МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Факультет математики і інформатики

Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

бакалавр

на тему «Побудова нейронної мережі для генерації музичних зразків із частотою ударів на хвилину із заданого інтервалу на основі колекції музичних зразків»

Виконав: студент 4 курсу, групи МФ-42

спеціальність 122 «Комп'ютерні науки»

освітньо-професійна програма «Інформатика»

Звагольський О. В.

(прізвище та ініціали)

Керівник Волков І. В.

(прізвище та ініціали)

Рецензент_____

(прізвище та ініціали)

Харків – 2023 року

ЗМІСТ

ВСТУП

Актуальність теми. Нейронні мережі набирають все більшої популярності і з кожним днем збільшується кількість компаній, які розробляють свій власний штучний інтелект для різних цілей. У зв'язку зі своїм величезним потенціалом і широтою можливостей вирішення різноманітних завдань, нейромережі займатимуть дедалі більшу частину ринку.

Створення та генерація музики з використанням нейронних мереж стало однією з найбільш захоплюючих областей досліджень останнім часом. Музичні композиції та семпли є важливими елементами в музичній індустрії, і розробка інструментів, які можуть автоматично генерувати якісну музику, має великий потенціал для різних програм.

Актуальність цього дослідження полягає в наступних аспектах:

1) творчий потенціал: генерація музичних зразків за допомогою нейронних мереж відкриває нові можливості для музичних композиторів та продюсерів. Це дозволяє їм експериментувати з різними стилями та звуками, отримуючи унікальні та оригінальні композиції;

2) економічний потенціал: музична індустрія є багатомільярдним ринком і автоматична генерація музики може призвести до нових джерел доходу. Наприклад, інструменти для створення музичних семплів можуть бути продані виробникам музичного обладнання або використовуватися як основа для створення нових композицій;

3) різноманітність музичних стилів: нейронні мережі дозволяють створювати музичні зразки у різних жанрах та стилях, що сприяє розширенню культурного різноманіття та задоволенню смаків різних аудиторій.

4) музичне освітнє середовище: використання нейронних мереж для створення музики може мати значний потенціал у музичній освіті. Це може допомогти студентам і музикантам-початківцям розвивати свої навички у створенні музики і дозволити їм краще зрозуміти музичні принципи.

Дослідження музичної творчості: глибоке вивчення процесу генерації музики з використанням нейронних мереж може допомогти вченим дослідити та зрозуміти, які аспекти та принципи лежать в основі творчості та сприйняття музики.

Таким чином, створення нейронних мереж для генерації музичних зразків із заданим "beats per minute" (BPM) з колекції музичних семплів є актуальним завданням, яке має великий потенціал для індустрії розваг, музичної освіти та досліджень у галузі музичної творчості.

Мета і задачі дослідження

Мета дипломної роботи полягає у розробці та побудові ефективної нейронної мережі для генерації музичних зразків з заданою частотою ударів на хвилину на основі колекції музичних зразків.

Для досягнення поставленої мети сформульовані та послідовно вирішуються такі задачі:

1) обрати формат вхідних даних;

2) розробити архітектуру нейронної мережі для генерації музичних зразків з заданою частотою ударів на хвилину;

3) провести аналіз і підготовку колекції зразків;

4) навчити нейронну мережу на тренувальних даних з метою генерації музичних зразків з заданою частотою ударів на хвилину;

5) оцінити ефективність навченої моделі за допомогою метрик, таких як точність, розмаїтість та задоволення користувача;

6) порівняти розроблену нейронну мережу з існуючими методами та моделями для генерації музичних зразків;

7) зробити висновки щодо результатів дослідження та запропонувати можливі напрямки подальших досліджень у даній області.

Методи дослідження. Дослідження публікацій, наукових статей, книг та інших джерел, що стосуються побудови нейронних мереж для генерації музики. Ознайомлення з різними підходами та алгоритмами, використовуваними в цій галузі. Аналіз підходів, які використовуються для генерації музичних зразків із

заданою частотою ударів на хвилину. Тестування та автоматизоване тестування, синтез дослідження та аналіз досліджуваного матеріалу.

Робота складається зі вступу, основної частини, висновків, списку використаних джерел (16). Зміст роботи висвітлено на 43 сторінках основного тексту і містить 32 рисунка.

РОЗДІЛ 1
РОЗРОБКА НЕЙРОННОЇ МЕРЕЖІ

1.1 Вибір формату вхідних даних

Моделі машинного навчання використовуються як вхідні дані у формі числових векторів, які представляють вхідні дані, які ми хочемо надати моделі в зрозумілій формі. Як результат, ось наша перша проблема — як ми можемо представити музику як числа?

Щоб досягти цього, ми повинні думати про мелодію як про послідовність числових токенів, тоді як кожна лексема (вектор) має певну інформацію про ноту, ритм і тембр, а також інші характеристики, які ми можемо представити.

Ми можемо використовувати MIDI-файли (які використовуються для збереження, транспортування та відкриття музичних послідовностей) для навчання моделей, які ми досліджуватимемо. Файли MIDI (рис.1.1) — це структуровані файли, які надають інформацію, упорядковану за нотами, змінами ритму, BPM (ударів за хвилину) тощо. Потім ми використовуємо це представлення як природну мову для навчання наших моделей.

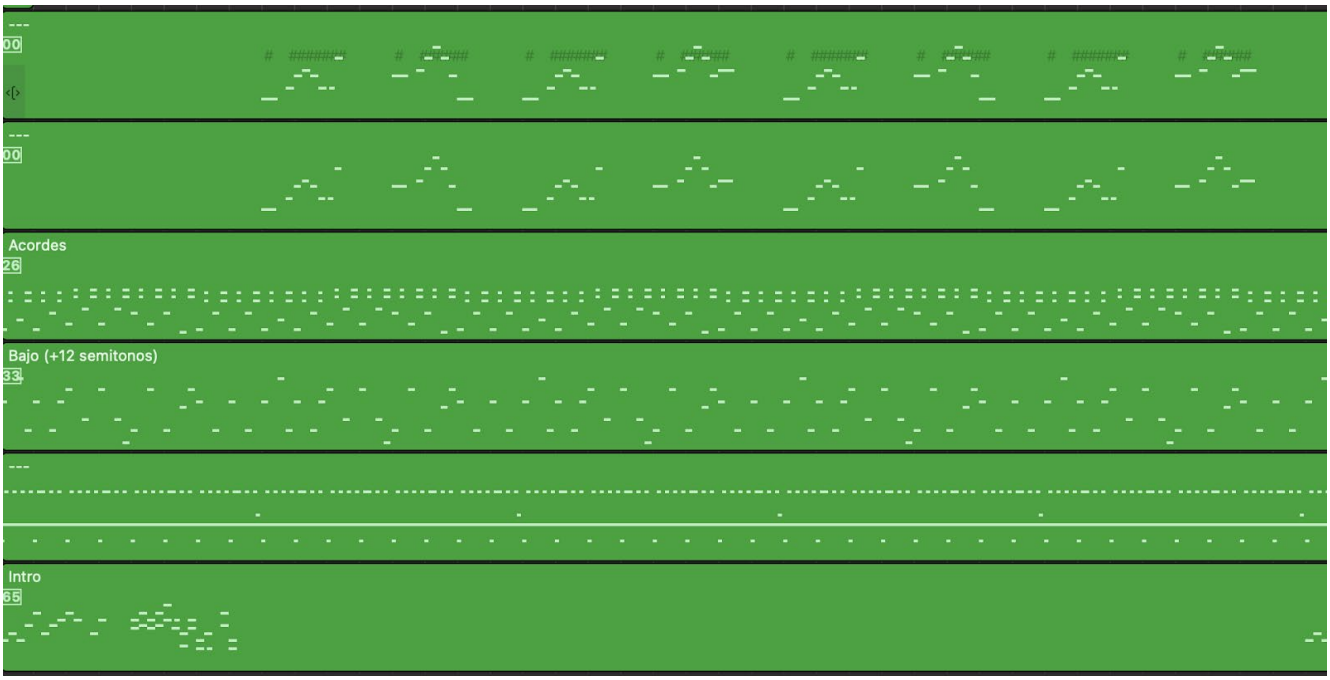


Рис.1.1 — MIDI файли з різними доріжками

З необробленим аудіо більшість алгоритмів використовуватиме необроблене представлення аудіо на кожному кроці часу. З послідовностями введення як вхідними векторами, більшість часу, моделі навчаються із завданням обробки природної мови (NLP) прогнозування наступного токена послідовності на кожному кроці часу.

1.2 Архітектура нейронної мережі

Існує декілька ефективних архітектур нейронних мереж для генерації музики: Recurrent Neural Networks, Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), Transformer-based моделі. Був обраний саме перший варіант, а саме LSTM (Long Short-Term Memory) який є типом рекурентних нейронних мереж, який спеціально розроблений для обробки та моделювання послідовних даних. Він має здатність ефективно управляти довготривалими залежностями в послідовностях із збереженням та передаванням важливої інформації.

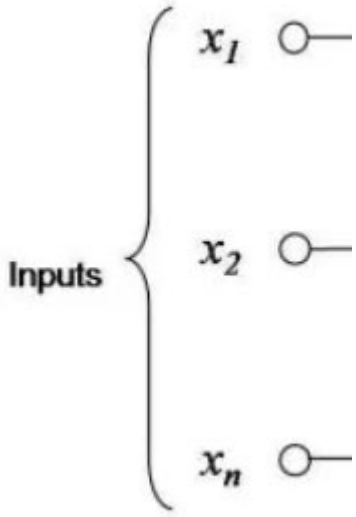
Нейронні мережі.

Що це взагалі таке і як воно працює?

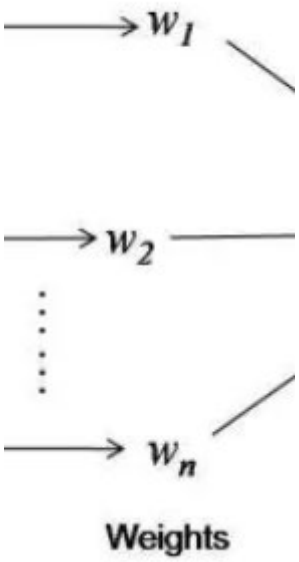
Основна ідея нейронної мережі полягає в тому, що вона може вчитися з досвіду, адаптуватися до змін і розв'язувати завдання, для яких вона була навчена. Ключовим елементом нейронної мережі є нейрон або вузол, який приймає вхідні дані, обчислює внутрішні операції на основі цих даних і генерує вихідний результат.

Нейрон складається з декількох компонентів:

1) вхідні дані: нейрон отримує вхідні дані або сигнали від інших нейронів або зовнішніх джерел. Ці дані представляються числовими значеннями або векторами, що відображають певні характеристики або властивості;

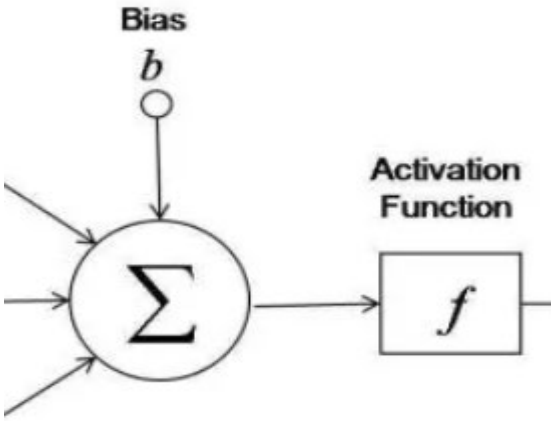


2) ваги: кожен вхідний сигнал множиться на вагу, яка відповідає його важливості для нейрона. Ваги визначають, наскільки сильно впливає кожен вхідний сигнал на вихід нейрона. Вони є налаштовуваними параметрами, які нейронна мережа вчиться оптимізувати під час тренування;



3) функція активації: після зваженого сумування вхідних сигналів і ваг нейрон застосовує функцію активації до отриманого значення. Функція активації визначає, як нейрон реагує на вхідні дані і як відображає свою активацію на вихідному сигналі. Вона може забезпечувати нелінійність, що дозволяє

нейронним мережам моделювати складні залежності між вхідними і вихідними даними;



4) вихідний сигнал: після застосування функції активації нейрон генерує вихідний сигнал, який може бути переданий іншим нейронам або використаний для розрахунків в мережі.

