УДК 004.89

Мурадян Д. С.

Реализация и внедрение архитектуры Gated Recurrent Unit в пакет нейросетевой аппроксимации дифференциальных уравнений DEGANN

Рекомендовано к публикации доцентом Гориховским В. И.

1. Введение. Дифференциальные уравнения играют ключевую роль в самых разных областях науки и техники. Решение таких уравнений – задача, довольно трудная, требующая больших вычислений. С усложнением и увеличением размерности задачи – традиционные численные методы решения дифференциальных уравнений начинают сталкиваться с проблемами, связанными с длительным временем вычислений, недостаточной точностью и так далее.

Это заставляет задуматься об альтернативных способах решения таких задач. Одним из таких способов является использование нейронных сетей, которые могут аппроксимировать сложные функции, описывающие дифференциальные уравнения. Такой подход используется в пакете нейросетевой аппроксимации дифференциальных уравнений DEGANN.

Функционал этого пакета ограничен одним типом нейронной сети – многослойным перцептроном (MLP – Multi-Layer Perceptron). Этот тип может быть недостаточно эффективен в некоторых задачах аппроксимации дифференциальных уравнений, так как MLP не способен выявлять последовательные зависимости в данных, в отличие от рекуррентных нейронных сетей (RNN – Recurrent Neural Network), специально предназначенных для работы с последовательностями, которые очень важны в анализе нелинейных дифференциальных уравнений.

Мурадян Денис Степанович – студент бакалавриата, Санкт-Петербургский государственный университет; e-mail: muradyan.denis@inbox.ru

Гориховский Вячеслав Игоревич – доцент, Санкт-Петербургский государственный университет; e-mail: gorihovskyvyacheslav@gmail.com

Однако RNN имеют известную проблему затухания и взрыва градиента, из-за чего обучение становится затруднительным на длинных последовательностях. Архитектура GRU – (Gated Recurrent Unit) является усовершенствованной версией RNN и частично решает эту проблему благодаря использованию элементов управления потоком данных.

Внедрение этой архитектуры в пакет DEGANN должно помочь оптимизировать и улучшить качество аппроксимации дифференциальных уравнений.

2. Реализация и внедрение.

2.1. Реализация.

Для реализации архитектуры GRU в пакете DEGANN разработан класс `TensorflowGRUNet`, наследуемый от `tf.keras.Model`. Сначала создаётся список GRU-слоёв с заданными функциями активации (основная — *tanh*, рекуррентная — *sigmoid*) и инициализаторами весов и смещений. Последовательная обработка входных данных этими слоями передаётся в полносвязный выходной слой с линейной активацией, что обеспечивает регрессионное предсказание.

Логика компиляции модели выполняется с помощью реализованного метода `custom_compile` — он отвечает за настройку оптимизатора, функции потерь и метрик и так далее, а также метод `to_dict`, экспортирует конфигурацию модели в виде словаря для дальнейшей интеграции с другими компонентами DEGANN. В функции `call` реализован прямой проход через нейросеть, а вычисление градиентов производится стандартными средствами TensorFlow. Разработанная оболочка обеспечивает унифицированный интерфейс для работы модели в системе.

Основным элементом данной реализации является класс `TensorflowGRUNet`, который наследуется от `tf.keras.Model`. Он отвечает за:

- Построение и инициализацию GRU-слоёв, каждый из которых имеет число нейронов (определяемое параметром **block_size**),
- Настройку ключевых параметров:
  - **input_size** — размер входных данных (количество признаков);
  - **output_size** — размер выхода;

  - **activation_func** и **recurrent_activation** — функции активации для основного и рекуррентного состояний (по умолчанию *tanh* и *sigmoid*);
  - **weight** и **biases** — инициализаторы весов и смещений.

- Организацию прямого прохода данных через сеть с помощью метода `call`. В этом методе входные данные последовательно обрабатываются всеми GRU-слоями, а затем результат передается в выходной полносвязный слой с линейной активацией, что оптимально для регрессионных задач.

