Санкт-Петербургский государственный университет
Математическое обеспечение и администрирование информационных систем
Информационные системы и базы данных

Назаренко Владимир Владимирович

Эффективная параллельная реализация нейронной сети

Бакалаврская работа

Научный руководитель:
ст. преп. Ярыгина А. С.

Рецензент:
ст. преп. Немешев М. Х.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY
Software and Administration of Information Systems
Information Systems and Databases

Vladimir Nazarenko

Efficient parallel implementation of neural network

Bachelor's Thesis

Scientific supervisor:
Senior Assistant Professor Anna Yarygina

Reviewer:
Senior Assistant Professor Marat Nemeshev

Saint-Petersburg
2016

Содержание

1 Введение

В настоящее время нейронные сети используются в широком спектре задач от распознавания текста на изображениях до детектирования частиц в коллайдерах[1]. Такая универсальность возможна благодаря свойству нейронной сети инкапсулировать нелинейные зависимости и наличию эффективных алгоритмов тренировки нейронной сети.

Одно из ключевых свойств нейронной сети это тип связи между её нейронами; различают:

- Сети прямого распространения (простой перцептрон, свёрточные нейронные сети)[2],
- Рекуррентные нейронные сети[3],
- Самоорганизующиеся карты[4] и другие.

В данной работе рассматриваются только рекуррентные нейронные сети с двумя вариантами рекуррентных слоёв: простой рекуррентный слой и слой LSTM(long short-term memory). Рекуррентные нейронные сети за счёт наличия связей между нейронами скрытого слоя могут превосходить сети прямого распространения в качестве работы, что и наблюдается на некоторых классах задач[5, 6]. Как правило, рекуррентные нейронные сети выигрывают у сетей прямого распространения в случае, когда данные получены из какого-либо последовательного процесса, что имеет место в задачах распознавания речи, автоматического перевода и генерации текста.

Тем не менее рекуррентные сети сложнее в применении, так как обычно требуют большего количества тренировочных данных и имеют более сложную структуру, что в том числе влечёт серьёзное увеличение времени, необходимого для обучения по сравнению с сетями прямого распространения. Кроме того, структура рекуррентных сетей задаёт некоторые ограничения на распараллеливание в связи наличием рекуррентных связей в слоях[7], но не исключает его.

Так как в процессе тренировки и использования натренированной нейронной сети интенсивно используются операции над матрицами и векторами, эти процессы могут быть существенно ускорены с использованием графического процессора (GPU) для вычислений, в частности с использованием платформы Nvidia CUDA[8], для которой такие операции имеют эффективные массово-параллельные реализации, что и продемонстрировано в работах[9, 10].

На сегодняшний день количество публично доступных параллельных реализаций рекуррентных нейронных сетей невелико[11, 7, 12, 13, 14], при этом исходный код некоторых из них закрыт.

2 Постановка задачи

В рамках работы были поставлены следующие задачи:

- рассмотрение и сравнение существующих параллельных реализаций рекуррентных нейронных сетей,
- выделение плюсов и минусов реализаций, их систематизация,
- параллельная реализация нейронной сети, учитывающая достоинства и недостатки предыдущих реализаций.

3 Обзор используемых понятий

В задаче обучения с учителем для того, чтобы модель, и в частности нейронная сеть, была практически полезной, то есть аппроксимировала какую-то функциональную зависимость из реального мира, её необходимо тренировать, то есть выбирать оптимальные параметры на основе имеющихся данных.

Организация процесса тренировки требует принятия нескольких решений, существенно влияющих на время, необходимое для тренировки, и качество полученных результатов, а именно:

- выбор целевой функции,
- выбор алгоритма оптимизации,
- выбор алгоритма вычисления градиента целевой функции.

Далее мы сначала дадим общее определение рекуррентной нейронной сети, затем представим некоторые альтернативные возможности выбора элементов реализации рекуррентной нейронной сети и её тренировки. Для простоты определений, все понятия будем давать для нейронной сети, решающей задачу классификации. Тем не менее определения легко переформулировать, например, для задачи регрессионного анализа.

3.1 Рекуррентная нейронная сеть

Рекуррентная нейронная сеть это нейронная сеть, хотя бы в одном слое которой присутствуют рекуррентные связи, то есть связи между нейронами одного и того же слоя. Такие слои естественно называть рекуррентными.

Как и любая нейронная сеть, рекуррентная нейронная сеть может состоять из практически любой комбинации слоёв, где активации предыдущего слоя являются входными данными для последующего слоя.

Далее рассмотрим простой рекуррентный слой (SRL), в котором каждый нейрон имеет в точности одну связь с собой.

Существуют и другие разновидности рекуррентных слоёв, из которых наиболее часто используется LSTM-слой. Описывать его в этой работе мы не будем, читатель может найти информацию о нём в работе[15]. Далее под рекуррентным слоем будем понимать либо простой рекуррентный слой, либо LSTM-слой.

Также часто используются двунаправленные слои. Так как суть рассуждений для двунаправленных слоёв остаётся такой же, как и для однонаправленных, мы не будем давать определение двунаправленных слоёв, а лишь сошлёмся на работу[16].

В отличие от нейронной сети прямого распространения рекуррентная нейронная сеть работает не с парой значений вида (входное значение, выходное значение), а с временным рядом, представляющими собой последовательность значений вида (входное значение, выходное значение, индекс элемента в последовательности), где индекс элемента в последовательности – целочисленное неотрицательное значение, которое можно интерпретировать как дискретное время. Формально, тренировочная последовательность имеет вид (x, y), где

- $x = [x(0), \ldots, x(T)]$, $x(t)$ – вектор признаков,
- $y = [y(0), \ldots, y(T)]$, $y(t) = [y_1(t) \ldots y_N(t)]$,
  где $y_k(t) = 1$ и $\forall j \neq k$ $(y_j(t) = 0)$, где k – номер класса, которому соответствует набор признаков $x(t)$,
- T – длина последовательности, t – индекс последовательности, время.

Далее для примера рассмотрим рекуррентную нейронную сеть Элмана. Если не оговорено иное, то под рекуррентной нейронной сетью будем понимать именно рекуррентную сеть Элмана. Тем не менее все последующие рассуждения с некоторыми дополнениями применимы и для более сложных рекуррентных нейронных сетей.

