РЕФЕРАТ

Магистерская диссертация содержит 30 страниц, 1 таблицу, 14 рисунков. Список использованных источников содержит 7 позиций.

НЕЙРОННЫЕ ДИФФЕРЕНЦИАЛЬНЫЕ УРАВНЕНИЯ, РЕКУРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ, СИСТЕМА СОПРЯЖЕННЫХ УРАВНЕНИЙ, ГЕНЕРАЦИЯ ТЕКСТА, ОЦЕНКА КАЧЕСТВА ГЕНЕРАТИВНЫХ МОДЕЛЕЙ.

Магистерская диссертация посвящена исследованию применимости нейронных дифференциальных уравнений к задачам обработки естественных языков, в частности для задачи генерации текста.

Для решения поставленной задачи была реализована модель генерации текста на основе нейронных дифференциальных уравнений с использованием фреймворка PyTorch. Проведено сравнение реализованной модели с классической моделью, использующей рекуррентную сеть с долгой краткосрочной памятью. Для оценки качества полученных моделей использовались различные метрики, применяемые для моделей генерации текста. Реализованная модель показывает более высокие значения метрик при большей скорости обучения и меньшем потреблением памяти.

Москва 2021 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время резидуальные и рекуррентные нейронные сети широко используются для обработки последовательных данных. Один из важнейших видов последовательных данных – тексты на естественном языке. Применение рекуррентных нейронных сетей в задачах обработки естественных языков позволило решить многие задачи. Однако, у таких моделей есть свои недостатки – медленное обучение и большое количество параметров модели. Это затрудняет обучение на больших корпусах, а также требует больших вычислительных ресурсов на этапе применения модели.

В статье [1] был предложен альтернативный подход к обработки последовательных данных – нейронные дифференциальные уравнения (Neural ODE). Данный подход позволяет представить резидуальную сеть как численную схему решения системы ОДУ. Обучение такой сети сводится к решению системы сопряженных уравнений. Это позволяет отказаться от классического алгоритма обратного распространения ошибки и сделать веса всех слоев общими за счет параметризации новой переменной t, которую следует интерпретировать как глубину слоя. Получаем существенный прирост скорости обучения и уменьшение потребления памяти. При этом мы еще можем улучшить качество работы сети с помощью выбора решателя системы ОДУ с большим порядком точности, а также использую методы с адаптивным шагом (это позволит получить сеть с динамически настраиваемым числом слоев).

Для обработки естественных языков резидуальные сети не подходят из-за отсутствия скрытого слоя. В статье [2] представлен управляемый рекуррентный блок (GRU-ODE), использующий подход нейронных ОДУ.

Таким образом, подход нейронных ОДУ может быть применим для задач обработки естественных языков.

Целью данной работы является исследование применимости подхода нейронных ОДУ для решения задач NLP на примере задачи генерации текста.

В ходе выполнения работы потребовалось решить следующие задачи:

1. Изучить подход нейронных ОДУ. Реализовать алгоритм обучения резидуальной сети и рекуррентного управляемого блока с использованием фреймворка PyTorch. Подготовить программный пакет на языке Python.
2. Реализовать модели генерации текста на языке Python с использованием фреймворка PyTorch для следующих рекуррентных блоков: vanila RNN, LSTM, GRU. Обучить модели на заданном корпусе текстов.
3. Реализовать модель генерации текста на основе GRU-ODE. Обучить модель на том же корпусе.
4. Изучить метрики качества для моделей генерации текста: BLEU, Rouge и Perplexity. Провести сравнение качества всех реализованных моделей.

С помощью фреймворка PyTorch были реализованы все упомянутые выше модели. Для обучения использовался корпус комментариев к статьям газеты New Your Times. Все модели показали хорошие значения метрик. Модель на базе GRU-ODE показала лучшую точность при меньшем числе итераций обучения.

В разделе 1.1 описан метод обучения резидуальных нейронных сетей с помощью решения системы сопряженных метод. В разделе 1.2 описано применение метода нейронных ОДУ для обучения управляемого рекуррентного блока (GRU). Раздел 1.3 содержит постановку задачи для модели генерации текста. В разделе 2 описан процесс реализации метода обучения резидуальных нейронных сетей с помощью решения сопряженной

системы уравнений, реализация GRU-ODE и модели генерации текста на классических рекуррентных блоках. Также представлены подготовка корпуса и обучение моделей. Приведены значения метрик для всех реализованных моделей. Во введении подведены итоги работы и сделаны основные выводы.

ОСНОВНАЯ ЧАСТЬ

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Нейронные дифференциальные уравнения

1.1.1. Резидуальные сети

В 2012 году Кижевским [3] была предложена архитектура глубокой сверточной сети с резидуальными связями ImageNet. Такая архитектура позволила решать задачу классификации изображений с достаточной точностью. Основу сети составляют резидуальные блоки, внутри которых используется резидуальная связь. Схематическое изображение блока представлено на (Рис 1.1).