2.2. Внедрение.

Интеграция архитектуры GRU в DEGANN осуществляется посредством параметра `net_type`, передаваемого при создании модели через базовый класс `IModel`. При установке `net_type="GRUNet"` внутри `IModel` вызывается метод, создающий экземпляр класса `TensorflowGRUNet`. Такой подход позволил сохранить единообразный интерфейс работы с моделями различных типов (например, с DenseNet или другими), не нарушая общую модульную структуру проекта.

Особое внимание уделено совместимости новой реализации с существующей инфраструктурой DEGANN. Это обеспечивает возможность использования стандартных методов обучения, тестирования и экспорта моделей. Например, метод `ToDict` позволяет сохранить ключевые параметры модели (тип сети, количество слоёв, число нейронов в каждом слое, размер выхода) в виде словаря для последующей интеграции с другими компонентами системы.

3. Валидация.

3.1. Параметры эксперемента.

В данном разделе представлено экспериментальное сравнение эффективности двух архитектур, реализованных в DEGANN: GRUNet (на основе рекуррентных слоёв GRU) и DenseNet (традиционный многослойный перцептрон, MLP). Цель эксперимента — продемонстрировать, что при одинаковых гиперпараметрах (5 слоёв, 30 нейронов в каждом, функция потерь MSE, метрики MAPE, R^2) и фиксированном времени обучения архитектура GRU обеспечивает более точное приближение целевых функций за счёт своей способности учитывать последовательные зависимости в данных.

Конфигурация тестирующей системы.

- Процессор: Intel i5 1240p
- Видео-чип: Iris Xe Graphics G7 80EUs
- ОЗУ: 16 гб

Тестовые функции. Для проверки моделей используются следующие функции, встроенные в DEGANN:

- $f_{\text{sin}}(x) = \sin(10x)$,
- $f_{\text{hyperbol}}(x) = \dfrac{x^2 + 0.5}{x + 0.1}$,
- $f_{\text{hardsin}}(x) = \sin\Big(\ln\big(x^{\sin(10x)}\big)\Big)$.

Обоснование выбора тестовых функций. Выбранные функции являются встроенными в DEGANN и представляют собой достаточно сложные для аппроксимации зависимости. Функция $f_{\text{sin}}(x) = \sin(10x)$ демонстрирует периодическую нелинейность, что позволяет проверить способность модели захватывать колебательные процессы. Функция $f_{\text{hyperbol}}(x) = \dfrac{x^2 + 0.5}{x + 0.1}$ обладает неоднородной зависимостью, где квадратичный рост сочетается с особенностями в малых значениях x, что требует от модели адаптивного поведения в различных областях определения. Функция $f_{\text{hardsin}}(x) = \sin\Big(\ln\big(x^{\sin(10x)}\big)\Big)$, включающая логарифмическое преобразование и экспоненциальные эффекты, ещё более усложняет задачу аппроксимации за счёт усиленной нелинейности. Такой набор тестовых функций позволяет всесторонне оценить эффективность сравниваемых архитектур, поскольку каждая из них предъявляет уникальные требования к способности модели моделировать сложные, неоднородные зависимости.

Параметры: Для каждой архитектуры эксперименты проводятся при фиксированном времени обучения. В частности, результаты тестирования собираются для трёх вариантов времени обучения: 15, 30 и 45 секунд. При каждом варианте фиксируются следующие показатели:

- **Функция потерь:** Mean Squared Error (MSE),
- **Метрики:** Mean Absolute Percentage Error (MAPE) и коэффициент детерминации (R^2)

- Дополнительно, фиксируются затраты оперативной памяти и общее время обучения.

