model.update_fun(*getPieceBatch(pieces))
        if i % 100 == 0:
            print ("епоха {}, помилка={}".format(i,error))
        if i % 500 == 0 or (i % 100 == 0 and i < 1000):
            xIpt, xOpt = map(numpy.array, getPieceSegment(pieces))
            noteStateMatrixToMidi(numpy.concatenate((numpy.expand_dims(xOpt[0],  0),
model.predict_fun(division_len, 1, xIpt[0])), axis=0),'output/sample{}'.format(i))
```

```
            pickle.dump(model.learned_config,open('output/params{}.p'.format(i),
'wb'))
    signal.signal(signal.SIGINT, old_handler)
```

Б.4 Програмний код модуля NoteStateMatrixConverter:

```
import midi
import numpy as np

lowerBound = 24
upperBound = 102
span = upperBound-lowerBound
num_rest = 0
num_play_hold = 1
num_play_artic = 2

def midiToNoteStateMatrix(midifile, squash=True):
    pattern = midi.read_midifile(midifile)

    timeleft = [track[0].tick for track in pattern]

    posns = [0 for track in pattern]

    statematrix = []
    time = 0

    state = [[0,0] for x in range(span)]
    statematrix.append(state)
    condition = True
    while condition:
        if time % (pattern.resolution / 4) == (pattern.resolution / 8):
            # Crossed a note boundary. Create a new state, defaulting to holding
notes
            oldstate = state
            state = [[oldstate[x][0],0] for x in range(span)]
            statematrix.append(state)
        for i in range(len(timeleft)): #For each track
            if not condition:
                break
            while timeleft[i] == 0:
                track = pattern[i]
                pos = posns[i]

                evt = track[pos]
                if isinstance(evt, midi.NoteEvent):
                    if (evt.pitch < lowerBound) or (evt.pitch >= upperBound):
                        pass
                        # print "Note {} at time {} out of bounds
(ignoring)".format(evt.pitch, time)
                    else:
                        if isinstance(evt, midi.NoteOffEvent) or evt.velocity == 0:
                            state[evt.pitch-lowerBound] = [0, 0]
                        else:
                            state[evt.pitch-lowerBound] = [1, 1]
                elif isinstance(evt, midi.TimeSignatureEvent):
                    if evt.numerator not in (2, 4):
                        # We don't want to worry about non-4 time signatures. Bail
early!
                        # print "Found time signature event {}.
Bailing!".format(evt)
                        out =  statematrix
```

```
                        condition = False
                        break
                try:
                    timeleft[i] = track[pos + 1].tick
                    posns[i] += 1
                except IndexError:
                    timeleft[i] = None

            if timeleft[i] is not None:
                timeleft[i] -= 1

        if all(t is None for t in timeleft):
            break

        time += 1

    S = np.array(statematrix)

    return statematrix

def noteStateMatrixToMidi(statematrix, name="example"):
    statematrix = np.array(statematrix)
    if not len(statematrix.shape) == 3:
        statematrix = np.dstack((statematrix[:, :span], statematrix[:, span:]))
    statematrix = np.asarray(statematrix)
    pattern = midi.Pattern()
    track = midi.Track()
    pattern.append(track)

    span = upperBound-lowerBound
    tickscale = 55

    lastcmdtime = 0
    prevstate = [[0,0] for x in range(span)]
    for time, state in enumerate(statematrix + [prevstate[:]]):
        offNotes = []
        onNotes = []
        for i in range(span):
            n = state[i]
            p = prevstate[i]
            if p[0] == 1:
                if n[0] == 0:
                    offNotes.append(i)
                elif n[1] == 1:
                    offNotes.append(i)
                    onNotes.append(i)
            elif n[0] == 1:
                onNotes.append(i)
        for note in offNotes:
            track.append(midi.NoteOffEvent(tick=(time-lastcmdtime)*tickscale,
pitch=note+lowerBound))
            lastcmdtime = time
        for note in onNotes:
            track.append(midi.NoteOnEvent(tick=(time-lastcmdtime)*tickscale,
velocity=40, pitch=note+lowerBound))
            lastcmdtime = time

        prevstate = state

    eot = midi.EndOfTrackEvent(tick=1)    track.append(eot)