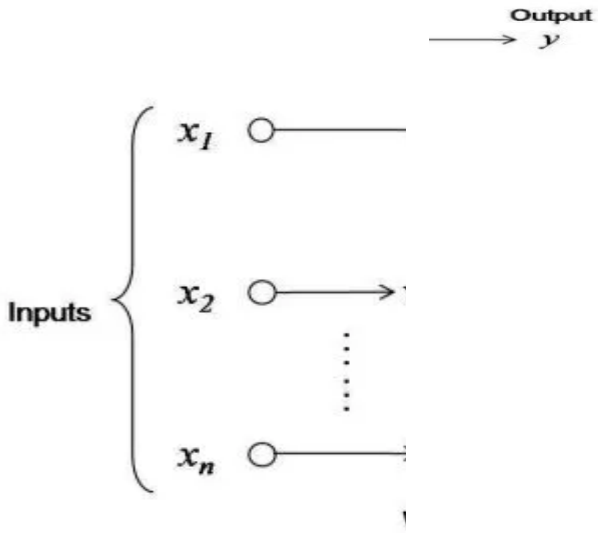


Рис.1.2 — Структура нейрона

Нейрони організовані в шари, де кожен шар приймає вихідні дані з попереднього шару і передає їх наступному шару. Перші шари називаються вхідними, останні — вихідними, а всі інші — прихованими. Кількість шарів та кількість нейронів в кожному шарі може варіюватися залежно від завдання.

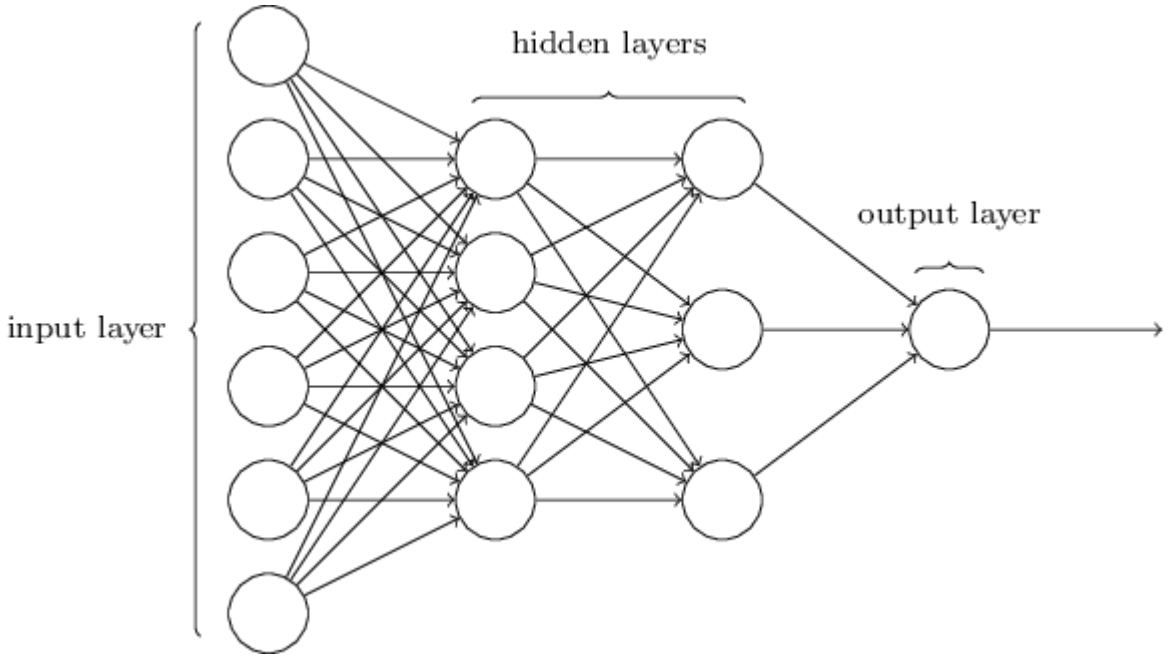


Рис.1.3 — Загальна структура нейронної мережі прямого розповсюдження

Коли дані пройшли через нейронну мережу, вона виконує процес називаний прямим поширенням (forward propagation). В кожному нейроні обчислюються зважені суми вхідних сигналів, які проходять через функцію активації. Це дозволяє нейронам "відповідати" на вхідні дані і генерувати вихідні значення. Процес прямого поширення продовжується через всі шари мережі до отримання вихідного результату.

Після прямого поширення використовується процес називаний зворотним поширенням (backpropagation), який дозволяє нейронній мережі вчитися і адаптуватися до навчальних даних. Зворотнє поширення використовується для обчислення градієнтів помилки від вихідного результату до початкових шарів мережі. Ці градієнти використовуються для оновлення ваг та параметрів нейронної мережі, що дозволяє їй набувати нових знань і покращувати свої результати.

1.3 Рекурентна нейронна мережа RNN (Recurrent Neural Network)

У той час коли нейронні мережі прямого розповсюдження эффективні у розпізнаванні образів, класифікації текстів та визначення їх емоційного забарвлення, оцінці ризиків, прогнозуванні цін і тд, у задачах генерації послідовностей будь то музика чи текст, використовується рекурентні нейронні мережі.

Основна відмінність RNN полягає в тому, що вона має рекурентні зв'язки, які дозволяють передавати інформацію з попередніх кроків обробки до поточного кроку.

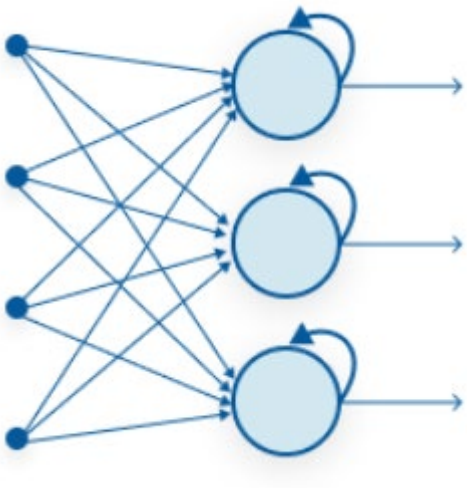


Рис.1.4 — Загальна структура RNN

LSTM (Long short-term memory).

LSTM — це покращена версія звичайної RNN завдяки здатності до довготривалої пам’яті. У звичайних RNN градієнт може швидко зникати або експоненційно зростати залежно від довжини послідовності. LSTM впроваджує механізми забудованої пам'яті, які допомагають зберігати та використовувати важливу інформацію з попередніх кроків часу. Це робить його більш ефективним у роботі з послідовностями, які містять довготривалі залежності. А саме це і важливо у задачі генерування музики.

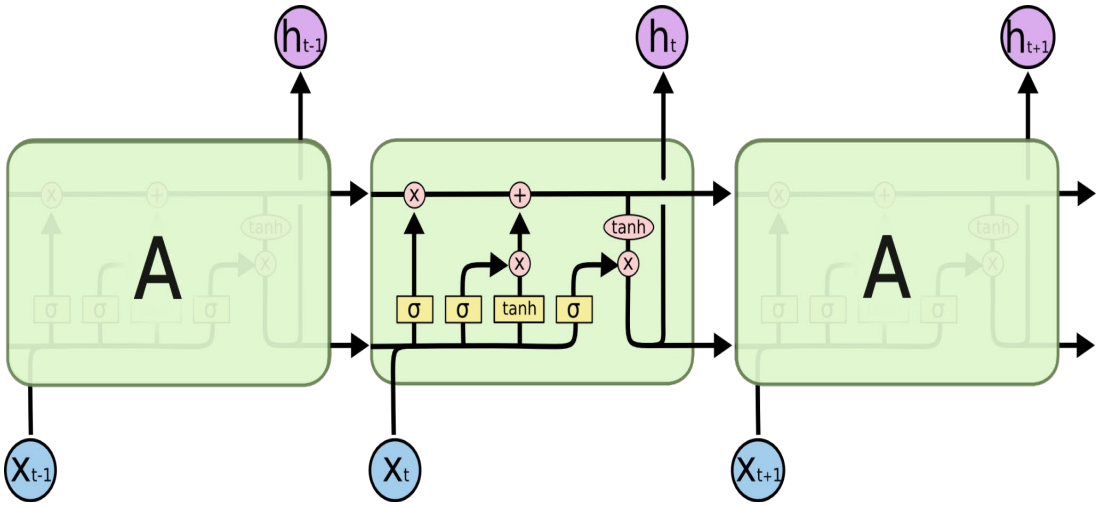


Рис.1.5 — Структура LSTM

Центральну роль у моделі LSTM відіграє комірка пам’яті, відома як «стан комірки», яка зберігає свій стан з часом. Стан комірки – це горизонтальна лінія, яка проходить через верхню частину діаграми нижче.

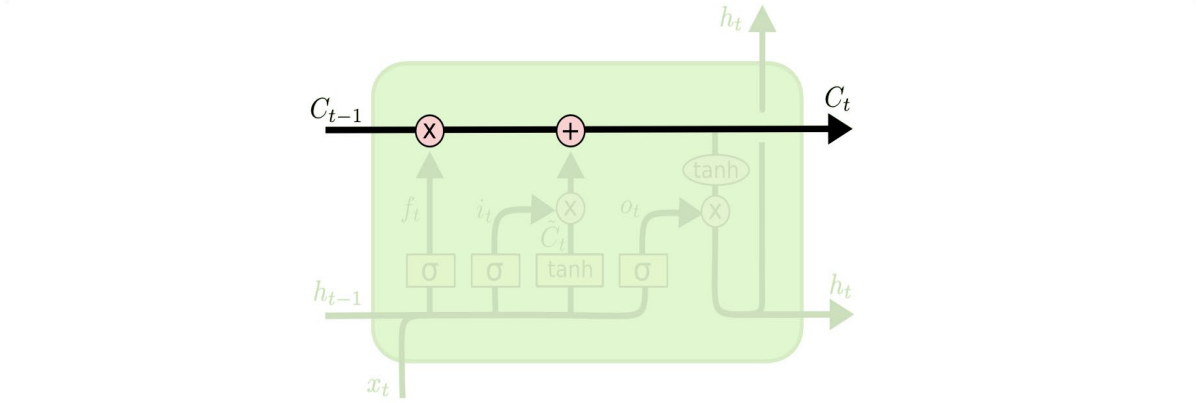


Рис.1.6 — Регулювання інформації в LSTM вентилями

Інформація може бути додана або видалена зі стану комірки в LSTM і регулюється вентилями. Ці ворота додатково пропускають інформацію в клітину та з неї. Він містить операцію поточкового множення та шар сигмоподібної нейронної мережі, які допомагають механізму.

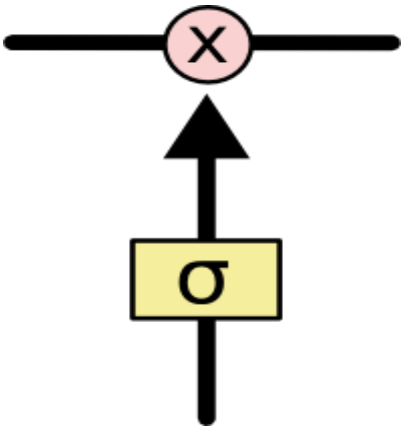


Рис.1.7 — Сигмоподібний шар

Сигмоподібний шар видає числа від нуля до одиниці, де нуль означає «нічого не повинно пропускатися», а один означає «все має пропускатися».

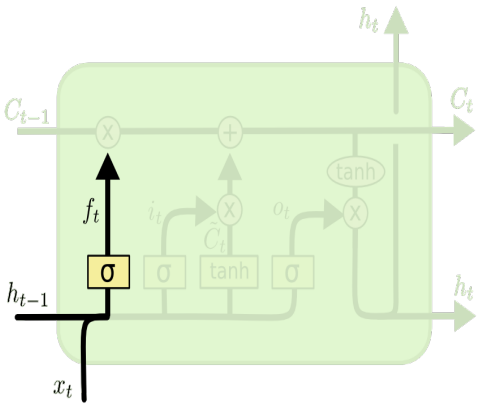
1.4 Використання LSTM в задачі генерування музики

Перший крок у LSTM — вирішити, яку інформацію ми збираємося викинути зі стану комірки. Це рішення приймається сигмоїдним шаром, який називається «шар забутих воріт». Він переглядає ht−1 і xt і виводить число від 0 до 1 для кожного числа в стані клітинки Ct−1. 1 означає «повністю зберегти це», а 0 означає «повністю позбутися цього».

