МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

ЭМУЛЯЦИЯ ЗВУКОВЫХ ЭФФЕКТОВ С ПОМОЩЬЮ ТЕХНОЛОГИЙ ГЛУБОКОГО ОБУЧЕНИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Рыданова Никиты Сергеевича

Научный руководитель
доцент _____ Б. А. Филиппов

Заведующий кафедрой
доцент, к. ф.-м. н. _____ С. В. Миронов

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последние десятилетия методы классического машинного обучения стали распространенной практикой в науке и прикладных проблемах, однако эти методы способны решать узкий спектр задач. Именно поэтому в последнее время получает развитие глубокое обучение — подраздел машинного обучения, предполагающий применение различных архитектур нейронных сетей.

Сегодня нейронные сети получают всё больше приложений в различных сферах деятельности — прогнозировании, классификации, регрессии, генерации и обработке изображений. Их широкое применение объясняется, в основном, универсальностью, адаптивностью и возможностью повторного использования для смежных задач.

Одной из наименее изученных областей применения нейронных сетей является обработка звука. Связано это, прежде всего, с большим объемом данных к обработке и наличием времени как одного из характеристик аудиосигнала. Между тем, нейронные сети в контексте обработки аудио потенциально могут иметь множество приложений, таких как: классификация, сегментация, исправление дефектов сигнала, обработка аудио и так далее.

Именно обучению с учителем в контексте обработки аудио и посвящена эта работа. Отметим, что под обработкой аудио можно понимать любые привычные операции над аудио (например, эквализация, компрессия или реверберация) или их комбинации. При этом, конкретная природа преобразования сигнала при решении задачи с помощью нейронных сетей не имеет значения — предполагается, что нейронные сети способны самостоятельно улавливать закономерности в данных.

Таким образом, не теряя общности, в качестве примера такого преобразования в рамках этой работы рассмотрим обработку снятого с гитары сигнала с помощью гитарных эффектов.

В последнее десятилетие технологии эмуляции гитарных эффектов становятся всё популярнее. Среди преимуществ использования цифровых эмуляторов можно выделить следующее:

1. их использование позволяет удешевить процесс производства музыкальных композиций;
2. обработка аудио не всегда воспроизводимый процесс, иными словами, не всегда возможно восстановить эффект с достаточной точностью даже при

наличии исходного аудио;

3. эмуляторы не требуют звукозаписывающего оборудования и легко интегрируются в т.н. цифровые звуковые рабочие станции (DAW).

Тем не менее, у классических эмуляторов есть и недостатки:

1. как правило, они программно реализуют аналоговые схемы, что дорого для разработки;
2. весь функционал плагинов-эмуляторов закладывается разработчиком и, соответственно, его дальнейшая модификация затруднена.

Эмуляторы на основе искусственных нейронных сетей способны решить эти проблемы. Кроме того, нейронные сети — хоть и дискретные структуры, но более гибкие, чем классические эмуляторы, что даёт преимущество в спектре возможных оттенков звучания.

Целью данной работы является изучение возможностей технологий глубокого обучения в контексте обработки аудио.

В ходе работы должны быть решены следующие задачи:

1. изучить способы цифрового представления аудио;
2. изучить основные концепции и понятия в машинном обучении;
3. выделить отличительные особенности методов глубокого обучения;
4. рассмотреть примеры архитектур нейронных сетей для обработки аудио;
5. изучить возможности библиотеки для глубокого обучения PyTorch;
6. разработать ПО для обучения и тестирования моделей на PyTorch;
7. представить несколько подходов к обучению с учителем в контексте обработки аудио;
8. применить описанные подходы и сравнить их возможности;
9. на основе полученных результатов создать приложение для обработки аудио в реальном времени.

1 Цифровое представление звука

Цифровое представление два звука имеет два очень важных свойства: дискретность по времени и амплитуде.