    midi.write_midifile("{}.mid".format(name), pattern)
```

ДОДАТОК В

(обов'язковий)

КОПІЇ НАУКОВИХ ПУБЛІКАЦІЙ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Хмельницький національний університет

Військовий інститут Київського національного університету ім.Тараса Шевченка

ПВНЗ "Університет економіки і підприємництва"

Вінницький національний технічний університет

Західноукраїнський національний університет

Інтелектуальний потенціал - 2020

збірник наукових праць молодих науковців і студентів

сформовано за матеріалами
Всеукраїнської науково-практичної конференції
молодих науковців і студентів
«Інтелектуальний потенціал – 2020»

9-10 листопада 2020 р.

Частина 2

Хмельницький
2020

ЗМІСТ

графом, що при моделюванні великомасштабних мереж (більше 10 млн. вузлів) може дати похибку прогнозування загрози поширення забороненої інформації більше 30%. Крім того, дані підходи мають в основному теоретичний характер, практика їх використання не виходить за межі експериментів. Таким чином, дослідження, спрямовані на створення моделей та алгоритмів загрози поширення забороненої інформації, актуальні і мають теоретичне і практичне значення у вирішенні проблеми забезпечення інформаційної безпеки в системах і мережах телекомунікацій.

Прведене дослідження проблем інформаційної безпеки виявило, що крім проблем, пов'язаних з використанням глобальної мережі Інтернет як розподіленої інформаційно-телекомунікаційної системи, які досить добре відомі і можна вирішити, існує маловивчена проблема забороненого контенту, існуючі рішення малоефективні.

Ще однією важливою проблемою є великомасштабність ІТКМ, яка заважає отримати дані з імітаційної моделі за прийнятний час. Розв'язання цієї задачі полягає у створенні аналітичної моделі загрози поширення забороненої інформації в ІТКМ.

Перелік посилань

1. Биячуев, Т.А. Безопасность корпоративных сетей: учеб. пособие / Т.А. Биячуев; под ред. Осовецкого Л.Г. – СПб.: СПбГУ ИТМО, 2016.– 161 с.

2. Лукацький, А.В. Предотвращение сетевых атак: технологии и решения / А.В. Лукацкий. – СПб. : Экспрес Электроника, 2014. – 268 с.

3. Тропіна М. Дослідження соціальних мереж як нового феномену сучасного світу / М. Тропіна // Наукові записки Малої академії наук України. Серія «Педагогічні науки» : [зб. наук. праць ; редкол. : С.О. Довгий (голова), О.Є. Стрижак, О.В. Лісовий, І.М. Савченко та ін.]. — Київ : Національний центр «Мала академія наук України», 2019. — Вип. 16. – С. 57-63

Модель тренувального процесу та метод обробки музичних даних програмної системи генерування музичних творів із використанням штучного інтелекту

Бабич І.Р.
Науковий керівник – к.т.н., доцент Яшина О. М.
Хмельницький національний університет

В основі моделі тренувального процесу системи генерування музичного контенту лежить удосконалене розуміння музичної структури та чітке передбачення нотних даних із наданням можливості генерування поліфонічної музики (коли одна нота на один крок у часі[1]), що узгоджується із музичними правилами.

Для відображення гармонійної та мелодійної структури між нотами, модель використовує двошарову архітектуру водночас з мережею довгої короткочасної пам'яті та з рекурентною мережею, в якій існують періодичні зв’язки вздовж осі ноти.

Маючи одну мережу довгої короткочасної пам'яті на осі часу, а іншу – на осі ноти, модель використовує вираз з роботи Даніеля Джонсона: «двоосьову» конфігурацію [2].

Дотримуючись наукової публікації Мун [3] як рекомендований орієнтир, до кожного шару LSTM було застосовано випадання 0,75. Обраним оптимізатором було ADADELTA [4]. Вибрана швидкість навчання - 1,0. Всі компоненти моделі тренувального процесу будуть оцінюватись у двох вимірах.

На основі моделі Джонсона буде розроблена модель тренувального процесу.

Дана модель має на меті вивчання гармонійних та мелодійних ритмічних ймовірностей з тренувальних поліфонічних MIDI файлів.

За допомогою цієї моделі можна здійснити тренування та валідацію моделі в чисельній формі. Використання потім натренованої моделі необхідне для створення згенерованої музичної композиції у вигляді файлу MIDI.

Загальний процес, що буде відбуватись в моделі, є заснованим на методі Джонсона і є вдосконаленим методом обробки музичних даних.

Даний метод має на меті використання вектору даних у нейронній мережі даного проекту, що називається "матрицею стану ноти", яку показано на рисунку 1. Вона представляє стан «відтворення» та «артикуляції» кожної ноти в діапазоні значень Midi та для кожного часового кроку через визначений проміжок часу (тобто 8 мір з 16 кроками часу на такт).

Алгоритм методу обробки музичних даних при тренуванні моделі є таким:

– генерування з тренувальних MIDI файлів партію векторів даних характеристик, єдиний тензор 4D, який називається «Note_State_Batch»;

– генерування відповідного тензору ймовірності відтворення ноти на кроці часу;

– зміщення тензору logP на 1 крок у часі і обчислення функції втрат;

– використання функції оптимізатора для оновлення параметрів.

Виходячи за рамки виразів Даніеля Джонсона, вдосконалений метод обробки музичних даних має більш розвинену гнучкість введення даних, а також більшу загальність у встановлені та зміні різних гіперпраметрів, таких як кількість шарів, розмір прихованої одиниці, довжина послідовності, проміжки часу, розмір пакетів, метод оптимізації та швидкість навчання. Також даний метод є параметризованим, тому користувачі, на відміну від оригінального методу Джонсона, можуть також встановити довжину