Нейронная сеть Элмана

Нейронная сеть Элмана, или простой рекуррентный перцептрон, – это одна из простейших конфигураций рекуррентной нейронной сети, состоящая из трёх слоёв:

- входной слой,
- простой рекуррентный слой,
- выходной слой.

Задаётся такая сеть следующими соотношениями (с использованием обозначений из таблицы 1)[17]:

$$net^{(1)}(t) = Vx(t) + Us(t-1), \tag{3.1.1}$$

$$s(t) = f(net^{(1)}(t)), \tag{3.1.2}$$

$$net^{(2)}(t) = Ws(t), \tag{3.1.3}$$

$$o(t) = g(net^{(2)}(t)). \tag{3.1.4}$$

Обозначим также за $h_\omega(x,t) = o(t)$ активации выходного слоя на шаге t рекуррентной нейронной сети при условии, что x – входная последовательность при конкретном значении $\omega = (U, V, W)$. Также обозначим $h_\omega(x) = [h_\omega(0,x) \dots h_\omega(T,x)]$.

Таблица 1: Основные обозначения

Символ	Значение
t	Индекс элемента последовательности (время)
f, g	Функции активации
$x(t)$	Входной вектор размерности N
$s(t)$	Вектор скрытого слоя размерности M
$o(t)$	Вектор выходного слоя размерности K
$V \in [M \times N]$, $U \in [M \times M]$, $W \in [K \times M]$	Матрицы весов

На рис.1 и рис.2 можно видеть схематичное изображение рекуррентной сети и так называемое "развёрнутое" изображение нейронной сети.

3.2 Целевые функции

Целевая функция это веществennозначная, имеющая непрерывную производную функция $Q(\omega, x, y) = l(h_\omega(x), y)$. При правильном выборе целевой функции задача тренировки нейронной сети сводится к задаче оптимизации целевой функции. Вариантов выбора отображения l существует очень много и разные целевые функции зачастую приводят к разным результатам, более того, некоторые целевые функции ориентированы на специальные классы задач. Приведём некоторые примеры.

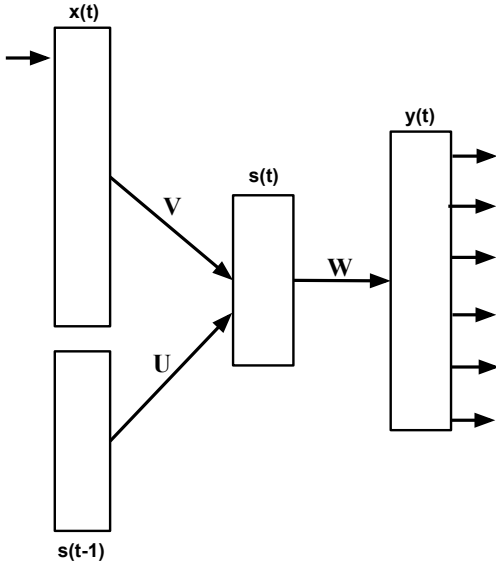

Рис. 1: Схема рекуррентной нейронной сети

Среднеквадратичное отклонение (MSE)

Среднеквадратичное отклонение – это одна из наиболее часто используемых целевых функций в задачах линейной регрессии. Выглядит она следующим образом:

$$l(h_\omega(x), y) = \frac{1}{T}\sum_{1}^{T} ||h_\omega(x,t) - y(t)||^2.$$

Перекрёстная энтропия (cross-entropy loss)

Перекрёстная энтропия – одна из простейших целевых функций, используемых при решении задач классификации. Задаётся она следующим выражением:

$$l(h_\omega(x), y) = \frac{1}{T}\sum_{1}^{T} < y(t), \log h_\omega(x,t) > .$$

Связанная временная классификация

Более сложным примером целевой функции является связанная временная классификация (CTC)[18], используемая, например, в распознавании речи и символьных последовательностей.

Предположим, что мы выполняем на последовательности классификацию среди n классов. При решении задачи распознавания речи это может быть, например, разметка звуковой дорожки на произнесённые буквы в каждом интервале. Выходной слой в таком случае представляет собой n

нейронов, каждый из которых соответствует отдельному классу. В выходной слой нейронной сети добавим ещё один нейрон – он будет соответствовать паузе. В качестве функции активации выходного слоя для нормализации рекомендуется использовать softmax, тогда мы можем говорить о вероятностях

$$Pr(k,t|x) = \frac{exp(h_\omega(x,t)_k)}{\sum_{k'} exp(h_\omega(x,t)_{k'})},$$

где k – класс, x – входная последовательность, t, как и ранее, индекс временного шага.

CTC-разметка a – это последовательность длины T из символов и пауз. Тогда вероятность, что входной последовательности будет соответствовать конкретная разметка, примем равной

$$Pr(a|x) = \prod_{t=1}^{T} Pr(a_t,t|x),$$

считая, что вероятности появления символов в строке независимы.

Далее предположим, что у нас в тренировочном множестве каждой входной последовательности x соответствует её транскрипция (строка) y^*. По этой строке мы генерируем все возможные разметки. Например, строке abc при $T = 5$ могут соответствовать разметки (a, -, b, c, -), (a, -, -, b, c) и другие, где "-" – пауза. Обозначим как $\Im(y^*)$ оператор, генерирующий все возможные разметки по транскрипции.

Тогда можем вычислить

$$Pr(y^*|x) = \sum_{a \in \Im(y^*)} Pr(a|x).$$

Целевой функцией будет являться

$$l(h_\omega(x), y^*) = -logPr(y^*|x).$$

3.3 Алгоритмы оптимизации

Пусть $\Psi : |\Psi| = S$ – тренировочное множество, т.е. набор тренировочных последовательностей. Рассмотрим среднее значение функции ошибки на тренировочном множестве, если принять веса нейронной сети равными ω, $L(\omega) = \frac{1}{n}\sum_{i=1}^{S} Q(\omega, x^i, y^i)$. Получаем составную целевую функцию, которую и будем минимизировать.

Также нам потребуется вычислять $\nabla L(\omega)$, для этого можно использовать как BPTT(3.4.1), так и RTRL-алгоритм(3.4.2).