Давайте повернемося до нашого прикладу музичної моделі, яка намагається генерувати наступну ноту на основі попередніх нот. У такій задачі стан комірки може включати певні характеристики, такі як темп, тембр або гучність, щоб мати можливість коректно передбачати наступну ноту з урахуванням цих характеристик. Коли ми маємо справу з новою мелодійною темою, ми хочемо забути попередні характеристики, щоб модель могла адаптуватися до нового стилю.

Таким чином, для використання LSTM в задачі генерування музики, ми можемо розширити стан комірки, щоб включити необхідні характеристики, які впливають на музичний контекст. Під час навчання моделі музичних

послідовностей, LSTM буде мати змогу зберігати та використовувати цю інформацію для генерації нових музичних фрагментів.

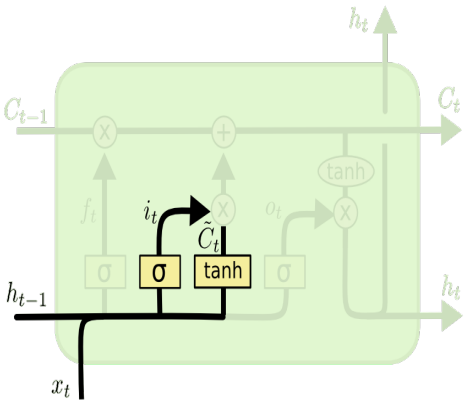


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Рис.1.8 — Зберігання інформації для генерації нових музичних фрагментів

Наступний крок — вирішити, яку нову інформацію ми збираємося зберігати в стані комірки. Це має дві частини. По-перше, сигмоподібний рівень, який називається «вхідним шлюзом», вирішує, які значення ми оновлюємо. Далі шар tanh створює вектор нових значень-кандидатів, C~t, які можна додати до стану. На наступному кроці ми об’єднаємо ці два, щоб створити оновлення стану.

У прикладі нашої мовної моделі ми хотіли б додати стан нового суб’єкта до стану клітинки, щоб замінити старий, про який ми забули.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

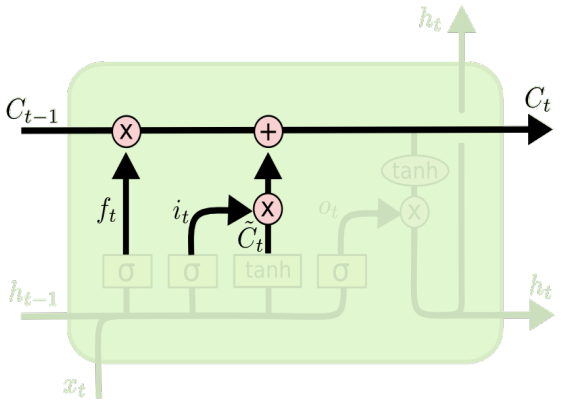
$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] \; + \; b_C\right)$$

Рис.1.9 — Додавання нового суб’єкта

Настав час оновити старий стан клітинки, Ct−1, на новий стан клітинки Ct. Попередні кроки вже вирішили, що робити, нам просто потрібно це зробити.

Ми множимо старий стан на фути, забуваючи те, що вирішили забути раніше. Потім ми додаємо його C~t. Це нові значення кандидатів, масштабовані за тим, наскільки ми вирішили оновити кожне значення стану.

У нашому випадку, це місце, де ми фактично скидаємо інформацію про характеристики ноти та додаємо нову інформацію, як ми вирішили на попередніх кроках.



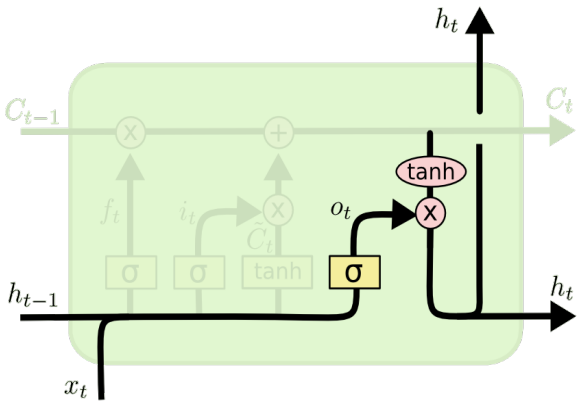
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Рис.1.10 — Додавання нової інформації

Нарешті, нам потрібно вирішити, що ми будемо виводити. Цей результат базуватиметься на стані клітинки, але буде відфільтрованою версією. Спочатку ми запускаємо сигмоподібний шар, який вирішує, які частини стану комірки ми збираємося вивести. Потім ми передаємо стан комірки через tanh (щоб підштовхнути значення між −1 і 1) і помножимо його на вихід сигмоїдного вентиля, щоб ми виводили лише ті частини, які вирішили.

Для прикладу мовної моделі, оскільки він щойно побачив суб’єкт, він може захотіти вивести інформацію, що стосується дієслова, на випадок, якщо це буде те, що буде далі. Наприклад, він може вивести, чи є підмет одниною чи

множиною, щоб ми знали, до якої форми слід відмінювати дієслово, якщо це те, що йде далі.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Рис.1.11 — Рішення о виводі

Для прикладу музичної моделі, оскільки вона щойно побачила ноту, вона може вивести інформацію, що стосується ритму або темпу, на випадок, якщо це буде те, що буде далі. Наприклад, вона може вивести, чи є наступна нота швидкою чи повільною, щоб ми знали, як треба відтворити наступну ноту у відповідності до темпу попередньої ноти.

Таким чином, у використанні LSTM для генерації музики, ми можемо додати до виводу моделі інформацію, що стосується ритму або темпу. Наприклад, після побачення попередньої ноти, модель може вивести відомості про те, чи має бути наступна нота швидкою або повільною. Це дозволить моделі генерувати музику з врахуванням ритмічних особливостей та темпу музичної композиції.

Як LSTM генерує послідовність?

На вхід до LSTM подається послідовність елементів (abc), на основі якого модель робить прогнозування наступного елемента (d). Далі з вхідної послідовності прибрається перший елемент (a), а у кінець додається сгенерований (d). Нова послідовність (bcd) знову подається на вхід до моделі. Таким чином можна робити скільки завгодно раз і отримувати нові дані.

bcd -> cde -> def -> efg -> …

На прикладі з генеруванням слів, це виглядає таким чином:

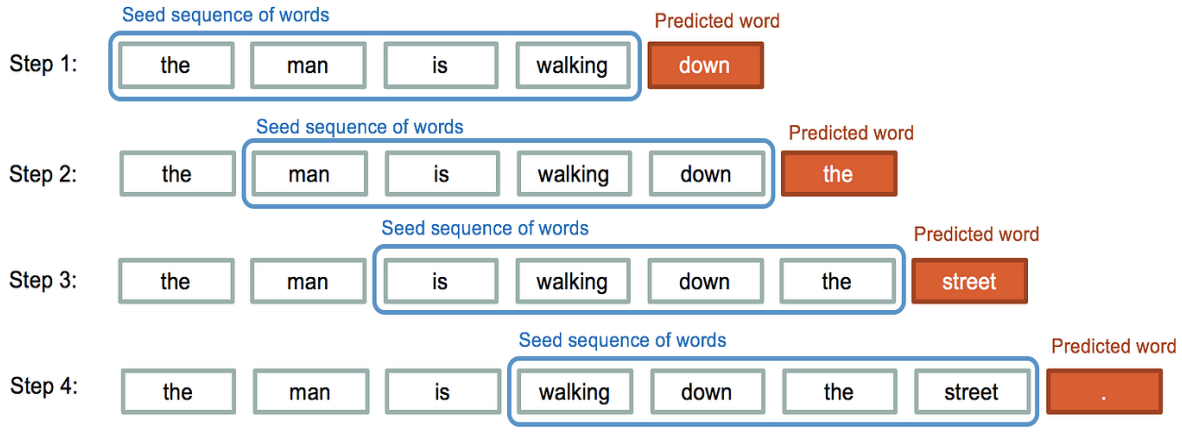


Рис.1.12 — Модель генерування слів

Генерування музики відбувається так само, але замість слів — ноти.

1.5 Підготовка даних

По-перше, з датасету обираються лише ті зразки, які мають лише одну доріжку, тобто основна мелодія для одного інструменту. Ті зразки, у яких перша доріжка відповідає за основну мелодію також можна залишити, і при навчанні використовувати лише її. На рисунку 1.13 зображений етап вибору зразків, які мають лише одну доріжку.

```
def filter_midi_with_n_instruments(filenames, n_instruments = 1):
  new_filenames = []

  for filename in filenames:
      try:
        pm = pretty_midi.PrettyMIDI(filename)
        if len(pm.instruments) == n_instruments:
            new_filenames.append(filename)
        else:
          for instrument in pm.instruments:
            instrument_name = pretty_midi.program_to_instrument_name(instrument.program)
            # print('Instrument name:', instrument_name)
      except:
        print(f"Преобразование прошло неудачно - {filename}")

  print(f'filter_midi_with_n_instruments - {len(new_filenames)}')
  return new_filenames
```

Рис.1.13 — Етап вибору зразків, які мають лише одну доріжку

По-друге, прибираю занадто короткі композиції: залишаю лише ті зразки які тривають хоча б 60 секунд. Занадто короткі зразки можуть бути короткими темами, звуками оточення, випадковими звуками чи будь-чим ще, залежно від датасету. Заздалегідь калібровані датасети зазвичай не містять таких семплів, але з метою обережності, буде пройдений цей етап. На рисунку 1.14 зображений етап прибирання занадто коротких зразків.

```
def filter_midi_with_n_length(filenames, n_length = 60):
  new_filenames = []

  for filename in filenames:
      try:
        pm = pretty_midi.PrettyMIDI(filename)
        if remove_initial_pauses(pm).get_end_time() > n_length:
            # print(f'{filename}: {pm.get_end_time()}')
            new_filenames.append(filename)
      except:
        print(f"Преобразование прошло неудачно - {filename}")

  print(f'filter_midi_with_n_length - {len(new_filenames)}')
  return new_filenames
```

Рис.1.14 — Етап прибирання занадто коротких зразків

По-третє, прибираю зразки у яких занадто низький bpm. Так само, причиною занадто низького bpm може бути те, що семпл був спроектований для конкретного випадку чи локації та містить у собі лише кілька нот, які програються протягом тривалого часу. На рисунку 1.15 зображений етап прибирання зразків з занадто низьким bpm.

```
def filter_midi_with_n_tempo(filenames, n_tempo = 60):
  new_filenames = []

  for filename in filenames:
      try:
        pm = pretty_midi.PrettyMIDI(filename)
        if pm.estimate_tempo() > n_tempo:
            # print(f'{filename}: {pm.estimate_tempo()}')
            new_filenames.append(filename)
      except:
        print(f"Преобразование прошло неудачно - {filename}")

  print(f'filter_midi_with_n_tempo - {len(new_filenames)}')
  return new_filenames
```

Рис. 1.15 — Етап прибирання зразків з занадто низьким bpm

Всі ці фільтрації збільшать ефективність навчання та покращать результат.

На вхід LSTM приймає послідовність з N нот. Нота складається з 3 параметрів — pitch (висота звучання), step (крок, або коли саме необхідно відтворити звучання) та duration (тривалість звучання). Тому після усіх фільтрацій усі MIDI файли за допомогою бібліотеки pretty_midi та функцій власноруч написаних мною у google colab, перетворюються на необхідну послідовність нот. На рисунку 1.16 зображене перетворення MIDI файлу на ноти.