партитури нот, поданих у LSTM нотних осей, і тривалість кроків часу, поданих у LSTM часових осей. Розмір партитури та тривалість часових кроків є важливими особливостями, оскільки музика сильно варіюється залежно від жанру та виконавця. Загалом, даний метод дозволить адаптувати її до конкретних потреб користувача. Також код, який буде написаний для даного методу, має мінімізувати використання циклів «for», щоб збільшити швидкість за обчислювальний час, збільшуючи швидкодію тренування нейронної мережі.

Оригінальні необроблені музичні дані у вигляді файлів .MIDI спочатку обробляються для генерації кожного Note_State_Batch за допомогою пакету Python-Midi[5]. У роботі було використано цей пакет лише для імпортування сегментів файлів MIDI як Note_State_Batches, а також для створення файлів MIDI із згенерованих зразків.

$$Note\ State\ Matrix = \begin{bmatrix} [p,a]_N^{(1)} & \cdots & [p,a]_N^{(T)} \\ & \vdots & \\ [p,a]_1^{(1)} & \cdots & [p,a]_1^{(T)} \end{bmatrix}$$

Рисунок 1 – Матриця стану ноти в режимі обробки

Дана матриця стану ноти - це оброблений вектор характеристик нейронної мережі, де N - номер ноти Midi, виділеної з музичної композиції, T - номер часових кроків в пакеті, p - вказує двійкове значення відтворюваної ноти, а «a» вказує двійкове значення відповідної артикуляції.

Загальна структура програмної реалізації розділена на дві основні задачі: тренування або валідація моделі чисельно, а потім використання натренованої моделі для створення нових файлів .MIDI для якісного оцінювання. Обидві функції використовують однаковий граф моделі в різних контекстах: тренувальний процес, показано на розробленій моделі у рисунку 2, ітеративно вводить у модель Note_State_Batch, запускає модель через усі відповідні кроки часу та ноти, присутні в пакетах, і потім виводить тензор відповідного логіту або зворотну сигмоподібну ймовірність того, що дана нота може відтворюватись на даному проміжку часу.

Розглянемо функціональність всіх компонентів моделі тренувального процесу.

Функція логарифмічної правдоподібності вхідних даних інтерпретується як здатність моделі приймати в якості вхідних даних вектор нот на даному кроці часу і прогнозувати набір нот на наступному кроці часу. Функція втрат, псевдокод якої показаний на рисунку 3, обчислює перехресну ентропію між згенерованими Logits та Note_State_Batch (після вибудовування Logits до елементів Note_State_Batch, що відповідають одному кроку часу в майбутньому).

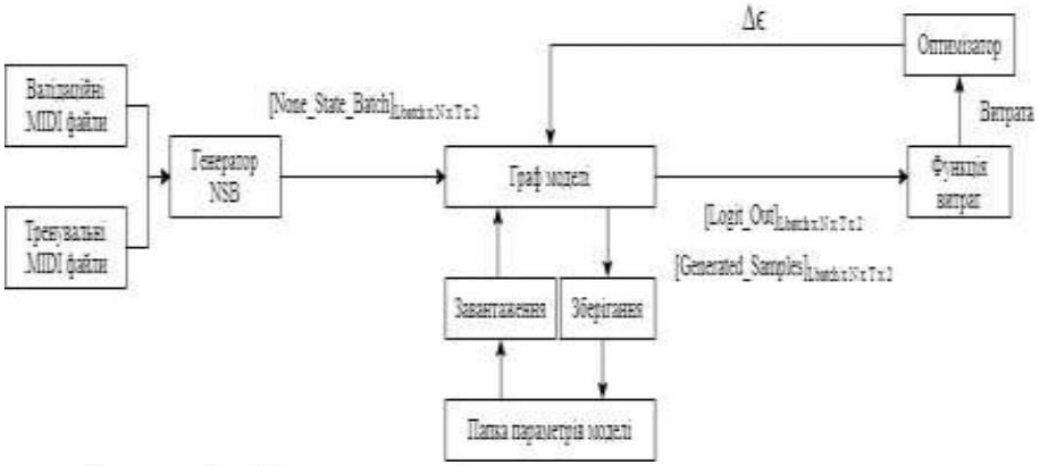


Рисунок 2 – Діаграма моделі тренувального процесу

Аргументи:

- Logits]$_{LbatchxNxTx2}$ (inverse sigmoid of Probability that play/articulate = 1;
- Labels(t-1) = [Note_State_Batch]$_{LbatchxNxTx2}$.

cross_entropy = sigmoid_cross_entropy_with_logits(logits=logits. labels=Labels) = - note_state ln(σ(logits)) + - (1 - note_state) In (1 - σ((logits)) = - note_state ln(Probability=1)+ -(1 - note_state) In (Probability=0)

$$Loss = \frac{1}{TNL\,L_{batch}} \sum_{b=1}^{L_{batch}} \sum_{t=1}^{T} \sum_{n=1}^{N} cross-entropy$$

$$\log-likelihood\ at\ 1\ time\ step = -\frac{1}{TL_{batch}} \sum_{b=1}^{L_{batch}} \sum_{t=1}^{T} \sum_{n=1}^{N} cross-entropy$$

$$\text{Log-likelihood at 1 time step} = -\frac{1}{TL_{batch}} \sum_{b=1}^{L_{batch}} \sum_{t=1}^{T} \sum_{n=1}^{N} cross-entropy$$

Повертає:

- Loss (scalar).

Рисунок 3 – Псевдокод для обчислення витрат

Під час задачі створення музики, представленої на рисунку 4, модель ітеративно виконується через один крок часу, кожен раз повертаючи генеровані вибірки(Generated_Samples) як вхідні дані Note_State_Batch для наступного кроку часу. Ці вибірки накопичуються, і це створює тензор генерованих вибірок у вигляді довільної довжини часу Note_State_Batchof. Потім згенеровані зразки перетворюються у файли .MIDI за допомогою функцій подальшої обробки з Python-Midi для якісної оцінки.

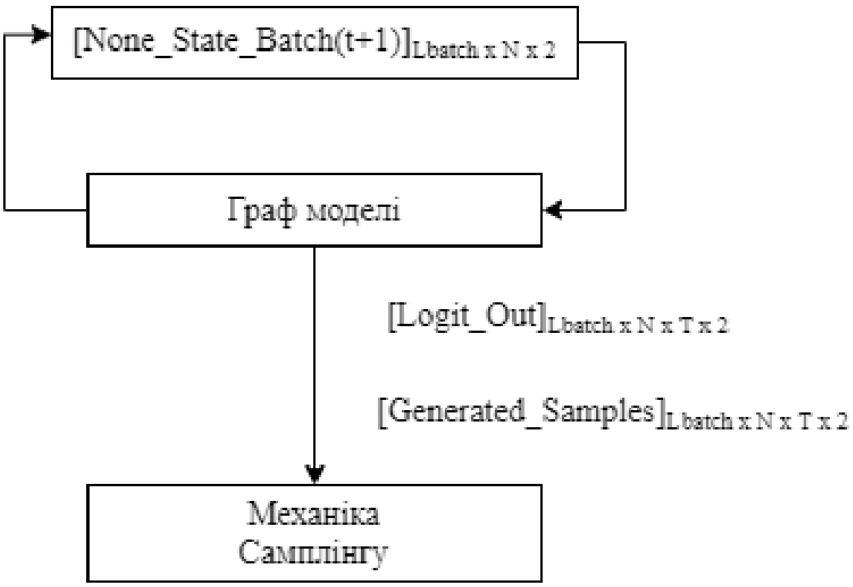


Рисунок 4 – Діаграма генерації музичної композиції

Функціональний розподіл самого графу моделі, показаного на рис. 5.

Було проведено декомпозицію вищого рівню графу моделі для того, щоб здійснити деталізацію кожного з компонентів генерації музичного контенту.

Компонент вхідного ядра приймає Note_State_Batch як свої вхідні дані і для кожної пари ноти й артикуляції генерує розширений вектор, який складається з:

- номера ноти Midi;
- одного гарячого вектора класу висоти ноти;
- вікна відтворення значень чи артикуляції відносно «n»-ої ноти (ефективний згорнутий аспект ядра моделі);
- вектор суми всіх відтворених нот у кожному класі висоти;
- бінарний вектор, що представляє позицію 16 ноти в межах міри.

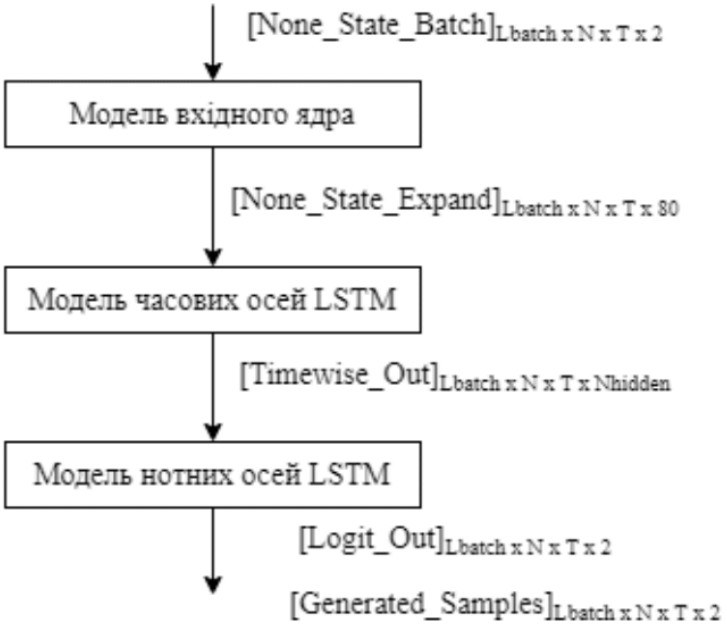


Рисунок 5 – Декомпозиція графу моделі

Псевдокод роботи вхідного ядра є показаним на рис. 6.

Наступний компонент декомпозиції графу ядра є компонентом обробки часових осей LSTM. У цьому компоненті комірка LSTM запускається вздовж часових осей на довжину виміру часового пакету. Ця операція виконується над вектором Note_State_Expand для кожної ноти паралельно з прив'язаними вагами. Ця частина графіку фіксує послідовні патерни музики і, у поєднанні з вхідним ядром, зберігає незмінність трансляції завдяки вікну введення відносних нот та прив'язаним вагам LSTM для всіх нот. Завдяки цим зв’язаним вагам, обчислення можуть виконуватися паралельно між нотами та між вибірками Матриці стану ноти як окремі ефективні пакети. Єдиний необхідний послідовний аспект - вздовж осі часу. Виконується довільна кількість каскадних комірок LSTM і після кожної комірки застосовується маска виключення.

Псевдокод для компоненту обробки часових осей LSTM зображений на рис. 7.