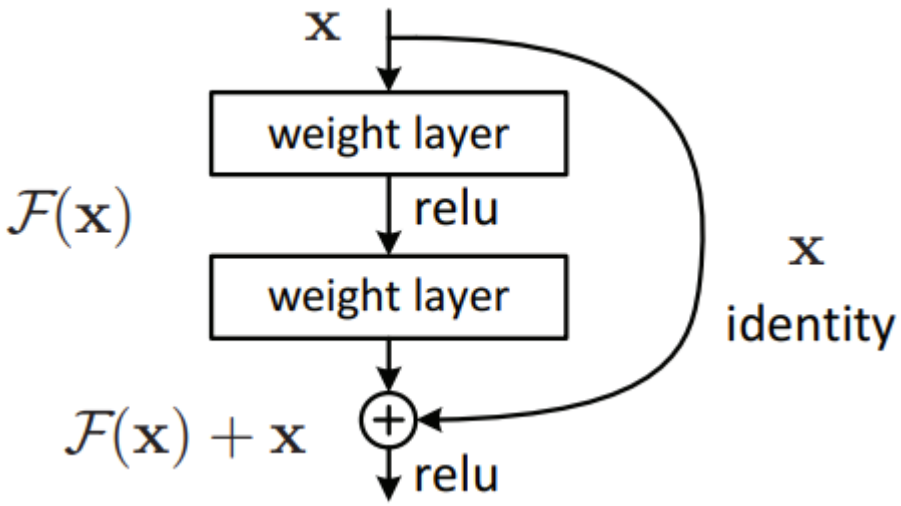

Рис. 1.1 Схематичное изображение резидуального блока сети ImageNet

В 2016 Кейминг [4] предложил улучшение архитектуры ImageNet – ResNet. Помимо изменения слоев внутри блока, была предложена концепция предактиваци, которая также положительно отразилась на качестве классификации. Описание блока приведено на (Рис 1.2).

При исследовании архитектуры ResNet выяснилось, что применение последовательности резидуальных блоков похоже на решение некоторого ОДУ методом Эйлера:

$$x_{i+1} = x_i + hf(x_i) \tag{1.1}$$

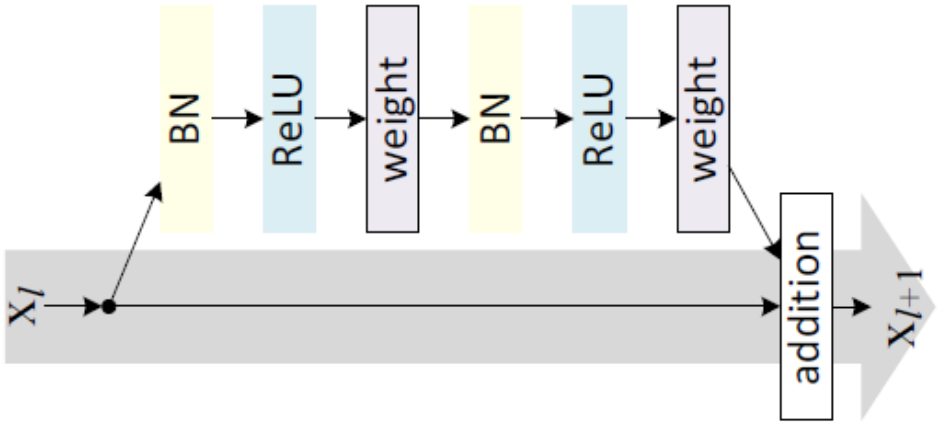

Рис. 3.2 Схематичное изображение резидуального блока сети ResNet

Под f понимается сам блок вместе с предактивацией. При дальнейшем изучении подобных моделей обнаружилась модель, использующая метод Рунге-Кутты пятого порядка точности – DenseNet [5]. Структура резидуального блока представлена на (Рис. 1.3). Блок представляет одну

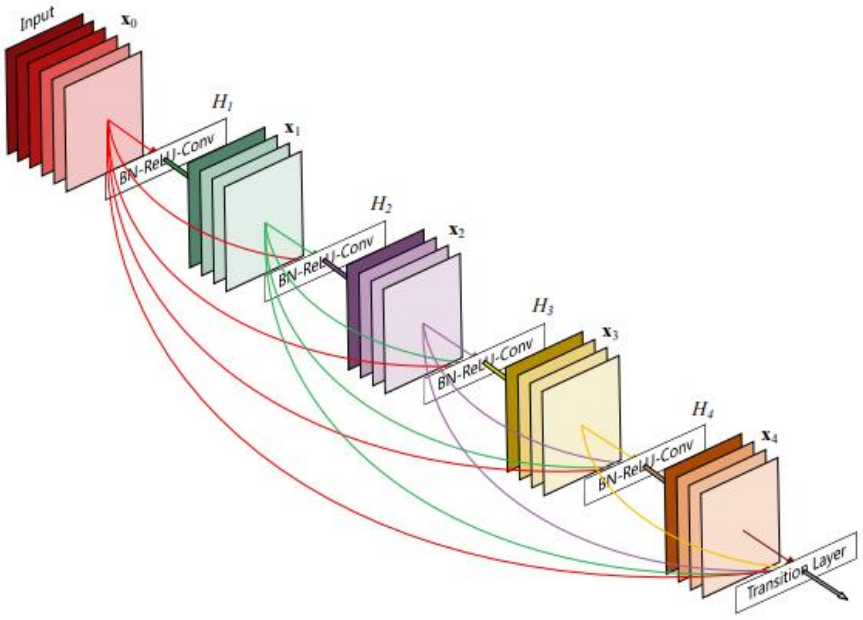

Рис. 2.3 Схематичное изображение резидуального блока сети DenseNet

итерацию метода Рунге-Кутты. Резидуальные связи позволяют последнему слою блока получить результат применения предыдущих слоев, что

эквивалентно вычислению четырех коэффициентов в методе Рунге-Кутты. Это натолкнуло авторов статьи [1] к созданию подхода нейронных ОДУ.

1.1.2. Обучение нейронных ОДУ

Резидуальные и рекуррентные используют сложные преобразования, основанные на последовательности преобразований скрытого состояния:

$$h_{t+1} = h_t + f(h_t, \theta_t), \tag{1.2}$$

где $t \in \{0, \dots, T\}$ и $h_t \in R^D$. Данный итерационный процесс можно рассматривать как применение метода Эйлера для решения некоторого ОДУ. Устремляя число итераций к бесконечности, получим

$$\frac{dh(t)}{dt} = f(h(t), t, \theta) \tag{1.3}$$

Для получения $h(T)$ при заданном $h(0)$ необходимо решить задачу Коши для этого ОДУ. Данная задача может быть решена с помощью любого решателя дифференциальных уравнений.

