МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет **ФИЗИЧЕСКИЙ**

Кафедра Автоматизации Физико-Технических Исследований

Направление подготовки **03.03.02 ФИЗИКА**

Образовательная программа: **БАКАЛАВРИАТ**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Ильин Иван Евгеньевич

(Фамилия, Имя, Отчество автора)

Тема работы Ускорение алгоритма генерации русскоязычной речи

«К защите допущен»

Заведующий кафедрой	**Научный руководитель**
ученая степень, звание	ученая степень, звание
должность, место работы	должность, место работы
……/…….. (фамилия И., О.) / (подпись, МП)	……/…….. (фамилия И., О.) / (подпись, МП)
«……»……20…г.	«……»……20…г.

Дата защиты: «……»……20…г.

Новосибирск, 2021

Ускорение алгоритма генерации русскоязычной речи

Ильин Иван Евгеньевич

Физический факультет. Выпускная Квалификационная Работа.

Группа №17345, 8 семестр, 2021 год.

Научный руководитель:

Александр Игоревич Гончаренко

Аннотация

Синтез речи является одной из важных задач в области обработки звука. Сейчас появляется всё больше голосовых ассистентов и колл-центров, для которых качество и скорость синтеза речи – определяющие факторы.

На сегодняшний день, в области синтеза речи доминирует подход генерации, основанный на нормализованных потоках (Normalizing Flow-based Synthesis), поскольку подобные нейронные сети способны генерировать речь, практически не отличающуюся от человеческой. Однако, типичные размеры таких моделей зачастую превышают тысячи мегабайт, что свидетельствует о высокой вычислительной сложности. Это в свою очередь затрудняет их работу на вычислительных устройствах, не обладающих таковой. Таким образом, ускорение нейронных сетей для генерации речи является актуальной научной задачей, что и является целью данной работы.

В качестве базового решения была выбрана современная генеративная архитектура, состоящая из энкодера Tacatron-2 и вокодера WaveGlow. В результате профилирования архитектуры, было выяснено, что самой медленной частью является вокодер WaveGlow. Для достижения наилучшего результата было принято решение модифицировать нейронную сеть на уровне архитектуры. В качестве метода оптимизации скорости и размеров сети выбран структурированный прунинг, который был применён к уже обученной модели с последующей тонкой настройкой.

В результате, удалось ускорить модель более чем в 1.5 раза на Nvidia GTX 1050-ti с улучшением качества (MOS: +0.18, WER: -1.8%) и в 2 раза без значительной потери качества (MOS: -0.29, WER: +2%).

Оглавление

1. Введение

В связи с развитием технологий телефонии и мобильной связи, стали актуальны колл-центры. Они необходимы как в бизнесе для поддержки пользователей и вовлечения новых клиентов, так и в государственных органах для оповещения людей и сбора статистики. Проблема колл-центров состоит в их высокой стоимости и сложной инфраструктуре: нужно не только иметь хорошую и устойчивую сеть, но и большое количество сотрудников, отвечающих на звонки.

Можно заменить ручной труд на машинный, ведь уже сейчас есть хорошие решения, основанные на глубоких нейронных сетях, позволяющие обрабатывать голос, генерировать ответ и синтезировать речь. Преимущество автоматизированной системы состоит в том, что графические процессоры не требуют дорогостоящего обслуживания и, в отличии от реальных сотрудников колл-центра, могут работать круглосуточно, без перерывов.

Проблема в том, что на данный момент такой подход имеет значительный недостаток: процесс синтеза речи слишком медленный, а сам алгоритм может работать только на мощных видеокартах, например, tesla v100 [4], стоимость которых – сотни тысяч рублей. Учитывая тот факт, что колл-центр должен иметь возможность обслуживать одновременно несколько клиентов, суммарная стоимость оборудования оказывается непозволительно высокой даже для больших организаций.

Если ускорить работу алгоритма так, чтобы можно было запускать его на слабых и дешёвых графических процессорах, например Nvidia GTX 1050-ti, то это не только сэкономит много средств больших и обеспеченных организаций, но и позволит маленьким компаниям открыть свой колл-центр, работающий на дешёвом оборудовании.

Помимо колл-центров, синтез речи также используется для создания голосовых ассистентов. Мощные графические процессоры имеют слишком высокую стоимость и большой размер, поэтому некоторые возможные ответы заранее синтезируются, а потом используются при генерации ответа.

Однако в данном случае возникает проблема комбинаторного взрыва, когда становится невозможным заготовить речь на все ответы ассистента. Такой подход ухудшает качество и эффективность голосовых ассистентов, поэтому ускорение алгоритма синтеза речи является очень важной задачей.

Методы ускорения основаны на идее того, что нейронные сети – избыточные решения, которые можно и нужно упрощать. При этом стоит помнить, что при их ускорении не всегда удаётся улучшить качество или хотя бы оставить его на прежнем уровне. Поэтому важно не только ускорить модель, но и протестировать её в реальных условиях. Если оказывается, что качество значительно ухудшилось, то нужно проводить больше вычислительных экспериментов и, возможно, пробовать другие методы ускорения.

В данной работе использовался **прунинг** как основной метод для ускорения.

Прунинг – это отсечение малозначимых частей нейронной сети. После этой процедуры обычно производят обучение модели для тонкой настройки модели.

Ускоренная архитектура сети для синтеза речи может быть эффективно использована и в других областях, где прямое взаимодействие с человеком не обязательно. Например, для озвучки подкастов, книг, радиоэфиров или сценариев к видеоматериалам.

2. Обзор предметной области

Автоматизированная система интерактивного взаимодействия выглядит следующим образом (рис. 1).

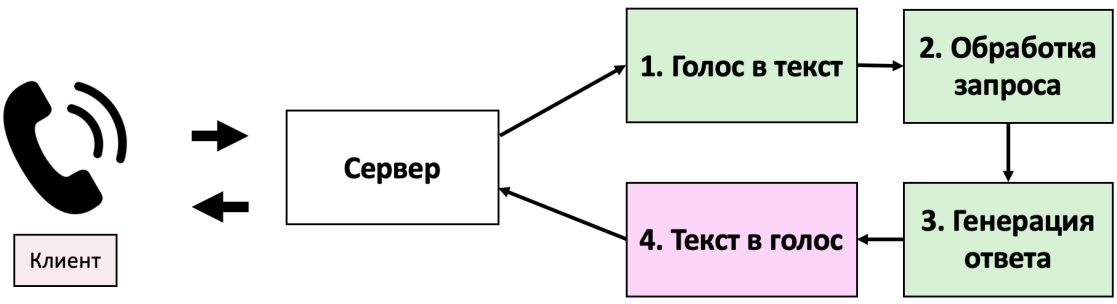

Рисунок 1. Работа интерактивной системы связи.

Клиент отправляет запрос серверу в виде аудиофайла, который переводится в текст и обрабатывается. После этого происходит генерация ответа и синтез речи. Важно отметить тот факт, что в современных системах нежелательно использование примитивных моделей синтеза речи ввиду того, что человек быстро распознаёт робота и перестаёт верить, что ему помогут решить проблему (это особенно актуально для автоматизированных колл-центров). Намного приятнее слышать натуральную человеческую речь даже при общении с виртуальным ассистентом. Поэтому важно использование самых лучших по качеству синтеза моделей генерации речи.
Самой вычислительно сложной частью является **синтез речи** (рис. 2).

Действие	Запрос-ответ серверу	1. Голос в текст	2. Обработка	3. Генерация ответа	**4. Текст в голос**
Длительность	~0.1 сек	~0.1 сек	~0.01 сек	~0.01 сек	~3 сек

Рисунок 2. Этапы работы интерактивной системы связи.

В качестве модели для распознавания голоса использовалась silero STT [19], для обработки и генерации ответа – решающие деревья, а для синтеза речи – Tacatron + WaveGlow [3], [4], [5]. Измерения произведены на Nvidia GTX 1050-ti.

Из таблицы видно, что, по сути, длительность синтеза речи определяет длительность ответа системы интерактивного взаимодействия. Это говорит о том, что стоит обратить большое внимание на **ускорение алгоритма генерации речи**.

На сегодняшний день можно выделить два способа генерировать речь. Первый – использование конкатетивных моделей [1], [2], а второй – параметрических [3], [4], [5].

У конкатетивных моделей на вход подаётся текст, который разбивается на синтаксические единицы. Для каждой такой единицы ищется соответствующий отрывок звука из большой базы дикторской озвучки. Потом эти кусочки склеиваются и получается готовый результат (рис. 3).

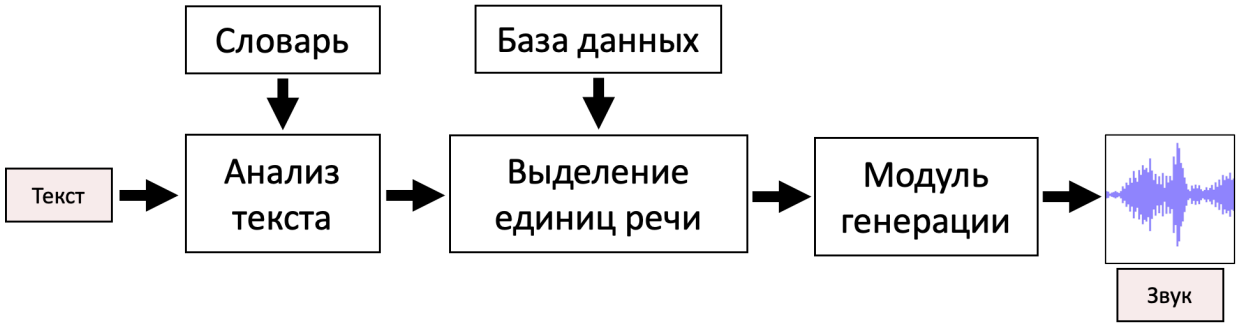

Рисунок 3. Синтез речи unit selection методов.

Работают такие методы очень быстро, но имеют ряд недостатков: такие модели требуют достаточно большой тренировочной базы аудиоданных для покрытия всевозможных контекстов, синтезируемая речь монотонна и не содержит эмоций, а также слышны характерные склейки звуков.

К счастью, существуют более продвинутые, параметрические методы синтеза [3], [4], [5], которые будут использованы в данной работе.

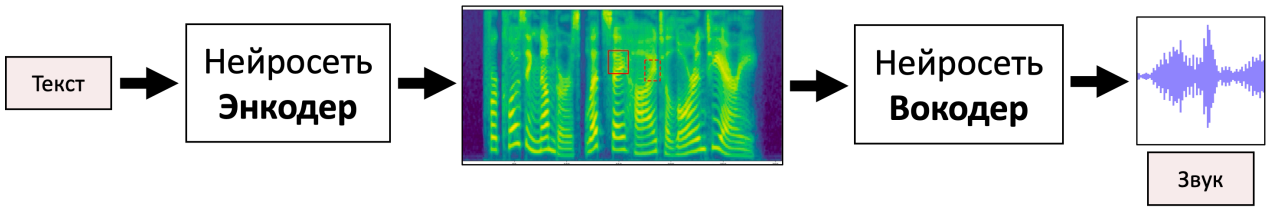

Рисунок 4. синтез речи параметрических методов.

Синтез происходит в два этапа (рис. 4): на первом этапе глубокая нейронная сеть – энкодер генерирует спектр, который подаётся на вход второй нейросети – вокодеру, выдающей готовый результат. Получается куда более естественное и плавное звучание, большее разнообразие в интонациях, а также, что не маловажно, для обучения требуется значительно меньшая база дикторской речи, поскольку при обучении нет необходимости выделять контекст в явном виде.

Параметрический синтез используется сейчас в реальных проектах крупных компаний, но большой их недостаток состоит в том, что они имеют крайне низкую скорость работы по сравнению с unit selection методами [1], [2]. Для того, чтобы понять, какую модель взять за основу, было проведено исследование предметной области.

2.1. Tacatron2 + WaveNet

Параметрические модели начинались с Tacatron и WaveNet от компании Google [3], [4], [5].

Звуковая волна – это большой массив амплитуд для каждого момента времени. Присоединённая вероятность звуковой волне иметь распределение $x = \{x_1, \dots, x_T\}$ может быть получена произведением условных вероятностей:

$$p(\boldsymbol{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, \dots, x_{t-1})$$

Каждое значение амплитуды x_t обусловлено всеми предыдущими значениями $x_1, \dots, x_{t-1}$.

Как и в PixelCNNs [6], распределение условных вероятностей моделируется применением нескольких последовательных свёрточных слоёв.

Слои пулинга не используются и размерность выхода сети равна размерности входа. Модель генерирует категориальное распределение на следующем шаге x_t с слоем softmax:

$$\sigma(x_t) = \frac{e^{x_t}}{\sum_{k=1}^{k=T} e^{x_k}}$$

и оптимизирует логарифм вероятности:

$$\log p(x) = \sum_{t=1}^{T} \log p(x_t \mid x_1, \dots, x_{t-1})$$

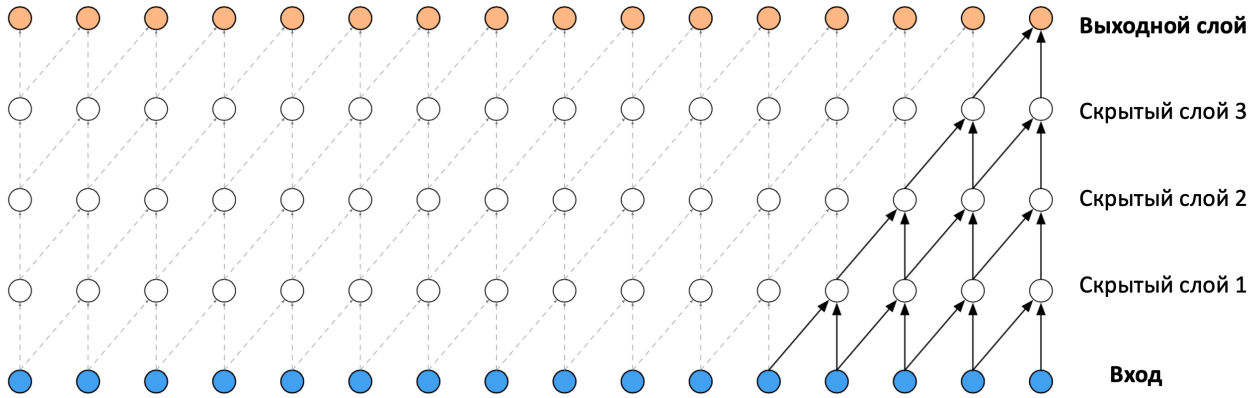

Рисунок 5. Последовательные свёрточные слои.