Заключним компонентом у декомпозиції, як описано у діаграмі, є компонент обробки нотних осей LSTM. Це потенційно один або багатошаровий компонент LSTM, такий як компонент часових осей LSTM, який включає в себе виключення після кожного шару. Однак, замість послідовного проходження по часової осі, цей етап проходить послідовно по нотній осі. Крім того, цей компонент включає в себе «локальний» зворотний зв'язок згенерованих вибірок до їхніх вхідних даних.

Аргументи:

– $[\text{None\_State\_Batch}]_{\text{LbatchxNxTx2}}$.

$$[p,a]_N^{(1)} = \text{'Kernel'} = \left[\begin{array}{l} [n]_{1x1} - \text{MIDI note №} \\ \begin{bmatrix} 0 \\ \vdots \\ [1] \\ \vdots \\ 0 \end{bmatrix}_{12x1} - \text{'nth' part} - \text{pitchclass} \\ \begin{bmatrix} p_{n-12}^{(t)} \\ a_{n-12}^{(t)} \\ \vdots \\ p_n^{(t)} \\ a_n^{(t)} \\ \vdots \\ p_{n+12}^{(t)} \\ a_{n+12}^{(t)} \end{bmatrix}_{50x1} - \text{part} - \text{prev} - \text{vicinity} \\ \begin{bmatrix} \text{pitchclass} - \text{count}_n^{(t)} \\ \vdots \\ \text{pitchclass} - \text{count}_{n+11}^{(t)} \end{bmatrix}_{12x1} - \text{part} - \text{context} \\ \begin{bmatrix} LSB \\ \vdots \\ MSB \end{bmatrix}_{4x1} - \text{beat} \\ [0] - \text{zero ?} \end{array}\right]_{80x1}$$

Повертає:

– $[\text{None\_State\_Expand}]_{\text{LbatchxNxTx80}}$.

Рисунок 6 – Псевдо-код для вхідного ядра

Аргументи:

– $[\text{None\_State\_Expand}]L_{batch}xNxTxL_{size}$. Where $L_{size} = 80$

$t = 1{:}T$

$h_{(1)n}^{(t)} = \text{LSTM}(h_{(1)n}^{(t-1)}, \text{None\_State\_Expand}_n^{(t)})$

$h_{(2)n}^{(t)} = \text{LSTM}(h_{(2)n}^{(t-1)}, h_{(1)n}^{(t)})$

⋮

$\text{timewise\_out} = h_{(\text{num-t-layer})n}^{(t)} = \text{LSTM}(h_{(\text{num-t-layer})n}^{(t-1)}, h_{(\text{num-t-layer})n}^{(t)})$

Повертає:

– $[\text{timewise\_out}]_{L_{batch}xNxTxN_{num-t-layer}}$.

Рисунок 7 – Псевдо-код для часових осей LSTM

Після кожного кроку ноти комірка LSTM виробляє пару логітів, що представляють зворотну сигмоїду ймовірності генерування відтворення або артикуляції для цієї ноти. Далі, з розподілу Бернуллі витягується зразок гри та артикуляції. Якщо вибірка відтворення має значення «0», що означає «не відтворено», артикуляційна вибірка також примусово доходить до «0», щоб уникнути генерації будь-яких значень, відсутніх у вхідних даних.

Згенерована дискретизована пара в ноті (n-1), об'єднана з вхідними даними часового LSTM на ноті (n), повертається назад на вхід нотатурного LSTM для кроку (n). Цей зворотний зв'язок створює умовну ймовірність для кожної ноти на основі фактичних значень, створених для нижчих нот. Це допомагає запобігти відтворенню дисонансних одночасних нот. Остаточними вихідними тензорами графу моделі є пакет Logits та відповідні згенеровані вибірки, які будуть використовуються для тренування та генерації музики, відповідно.

Псевдокод для компоненту обробки нотних осей LSTM зображений на рис. 8.

Аргументи:

– $[\text{timewise\_out}]_{L_{batch} \times N \times T \times N_{num\text{-}t\text{-}layer}}$.

For n = 1 to N:

$\text{cell\_input}_n = \begin{bmatrix} timewise-out_n \\ note-gen_{n-1} \end{bmatrix}$

$h_{(1)n} = \text{LSTM}(h_{(1)(n-1)}, \text{cell\_input}_n)$

$h_{(2)n} = \text{LSTM}(h_{(2)(n-1)}, h_{(1)n})$

⋮

$h_{(num\text{-}t\text{-}layer)n} = \text{LSTM}(h_{(num\text{-}t\text{-}layer)(n-1)}, h_{(num\text{-}t\text{-}layer-1)n})$

$\text{Logits}_n = Wh_{(num\text{-}t\text{-}layer)n} + b = \begin{bmatrix} Logit_n - play \\ Logit_n - artic \end{bmatrix}$

$\text{note\_gen}_n = \text{Sample}[\sigma(\text{Logits}_n) = \text{Prob}(\text{note\_n}=1)] = \begin{bmatrix} play-gen_n \\ artic-gen_n \end{bmatrix}$

if $(\text{play\_gen}_n = 0)$ then $\text{artic\_gen}_n = 0$

Повертає:

– $[\text{Logits}]_{L_{batch} \times N \times T \times 2}$.

– $[\text{note\_gen}]_{L_{batch} \times N \times T \times 2}$.

Рисунок 8 – Псевдо-код для нотних осей LSTM

Отже, на основі моделі Джонсона була розроблена модель

тренувального процесу та вдосконалений метод обробки музичних даних.

Дана модель має на меті вивчання гармонійних та мелодійних ритмічних ймовірностей з тренувальних поліфонічних MIDI файлів.

За допомогою цієї моделі можна здійснити тренування та валідацію моделі в чисельній формі. Використання потім натренованої моделі необхідне для створення згенерованої музичної композиції у вигляді файлу MIDI.

У вдосконаленого метода обробки музичних даних, на відміну від оригінального методу Деніела Джонсона, є розвинута гнучкість при введенні даних користувачем, загальність у встановлюванні різних гіперпараметрів, а також наявна параметризація, що дозволяє встановити довжину партитури нот та тривалість кроків часу необхідної для відповідної часової та нотної осі. Дані особливості збільшують можливість інтерактивної взаємодії живої людини з нейронною мережею, дозволяючи розробляти більш оригінальні мелодії.

Розроблюваний псевдокод з мінімальним використанням циклів дає також більшу швидкодію роботи програмного забезпечення для тренування нейронної мережі.

Перелік посилань

1. arXiv.org [Електронний ресурс] : [архів з відкритим доступом]. – Електронні дані (1 789 904 записів). – США : Корнельський університет, 2020. – Режим доступу: https://arxiv.org/pdf/1709.01620.pdf (дата звернення 07.08.2019). – Назва з екрана.

2. Джонсон Д. Д. Генерування поліфонічної музики за допомогою пов'язаних паралельних мереж / Д. Д. Джонсон // Обчислювальний інтелект у музиці, звуці, мистецтві та дизайні : зб. наук. Праць / 6-а Міжнародна конференція з еволюційних обчислень у комбінаторній оптимізації – Амстердам, 2017 – № 9 – С. 128 – 143

3. Мун Т, Чой Х, Лі Х, Сонг І. RnnDrop: новий випадок для RNN в ASR [Електронний ресурс] / Інститут інженерів електротехніки та електроніки – Електрон. дані. – Скоттсдейл, Арізона, США, 2015. – Режим доступу: https://ieeexplore.ieee.org/abstract/document/7404775 (дата звернення 11.02.2016). – Назва з екрана.