Для обучения нейронных сетей, использующих такое представления скрытого состояния, будем вычислять градиенты с помощью решения сопряженной системы уравнений (Adjoint sensitivity method). Будем использовать скалярную функцию потерь L, на вход которой будет подаваться результат решателя:

$$L(z(t_1) = L\left(z(t_0) + \int_{t0}^{t1} f(z(t), t, \theta)dt\right) = L\big(ODESolve(z(t_0), f, t_0, \theta)\big) \tag{1.4}$$

Для минимизации L нам необходимы градиенты по компонентам вектора весом θ. Определим, как градиент функции ошибки зависит от скрытого состояния $z(t)$. Введем сопряженную систему:

$$a(t) = \frac{\partial L}{\partial z(t)} \tag{1.5}$$

$$\frac{da(t)}{dt} = -a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial z} \tag{1.6}$$

$\frac{\partial L}{\partial z(t_0)}$ может быть найден решением ОДУ назад во времени. Сложность состоит в том, что для нахождения $z(t)$ систему требуется решать вперед. Это можно обойти, решая систему назад от значения $z(t_1)$. Таким образом, мы можем получить $z(t)$ и $a(t)$ за один вызов ODESolve.

Для нахождения градиентов функции ошибки, необходимо вычислить следующий интеграл, зависящий от $z(t)$ и a(t):

$$\frac{dL}{d\theta} = -\int_{t_0}^{t_1} a(t)^T \frac{\partial f(z(t), t, \theta)}{\partial \theta} dt \tag{1.7}$$

Произведения $\frac{a(t)^T \partial f}{\partial z}$ и $\frac{a(t)^T \partial f}{\partial \theta}$ в (1.6) и (1.7) могут быть эффективно вычислены с помощью автоматического дифференцирования. Искомые решения $z, a, \frac{\partial L}{\partial \theta}$ могут быть получены за один вызов решателя системы ОДУ. Подробный вывод данных формул можно найти в [1]. Схематически связь исходной и сопряженной системы представлена на (Рис. 1.4).

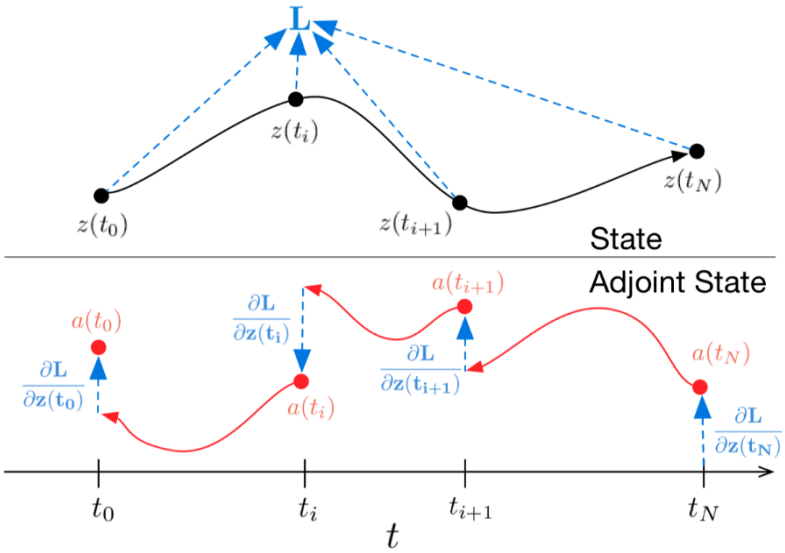

Рис. 4.4 Связь между исходной и сопряженной системой

Приведем описание алгоритма:

Вход: вектор весов θ, начальное время t_0, конечное время t_1, конечное состояние $z(t_1)$, $\frac{\partial L}{\partial z(t_1)}$

Алгоритм:

1. $s_0 = [z(t_1), \frac{\partial L}{\partial z(t_1)}, 0_{|\theta|}]$
2. $augdynamics([z(t), a(t), \ *], t, \theta] \rightarrow [f(z(t), t, \theta), -\frac{a(t)^T \partial f}{\partial z}, -\frac{a(t)^T \partial f}{\partial \theta}]$
3. $[z(t_0), \frac{\partial L}{\partial z(t_0)}, \frac{\partial L}{\partial \theta}] = ODESolve(s_0, augdynamics, t_1, t_2, \theta)$

Выход: $\frac{\partial L}{\partial z(t_0)}, \frac{\partial L}{\partial \theta}$

Теперь необходимо обсудить преимущества такого подхода. Во-первых, мы отказываемся от алгоритма обратного распространения ошибки. Он весьма ресурсоемкий, требует много дополнительной памяти. В данном подходе требуется только фиксированное количество памяти под веса θ и вспомогательная память для работы ODESolve. Это делает данный подход более экономичным в плане вычислительных ресурсов и потребления памяти, что является главной проблемой при использовании глубоких резидуальных и рекуррентных сетей. Во-вторых, замена итерационного процесса решением, связанным с ОДУ, позволяет заметно улучшить точность работы сети. Действительно, теперь мы не привязаны к методу решения ОДУ и можем использовать решатель с нужным порядком точности. Кроме того, наш резидуальный блок f теперь параметризован временем t, которое следует интерпретировать как глубину слоя. Так как количество шагов k метода по t задается решателем (в том числе динамически), то наш блок будет применен последовательно k раз. При этом коэффициенты блока одинаковы, но параметрически зависят от t. Таким образом, мы экономим память под коэффициенты θ.

