Александр Биданец

АНАЛИЗ И ВИЗУАЛИЗАЦИЯ НЕЙРОННЫХ СЕТЕЙ С ВНЕШНЕЙ ПАМЯТЬЮ

Симферополь
«Полипринт»
2021

УДК 004.032.26
ББК 16.63
Б 59

БИДАНЕЦ А.
Б 59 **Анализ и визуализация нейронных сетей с внешней памятью**. – Симферополь, Полипринт, 2021. – 98 с.

ISBN 978-5-6046943-8-1

В данной книге детально описаны принципы работы нейросетевых моделей: нейронная машина Тьюринга, дифференцируемый нейронный компьютер, а также модификации последнего. Перечислены сферы применимости этих моделей. Выделены преимущества по сравнению с более ранней успешной моделью LSTM. Описаны недостатки этих моделей, а также способы их устранения. Дано теоретическое обоснование того факта, что перечисленные нейронные сети с внешней памятью, обладают большим потенциалом для решения многих задач, чем LSTM.

УДК 004.032.26
ББК 16.63

ISBN 978-5-6046943-8-1

Оглавление

Введение

Рекуррентные нейронные сети (RNN) отличаются от других методов машинного обучения тем, что они способны обрабатывать серии событий во времени или последовательные логические цепочки. Рекуррентные нейронные сети могут использовать свою внутреннюю память для обработки последовательностей разной длины. RNN применимы в таких задачах как, например: распознавание рукописного текста, анализ текстов, распознавание речи и др [19]. Кроме того, известно, что RNN являются полными по Тьюрингу [4, 5], и поэтому имеют возможность имитировать произвольные программные процедуры. Но на практике это не всегда просто сделать.

Рекуррентные нейронные сети хорошо справляются с задачами обучения на последовательностных данных и с задачами обучения с подкреплением, но очень ограничены в возможностях для решения задач, связанных с работой со структурами данных и переменными, а также хранением данных в течение длинных временных промежутков из-за отсутствия долгосрочной памяти.

Одним из способов улучшения стандартных рекуррентных сетей для успешного решения алгоритмических задач является введение адресной памяти большого размера. В отличие от машины Тьюринга, нейронная машина Тьюринга (NTM) является полностью дифференцируемой моделью, которая может быть обучена модификациями метода градиентного спуска (например, RMSProp), что дает практический механизм для обучения программ на примерах.

Модель NTM была предложена в 2014-ом году в работе [1]. В этой работе не описаны подробно детали функционирования данной нейросетевой модели. Одной из задач выпускной квалификационной работы является предоставление детального описания работы нейронной машины Тьюринга.

Основным фактором появления нейронных сетей с внешней памятью является изобретение дифференцируемых механизмов внимания [33, 34, 35, 36].

В 2016-ом году в работе [2] была предложена усовершенствованная модель нейронной сети с внешней памятью под названием дифференцируемый нейронный компьютер. В ней также было лишь краткое описание принципов работы этой модели.

В 2018-ом году в работе [29] были предложены четыре модификации для

дифференцируемого нейронного компьютера, которые позволяли улучшить качество решения задач, связанных с вопросно-ответными системами (QA-tasks). Эти модификации были основаны на работах [31, 32].

На сегодняшний день очень высока актуальность создания новых рекуррентных нейросетевых моделей, способных хранить большие объёмы данных, а также успешно решать задачи, предъявляемые к вопросно-ответным системам (QA-задачи).

К таким нейросетевым моделям предъявляются следующие требования:

– наличие «долгосрочной» обучаемой памяти;
– высокая скорость обучения;
– устойчивость процесса обучения (процесс обучения не должен существенно зависеть от начальной инициализации);
– прозрачность принятия решений моделью и интерпретируемость работы нейронной сети (попытка уйти от концепции «черного ящика»);
– способность решать QA-задачи;
– модель должна содержать относительно небольшое количество обучаемых параметров;
– способность работать с переменными, а также со структурами данных (например, с графами), решать алгоритмические задачи.

Объектом исследования данной работы являются нейронные сети с внешней памятью, а именно нейронная машина Тьюринга и дифференцируемый нейронный компьютер.

Предметом исследования является подробный анализ вышеуказанных нейросетевых моделей, а также их применимость к решению задач, связанных с построением вопросно-ответных систем.

Целью работы является подробное теоретическое и экспериментальное исследование вышеуказанных нейросетевых моделей, а также разработка приложения для визуализации внутренних процессов дифференцируемого нейронного компьютера.

В работе были поставлены следующие задачи:

– подробно описать работу нейронной машины Тьюринга, дифференцируемого нейронного компьютера (DNC), модификаций DNC, а также простроить необходимые схемы;
– отметить преимущества и недостатки этих моделей;
– сравнить качество работы этих моделей по отношению к более ранней успешной модели LSTM на примере двух задач: копирования последо-

вательностей битовых векторов и bAbi-task (базовые навыки вопросно-ответных систем);

– разработать приложение для визуализации внутренних процессов модели дифференцируемого нейронного компьютера;

В первом разделе предоставляются общие сведения по теме рекуррентных нейронных сетей.

Во втором разделе даётся подробное описание структуры и процесса обучения нейронной машины Тьюринга, перечисляются сферы применимости, достоинства и недостатки данной модели.

В третьем разделе даётся подробное описание структуры и процесса обучения дифференцируемого нейронного компьютера, перечисляются достоинства и недостатки данной модели.

В четвертом разделе описываются результаты вычислительных экспериментов для задач копирования битовых векторов и bAbi-task. Даётся сравнительный анализ работы моделей LSTM, NTM и DNC.

В разделе 5 описывается программное средство для визуализации DNC.

В разделе 6 перечисляются недостатки DNC.

В разделе 7 описываются модификации DNC.

Раздел 1. Общие сведения

1.1 Рекуррентные нейронные сети

Рекуррентные нейронные сети представляют собой широкий класс систем с динамическим состоянием. Текущее состояние таких систем зависит от текущего входного вектора (например, из обучающей выборки) и от предыдущего состояния. Динамическое состояние имеет решающее значение, поскольку оно дает возможность выполнять контекстно-зависимые вычисления: входной вектор, полученный сетью в данный момент может изменить её поведение в гораздо более поздний момент.

Рекуррентные сети легко обрабатывают структуры переменной длины. Поэтому они недавно использовались во множестве когнитивных задач, включая распознавание речи [16] [10]), генерация текста [13], генерация почерка [11] и машинный перевод [12].

Важнейшим новшеством для рекуррентных сетей было создание длинной краткосрочной памяти (LSTM) [8]. Эта архитектура была разработана для решения проблемы «исчезновения и взрыва градиента» [14]. LSTM решает данную проблему путем внедрения специальных вентилей [27].

Современный этап развития рекуррентных нейронных сетей характеризуется внедрением внешней памяти. Адресация внешней памяти основана на недавно изобретённых дифференцируемых моделях внимания. Фактором появления таких моделей внимания послужили исследования рабочей памяти человека, см. раздел 1.2.

Введение внешней памяти сокращает значительную часть пространства поиска, поскольку теперь мы просто ищем RNN, которая может обрабатывать информацию, хранящуюся за её пределами. Это усечение пространства поиска позволяет нам начать использовать некоторые из возможностей RNN, которые были недоступны ранее, что видно из множества задач, которые может изучить нейронная машина Тьюринга: от копирования входных последовательностей после их просмотра, до эмуляции N-грамм, реализации вопросно-ответных систем [9], решения задачи обхода и ориентирования на графе, а также поиска кратчайшего пути в графе и решения задач, связанных с обучением с подкреплением [2].

Приведём список названий некоторых нейронных сетей с внешней памятью:

– Графовая нейронная сеть (Graph Neural Network)
– Нейронная стек машина (нейронный стек, Neural stack machine)
– Нейронная очередь (neural queue)
– Нейронный дек (neural deque)
– Сеть указателей (Pointer Network)
– Сквозная сеть памяти (End-to-end memory network)
– Сеть памяти (Memory network)
– Динамическая сеть памяти (Dynamic memory network)
– Нейронная карта (Neural map)
– Нейронная машина Тьюринга (Neural Turing Machine)
– Дифференцируемый нейронный компьютер (Differentiable Neural Computer)

1.2 Концепция рабочей памяти

Концепция рабочей памяти наиболее сильно развита в психологии. Она объясняет процесс выполнение задач, связанных с кратковременным манипулированием информацией. Принцип заключается в том, что «центральный исполнительный орган» фокусирует внимание и выполняет операции с данными в буфере памяти [6]. Психологи широко изучили ограничения емкости рабочей памяти человека, которые часто количественно оценивают по количеству фрагментов информации, которую можно легко запомнить [7]. Эти ограничения емкости приводят к пониманию ограничений в работе памяти человека, но в моделях NTM и дифференцируемого нейронного компьютера (DNC) размер рабочей памяти можно задавать произвольным образом.