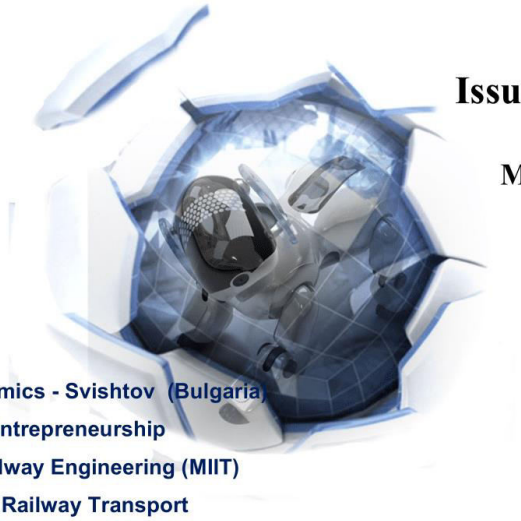
4. arXiv.org [Електронний ресурс] : [архів з відкритим доступом]. – Електронні дані (1 789 904 записів). – США : Корнельський університет, 2020. – Режим доступу: https://arxiv.org/pdf/1212.5701.pdf (дата звернення 22.12.2012). – Назва з екрана.

5. Github [Електронний ресурс] : [Інтернет-портал]. – Електронні дані. – [Сан Франциско, 2020]. – Режим доступу: https://github.com /vishnubob/python-midi (дата звернення 08.06.2015). – Назва з екрана.

International periodic scientific journal
ONLINE
www.modscires.pro

Indexed in
INDEXCOPERNICUS
(ICV: 86.17)

Modern Scientific Researches



Issue №12
Part 1
May 2020

With the support of:

- D.A.Tsenov Academy of Economics - Svishtov (Bulgaria)
- Institute of Sea Economy and Entrepreneurship
- Moscow State University of Railway Engineering (MIIT)
- Ukrainian National Academy of Railway Transport
- State Research and Development Institute of the Merchant Marine of Ukraine (UkrNIIMF)
- Lugansk State Medical University
- Kharkiv Medical Academy of Postgraduate Education
- Alecu Russo State University of Bălţi
- GUUPO "Belarusian-Russian University"
- Institute of Water Problems and Land Reclamation of the National Academy of Agrarian Sciences
- Odessa Research Institute of Communications

Published by:
Yolnat PE, Minsk, Belarus

ISSN 2523-4692
DOI: 10.30889/2523-4692

UDC 08
LBC 94

The International Scientific Periodical Journal "***Modern Scientific Researches***" has been published since 2017 and has gained considerable recognition among domestic and foreign researchers and scholars.

Periodicity of publication: Quarterly

The journal activity is driven by the following objectives:

- Broadcasting young researchers and scholars outcomes to wide scientific audience
- Fostering knowledge exchange in scientific community
- Promotion of the unification in scientific approach
- Creation of basis for innovation and new scientific approaches as well as discoveries in unknown domains

The journal purposefully acquaints the reader with the original research of authors in various fields of science, the best examples of scientific journalism.

Publications of the journal are intended for a wide readership - all those who love science. The materials published in the journal reflect current problems and affect the interests of the entire public.

UDC 08
LBC 94
DOI: 10.30889/2523-4692.2020-12-01

Published by:
Yolnat PE,
Minsk, Belarus
e-mail: editor@modscires.pro

СОДЕРЖАНИЕ / CONTENTS

https://www.modscires.pro/index.php/msr/article/view/msr12-01-061
DOI: 10.30889/2523-4692.2020-12-01-061

УДК 004.2

APPROACHES TO ARTIFICIAL INTELLIGENCE DEVELOPMENT AND THEIR INFLUENCE ON MUSIC WRITING AUTOMATION

ПІДХОДИ ДО РОЗРОБКИ ШТУЧНОГО ІНТЕЛЕКТУ ТА ЇХНІЙ ВПЛИВ НА АВТОМАТИЗАЦІЮ НАПИСАННЯ МУЗИКИ

Babich I.R. / Бабич І.Р.
студент
Yashyna O.M. / Яшина О.М.
к.т.н. доцент кафедри інженерії програмного забезпечення
Хмельницький національний університет

Анотація. *В роботі розглядається автоматизоване вдосконалення штучного інтелекту (ШІ) генерування музичних композицій в контексті його розвитку та аналізуються його можливі оптимізації в майбутньому.*

Під час огляду всіх можливих варіантів застосування ШІ в музиці в різні періоди часу, вирішувалося питання про те, як ШІ змінили для написання гармонійної музики і чи є можливі методи, щоб покращити його можливості для композиції музики до людського рівня творчості.

При аналізі було проведено огляд розвитку ШІ і різних підходів до створення генераторів музики, щоб краще зрозуміти труднощі в розробці будь-якої системи ШІ. Провівши аналоги із розглянутими підходами, було виявлено, що розробники використовували чотири підходи в створенні систем ШІ.

Незважаючи на успіх продемонстрованих технологій, стан самого оптимізування ШІ для використання в написанні музики, як і самого оціночного поняття про те, що таке «хороша музика» все ще залишається на концептуальному рівні.

Ключові слова: *штучний інтелект (ШІ), алгоритм, музика, система.*

Вступ.

На сьогоднішній день штучний інтелект дає можливість розв’язувати велике коло різних задач за допомогою автоматизації процесу, від розпізнавання об'єктів на зображенні до виконання більш складних функцій, тому його застосовують у різних сферах життєдіяльності людини, зокрема мистецтві, написанні музичних творів тощо.

Гарним прикладом програмного забезпечення, що дозволяє здійснювати конвертацію зображення у музичний файл є програма i2sm [1]. Цей додаток пропонує унікальний підхід до створення музики шляхом створення звукового файлу MIDI на основі будь-якого зображення, яке користувач завантажує в програму. Програма аналізує зображення пікселя за пікселем, перетворюючи їх яскравість, колір та насиченість у музичні ноти.

Під час експериментування над програмою виникло таке проблематичне питання: як використовували штучний інтелект при автономному створенні музики і чи є можливість покращити цю область у майбутньому.

Було досліджено та зібрано інформацію щодо формування штучного інтелекту як генератора музики протягом часу, а також сформовано поняття про представлення штучного інтелекту в цілому.

Мета цієї статті полягає у дослідженні використання засобів та підходів використання штучного інтелекту в музиці, тим самим надаючи інформацію

про те, якими методами можна підійти до розробки алгоритмізації створення музики та як можна представляти сам штучний інтелект у формальному понятті.

Основний текст

Щоб правильно зрозуміти музику, згенеровану штучним інтелектом, потрібно спочатку переглянути історію та розвиток штучного інтелекту загалом, щоб краще зрозуміти прогрес, який досяг штучний інтелект та які бар'єри існують у галузі музики.