1.1 Дискретность по времени

Характеристикой дискретности сигнала по времени является *частота дискретизации*. Это число, характеризующее число измерений амплитуды сигнала в секунду времени. Иллюстрация представлена на рис. 1.1.

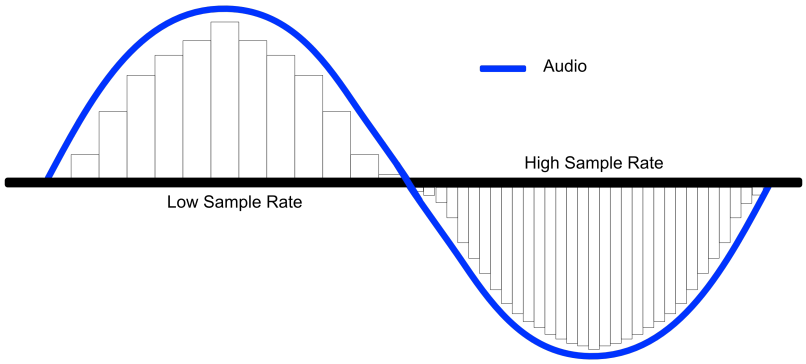

Рисунок 1.1 – Визуализация дискретизации сигнала при различных значениях её частоты

В каждый из дискретных моментов времени фиксируется амплитуда сигнала. Диапазон возможных значений амплитуды, при этом, определяется глубиной кодирования.

1.2 Дискретность по амплитуде

Глубина кодирования определяет диапазон возможных значений амплитуды сигнала. Непрерывные значения, не попадающие в сетку дискретизации, «выравниваются» к ближайшему значению в сетке.

Таким образом, аналоговый сигнал приближается сеткой, по одной из осей которых располагается время, а по другой — одно из возможных значений амплитуды, что проиллюстрировано на рис. 1.2.

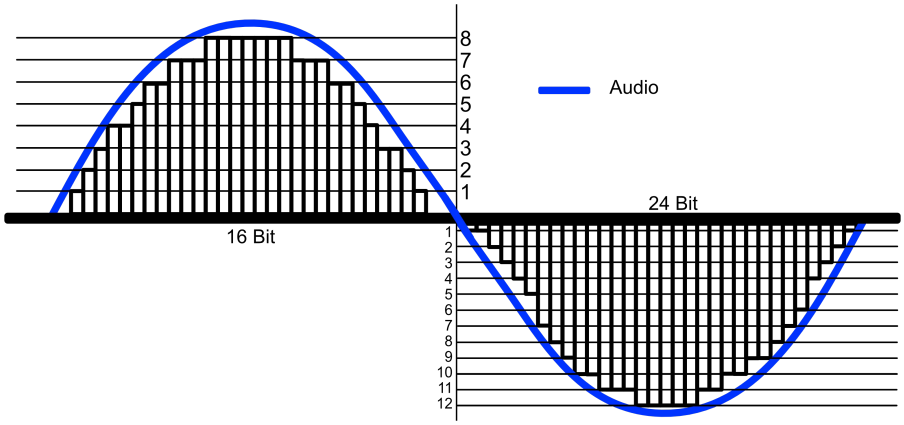

Рисунок 1.2 – Визуализация оцифровки аналогового сигнала при различных значениях глубины кодирования

1.3 Амплитудно-частотная характеристика сигнала

Иногда полезно перейти к амплитудно-частотной характеристике сигнала.

На практике для анализа оцифрованного сигнала для этого применяют *дискретное оконное преобразование Фурье*:

$$F(m,\omega) = \sum_{n=-\infty}^{\infty} f(n)w(n-m)e^{-j\omega n}$$

где $F(m,\omega)$ — результат дискретного преобразования Фурье над m-ым окном преобразования.

Стоит отметить, что такое преобразование является обратимым, причем даже в случае утери исходной фазы сигнала.