Раздел 2. Нейронная машина Тьюринга

2.1 Общий принцип работы NTM

Архитектура NTM содержит два основных компонента: контроллер (нейронная сеть) и матрицу внешней памяти (см. рис. 2.1). Как и большинство нейронных сетей, контроллер взаимодействует с внешним миром через входные и выходные векторы. В отличие от стандартных сетей, он также взаимодействует с матрицей памяти, используя операции чтения и записи. Дополнительные выходы нейронной сети являются параметрами этих операций, и, по аналогии с машиной Тьюринга, будем считать их параметрами «головки».

Существенно, что каждый компонент архитектуры является дифференцируемым, что делает NTM простой моделью для обучения при помощи модифицированных методов градиентного спуска. Это достигается за счет определения «размытых» операций чтения и записи, которые в большей или меньшей степени взаимодействуют со всеми элементами памяти (вместо того, чтобы обращаться к одному элементу, как в обычной машине Тьюринга или на цифровом компьютере). Степень размытости определяется механизмом фокусировки внимания, который ограничивает каждую операцию чтения и записи взаимодействием с небольшой частью памяти, игнорируя остальные её части. Слоты памяти, входящие в фокус внимания операций чтения и записи, определяются специализированными величинами (скалярными и векторными) – выходами, исходящими от головок. Эти величины определяют нормированный вес (распределение) по строкам в матрице памяти. Весовой вектор определяет степень, в которой головка считывает (или записывает). Таким образом, головка может сосредоточенно обращаться к памяти в одном месте или слабо влиять во многих местах.

Основной принцип работы NTM заключается в том, чтобы предоставить нейронной сети доступ к внешней памяти (по аналогии с моделью машины Тьюринга, в которой имеются контроллер, головки чтения и записи, а также внешняя память). Эта модификация позволяет увеличить размер «долгосрочной» памяти модели.

Нейронная машина Тьюринга (NTM) обучается как обычная нейронная сеть используя входные данные, получаемые из выборки, но также обучается как ранить эту информацию и когда её извлекать из внешней памяти.

Вспомогательные выходы контроллера: вектор ключа, сила ключа, вен-

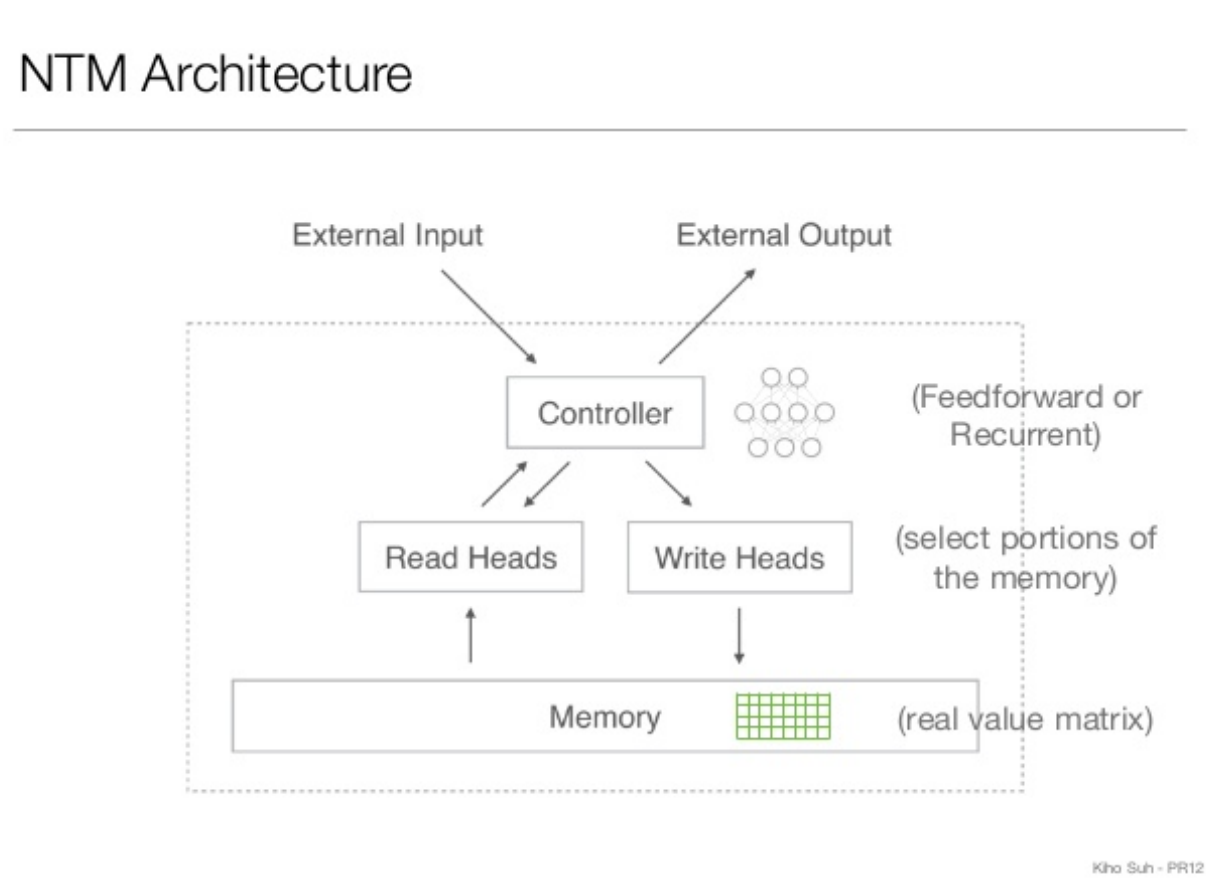


Рис. 1: Общая схема нейронной машины Тьюринга

тиль интерполяции, вектор свёрточного сдвига, коэффициент «заострения (или уточнения)».

Нейронная машина Тьюринга, являющаяся рекуррентной моделью, может решать задачи, которые решают обычные рекуррентные нейронные сети, но также и алгоритмические задачи, такие как: копирование, сортировка, ассоциативный поиск на основе примеров, взятых из обучающей выборки [1].

Предположим, что мы индексируем память, указав строку и колонку, как в обычной матрице. Мы бы хотели обучить нашу нейросеть с помощью метода обратного распространения ошибки и некоторого метода оптимизации (например, RMSProp, см. раздел 2.6), но получить градиент определённого индекса внешней памяти вряд ли получится. Вместо этого контроллер осуществляет операции чтения и записи в рамках «размытых» операций, которые взаимодействуют со всеми элементами памяти в той или иной степени.

Рассмотрим принцип работы NTM более детально.

2.2 Компоненты и гиперпараметры

При инициализации NTM требуется задать некоторые гиперпараметры:

- размер внешней памяти: количество слотов памяти m и длина слота n
- диапазон перемещений головки r
- количество считывающих головок h_r
- количество записывающих головок h_w
- тип контроллера: RNN, LSTM, GRU, FFNN
- количество скрытых слоёв H и количество нейронов в скрытых слоях контроллера h_i

  $\mathbf{M}_0$ – начальное состояние матрицы внешней памяти. Её можно инициализировать случайным образом (обычно это делают при помощи стандартного нормального распределения с малой дисперсией).

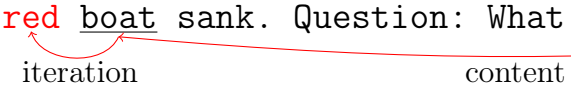
Нейронная машина Тьюринга состоит из контроллера *controller*, внешней памяти M, считывающих и записывающих головок, реализованных при помощи функций чтения, записи и механизмов адресации.

NTM использует комбинацию двух методов адресации: адресацию на основе контента (на основе содержимого) и механизм внимания, основанный на местоположении слотов в памяти.

Рассмотрим упрощённый пример использования адресации по контенту и адресации по местоположению для задачи понимания и анализа текста.

2.3 Пример

Context: The red boat sank. Question: What color was the boat?



boat → boat (content)

boat → red (iteration)

Сначала модель получает от пользователя предложение «Context». После этого пользователь задаёт вопрос «Question», в котором фигурирует слово «boat». Далее модель ищет в своей памяти слово «boat» за счет механизма поиска ассоциативных связей. В вопросе пользователя также фигурировало слово «color». За счет итеративного сдвига модель ищет ответ на вопрос. Ответом на вопрос является слово «red».