3.3.1 Градиентный спуск

Градиентный спуск и его модификации на сегодняшний день чаще всего применяется для тренировки рекуррентных нейронных сетей в виду простоты реализации, низкой вычислительной сложности и эффективности некоторых модификаций. В листингах 1,2 и 3 (в приложении) приведены некоторые часто используемые варианты градиентного спуска.

3.3.2 Расширенный фильтр Калмана

Фильтр Калмана это относительно давно известная техника оптимизации линейных динамических систем. В работе[11] и предшествующих ей работах Michal Cernansky предпринята попытка расширить понятие фильтра Калмана и применить его к обучению рекуррентных нейронных сетей. Описание алгоритма представлено в листинге 4.

3.3.3 Алгоритм l-BFGS

Алгоритм l-BFGS (limited-memory Broyden – Fletcher – Goldfarb – Shanno algorithm)[19] является приближением алгоритма BFGS с линейной относительно длины входных данных асимптотикой используемой памяти.

Введём некоторые обозначения:

ω_k – вектор, содержащий в себе все веса нейронной сети на k-м шаге алгоритма.

$$s_k = \omega_{k+1} - \omega_k$$

$$y_k = \nabla L(\omega_{k+1}) - \nabla L(\omega_k)$$

$$\rho_k = \frac{1}{y_k^T s_k}$$

Данный алгоритм относится к квази-линейным. Он не только учитывает производную целевой функции, но и аппроксимирует её гессиан, что приводит к ускорению сходимости. Псевдокод алгоритма представлен в листинге 5.

Кроме того, одной из важных частей алгоритма является линейный поиск, который должен опираться на условия Вольфе[19]. Один из примеров такого линейного поиска представлен в листинге 6.

3.4 Вычисление градиента целевой функции

3.4.1 Обратное распространение ошибки сквозь время

Одним из наиболее часто используемых методов вычисления градиента функции ошибки является так называемый метод обратного распространения ошибки сквозь время (BPTT). Суть метода в том, что мы "разворачиваем" рекуррентную сеть и считаем градиенты с помощью обычного метода распространения ошибки.

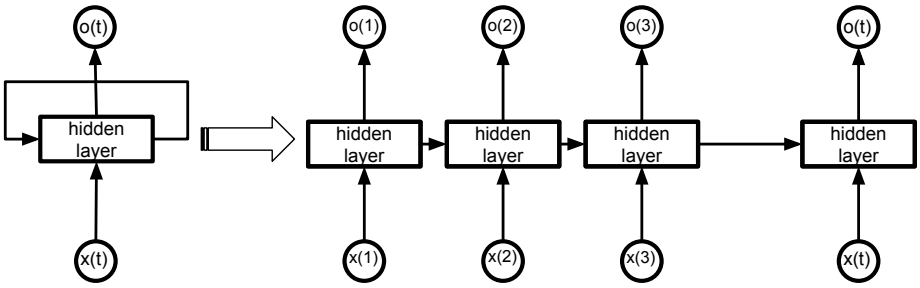

Рис. 2: "Развёрнутая" нейронная сеть

Ниже представлены вычисления градиентов функции ошибок от матриц параметров.

Используя цепное правило, получаем:

$$\frac{\partial o(t)}{\partial w_{ij}} = \frac{\partial o(t)}{\partial net^{(2)}(t)} \frac{\partial net^{(2)}(t)}{\partial w_{ij}}, \tag{3.4.1}$$

$$\frac{\partial o(t)}{\partial v_{ij}} = \frac{\partial o(t)}{\partial net^{(2)}(t)} \frac{\partial net^{(2)}(t)}{\partial s(t)} \frac{\partial s(t)}{\partial v_{ij}}, \tag{3.4.2}$$

$$\frac{\partial o(t)}{\partial u_{ij}} = \frac{\partial o(t)}{\partial net^{(2)}(t)} \frac{\partial net^{(2)}(t)}{\partial s(t)} \frac{\partial s(t)}{\partial u_{ij}}. \tag{3.4.3}$$

Согласно 3.1.3, 3.1.4:

$$\frac{\partial o(t)}{\partial net^{(2)}(t)} = J_g(net^{(2)}(t)) = \begin{bmatrix} \nabla g_1 \\ \vdots \\ \nabla g_N \end{bmatrix} (net^{(2)}(t)), \tag{3.4.4}$$

$$\frac{\partial net^{(2)}(t)}{\partial w_{ij}} = \begin{bmatrix} 0 \\ \vdots \\ s_j(t) \\ \vdots \\ 0 \end{bmatrix} \leftarrow (i). \tag{3.4.5}$$

Таким образом, 3.4.4 и 3.4.5 влекут

$$\frac{\partial o(t)}{\partial w_{ij}} = s_j(t) \left[g_1'(net_i^{(2)}(t)) \quad \dots \quad g_n'(net_i^{(2)}(t)) \right]. \tag{3.4.6}$$

Из 3.1.3 очевидно следует

$$\frac{\partial net^{(2)}(t)}{\partial s(t)} = W. \tag{3.4.7}$$

Используя предыдущие тождества, введём обозначение

$$\delta(t) = \frac{\partial o(t)}{\partial net^{(2)}(t)} \frac{\partial net^{(2)}(t)}{\partial s(t)} = J_g(net^{(2)}(t))W. \tag{3.4.8}$$

Тогда 3.4.2 и 3.4.3 можно переписать в виде

$$\frac{\partial o(t)}{\partial v_{ij}} = \delta(t) \frac{\partial s(t)}{\partial v_{ij}}, \tag{3.4.9}$$

$$\frac{\partial o(t)}{\partial u_{ij}} = \delta(t) \frac{\partial s(t)}{\partial u_{ij}}. \tag{3.4.10}$$

Снова используя цепное правило, получаем:

$$\frac{\partial s(t)}{\partial v_{ij}} = \sum_{\tau=0}^{t} \frac{\partial s(t)}{\partial s(t-\tau)} \frac{\partial s(t-\tau)}{\partial v_{ij}}, \tag{3.4.11}$$

$$\frac{\partial s(t)}{\partial s(t-\tau)} = J_f(net^{(1)}(t))U * \dots * J_f(net^{(1)}(t-\tau+1))U \quad \forall \tau > 0, \tag{3.4.12}$$

$$\frac{\partial s(t-\tau)}{\partial v_{ij}} = J_f(net^{(1)}(t-\tau)) \begin{bmatrix} 0 \\ \vdots \\ x_j(t-\tau) \\ \vdots \\ 0 \end{bmatrix} \leftarrow (i). \tag{3.4.13}$$