1.3.1 Мел-преобразование

У дискретного оконного преобразования Фурье есть недостаток — шкала амплитуды является линейной.

Компенсировать этот недостаток можно, применив нелинейное преобразование над частотной шкалой. По закону Вебера-Фехнера высота тона воспринимается пропорционально логарифму частоты, а значит в качестве нелинейного преобразования предпочтительно использовать именно логарифм. Именно в этом заключается идея мел-преобразования.

Несмотря на свои достоинства, мел-преобразование является трудно обратимым, поэтому не будет использовано для практической реализации в рамках данной работы.

2 Преобразование над аудиосигналом

В любом из представлений аудиосигналом является временным рядом, а значит все привычные инструменты для анализа временных рядов актуальны и в этом случае. В частности, применимы и рекуррентные нейронные сети.

2.1 Обработка на основе исходного сигнала

Важно отметить, что речь идет о генерации нового, обработанного аудио, что в свою очередь соответствует генерации нового временного ряда на основе исходного.

Реализовать это можно, например, возвращая не только финальное состояние h_n и C_n, но и промежуточные состояния (h_t, C_t) (см. рис. 2.1).

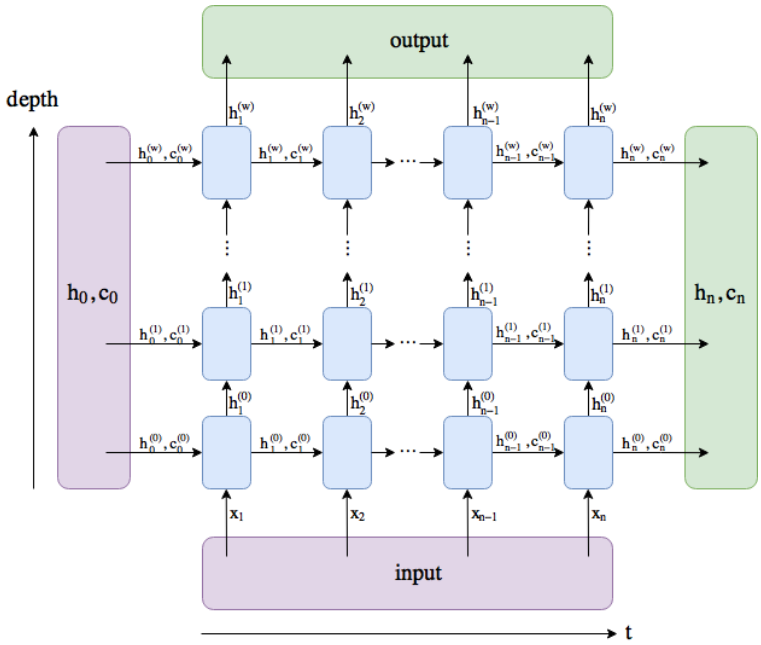

Рисунок 2.1 – Получение промежуточных состояний модели

после чего применить линейную трансформацию вида

$$output_t = \langle h_t, w \rangle + b$$

Для того, чтобы сохранить природу исходного сигнала, будем требовать от модели предсказывать лишь изменение исходного сигнала, что можно сделать, складывая вход с предсказанием модели.

Получим итоговое преобразование:

$$\hat{y}_t = input_t + output_t = input_t + \langle h_t, w \rangle + b$$

В режиме вычислений необходимо также сохранять финальное состояние, чтобы проинициализировать им модель при обработке следующего блока данных, чтобы избавиться от резких скачков амплитуды и, следовательно, избежать артефактов звучания.

2.2 Обработка на основе АЧХ сигнала

Основной проблемой при анализе сигнала является размер входных данных. Это отражается на объеме занимаемой при обучении памяти, скорости обучения модели и необходимых ресурсов для её использования. Вместо этого, можно работать с результатом оконного преобразования Фурье.