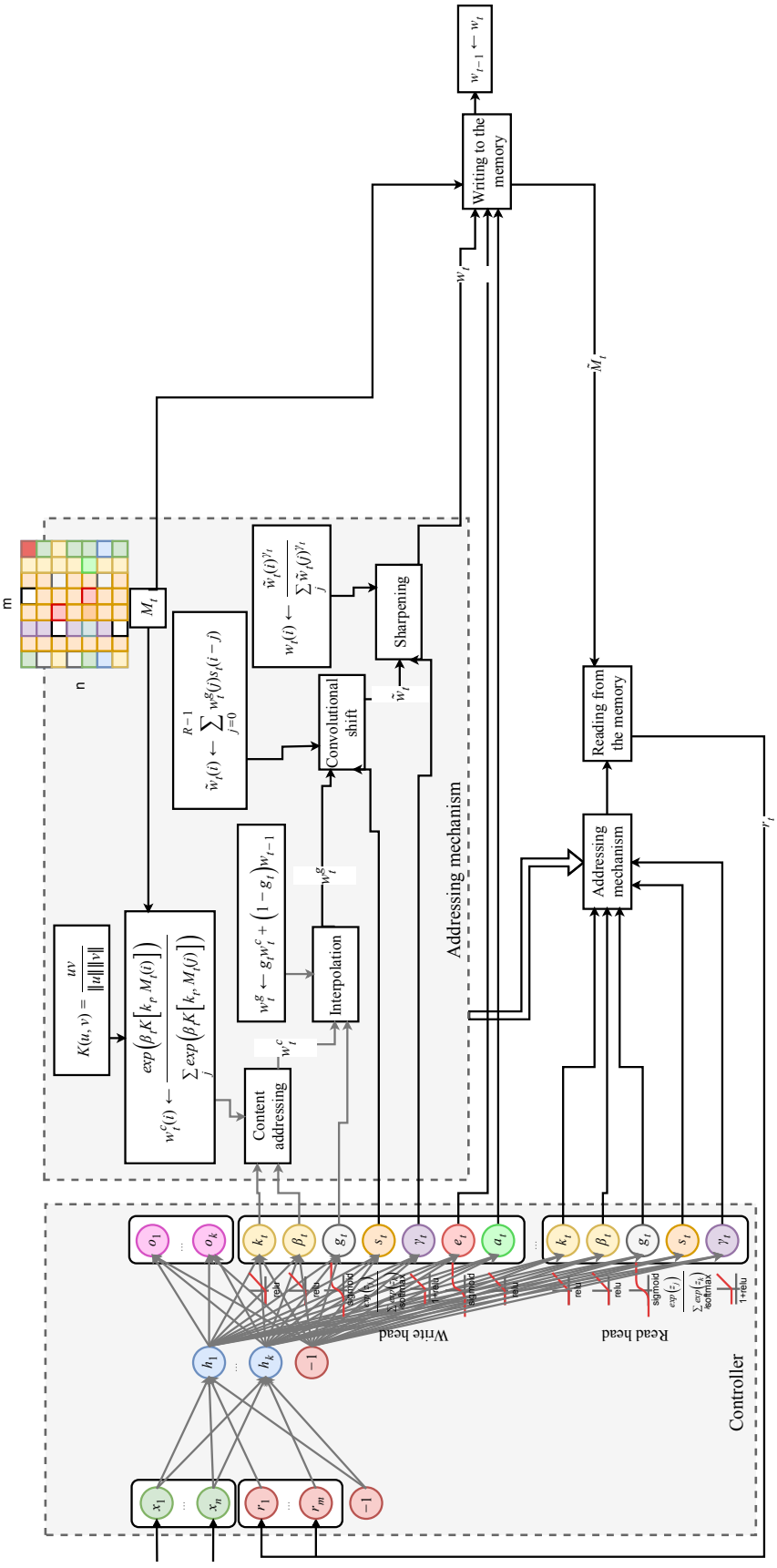


Рис. 2: Нейронная машина Тьюринга

2.4 Контроллер

Контроллер является обучаемой частью нейронной машины Тьюринга. Контроллер выполняет все необходимые вычисления, в том числе обеспечивает управление памятью в автоматическом режиме (без участия человека). В качестве контроллера может выступать нейронная сеть прямого распространения, любая рекуррентная нейронная сеть, в частности, LSTM или управляемый рекуррентный блок (gated recurrent unit, GRU).

На каждой итерации работы нейронной машины Тьюринга контроллер принимает на вход вектор x_t, поступающий из выборки (обучающей или тестовой) и вектор чтения r_t, полученный при чтении предыдущего состояния внешней памяти.

Входной слой имеет размер:

$controller\_input\_shape = d + h_r * m$, где d – длина входного вектора, h_r – количество считывающих головок, m – длина слота внешней памяти.

Выходной слой контроллера имеет размер

$controller\_output\_shape = d^{'} + (h_r + h_w) * (m + r + 3) + h_w * 2 * m$, где $d^{'}$ – выходной вектор, m – длина слота внешней памяти, h_r – количество считывающих головок, h_w – количество записывающих головок, r – длина вектора сверточного сдвига.

2.5 Механизм адресации памяти

Отметим, что контроллер не знает длину m матрицы внешней памяти. Эта величина связана только с механизмом адресации памяти.

На каждой итерации работы нейронной машины Тьюринга взаимодействие происходит целиком со всей матрицей внешней памяти. Степень взаимодействия с каждой строкой (слотом) матрицы определяется весовым вектором $w_t \in \mathbb{R}^n$ (для каждой считывающей и записывающей головки свой весовой вектор). Этот весовой вектор рассчитывается в зависимости от параметров головки, получаемых от контроллера, а также текущего состояния памяти M_t и предыдущего весового вектора w_{t-1}.

Процесс создания таких весовых векторов для определения мест, где следует считывать и записывать данные можно представить в виде четырёх стадий. На каждой стадии генерируется промежуточный весовой вектор, который передаётся на следующую стадию (см. рис 4. приложений).

2.5.1 Адресация по контенту (Content Addressing)

Цель первой стадии – сгенерировать весовой вектор на основании того, насколько близка каждая строка в памяти к length-N вектору k_t, полученному от контроллера. Будем называть этот весовой вектор w_t^c весовым вектором контента (содержимого).

Весовой вектор контента позволяет контроллеру выбирать значения, похожие на уже знакомые значения, что называется адресацией по контенту. Для каждой головки контроллер производит вектор-ключ k_t, который сравнивается с каждой строкой M_t, используя меру сходства. В работе [1] использовалась косинус-мера сходства, которая определяется выражением (2).

Положительный скалярный параметр β_t называется прочностью ключа, используется для определения, насколько сконцентрирован должен быть весовой вектор контента. При малых значениях β_t весовой вектор будет размытым, а при больших значениях – будет сконцентрирован на наиболее похожей строке в памяти. Если ключ $k_t = [0.1, 0.5, 0.25, 0.1, 0.05]$, то весовой вектор контента будет изменяться в зависимости от β_t по закону, представленному на рис. 3 [26].

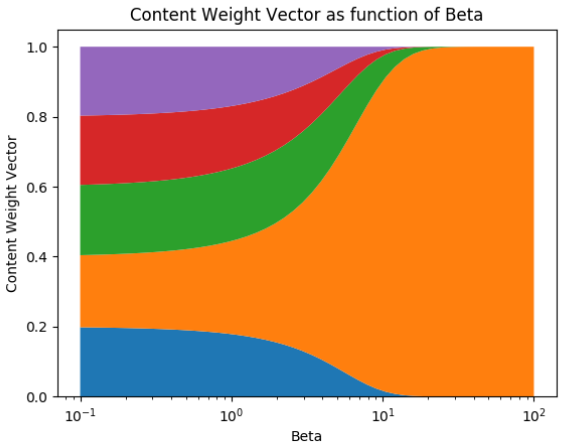


Рис. 3: Изменение вектора контента в зависимости от параметра β_t

Контроллер производит для каждой головки вектор ключа $k \in \mathbb{R}^m$, три скалярных величины $\beta, g, \gamma \in \mathbb{R}$ и весовой вектор $s \in \mathbb{R}^r$, называемый вектором сдвига. Также необходимо хранить предыдущий весовой вектор w_{t-1} и предыдущее состояние памяти $M_{t-1} \in \mathbb{R}^{m \times n}$. Данные преобразуются в текущий весовой вектор w_t:

Сначала вычисляется косинусная мера сходства между ключом k и

каждой строкой матрицы M_{t-1}. Затем к результату применяется операция softmax с множителем β в показателе экспоненты. Затем результирующий весовой вектор смешивается (путём выпуклой комбинации с параметром g) с весовым вектором предыдущей итерации w_{t-1}. Число g будем называть вентилем интерполяции. Затем последовательно выполняются операции свёрточного сдвига и «уточнения». Пример построения весового вектора представлен на рис. 7.9 приложений.

$$w_t^c(i) \leftarrow \frac{exp(\beta_t \mathbf{K}[k_t, M_t(i)])}{\sum\limits_j exp(\beta_t \mathbf{K}[k_t, M_t(j)])}. \tag{1}$$

Косинусная мера сходства определяется следующим образом:

$$\mathbf{K}[\mathbf{u}, \mathbf{v}] \leftarrow \frac{\mathbf{u}.\mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}. \tag{2}$$

Производные w_t^c по переменным β_t, $\mathbf{k}_t$ и $\mathbf{M}_t$:

$$\frac{\partial w_t^c(i)}{\partial \beta_t} \leftarrow \frac{exp(\beta_t c_i)\left(\sum\limits_j (c_i - c_j)\, exp(\beta_t c_j)\right)}{\left(\sum\limits_j exp(\beta_t c_j)\right)^2}, \text{ где} \tag{3}$$

$$c_i \leftarrow \mathbf{K}[\mathbf{k}_t, \mathbf{M}_t(i)]; \tag{4}$$

$$\text{Также, } \frac{\partial w_t^c(i)}{\partial c_i} \leftarrow \frac{\sum\limits_{j \neq i} \beta_t exp(\beta_t c_j)}{\left(\sum\limits_j exp(\beta_t c_j)\right)^2}, \tag{5}$$

$$\frac{\partial w_t^c(i)}{\partial c_j} \leftarrow \frac{-\beta_t exp(\beta_t c_i)\, exp(\beta_t c_j)}{\left(\sum\limits_j exp(\beta_t c_j)\right)^2}, \text{ где} \tag{6}$$

$$\frac{\partial \mathbf{K}[\mathbf{u}, \mathbf{v}]}{\partial u(i)} \leftarrow \frac{1}{\|\mathbf{u}\|\|\mathbf{v}\|}\left[v(i) - \frac{u(i)(\mathbf{u}.\mathbf{v})}{\|\mathbf{u}\|^2}\right], \text{ при} \tag{7}$$

$\mathbf{u} = \mathbf{k}_t$ и $\mathbf{v} = \mathbf{M}_t(i)$. Теперь мы можем использовать цепное правило для поиска производных w_t^c по $\mathbf{k}_t$ и $\mathbf{M}_t$.