1.2. Управляемый рекуррентный блок

1.2.1. Постановка задачи

Будем решать задачу прогнозирования на N спорадически наблюдаемых D-мерных временных рядах. Каждый ряд сэмплирован в K_i точках, определяемых вектором времени наблюдений $\mathrm{t}_i \in R^{K_i}$. Значения этих наблюдений представлены матрицей $y_i \in R^{K_i \times D}$ и маской наблюдений $m_i \in \{0,1\}^{K_i \times D}$ для определения соответствия значений и времени наблюдения.

Предположим, что наблюдения y_i получены как реализация D-мерного стохастического процесса $Y(t)$, динамика которого описывается неизвестным стохастическим дифференциальным уравнением:

$$dY(t) = \mu\big(Y(t)\big)dt + \sigma\big(Y(t)\big)dW(t), \tag{1.8}$$

где $dW(t)$ – Виннеровский процесс. Распределение $Y(t)$ изменяется согласно уравнению Фоккера-Планка. Среднее значения и ковариацию функции плотности вероятности $Y(t)$ будем обозначать $\mu_{Y(t)}$ и $\Sigma_{Y(t)}$.

Необходимо моделировать неизвестные временные функции $\mu_{Y(t)}$ и $\Sigma_{Y(t)}$ на основе спорадических измерений y_i. Для это будем параметризировать $\mu_{Y(t)}$ и $\Sigma_{Y(t)}$ с помощью нейронных сетей.

1.2.2. GRU

Для задачи генерации текста одной резидуальной сети недостаточно. Требуется обобщить поход нейронных ОДУ к рекуррентным сетям, а именно GRU блокам.

Пусть дана последовательность x_t. Применение GRU блока к этой последовательности можно описать следующими уравнениями:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{1.9}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{1.10}$$

$$g_t = tanh(W_h x_t + U_h(r_t \times h_{t-1}) + b_h) \quad (1.11)$$

где × - поэлементное умножение векторов, σ – сигмоидальная функция, $W_r, W_z, W_h, U_r, U_z, U_h$ - матрицы весов, b_r, b_z, b_h - вектора весов, h_{t-1} – скрытое состояние, соответствующее элементу последовательности x_{t-1}. Обновление скрытого состояния осуществляется по формуле

$$h_t = z_t \times h_{t-1} + (1 - z_t) \times g_t, \quad (1.12)$$

что также можно записать в виде

$$h_t = GRU(h_{t-1}, x_t) \quad (1.13)$$

1.2.3. Адаптация GRU блока для подхода нейронных ОДУ

Для применения подхода нейронных ОДУ, необходимо получить ОДУ вида (1.3). Для этого запишем разность h_t и h_{t-1}:

$$\Delta h_t = z_t \times h_{t-1} + (1 - z_t) \times g_t - h_{t-1} = (1 - z_t) \times (g_t - h_{t-1}) \quad (1.14)$$

Из данной записи естественным образом следует соответствующее ОДУ:

$$\frac{dh(t)}{dt} = \big(1 - z(t)\big) \times \big(g(t) - h(t)\big) \quad (1.15)$$

Результирующая система носит название ODE-GRU [2].

GRU-ODE обладает несколькими полезными свойствами. Во-первых, это ограниченность скрытого состояния. В [2] показано, что $h(t) \in R^n_{[1,-1]}$, что позволяет применить подход GRU-Bayes. Во-вторых, GRU-ODE является липшицевым отображением [6] с константой К = 2. Это означает, что GRU-ODE кодирует непрерывную априорную вероятность для скрытого процесса $h(t)$. В-третьих, как и нейронные ОДУ, GRU-ODE может быть проинтегрирована с помощью любого решателя ОДУ. Предпочтительнее использовать решатели с адаптивным шагом (например, метод Дорманда-Принца [7]).

1.3. Модели генерации текста

1.3.1 Постановка задачи

Существует два типа задачи генерации текста: генерация на основе нетекстовых данных (data-to-text) и текстовых (text-to-text). В качестве примера для первого типа можно привести получение текстового описания изображений. Для второго типа наиболее важными примерами является перевод текста и генерация ответа по текстовому вопросу пользователя.

В данной работе будет использоваться генерация теста второго типа. Генерация будет производиться на основе корпуса текстов K. Модель M будет использовать ввод пользователя I как стартовую последовательность генерации. Ввод может состоять из одного слова или из нескольких предложений. Далее на основе стартовой последовательности будет сгенерирована выходная последовательность O желаемой длины: $I \rightarrow M(K) \rightarrow O$. С помощью подобных моделей можно, например, копировать стилистику авторского текста. Данная модель обучена на корпусе комментариев к статьям газеты New York Times. Далее будет подробно описан процесс обучения модели.

1.3.2 Предобработка корпуса

Корпус был очищен от нежелательных символов, приведен к нижнему регистру и разбит на слова. Был составлен список частотности слов. 15% самых часто встречаемых слов были удалены.

Далее из уникальных слов текста был составлен словарь. Каждому слову в словаре соответствует его числовой индекс. Весь текст был заменен на последовательность индексом слов.