Основная идея WaveNet [4] заключается в использовании последовательных свёрток. Используя последовательные свёрточные слои, можно быть уверенным в том, что модель генерирует данные, не нарушая временной порядок: предсказание $p(x_{t+1} \mid x_1, \dots, x_t)$, полученное моделью на шаге t, не зависит от каких либо данных на последующих шагах $x_{t+1}, x_{t+2}, \dots, x_T$ (рис. 5).

Во время обучения, условные вероятности для всех временных шагов могут быть получены параллельно так как разметка данных $\boldsymbol{x}$ на всех шагах заранее известна. С другой стороны, во время работы сети все предсказания требуют последовательных вычислений: предсказание очередного шага, подаётся в начало сети для вычисления следующего.

Поскольку модели с условными свёрточными слоями не имеют рекуррентных соединений, они обучаются быстрее RNNs [6], особенно в случае длинных последовательностей. Однако их проблема состоит в том, что они требуют использования большого количества слоёв для увеличения области чувствительности к входным данным. Например, на рис. 5 область чувствительности равна 5 (= количество слоёв + размер фильтра - 1). Поэтому WaveNet использует расширенные (dilated) свёртки вместо обычных для увеличения области чувствительности на порядок без больших затрат на вычисления.

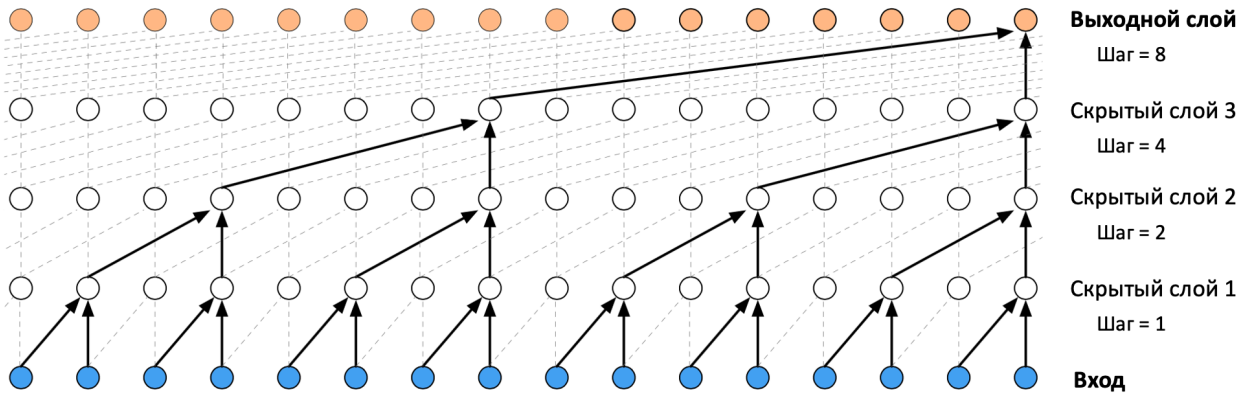

Рисунок 6. Принцип работы расширенных свёрточных слоёв.

Расширенные свёртка – это свёртка, в которой фильтр примеряется к области, большей его длины при помощи пропуска значений с определённым

шагом. Это аналог свёртки с большим фильтром, в котором на определённых местах стоят нули. На рис. 6 изображён принцип действия расширенных свёрточных слоёв с шагом 1, 2, 4 и 8.

Расширенные свёрточные слои позволяют значительно увеличить область чувствительности при использовании лишь нескольких слоёв, что эффективно ускоряет вычисления. WaveNet удваивает шаг пропуска в два раза на каждом слое до лимита, а затем повторяет сначала: т.е.

$$1, 2, 4, \dots, 512, 1, 2, 4, \dots, 512, 1, 2, 4, \dots, 512.$$

Используется та же функция активации, что и в PixelCNN [6]:

$$\boldsymbol{z} = \tanh(W_{f,k} * \boldsymbol{x}) \odot \sigma(W_{g,k} * \boldsymbol{x}),$$

где * отвечает за операцию свёртки, ⊙ - поэлементное умножение векторов, $\sigma(\cdot)$ – функция сигмойда, k – индекс слоя, f и g – обозначение filter и gate соответственно. W – это обучаемый слой свёртки.

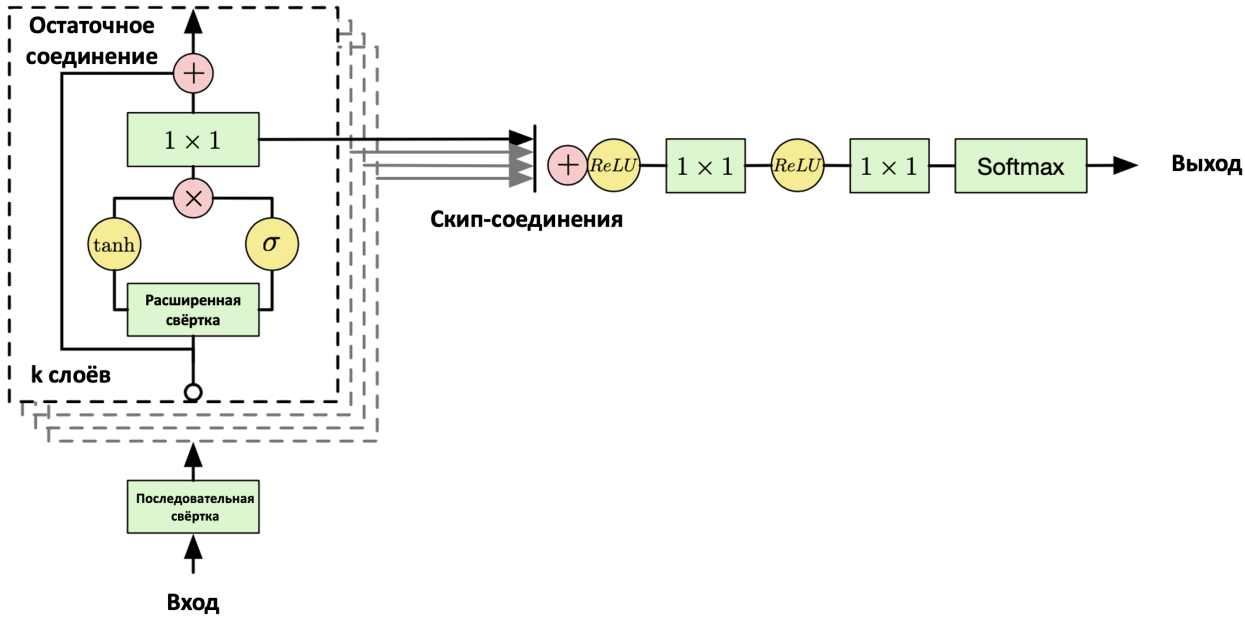

Рисунок 7. Остаточный блок и архитектура.

Остаточные соединения и параметризированные скип-соединения используются для ускорения сходимости и обучения более глубоких моделей. На рис. 7 показан остаточный блок WaveNet, который повторяется в сети много раз.

Образцы звуков	Английский MOS	Китайский MOS
LSTM-RNN parametric	3.67 ± 0.098	3.79 ± 0.084
HMM-driven concatenative	3.86 ± 0.137	3.47 ± 0.108
WaveNet (L+F)	4.21 ± 0.081	4.08 ± 0.085
Диктор (8-bit μ-law)	4.46 ± 0.067	4.25 ± 0.082
Диктор (16-bit linear PCM)	4.55 ± 0.075	4.21 ± 0.071

Таблица 1. Сравнение MOS для WaveNet.

Эксперименты показали, что данная архитектура может быть успешно применена для синтеза натуральной и естественной речи (таблица 1): Однако такое решение не может быть использовано в интерактивных системах ввиду непозволительно больших затрат на синтез одной секунды речи (может доходить до нескольких минут).

2.2. Tacatron2 + WaveRNN

Свёрточные нейронные сети типа WaveNet [4] требуют большого количества ресурсов для вычислений. WaveRNN [9] – это попытка ускорить процесс нелинейного преобразования распределения, используя преимущества рекуррентных нейронных сетей (RNN) [6]. В качестве основы была взята архитектура управляемых рекуррентных блоков (GRU) (рис. 8).

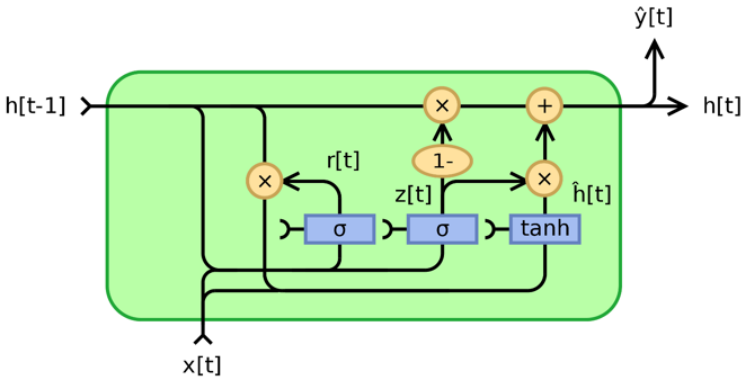

Рисунок 8. Архитектура управляемого рекуррентного блока RNN (GRU).

GRU цепочка была адаптирована таким образом, чтобы все необходимые нелинейные преобразования выполнялись за один раз. Внутреннее состояние RNN разделено на две части: 8 верхних битов c_t и 8 нижних битов f_t (от 16

битного аудио сигнала). Выход 8 нижних битов идёт в паре к 8 верхним битам. Оказывается, что разделение на две части работает лучше, чем работа сразу с 16 битами.

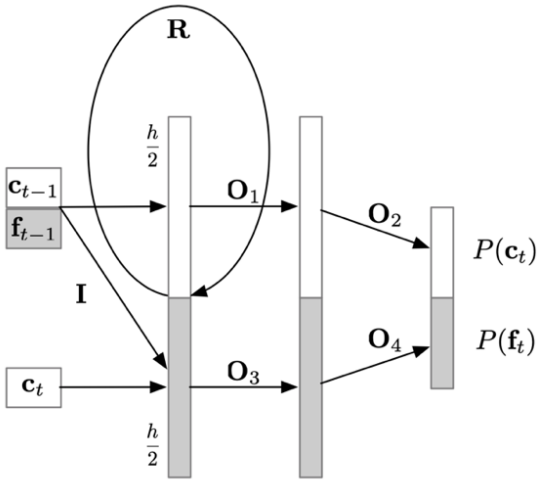

Рисунок 9. Архитектура WaveRNN.

На рис. 9 изображена изменённая структура GRU блока.

Внутри происходят следующие вычисления:

$$\boldsymbol{x}_t = [\boldsymbol{c}_{t-1}, \boldsymbol{f}_{t-1}, \boldsymbol{c}_t] \quad (1)$$

$$\boldsymbol{u}_t = \sigma(\boldsymbol{R}_u \boldsymbol{h}_{t-1} + \boldsymbol{I}_u^{\star} \boldsymbol{x}_t) \quad (2)$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{R}_r \boldsymbol{h}_{t-1} + \boldsymbol{I}_r^{\star} \boldsymbol{x}_t) \quad (3)$$

$$\boldsymbol{e}_t = \tau(\boldsymbol{r}_r \circ (\boldsymbol{R}_e \boldsymbol{h}_{t-1}) + \boldsymbol{I}_e^{\star} \boldsymbol{x}_t) \quad (4)$$

$$\boldsymbol{h}_t = \boldsymbol{u}_t \circ \boldsymbol{h}_{t-1} + (1 - \boldsymbol{u}_t) \circ \boldsymbol{e}_t \quad (5)$$

$$\boldsymbol{y}_c, \boldsymbol{y}_f = split(\boldsymbol{h}_t) \quad (6)$$

$$P(\boldsymbol{c}_t) = softmax\big(\mathbf{O}_2\, relu(\mathbf{O}_1 \boldsymbol{y}_c)\big) \quad (7)$$

$$P(\boldsymbol{f}_t) = softmax\big(\mathbf{O}_4\, relu(\mathbf{O}_3 \boldsymbol{y}_c)\big), \quad (8)$$

где $\star$ обозначение для матрицы, в которой последний выход первых восьми бит $\boldsymbol{c}_t$ соединён только с состояниями $\boldsymbol{u}_t, \boldsymbol{r}_t, \boldsymbol{e}_t$ и $\boldsymbol{h}_t$, которые определяют выход $\boldsymbol{y}_f$. Матрица $\boldsymbol{R}$ сформирована из матриц $\boldsymbol{R}_u$, $\boldsymbol{R}_r, \boldsymbol{R}_e$ и вычисляется матрично-векторным умножением для дальнейшего вычисления $\boldsymbol{u}_t, \boldsymbol{r}_t$ и $\boldsymbol{e}_t$. σ и τ – функция сигмойда и tanh.