Аналогично,

$$\frac{\partial s(t)}{\partial u_{ij}} = \sum_{\tau=0}^{t} \frac{\partial s(t)}{\partial s(t-\tau)} \frac{\partial s(t-\tau)}{\partial u_{ij}}, \tag{3.4.14}$$

$$\frac{\partial s(t-\tau)}{\partial u_{ij}} = J_f(net^{(1)}(t-\tau)) \begin{bmatrix} 0 \\ \vdots \\ s_j(t-\tau-1) \\ \vdots \\ 0 \end{bmatrix} \leftarrow (i). \tag{3.4.15}$$

3.4.2 Рекуррентное обучение в реальном времени

Для простоты рассмотрим ещё более простую рекуррентную нейронную сеть – сеть Элмана без выходного слоя, то есть активациями выходного слоя в такой сети будут являться активации скрытого слоя.

Оставшиеся матрицы весов, (U, V) соберём в одну матрицу $\Omega \in [M, N+M]$. Рекуррентное обучение в реальном времени (RTRL) основывается на несколько ином взгляде на рекуррентную нейронную сеть. Во-первых, объединим весовые матрицы из таб.1 в одну – $\Omega \in [M, N+M+K]$. Также обозначим $z(t) = [x(t), s(t)]$. Введём множества индексов I, J и проиндексируем ими соответственно $x(t), s(t)$. Тогда такую сеть можно описать двумя выражениями:

$$net(t) = \Omega z(t), \tag{3.4.16}$$

$$o(t+1) = f(net(t)). \tag{3.4.17}$$

На этапе тренировки нейронной сети для некоторых активаций, а именно активаций выходного слоя у нас заданы целевые значения $d_k(t)$.

Продифференцируем выходы сети по весам:

$$\frac{\partial o_k(t+1)}{\partial \omega_{ij}} = f'_k(net_k(t)) \left[\sum_{l \in U \cup I} \omega_{kl} \frac{\partial z_l(t)}{\partial \omega_{ij}} + \delta_{ik} z_j(t) \right]. \tag{3.4.18}$$

Так как входные активации не зависят от весов,

$$\frac{\partial z_l(t)}{\partial \omega_{ij}} = 0 \quad \forall l \in I. \tag{3.4.19}$$

Также можем считать, что данные в начальный момент времени не зависят от весов, то есть

$$\frac{\partial o_k(0)}{\partial \omega_{ij}} = 0. \tag{3.4.20}$$

Используя выражение 3.4.20 как базу рекурсии на основании выражений 3.4.18, 3.4.19 получаем рекуррентную формулу:

$$\frac{\partial o_k(t+1)}{\partial \omega_{ij}} = f'_k(net_k(t)) \left[\sum_{l \in U} \omega_{kl} \frac{\partial o_l(t)}{\partial \omega_{ij}} + \delta_{ik} z_j(t) \right]. \tag{3.4.21}$$

Таким образом число $\Delta\omega_{ij} = \sum_{k \in U} \frac{\partial y_k(t+1)}{\partial \omega_{ij}}$ задаём оптимальное изменение веса. По-настоящему оптимальным это изменение будем в случае, если такие изменения будут накапливаться со всех элементов тренировочной последовательности, но в данном алгоритме изменения сразу прибавляются к весам и следующие итерации работают с уже обновлёнными весами. Это позволяет очень существенно распараллелить алгоритм.

К сожалению, как отмечено в работе[3], данный алгоритм слабо применим на практике из-за его большой ресурсоёмкости.

4 Платформа для параллельных вычислений CUDA

Так как некоторые работы[9, 10] свидетельствуют о том, что тренировка нейронной сети может быть существенно ускорена с использованием GPU, рассмотрим одну из популярных платформ для реализации математических вычислений на GPU общего назначения: NVIDIA CUDA[8]. Достоинства этой платформы заключаются в её широкой распространённости и простоте использования. Также NVIDIA и сторонние разработчики предлагают удобные и эффективные библиотеки операций линейной алгебры.

В данной работе интенсивно использовались две из них: Thrust и CUDA BLAS.

Thrust. Библиотека NVIDIA Thrust позволяет эффективно реализовывать операции над векторами на графическом процессоре в терминах функционального программирования, то есть операций map, reduce и других. К сожалению, геометрических операций над векторами в этой библиотеке практически нет. Авторы ограничились реализацией скалярного произведения.

CUDA BLAS. Библиотека CUDA BLAS позволяет осуществлять большой спектр операций линейной алгебры на графическом процессоре. К недостаткам можно отнести разве что нестандартное расположение матриц в памяти (column wise), что в некоторых случаях приводит к необходимости реаллокации матриц.

5 Существующие реализации

Непараллельная реализация рекуррентной нейронной сети не является сложной задачей[20], но, к сожалению, такая реализация обладает плохой масштабируемостью и не подходит для больших задач. Параллельная реализация сложнее в исполнении в виду наличия временных зависимостей в рекуррентных нейронных сетях, но такая реализация необходима для больших задач.

На данный момент существует ряд крупных параллельных реализаций рекуррентных нейронных сетей.

5.1 CURRENNT

Библиотека CURRENNT[7] представляет собой полноценный инструмент для создания, конфигурирования и тренировки нейронных сетей, в том числе рекуррентных. В данной реализации представлены

- LSTM и двунаправленный LSTM слои,
- Слои прямого распространения с различными функциями активации,
- Различные выходные слои, представляющие собой различные функции ошибок,
- Единственный алгоритм тренировки – стохастический градиентный спуск с моментом.

Распараллеливание в этой библиотеке реализовано на двух уровнях:

- на уровне матричных и векторных операций средствами библиотек NVIDIA Thrust и CUDA BLAS,
- тренировкой нейронной сети на нескольких последовательностях одновременно.

Архитектура CURRENNT

CURRENNT имеет модульную архитектуру (рис.3, рис.6), что даёт простор для расширения функциональности. Такая структура позволяет добавлять в код реализацию новых методов тренировки сети, новых тренируемых слоёв, новых функций ошибки.