Как было отмечено в разделе 1.3, оконное преобразование Фурье обратимо, а значит после обработки сигнала в амплитудно-частотной характеристике с некоторой погрешностью возможно вернуться к амплитудно-временному представлению и сохранить получившийся файл.

В результате получим временной ряд, однако теперь вместо одного значения-признака — амплитуды, в каждый момент времени будем иметь дело с $\frac{n\_fft}{2} + 1$ значениями, соответствующим различным частотам в оконном преобразовании Фурье (см. рис.2.2).

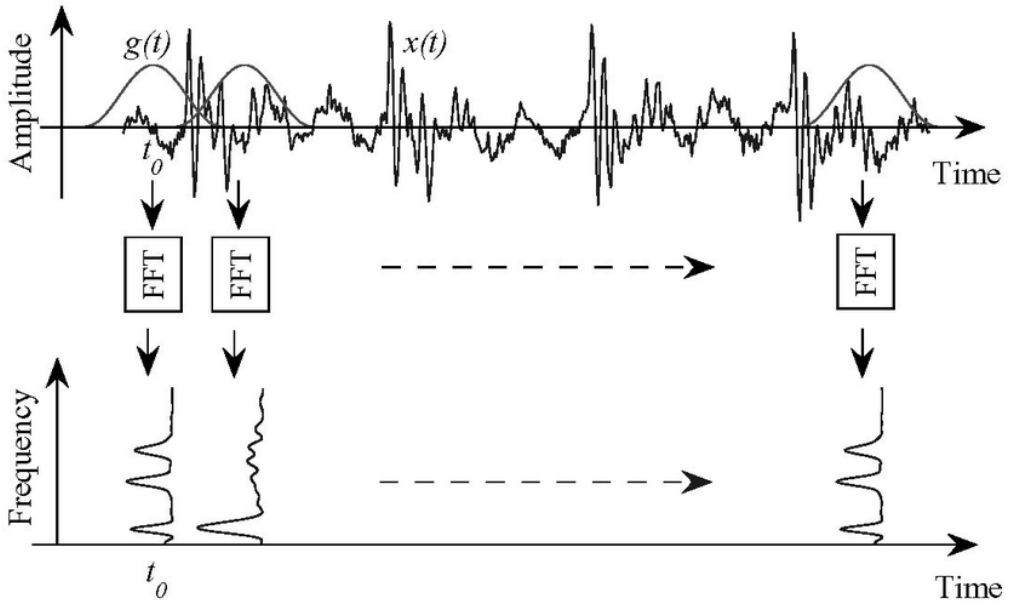

Рисунок 2.2 – Устройство оконного преобразования Фурье

3 Обучение моделей с помощью PyTorch

Для реализации процесса обучения моделей был выбран язык Python и библиотека для глубокого обучения PyTorch.

Преимуществами PyTorch над Tensorflow/Keras является более простой и универсальный процесс настройки под конкретную платформу, быстрая скорость работы, а также подробная документация.

3.1 Реализация ПО для обучения моделей

Процесс обучения нейронных сетей обычно включает в себя:

1. инициализацию интерфейса командной строки;
2. инициализацию процесса логирования;
3. инициализацию работу с GPU, если это возможно;
4. инициализацию состояние модели;
5. формирование данных для обучения;
6. инициализацию планировщика скорости обучения (scheduler);
7. запуск основного цикла обучения.

Зачастую, решая эти задачи на Python, разработчики прибегают к реализации в Jupyter Notebook с сопутствующим дублированием плохо структурируемого кода, что затрудняет его автоматическое размещение на вычислительных кластерах.

В рамках данной работы все эти шаги реализованы в виде единого программного продукта.

3.1.1 Интерфейс командной строки

Интерфейс командной строки реализован с помощью встроенной библиотеки `argparse`. Логика обработки аргументов командной строки представлена функцией `init_parser`, которая возвращает набор доступных аргументов в зависимости от выбранного режима — тренировки или вычислений.