WaveRNN работает быстрее оригинальной WaveNet, но все ещё имеет низкую скорость работы. По результатам опросов было получено, что

качество синтезируемой речи ниже, чем у оригинальной архитектуры WaveNet (таблица 2).

Образцы звуков	Английский MOS
WaveNet	4.51 ± 0.08
WaveRNN 224	3.73 ± 0.09
WaveRNN 384	4.23 ± 0.09
WaveRNN 896	4.37 ± 0.07

Таблица 2. Сравнение MOS для WaveRNN.

2.3. Normalizing Flow based модели синтеза

На сегодняшний день, в области синтеза речи доминирует подход генерации, основанный на **нормализованных потоках** (Normalizing Flow-based Synthesis) [7], [8], [10], [11], поскольку подобные нейронные сети способны генерировать речь, практически не отличающуюся от человеческой. Такой подход позволяет использовать единообразную и простую функцию потерь.

Особенность flow based моделей состоит в том, что, в отличии от обычных нейронных сетей, их преобразования должны быть обратимыми.

Пусть дана переменная z и её распределение известно: $z \sim \pi(z)$. Переменная x получена помощью обратимой функции f: $x = f(z)$. Неизвестное распределение новой переменной, $p(x)$ можно найти следующим образом:

$$\int p(x)dx = \int \pi(z)dz = 1$$

$$p(x) = \pi(z)\left|\frac{dz}{dx}\right| = \pi\left(f^{-1}(x)\right)\left|\frac{df^{-1}}{dx}\right| = \pi(f^{-1}(x))|(f^{-1})'(x)|$$

Для многомерного распределения имеем похожую форму:

$$p(\boldsymbol{x}) = \pi(\boldsymbol{z})\left|\det\frac{d\boldsymbol{z}}{d\boldsymbol{x}}\right| = \pi\left(f^{-1}(\boldsymbol{x})\right)\left|\det\frac{df^{-1}}{d\boldsymbol{x}}\right|$$

Распределения вероятностей изменяются цепочкой обратимых преобразований: из простого распределения (например, распределение Гаусса) получается сложное, которое хорошо имитирует реальный человеческий голос.

При обучении эти операции производятся в обратную сторону. То есть модель учится получать простое распределение, начиная с готовой звуковой волны. Это позволяет напрямую минимизировать логарифм вероятности простого распределения.

Процесс хорошо иллюстрирует следующая схема (рис. 10):

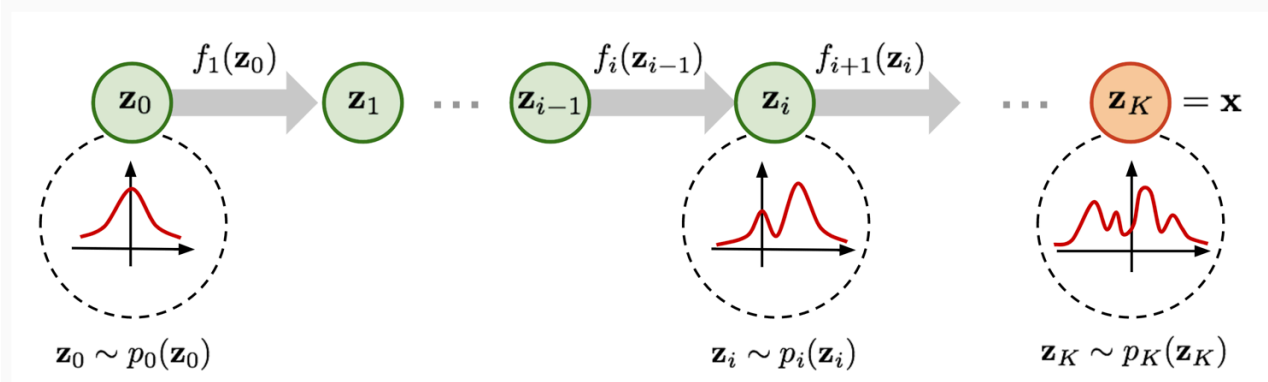

Рисунок 10. Последовательные обратимые преобразования flow based моделей.

Как показано на рис. 10:

$$\mathbf{z}_{i-1} \sim p_{i-1}(\mathbf{z}_{i-1})$$

$$\mathbf{z}_i = f_i(\mathbf{z}_{i-1}), \text{следовательно } \mathbf{z}_{i-1} = f_i^{-1}(\mathbf{z}_i)$$

$$p_i(\mathbf{z}_i) = p_{i-1}\left(f_i^{-1}(\mathbf{z}_i)\right)\left|\det\frac{df_i^{-1}}{d\mathbf{z}_i}\right| \quad (1)$$

Изменим уравнение таким образом, чтобы правая часть (1) была функцией от z_{i-1}. Это позволит оценить логарифм вероятности сложного распределения z_i с помощью простого z_{i-1} и обратной функции:

$$p_i(\mathbf{z}_i) = p_{i-1}(f_i^{-1}(\mathbf{z}_i))\left|\det\frac{df_i^{-1}}{d\mathbf{z}_i}\right|$$

$$= p_{i-1}(\mathbf{z}_{i-1})\left|\det\left(\frac{df_i}{d\mathbf{z}_{i-1}}\right)^{-1}\right| \quad (2)$$

$$= p_{i-1}(\mathbf{z}_{i-1})\left|\det\frac{df_i}{d\mathbf{z}_{i-1}}\right|^{-1}$$

$$\log p_i(\boldsymbol{z}_i) = \log p_{i-1}(\boldsymbol{z}_{i-1}) - \log \left| \det \frac{df_i}{d\boldsymbol{z}_{i-1}} \right|$$

Переход в (2) возможен на основании теоремы об обратной функции:

$$\frac{df^{-1}(y)}{dy} = \frac{dx}{dy} = \left(\frac{dy}{dx}\right)^{-1} = \left(\frac{df(x)}{dx}\right)^{-1}$$

В случае, когда имеется цепочка таких преобразований:

$$x = z_k = f_K \circ f_{K-1} \circ \ldots \circ f_1(\boldsymbol{z}_0)$$

$$\log p(x) = \log \pi_K(z_K) = \log \pi_{K-1}(z_{K-1}) - \log \left| det \frac{df_K}{dz_{K-1}} \right|$$

$$= \log \pi_{K-2}(z_{K-2}) - \log \left| det \frac{df_{K-1}}{dz_{K-2}} \right| - \log \left| det \frac{df_K}{dz_{K-1}} \right|$$

$$= \cdots$$

$$= \log \pi_0(z_0) - \sum_{i=1}^{K} \log \left| det \frac{df_i}{dz_{i-1}} \right|$$

Процесс преобразований переменной $z_i = f_i(z_{i-1})$ называется **потоком** (flow), а цепочка последовательных преобразований распределений π_i называется **нормализованным потоком** (normalizing flow). Для функций f_i необходимо выполнение двух условий:

1. Она легко обратима
2. Её Якобиан легко посчитать

2.3.1. Tacatron2 + Parallel WaveNet

Архитектура WaveNet [4] позволяет выполнять быструю тренировку, но синтез речи всё ещё достаточно медленный. Обратимые авто регрессивные flow модели (Inverse-autoregressive flows – IAFs) [7] устроены так, что все элементы могут быть сгенерированы параллельно. IAFs моделируют x_t помощью $p(x_t|\boldsymbol{z}_{\leq t})$ то есть $x_t = f(\boldsymbol{z}_{\leq t})$. Преобразование имеет диагональный якобиан, а значит детерминант равен произведению диагональных элементов:

$$\log \left| \frac{d\boldsymbol{x}}{d\boldsymbol{z}} \right| = \sum_t \log \frac{\partial f(\boldsymbol{z}_{\leq t})}{\partial \boldsymbol{z}_{\leq t}}.$$

Сначала имеется простое распределение $z \sim Logistic(0, 1)$, а после применяется преобразование:

$$x_t = z_t \cdot s(\boldsymbol{z}_{\leq t}, \theta) + \mu(\boldsymbol{z}_{\leq t}, \theta)$$

Модель генерирует $\boldsymbol{x}$, $\boldsymbol{\mu}$ и $\boldsymbol{s}$. То есть $p(x_t|\boldsymbol{z}_{\leq t})$ – это логистическое распределение с параметрами μ_t и s_t.

$$p(x_t|\boldsymbol{z}_{\leq t}, \theta) = \mathbb{L}\big(x_t\big|\mu(\boldsymbol{z}_{\leq t}, \theta), s(\boldsymbol{z}_{\leq t}, \theta)\big),$$

где $\mu(\boldsymbol{z}_{\leq t}, \theta)$ и $s(\boldsymbol{z}_{\leq t}, \theta)$ могут быть любыми авторегрессионными моделями.

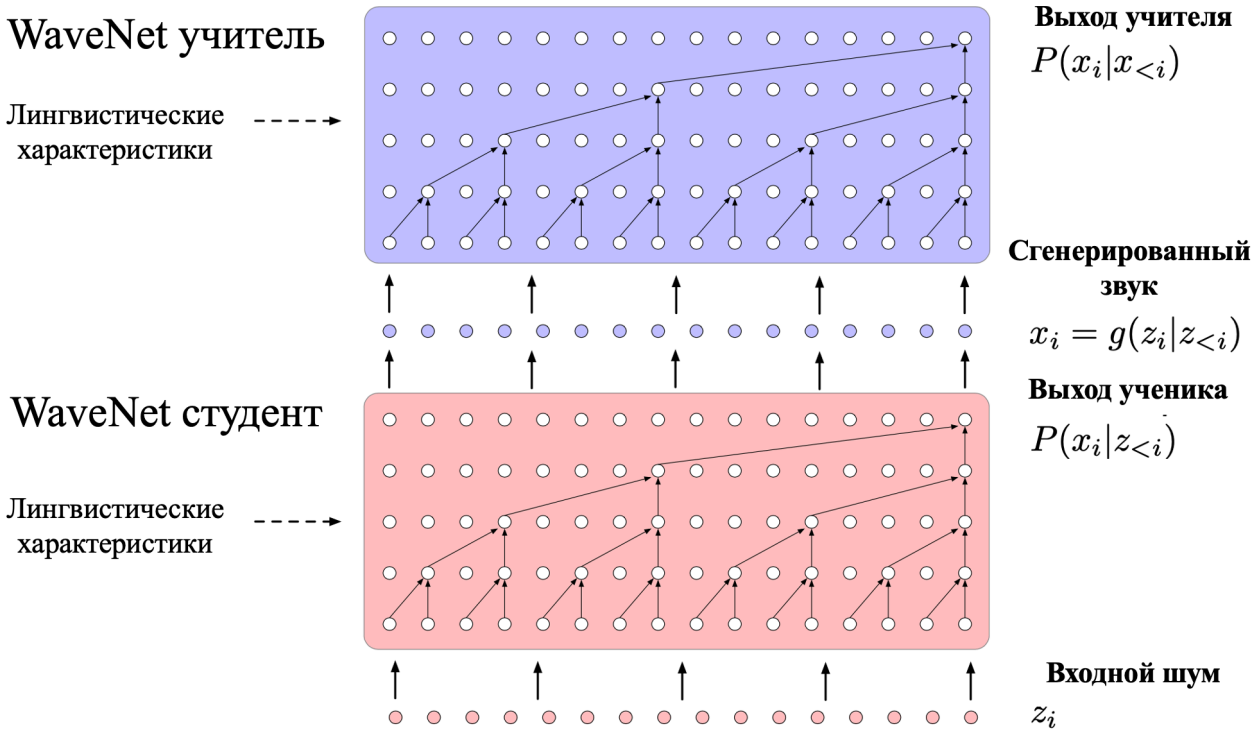

Рисунок 11. Схема обучения Parallel WaveNet.

Normalising flows модели требуют применения нескольких итераций для того, чтобы из шума сгенерировать связный звук. Выход каждой цепочки IAFs идёт на вход следующей итерации. Используются те же свёрточные авторегрессионные модели, что и в WaveNet. Первый слой сети берет шум из логистического распределения $x^0 = z$, а после преобразования передаёт результат второму слою:

$$x^i = x^{i-1} \cdot s^i + \mu^i$$

Так как такой порядок операций происходит на всех слоях, финальное распределение $p(x_t|\boldsymbol{z}_{\leq t}, \theta)$ – логистическое с параметрами:

$$\mu_{tot} = \sum_{i}^{N} \mu^i \left(\prod_{j>i}^{N} s^j \right)$$

$$s_{tot} = \prod_{i}^{N} s_i$$

где N – это число слоёв.

Для обучения используется претренированная модель WaveNet в качестве учителя и IAF Parallel WaveNet [8] в качестве ученика (рис. 11). Имея parallel WaveNet (студента) $p_S(x)$ и WaveNet (учителя) $p_T(x)$, можно определить функцию потерь следующим образом:

$$D_{KL}(P_S||P_T) = H(P_S, P_T) - H(P_S),$$

где D_{KL} – Расстояние Кульбака — Лейблера, $H(P_S, P_T)$ – кросс энтропия между студентом P_S и учителем P_T. $H(P_S)$ – энтропия распределения студента. Когда расстояние Кульбака — Лейблера станет равным нулю, модель ученик будет иметь распределение модели учителя.

Было получено, что обученная таким образом модель имеет приемлемое качество (таблица 3) при существенном ускорении (более чем в 1000 раз по сравнению с оригинальной моделью WaveNet).

Образцы звуков	Английский MOS
LSTM-RNN parametric	3.67 ± 0.098
HMM-driven concatenative	3.86 ± 0.137
Autoregressive WaveNet	4.41 ± 0.069
Distilled WaveNet	4.41 ± 0.078

Таблица 3. Сравнение MOS для parallel WaveNet.