У Додатку 1 представлена функція remove_initial_pauses. Вона потрібна для прибирання паузи на початку семплу, якщо вона є.

```python
def midi_to_notes(filename: str) -> pd.DataFrame:
  pm = pretty_midi.PrettyMIDI(filename)
  pm = remove_initial_pauses(pm)
  # pm = set_tempo(pm, 100)
  instrument = pm.instruments[0]
  notes = collections.defaultdict(list)

  # Sort the notes by start time
  sorted_notes = sorted(instrument.notes, key=lambda note: note.start)
  prev_start = sorted_notes[0].start

  for note in sorted_notes:
    start = note.start
    end = note.end
    notes['pitch'].append(note.pitch)
    notes['start'].append(start)
    notes['end'].append(end)
    notes['step'].append(start - prev_start)
    notes['duration'].append(end - start)
    prev_start = start

  return pd.DataFrame({name: np.array(value) for name, value in notes.items()})
```

Рис. 1.16 — Етап перетворення MIDI файлу на ноти

За допомогою функції filenames_to_notes ми перетворюємо на ноти усі підготовлені файли. Цей процес зображений на рисунку 1.17.

```python
def filenames_to_notes(filenames):
  all_notes = []
  for f in filenames[:num_files]:
    try:
        notes = midi_to_notes(f)
        all_notes.append(notes)
    except:
        print(f"Преобразование прошло неудачно - {f}")

  all_notes = pd.concat(all_notes)
  print(f'filenames_to_notes - {len(all_notes)}')
  return all_notes
```

Рис. 1.17 — Етап перетворення усіх підготовлених файлів на ноти

У Додатку 2 показана функція create_sequence.

За допомогою перелічених вище функцій ми підготували тренувальні дані для моделі, цей процес зображений на рисунку 1.18.

```
[ ]  key_order = ['pitch', 'step', 'duration']
     train_notes = np.stack([all_notes[key] for key in key_order], axis=1)

[ ]  notes_ds = tf.data.Dataset.from_tensor_slices(train_notes)
     notes_ds.element_spec

     TensorSpec(shape=(3,), dtype=tf.float64, name=None)

[ ]  seq_ds = create_sequences(notes_ds, seq_length, vocab_size)
     seq_ds.element_spec

     (TensorSpec(shape=(50, 3), dtype=tf.float64, name=None),
      {'pitch': TensorSpec(shape=(), dtype=tf.float64, name=None),
       'step': TensorSpec(shape=(), dtype=tf.float64, name=None),
       'duration': TensorSpec(shape=(), dtype=tf.float64, name=None)})

     buffer_size = n_notes - seq_length  # the number of items in the dataset
     train_ds = (seq_ds
                 .shuffle(buffer_size)
                 .batch(batch_size, drop_remainder=True)
                 .cache()
                 .prefetch(tf.data.experimental.AUTOTUNE))
     train_ds.element_spec

     (TensorSpec(shape=(100, 50, 3), dtype=tf.float64, name=None),
      {'pitch': TensorSpec(shape=(100,), dtype=tf.float64, name=None),
       'step': TensorSpec(shape=(100,), dtype=tf.float64, name=None),
       'duration': TensorSpec(shape=(100,), dtype=tf.float64, name=None)})
```

Рис. 1.18 — Процес підготовки даних

1.6 Навчання нейромережі

Відразу слід зазначити, що кожна моя наступна модель буде завершуватися трьома повнозв’язними паралельними (Dense) шарами:

1) шар “pitch_output”, який складається з 128 нейронів із функцією втрат (loss) SparseCategoricalCrossentropy (за допомогою цієї функції втрат можна

оцінювати розрізненість між класами висоти звучання). Бібліотека для взаємодії з midi файлами, яка використовується, а саме pretty_midi має 128 классів для pitch, тому був обраний саме таку кількість нейронів;

2) шар “step_output” має один нейрон та використовує функцію втрат, яка поєднує середньоквадратичну помилку (MSE) з додатковим терміном "positive pressure" (позитивний тиск), який заохочує модель генерувати значення, більші або рівні нулю. Головна причина використання такої функції втрати полягає в тому, що вихідні значення кроку і тривалості музичних нот можуть бути додатними числами, або навіть нулем. Це означає, що прогнозовані значення також повинні бути не меншими за нуль, щоб бути фізично вірними;

3) шар “duration_output”, який так само, як і попередній, має 1 нейрон, то функцію активації mse_with_positive_pressure.

Стартові характеристики такі:

1) довжина послідовності = 25;

2) кількість епох = 50;

3) швидкість навчання = 0.005.

Для оцінювання результатів моделі, використовується «втрата» або «функція втрати» (англ. loss function), яка використовується для вимірювання різниці між прогнозованими значеннями моделі і фактичними значеннями даних. Вона є метрикою, яка вказує на те, наскільки добре модель відповідає на поставлену задачу. У моєму випадку втрати дорівнюють сумам втрат з pitch помножений на 0.05, step та duration.

Також треба відзначити, що на початку не використовується увесь датасет, а лише кілька семплів з нього, щоб прискорити навчання.

Перша спроба

Модель складається з одного шару LSTM на 128 нейронів.

Загальна кількість параметрів для навчання складає 84 354.

Layer (type)	Output Shape	Param #	Connected to
input_4 (InputLayer)	[(None, 25, 3)]	0	[]
lstm_3 (LSTM)	(None, 128)	67584	['input_4[0][0]']
duration (Dense)	(None, 1)	129	['lstm_3[0][0]']
pitch (Dense)	(None, 128)	16512	['lstm_3[0][0]']
step (Dense)	(None, 1)	129	['lstm_3[0][0]']

Total params: 84,354
Trainable params: 84,354
Non-trainable params: 0

Рис.1.19 — Модель з одного шару LSTM на 128 нейронів

Графік втрат має наступний вигляд: по осі абсцис — кількість епох, по осі ординат — втрати.

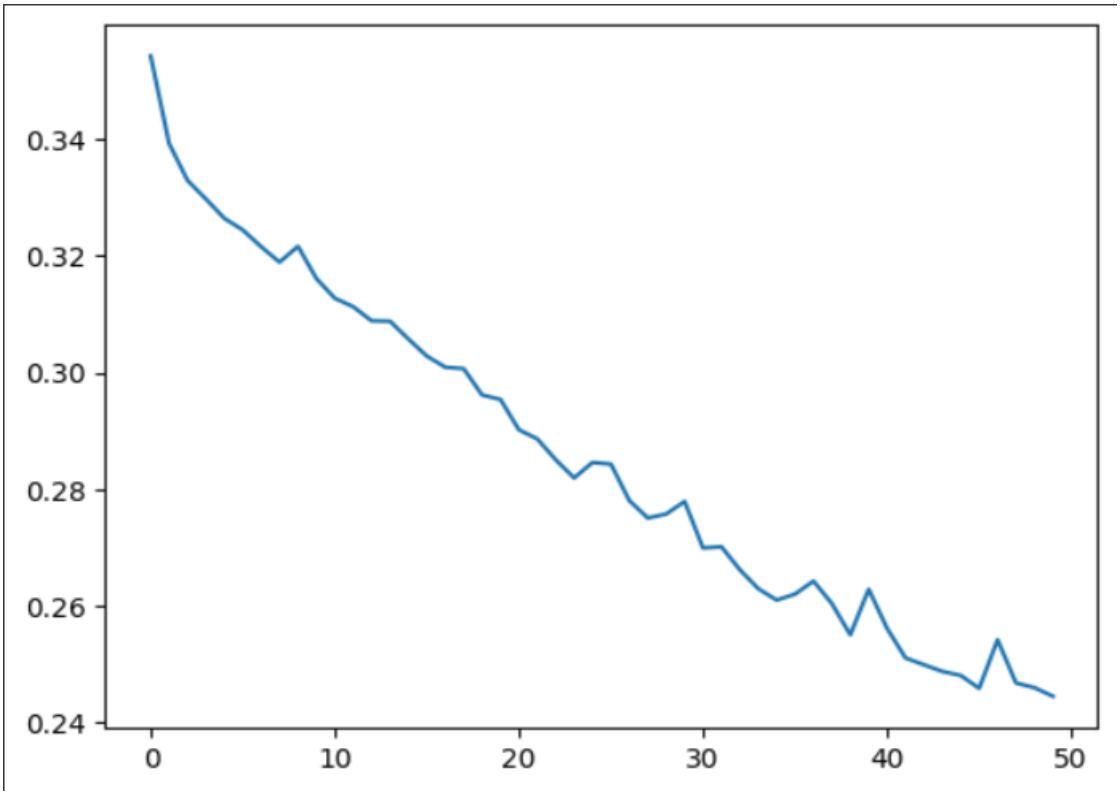


Рис.1.20 — Графік втрат для спроби 1

Семпли, що вона генерує більше схожі на випадкове натискання клавіш, ніж на музику.

Друга спроба

Зроблю перший шар LSTM двонаправленим.

Двонаправлений (Bidirectional) шар в нейронних мережах дозволяє одночасно обробляти вхідні дані в обох напрямках: в прямому (вперед) і зворотному (назад). Він використовує дві копії рекурентного шару LSTM і передає вхідні дані через цей шар в обох напрямках.

Перевагою Bidirectional слою є те, що він може використовувати як контекст інформацію з майбутніх вхідних значень (зворотний напрямок), так і з попередніх вхідних значень (прямий напрямок). Це дозволяє моделі ухватити залежності і контекст як з переду, так і зі звороту, що може бути корисним у багатьох завданнях обробки послідовностей.

Загальна кількість параметрів для навчання складає 168 578

Графік втрат зображений на рис. 1.21.

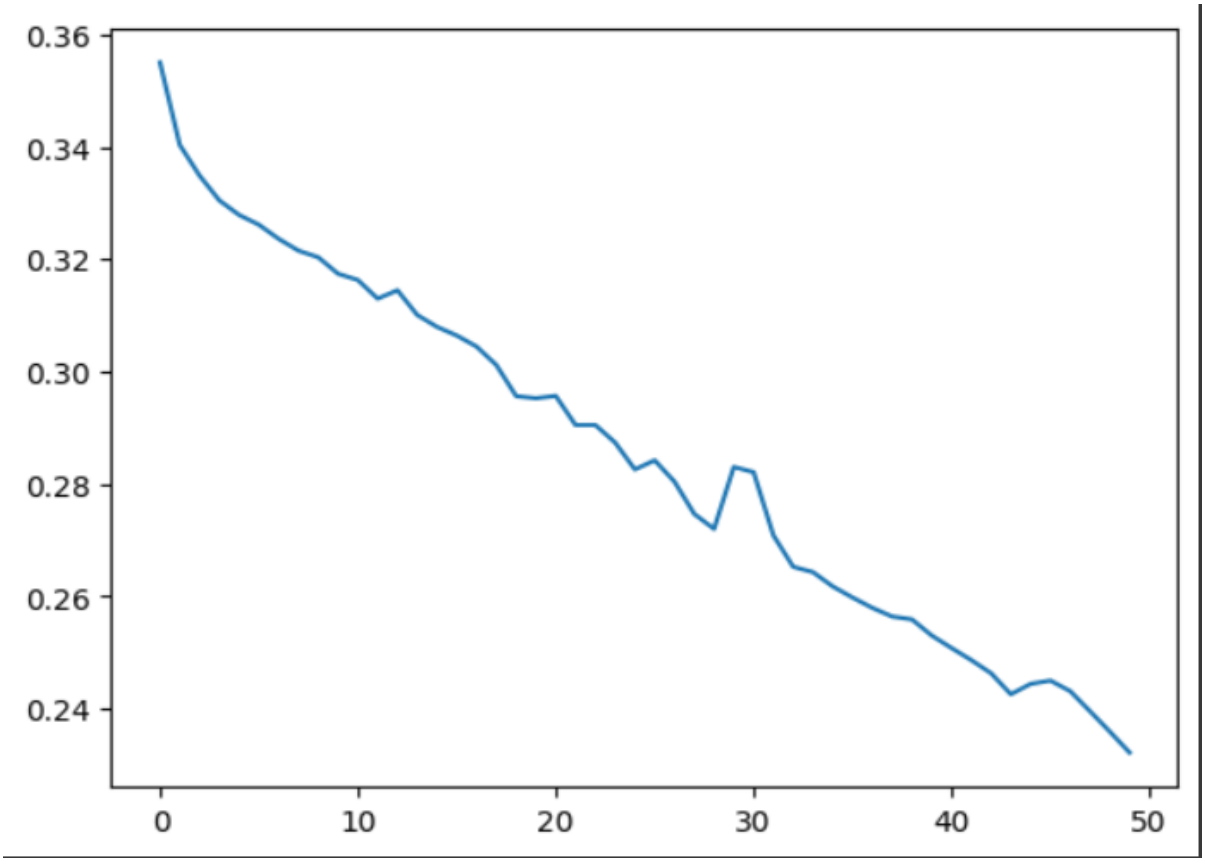


Рис.1.21 — Графік втрат для спроби 2

Можемо побачити, що втрати нібито зменшились, але така модель генерує ноти, які відтворюються всі одночасно на початку семплу, що звучить як суцільний звук або шум, та зовсім не схоже на музику.