Однако в некоторых случаях нам необходимо уметь извлечь значения из конкретных ячеек памяти, а не прочитать конкретные значения в памяти. Это называется адресация по ячейкам (по местоположению). А для её реализации нам нужны еще три этапа адресации.

2.5.2 Интерполяция (Interpolation)

На втором этапе скалярный параметр $g_t \in (0,1)$, который называется вентилем интерполяции (interpolation gate), смешивает в некоторой пропорции вектор контента w_t^c с весовым вектором предыдущего шага работы NTM w_{t-1} для производства вентильного весового вектора w_t^g. Это позволяет системе понять, когда использовать (или игнорировать) адресацию по контенту (и в какой степени)

$$w_t^g \leftarrow g_t w_t^c + (1 - g_t) w_{t-1}. \tag{8}$$

Выпишем частные производные весового вектора интерполяции по параметрам g, w_t^c, w_{t-1}:

$$\frac{\partial w_t^g(i)}{\partial g_t} \leftarrow w_t^c(i) - w_{t-1}(i), \tag{9}$$

$$\frac{\partial w_t^g(i)}{\partial w_t^c(j)} \leftarrow g_t \mathbb{I}(i=j), \tag{10}$$

$$\frac{\partial w_t^g(i)}{\partial w_{t-1}(j)} \leftarrow (1 - g_t) \mathbb{I}(i=j). \tag{11}$$

2.5.3 Сверточный сдвиг (Convolutional shift)

Также необходимо ввести возможность смещения фокуса на другие строки. Предположим, что одним из системных параметров ограничен диапазон допустимых смещений. Например, внимание головки может сместиться вперед на одну строку (+1), остаться без изменений (0) или сместиться на одну строку назад (-1). Будем производить сдвиги по модулю m, т.е. сдвиг вперёд с нижней строки памяти перемещает внимание головки на верхнюю строку, а сдвиг назад верхней строки перемещает внимание головки на нижнюю строку. После операции вентильной интерполяции каждая головка производит нормированное взвешивание (распределение) сдвига s_t и происходит следующее перемещение (при помощи операции свертки) для расчёта веса сдвига $\tilde{w}_t$:

$$\tilde{w}_t(i) = \sum_{j=0}^{N-1} w_t^g(j)\, s_t(i-j). \tag{12}$$

Выпишем частные производные $\tilde{w}_t$ по параметрам s_t и w_t^g:

$$\frac{\partial \tilde{w}_t(i)}{\partial s_t(k)} \leftarrow \sum_{k=0}^{N-1} w_t^g(i-k), \tag{13}$$

$$\frac{\partial \tilde{w}_t(i)}{\partial w_t^g(j)} \leftarrow \sum_{j=0}^{N-1} s_t(i-j). \tag{14}$$

Пример вычисления операции свёрточного сдвига изображён на рис. 8 приложений.

2.5.4 Заострение внимания, уточнение (Sharpening)

Четвёртая и окончательная стадия, «уточнение» (15), используется чтобы предотвратить размытие веса (распределения) $\tilde{w}_t$ из-за операции сверточного сдвига. Для этого требуется скаляр $\gamma \geq 1$. Снова используем операцию $softmax$.

$$w_t(i) \leftarrow \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_j \tilde{w}_t(i)^{\gamma_t}}. \tag{15}$$

Производные w_t по $\tilde{w}_t$ и по γ_t:

$$\frac{\partial w_t(i)}{\partial \gamma_t} \leftarrow \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_j \tilde{w}_t(i)^{\gamma_t}} \left[ln\left(\tilde{w}_t(i)\right) - \frac{\sum_j ln\left(\tilde{w}_t(j)\right) \tilde{w}_t(j)^{\gamma_j}}{\sum_k \tilde{w}_t(k)^{\gamma_t}} \right], \tag{16}$$

$$\frac{\partial w_t(i)}{\partial \tilde{w}_t(i)} \leftarrow \frac{\gamma_t \tilde{w}_t(i)^{\gamma_t - 1}}{\sum_j \tilde{w}_t(j)^{\gamma_t}} \left[1 - \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_k \tilde{w}_t(k)^{\gamma_t}} \right], \text{при } i = j. \tag{17}$$

$$\frac{\partial w_t(i)}{\partial \tilde{w}_t(j)} \leftarrow \frac{-\gamma_t \tilde{w}_t(i)^{\gamma_t} \tilde{w}_t(j)^{\gamma_t - 1}}{\sum_k \tilde{w}_t(k)^{\gamma_t}}, \text{при } i \neq j, \tag{18}$$

где $\tilde{w}_t$ – весовой вектор, полученный после операции сверточного сдвига.

Пример вычисления операции «уточнение» представлен на рис. 9 приложений.

2.5.5 Чтение из памяти

Пусть M_t – матрица внешней памяти с m строками и n столбцами в момент времени t. Чтобы осуществить чтение (и запись) требуется некий механизм внимания, который определяет, откуда головка должна считать данные. Механизм внимания будет нормированным по длине m весовым вектором w_t.

Под «нормированием» подразумевается соблюдение двух следующих ограничений:

$$0 \leq w_t(i) \leq 1, \tag{19}$$

$$\sum_{i=1}^{R} w_t(i) = 1. \tag{20}$$

Операция чтения довольно проста. Для неё мы будем использовать полученный весовой вектор $w_r \in \mathbb{R}^n$. Каждую строку $M(i)$ матрицы M домножим на соответствующий коэффициент весового вектора $w_t(i)$, и, затем, элементы каждой строки просуммируем. В результате получим новый вектор, каждая компонента которого соответствует взвешенной сумме элементов соответствующего столбца матрицы M.

Головка чтения вернёт вектор r_t, который является выпуклой комбинацией строк памяти $M_t(i)$, масштабированных весовым вектором:

$$r_t \leftarrow \sum_i w_t(i)\,\mathbf{M_t}(i). \tag{21}$$

r_t дифференцируем по внешней памяти и по весовому вектору. Производная $\mathbf{r}_t$ по $\mathbf{w}_t$:

$$\frac{\partial r_t(i)}{\partial w_t(j)} = \mathbf{M}_t(i,j), \tag{22}$$

и по $\mathbf{M}_t$:

$$\frac{\partial r_t(k)}{\partial \mathbf{M}_t(i,j)} = w_t(i)\,\mathbb{I}(k=j). \tag{23}$$

Пример выполнения операции чтения представлен на рис. 5 приложений.

2.5.6 Запись в память

Запись немного сложнее, чем чтение, поскольку включает в себя два отдельных шага: стирание, а затем добавление. Чтобы стереть старые данные, записывающей головке нужен новый вектор – это length-N стирающий вектор e_t. Стирающий вектор используется в конъюнкции с весовым вектором для определения, какие элементы в строке следует удалить, оставить неизменными или что-то среднее.

После преобразования всеми записывающими головками M_{t-1} в M_t^{erased} записывающая головка использует length-N добавляющий вектор a_t для завершения операции записи (28).

На каждом шаге t записывающая головка производит весовой вектор w_t, а также стирающий вектор (erase vector) e_t. Размер вектора w_t равен M. Размер вектора e_t равен n. Каждая компонента стирающего вектора лежит в интервале $(0, 1)$ (контроллер имеет сигмоидные функции активации для компонент этого вектора). Векторы памяти $M_{t-1}(i)$ с предыдущего шага модифицируются следующим образом:

$$\tilde{M}_t(i) \leftarrow M_{t-1}(i)\left[1 - w_t(i)\, e_t\right], \text{ где } \mathbf{1} - \text{вектор-строка}, \tag{24}$$

все компоненты которой – единицы.