Тлумачення поняття «штучне» подається як щось таке, що не зустрічається в природі і є результатом людських зусиль [2]. Перший крок у розробці штучного інтелекту – це зрозуміти, як люди думають інтелектуально. Ці дослідження впродовж тисячоліть були зусиллями багатьох людей [3]. Інтелект – це такий процес, що мозок використовує для зберігання всіх вхідних даних, які він отримує через п’ять органів чуття, а потім перекладає та застосовує ці входи для створення інструментів, мистецтва, літератури, прогнози та сприйняття, що змінюють світ, в якому він працює. Мета штучного інтелекту – це не тільки зрозуміти процес мозку, а й створити механізми, що копіюють усі інтелектуальні процеси, такі як творчість, прогнозування та всі інші.

В галузі інформаційних технологій використовуються чотири підходи при поясненні штучного інтелекту. Підходи – Тьюрінг (діючи по-людськи), Когнітивний (мислити по-людськи), Логічний (мислити раціонально) та Агент (діяти раціонально).

– Підхід «діючи по-людськи», визначений Аланом Тьюрінгом, називається загальним тестом Тьюрінга. Інтелектуальну поведінку Тьюрінг визначив як здатність досягати виконання на людському рівні всіх пізнавальних завдань, достатніх, щоб обдурити тестуючого. Іншими словами, тест, який він запропонував, полягає в тому, що комп'ютер повинен опитуватися людиною через телетайп. Якщо тестувальник не може конкретно стверджувати комп'ютер це чи людина, то комп'ютер можна вважати інтелектуальним [3].
– Підхід «мислити по-людськи” ще називають підходом когнітивного моделювання. Він фокусується на розумінні того, як працює мозок, а потім штучно копіює його для отримання подібних результатів [3].
– Підхід «мислити раціонально» розпочався з грецького філософа Аристотеля котрий спробував визначити «правильне мислення» як незаперечні процеси мислення. Його знамениті силогізми демонструють приклади структур аргументів, які завжди роблять правильні висновки при коректно заданих передумовах. Наприклад, «Сократ – людина; всі люди смертні: отже Сократ смертний». Будь-яка система, що ґрунтується на логіці та/або правилах, підпадає під цю категорію. У цього підходу є дві проблеми. Перша проблема виникає, коли інформація або неповна, або не може бути відформатована в логічній нотації. Друга проблема виникає, коли розмір інформації / бази даних перевищує ємність системи [3].
– Підхід «діяти раціонально» розглядає штучний інтелект як створення і

дослідження раціональних агентів, під яким мається на увазі програма або робот, які сприймають інформацію і виконують певні дії. Діяти раціонально передбачає оптимальне досягнення заданої мети при наявних або заданих обмеженнях [3].

Із розумінням концепту штучного інтелекту в цілому, можна дослідити процес його розвитку та застосування в області музики, а також проаналізувати, як розробники в різні періоди часу використовували відомі підходи до своєї проблеми генерування музики.

Переглядаючи різні теорії та програми, було оцінено їхню корисність з точки зору чотирьох підходів, визначених вище.

Знаючи, який підхід використовується до кожного методу, визначено бар'єри, які запропонований метод повинен був подолати, щоб розробити свою систему генерування музики за допомогою штучного інтелекту.

У таблиці 1 подано стислий графік подій [4].

Таблиця 1

Рік	Автор	Метод	Підхід
1956	Брукс, Хопкінс, Нейман, Райт	Перша музична система ШІ, що використовує теорію аналізу-синтезу	Агент
1958	Гіллер, Ісаксон	Монте-Карло з правилами контрапункту	Логічний
1963	Гілл	Деревообробний процес	Логічний
1966	Форте	Представлено SNOBOL3 - розпізнавання шаблонів	Агент
1968	Саймон, Саммер	Теорія психології про розпізнавання образів	Конгінітивний
1968	Виноград	Мовна теорія	Конгінітивний
1972	Мурер	Ритм як універсальна теорія	Конгінітивний
1974	Сомліар	Психологічна теорія про те, чому люди люблять музику	Конгінітивний
1974	Ласке	Теорія компонентів компетентності	Конгінітивний
1974	Радер	Евристична теорія	Логічний
1976	Сунберг, Ліндблум	Мовна теорія з правилами	Конгінітивний
1980	Міхан	Теорія природної мови	Конгінітивний
1981	Мінскі	Теорія слухача	Конгінітивний
1986	Шванауер	Система розуміння музики (MUSE)	Логічний
1987	Бхаруча	Коннекціоністська теорія	Конгінітивний
1990	Коуп	Експерименти з музичним інтелектом	Конгінітивний
1994	Лонгет-Хіггінс	Як люди ототожнюють теорію мелодії	Конгінітивний

Висновки.

Отже, на основі здійсненого дослідження можна зробити наступні висновки. Після аналізу різних підходів стає зрозумілим, що сама область музики, згенерованої штучним інтелектом, виросла із суворого застосування вивчених ймовірностей та правил до надання рекомендацій щодо наступної ноти, оскільки композитор створює композицію у своєму власному стилі.

Ранні спроби були зосереджені на раціональному агентурно-логічному підході, оскільки ніхто не розробив теорію про те, як мозок мав обробляти інформацію. Як бачимо із таблиці 1 на початку 90-х років 20-го століття був розроблений метод, що полягав у ототожненні людьми теорії мелодій. З вивченням психологами, неврологами та біхевіористами того, як люди слухають, реагують на музичні звуки та створюють композиції, були розроблені деякі фундаментальні концепції для використання когнітивного підходу.

Також потрібно відмітити, що жоден із підходів до штучного інтелекту не дає однієї однозначної відповіді про рівень якості музичного твору. На певному етапі розробки повинно бути визначено, що створена хороша музика, залежно від культурних та соціальних норм як композитора, так і аудиторії. Слухаючи композицію, кожен може по-своєму визначити, що таке добре, що є середнім чи що є просто шумом.