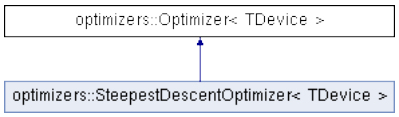

Рис. 3: Иерархия оптимизаторов CURRENNT

Библиотеку можно условно поделить на несколько частей:

- Модуль вычисления операций линейной алгебры. Операции линейной алгебры реализованы в трёх вариантах – с помощью библиотеки cuda blas, с помощью библиотеки cuda thrust и в виде кода для исполнения на CPU. К сожалению, ни одна из этих реализаций не может эффективно перемножать матрицы, размер которых больше размера памяти графического процессора, но эту функциональность несложно добавить.
- Модуль построения нейронной сети в виде композиции слоёв.

- Модуль тренировки нейронной сети. В данный момент реализован только стохастический градиентный спуск
- Модуль работы с датасетом. Поддерживается распространённый формат netCDF.

5.2 Реализация Michal Cernansky

Данная реализация[11] является результатом цикла работ по применению фильтраций Калмана к тренировке нейронных сетей. К сожалению, эта реализация носит лишь демонстрационный характер. Хоть она и является параллельной, эффективной её назвать сложно, так как тренировка нейронной сети с помощью фильтра Калмана требует большого количества перемножений плотных матриц большой размерности.

5.3 Caffe

Библиотека Caffe[14] на данный момент не поддерживает рекуррентные нейронные сети, но заслуживает упоминания как одна из самых известных. Кроме того, авторами заявлено добавление поддержки рекуррентных нейронных сетей в будущем.

5.4 Nvidia CuDNN

Nvidia CuDNN[13] это специализированная библиотека для ускорения тренировки нейронных сетей на графическом процессоре. Содержит также примитивы для рекуррентных сетей. Данная библиотека не является полноценной реализацией и не имеет открытого исходного кода в отличие от всех других представленных решений.

5.5 Microsoft CNTK

Библиотека Microsoft CNTK[12] представляет собой высокоэффективный фреймворк, способный осуществлять конфигурацию и тренировку различных типов нейронных сетей, в том числе рекуррентных, но он не является самостоятельной, а опирается на библиотеку CuDNN.

6 Реализация

В процессе уточнения задачи было решено вместо создания собственной реализации с нуля расширить функционал библиотеки CURRENNT[7].

На момент написания работы в этой реализации были представлены LSTM и двунаправленный LSTM слои, стохастический градиентный спуск с моментом и множество вспомогательных инструментов.

Решено было добавить:

- простой рекуррентный слой,
- двунаправленный рекуррентный слой,
- алгоритм тренировки l-BFGS.

К сожалению, добавить алгоритм l-BFGS без модификаций изначального кода библиотеки CURRENNT не удалось, так как интерфейс оптимизатора не позволял в течение одной итерации просматривать набор данных несколько раз, что необходимо для работы алгоритма линейного поиска. Таким образом, пришлось расширить интерфейс оптимизатора, добавив в него метод ComputeForwardBackwardPassOnTrainset, вычисляющий градиент и значение функции ошибки на части тренировочного множества в случае стохастической тренировки и на всех тренировочных данных иначе.

В процессе разработки выяснилось, что путём внесения небольших модификаций в упомянутые алгоритмы можно существенно увеличить качество работы нейронной сети. Так для l-BFGS был добавлен стохастический режим, когда обновление весов делается после просмотра части тренировочных последовательностей, а не всего датасета. Такой модифицированный алгоритм будем называть sl-BFGS в простой рекуррентный слой была добавлена нормализация градиента, чтобы избежать возникшей проблемы экспоненциального роста градиента. Эти модификации существенно повлияли на эффективность алгоритмов.

7 Эксперименты

Тестирование и сравнение реализаций проводилось на датасете TIMIT[21]; с помощью рекуррентных нейронных сетей различных конфигураций на этом датасете решалась задача классификации на 39 классах. Размерность пространства признаков – 75. Длины последовательностей варьируются от 85 до 760. В тренировочном множестве содержится 4620 последовательностей, в тестовом – 1680.

Эксперименты проводились на виртуальном сервере amazon g2.2xlarge с графическим вычислителем NVIDIA GRID K520, совместимым с CUDA 3 и обладающим 4гб видеопамяти.

В связи с недоступностью или неприменимостью остальных реализаций, эксперименты проводились с использованием расширенной библиотеки CURRENNT. В экспериментах затронуты реализованные авторами

библиотеки градиентный спуск с моментом и LSTM-слой, а также реализованные в рамках данной работы простой рекуррентный слой и алгоритм sl-BFGS.

Во всех конфигурациях, протестированных в ходе экспериментов, нейронная сеть помимо входного и выходного слоёв включала в себя один или более рекуррентных слоёв и слой прямого распространения с функцией активации softmax размерности 39. Далее мы будем указывать только конфигурации рекуррентных слоёв, так как остальные слои оставались неизменными в ходе экспериментов. Используемый алгоритм оптимизации, тип и размерности рекуррентных слоёв варьировались. Кроме того, для оптимизатора sl-BFGS варьировался также размер хранилища, так как этот параметр влияет на среднюю скорость одной итерации тренировки[19].

7.1 Сравнение скорости тренировки

Сначала мы зафиксировали две конфигурации нейронной сети:

- сеть с одним простым рекуррентным слоем размерности 100,
- сеть с одним LSTM-слоем размерности 50.

Для обеих конфигураций мы запустили 100 итераций тренировки нейронной сети алгоритмом sl-BFGS с размером хранилища 15. Результаты этого эксперимента можно увидеть на рис. 4. Такие размерности слоёв были выбраны, так как количества оптимизируемых параметров в них соизмеримы (17600 у SRL против 26350 у LSTM). Несмотря на этот факт, легко видеть, что простой рекуррентный слой даёт гораздо менее точные предсказания, чем LSTM-слой. Это связано с неспособностью модели запоминать дальние зависимости[15], которые скорее всего присутствуют в выбранном датасете, так как длина последовательностей достаточно велика.