3.1.2 Логирование

Логирование реализовано с помощью встроенной библиотеки `logging`. Логика инициализации логирования реализована функцией `init_logger`.

3.1.3 Инициализация работы с GPU

Важным критерием любого ПО для глубокого обучения является поддержка вычислений на графических ускорителях. В реализованном программном продукте эта возможность поддерживается, причем вне зависимости от производителя графической карты — будь то AMD, Intel или Apple.

Реализация этой логики представлена в функции `init_device`, которая возвращает объект типа `torch.device` в зависимости от полученного аргумента командной строки. Кроме того, поддерживается автоматическое обнаружение графического ускорителя через служебные функции библиотеки PyTorch, если вычисляющее устройство не задано явно.

3.1.4 Инициализация состояния модели

Выбор требуемой модели и инициализация её состояния реализуется через механизм рефлексии, предоставленный языком Python.

Предполагается, что список всех доступных к вызову моделей перечислен в модуле `models`, а вместе с именем модели передается конфигурационный файл с её необходимыми параметрами. В случае, если для выбранной модели реализованы все необходимые методы, гарантируется её корректная работа.

Кроме того, с помощью аргумента `--checkpoint` возможно восстановление значений обучаемых параметров модели из `.pt` файла.

3.1.5 Формирование данных для обучения

Поведение участка кода, ответственного за формирование данных для обучения, конечно, зависит от выбранной модели. Поэтому необходимо некоторое соответствие между моделью и объектом, реализующим загрузку данных. Это реализовано путем добавления специально метода `get_provider` для каждой из моделей — в таком случае результат вызова `get_provider` для полученного на предыдущем этапе объекта гарантировано вернет нужный класс-загрузчик.

Инициализация, как и на предыдущем шаге, осуществляется с помощью конфигурационного файла с необходимыми параметрами.

3.1.6 Инициализация планировщика скорости обучения

Инициализация планировщика осуществляется аналогично предыдущим двум шагам — получение необходимого класса-планировщика осуществляется

с помощью метода `get_scheduler`, а набор параметров получается из командной строки.

В рамках данной работы было реализовано два основных планировщика — с постоянным значением шага обучения и динамическим с помощью эвристики косинусного отжига.

3.1.7 Запуск основного цикла обучения

Наконец, в случае корректного выполнения предыдущих шагов возможен запуск основного цикла обучения, который выполняет следующие шаги:

1. переводит модель в обучающий режим;
2. делит данные на шаги, определяемые аргументом `batch_size`;
3. отправляет данные из загрузчика на вычисляющее устройство;
4. вычисляет предсказание модели и изменяет значение весов исходя из из значения градиента;
5. переводит модель в режим вычислений;
6. получает ошибку модели на валидационной выборке.

Эти шаги повторяются, пока не будет достигнуто необходимое число эпох обучения.

3.1.8 Сопровождение исходного кода

При разработке ПО рассматривалась возможность его дальнейшего сопровождения, поэтому все программные зависимости контролировались с помощью Poetry.

Кроме того, полученный продукт доступен для развертывания на любом устройстве с помощью технологии Docker, вне зависимости от операционной системы или архитектуры процессора.

4 Экспериментальная часть

В рамках данной работы была проведена серия экспериментов с целью выявления оптимальных гиперпараметров.

4.1 Подбор гиперпараметров обучения

4.1.1 Влияние шага обучения

Выбор шага обучения является одним из определяющих факторов, влияющих на итоговое качество модели. В рамках исследования были предложены несколько постоянных значений шага обучения: 0.01, 0.001 и 0.004. На основе этих экспериментов оптимальным значением оказалось значение, равное 0.004.

Однако было замечено, что выбор постоянного шага обучения негативно влияет на оптимизацию параметров модели на поздних стадиях обучения, поэтому была проверена эвристика косинусного отжига. Таким образом, удалось стабилизировать поведение оптимизатора на поздних этапах обучения.