Улучшенное решение имеет более высокую скорость, но требует качественно обученную нейронную сеть - учителя. Это большой недостаток, ведь перед обучением Parallel WaveNet, нужно сначала обучить оригинальную WaveNet.

Так же стоит отметить тот факт, что в открытом доступе нет официальной имплементации Parallel WaveNet, а неофициальные демонстрируют плохое качество синтеза.

2.3.2. Tacatron2 + WaveGlow

Архитектура вокодера WaveGlow [10] основана на flow based подходе [11], при котором каждый слой нейросети является обратимым преобразованием. WaveGlow содержит 12 последовательных цепочек обратимых преобразований следующего вида (рис. 12).

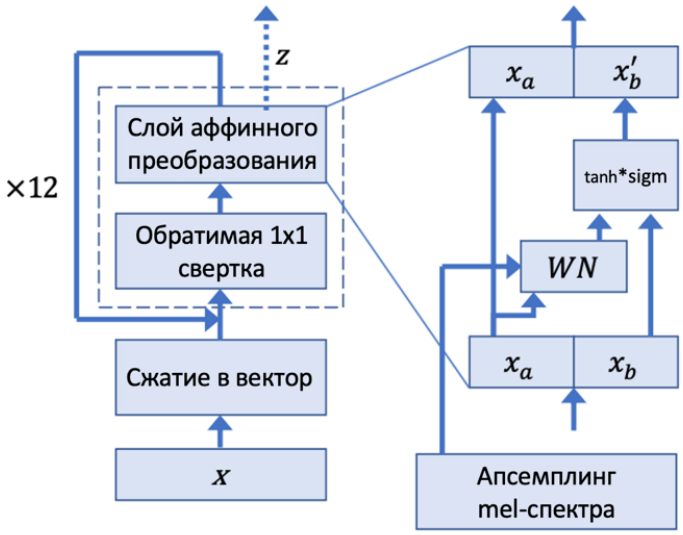

Рисунок 12. Архитектура WaveGlow.

В отличии от генеративных моделей типа WaveNet, WaveGlow – вероятностная модель. WaveGlow строит обратимые преобразования, изменяющие распределения вероятностей, из распределения Гаусса.

Работа каждой из двенадцати цепочек преобразования происходит в несколько этапов:

1. Исходное распределение Гаусса подаётся на вход первой цепочки преобразований.
2. Происходит перемешивание каналов с помощью обратимой 1x1 свёртки.
3. Производится нелинейное обратимое преобразование полученного вектора:

a. Первая половина вектора и честь mel-спектра подаётся на вход нейросети WaveNet-mini.
b. Выход WN объединяется со второй частью вектора с помощью перемножения tanh * sigm.
c. Происходит склейка первой части входного вектора с вектором, полученным на предыдущем этапе.

4. Трансформированный вектор подаётся на вход следующей цепочке.

В итоге, из простого распределения Гаусса и mel-спектрограммы, получается готовая звуковая волна.

При обучении, все эти операции производятся в обратную сторону. То есть задача – научить WaveGlow получать распределение Гаусса, стартуя с готовой звуковой волны. Это позволяет напрямую минимизировать логарифм вероятности распределения Гаусса:

$$\log p_\theta(\boldsymbol{x}) = -\frac{\boldsymbol{z}(\boldsymbol{x})^T \boldsymbol{z}(\boldsymbol{x})}{2\sigma^2} + \sum_{j=0}^{\#coupling} \log \boldsymbol{s}_j(\boldsymbol{x}, mel) + \sum_{k=0}^{\#conv} \log \det|\boldsymbol{W}_k|$$

Здесь первое слагаемое пришло от распределения Гаусса, а второе и третье из-за учёта обратных преобразований.

А теперь обратим особое внимание на то, как работает каждый блок WaveNet-mini внутри обратимого аффинного преобразования (рис. 13).

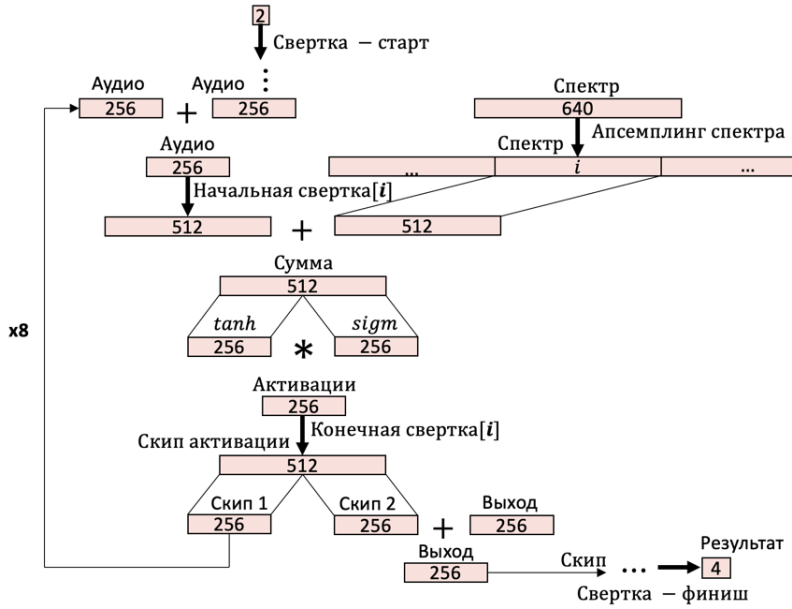

Рисунок 13. Архитектура WaveNet mini.

В начале работы вход сети подаётся на «Свёртка – старт», которая апсемплит его в вектор с размером 256. С помощью свёртки «Апсемплинг спектра» происходит изменение размеров спектра таким образом, чтобы его длина нацело делилась на 8, и длина каждой восьмой части была равна 512. После этого происходит 8 последовательных преобразований следующего вида:

1. Вектор «Аудио» подаётся на вход «Начальная свёртка[i]», получается вектор с размером 512.
2. i-я часть спектра складывается с вектором «Аудио» и от первой части суммы берётся гиперболический тангенс, а от второй – сигмойда. Получившиеся векторы поэлементно перемножаются, а результат подаётся на вход «Конечная свёртка[i]».
3. Первая часть результата свёртки суммируется с первой частью результата предыдущего шага, а вторая часть – с результатом второй части предыдущего шага.

Таким образом происходит 8 последовательных преобразований, после которых суммарный вектор «Выход» подаётся на вход «Свёртка – финиш» и работа блока WN прекращается.

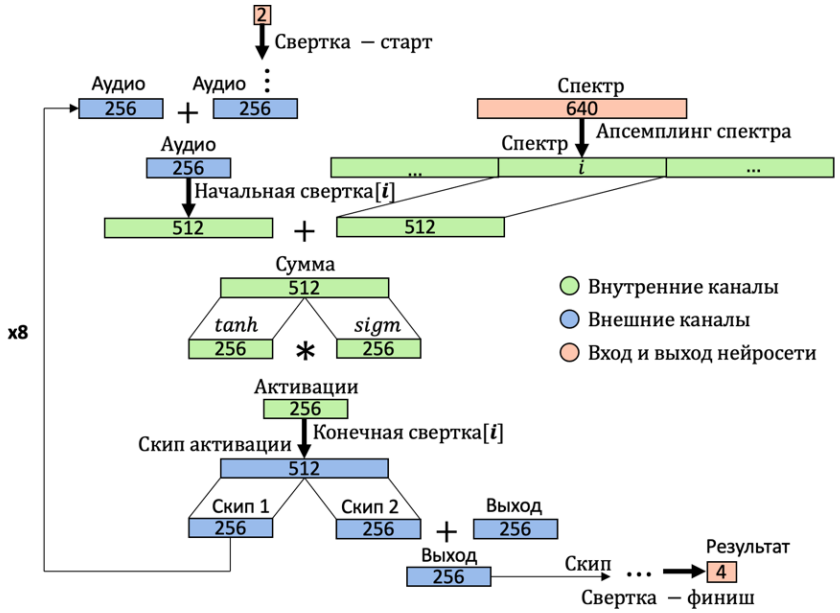

Рисунок 14. Внутренние и внешние каналы WaveNet mini.

Видно, что в этой системе есть такие векторы, которые обособлены от внешних операций свёртками и участвуют только во внутренних операциях. Такие вектора называются внутренними. С другой стороны, есть такие векторы, которые участвуют в глобальном суммировании и формировании результата. Такие вектора называются внешними (рис. 14). На рисунке внешние и внутренние вектора выделены разными цветами.

В эксперименте было получено, что модель WaveGlow может синтезировать речь практически не отличимую от человеческой. Средняя оценка MOS превосходит качество модели WaveNet [4] (таблица 4).

Образцы звуков	Английский MOS
Griffin-Lim	3.823 ± 0.1349
WaveNet	3.885 ± 0.1238
WaveGlow	3.961 ± 0.1343
Ground Truth	4.274 ± 0.1340

Таблица 4. Сравнение MOS для parallel WaveNet.

2.4. Прунинг нейросетей

2.4.1. Идея и основные концепции

Прунинг – отсечение малозначимых частей нейронной сети. Начать стоит с того, как в общем случае происходит процедура прунинга. Она состоит из следующих шагов (рис. 15).

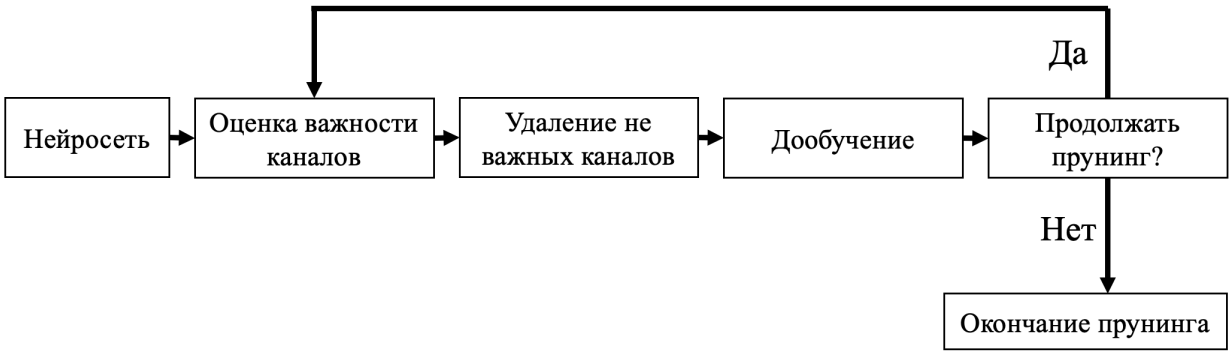

Рисунок 15. Процедура прунинга.

На первом шаге происходит оценка важности каналов в каждой свёртке сети. На втором шаге происходит удаление наименее значимых каналов из архитектуры. После этого нужно дообучить нейронную сеть для того, чтобы она смогла подстроить оставшиеся параметры под новую архитектуру. Эту процедуру стоит продолжать до тех пор, пока не будет найден компромисс между желаемой скоростью работы и качеством синтеза модели.

Процедура прунинга очень простая, однако она основывается на том, известен хороший критерий, по которому нейроны считаются важными.

Множество параметров изменяется таким образом, чтобы сохранить точность сети: $C(D|W') \approx C(D|W)$, где W' – число ненулевых параметров после прунинга. Это можно трактовать как решение комбинаторной задачи:

$$\min_{W'} |C(D|W') - C(D|W)|$$

Если $W' = W$, модель находится в минимуме этой функции.

Исследования в области отсечения лишних частей свёрточных нейронных сетей начинались с самого раннего развития глубокого машинного обучения. Optimal Brain Damage (LeCun et al., 1990) и Optimal Brain Surgeon (Hassibi &

Stork, 1993) [14], [15] полагались на члены второго порядка в разложении Тейлора для выбора нужных параметров, используя прунинг для ускорения тренировки и улучшения генерализации. Эти методы требуют частичного или полного вычисления Гессиана матриц, что добавляет сложности вычислений и требует больше памяти по сравнению со стандартным обучением параметров.

В работе Anwar et al. (2015) [9] было исследовано применение структурного прунинга для свёрточных нейросетей на уровне feature maps и ядер. Прунинг выполняется частичным отсеиванием, в котором каждая конфигурация сети оценивалась количеством ошибок при работе. Этот метод показал неплохие результаты на маленьких CNN, однако не было исследования больших CNN архитектур.

Han et al. (2015) [16] предложил подстройку параметров строгой L_2 регуляризацией, когда происходит отсечение всех весов, значение которых меньше некоторого порогового значения. Такой неструктурированный прунинг крайне эффективен для сжатия размеров нейросетей и показывает хорошие результаты при использовании на внутренних ядрах. Однако сжатие не всегда ведёт к ускорение работы так как современные графические процессоры имеют хорошую пропускную способность. Этот подход требует долгой подстройки и общее время тренировки может быть увеличено в несколько раз.

В работе Pavlo Molchanov (2017) [17] были исследованы различные критерии прунинга свёрточных сетей.

2.4.2. Минимизация веса

Прунинг на основе величины весов ядер, пожалуй, самый простой из возможных критериев прунинга, не требующий никаких дополнительных вычислений во время процесса подстройки. Требуется выбрать веса таким образом, чтобы функция

$$\Theta(W) = \frac{1}{|W|}\sum_i \omega_i^2$$

была минимальной. Обоснование применимости этого критерия заключается в том, что те ядра, которые имеют веса с малой L_2 нормой работают с менее важными частями сети. Этот метод может применяться совместно с L_1 или L_2 нормализацией во время обучения для того, чтобы заранее уменьшить не важные веса.