Показательно сравнить скорость тренировки нейронной сети с помощью стохастического градиентного спуска с моментом и с помощью sl-BFGS. Так как подбор оптимальных параметров не является целью данной работы, были использованы стандартные конфигурации оптимизаторов и изменялись лишь число и размеры слоёв нейронной сети. Так для градиентного спуска с моментом использовались скорость обучения 10^{-5} и момент 0.5, стандартные параметры, стандартная конфигурация для sl-BFGS взята из [19].

На рисунке 5 изображёно изменение ошибки на тестовом множестве в процессе тренировки при тренировке нейронной сети с помощью стохастического градиентного спуска (SGD) и с помощью sl-BFGS, конфигурации которых описаны выше. Заметно, что алгоритм sl-BFGS в начале процесса тренировки сходится быстрее благодаря использованию аппроксимации производных второго порядка.

Рис. 4: Тренировка рекуррентных слоёв

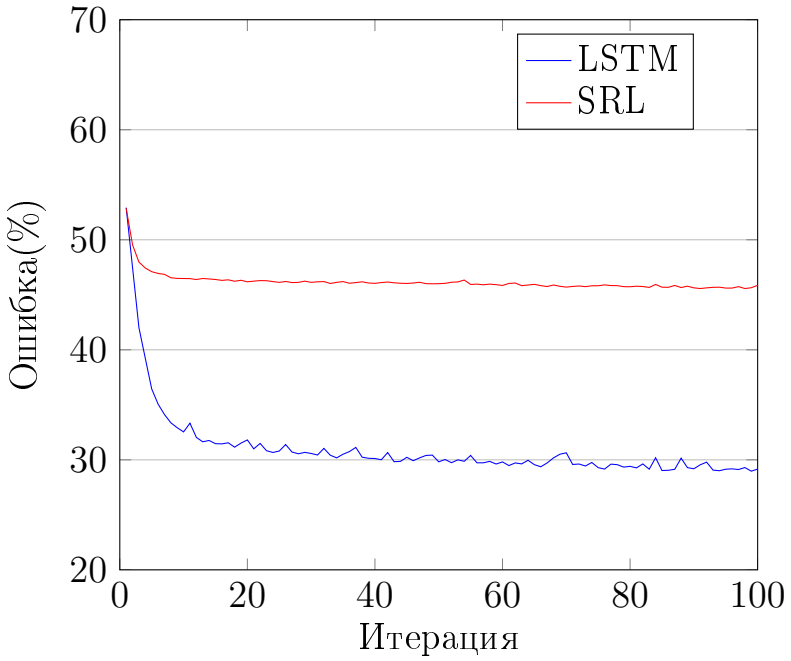

Рис. 5: Тренировка нейронной сети алгоритмами sl-BFGS и SGD

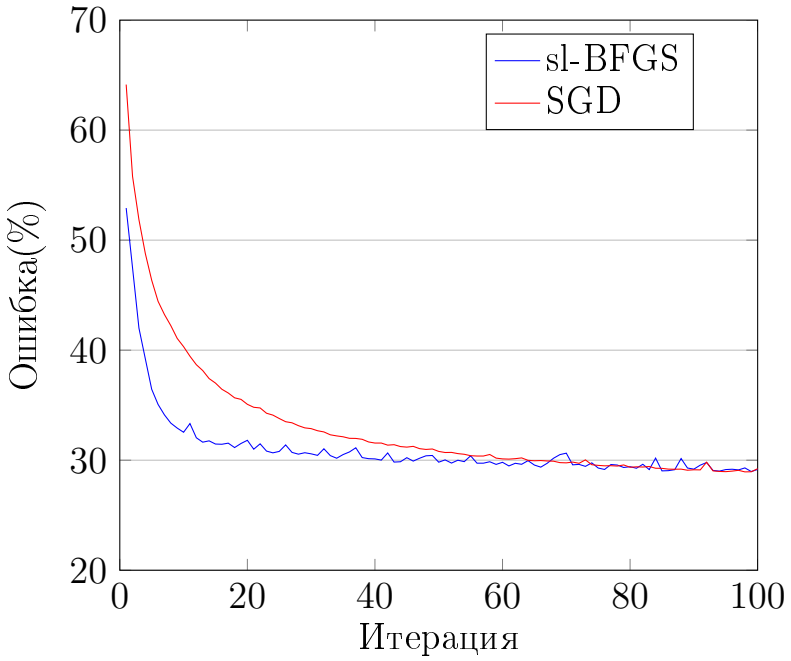

Таблица 2: Среднее время итерации (в секундах)

Конфигурация сети	**SGD**	**sl-BFGS5**	**sl-BFGS15**
LSTM20	10.283	10.78	11.04
LSTM50	10.262	17.39	17.62
LSTM100	24.52	29.96	29.86
LSTM20+LSTM20	10.293	14.23	14.53
SRL40	10.223	10.57	10.56
SRL100	10.326	13.75	13.90
SRL200	12.314	20.90	20.97
SRL40+SRL40	10.218	14.17	14.84

Также было измерено время одной итерации в различных конфигурациях. Результаты можно видеть в таблице 2. В этой таблице sl-BFGS5 и sl-BFGS15 соответствую алгоритму sl-BFGS с размерами хранилища 5 и 15 соответственно. LSTM20+LSTM20 означает, что в качестве рекуррентных слоёв были использованы два LSTM-слоя, каждый размерности 20.

Можно заметить, что итерация алгоритма sl-BFGS в среднем длится дольше итерации градиентного спуска. Связано это с необходимостью выполнения дополнительных матричных операций. Более того, время итерации sl-BFGS варьируется в широких пределах, так как в этом алгоритме используется линейный поиск. Ещё можно заметить, что тренировка нейронной сети с простым рекуррентным слоем занимает меньше времени, что логично, поскольку структура простого рекуррентного слоя гораздо проще, чем структура LSTM-слоя.

Также на сети, содержащей один LSTM-слой размерности 50 были проведены замеры среднего времени итерации при различном количестве последовательностей, обрабатываемых параллельно. Результаты представлены в таблице 3. Легко видеть, что увеличение числа последовательностей, обрабатываемых параллельно приводит к существенному ускорению процесса тренировки нейронной сети.

7.2 Сравнение качества тренировки

Наконец, рассмотрим точность классификации на тестовом множестве при помощи нейронных сетей различных конфигураций, натренированных различными алгоритмами классификации за 100 итераций. Заметно, что нейронные сети одной конфигурации, натренированные разными алгоритмами почти всегда дают идентичные результаты. Существенное различие качества зафиксировано только для случая двух простых рекуррентных слоёв размерности 40.