1.3.3 Модель N-грамм

Рассмотрим последовательность из N слов. Модель генерации текста должна для заданной последовательности слов $w_{i-N+1}, \dots, w_{i-1}$ определять вероятность следующего слова w_i:

$$P(w_i | w_{i-1}, \dots, w_{i-N+1}) \tag{1.16}$$

Для этого будем использовать нейронную сеть со следующей архитектурой (Рис. 1.5):

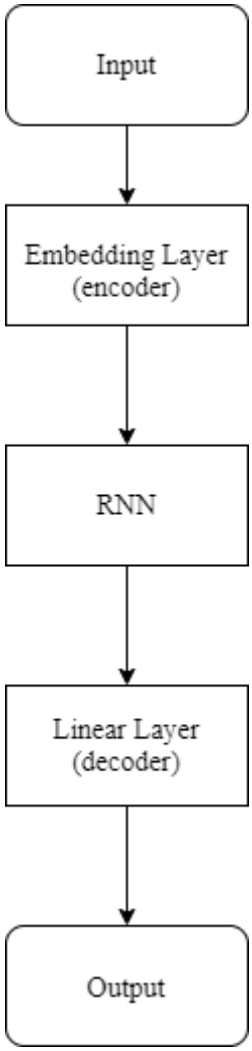

Рис. 5.5 Архитектура нейронной сети для генерации текста

Последовательность индексов слов была преобразована в последовательность N-грамм. В данном случае использовалось N равное двум.

Последовательность N-грамм передавалась в слой эмбеддингов. Данный слой необходим для модели. С его помощью сразу обучаются векторные представления слов. Предобученные векторные представления не использовались из-за ухудшения качества модели.

Следующий слой – любая рекуррентная ячейка. В оригинальной модели это управляемый рекуррентный блок (GRU). Данный слой обучается для данной последовательности слов выдавать искомую вероятность следующего слова.

Последний – полносвязный линейный слой. Используются в качестве декодировщика, преобразовывающего внутренние векторный представления слов в индексы.

В качестве функции ошибки использовалась кросс-энтропия для многоклассового случая (число классов равно мощности словаря):

$$L = -\sum_{c=1}^{M} y_{o,c}\, log(p_{o,c}) \tag{1.17}$$

1.3.4 Генерация текста

Для генерации текста используется некоторая стартовая последовательность слов s_i. Выбор слова осуществляется с помощью семплирования из мультиномиального распределения:

$$w_{index} = multinomial\left(\exp\left(\frac{X}{T}\right)\right) \tag{1.18}$$

Величина T называется температурой. Данный параметр является гиперпараметром модели и должен быть найден с помощью соответствующего алгоритма.

1.3.5 Метрики качества генерации текста

Метрики качества для генерации текста в первую очередь ориентированы на машинный перевод. Метрика BLEU не исключение. Для получения оценки метрики подают эталонный и сгенерированный текст. Для задачи машинного перевода это означает сравнение ручного и машинного перевода. Далее алгоритм сравнивает наличие в сгенерированном тексте слов из эталона. Результат метрики – отношение числа входящих в эталонный текст слов и число слов в сгенерированном тексте. В данной работе в качестве эталонных текстов использовалась часть корпуса, на которой не обучалась модель.

Следующая используемая метрика – ROUGE. Это акроним от Recall-Oriented Understudy for Gisting Evulation. Это метрика использует число верных ответов (precision) и полноту (recall). Для сгенерированного и эталонного текста вычисляются последовательности N-грамм (часто биграмм), между которыми вычисляются число верных ответов и полнота. Далее результат объединяется с помощью F_1 меры:

$$F_1 = \frac{2(precision * recall)}{precision + recall} \tag{1.19}$$

Последняя метрика – перплексия (perplection). Для ее вычисления необходимо вычислить вероятности каждого слова в последовательности с учетом предыдущих слов. Обозначим их P_i. Тогда перплексия определяется по формуле

$$PP = (P_1 * \dots * P_N)^{-\frac{1}{N}} \tag{1.20}$$

Как можно видеть, при идеальной генерации ее значение равно 1.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Используемое программное обеспечение

Все описываемые в работе модели были реализованы на языке программирования Python версии 3 с использованием фреймворка PyTorch. Язык Python широко используется среди разработчиков моделей машинного обучения. Причин этому несколько. Во-первых, это независимость от платформы. Python доступен на всех популярных архитектурах: Windows, MacOS и GNU\Linux. Помимо этого Python обеспечивает полную переносимость кода между платформами, что часто позволяет ускорить внедрение реализованных моделей. Во-вторых, это простота и стабильность. Синтаксис языка максимально прост. Вместе с интерпретатором распространяется библиотека, в которой можно найти многие структуры данных и алгоритмы. После релиза третей версии Python все вносимые в синтаксис языка изменения являются обратно совместимыми, что позволяет использовать старый код без каких-либо исправлений. В-третьих, это наличие большого количества библиотек и фреймворков. Разработчики также распространяют пакетный менеджер pip и поддерживают репозиторий пакетов PyPi.

За эти качества в машинном обучении отдается предпочтение этому языку программирования. Большинство библиотек были изначально ориентированы на Python. Для остальных были реализованы обертки, позволяющие вызывать код библиотек из Python.

Стоит отдельно отметить, что Python исполняет код достаточно медленно. Так как в машинном обучении активно используются различные вычисления, сообществу пришлось придумать библиотеку для их ускорения. Эта библиотека – Numpy. Numpy позволяет выполнять векторизированные операции над векторами и матрицами. Такие операции выполняются крайне

быстро, так как в основе Numpy лежит ядро на языке C++, которое выполняет эти операции в многопоточном режиме на CPU.

Как известно, для обучения нейронных сетей требуется большое количество вычислительных ресурсов. Кажется, что Python точно не должен подходить для этого. Однако, появление фреймворков для построения и обучения нейронных сетей не заставило себя ждать. В них используется подход, аналогичных Numpy – ядро на C++. Это позволило не только обучать нейронные сети, но и делать это эффективно с использованием GPU.