Можлива причина такої поведінки — перенавчання (англ. overfitting) — це явище в машинному навчанні, коли модель навчається "занадто добре" на тренувальних даних, але показує погані результати на нових даних.

Існує декілька методів боротьби з перенавчанням: збільшення об'єму тренувальних даних, використання регуляризації (наприклад, L1 або L2 регуляризація), використання методів зниження швидкості навчання та використання методів ансамблювання, таких як випадкові ліси або градієнтний бустинг. Спочатку використовується Dropout. Механізм Dropout полягає в тимчасовому випадковому вимкненні (викиданні) деяких нейронів під час навчання. Кожен нейрон, який підлягає Dropout, має певну ймовірність бути вимкненим у кожній ітерації тренування. Це означає, що інші нейрони в мережі повинні взяти на себе відповідальність за розпізнавання та використання функцій, які попередньо виконувалися вимкненими нейронами. Внаслідок цього зменшується кореляція між нейронами, що сприяє більшому розподілу обчислювального навантаження та покращенню ефективності обчислень.

Третя спроба

Перший шар як і раніше — двонаправлений LSTM зі 128 нейронами.

За ним другий шар Dropout з коефіцієнтом 0.3. Це означає, що під час навчання 30 відсотків випадкових нейронів першого шару вимикається, залишаючи усі інші 70 відсотків (89 нейронів) працювати.

Загальна кількість параметрів для навчання не змінилася та складає 168 578

Графік втрат зображений на рис.1.22.

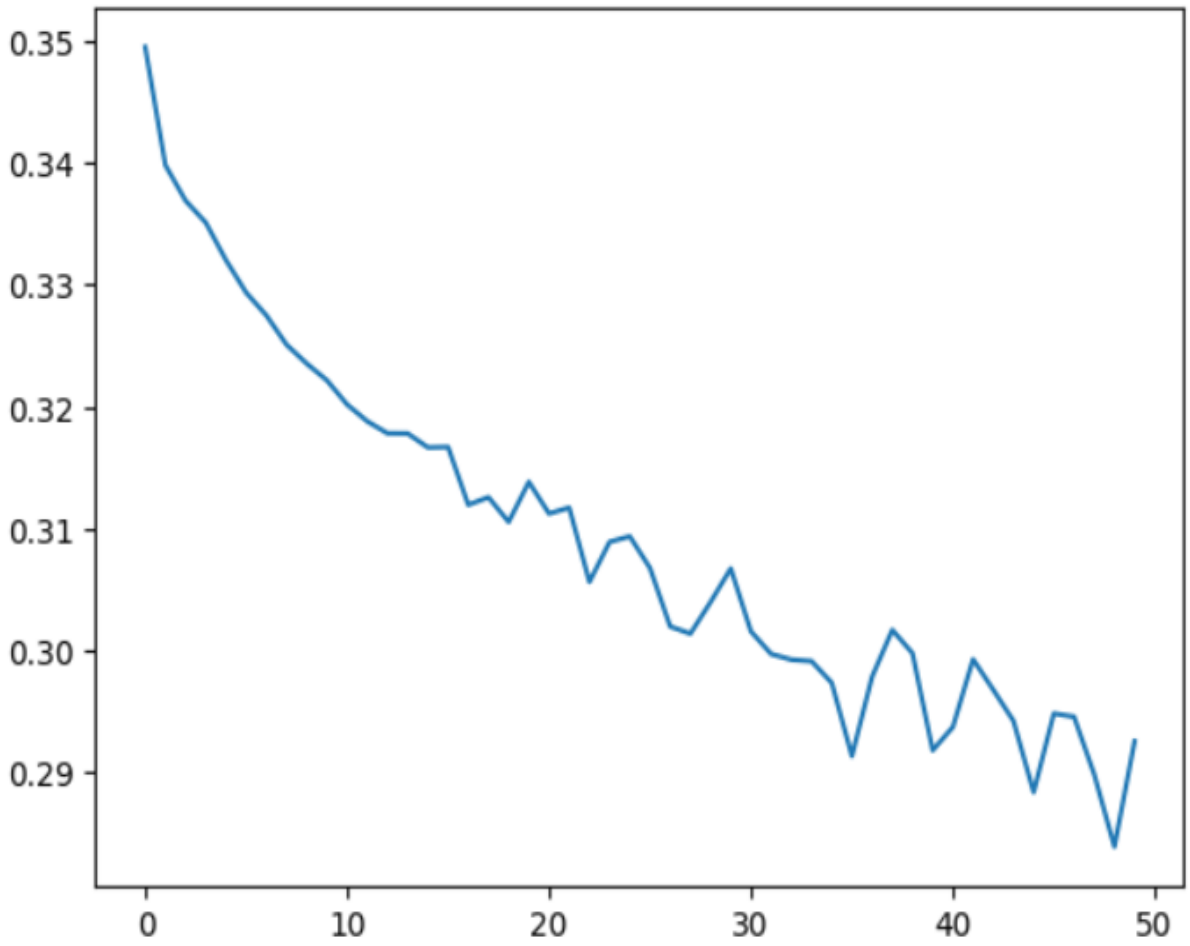


Рис.1.22 — Графік втрат для спроби 3

Отримані семпли складаються з декількох нот, які почергово програються протягом усього часу. Це краще, але все ще не схоже на музику. Приклад такого семплу виглядає так:

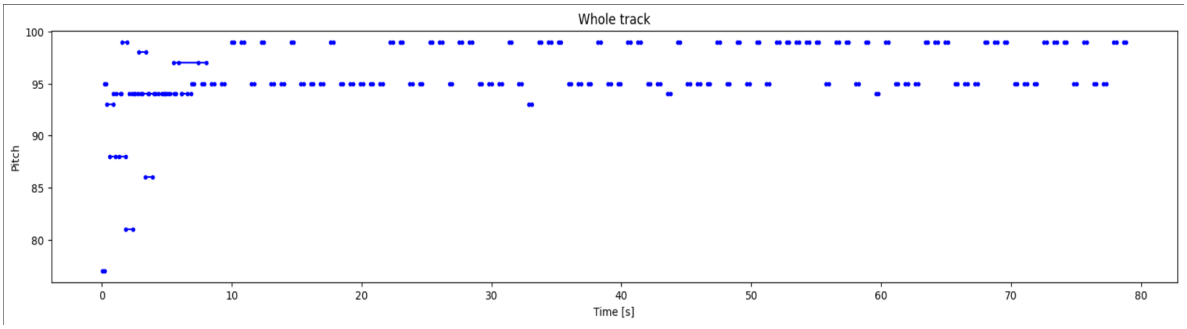


Рис.1.23 — Приклад семплу

Ще один спосіб покращити результати роботи нейронної мережі — збільшення кількості шарів. Такий прийом дозволяє моделі мати більше рівнів абстракції та складніші представлення даних. Кожен шар може виконувати

послідовну обробку даних, поступово визначаючи все більш високорівневі особливості. Це дозволяє моделі вчитися складнішим залежностям і отримувати більш точні прогнози.

Найлегшим прикладом для сприйняття я вважаю класифікацію рукописних цифр за допомогою мережі прямого поширення. Ця мережа складається з кількох шарів: перший шар має N нейронів, що відповідає кількості пікселів у зображенні цифри. Другий шар налічує M нейронів, що можна уявити як поєднання декількох пікселів або рисочки з різними кутами нахилу та різної величини, з яких складається цифра. Наступний шар ще більш високорівневий і відповідає вже за складніші фігури, такі як кружечок у вісімці та шістці та паличка у четвірці та одиниці. Останній шар складається з 10 нейронів, кожен з яких відповідає цифрі від 0 до 9. У музиці це більш складні та довгі послідовності нот. Сказати який шар за що істотно відповідає важко, але це і не так важливо, головне — це працююча концепція.

Четверта спроба

Структура мережі така:

1) Двонаправлений LSTM зі 128 нейронами. Return_sequences=True.
2) Dropout з коефіцієнтом 0.3.
3) Двонаправлений LSTM зі 128 нейронами.
4) Dropout з коефіцієнтом 0.3.

Return_sequences=True між двома шарами нейронної мережі дозволяє забезпечити правильну передачу і збереження інформації про послідовність через всю мережу, що є важливим для генерації музики з врахуванням контексту та залежностей між музичними елементами.

Щоб LSTM працювала так, як і повинна, на вхід вона повинна приймати послідовність. Якщо ж не повертати послідовність з першого шару мережи, то до другого шару буде потрапляти лише одна нота яка є прогнозом першого шару, що не має багато сенсу.

Кількість нейронів такої мережі дорівнює 562,818

Графік втрат зображений на рис. 1.24.

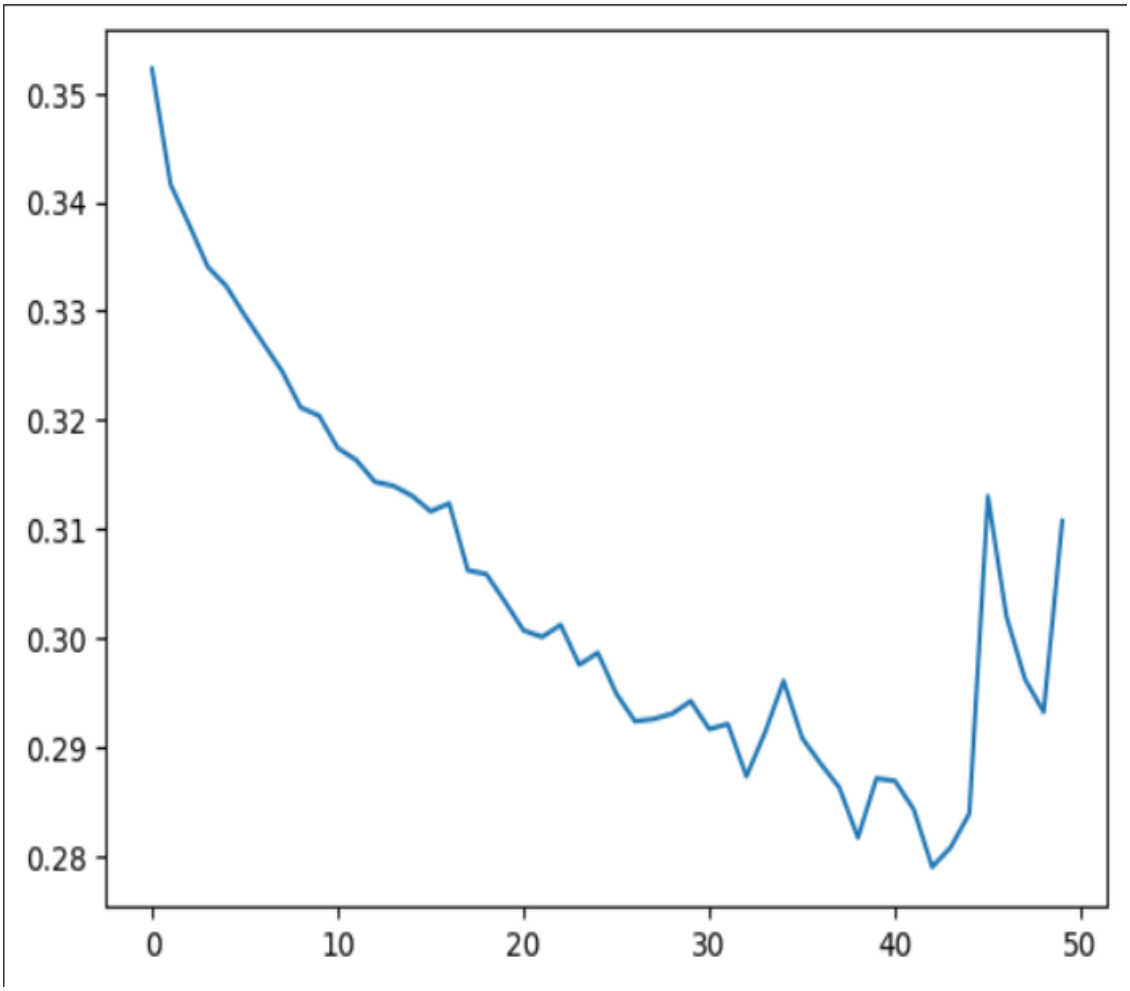


Рис.1.24 — Графік втрат для спроби 4

Результат на початку схожий на музику, але у кінці щось йде не так. Скоріш за все справа у недостатній потужності моделі, тому треба її збільшити.