Таблица 3: Зависимость времени итерации (в секундах) от числа последовательностей, обрабатываемых параллельно

# последовательностей	**SGD**	**sl-BFGS**
10	16.811	33.344
20	10.6	20
50	8.844	16.578
100	8.2222	15.211

Таблица 4: Точность на тестовом множестве после 100 итераций алгоритма тренировки

Конфигурация сети	**SGD**	**sl-BFGS5**	**sl-BFGS15**
LSTM20	66.07	66.28	66.63
LSTM50	71.06	70.69	71.03
LSTM100	74.33	72.45	73.08
LSTM20+LSTM20	68.22	68.18	68.3
SRL40	44.14	46.74	46.92
SRL100	53.52	54.58	54.43
SRL200	56.28	56.3	56.33
SRL40+SRL40	41.11	49.04	48.43

8 Заключение

Таким образом в ходе работы были рассмотрены различные подходы к построению и тренировке рекуррентных нейронных сетей, проведён анализ существующих параллельных реализаций рекуррентных нейронных сетей. В одну из таких параллельных реализаций добавлена новая функциональность, а именно простой рекуррентный слой и алгоритм тренировки l-BFGS.

В ходе экспериментов выяснилось, что на выбранном наборе данных эффективнее проводить тренировку нейронной сети с помощью стохастического градиентного спуска, нежели с помощью алгоритма sl-BFGS, так как время одной итерации sl-BFGS существенно превосходит время итерации стохастического градиентного спуска, нивелируя выигрыш в увеличении точности после каждой итерации.

Также данная работа ещё раз иллюстрирует превосходство нейронных сетей с LSTM-слоями над сетями на основе простого рекуррентного слоя, обусловленное способностью LSTM-слоя моделировать отдалённые зависимости в данных. Тем не менее, простой рекуррентный слой используется в современных исследованиях[18] ввиду низкого времени тренировки, так что выполненная реализация может оказаться полезной.

Хотя работа над поставленными задачами закончена, исследование

можно продолжить, например, реализовав дополнительные алгоритмы тренировки, например AdaGrad, который, согласно работе[22], показывает более высокую скорость тренировки, чем стохастический градиентный спуск на рекурретных нейронных сетях.

Список литературы

[1] Bernard Widrow, David E. Rumelhart, and Michael A. Lehr. Neural networks: Applications in industry, business and science. *Commun. ACM*, 37(3):93–105, March 1994.

[2] Anil K Jain, Jianchang Mao, and KM Mohiuddin. Artificial neural networks: A tutorial. *Computer*, (3):31–44, 1996.

[3] Ilya Sutskever. *Training recurrent neural networks*. PhD thesis, University of Toronto, 2013.

[4] Fernando Baçao, Victor Lobo, and Marco Painho. Self-organizing maps as substitutes for k-means clustering. In *Computational Science–ICCS 2005*, pages 476–483. Springer, 2005.

[5] MN Karim and SL Rivera. Comparison of feed-forward and recurrent neural networks for bioprocess state estimation. *Computers & chemical engineering*, 16:S369–S377, 1992.

[6] Martin Sundermeyer, Ilya Oparin, Jean-Luc Gauvain, Ben Freiberg, Ralf Schluter, and Hermann Ney. Comparison of feedforward and recurrent neural network language models. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8430–8434. IEEE, 2013.

[7] Felix Weninger. Introducing currennt: The munich open-source cuda recurrent neural network toolkit. *Journal of Machine Learning Research*, 16:547–551, 2015.

[8] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, March 2008.

[9] Kyoung-Su Oh and Keechul Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004.

[10] Boxun Li, Erjin Zhou, Bo Huang, Jiayi Duan, Yu Wang, Ningyi Xu, Jiaxing Zhang, and Huazhong Yang. Large scale recurrent neural network on gpu. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 4062–4069. IEEE, 2014.

[11] Michal Čerňanský. Training recurrent neural network using multistream extended kalman filter on multicore processor and cuda enabled graphic processor unit. In Cesare Alippi, Marios Polycarpou, Christos Panayiotou, and Georgios Ellinas, editors, *Artificial Neural Networks – ICANN 2009*, volume 5768 of *Lecture Notes in Computer Science*, pages 381–390. Springer Berlin Heidelberg, 2009.

[12] Chris Basoglu Guoguo Chen Scott Cyphers Amit Agarwal, Eldar Akchurin et al. An introduction to computational networks and the computational network toolkit. Technical Report MSR-TR-2014-112, August 2014.

[13] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681, 1997.

[17] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[18] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1764–1772, 2014.

[19] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[20] Alex Graves. Rnnlib: A recurrent neural network library for sequence learning problems. `http://sourceforge.net/projects/rnnl/`.

[21] John S Garofolo, Lori F Lamel, William M Fisher, Jonathan G Fiscus, and David S Pallett. Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1. *NASA STI/Recon Technical Report N*, 93, 1993.

[22] Hieu Pham, Zihang Dai, and Lei Li. On optimization algorithms for recurrent networks with long short-term memory. *Momentum*, 1:0–6.

9 Приложение

Алгоритм 1 Градиентный спуск

$j \leftarrow 0$
инициализировать ω_0, α
repeat
    $Gradient \leftarrow \mathbb{O}$
    $N \leftarrow 0$
    for (x_k, y_k) из тренировочного множества **do**
        $Gradient \leftarrow Gradient + \nabla_{\omega_j} Q(\omega_j, x_k, y_k)$
        $N \leftarrow N + 1$
    end for
    $\omega_{j+1} = \omega_j - \alpha \frac{1}{N} Gradient$
    $j \leftarrow j + 1$
until convergence

Алгоритм 2 Стохастический градиентный спуск

$j \leftarrow 0$
инициализировать ω_0, α
repeat
    $N \leftarrow 0$
    $\epsilon_0 \leftarrow \omega_j$
    for (x_k, y_k) из тренировочного множества **do**
        $\epsilon_k \leftarrow \epsilon_{k-1} - \alpha \nabla_{\epsilon_{k-1}} Q(\epsilon_{k-1}, x_k, y_k)$
        $N \leftarrow N + 1$
    end for
    $\omega_{j+1} = \epsilon_N$
    $j \leftarrow j + 1$
until convergence