Для разработки GRU-ODE и модели генерации текста был выбран фреймворк PyTorch. Предварительно было произведено сравнение данного фреймворка и Tensorflow. PyTorch показал лучший дизайн API. Также его гораздо удобнее расширять. Главным преимуществом стало меньшее время обучения глубоких сетей.

Весь код был выполнен в среде разработки Jupyter, которая позволяет сильно ускорить разработку и упростить получение информации о качестве обучения моделей. Кроме того, среда может быть запущенна на удаленном сервере, что позволяет обучать модели без привязки к аппаратному обеспечению.

Для хранения кода использовалась система Git и интернет хостинг для совместной разработки GitHub. Это позволило реализовать удобное версионирование кода.

2.2. Реализация тестовой модели ResNet

В первую очередь была реализована тестовая модель резидуальной сети, состоящей из одного ResNet блока. Это позволило проверить корректность метода обучения нейронных сетей с помощью сопряженных уравнений. Сеть обучалась для решения задачи классификации на наборе изображений MNIST. В качестве функции ошибки использовалась кросс-энтропия.

Для сравнения качества обучения была также реализована модель, использующая классическое обучение с помощью обратного распространения ошибки. Обе модели обучались в течение пяти эпох. Значения функции ошибки и точность классификации на тестовой выборке приведены в (Таблице 2.1).

Таблица 2.1. Качество обучения моделей ODE-Net и ResNet

Модель	L_1	L_2	L_3	L_4	L_5	Точность
ODE-Net	0.157	0.050	0.037	0.030	0.025	99.01%
ResNet	0.213	0.078	0.062	0.052	0.046	98.81%

Из значений таблицы (Таблицы 2.1) видно, что обучение модели с помощью метода сопряженных уравнений происходит быстрее. Это достигается за счет более точного вычисления градиентов функции ошибки в случае метода сопряженных уравнений. Точность классификации также подтверждает это.

2.3. Реализация GRU-ODE

Рекуррентная ячейка GRU-ODE была также реализована на основе фреймворка PyTorch. Метод обучения был взят из тестовой модели. Реализовано обновление внутреннего состояния с помощью сопряженных уравнений.

2.3. Реализация модели генерации текста

Модель генерации текста включает три слоя: слой эмбеддингов, рекуррентный слой, полносвязный слой. Слой эмбеддингов необходим для обучения векторных представлений слов. Такой подход позволяет получить

эмбеддинги лучшего качества по сравнению, например, с Word2Vec подходом, так как учитывается смысл слов в обучающем корпусе. Недостатком является больше время обучения модели.

Рекуррентный слой необходим для получения вероятности следующего слова по последовательно предыдущих слов. Он обрабатывает последовательности векторных представлений слов, полученных с помощью слоя эмбеддингов. Качество генерации текста полностью определяется этим слоем. На выходе для каждой последовательности получаем вектор, имеющий размерность скрытого состояния рекуррентного слоя.

Выход рекуррентного слоя не может быть напрямую использован для получения следующего слоя последовательности. Предварительно он должен быть декодирован. Для этого необходим полносвязный слой, который выступает в роли декодера. Этот слой будет обучаться преобразовывать выходные векторы рекуррентного слоя в вероятности класса слов.

Было реализовано четыре модели, различающиеся типом рекуррентного слоя: vanilla RNN, GRU, LSTM и GRU-ODE. Классические модели были использованы в качестве базовых для сравнения с GRU-ODE. Результаты обучения и качества генерации текста всех четырех моделей приведены в следующем разделе.

2.4. Обучение моделей

В качестве корпуса для обучения использовались комментарии к статьям газеты New York Times. Модель генерации текста с GRU-ODE сравнивалась с классическими рекуррентными ячейками: RNN, LSTM и GRU.

В качестве алгоритма оптимизации использовался Adam (adaptive moment estimation), который является улучшением алгоритма RMSProp. Adam использует адаптивный шаг для каждого параметра. Кроме того, в методе используются первые и вторые начальные моменты градиентов, что позволяет ускорить сходимость алгоритма. При обучении использовался шаг равный 0.015. Было проведено 300 эпох обучения. В качестве функции ошибки

использовалась кросс-энтропия. Графики ошибки для модели с RNN, GRU, LSTM и GRU-ODE приведены на рисунках (Рис. 2.1), (Рис. 2.2), (Рис. 2.3) и (Рис. 2.4) соответственно.