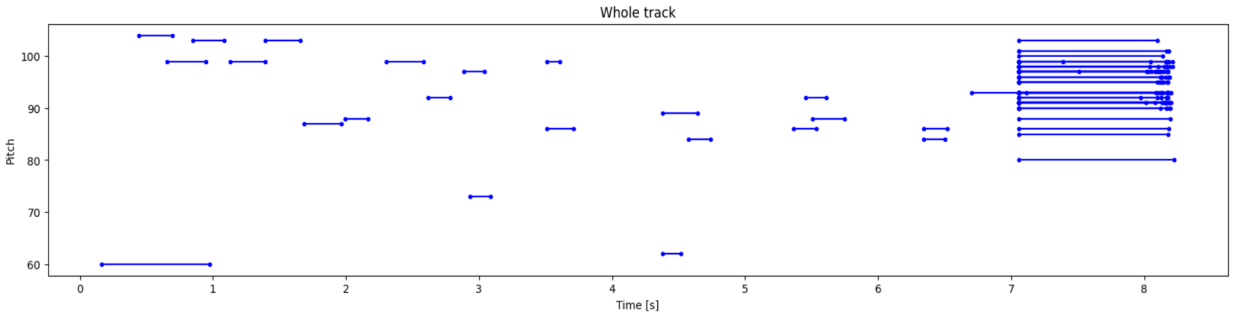


Рис.1.25 — Приклад семплу для спроби 4

Щоб збільшити потужність моделі можна збільшити кількість нейронів у кожному з шарів. Більша кількість нейронів дозволяє моделі мати більшу обчислювальну потужність і здатність вчитися складнішим залежностям у даних.

Це особливо корисно для завдань, де простіші моделі не здатні зробити точні прогнози.

Збільшення кількості нейронів дозволяє моделі виразити більше різноманітних функцій та залежностей між вхідними та вихідними даними. Це може бути корисно для завдань, де важлива точність та деталізація прогнозів.

Ще важливою перевагою такого підходу є зменшення недофітінгу: занадто мало нейронів у шарі може призвести до недофітінгу моделі, коли вона не здатна адекватно апроксимувати вихідні дані. Збільшення кількості нейронів допомагає зменшити цей ризик і покращити апроксимацію функції.

Важливо зазначити, що збільшення кількості нейронів також може призвести до більшої обчислювальної складності та затрат пам'яті, особливо для великих моделей. Тому вибір оптимальної кількості нейронів потребує балансу між точністю моделі та обчислювальною ефективністю.

П’ята спроба

Структура мережі:

1) Двонаправлений LSTM зі 512 нейронами. return_sequences=True.

2) Dropout з коефіцієнтом 0.3.

3) Двонаправлений LSTM зі 512 нейронами.

4) Dropout з коефіцієнтом 0.3

Така мережа вже містить 8,542,338 параметрів і її навчання може зайняти дуже багато часу.

Графік втрат зображений на рис.1.26.

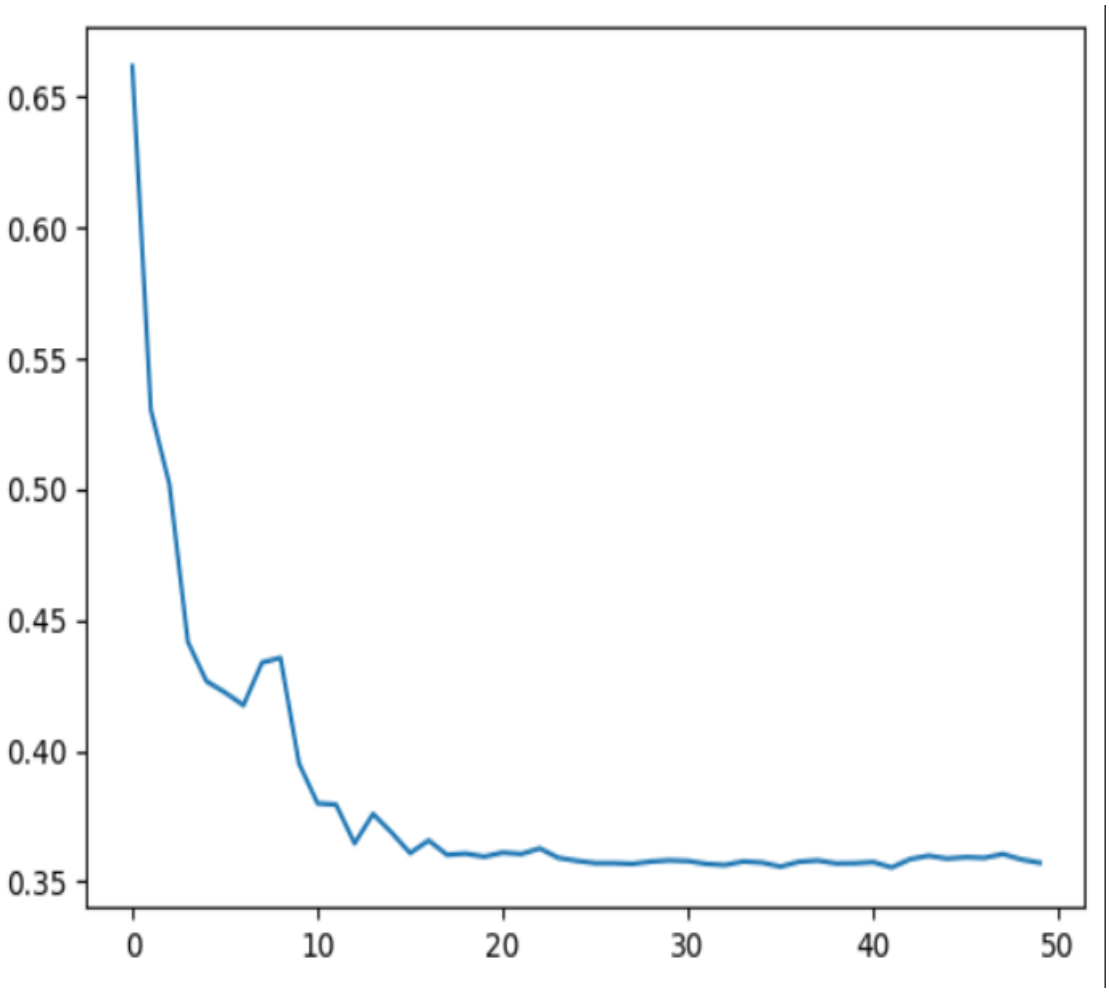


Рис.1.26 — Графік втрат для спроби 5

Приклади семплів:

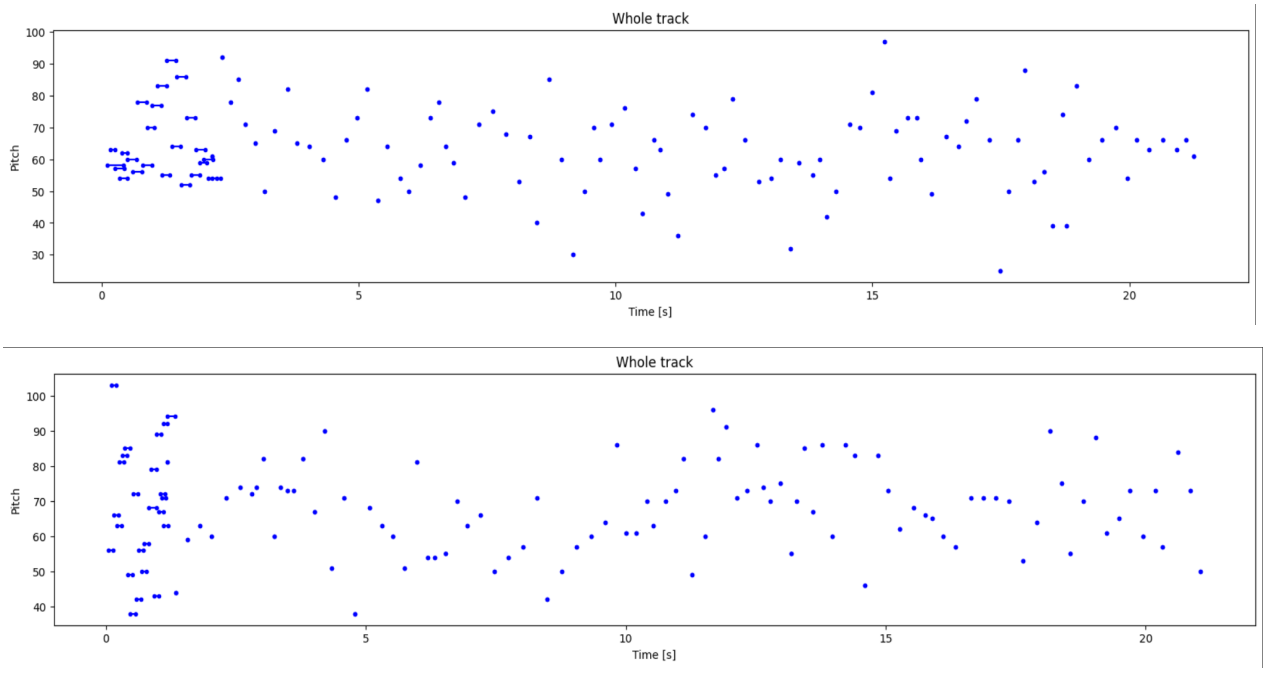


Рис.1.27 — Приклади семплів для спроби 5

Це вже більше схоже на музику, можна справді почути деякі відомі патерни, але здебільшого це все ще досить випадкове натискання клавіш.

Що можна зробити далі?

Я пропоную використати механізм Self Attention (самоувага). Самоувага є ключовим елементом архітектури трансформерів, що використовується в багатьох задачах обробки природної мови, включаючи генерування музики. Використання самоуваги у задачі генерації музики дозволяє моделі взаємодіяти з різними частинами вхідної послідовності та визначати їх вагомість в контексті генерації наступної частини музичного зразка.

Шоста спроба

Структура мережі:

1) Двонаправлений LSTM зі 512 нейронами. return_sequences=True.

2) Self Attention. attention_activation='sigmoid'.

3) Dropout з коефіцієнтом 0.3.

4) Двонаправлений LSTM зі 512 нейронами. return_sequences=True.

5) Dropout з коефіцієнтом 0.3.

6) Flatten.

Активаційна функція sigmoid приймає значення від нуля до одиниці, і вона зазвичай використовується для моделювання ймовірності або ваг у діапазоні від 0 до 1. У контексті Self Attention шару, використання sigmoid дозволяє моделі приділити більшу увагу тим частинам вхідного сигналу, для яких значення уваги ближче до 1, тобто вони вважаються більш важливими або суттєвими для обробки.

Використання активаційної функції sigmoid у Self Attention шарі дозволяє моделі акцентувати увагу на важливих аспектах вхідного сигналу, фокусуючись на відповідних деталях, які впливають на генерацію музичних зразків у вашій задачі. Це може допомогти поліпшити якість та точність генерації музики, оскільки модель може зосередитись на важливих аспектах ритму, мелодії та інших аспектів музичної композиції.