Следовательно, элементы ячейки памяти сбрасываются на ноль только в том случае, когда весовой вектор и элемент стирающего вектора являются единичными; если либо компонента весового вектора равна нулю, либо компонента стирающего вектора равна нулю, память остается неизменной.

При наличии нескольких записывающих головок, операции стирания могут выполняться в любом порядке, поскольку умножение является коммутативным.

Но вначале должны быть выполнены все операции стирания по всем записывающим головкам, а уже затем должны быть выполнены все операции добавления (add) к памяти по всем записывающим головкам.

Переменная $\tilde{M}_t$ является дифференцируемой по M_t, w_t и e_t, и соответствующие частные производные такие:

$$\frac{\partial \tilde{M}_t(i,j)}{\partial M_t\left(i^{'}, j^{'}\right)} = \left(1 - w_t(i)\, e_t(j)\right) \mathbb{I}\left(i = i^{'}\right) \mathbb{I}\left(j = j^{'}\right), \tag{25}$$

$$\frac{\partial \tilde{M}_t(i,j)}{\partial w_t(k)} = -M_{t-1}(i,j)\, e_t(j)\, \mathbb{I}(i = k), \tag{26}$$

$$\frac{\partial \tilde{M}_t(i,j)}{\partial e_t(k)} = -M_{t-1}(i,j)\, w_t(i)\, \mathbb{I}(k = j). \tag{27}$$

Операция стирания происходит за счет умножения матрицы M на $(1 - w_w * e)$.

Каждая записывающая головка также производит добавляющий вектор a_t длинной n (slot length), который прибавляется к памяти после шага стирания

$$M_t(i) \leftarrow \tilde{M}_t(i) + w_t(i)\, a_t. \tag{28}$$

Добавляющий вектор – это новая запись (новая информация).

Примеры использования вектора добавления и стирающего вектора представлены на рисунках 6, 7 приложений соответственно.

Порядок, в котором выполняются операции добавления к памяти, не имеет значения. Объединенные операции стирания и добавления для всех головок записи дают окончательное содержимое памяти в момент времени t.

M_t дифференцируем в соответствии w_t, a_t и e_t, и соответствующие частные производные такие:

$$\frac{\partial M_t(i,j)}{\partial \tilde{M}_t\left(i^{'}, j^{'}\right)} = \mathbb{I}\left(i = i^{'}\right)\mathbb{I}\left(j = j^{'}\right), \tag{29}$$

$$\frac{\partial M_t(i,j)}{\partial w_t(k)} = w_t(i)\,\mathbb{I}(j = k), \tag{30}$$

$$\frac{\partial M_t(i,j)}{\partial w_t(k)} = a_t(j)\,\mathbb{I}(i = k). \tag{31}$$

2.6 RMSProp (Root Mean Square Propagation)

RMSProp – это одна из модификаций стохастического градиентного спуска [15].

Это метод, в котором темп обучения (learning rate) адаптируется по каждому параметру. Идея заключается в том, чтобы делить темп обучения на среднее значения величин последних градиентов по каждому параметру.

RMSProp использует скользящее среднее квадратов градиента для нормализации градиента.

Итак, сначала вычисляется скользящее среднее значение квадрата градиента для каждого веса w:

$$MeanSquare(w,t) := \gamma MeanSquare(w,t-1) + (1-\gamma)\left(\frac{\partial Q_i(w)}{\partial w}\right)^2, \tag{32}$$

где γ – это множитель «забывания».

И параметры обновляются следующим образом:

$$w := w - \frac{\eta}{\sqrt{MeanSquare(w,t)}}\frac{\partial Q_i(w)}{\partial w}, \tag{33}$$

где η – темп обучения.

RMSProp показывает отличный уровень адаптации темпа обучения в различных приложениях. RMSProp может рассматриваться как обобщение Rprop и способен работать с пакетами малого размера (mini-batches).

2.7 Процесс обучения нейронной машины Тьюринга

Уточним условия процесса обучения NTM.

Наиболее часто в качестве метода обучения NTM используют не обычный метод обратного распространения ошибки с методом стохастического градиента, а RMSProp, описанный в разделе 2.6.

Обучение NTM производится методом обратного распространения ошибки, используя частные производные, описанные в разделе 2.5.

В данной работе в качестве контроллера NTM будет использоваться многослойный перцептрон. Вариант NTM, в качестве контроллера которой используется LSTM, представлен на рис. 12 приложений.

Пусть дана обучающая выборка $X^l = \{(x_1, y_1), (x_2, y_2), \ldots, (x_l, y_l)\}$

Пусть C – скрытый слой контроллера, C_t – выходы скрытого слоя в момент времени t, C_0 – выходы скрытого слоя в момент времени 0.

Скрытый или несколько промежуточных слоев контроллера являются промежуточным звеном для генерации весового вектора w_t, а для записывающих головок – еще и для генерации стирающего вектора e_t и добавляющего вектора a_t.

Параметры W_{eC} и b_{eC} используются для генерации вектора e_t, а параметры W_{aC} и b_{aC} используются для генерации вектора a_t.

Здесь компоненты добавляющего вектора a_t приравниваются нулю, если их значения выходят за границы отрезка $[-1, 1]$.

$$e_t \leftarrow \sigma\left(\mathbf{W_{eC}C_t} + \mathbf{b_{eC}}\right), \tag{34}$$

$$a_t \leftarrow \left(\mathbf{W_{aC}C_t} + \mathbf{b_{aC}}\right). \tag{35}$$

$\mathbf{M_t}$ – матрица памяти в момент времени t, которая обновляется до M_{t+1}, применяя равенства (24) и (28), которые используют весовой вектор записи $\mathbf{w_{w,t}}$, стирающий вектор $\mathbf{e_t}$ и добавляющий вектор $\mathbf{a_t}$.

Вектор чтения r_t строится при помощи уравнения (21), которое использует обновленную матрицу памяти $\mathbf{M}_t$ и весовой вектор $w_{r,t}$.

Входной вектор x_t и вектор чтения r_t используются для обновления нейронов скрытого слоя C_t, используя параметры W_{CX}, b_{CX}, W_{Cr} и b_{Cr} получаем

$$C_t \leftarrow relu\left(\mathbf{W_{CX}} X_t + \mathbf{b_{CX}} + \mathbf{W_{Cr}} r_t + \mathbf{b_{Cr}}\right). \tag{36}$$

Используя обновленный скрытый слой C_t и набор параметров:
$\left\{\mathbf{W_{PC}}, \mathbf{b_{PC}}, \mathbf{W_{k_uC}}, \mathbf{b_{k_uC}}, \mathbf{W}_{\beta_\mathbf{u}\mathbf{C}}, \mathbf{W_{g_uC}}, \mathbf{b_{g_uC}}, \mathbf{W_{s_uC}}, \mathbf{b_{s_uC}}, \mathbf{W}_{\gamma_\mathbf{u}\mathbf{C}}, \mathbf{b}_{\gamma_\mathbf{u}\mathbf{C}}\right\}$,
получим внешний выход контроллера P_t (предположим, что $y_1 \in [0,1]^k, \ldots, y_l \in [0,1]^k$), набор выходов контроллера для обновления весового вектора чтения (обозначим как $\mathbf{H_{r,t}}$) и набор выходов контроллера для обновления весового вектора записи (обозначим как $\mathbf{H_{w,t}}$). Здесь $\mathbf{H_{u,t}}$ включает вектор-ключ $\mathbf{k_{u,t}}$, силу ключа (скалярная величина) $\beta_{u,t}$, вентиль интерполяции (скалярная величина) $g_{u,t}$, свёрточный сдвиг (векторная величина) $s_{u,t}$, коэффициент «заострения» $\gamma_{u,t}$, где $u = r$ для весового вектора чтения и $u = w$ для весового вектора записи.

Отметим, что вектор ключа обрезается в диапазоне от -1 до 1.

$$P_t \leftarrow \sigma\left(\mathbf{W_{PC}} C_t + \mathbf{b_{PC}}\right), \tag{37}$$

$$\mathbf{k_{u,t}} \leftarrow \mathbf{W_{k_uC}} C_t + \mathbf{b_{k_u}C}, \tag{38}$$

$$\beta_{u,t} \leftarrow relu\left(\mathbf{W}_{\beta_\mathbf{u}\mathbf{C}} C_t + \mathbf{b}_{\beta_\mathbf{u}}\mathbf{C}\right), \tag{39}$$

$$g_{u,t} \leftarrow \sigma\left(\mathbf{W_{g_uC}} C_t + \mathbf{b_{g_u}C}\right), \tag{40}$$

$$\mathbf{s_{u,t}} \leftarrow \sigma\left(\mathbf{W_{s_uC}} C_t + \mathbf{b_{s_u}} C\right), \tag{41}$$

$$\gamma_{u,t} \leftarrow relu\left(\mathbf{W}_{\gamma_\mathbf{u}\mathbf{C}} C_t + \mathbf{b}_{\gamma_\mathbf{u}}\mathbf{C}\right). \tag{42}$$

Обновленный скрытый слой также производит стирающий и добавляющий вектор, используя уравнения (34) и (35). Также обновленный скрытый слой производит обновленный весовой вектор записи. Эти три вектора используются для обновления памяти.