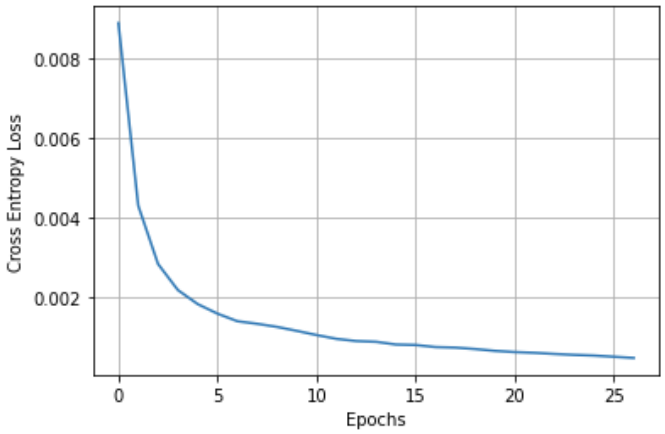

Рис. 2.1 График ошибки для модели с RNN

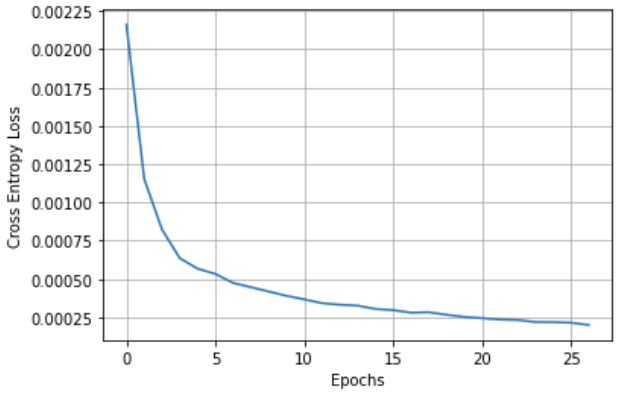

Рис. 2.2 График ошибки для модели с GRU

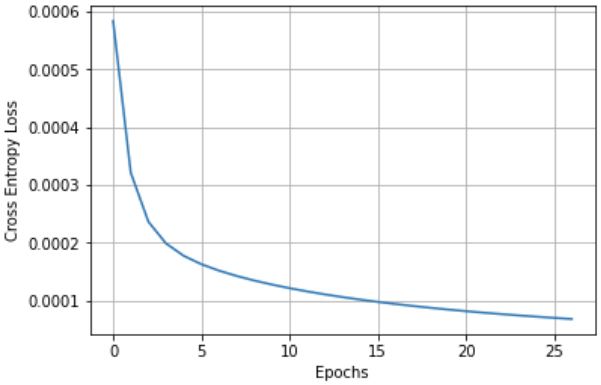

Рис. 2.3 График ошибки для модели с LSTM

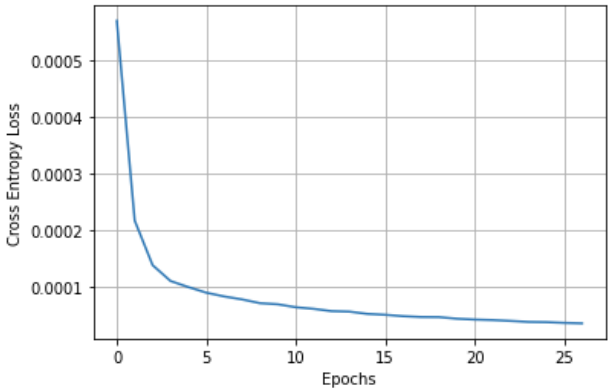

Рис. 2.4 График ошибки для модели с GRU-ODE

Как можно видеть данных графиков, модель на основе GRU-ODE учится быстрее всех остальных. Также данная модель достигла наименьшей ошибки. Характер график для остальных моделей вполне соответствует ожиданиям. Модель с RNN показала худший результат ввиду недостаточной связи между слоями. GRU ячейка решает данную проблему. В LSTM присутствует механизм забывания, что делает модель лучшей среди классических архитектур.

2.5. Значения метрик

Для контроля качества генерируемого текста были выбраны метрики BLEU, ROUGE и перплексия. Вместе они позволяют адекватно оценить

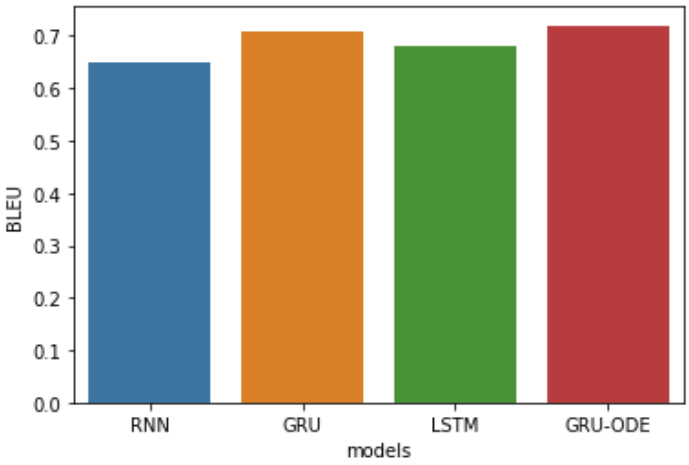

Рис. 2.5 Значения метрики BLEU

качество генерации текста. Метрики BLEU и ROUGE имеют диапазон значений $[0, 1]$, при лучшем значении равным единице. Диапазон значений перплексии – $[1, +\infty]$ при лучшем значении 1. Графики метрик приведены на рисунках (Рис. 2.5), (Рис. 2.6) и (Рис. 2.7).