Другий LSTM шар тепер також повертає послідовність. Але це збільшує послідовність вихідного вектора, тому треба використати шар Flatten який розгортає вхідні дані з багатовимірного формату до одновимірного формату.

Кількість параметрів: 11,802,819.

Графік втрат зображений на рис. 1.28.

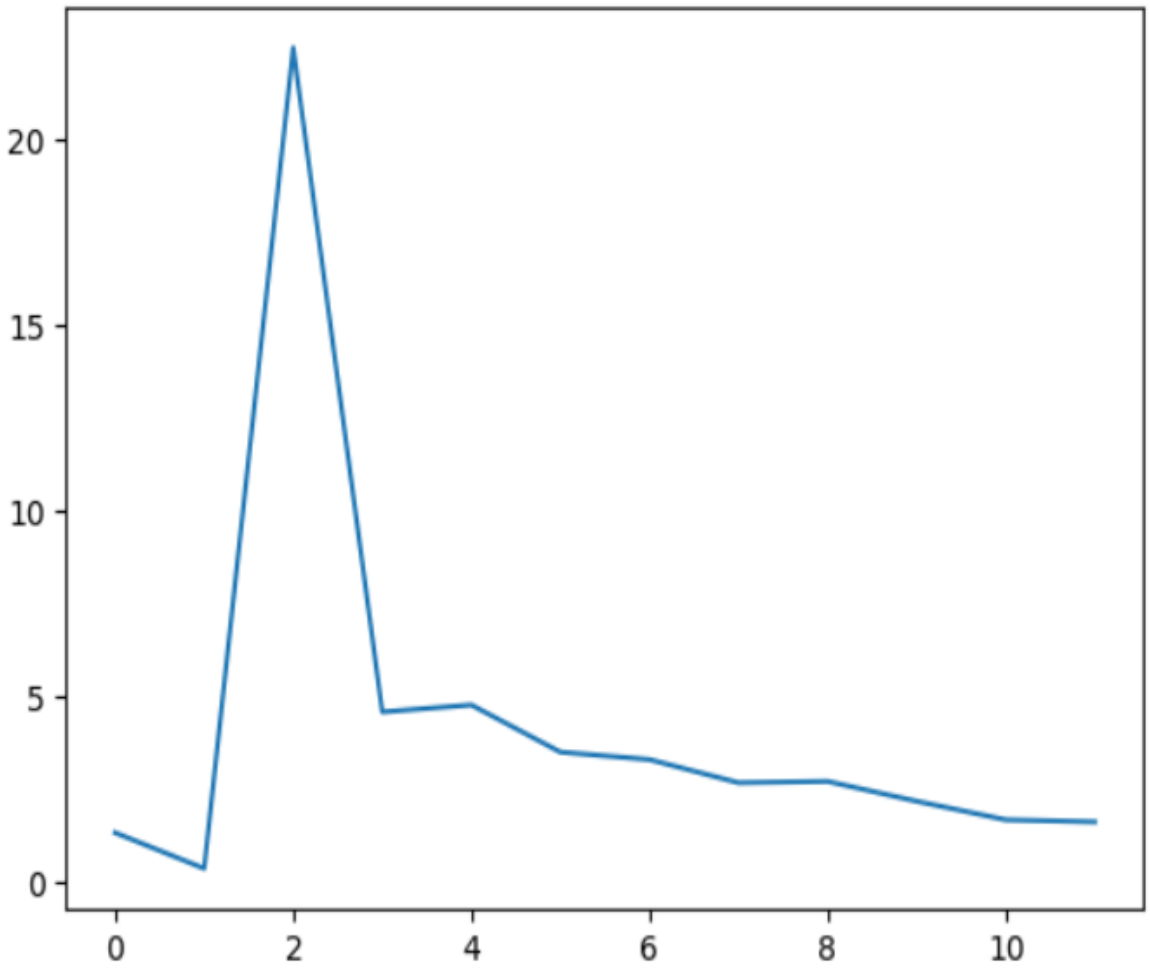


Рис.1.28 — Графік втрат для спроби 6

Спроба 6 була невдалою, скоріш за все, справа у тому, що кількість нейронів занадто велика, і модель не здатна навчити їх усі.

Семпл, отриманий у результаті музикою не назвати.

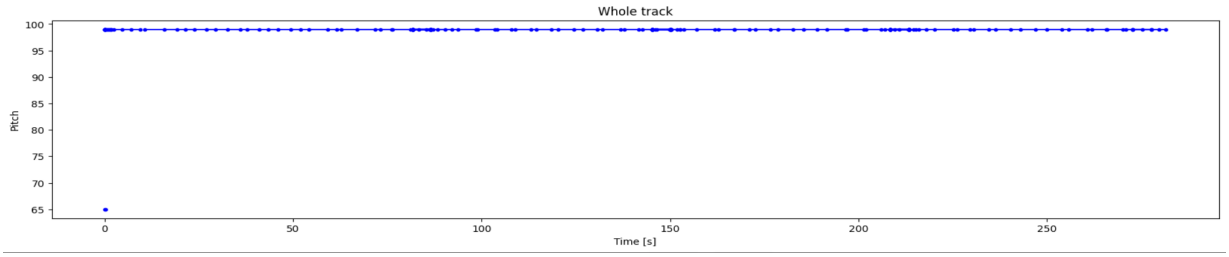


Рис.1.29 — Приклад семплу для спроби 6

Для наступної спроби буде зменшено кількість нейронів у кожному LSTM шару та збільшений розмір вхідної послідовності нот.

Сьома спроба

Структура мережі:

1. Двонаправлений LSTM зі 256 нейронами. return_sequences=True.
2. Self Attention. attention_activation='sigmoid'.
3. Dropout з коефіцієнтом 0.3
4. Двонаправлений LSTM зі 256 нейронами. return_sequences=True
5. Dropout з коефіцієнтом 0.3
6. Flatten

Довжина послідовності збільшую з 25 до 50 нот.

Кількість параметрів 5,468,355.

Графік втрат зображений на рис.1.24.

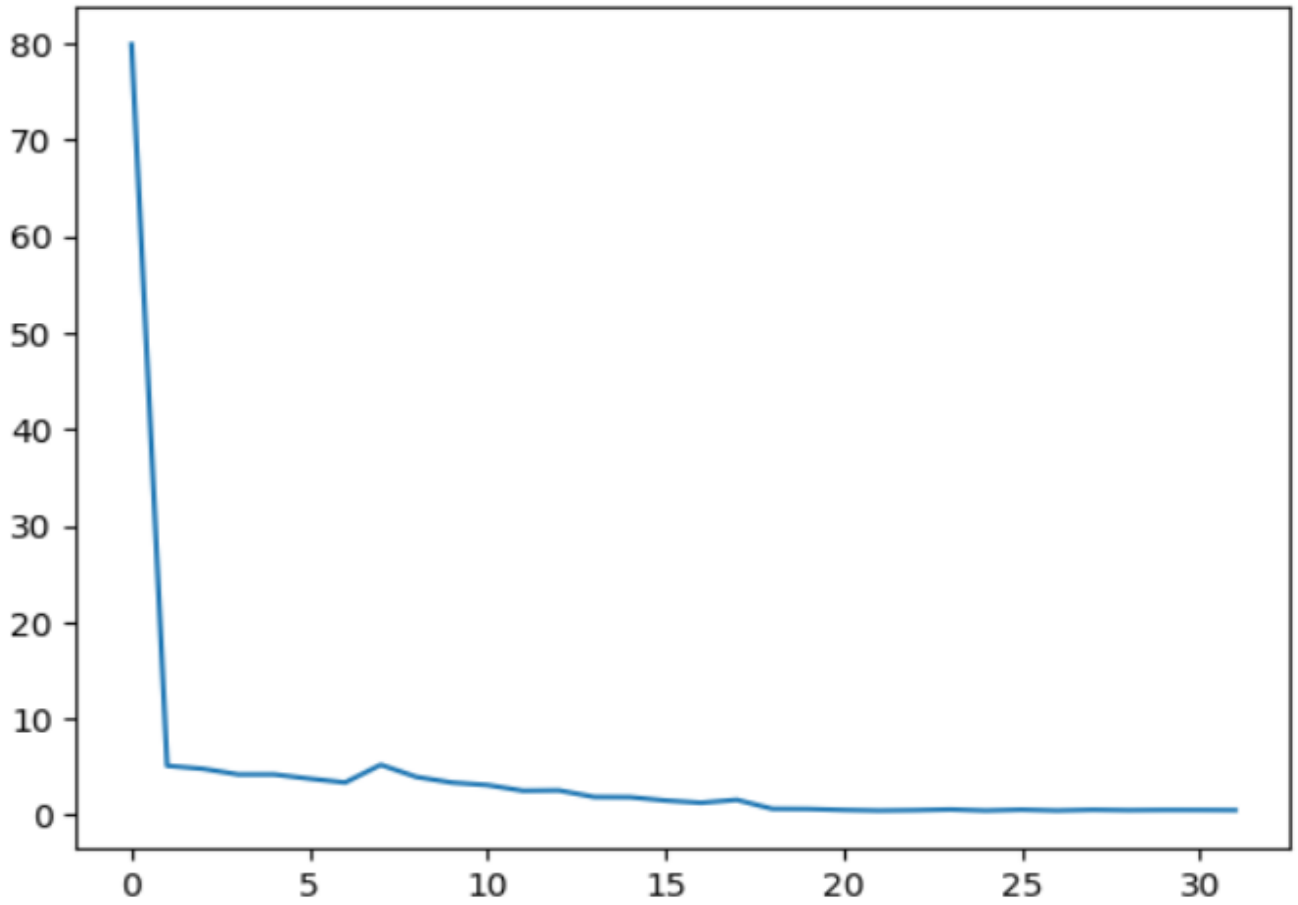


Рис.1.30 — Графік втрат для спроби 7

Це ще більше схоже на музику, але можна покращити ще.

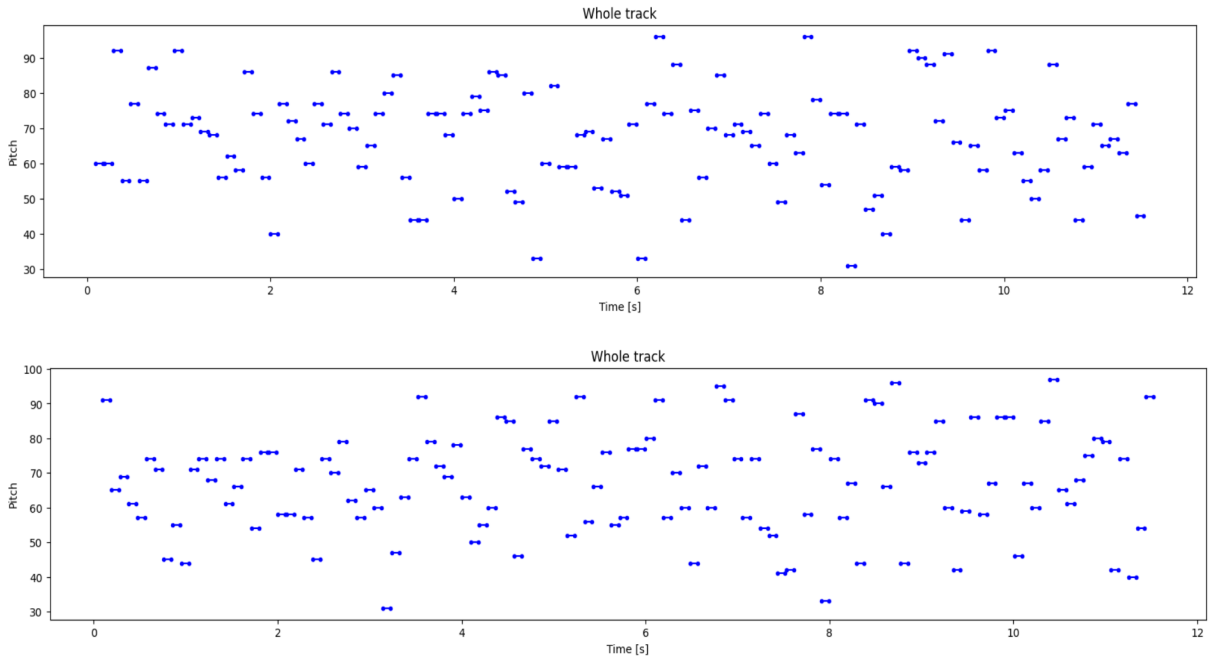


Рис.1.31 — Приклад семплів для спроби 7

Ітогова конфігурація моделі

Після безліч спроб з різними параметрами, включаючи кількість нейронів у кожному шарі, кількість шарів, коефіцієнти Dropout, довжину послідовності, швидкість навчання та ін, було зроблено висновок, що найкраще в задачі генерації музики себе показує модель з такою структурою:

1. Двонаправлений LSTM зі 256 нейронами. return_sequences=True.
2. Self Attention. attention_activation='sigmoid'.
3. Dropout з коефіцієнтом 0.3
4. Двонаправлений LSTM зі 256 нейронами. return_sequences=True
5. Dropout з коефіцієнтом 0.3
6. Flatten

При цьому довжину послідовності збільшую до сотні і навчаю модель вже на всьому наборі даних. З кожним поколінням модель демонструє себе дедалі краще, тому кількість поколінь можна збільшити до 200.