2.4.3. Дисперсии активаций

Разумно предположить, что если активации на некотором ядре малы, то оно мало влияет в формировании результата. Поэтому задачу можно поставить так, чтобы требовалось отсекать каналы с малым средним значений активаций:

$$\Theta(\boldsymbol{a}) = \frac{1}{|\boldsymbol{a}|}\sum_i a_i$$

или малым стандартным отклонением:

$$\Theta(\boldsymbol{a}) = \sqrt{\frac{1}{|\boldsymbol{a}|}\sum_i (a_i - \mu_{\boldsymbol{a}})^2}$$

Во время синтеза выхода сети некоторые активации меняются очень сильно, подстраиваясь под контекст и определяя результат, а некоторые – практически не меняются, а значит на соответствующих каналах формируется константа. Интуиция подсказывает, что вклад констант с неизменяющихся каналов может быть заменён изменением смещений каналов с большой дисперсией.

То есть если во время работы выяснится, что некоторые каналы имеют очень маленькую дисперсию и практически не изменяются во время синтеза речи, то разумно будет попытаться исключить их из архитектуры.

Для этого метода важно чтобы значения дисперсий каналов различались на порядки. В случае, если на всех каналах дисперсии одного порядка, данный критерий не может быть применён.

2.4.4. Взаимная информация

Взаимная информация (MI) – это мера того, как много информации представляет одна переменная о другой. MI критерий применяется следующим образом: $\Theta(\boldsymbol{a}) = \mathrm{MI}(\boldsymbol{a}, y)$, где y – разметка тренировочных данных. MI определяется для непрерывных переменных и для упрощения вычислений он заменяется information gain, определяемый для дискретных переменных: $IG(y|x) = H(x) + H(y) - H(x, y)$, где $H(x)$ – энтропия переменной x. Нужно собрать статистику по всем активациям и разметке на нескольких примерах, а потом квантовать переменные и вычислять IG.

2.4.5. Разложение в ряд Тейлора

Как упоминалось ранее, задача прунинга может интерпретироваться как задача нахождения такого множества не отсечённых параметров W', при котором разность $|\Delta C(h_i)| = |C(D|W') - C(D|W)|$ будет минимальной. С помощью разложения Тейлора можно оценить изменение loss функции из-за удаления одного параметра. Пусть h_i – часть выхода нейросети, полученная от параметра i. Считая параметры сети независимыми, имеем:

$$|\Delta C(h_i)| = |C(D|h_i = 0) - C(D|h_i)|,$$

На самом деле, предположение о независимости параметров сети уже было сделано при использовании метода обратного распространения ошибки для вычисления градиентов.

Для оценки $\Delta C(h_i)$ используется первое приближения ряда Тейлора. Функция $f(x)$ вокруг точки $x = a$ выражается полиномом Тейлора:

$$f(x) = \sum_{p=0}^{P} \frac{f^p(a)}{p!}(x - a)^p + R_P(x),$$

где $f^p(a)$ – производная порядка p в точке a и $R_P(x)$ – остаток разложения порядка P. Разложим функцию $C(D|h_i)$ вокруг точки h_i:

$$C(D, h_i = 0) = C(D, h_i) - \frac{\delta C}{\delta h_i} h_i + R_1(h_i)$$

Остаток $R_1(h_i)$ может быть записан в форме Лагранжа:

$$R_1(h_i) = \frac{\delta^2 \mathrm{C}}{\delta(h_i^2 = \xi)} \frac{h_i^2}{2}$$

Однако он не будет учитываться при вычислении $\Delta C(h_i)$:

$$\Theta(h_i) = |\Delta C(h_i)| = \left| C(D|h_i) - \frac{\delta C}{\delta h_i} h_i - C(D|h_i) \right| = \left| \frac{\delta C}{\delta h_i} h_i \right|$$

Интуитивно понятно, что такой критерий отсекает параметры, по которым производная loss функции пренебрежительно мала. Этот подход требует подсчёта производной loss функции по выходу от параметра h_i, но эти вычисления можно заменить на вычисления градиентов из метода обратного распространения ошибки. Для мультиканального выхода используется цепное правило:

$$\Theta\left(z_l^{(k)}\right) = \left| \frac{1}{M} \sum_m \frac{\delta C}{\delta z_{l,m}^{(k)}} z_{l,m}^{(k)} \right|,$$

где M – длина выхода. Для нескольких примеров эта метрика усредняется.

2.4.6. Средний процент нулей

В работе Hu et al. (2016) [18] предложено использовать размер активаций в качестве критерия прунинга.

ReLU активации убирают малые активации во время работы и среднее количество положительных активаций могут определить важность нейрона.

Может показаться, что это неплохой критерий, однако карты распределений активаций на первых слоях подстраиваются под выход нейросети так как слои обучаются работать в качестве фильтров Габора [21] [22].

3. Постановка задачи

Из таблицы (рис. 2) видно, что длительность синтеза речи определяет длительность ответа системы интерактивного взаимодействия. Это говорит о том, что необходимо ускорить алгоритм генерации речи.

Принято решение ускорять алгоритм синтеза речи с помощью структурного прунинга.

Были поставлены следующие цели:

- Исследовать особенности работы flow based моделей
- Исследовать современные методы прунинга
- Реализовать модуль для гибкого прунинга flow based архитектуры
- Ускорить алгоритм синтеза речи в 2 раза на Nvidia GTX 1050-ti без существенной потери качества
- Протестировать качество ускоренных моделей (MOS, WER)

4. Разработка алгоритма ускорения

4.1. Измерение скорости этапов синтеза речи

Решение, на котором мы решили остановится - нейросеть от Nvidia, WaveGlow [10], которая, по сути, взяла за основу WaveNet [3], [4], [5] и перенесла на flow-based архитектуру [11]. Данное решение демонстрирует лучшее качество синтеза и находится в открытом доступе. А также, что немаловажно, доступна предобученная модель, с которой можно работать.

Другие решения, рассмотренные нами в обзоре научной области, не подходят либо из-за низкого качества синтеза, либо из-за отсутствия модели в открытом доступе.

Для того, чтобы понять, как именно стоит ускорять алгоритм генерации речи, нужно сделать измерения скорости работы каждой части системы.

По отдельности были проанализированы все необходимые модули сети, начиная от загрузки конфигурационных файлов, заканчивая синтезом готового звука. Синтез речи происходит в несколько этапов (таблица 5).

Процесс	Время, ms	% Времени
Загрузка конфиг. Файлов	28233.00	–
Преобработчик текста	2.10	0.004
Tacatron-2 (encoder)	12783.00	20.110
WaveGlow (vocoder)	50780.00	79.886

Таблица 5. Этапы синтеза речи.

1. Загрузка конфигурационных файлов

2. Работа преобработчика текста
3. Энкодер Tacatron2
4. Вокодер WaveGlow

Из таблицы видно, что самая медленная часть – вокодер WaveGlow. Это та часть системы, которая синтезирует звук из mel-спектрограммы. Поэтому именно с этого модуля было решено начать ускорение системы.

Следующий этап – измерение скорости каждого шага внутри вокодера WaveGlow. Для этого использовался профилировщик от PyTorch [20] (рис. 16).

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
model_inference	11.91%	4.813s	100.00%	40.396s	40.396s	1
aten::conv1d	1.51%	610.425ms	36.11%	14.585s	60.771ms	240
aten::convolution	1.24%	501.203ms	34.64%	13.991s	58.055ms	241
aten::_convolution	6.91%	2.789s	33.39%	13.490s	55.976ms	241
aten::_weight_norm	2.12%	858.008ms	19.42%	7.847s	36.328ms	216
aten::slice	14.98%	6.051s	18.92%	7.645s	5.387ms	1419
aten::mkldnn_convolution	16.81%	6.789s	17.60%	7.108s	32.906ms	216
fused_add_tanh_sigmoid_multiply	2.38%	962.862ms	17.44%	7.043s	73.366ms	96
aten::norm_except_dim	2.96%	1.197s	11.96%	4.832s	22.371ms	216
aten::norm	5.19%	2.096s	6.06%	2.449s	11.336ms	216

Рисунок 16. Список самых долгих операций WaveGlow.

На рисунке видно, что наибольшее время занимают свёртки, это говорит о необходимости их первоочередном ускорении. Работы в области ускорения нейросетей показали эффективность применения прунинга – отсечения малозначимых частей нейронной сети. Для применения этого способа, нужно выбрать критерий, по которому некоторые каналы будут считаться малозначимыми, а также выбрать, в каком месте будет происходить отсечение.

В качестве критерия значимости каналов было решено выбрать **оценку дисперсии**. Для того, чтобы понять, как и в каком месте вокодера WaveGlow будет происходить отсечение каналов, нужно провести анализ архитектуры и выяснить, где происходят все самые долгие операции свёрток.

4.2. Прунинг архитектуры WaveGlow

Для того, чтобы понять, как правильно отсекать каналы, нужно выбрать вектор, на котором будет измеряться дисперсия, а после отсечения распространить изменения на все вектора, которые участвуют в его

формировании. В качестве такого вектора была выбрана «Активация» (рис. 17). Именно он напрямую определяет результат выхода и тот вектор, который пойдёт на вход следующей цепочке. Это значит, что именно с прунинга внутренних каналов начнётся ускорение данной архитектуры.

Важно при попытке ускорения сети не потерять свойство обратимости. Оказывается, что вся вычислительная сложность сосредоточена внутри блоков WaveNet mini, ускоряя которые мы не потеряем обратимость преобразований. Поэтому нужно ускорять именно эту часть WaveGlow.

4.3. Прунинг внутренних каналов WaveNet mini

Для того, чтобы произвести прунинг внутренних каналов, нужно не только отсечь каналы вектора «Активации», но и распространить отсечение на все внутренние векторы, которые участвуют в формировании «Активации». Это нужно сделать аккуратно, поэтому для начала рассмотрим удаление одного канала, а все остальные будем отсекать по аналогии.

Представим, что перед нами встала задача вырезать канал с номером k у вектора «Активации» (рис. 17). Тогда у «Конечная свёртка[i]» нужно вырезать те веса, которые умножаются на канал k. Получится уменьшенная версия свёртки, а значит время её исполнения сократится. В формировании «Активации» поучаствовало два вектора – tanh и sigm. Значит у каждого из них нужно вырезать канал с номером k. tanh и sigm были сформированы из вектора «Сумма». Это значит, что у этого вектора нужно отсечь два канала: один из первой половины (под номером k), а один из второй половины (под номером 256 + k). Вектор «Сумма» была сформирована суммой двух векторов: частью спектра и выходом «Начальная свёртка[i]».

Теперь нужно удалить из свёртки все веса, которые отвечают за формирование каналов k и 256 + k. Остался ещё второй вектор – «Спектр». Этот вектор – одна восьмая часть всего спектра, а значит из большого спектра нужно удалить 2 канала с номерами 512*i + k и 512*i + 256 + k. Из свёртки «Апсемплинг спектра» нужно удалить все веса, которые влияют на формирование каналов с номерами 512*i + k и 512*i + 256 + k.

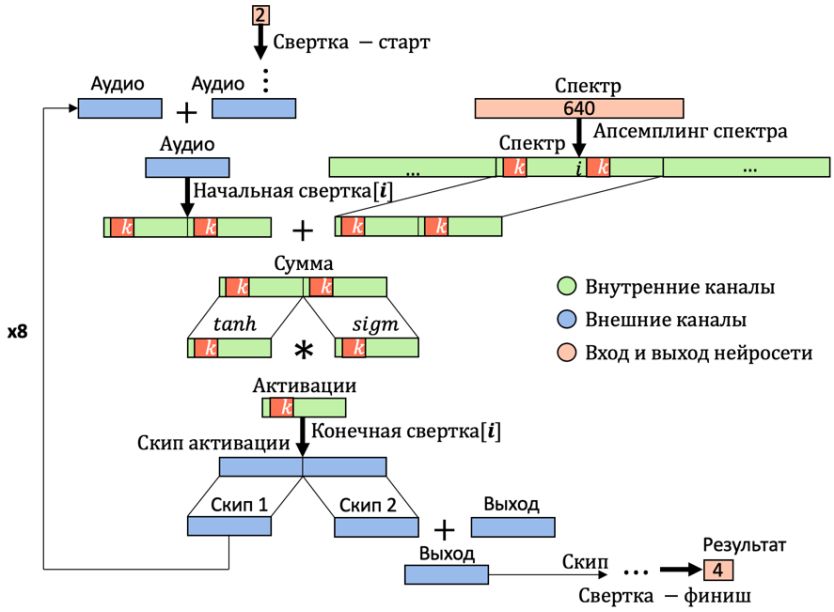

Рисунок 17. Прунинг внутренних каналов WaveNet mini.

Эту процедуру нужно проделать со всеми 8 слоями WN, собирая индивидуальную статистику дисперсий в каждом из них.

WN – это часть аффинного преобразования внутри одной из двенадцати цепочек обратимых преобразований WaveGlow, а значит всё это вместе нужно проделать 12 раз. Такую операцию нужно произвести столько раз, сколько требуется удалить каналов.

Стоит отметить, что перед тем, как производить отсечение, нужно запустить уже обученную и хорошо работающую модель на некотором примере для набора статистики по дисперсиям.

Во время работы стало понятно, что ни одна из существующих библиотек не даёт возможность быстрого и гибкого прунинга подобных flow based моделей, поэтому было решено написать небольшую библиотеку специально для этой цели. Задача состояла в том, чтобы можно было совершать прунинг модели в несколько строчек.

Реализованные методы имеют следующую сигнатуру:

1. prune_conv1d (conv: Conv1d, channels: array, out: bool = True). Эта функция принимает на вход свёртку, прунинг которой нужно произвести, массив каналов, которые нужно оставить и флаг, говорящий о том, вход или выход свёртки нужно прунить.