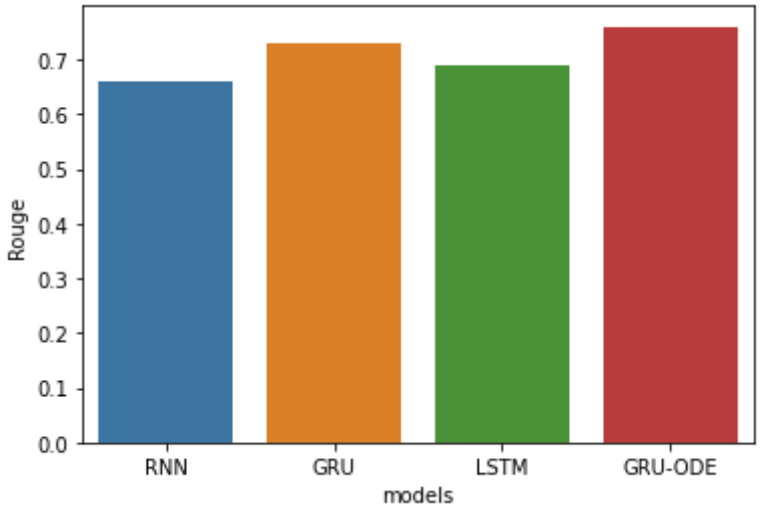

Рис. 2.6 Значения метрики ROUGE

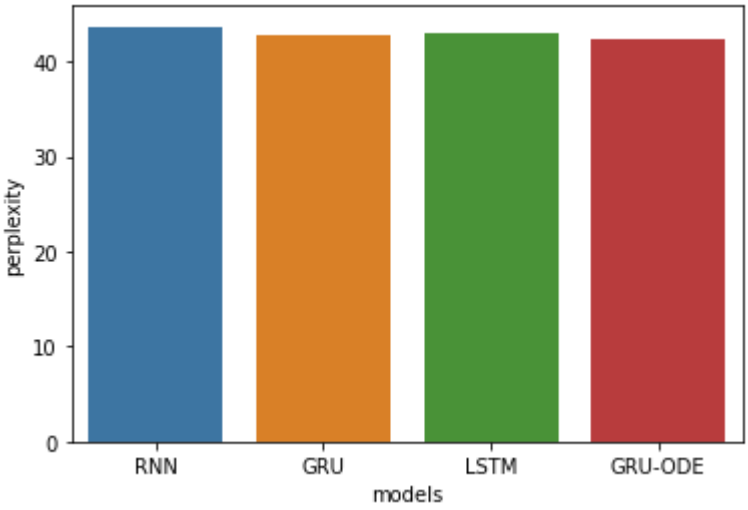

Рис. 2.7 Значения перплексии

Из графиков метрик видно, что модель с GRU-ODE имеет лучшие показатели по метрикам BLEU и ROUGE. Значения перплексии для всех трех моделей примерно одинаковы. Отсюда можно сделать вывод, что перплексия не очень хорошо подходит для оценки качества генерации текста.

Лучшее качество генерации текста обусловлено более точным вычислением градиентов функции ошибки методом сопряженных уравнений по сравнению с классическим обратным распространением ошибки.

ЗАКЛЮЧЕНИЕ

Нестандартные подходы к обучению нейронных сетей потенциально могут иметь множество приложений. Например, после рассмотрения резидуальных сетей как нейронных ОДУ были предложены сети, структура которых основывается на более точных численных схемах. Такие архитектуры показали большую точность по сравнению с классическими.

Напротив, иногда изменение метода обучения позволяет улучшить результаты текущей модели. В [1] описано применение нейронных ОДУ к модели нормализованных потоков. Это генеративная модель, являющаяся альтернативой генеративно-состязательных сетей. При использовании метода обучения с помощью сопряженной системы данная модель показала очень хорошие результаты в задаче генерации произвольных реалистичных лиц.

В данной работе была исследована возможность применения метода обучения нейронных сетей с помощью сопряженных уравнений в задачах обработки естественных языков. В этой области рекуррентные сети используются повсеместно. Такие модели учатся медленно за счет большого числа переменных. Метод сопряженных уравнений в этой области выглядит наиболее привлекательно, так позволяет сократить число эпох обучения сети.

В качестве базовой модели для сравнения была выбрана модель генерации текста. Несмотря на относительную простоту модели, она позволяет полностью оценить пользу применение метода сопряженных уравнений для задач обработки естественных языков.

В первую очередь метод сопряженных уравнений был реализован для классической резидуальной сети ResNet. Модель классифицировала рукописные цифры из набора изображений MNIST. Метод показал улучшение скорости обучения сети за счет более точного вычисления градиента функции ошибки. После верификации корректности работы метода сопряженных

уравнений была начата его реализация для управляемого рекуррентного блока.

Метод сопряженных уравнений был адаптирован для обучения управляемого рекуррентного блока (GRU). GRU слой был протестирован отдельно в задаче обработки временного ряда на искусственном наборе данных. Также было отмечено ускорение обучение модели.

Далее была реализована модель генерации текста на основе классических рекуррентных слоев (GRU, LSTM, RNN) и реализованного GRU-ODE. Все модели были обучены на корпусе комментариев к статьям газеты New York Times. Проведено сравнение значения функции ошибки на различных эпохах и метрик качества генерации текста. Модель, обученная с использованием метода сопряженных уравнений показала улучшение скорости обучения модели по сравнению с классическим GRU слоем.

Метод обучения нейронных сетей с помощью сопряженных уравнений показал отличные результаты при использовании в модели генерации текста. Улучшение точности вычисления градиентов функции ошибки позволило сократить число эпох обучения при том же значении качества генерации текста. Данный результат позволяет обобщить применимость метода на другие архитектуры моделей обработки естественных языков, использующие рекуррентные слои.

Также были найдены способы улучшения текущих результатов. В первую очередь это реализация метода сопряженных уравнений на C++ с использованием API LibTorch. Это позволит ускорить выполнение метода сопряженных уравнений.

Выбор решателя системы ОДУ также позволит улучшить качество обучения. Следует провести сравнение качества обучения для различных решателей. Наиболее перспективными являются методы с адаптивным шагом, например метод Дорманда-Принца [7]. Также в литературе не описано применение решателей, использующих неявные методы. Успешное

применение подобных решателей позволит еще сильнее улучшить качество обучения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, David Duvenaud. Neural Ordinary Differential Equations, 2019

[2] Edward De Brouwer, Jaak Simm, Adam Arany, Yves Moreau. GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series, 2019

[3] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks, 2019

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks, 2016

[5] Gao Huang, Zhuang Liu, Laurens van der Maaten. Densely Connected Convolutional Networks, 2018

[6] Федерер Г. Геометрическая теория меры. – 1987. – 760 с.

[7] Dormand, J. R., Prince, P. J., A family of embedded Runge-Kutta formulae, Journal of Computational and Applied Mathematics, 6 (1): 19–26, 1980