ВИСНОВКИ

Використання існуючих методів дослідження допомогло нам ретельно вивчити обрану тему та розробити нейронну мережу для генерації музичних зразків із заданою частотою ударів на хвилину.

Згенеровані семпли можуть бути цінним інструментом для музикантів і композиторів, які шукають нові ідеї і натхнення. Вони можуть використовувати ці семпли як початкову точку для створення своєї унікальної музики. Згенеровані мелодії, ритми та гармонії можуть послужити як творча база, яку вони можуть розширити, модифікувати та адаптувати до своїх власних потреб та стилів.

Крім того, згенеровані семпли можуть бути використані як засіб експериментування зі звуком і аранжуванням. Музиканти можуть використовувати їх для створення нових звукових ефектів, стилістичних поєднань та аранжування, що в результаті дасть цікаві та непередбачувані музичні рішення.

Незважаючи на те, що згенеровані семпли зараз ще не можуть конкурувати з роботами відомих музикантів, вони відкривають нові можливості для творчості та експериментів. З часом, з удосконаленням технологій та підвищенням якості генерації, можливо будуть створені семпли, які вражатимуть своєю якістю та унікальністю.

Таким чином, нейромережеві моделі в генерації музики відкривають нові горизонти для музичної творчості. Вони не тільки допомагають музикантам та композиторам знайти натхнення, але й стимулюють експериментацію, розширення стильового спектру та творчу самовираженість. Це заохочує розвиток музичної культури та сприяє появі нових, унікальних музичних творів.

Проблеми. Під час роботи над цим проектом, зустрівся ряд викликів та складнощів, які варто відзначити. Вибір оптимальних параметрів і створення архітектури мережі також є нетривіальною задачею, але цей етап стане набагато легше, якщо модель буде навчатися швидше, тобто можна буде зробити більше спроб настройок параметрів за той самий час. Але дві найбільші проблеми полягають у обмежених ресурсах та якістю датасету.

Обмежені ресурси. Обмежені ресурси можуть стати суттєвим викликом під час навчання великих нейромереж. Навіть при використанні графічного прискорювача, як от Google Colab, який значно прискорює процес порівняно з виконанням на локальній машині, можуть виникати труднощі. Навчання може займати кілька десятків годин, добу, чи навіть більше, що створює проблеми для довготривалих процесів, особливо на віддалених ресурсах.

Протягом тривалого навчання можуть виникати непередбачувані ситуації, такі як переривання зв'язку або скидання прогресу навчання, що призводить до втрати часу і зусиль. Ці проблеми можуть бути особливо значущими у випадку використання віддалених серверів або розподілених обчислень.

Закон Мура, сформульований Гордоном Муром, одним з засновників Intel, може пролити світло на перспективи подальшого розвитку обчислювальної потужності та вирішення проблем обмежених ресурсів. Згідно з цим законом, обчислювальна потужність подвоюється приблизно кожні два роки. Це означає, що з часом ресурси, доступні для навчання нейромереж, будуть ставати все потужнішими та доступнішими.

Закон Мура дозволяє передбачати, що з часом проблема обмежених ресурсів буде зменшуватися. Завдяки постійному зростанню обчислювальної потужності, а саме подвоєнню потужності процесорів кожні 18 місяців, можна очікувати, що процес навчання нейромереж стане швидшим та більш ефективним, навіть для великих моделей.

Однак, важливо зазначити, охоча закон Мура дійсно спостерігався впродовж багатьох років, він може зустрічати обмеження в майбутньому, оскільки фізичні обмеження можуть уповільнити подальше зростання обчислювальної потужності.

Тому, хоча закон Мура може пролити світло на проблему обмежених ресурсів у навчанні нейромереж, все ж необхідно постійно шукати нові підходи та оптимізації для ефективного використання доступних ресурсів та забезпечення успішного навчання моделей.

Датасет. Для розв'язання проблеми обмеженості датасетів та їх недостатньої репрезентативності, важливо продовжувати зусилля по їх створенню та покращенню. Велику роль може відігравати спільнота дослідників та ентузіастів, які можуть об'єднатися для спільного збору та анотування даних.

Покращення якості датасетів передбачає виявлення та виправлення наявних дефектів, таких як шум або інші артефакти. Додатковою перевагою може бути автоматизований аналіз та фільтрація даних, що дозволяє зменшити зусилля, необхідні для ручного видалення аномалій.

Нові технології, такі як методи підсиленого навчання (reinforcement learning) та глибоке навчання з посиленням (adversarial training), можуть використовуватися для створення синтетичних датасетів, що репрезентують більш широкий спектр музичних жанрів та стилів.

Крім того, спільна робота з музичними виконавцями, композиторами та експертами з музики може сприяти покращенню датасетів, оскільки їх досвід та знання можуть допомогти встановити більш точні музичні параметри та вимоги.

Збільшення якості та розміру доступних датасетів покращить ефективність роботи нейронних мереж у генерації музики, дозволяючи їм навчатися на більш репрезентативних та різноманітних даних. Це відкриє шлях до створення більш креативної та високоякісної музики за допомогою нейромережевих моделей.

Сейчас полученные семплы могут послужить вдохновением для музыкантов и композиторов.

Генерування семплів з визначеним bpm. На практиці, наша модель генерує MIDI-файли, які можуть бути відтворені з будь-яким темпом ударів на хвилину. Проте, музика звучить найкраще, коли темп, з якого вона була розроблена, збігається з початковим темпом (bpm). Тому, щоб отримати семпл із визначеним bpm, важливо подати на вхід моделі фрагмент музичного зразка з таким самим темпом. Маленька похибка у ± 5 ударів на хвилину не є критичною, оскільки ми все одно можемо відтворити семпл, прискоривши або сповільнивши його.

Існує ще один спосіб генерації семплів, і це використання послідовності, отриманої з іншого джерела, на вхід до моделі. Один з таких джерел – це VAE (Variational Autoencoder), який використовується для стохастичної генерації нових музичних зразків з варіацією, що дозволяє отримувати різноманітні та креативні результати.

Проте, при використанні VAE збільшується ймовірність того, що модель LSTM може згенерувати послідовність, яка не нагадує музику. Це може статися через недостатню потужність моделі або недостатню кількість навчальних даних. Щоб запобігти цьому, модель потребує значно більшої потужності та навчання на великій кількості даних. Важливо враховувати, що результати стохастичної генерації з використанням VAE можуть бути більш експериментальними та менш передбачуваними, що відкриває простір для творчості та нових музичних експериментів, однак саме через це погіршує можливість продовження зразка за допомогою LSTM.

Щоб покращити результат можна скористатись такими порадами:

1. Збільшити датасет. Чим більше даних для навчання, тим різноманітнішим і точнішою є нейромережа.
2. Поекспериментувати зі структурою моделі, можливо буде знайдений варіант краще представленого мною.
3. Комбінувати LSTM з іншими архітектурами нейромереж.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jean-Pierre Briot, Ga¨etan Hadjeres, Franc ̧ois-David Pachet Deep Learning Techniques for Music Generation https://www.researchgate.net/publication/333014972_Deep_Learning_Techniques_for_Music_Generation_-_A_Survey
2. Нейронные сети: краткая история триумфа https://www.youtube.com/watch?v=nV7cI5zgOpk&list=PLA0M1Bcd0w8yv0XGiF1wj
erjSZVSrYbjh
3. Connor Hayes. Neural Networks. https://www.academia.edu/41114435/Neural_Networks
4. Grant Sanderson. But what is a Neural Network? https://www.3blue1brown.com/lessons/neural-networks
5. Introduction to Recurrent Neural Network. https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/
6. Saul Dobilas. LSTM Recurrent Neural Networks — How to Teach a Network to Remember the Past. https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e
7. Piyush Arya. Music Generation Using LSTM and Its Comparison with Traditional Method https://www.academia.edu/98651186/Music_Generation_Using_LSTM_and_Its_Comparison_with_Traditional_Method
8. Music Generation with LSTMs. https://blog.paperspace.com/music-generation-with-lstms/
9. Gang Liu. Bidirectional LSTM with attention mechanism and convolutional layer for text classification https://www.sciencedirect.com/science/article/abs/pii/S0925231219301067
10. Rahul Modak. LSTM Based Music Generation System

https://www.academia.edu/53705014/LSTM_Based_Music_Generation_System
11. Raphael Abbou. DeepClassic: Music Generation with Neural Neural Networks
https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1204/reports/custom/report11.pdf
12. Nitish Srivastava. Dropout: A Simple Way to Prevent Neural Networks from Overfitting
https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf
13. Faqian Guan. A GAN Model With Self-attention Mechanism To Generate Multi-instruments Symbolic Music
https://ieeexplore.ieee.org/abstract/document/8852291/
14. Raghav Agrawal. A Comprehensive Guide on Neural Networks Performance Optimization
https://www.analyticsvidhya.com/blog/2021/09/a-comprehensive-guide-on-neural-networks-performance-optimization/
15. Zhaoyang Niu. A review on the attention mechanism of deep learning
https://www.sciencedirect.com/science/article/abs/pii/S092523122100477X
16. This is How Experts Predict the Future of AI.
https://www.analyticsvidhya.com/blog/2023/04/future-of-ai/

ДОДАТОК 1

Функція remove_initial_pauses

```python
def remove_initial_pauses(midi_data: pretty_midi.PrettyMIDI):
    first_note_start = float('inf')

    for instrument in midi_data.instruments:
        for note in instrument.notes:
            if note.start < first_note_start:
                first_note_start = note.start

    # print(first_note_start)
    for instrument in midi_data.instruments:
        for note in instrument.notes:
            note.start -= first_note_start
            note.end -= first_note_start

    current_duration = midi_data.get_end_time()
    new_duration = current_duration - first_note_start
    # print(f'Old duration: {current_duration}')
    # print(f'New duration: {new_duration}')

    return midi_data
```

ДОДАТОК 2

Функція create_sequeces

```python
def create_sequences(
    dataset: tf.data.Dataset,
    seq_length: int,
    vocab_size = 128,
) -> tf.data.Dataset:
  """Returns TF Dataset of sequence and label examples."""
  seq_length = seq_length+1

  # Take 1 extra for the labels
  windows = dataset.window(seq_length, shift=1, stride=1,
                           drop_remainder=True)

  # `flat_map` flattens the" dataset of datasets" into a dataset of tensors
  flatten = lambda x: x.batch(seq_length, drop_remainder=True)
  sequences = windows.flat_map(flatten)

  # Normalize note pitch
  def scale_pitch(x):
    x = x/[vocab_size,1.0,1.0]
    return x

  # Split the labels
  def split_labels(sequences):
    inputs = sequences[:-1]
    labels_dense = sequences[-1]
    labels = {key:labels_dense[i] for i,key in enumerate(key_order)}

    return scale_pitch(inputs), labels

  return sequences.map(split_labels, num_parallel_calls=tf.data.AUTOTUNE)
```