2. prune_wn_inside (wn: WN, inside_channels: int), которая принимает на вход нейронную сеть WaveNet mini и количество внутренних каналов, которые останутся после прунинга.
3. prune_waveglow (waveglow: WaveGlow, inside_channels: int), которая по очереди вызывает функцию prune_wn_inside для каждого из двенадцати WaveNet mini.

Была собрана статистика по самым популярным фразам колл-центра и набрали из них большой текст. На этом тексте были посчитаны дисперсии по каналам у каждого блока WN. Если посмотреть на один из графиков статистики дисперсий, то мы увидим следующую картину (рис. 18).

Здесь по оси Y – логарифм дисперсии по каналу, а по оси X – номер канала. Для того, чтобы график можно было удобно анализировать, перед построением, каналы сортируются по возрастанию, поэтому получается неубывающая функция. График построен в логарифмическом масштабе, чтобы было лучше видно разительное отличие в дисперсиях. Действительно, на графике видно, что значения дисперсий могут отличаться между собой на несколько порядков. Это значит, что некоторые каналы вообще не изменяются (по порядку величины) относительно других.]

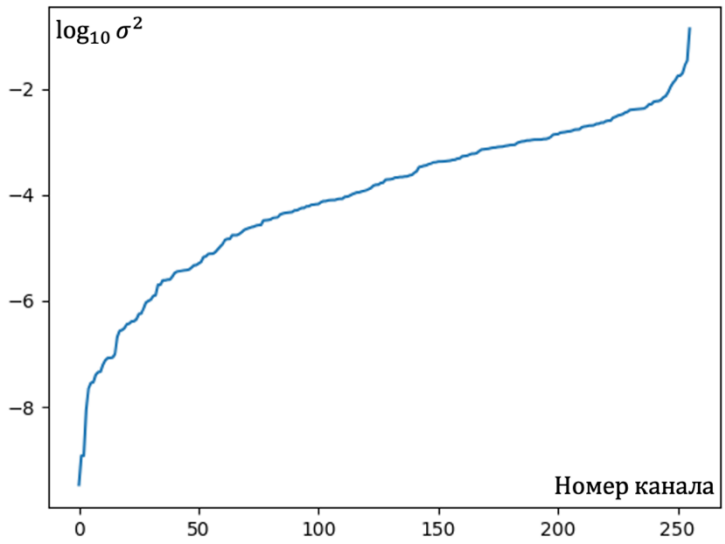

Рисунок 18. Распределение дисперсий для каналов одного слоя блока WaveNet mini. Наблюдается большой разброс.

Но это лишь один график дисперсии для одной цепочки и одного слоя WaveNet mini. Для того чтобы убедится в том, что такое поведение дисперсий характерно для всех каналов в сети WaveGlow, нужно проанализировать все графики. Так как мы имеем двенадцать WaveNet mini с восьмью статистиками на каждой = 96 статистик по дисперсиям, статистики из одной и той же сети WaveNet mini размещаются на одном графике. Мы всё ещё сможем анализировать поведение дисперсий по каналам, но теперь вместо 96 графиков нам потребуется построить только 12 (рис. 19).

Видно, что на всех активациях поведение дисперсий очень похожее, а значит использование дисперсии в качестве критерия прунинга может оказаться крайне эффективным.

Для демонстрации того, что каналы с большой дисперсией действительно самые важные, мы провели следующий показательный эксперимент: сначала мы вырезали половину каналов с наибольшей дисперсией. Оказалось, что модель в этом случае синтезирует тишину. То есть в результате ничего не слышно.

После этого, была удалена половина каналов с наименьшей дисперсией. В этом случае модель синтезировала разборчивую речь, но с некоторым ухудшением качества синтеза. Однако сам факт того, что речь была отчётливо слышна свидетельствует о корректности выбранного критерия.

Это показывает, что каналы с большой дисперсией больше всего влияют на результат работы сети WaveGlow.

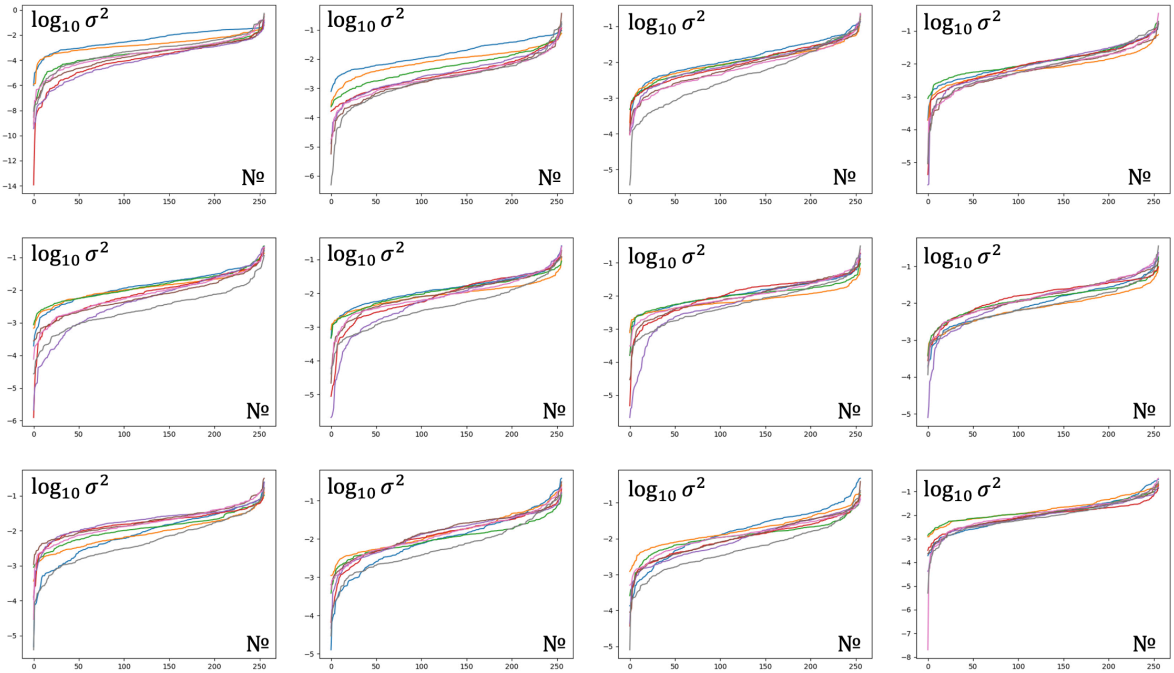

Рисунок 19. Распределения дисперсий всех слоёв каждого блока WaveNet mini. Наблюдается большой разброс.

Было решено сделать прунинг модели, оставив половину внутренних каналов.

Для сравнения с оригинальной архитектурой (рис. 14), приведена схема обновлённой WaveNet mini (рис. 20). На ней видно, какие вектора были уменьшены.

Как уже было сказано, качество синтеза речи сразу после отсечения каналов плохое. Так и должно быть, ведь модель потеряла часть константных каналов. Теперь нужно провести дообучение, изменив веса и смещения так, чтобы сократить отставание от оригинальной модели.

Как и в оригинальной реализации WaveGlow, обучение производилось с оптимизатором Adam с коэффициентом скорости обучения = 10^-4 на видеокарте Nvidia GTX 1080-ti с batch size = 12. Обучение длилось неделю, в течении которой прошло более 70 000 итераций (рис. 21).

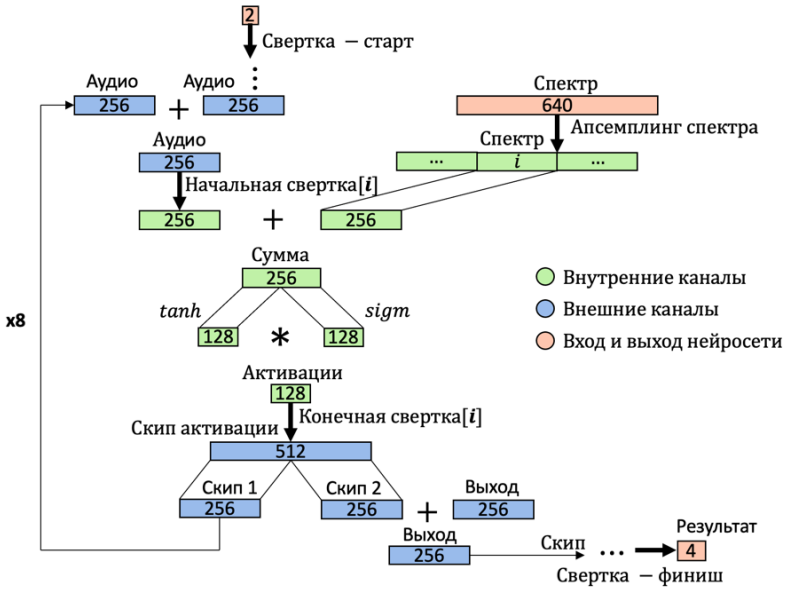

Рисунок 20. Модифицированная версия WaveGlow (половина внутренних каналов отсечена).

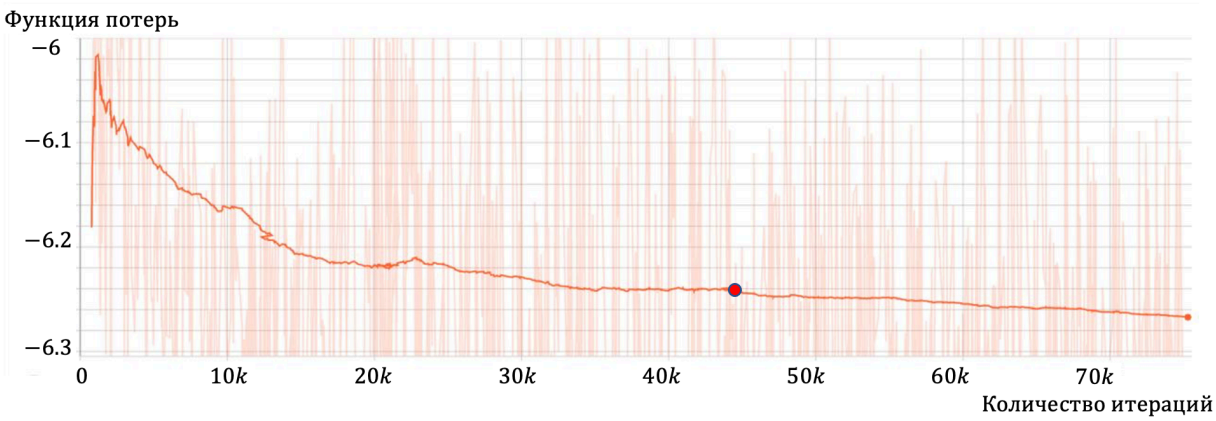

Рисунок 21. График зависимости функции потерь от количества итераций обучения.

Качество модели оказалось практически идентичным исходной модели: на слух отличить оригинальную и ускоренную модель оказалось крайне тяжело. Мы уменьшили коэффициент скорости обучения в два раза и продолжили обучение до 80 000 итераций, но улучшений это не дало. Напротив, модель при обучении с коэффициентом скорости обучения = 5*10^-5 становилась хуже и как будто теряла высокие частоты в синтезированной речи. Было решено оставить модель на 128 внутренних каналов как готовую к использованию и перейти к следующим экспериментам.

Теперь, когда готова хорошая модель WaveGlow, ускоренная почти в два раза, было решено двигаться дальше, оставив лишь 64 внутренних канала (то есть уменьшив размер внутренних векторов в 4 раза по сравнению с оригинальной моделью).

Обучение ускоренной модели проводилось с теми же самыми параметрами оптимизатора на протяжении 80 000 итераций (рис. 22).

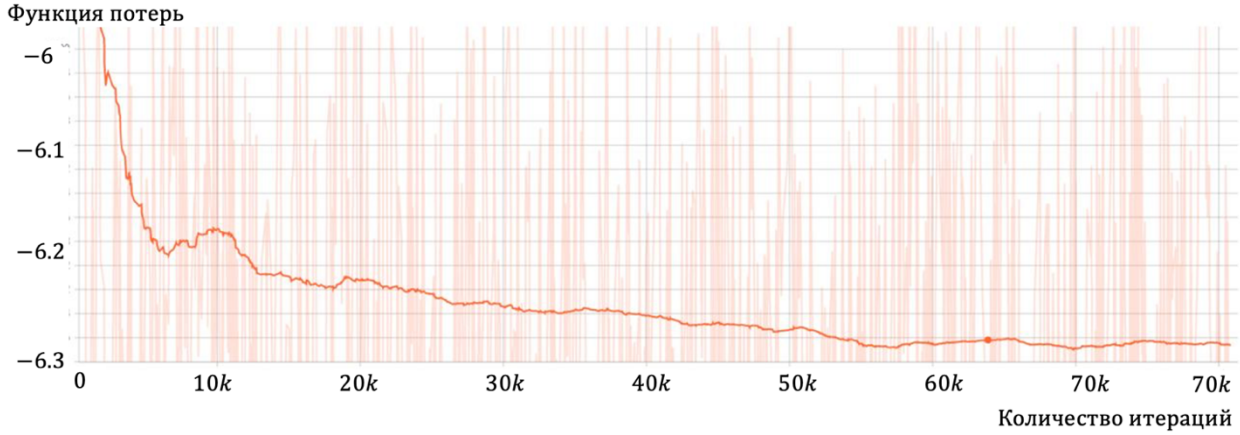

Рисунок 22. Обучение модели на 64 канала.