Для каждого входного вектора x_t нейронная машина Тьюринга производит выходной вектор P_t. Функция потерь (ошибка) $Loss_t$ вычисляется

между внешним выходом P_t и целевым вектором y_t. Выбор функции потерь определяется задачей.

Если мы предположим, что компоненты целевого вектора лежат в отрезке $[0,1]$, то выбор бинарной кросс-энтропии в качестве функции потерь является подходящим выбором

$$Loss_t = binary\_crossentropy\,(P_t, y_t)\,. \tag{43}$$

Пусть P – матрица спрогнозированных внешних выходов, состоящая из векторов $[P_1, P_2, \ldots, P_l,]$ и Y – матрица целевого внешнего выхода, состоящая из векторов $[y_1, y_2, \ldots, y_l]$

Теперь мы можем вычислить ошибку в соответствии с полной выходной последовательностью, вычислив бинарную кросс-энтропию между P и Y:

$$L = binary\_cross - entropy\,(P, Y)\,. \tag{44}$$

$$L_{binary\_cross\_entropy}\,(y, p\,(\Theta)) = -\sum_i y_i log\,(p\,(\Theta)_i)\,,$$

где y – это правильный ответ (вектор в форме унитарного кода).

В вероятностном смысле функция потерь $L2$ (Евклидово расстояние) соответствует отрицательному логарифмическому правдоподобию нормального распределения со средним значением $p(\Theta)$, функция потерь $L1$ (функция потерь Лапласса) соответствует отрицательному логарифмическому правдоподобию распределения по Лапласу со средним $p(\Theta)$, а бинарная функция потерь кросс-энтропии соответствует отрицательному логарифмическому правдоподобию полиномиального распределения с вероятностями классов, заданными вектором $p(\Theta)$.

Теперь, частная производная величины L вычисляется по каждому параметру модели при помощи правила вычисления сложной функции (цепного правила). После вычисления производных L в соответствии с каждым параметром, параметры модели обновляются используя RMSProp.

Процесс обучения продолжается до тех пор, пока не выполнен критерий останова.

2.8 Один шаг процесса обучения NTM

1. Контроллер принимает на вход очередной объект выборки x_t, а также вектор чтения r_t.

2. Контроллер производит вектор, необходимый для построения весового вектора для операций чтения и записи, а также контроллер производит векторы записи (add_vector, $erase\_vector$) для каждой записывающей головки.

3. Контроллер вычисляет целевой вектор $\hat{y}_t$.

4. Обратное распространение ошибки для обучения контроллера.

5. При помощи механизма адресации и параметров, полученных на шаге 2 вычисляются весовые векторы для головок чтения и записи.

6. Производится операция записи во внешнюю память, и чтения, а затем снова переходим к шагу 1.

Рассмотрим решения нескольких задач при помощи нейронной машины Тьюринга. Поскольку все далее описанные задачи будут эпизодическими, то необходимо во время процессов обучения и тестирования сбрасывать динамическое состояние (внутреннюю память) сетей в начале каждой входной последовательности. Для сетей LSTM это означает установку (предыдущего скрытого состояния равным настроенному в процессе обучения вектору смещения (bias)). Для NTM предыдущее состояние контроллера, значение (предыдущих векторов чтения и содержимое памяти были сброшены до значений настроенного bias. Все задачи были задачами обучения с учителем с бинарными целями; все сети имели логистические сигмоидные выходные слои и были обучены с целевой функцией кросс-энтропии. Ошибки прогнозирования последовательности сообщаются в битах на последовательность.

Рассмотрим задачу копирования последовательности битовых векторов.

2.9 Области применения NTM

- Обработка естественного языка (например, машинный перевод, вопросно-ответные системы).
- Прогнозирование временных рядов
- Распознавание рукописного текста
- Распознавание речи

– Генерация текстов
– Алгоритмические задачи
– Логические игры (при помощи обучения с подкреплением)

2.10 Недостатки нейронной машины Тьюринга

– Медленный процесс обучения [3, 18].
– Скорость обучения зависит от начальной инициализации внешней памяти и весовых коэффициентов контроллера.
– Размер памяти является гиперпараметром, как следствие необходимо несколько запусков процесса обучения для определения оптимального размера памяти.
– Косинусная мера сходства в качестве результата возвращает NaN каждый раз, когда нулевой вектор фигурирует в вычислениях. В работе [3] эта проблема решается при помощи добавления ϵ в знаменателе в случае, если на вход был подан нулевой вектор.
– Операция «уточнение» может привести к результату **Inf/Inf**, если γ слишком большое. Проблему с большим γ можно решить при помощи простого отсеченния, если результат операции выходит за границы заданного диапазона.

Выводы по разделу

В этом разделе была рассмотрена архитектура и процесс обучения NTM. Главным преимуществом NTM по сравнению с другими рекуррентными нейронными сетями является наличие «долгосрочной памяти». Это позволяет решать более сложные задачи, как, например: анализ текста, построение чатботов, генерация текстов, а также некоторые алгоритмические задачи. Была построена схема работы NTM, выделены главные недостатки модели, перечислены области применения.

Раздел 3. Дифференцируемый нейронный компьютер

В работе [2] была предложена нейросетевая модель дифференцируемого нейронного компьютера. Так же как и модель NTM, дифференцируемый нейронный компьютер обладает внешней памятью.

Дифференцируемый нейронный компьютер может быть обучен при помощи техник: обучения с учителем или обучения с подкреплением.

Вся система дифференцируема, и поэтому может быть обучена от начала и до конца при помощи модифицированных методов градиентного спуска и метода обратного распространения ошибки, позволяя сети научиться организовывать внешнюю память целенаправленным образом [2].

Нейронная машина Тьюринга может извлекать информацию из памяти в порядке индексов её слотов, но не в порядке, в котором была произведена запись в слоты памяти. Дифференцируемый нейронный компьютер обладает этой способностью.

3.1 LSTM-контроллер

Выпишем уравнения, описывающие работу LSTM-контроллера:

$$i_t^l = \sigma\left(W_i^l\left[\chi_t; h_{t-1}^l; h_t^{l-1}\right] + b_i^l\right); \tag{45}$$

$$f_t^l = \sigma\left(W_f^l\left[\chi_t; h_{t-1}^l; h_t^{l-1}\right] + b_f^l\right); \tag{46}$$

$$s_t^l = f_t^l s_{t-1}^l + i_t^l tanh\left(W_s^l\left[\chi_t; h_{t-1}^l; h_t^{l-1}\right] + b_s^l\right); \tag{47}$$

$$o_t^l = \sigma\left(W_o^l\left[\chi_t; h_{t-1}^l; h_t^{l-1}\right] + b_o^l\right); \tag{48}$$

$$h_t^l = o_t^l tanh\left(s_t^l\right); \tag{49}$$

где l – это индекс слоя, $\sigma(x) = 1/(1+exp(-x))$ – это сигмоидная функция активации, h_t^l, i_t^l, f_t^l, s_t^l и o_t^l – это векторы активации скрытого, входного вентиля, вентиля забывания, выходного вентиля слоя l в момент времени t.

$h_t^0 = 0$ для всех t.

В каждый момент времени контроллер производит выходной вектор v_t и вектор интерфейса $\xi_t \in \mathbb{R}^{(W\times R)+3W+5R+3}$, которые определяются так

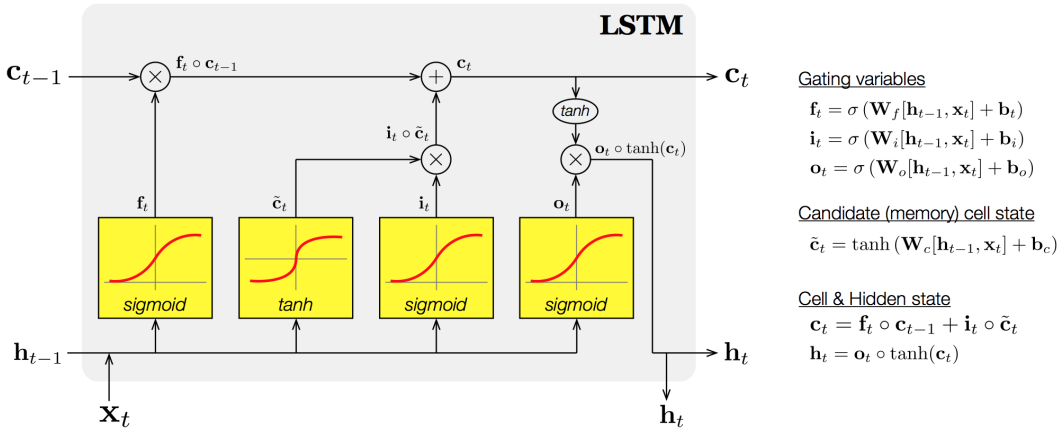


Рис. 4: LSTM-контроллер

$$v_t = W_y \left[h_t^1, \ldots, h_t^L\right],$$
$$\xi_t = W_\xi \left[h_t^1, \ldots, h_t^L\right].$$