Алгоритм 3 Пакетный градиентный спуск

$j \leftarrow 0$
инициализировать ω_0, α
repeat
    $N \leftarrow 0$
    for $i = 1..|\omega_j|$ **do**
        $g \leftarrow 0$
        for (x_k, y_k) из тренировочного множества **do**
            $g \leftarrow g + \nabla_{\omega_j^i} Q(\omega_j, x_k, y_k)$
            $N \leftarrow N + 1$
        end for
        $\omega_j^i \leftarrow \omega_j^i - \alpha \frac{1}{N} g$
    end for
    $\omega_{j+1} \leftarrow \omega_j$
    $j \leftarrow j + 1$
until convergence

Алгоритм 4 Тренировка рекуррентной нейронной сети с помощью фильтра Калмана

Выбрать начальный вектор весов ω_0;
$k \leftarrow 0$;
$P(0) \leftarrow$ диагональная матрица размера $(M * N + M * M + K * M) \times K$, на диагонали которой стоят довольно большие числа (порядка 100);
repeat
    for (x, y) из тренировочного множества **do**
        $\hat{w}(0) \leftarrow \omega_k$
        for $t = 0 \ldots T$ **do**
            $H(t) \leftarrow \frac{\partial o(t)}{\partial \hat{w}(t)}$;
            $K(t) = P(t)H(t)[H(t)^T P(t)H(t)]^{-1}$
            $\xi(t) \leftarrow y(t) - o(t)$
            $\hat{w}(t+1) = \hat{w}(t) + K(t)\xi(t)$
            $P(t+1) = P(t) - K(t)H(t)^T P(t)$
        end for
        $\omega_k \leftarrow \hat{w}(T+1)$
    end for
    $\omega_{k+1} \leftarrow \omega_k$
until convergence

Алгоритм 5 l-BFGS для нейронной сети

Выбрать начальный вектор весов ω_0, выбрать число $m > 0$ – размер хранилища
k $\leftarrow 0$;
repeat
  if $k = 0$ **then**
    $H_k^0 \leftarrow I$;
  else
    $\gamma_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$;
    $H_k^0 \leftarrow \gamma_k I$;
  end if
  q $\leftarrow \nabla L(\omega_k)$;
  for $i = k-1, k-2, \ldots, k-m$ **do**
    $\alpha_i \leftarrow \rho_i s_i^T q$;
    $q \leftarrow q - \alpha_i y_i$;
  end for
  $r \leftarrow H_k^0 q$;
  for $i = k-m, k-m+1, \ldots, k-1$ **do**
    $\beta \leftarrow \rho_i y_i^T r$;
    $r \leftarrow r - s_i(\alpha_i - \beta)$;
  end for
  $p_k \leftarrow -r$;
  $\alpha_k \leftarrow$ LineSearch(l, 1, p_k, ω_k);
  $\omega_{k+1} \leftarrow \omega_k + \alpha_k p_k$;
  if $k > m$ **then**
    Удалить пару (s_{k-m}, y_{k-m}) из хранилища;
  end if
  Вычислить и сохранить в хранилище пару $(s_k = \omega_{k+1} - \omega_k, y_k = \nabla L(\omega_{k+1}) - \nabla L(\omega_k))$;
  $k \leftarrow k + 1$
until convergence

Алгоритм 6 Линейный поиск с возвратом

function BacktrackingLineSearch(l, α, p, ω)
  $k \leftarrow 0$
  $\alpha_0 \leftarrow \alpha$
  while not $(l(\omega + \alpha_k p) \leq l(\omega) + c_1 \alpha_k \nabla l(\omega)^T p$ and $|\nabla l(\omega + \alpha_k p)^T p| \geq c_2 |\nabla l(\omega)^T p|)$ **do**
    $\alpha_{k+1} \leftarrow \frac{\alpha_k}{m}$
  end while
end function

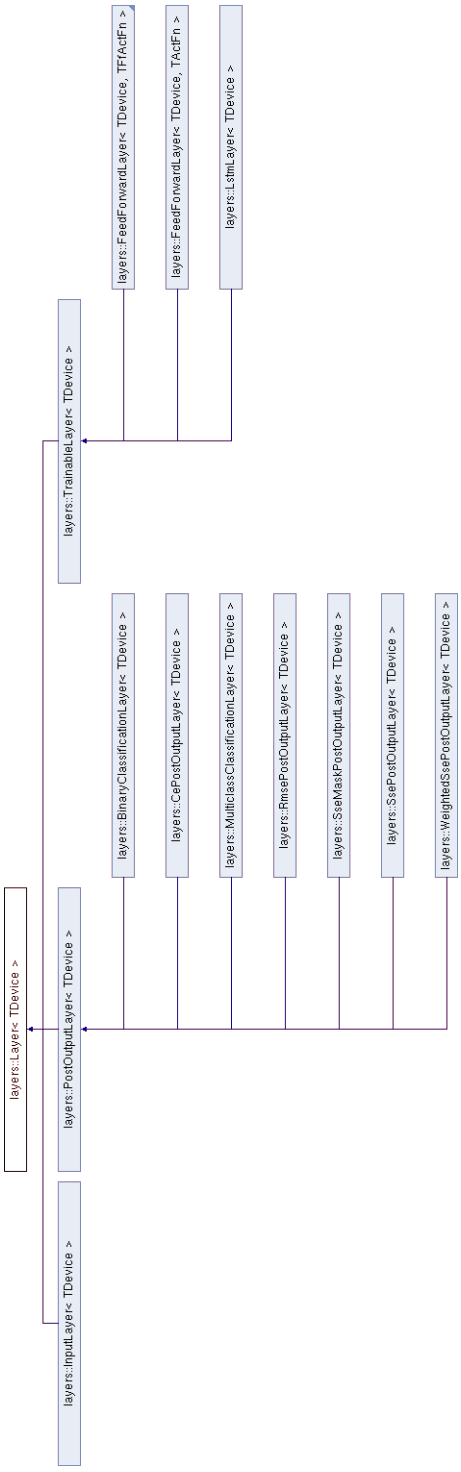

Рис. 6: Иерархия слоёв CURRENNT