Оказалось, что она даёт неплохое качество синтеза, но в данном случае уже слышно различие от оригинальной WaveGlow: на некоторых фразах голос диктора как будто становится немного хриплым, а общее качество звука немного ухудшилось. Такую модель можно использовать в колл-центрах, где связь не чувствительна к таким дефектам.

Также стоит отметить тот факт, что на спектрограммах, на которых WaveGlow обучалась, качество синтеза после дообучения практически не изменилось, но при тестировании на спектрограммах, сгенерированных с помощью Tacatron-2, качество модели заметно упало. То есть экстремальный прунинг каналов приводит к потере генерализации.

4.4. Измерения скорости при прунинге внутренних каналов

После применения процедуры прунинга необходимо было протестировать скорость работы ускоренных и базовой модели. Результаты измерений приведены в таблице на рис. 28. Все измерения производились на видеокарте Nvidia GTX 1050-ti с помощью профилировщика от PyTorch.

Для начала, были произведены замеры времени синтеза на тексте, длинною больше ста слов на исходной модели. Получилось чуть больше минуты синтезированной речи. После этого, был проведён тот же тест, но с ускоренной моделью на 128 и 64 каналов (таблица 6).

Модель	Время синтеза, сек	Ускорение
WaveGlow 256 (исх.)	27.337	x1
WaveGlow 128	18.557	x1.5
WaveGlow 64	14.305	x2

Таблица 6. Сравнение скорости моделей.

Как видно, с моделью на 128 каналов мы получили значительный прирост по скорости синтеза. С моделью на 64 канала мы добились ускорение в два раза.

4.5. Прунинг внешних каналов WaveNet mini

С целью дальнейшего ускорения было решено попробовать применить прунинг внешних каналов. Необходимо отметить, что внешние каналы разных блоков WaveNet mini связаны между собой (в отличии от внутренних). Поэтому стоит выбрать одну цепочку WN, в которой будут отслеживаться дисперсии вектора, провести отсечение, а потом распространить отсечение на все остальные цепочки WN.

В качестве вектора, на котором будут вычисляться дисперсии, был выбрал вектор «Выход» у самого последнего блока WN, который напрямую определяет конечный результат. Допустим, что нам требуется отсечь канал с номером s (рис. 23).

Точно так же, как и в случае внутренних каналов, нужно распространить отсечение на все векторы, которые участвуют в формировании «Выход». То есть нужно так же отсечь канал с номером s у векторов «Скип 2», «Скип 1» и «Аудио», а также каналы s и 256 + s у вектора «Скип активации». Теперь нужно изменить свёртки «Свёртка – финиш», «Свёртка – старт» и

«Начальная свёртка[i]», удалив все веса, которые умножаются на соответствующие каналы.

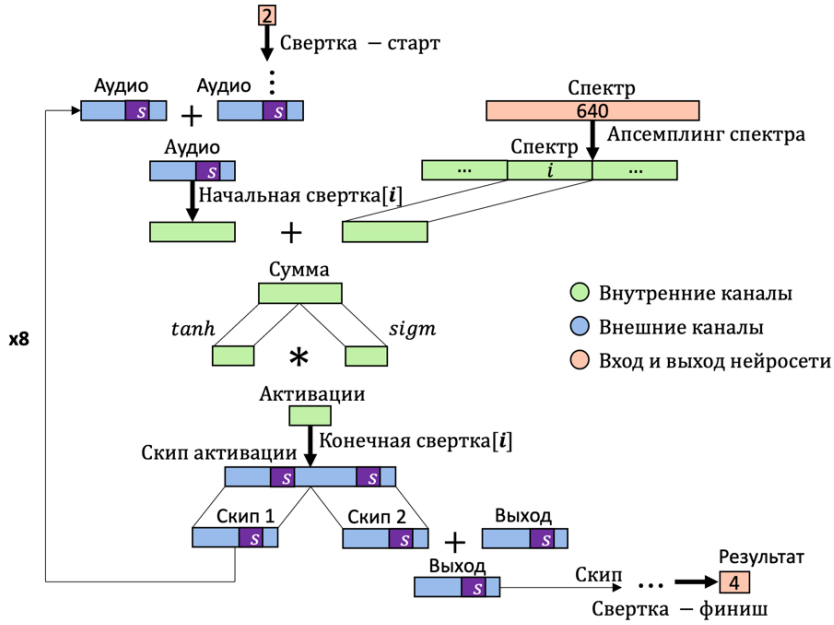

Рисунок 23. Прунинг внешних каналов WaveNet mini.

После этого, стоит обратить внимание на то, что вектор «Аудио» пришёл из предыдущего блока WaveNet mini, а значит нужно распространить это отсечение на все предыдущие 7 слоёв WN (рис. 24).

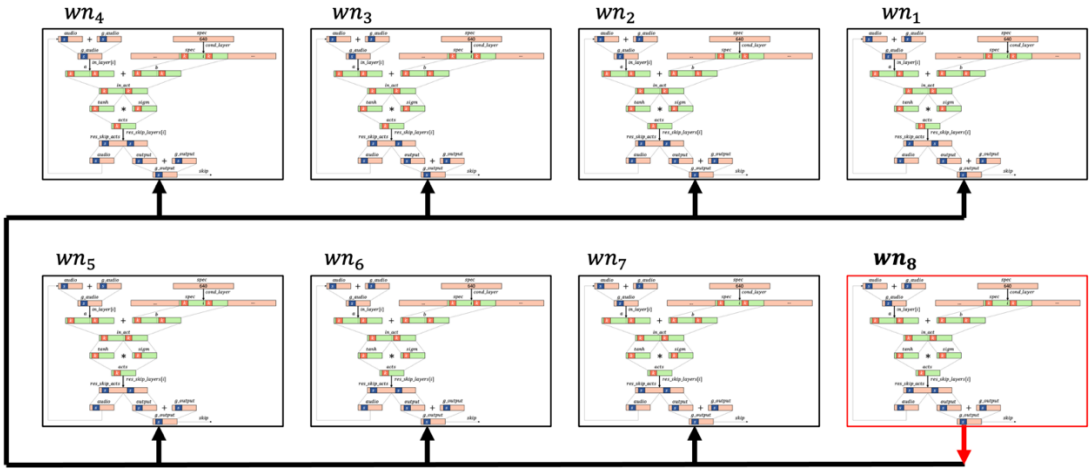

Рисунок 24. Принцип распространения каналов отсечения по всем слоям WaveNet mini.

Теперь необходимо сделать все те же самые действия с остальными 11 блоками WaveNet mini, собрав в каждом статистику по дисперсиям последней цепочки и распространив на них отсечение рассматриваемых каналов.

Требуется проделать перечисленные действия со всеми каналами, которые должны быть удалены из архитектуры.

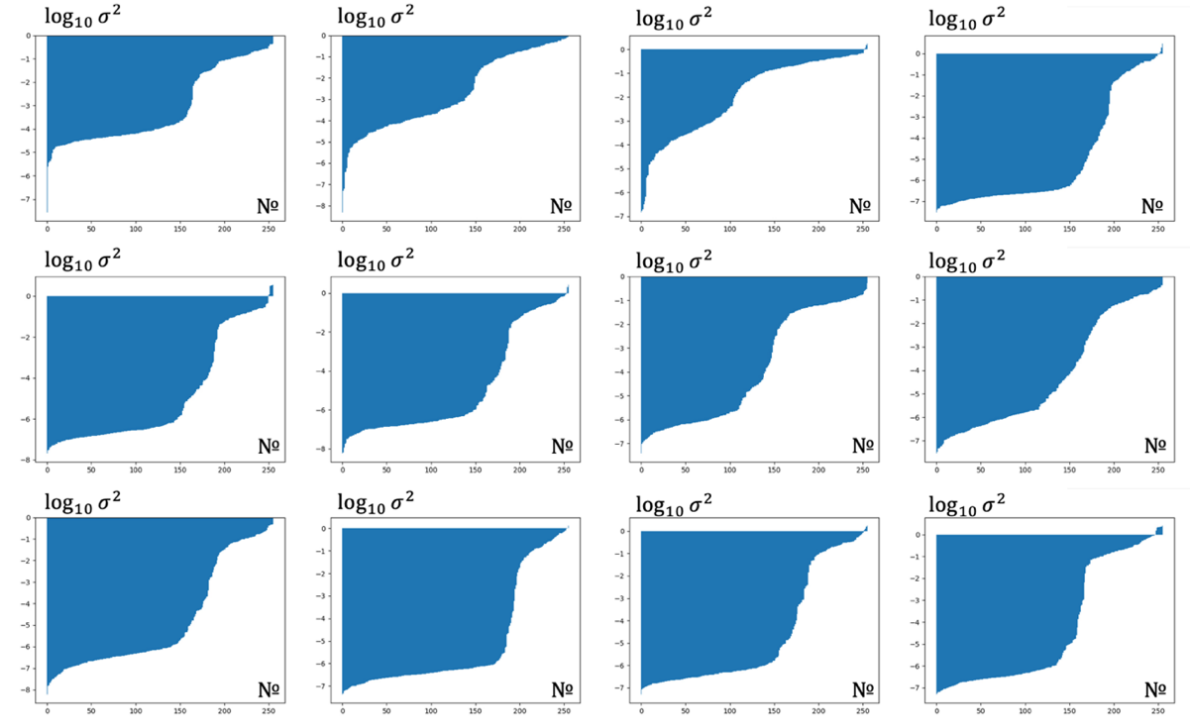

Рисунок 25. Распределение дисперсий векторов "Выход" блоков WaveNet.

Для анализа поведения дисперсий, были построены графики дисперсий для каждого канала. Теперь все внешние каналы внутри одного блока WN – общие и всего мы имеем 12 графиков для WaveGlow (рис. 25).

Здесь по оси Y – логарифм дисперсии по данному каналу, а по оси X – номер канала. Для того, чтобы график можно было удобно анализировать, перед построением, каналы сортируются по возрастанию, поэтому получается неубывающая функция. График построен в логарифмическом масштабе, чтобы было лучше видно разительное отличие в дисперсиях. У всех этих графиков есть одна общая черта: на всех из них большая часть каналов имеют очень маленькое значение дисперсии (порядка 0.00001). Это хорошо, ведь это значит, что использование дисперсии в качестве критерия прунинга внешних каналов может оказаться крайне эффективным.

После отсечения половины внешних каналов, было запущено обучение с оптимизатором Adam с коэффициентом скорости обучения = 10^-4 на

видеокарте Nvidia GTX 1080-ti с batch size = 12. Обучение длилось неделю, в течении которой прошло более 70 000 итераций (рис. 26).
Качество не изменилось и отличий от оригинальной модели не наблюдалось.

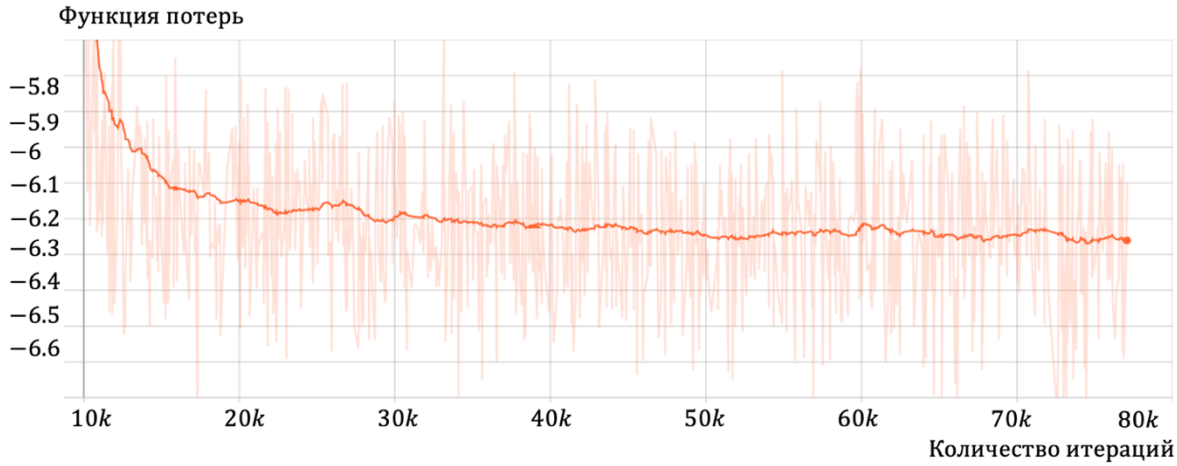

Рисунок 26. Обучение модели с прунингом внешних каналов WaveNet mini.

4.6. Измерения скорости при прунинге внешних каналов

Измерения скорости производились на видеокарте Nvidia GTX 1050-ti с помощью профилировщика от PyTorch.

Для начала, были сделаны замеры времени синтеза на тексте, длинною чуть больше ста слов на исходной модели. Получилось чуть больше минуты синтезированной речи. После этого был проведён тот же тест, но с ускоренной моделью на 128 внешних каналов. По результатам замеров, значительного прироста получено не было (+10% к скорости работы). Это связано с тем, что самое большое влияние на скорость работы WaveNet mini оказывает свёртка «Апсемплинг спектра», которая находится внутри и не урезается при отсечении внешних каналов. Поэтому было решено не прунить внешние каналы и отсекать только внутренние.

4.7. Обучение и подбор оптимизатора

В оригинальной статье WaveGlow при обучении использовался стандартный оптимизатор Adam с коэффициентом скорости обучения = 10^-4. Было решено использовать RAdam[23], MADGRAD[24] и SGD[25].

В отличии от Adam, RAdam удаётся значительно уменьшить шумы функции потерь, но существенного прироста в качестве синтеза обнаружено не было.

Оптимизатор MADGRAD оказался неустойчивым и по прошествии некоторого времени обучения, градиенты перестали вычисляться корректно.