4.1.2 Влияние объема набора данных

Не менее важным гиперпараметром является объем и разнообразие имеющихся данных, поскольку он влияет на способность модели к обобщению.

Было проведено два эксперимента: в первом модель обучалась фиксированное время $T = 15$ мин, во втором модель обучалась до сходимости.

Результаты экспериментов позволяют утверждать, что увеличение объема данных действительно повышает качество модели, но только при наличии достаточного времени для обучения до сходимости.

4.2 Подбор параметров модели

4.2.1 Влияние числа обучаемых параметров модели

Значение параметра `hidden_size` влияет на размерность выхода сети LSTM. Увеличение этого параметра влечет к увеличению общего числа обучаемых параметров, что очевидным образом влияет на общее качество модели.

Стоит отметить, что чрезмерное увеличение значения этого параметра потенциально может привести к переобучению, однако в данном случае этого не произошло.

5 Реализация вычислений в реальном времени

Большинство современные приложений для цифровых звуковых рабочих станций реализовано на языке C++. Это связано с его быстродействием и низкоуровневостью, что очень важно при вычислениях в реальном времени.

Для адаптации моделей к вычислению в реальном времени была выбрана библиотека с открытым исходным кодом RTNeural. Она предоставляет набор готовых классов, реализующих поведение наиболее популярных слоёв нейронных сетей, в том числе сетей LSTM.

Для интеграции приложения в DAW была выбрана библиотека с открытым исходным кодом для создания графических приложений JUCE. Она предоставляет возможность экспортирования приложения в основных форматах плагинов для обработки звука — VST3, AU, AAX и др.

Логически, JUCE разделяет приложение на две составных части — обработчик (processor) и редактор (editor). Обработчик реализует логику обработки аудио и представляется наследником базового класса `juce::AudioProcessor`. Редактор реализует интерфейс взаимодействия с обработчиком и представляется в виде наследника базового класса `juce::AudioProcessorEditor`.

Реализация предложенной ранее архитектуры с помощью RTNeural была использована в основном цикле обработки метода `processAbstractBlock` наследника базового класса-обработчика в JUCE.

ЗАКЛЮЧЕНИЕ

В ходе данной работы изучались возможности технологий глубокого обучения в контексте обработки аудио.

Для достижения поставленной цели в рамках работы:

1. были рассмотрены некоторые из существующих подходов к представлению аудиосигнала в цифровом виде;
2. изучены основные концепции и понятия в машинном обучении;
3. выделены особенности методов глубокого обучения;
4. рассмотрена архитектура рекуррентных нейронных сетей (в частности, LSTM) в качестве одной из возможных архитектур для обработки аудио;
5. изучены возможности библиотеки для глубокого обучения в PyTorch;
6. разработано ПО для обучения и тестирования моделей;
7. представлены два подхода к обработке аудио: на основе исходного сигнала и на основе амплитудно-частотной характеристики сигнала;
8. описанные подходы были применены и исследованы в ходе серии экспериментов;
9. полученные в рамках исследования результаты были учтены при реализации обработки аудио в реальном времени.

Таким образом, все поставленные в рамках работы задачи были выполнены.

Полученные в ходе работы результаты позволяют утверждать, что глубокое обучение в контексте обработки аудио является перспективным направлением исследований.

Подход к обработке аудио на основе исходного сигнала, несмотря на хороший результат, оказался вычислительно сложным как для обучения, так и для вычисления в режиме реального времени и требует оптимизации или пересмотра существующей архитектуры модели.

В то же время подход на основе амплитудно-частотной характеристики сигнала хоть и требует значительно меньше ресурсов при обучении, но нуждается в дальнейшем улучшении.

Тем не менее, стоит отметить, что имеющихся на текущий момент технических и вычислительных средств уже достаточно для создания универсальных обработчиков сигнала с помощью нейронных сетей.