Таким образом, основная задача эксперимента состоит в сравнении показателей качества аппроксимации для GRUNet и DenseNet при идентичных условиях обучения. При этом важно отметить, что использование одинаковых гиперпараметров позволяет провести объективное сравнение, где разница в результатах обусловлена внутренней структурой моделей, а не различиями в настройках.

3.2. Результаты эксперемента.

Ниже приведены три таблицы, демонстрирующие структуру представления результатов эксперимента для каждого из выбранных интервалов времени обучения.

Таблица 1. Результаты эксперимента (время обучения: 15 секунд)

Архитектура	**Тестовая функция**	**Эпох**	**Память (MB)**	**Loss MSE**	**MAPE**	R^2
GRUNet	hyperbol	155	112.460	0.002	1.536	0.993
GRUNet	sin	155	111.360	0.003	4.149	0.994
GRUNet	hardsin	155	112.450	0.045	14.647	0.738
DenseNet	sin	215	35.000	0.091	23.095	0.812
DenseNet	hardsin	215	41.770	0.017	10.100	0.902
DenseNet	hyperbol	215	33.990	0.051	13.346	0.819

Таблица 2. Результаты эксперимента (время обучения: 30 секунд)

Архитектура	**Тестовая функция**	**Эпох**	**Память (MB)**	**Loss MSE**	**MAPE**	R^2
GRUNet	hardsin	330	112.62	0.004	4.008	0.976
GRUNet	hyperbol	330	116.27	0.005	3.508	0.980
DenseNet	sin	445	37.52	0.012	7.509	0.975
DenseNet	hyperbol	445	36.95	0.011	5.989	0.961
DenseNet	hardsin	445	44.78	0.005	5.101	0.971
GRUNet	sin	345	115.39	0.003	4.169	0.994

Таблица 3. Результаты эксперимента (время обучения: 45 секунд)

Архитектура	**Тестовая функция**	**Эпох**	**Память (MB)**	**Loss MSE**	**MAPE**	R^2
GRUNet	sin	450	120.59	0.003	4.101	0.995
GRUNet	hardsin	470	116.84	0.002	3.158	0.987
GRUNet	hyperbol	450	117.79	0.002	1.174	0.992
DenseNet	sin	530	38.98	0.007	7.339	0.985
DenseNet	hyperbol	600	47.98	0.004	3.138	0.987
DenseNet	hardsin	600	47.71	0.006	5.741	0.962

3.3. Вывод.

Анализ проведённых экспериментов демонстрирует, что архитектура GRUNet при идентичных гиперпараметрах и фиксированном времени обучения обеспечивает более высокое качество аппроксимации целевых функций по сравнению с DenseNet. По метрикам MAPE и коэффициенту детерминации (R^2) наблюдается стабильное преимущество GRUNet, что указывает на его лучшую способность учитывать последовательные зависимости в данных.

Несмотря на несколько более высокие затраты памяти, разница в потреблении ресурсов (порядка 80 МБ) не является значительной для пользователей в большинстве практических конфигураций, в отличие от времени. GRUNet демонстрирует более быструю сходимость, обеспечивая лучшее качество аппроксимации за меньшее количество эпох. Это делает данную архитектуру перспективной для задач аппроксимации дифференциальных уравнений в рамках пакета DEGANN, где важна эффективность обучения при сохранении высокой точности результатов.

Литература

1. Пакет нейросетевой аппроксимации дифференциальных уравнений // DEGANN URL: https://github.com/Krekep/degann (дата обращения: 10.03.2025).
2. Goodfellow I., Bengio Y., Courville A. Deep Learning. 1-е изд. Кембридж,Массачусетс, США: MIT Press, 2016. 800 с.
3. SStryker C. What is a Recurrent Neural Network? // IBM . 2024.
4. Документация к пользованию tensorflow // TFnet URL: https://tensorflownet.readthedocs.io (дата обращения: 04.03.2025).
5. Geron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 3-е изд. Себастопол, Калифорния, США: O'Reilly Media, Inc., 8.11.2022. 856 с.