SGD – стандартный оптимизатор, который, тем не менее, может быть крайне эффективен при дообучении модели. При обучении модели на 64 канала, не было получено значительных изменений функции потерь.

4.8. Итеративный прунинг

Можно отсекать каналы не сразу, а постепенно – по ходу обучения модели. Например, каждую тысячу итераций отсекать по 10 внутренних каналов, стартуя с оригинальной модели.

Мы запустили модель на обучение с итеративным прунингом внутренних каналов, отсекая по 10 каналов каждую тысячу итераций.

На графике функции потерь (рис. 27) сразу видны характерные пики, в которых происходило отсечение каналов. Функция потерь резко увеличивается сразу после отсечения, а затем уменьшается до следующего отсечения.

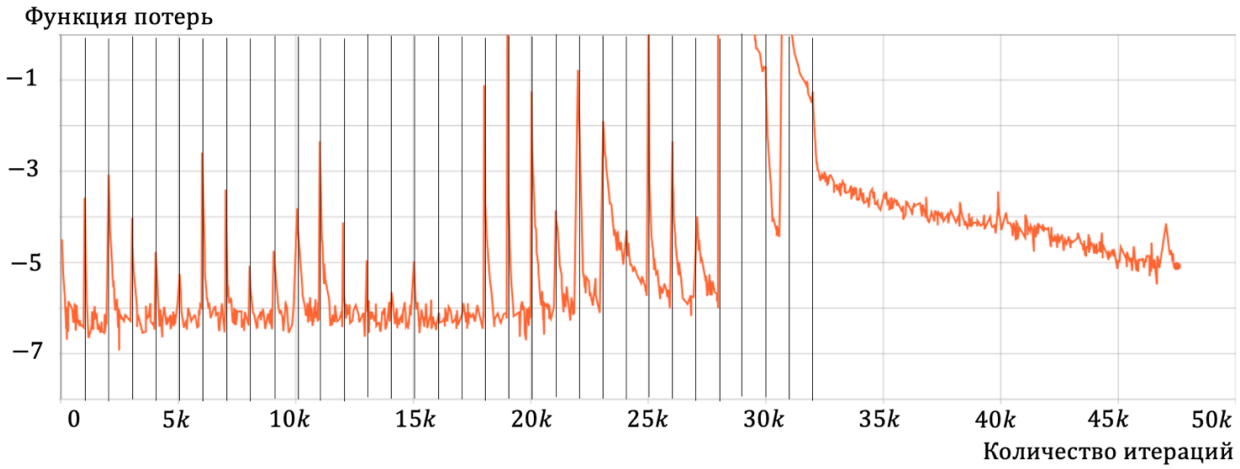

Рисунок 27. Обучение WaveGlow с итеративным прунингом WaveNet.

В сравнении с обычным прунингом, когда каналы удаляются за один раз, данная схема не дала видимого преимущества. Поэтому было решено остановить исследования в этом направлении.

5. Метрики качества

5.1. Mean Opinion Score

В качестве стандартной метрики обычно используется MOS [12] – Mean Opinion Score. Это средняя оценка случайных слушателей от 1 до 5, где 5 – наилучший результат. В опросе будут участвовать несколько моделей и диктор (для образца идеального синтеза). Все примеры для прослушивания будут перемешаны между собой для того, чтобы участники опроса не знали заранее, какую модель они оценивают. Для опроса необходимо выбрать незнакомых людей из разных городов и разных возрастов. Таким образом, мы добьёмся максимально объективной оценки. В опросе будут участвовать четыре типа записей:

1. Модель на 64 канала
2. Модель на 128 каналов
3. Исходная модель на 256 каналов
4. Дикторская речь (в качестве эталона)

Для каждой модели будет по 3 примера. То есть всего опрос содержит 12 примеров. После сбора статистики, нужно вычислить среднюю оценку для каждой модели по её трём примерам: $MOS = \frac{\sum_{n=1}^{N} R_n}{N}$

Для сбора статистики была создана google форма, в которой требовалось поставить оценку случайно перемешанным примерам от 1 до 5. В опросе приняли участие 1000 человек. Результаты представлены в таблице 7.

Модель	Ускорение	Русский MOS
Диктор	–	4.4454
WaveGlow 256 (исх.)	x1	3.6354
WaveGlow 128	x1.5	3.8185
WaveGlow 64	x2	3.3399

Таблица 7. Результаты MOS.

Из табличных данных видно, что у модели WaveGlow-128 наблюдается не только ускорение работы, но и ещё увеличение качества синтеза. Это говорит

о том, что прунинг после применения к оригинальной модели выполняет роль регуляризации, увеличивая обобщающую способность модели.

Однако, модель Waveglow-64 канала, ускоренная в 2 раза, наоборот показалась людям менее натуральной. Как уже упоминалось ранее, она синтезирует речь с небольшим хрипом в голосе. Поэтому эту модель лучше применять в колл центрах, где такие изменения не будут заметны пользователям.

5.2. Word Error Rate

Было решено использовать ещё одну стандартную метрику, WER – Word Error Rate [13], которая вычисляется следующим образом:

$$WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}$$

Где S – количество замен, D – количество удалений, I – количество вставок, C – количество правильных слов, N – количество слов ($N = S + D + C$).

Метрика используется по следующей схеме (рис. 28):

1. Исходный текст подаётся на вход нашему синтезатору речи
2. Синтезированная речь подаётся на вход Google Speech To Text модели, которая из звука получает текст
3. Исходный текст и текст на выходе используются для подсчёта метрики WER

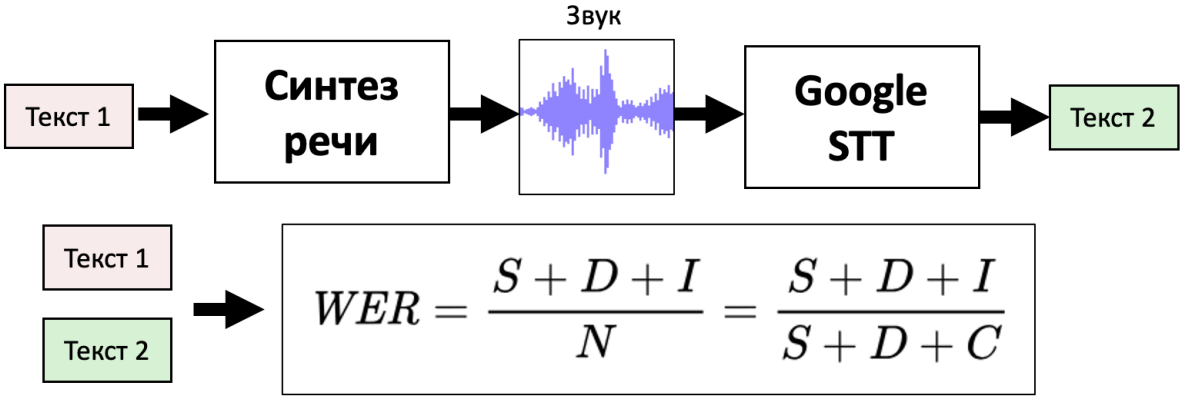

Рисунок 28. Принцип работы метрики WER для измерения качества синтеза речи.

Использовалось 4 типа записей:

1. Синтезированные моделью на 64 канала

2. Синтезированные моделью на 128 каналов
3. Синтезированные исходной моделью на 256 каналов
4. Наговорённые диктором (для эталона)

Для того, чтобы получить максимально объективную метрику, было выбрано 20 больших текстов разного характера и содержания (начиная от озвучек видеороликов и рекламы, заканчивая литературными произведениями и стихотворениями). Для каждой модели метрика усреднялась.

Результаты WER и сравнение с MOS вы можете видеть в таблице 8.

Модель	Ускорение	Русский MOS	Русский WER, %
Диктор	–	4.4454	10.4061
WaveGlow 256 (исх.)	x1	3.6354	18.8701
WaveGlow 128	x1.5	3.8185	17.0594
WaveGlow 64	x2	3.3399	20.9278

Таблица 8. Метрики MOS и WER.

Как видно, даже настоящий диктор не распознаётся google STT без ошибок. Это связано с не идеальностью алгоритма распознавания речи. При сравнении оригинальной модели с нашей, ускоренной на 128 каналов, видно, что ускоренная модель распознаётся лучше оригинальной, ошибаясь на 2% меньше. То есть метрика WER не противоречит MOS, а наоборот в очередной раз подтверждает, что наша модель не только в 1.5 раза быстрее синтезирует речь, но и даёт качество, превосходящее оригинальную модель WaveGlow. С другой стороны, как и предсказывала метрика MOS, модель на 64 канала ошибается на 2% чаще оригинальной. Поэтому эту модель лучше применять в колл центрах, где такие изменения не будут заметны пользователям.

5.3. Результаты

В результате прунинга нам удалось ускорить модель более чем в 1.5 раза на Nvidia GTX 1050-ti с улучшением качества синтеза (MOS: +0.18, WER: -

1.8%) и в 2 раза без значительной потери качества (MOS: -0.29, WER: +2%). Оказалось, что самым эффективным оказался прунинг внешних каналов ввиду того, что он затронул свёртку «Апсемплинг спектра», на которую уходит много вычислительных ресурсов. Выбор разных оптимизаторов (Adam, RAdam, MADGRAD и SGD) не дал существенной разницы в скорости обучения и качестве модели. Применение итеративного прунинга не дало заметного преимущества по сравнению со стандартным обучением.

6. Вывод

Была подробно исследована и ускорена одна из самых популярных архитектур синтеза речи Tacatron2 + WaveGlow. В качестве критерия отсечения каналов использовалась дисперсия.

Численные эксперименты показали, что прунинг может быть эффективно использован для ускорения архитектуры WaveGlow: мы ускорили работу вокодера в полтора раза с улучшением качества звука (MOS: +0.18, WER: -1.8%) и в два раза с незначительным ухудшением (MOS: -0.29, WER: +2%).

Метрики MOS и WER подтвердили состоятельность и эффективность нашей архитектуры. Мы считаем, что такой подход может быть применён для эффективного ускорения других flow based моделей (например, glow [11]), что планируется показать в дальнейших исследованиях.

Планируется дальнейшая оптимизация сети. Мы ведём активные исследования применения квантования (перехода в целочисленное пространство) и дистилляции.

7. Список литературы

1. Hunt A. J., Black A. W. Unit selection in a concatenative speech synthesis system using a large speech database //1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings. – IEEE, 1996. – Т. 1. – С. 373-376.

2. Capes T. et al. Siri On-Device Deep Learning-Guided Unit Selection Text-to-Speech System //INTERSPEECH. – 2017. – С. 4011-4015.

3. Wang Y. et al. Tacotron: Towards end-to-end speech synthesis //arXiv preprint arXiv:1703.10135. – 2017.

4. Shen J. et al. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions //2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). – IEEE, 2018. – С. 4779-4783.

5. Oord A. et al. Wavenet: A generative model for raw audio //arXiv preprint arXiv:1609.03499. – 2016.

6. Van Oord A., Kalchbrenner N., Kavukcuoglu K. Pixel recurrent neural networks //International Conference on Machine Learning. – PMLR, 2016. – С. 1747-1756.

7. Kingma D. P. et al. Improving variational inference with inverse autoregressive flow //arXiv preprint arXiv:1606.04934. – 2016.

8. Ping W., Peng K., Chen J. Clarinet: Parallel wave generation in end-to-end text-to-speech //arXiv preprint arXiv:1807.07281. – 2018.

9. Kalchbrenner N. et al. Efficient neural audio synthesis //International Conference on Machine Learning. – PMLR, 2018. – С. 2410-2419.

10. Prenger R., Valle R., Catanzaro B. Waveglow: A flow-based generative network for speech synthesis //ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). – IEEE, 2019. – С. 3617-3621.

11. Kingma D. P., Dhariwal P. Glow: Generative flow with invertible 1x1 convolutions //arXiv preprint arXiv:1807.03039. – 2018.

12. Katsigiannis S. et al. Interpreting MOS scores, when can users see a difference? Understanding user experience differences for photo quality //Quality and User Experience. – 2018. – Т. 3. – №. 1. – С. 1-14.

13. Zechner K., Waibel A. Minimizing word error rate in textual summaries of spoken language //1st Meeting of the North American Chapter of the Association for Computational Linguistics. – 2000.

14. LeCun Y., Denker J. S., Solla S. A. Optimal brain damage //Advances in neural information processing systems. – 1990. – С. 598-605.

15. Hassibi B., Stork D. G. Second order derivatives for network pruning: Optimal brain surgeon. – Morgan Kaufmann, 1993. – С. 164-171.

16. Han S. et al. Learning both weights and connections for efficient neural networks //arXiv preprint arXiv:1506.02626. – 2015.

17. Molchanov P. et al. Pruning convolutional neural networks for resource efficient inference //arXiv preprint arXiv:1611.06440. – 2016.

18. Han S. et al. EIE: Efficient inference engine on compressed deep neural network //ACM SIGARCH Computer Architecture News. – 2016. – Т. 44. – №. 3. – С. 243-254.

19. www.silero.ai/russian-stt

20. pytorch.org/docs/stable/profiler.html

21. Гашников М. В. и др. Методы компьютерной обработки изображений. – 2003.— С. 459.

22. Храмов, А. Г. Методы восстановления интерферограмм на ЭВМ. — КуАИ, 1988. — С. 88.

23. Liu L. et al. On the variance of the adaptive learning rate and beyond //arXiv preprint arXiv:1908.03265. – 2019.

24. Defazio A., Jelassi S. Adaptivity without Compromise: A Momentumized, Adaptive, Dual Averaged Gradient Method for Stochastic Optimization //arXiv preprint arXiv:2101.11075. – 2021.