Предположим, что контроллер является рекуррентным, его выходы – это функции от полной истории её входов $(\chi_1, \ldots, \chi_t)$ вплоть до текущего временного шага. Поэтому мы можем представить работу контроллера следующим образом

$$(v_t, \xi_t) = \mathcal{N}([\chi_1; \ldots; \chi_t]; \Theta),$$

где Θ – это набор обучаемых весов нейронной сети. В качестве контроллера также возможно использование нейронной сети прямого распространения. Выходной вектор y_t определяется как сумма вектора v_t с вектором, полученным путем применения весовой матрицы W_r размерности $RW \times Y$ к конкатенации текущих считывающих векторов.

$$y_t = v_t + W_r \left[r_t^1; \ldots; r_t^R\right].$$

3.2 Параметры интерфейса

Перед тем как быть использованным для параметризации взаимодействия с памятью, вектор интерфейса ξ_t подразделяется следующим образом:

$$\xi_t = \left[\mathbf{k}_t^{r,1}, \ldots, \mathbf{k}_t^{r,R}; \hat{\beta}_t^{r,1} \ldots; \hat{\beta}_t^{r,R}; \mathbf{k}_w^t; \hat{\beta}_t^w; \hat{\mathbf{e}}_t; \mathbf{v}_t; \hat{f}_t^1; \ldots; \hat{f}_t^R; \hat{g}_t^a, \hat{g}_t^w; \hat{\pi}_t^1; \ldots; \hat{\pi}_t^R\right].$$

Затем отдельные компоненты обрабатываются различными функциями активации, чтобы гарантировать, что они находятся в правильном домене. Сигмоидная функция используется для ограничения на $[0,1]$. Функция $oneplus$ используется для ограничения на $[1,\infty)$, где

$$oneplus\left(x\right)=1+log\left(1+e^{x}\right),$$

и $softmax$-функция используется для ограничения векторов на $\mathcal{S}_N$ – симплекс размерности $N-1$

$$S_N=\left\{\alpha\in\mathbb{R}^n:\alpha_i\in[0,1],\sum_{i=1}^{N}\alpha_i=1\right\}.$$

После обработки всех компонент вектора интерфейса, мы имеем следующий набор скаляров и векторов:

– R ключей чтения $\{k_t^{r,i}\in\mathbb{R};1\leq i\leq R\}$;
– R интенсивность чтения (прочность ключа чтения) $\{\beta_t^{r,i}=oneplus\left(\hat{\beta}_t^{r,i}\right)\in[1,\infty);1\leq i\leq R\}$;
– ключ записи $\mathbf{k}_t^w\in\mathbb{R}^W$;
– интенсивность записи (прочность ключа записи) $\beta_t^w=oneplus\left(\hat{\beta}_t^w\right)\in[1,\infty)$;
– стирающий вектор $\mathbf{e}_t=\sigma\left(\hat{\mathbf{e}}_t\right)\in[0,1]^W$;
– записывающий вектор $\mathbf{v}_t\in\mathbb{R}^W$;
– R освобождающих вентилей $\left\{f_t^i=\sigma\left(\hat{f}_t^i\right)\in[0,1];1\leq i\leq R\right\}$;
– вентиль распределения (вентиль выделения памяти) $g_t^a=\sigma\left(\hat{g}_t^a\right)\in[0,1]$;
– вентиль записи $g_t^w=\sigma\left(\hat{g}_t^W\right)\in[0,1]$;
– R режимов считывания $\left\{\pi_t^i=softmax\left(\hat{\pi}_t^i\right)\in\mathcal{S}_3;1\leq i\leq R\right\}$.

Интерпретация и использование этих терминов будут рассмотрены далее.

3.3 Чтение и запись в память

Выбор ячеек для чтения и записи зависит от весовых векторов, компоненты которых неотрицательные вещественные числа, сумма которых равна не более 1. Полный набор допустимых весов по N координатам – это неотрицательный гипероктант в пространстве $\mathbb{R}^N$ с единичным симплексом в качестве границы

$$\Delta_N = \left\{ \alpha \in \mathbb{R}^N : \alpha_i \in [0,1], \sum_{i=1}^{N} \alpha_i \leq 1 \right\}.$$

Для операции чтения используются R весовых векторов чтения $\left\{ \mathbf{w}_t^{r,i}, \ldots, \mathbf{w}_t^{r,R} \in \Delta_N \right\}$. Они используютя для вычисления векторов чтения $\left\{ \mathbf{r}_t^1, \ldots, \mathbf{r}_t^R \right\}$ следующим образом:

$$r_t^i = M_t^\top \mathbf{w}_t^{r,i}.$$

Векторы чтения присоединяются (при помощи операции конкатенации) ко входному вектору χ_t контроллера на следующем шаге, предоставляя ему доступ к содержимому памяти.

Операция записи регулируется весовым вектором $\mathbf{w}_t^w \in \Delta_N$, который используется совместно со стирающим вектором $\mathbf{e}_t \in [0,1]^W$ и записывающим вектором $\mathbf{v}_t \in \mathbb{R}^W$ (два последних генерируются непосредственно контроллером) для изменения памяти следующим образом:

$$M_t = M_{t-1} \circ \left(E - \mathbf{w}_t^w e_t^\top \right) + \mathbf{w}_t^w \mathbf{v}_t^\top,$$

где $\circ$ обозначает поэлементное произведение и E – это единичная матрица размерности $N \times W$.

3.4 Динамическое выделение памяти

Для того, чтобы у контроллера была возможность освобождать память по мере необходимости, был разработан дифференцируемый аналог схемы распределения памяти «список освобождения», в соответствии с которым список доступных мест памяти поддерживается путем добавления и удаления адресов из связанного списка. Обозначим через $\mathbf{u}_t \in [0,1]^N$ вектор использования памяти в момент времени t, и обозначим $\mathbf{u}_0 = \mathbf{0}$. Перед тем как выполнять операцию записи в память, контроллер генерирует набор освобождающих вентилей f_t^i (по одному на каждую считывающую головку), которые определяют, могут ли быть освобождены последние прочитанные слоты памяти. Вектор удержания памяти (**удерживающий вектор**) строится по следующей формуле

$$\psi_t = \prod_{i=1}^{R} \left(\mathbf{1} - f_t^i \mathbf{w}_{t-1}^{r,i} \right), \text{ где } R\text{– это количество считывающих головок.}$$

Дадим пояснение по этой формуле.

В момент времени $t-1$ дифференцируемый нейронный компьютер считал данные из памяти. Теперь у нас есть выбор: либо мы оставляем эти данные в памяти, либо если они нам уже не нужны, то стираем их из памяти. В момент времени t текущее состояние контроллера определяет значения освобождающих вентилей, которые и определяют в какой степени мы должны удалить только что прочитанные данные (освободить соответствующие слоты памяти). Вектор удержания – это, в некотором смысле, обратная к набору освобождающих вентилей величина.

Вентили освобождения f_t^i определяют насколько важно сейчас (на текущем шаге) то, что мы прочитали на предыдущем шаге. Итак, если то, что мы прочитали на предыдущем шаге сейчас важно, то f_t^i будет близко к нулю. Если же только что прочитанная информация для нас не важна больше, то f_t^i будет близко к единице.

Затем вектор использования можно определить как

$$\mathbf{u}_t = \left(\mathbf{u}_{t-1} + w_{t-1}^w - u_{t-1} \circ w_{t-1}^w\right) \circ \psi_t,$$

где $\circ$ обозначает поэлементное произведение, ψ_t – это вектор удержания (удерживающий вектор). Выражение в скобках – это правило сложения вероятностей совместных событий (а точнее, это нормировка, необходимая, чтобы значения вектора использования оставались в диапазоне $[0,1]$).

Вектор использования показывает на сколько важен каждый слот памяти. Значение $u[i]$ уменьшается после выполнения операции чтения (как видно из формул выше $u_t[i]$ уменьшается за счет вектора удержания ψ_t, последний в свою очередь зависит от весового вектора чтения на шаге $t-1$), и повышается после выполнения операции записи (за счет весового вектора записи).

Дадим интуитивное пояснение: слоты памяти используются, если они были сохранены освобождающими вентилями ($\psi_t \approx 1$), и были либо уже использованы, либо только что были записаны. Каждая запись в слот увеличивает значение его использования, максимум до 1, и коэффициент использования может быть уменьшен только до 0 с помощью освобождающих вентилей (входящих в вектор удержания памяти); поэтому элементы $\mathbf{u}_t$ ограничены в диапазоне $[0,1]$.

С одной стороны, значение $u_t[i]$ уменьшается, когда слот i был про-

читан на шаге времени $t-1$; это может быть хорошим признаком того, что содержимое слота i больше не требуется. С другой стороны, значение $u_t[i]$ увеличивается, когда запись в слот i была произведена на временном шаге $t-1$, что ясно указывает на то, что использование i-го слота необходимо усилить, поскольку значение, хранящееся в i-ом слоте, ранее еще не было доступно для чтения.

