



Санкт-Петербургский государственный университет
Кафедра информатики

Реализация и внедрение архитектуры Gated Recurrent Unit в пакет DEGANN

Мурадян Денис Степанович, группа 23.Б16-мм

Научный руководитель: доцент кафедры системного программирования, к.ф.-м.н., Гориховский В. И.
Консультант: Преподаватель СПбГУ, Алимов П. Г.

Санкт-Петербург
2024

- Дифференциальные уравнения широко применяются в науке и технике
- Традиционные численные методы сталкиваются с проблемами, связанными с длительным вычислением и недостаточной точностью
- Библиотека для нейросетевой аппроксимации дифференциальных уравнений
- Цель работы: интеграция архитектуры GRU в DEGANN

- **RNN (Recurrent Neural Networks):**

- ▶ Преимущества: используют скрытые состояния для учёта временных зависимостей.
- ▶ Проблемы: затухание и взрыв градиентов, затрудняющие обучение на длинных последовательностях.

- **GRU (Gated Recurrent Unit):**

- ▶ обновляющий (z_t) и сбрасывающий (r_t) гейты:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z), \quad r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

- ▶ Новое состояние:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t, \quad \tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

- **Преимущества GRU:**

- ▶ Сохраняет зависимость в последовательности данных
- ▶ Частично решает затухание градиентов, учитывая долгосрочные зависимости.
- ▶ Компактная структура.

- **Использование TensorFlow:**

- ▶ Для реализации GRU использовался встроенный слой `tf.keras.layers.GRU`.
- ▶ Слой предоставляет механизмы управления потоками данных:
 - ★ Обновляющий и сбрасывающий гейты.
 - ★ Поддержка скрытых состояний.
- ▶ Интеграция GRU с TensorFlow позволяет сосредоточиться на настройке модели и анализе её поведения, а так же настройке гиперпараметров.

- **Существующая проблема:**

- ▶ Текущая версия DEGANN имеет только полносвязные сети, что ограничивает возможности аппроксимации сложных функций.

- **Выводы:**

- ▶ GRU может помочь в аппроксимации для дифференциальных уравнений, где каждое значение функции связано с предыдущими.
- ▶ Архитектура GRU решает проблему затухания градиентов, присущую классическим RNN.
- ▶ TensorFlow предоставляет гибкие инструменты для разработки и ускоряет процесс интеграции.

Постановка задачи

Целью работы является реализация и внедрение рекуррентных нейронных сетей с архитектурой GRU в пакет DEGANN

Задачи:

- Общая архитектура решения
 - ① Реализация модуля с топологией рекуррентной сети со слоями tensorflow для архитектуры GRU и внедрение его в пакет DEGANN.
 - ② Создание модели с архитектурой GRU. Создание и интеграция callback-функции для расширения функциональности обучения и его трекинга.
- Валидация решения
 - ① Создание датасета для корректной передачи данных в модель. Реализация сложно-аппроксимируемой функции для тестирования.
 - ② Подбор метрики, настройка гиперпараметров.

Реализация модуля с топологией рекуррентной сети GRU

- **Класс `TensorflowGRUNet`:**

- ▶ Наследуется от `tf.keras.Model`.
- ▶ Реализует логику построения и инициализации GRU-слоёв.
- ▶ Поддерживает настройку параметров, таких как `input_size`, `block_size`, `output_size`, `dropout_rate`.

- **Интеграция в `DEGANN`:**

- ▶ Через параметр `net_type="GRUNet"` в `IModel`.
- ▶ Единообразный интерфейс для различных типов сетей (`DenseNet`, `GRUNet`).

- **Принципы работы:**

- ▶ Все слои GRU имеют одинаковое количество нейронов.
- ▶ После каждого слоя применяется `Dropout` для предотвращения переобучения.
- ▶ Функция `call` выполняет последовательный проход данных через GRU-слои и выходной слой.

- **Особенности:**

- ▶ Совместимость с другими модулями `DEGANN`.
- ▶ Экспорт параметров через метод `to_dict`.

- **Создание модели:**

- ▶ Использована стандартная архитектура DEGANN на базе класса `IModel`.
- ▶ При создании модели мы просто наследуемся от `IModel` и передаем нужные параметры сети.

- **Основные callback-функции:**

- ▶ `EarlyStopping`: Раннее прекращение обучения при отсутствии улучшений.
- ▶ `SaveBestModel`: Сохранение параметров модели с минимальной валидационной ошибкой.
- ▶ `VisualizationTS`: Визуализация функции потерь и предсказаний модели. На графиках можно посмотреть на качество аппроксимации функции.

- **Цели callback-функций:**

- ▶ Отслеживание метрик и динамики обучения. (используем записи `history` из `Tensorflow`)
- ▶ Оптимизация процесса обучения и предотвращение переобучения.
- ▶ Удобный анализ и визуализация результатов.

- **Генерация датасета:**

- ▶ Использован инструмент DEGANN для генерации данных.
- ▶ Выбрана сложно-аппроксимируемая функция `hardsin`:

$$f(x) = \sin \left(\ln(x^{\sin(10x)}) \right)$$

- ▶ График функции предоставлен на следующем слайде

- **Создание временных последовательностей:**

- ▶ Преобразование данных в последовательности длиной `time_steps`.
- ▶ Это позволяет GRU использовать контекст предыдущих шагов.

- **Добавление шума:**

- ▶ Шум с нормальным (Гауссовским) распределением.
- ▶ Повышает устойчивость модели и адаптацию к реальным данным.