З часом із збільшенням кількості музичних творів змінюються стандарти та їх класифікації. Сучасне розуміння функцій мозку схоже на схеми в комп'ютері. Схеми відкриваються і закриваються, щоб переглянути минулий досвід і вжити заходів для створення музики. Загалом же, процес, за допомогою якого мозок здійснює створення музики, невідомий. У цьому полягає бар'єр, який повинен подолати той, хто працює зі штучними ін телектами для створення музики.

Література:

1. Retrieved from https://i2sm.en.softonic.com/

2. Artificial, Merriam-Webster, website, accessed January 2, 2019, https://www.merriamwebster.com/dictionary/artificial.

3. Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, Third Edition (London, Pearson Education, Inc., 2010), 17.

4. The Status Of Ai In Music, 24.

Abstract. *This work shows the history of improving the artificial language for generating musical compositions to its current state and analyzes its possible optimizations in the future.*

During the examination of all possible options for using AI in music at different periods of time, the question was put on about how AI was modernized to write harmonic music and are there any possible methods to improve its ability to compose music to the human level of creativity.

When analyzing the robot conferences, a review was conducted of the development of AI and various approaches to creating their systems in order to better understand the difficulties in developing any AI system. I found that the developers used four approaches in the creation of AI systems and conducted their analogues with the approaches.

Despite the success of the demonstrated technologies, the situation about the most optimized

AI for use in composing music, as well as the most evaluative concept about what "good music" is, remains at this stage of the concept.

Key words: *AI, algorithm, music, system.*

Науковий керівник: к.т.н. Яшина О. М.

Стаття відправлена: 02.06.2020 р.

ДОДАТОК Г

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Технологія розробки програмної системи для генерування музичних творів із використанням штучного інтелекту

Автор роботи:
ст. гр. ІПЗм-19-1 Бабич І. Р.
Керівник роботи:
к. т. н., доцент Яшина О. М.

Мета: створення моделі та удосконалення методу врахування введення даних користувача для генерування музичних творів із використанням штучного інтелекту.

Об’єкт дослідження: програмні системи генерування музичних творів із використанням штучного інтелекту.

Предмет дослідження: моделі та методи створення та вдосконаленння систем генерування музичних творів із використанням штучного інтелекту.

Завдання

- здійснити аналіз існуючих підходів генерування музичних творів;
- розробити модель вивчення штучного інтелекту теорії музики та аналізу музичних творів;
- вдосконалити метод обробки музичних треків шляхом включення введення більше користувацьких даних;
- розробити програмну реалізацію системи генерування музичних творів із використанням штучного інтелекту;
- провести тестування отриманих результатів.

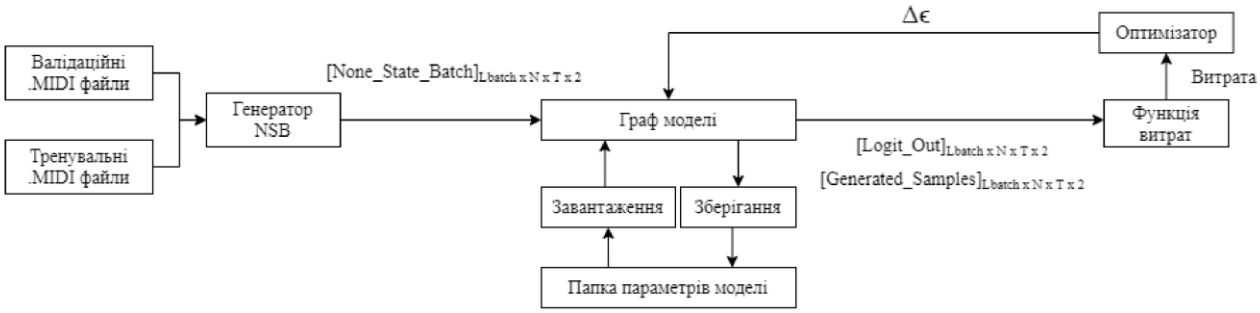
Актуальність теми

Все більшою сферою застосування методів глибокого навчання є генерація контенту. Вміст може бути різного роду: зображення, текст та музика, що є основою нашого аналізу.
Більшість з розглянутих існуючих методів генерування музичних творів не мають достатньої інтерактивності.
Розробка даної роботи також актуальна в плані підтримки музикантів (для композиції, аналізу, гармонізації, аранжування, продюсування, міксінгу, тощо) та може підвищити ефективність створення музичних творів із співпрацею з самими музикантом.
Таким чином зявляється актуальність даного виконання проекту, яке підтверджується необхідностю розроблення або вдосконалення метода обробки згенерованих музичних композицій користувачем та потребою розвитку в даній предметній області.

Аналіз стану проблеми і інших рішень

Переважна більшість проаналізованих систем показала, що здатна вирішити низку проблем, пов’язаних з генеруванням музичних творів, але досі існують виключення, які були недостатньо розглянуті науковцями та інженерами для створення приємної на прослуховування мелодії: оригінальність музичного твору та інтерактивність користувача, що можливо було помітити в згенерованих зразках від проаналізованих систем.

Модель роботи системи



Матриця стану ноти

$$Note\ State\ Matrix = \begin{bmatrix} [p,a]_N^{(1)} & \cdots & [p,a]_N^{(T)} \\ & \vdots & \\ [p,a]_1^{(1)} & \cdots & [p,a]_1^{(T)} \end{bmatrix}$$

Демонстрація згенерованих файлів

Наукова новизна

– розроблено модель створення музики на основі вивчення штучного інтелекту теорії музики та аналізу сучасних музичних творів;

– вдосконалено метод обробки музичних треків із врахуванням запиту користувача.

НАУЧНО-ОБРАЗОВАТЕЛЬНЫЙ ИНТЕРНЕТ ПРОЕКТ
SCIENTIFIC WORLD "SWORLD"
www.sworld.education

В сотрудничестве с:

PROJOURNAL ISSNpro
SCIENTWORLD DOIcenter

Crossref Google

Совместно с:
Институт МиП

При научной поддержке:
ВУЗов и Научно-исследовательских институтов:
Экономическая академия им.Д.А.Ценова-Свиштов (Болгария)
Одесский национальный морской университет
Московский государственный университет путей сообщения (МИИТ)
Украинская государственная академия железнодорожного транспорта
Научно-исследовательский проектно-конструкторский институт морского флота
Луганский государственный медицинский университет
Харьковская медицинская академия последипломного образования
Белыший Государственный Университет «Алеку Руссо»
Институт водных проблем и мелиорации Национальной академии аграрных наук
Одесский научно-исследовательский институт связи

Исх. ВЕ12-061 от 05.06.2020 г.

СПРАВКА