- **Функция потерь:**

- ▶ Использована среднеквадратичная ошибка (MSE):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- ▶ MSE минимизирует крупные отклонения и стабилизирует сходимость.

- **Гиперпараметры модели:**

- ▶ **Количество слоёв: 5, нейронов в слое: 30.**

- **Оптимизатор:**

- ▶ Выбран *Adam*, который стабилизирует градиентный спуск и адаптирует скорость обучения.
- ▶ Эффективен для временных последовательностей и рекуррентных сетей.

- **Цель эксперимента:**

- ▶ Сравнить рекуррентные и полносвязные сети на задачах аппроксимации функций.
- ▶ Оценить время обучения, значение функции потерь (лосс) и качество аппроксимации.

- **Условия эксперимента:**

- ▶ Архитектура:
 - ★ Количество слоёв: 5.
 - ★ Количество нейронов в слое: 30.
- ▶ Количество эпох: 100, 200, 500, 1000 (тестирование на разных значениях).

- **Набор данных:**

- ▶ Использованы встроенные функции DEGANN и функция `hardsin`.
- ▶ `hardsin` была выбрана из-за её сложной математической структуры, которая делает её трудной для аппроксимации.
- ▶ Набор данных позволяет объективно оценить возможности архитектур для задач аппроксимации.

- **Исследовательские вопросы:**

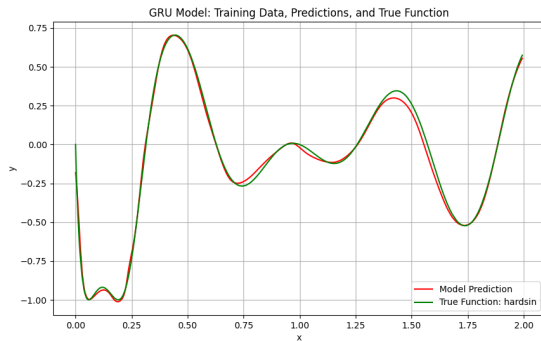
RQ1: Способна ли рекуррентная сеть аппроксимировать встроенные функции DEGANN с меньшим значением функции потерь MSE по сравнению с полносвязной сетью?

RQ2: Насколько значительна разница в точности аппроксимации и скорости обучения?

- **Итоги:**

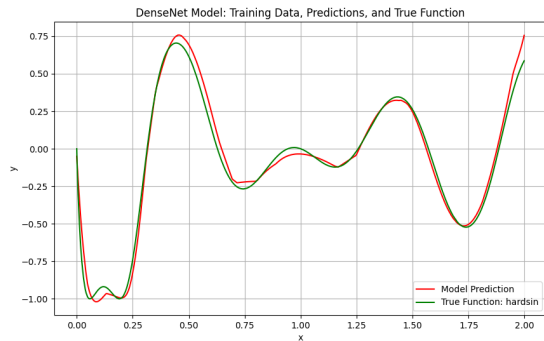
- ▶ Эксперимент находится на стадии проведения.
- ▶ Полные результаты анализа будут готовы на завершающем этапе работы.

Результаты аппроксимации функции hardsin



Training loss = 0.002620

Validation loss = 0.00145



Training loss = 0.004589

Validation loss = 0.003332

- Разработан и реализован модуль с топологией рекуррентной сети для архитектуры GRU, интегрированный в библиотеку DEGANN.
- Создана модель с архитектурой GRU, включающая следующие callback-функции:
 - ▶ Раннее прекращение обучения.
 - ▶ Сохранение лучшей модели.
 - ▶ Визуализация результатов обучения.
- Подготовлены специальные датасеты для тестирования, включая сложно-аппроксимируемую функцию `hardsin`. Реализовано добавление шума для повышения устойчивости модели.
- Проведён подбор функции потерь (MSE) и настройка гиперпараметров, обеспечившая баланс между качеством аппроксимации и вычислительными затратами.

Дальнейшее развитие работы и планы:

- Работа с экспериментами и более глубокий анализ результатов пока находятся на стадии тестирования.
- Планируется разработать кастомную реализацию слоёв GRU для расширения функционала DEGANN.
- Продолжить тестирование модели на дополнительных наборах данных.

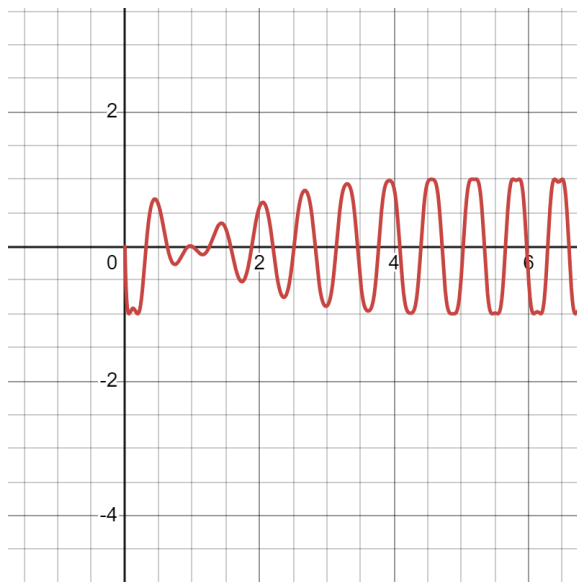


Рис.: График функции $f(x) = \sin(\ln(x^{\sin(10x)}))$