Авторы:
Babich Ivan Ruslanovich
Khmelnitsky National University

Научная работа:
"APPROACHES TO THE DEVELOPMENT OF ARTIFICIAL INTELLIGENCE AND THEIR IMPACT ON THE HISTORY OF AUTOMATION OF MUSIC WRITING"

Одобрена Редколлегией для публикации в журнале:
«**Modern scientific researches**» Вып.12.
Выход журнала запланирован на конец июня.

Номер заявки:
No.be12-061.

Председатель Редколлегии
д.т.н., проф., академик А.Г.Шибаев

Научный Секретарь
Редколлегии,
Руководитель Проекта SWorld С. В. Куприенко

Висновки

- на основі огляду літератури та досліджень було проаналізовано процес генерування музики штучним інтелектом;
- на основі існуючих методів та систем були проаналізовані існуючі рішення в предметній області;
- подано опис до розробленої моделі тренувального процесу;
- подано опис до вдосконаленого методу обробки музичних даних;
- було здійснено програмну реалізацію дипломного проекту;
- сформульовано наукову новизну дипломного проекту.

Дякую за увагу, доповідь закінчено

Mon Dec 07 08:46:21 EET 2020, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 14%**

	Документ		Сумарний збіг по Базі Даних	
ID: 82588 Назва: Технологія розробки програмної системи для генерування музичних творів із використанням штучного інтелекту Додано в БД: 2020-12-07 Автора: І.Р. Бабич Керівники: О. М. Яшина Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	129855	1005	2979 (2%)	34 (3%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНУ РОБОТУ

Дипломник студент групи ІПЗм-19-1 Бабич І.Р.

Тема Технологія розробки програмної системи для генерування музичних творів із використанням штучного інтелекту

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломної роботи:

Кількість листів креслень 0; кількість сторінок записки 87

1.Короткий зміст ДР та прийнятих рішень Представлена робота присвячена актуальній темі в області генерування музичних творів штучним інтелектом і складається з наступних розділів: вступ, аналіз відомих моделей, методів та засобів, моделі та методи для вирішення задачі, проектування програмного забезпечення для вирішення проблеми, реалізація програмного забезпечення для вирішення проблеми, висновки, додатки.

2. Висновок про відповідність ДР поставленому завданню Магістерська кваліфікаційна робота виконана у відповідності з завданням із дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі студент провів детальний аналіз предметної області, дослідив процеси генерування музичних творів з різних систем, представив та описав особливості існуючих рішень в розглянутій сфері, на основі яких довів актуальність роботи і визначив вимоги для створюваної системи. В другому розділі на основі досліджених джерел було розроблено власну модель для створення музики на основі вивчення штучного інтелекту теорії музики та аналізу музичних творів та обгрунтовано як саме дана модель вирішує проблему, а також вдосконалено новий метод існуючої системи для обробки музичних треків із врахуванням введення даних користувача і також обгрунтовано як введені зміни оптимізують даний метод. Відповідно в третьому і четвертому розділах автором проведено проектування системи, здійснено опис програмної реалізації і протестовано розроблений продукт.

4. Позитивні сторони роботи До позитивних сторін роботи слід віднести актуальність даного направлення дослідження, деталізацію аналізу усіх розглянутих стратегій вирішення проблеми та поглиблене опрацювання всіх аспектів реалізації з практичним використанням запропонованого рішення.

5. Негативні сторони роботи До негативних сторін роботи слід віднести недоліки по оформленню представленого матеріалу, що були виправлені.

6. Оцінка графічного оформлення та пояснювальної записки роботи Дані матеріали роботи є структурованими у чіткій та логічній формі та відображають послідовність виконання поставлених завдань. І хоча й в них було знайдено декілька стилістичних та орфографічних помилок, вони були пізніше усунені. Тому дане виконання пояснювальної записки та графічного оформлення заслуговує оцінки «добре».

7. Відгук про роботу в цілому Загалом, зміст представленої роботи в повній мірі розкриває обрану тему. Дослідження, проведені в матеріалах є достатньо аргументованими. Прослідковуються високі теоретичні та практичні рівні у даному виконанні. Результатом проведення досліджень стали відповідні висновки і конкретні пропозиції щодо вдосконалення процесу генерування музичних творів із використанням штучного інтелекту.

8. Інші зауваження

9. Оцінка дипломної роботи Робота заслуговує оцінки «добре», а її автор – присвоєння кваліфікації «магістра» з інженерії програмного забезпечення.

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) Лисенко Сергій Миколайович, доктор технічних наук, доцент кафедри комп'ютерної інженерії та системного програмування ХНУ

"2" грудня 2020 р. (підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ ПО КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: **Технологія розробки програмної системи для генерування музичних творів**

Автор: **Бабич Іван Русланович**

Спеціальність: **121 «Інженерія програмного забезпечення»**

Освітня програма: **«Інженерія програмного забезпечення»**

Науковий керівник: **к.т.н., доцент Яшина Оксана Миколаївна**

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	–
3	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	–
4	Інше:	–

Підтвердження: Виявлені запозичення не є плагіатом, так як розміщені в розділах, що не описують безпосередньо авторське дослідження, складають 6,02 % та мають посилання на приведений список літературних джерел, стандартні конструкції коду.

7.12.2020 р.
Дата

Підпис керівника

Підпис завідувача кафедри

Гарант ОП