Принимая во внимание все это, вектор использования u_t может быть, в принципе, сформулирован следующим образом:

$$u_t[i] = \left(u_{t-1}[i] + w^w_{t-1}[i]\right)\psi_t[i].$$

Имеется ограничение в уравнении, только что сформулированном для вектора хранения $\psi_t \in [0,1]^N$: если слоты памяти освобождаются сразу после выполнения операции чтения из них, тогда может быть невозможно прочитать определенное значение, хранящееся в ячейке памяти более одного раза. Это было бы существенным ограничением, которое могло бы ограничить вычислительную мощность DNC и оставить многие приложения недоступными. Чтобы преодолеть это ограничение, контроллер генерирует скалярную величину освобождающий вентиль $f_t \in [0,1]$, чтобы гарантировать возможность сохранения местоположения памяти даже после операции чтения. Получаем окончательный вид уравнения для вектора хранения:

$$\psi_t[i] = \prod_{j=1}^{R}\left(1 - f^j_t w^{r,j}_{t-1}[i]\right).$$

Как видно, для каждой считывающей головки j существует свой собственный освобождающий вентиль f^j_t. Если $f^j_t \approx 0$ для всех считывающих головок, то $\psi_t[i] \approx 1$, что указывает на то, что i-ый слот памяти не может быть освобождён в момент времени t независимо от того, был ли i-ый слот прочитан на временном шаге $t-1$ или нет. Итоговое уравнение в векторной форме:

$$\psi_t = \prod_{j=1}^{R}\left(1 - f^j_t w^{r,j}_{t-1}\right),$$

где произведение векторов представляет собой поэлементное произведение, как и раньше.

Теперь рассмотрим как вычисляются весовые коэффициенты выделения памяти при $a_t \in \Delta_N$. Более высокие веса будут указаны в тех местах,

j	$u_t[j]$	$\gamma : \phi_t[\gamma] = j$	$a_t[j]$
1	1	4	0
2	0	1	1/1
3	0.8	3	1/3
4	0.4	2	1/2

Таблица 1: Пример вычисления весовых коэффициентов выделения памяти

которые ближе к началу списка слотов памяти, отсортированных по отношению к u_t в порядке возрастания. Поэтому первым шагом является получение списка освобождения $\phi_t \in N^N$, состоящего из индексов слотов памяти (в диапазоне $1, \ldots, N$) в порядке возрастания использования в момент времени t, определяемом u_t:

Например, учитывая вектор использования $u_t = [1; 0; 0,8; 0,4]$, получившийся список освобождения будет равен $\phi_t = [2; 4; 3; 1]$; наименее используемой ячейкой памяти будет $\phi_t[1]$ (слот памяти 2 в примере), а наиболее используемой ячейкой памяти будет $\phi_t[N]$ (слот памяти 1 в примере).

При заданных ϕ_y и u_t существует множество различных способов получения весовых коэффициентов выделения памяти. При этом необходимо учесть следующие ограничения:

– ячейка памяти, соответствующая k-ом элементу ϕ_y, получает вес выделения памяти, больший или равный, чем тот, который получен в ячейке памяти, соответствующей $(k+1)$-ому элементу ϕ_t
– вес выделения памяти в $a_t[i]$ равно 0 для всех ячеек памяти, где $u_t[i] = 1$.

Один из вариантов получения весовых коэффициентов выделения памяти следующий:

$$a_t[j] = \begin{cases} 0, \text{ если } u_t[j] = 1 \\ \frac{1}{\gamma}, \text{ иначе} \end{cases} \qquad j = 1, \ldots, N$$

γ — это значение, удовлетворяющее $\phi_t[\gamma] = j$. Если снова использовать
вектор $u_t = [1; 0; 0,8; 0,4]$, то полученное распределение распределения, полученное из уравнения (77), будет равно $a_t = [0; 1; 1/3; 1/2]$:

Однако вышеприведенное уравнение отбрасывает важную информацию о степени использования, содержащейся в u_t; кроме того, это не гарантирует,

j	$u_t[j]$	$\gamma : \phi_t[\gamma] = j$	$1 - u_t[j]$	$\prod_{i=1}^{\gamma-1} u_t[\phi_t[i]]$	$a_t[j]$
1	0.4	2	0.6	0.2	0.12
2	0.6	4	0.4	$0.2 \times 0.4 \times 0.5 = 0.04$	0.016
3	0.2	1	0.8	1	0.8
4	0.5	3	0.5	$0.2 \times 0.4 = 0.08$	0.04

Таблица 2: Пример вычисления весовых коэффициентов выделения памяти (улучшенный вариант)

что $a_t \in \Delta_N$. В реальности же DNC используют другой дифференцируемый подход:

$$a_t[j] = (1 - u_t[j]) \prod_{i=1}^{\gamma-1} u_t[\phi_t[i]],$$

в которой γ снова является значением, которое удовлетворяет $\phi_t[\gamma] = j$ вам. Для приведенного примера это уравнение даст в $a_t[1] = 0$, в $a_t[2] = 1$, в $a_t[3] = 0$, в $a_t[4] = 0$, $a_t = [0; 1; 0; 0]$.

Заметим, что для полностью используемых ячееек памяти ($u_t[\phi_t[j]] \approx 1$) весовой коэффициент выделения памяти приблизительно равен 0. Заметим также, что если существует хотя бы одно значение коэффициента использования 0, то первому слоту в списке ϕ_t будет присвоено 1, а оставшимся местам будет присвоен нуль. Нет доказательств того, что вектор выделения памяти всегда принадлежит Δ_N.

Еще один пример.

В случае вектора использования $u_t = [0,4; 0,6; 0,2; 0,5]$ итоговое взвешивание распределения составляет $a_t = [0,12; 0,016; 0,8; 0,04]$;

В оригинальной работе весовой вектор выделения памяти представлен в несколько иной, но эквивалентной форме, что позволяет избежать необходимости определять γ:

$$a_t[\phi_t[j]] = (1 - u_t[\phi_t[j]]) \prod_{i=1}^{j-1} u_t[\phi_t[i]].$$

Сортировка не является обычной операцией в нейронных сетях. Это может даже привести к некоторым проблемам при вычислении градиента. Но, по мнению авторов статьи, эти вопросы экспериментально не заметны:

«Операция сортировки вызывает разрывы в точках, в которых изменяется порядок сортировки. Мы игнорируем эти разрывы при расчете градиента, поскольку они, похоже, не имеют отношения к обучению» [2].

Контроллер производит R скалярных величин $f_t^i, i = 1, \ldots, R$

$$\phi_t = SortIndicesAscending\left(\mathbf{u}_t\right).$$

После определения $\mathbf{u}_t$ список освобождения $\phi_t \in \mathbb{Z}^N$ определяется путём сортировки индексов слотов памяти в порядке возрастания **коэффициентов использования**; $\phi_t\left[1\right]$ является индексом наименее используемого слота памяти. Вес распределения (attention) $\mathbf{a}_t \in \Delta_N$, которое используется для предоставления новых слотов памяти для записи:

$$\mathbf{a}_t\left[\phi_t\left[j\right]\right] = \left(1 - \mathbf{u}_t\left[\phi_t\left[j\right]\right]\right) \prod_{i=1}^{j-1} u_t\left[\phi_t\left[i\right]\right]. \tag{50}$$

Если все **коэффициенты использования** равны 1, тогда $\mathbf{a}_t = \mathbf{0}$ и контроллер больше не может выделять память без предварительного освобождения используемых слотов памяти.

Как уже указывалось, адресация на основе контента сочетается с динамическим распределением памяти, чтобы получить весовой коэффициент w_t^w записи.

Задача распределения динамической памяти состоит в том, чтобы заставить DNC вычислять новый вид взвешивания на каждом временном шаге, весовые коэффициент выделения памяти, которые являются компонентами вектора $a_t \in \Delta_N$, который указывает, в какой степени каждая ячейка памяти может использоваться для записи новой информации (то есть не защищена от записи). Тот факт, что $a_t \in \Delta_N$ подразумевает, что бинарная аналогия не может быть соблюдена здесь. Распределение, соответствующее $N = 4$ одинаково распределяемых мест памяти, будет, например, равным $[0,1;0,1;0,1;0,1]$, а не $[1;1;1;1]$; тот факт, что вторая ячейка памяти полностью используется для записи новой информации, но никакой другой слот не может быть зарезервирован, будет представлен следующим образом:

$a_t = [0;1;0;0]$; вес выделения новой памяти при $a_t = [0,4;0,2;0;0]$ представляет ситуацию, в которой первой слот будет больше выделяться для но-

вой информации, чем второй. Если $a_t = 0$, то DNC израсходовала свободные слоты памяти, и поэтому уже нет доступа к слотам памяти для записи посредством распределения динамической памяти в момент времени t; однако важно отметить, что все еще можно произвести запись в слоты, доступные с помощью адресации по контенту.

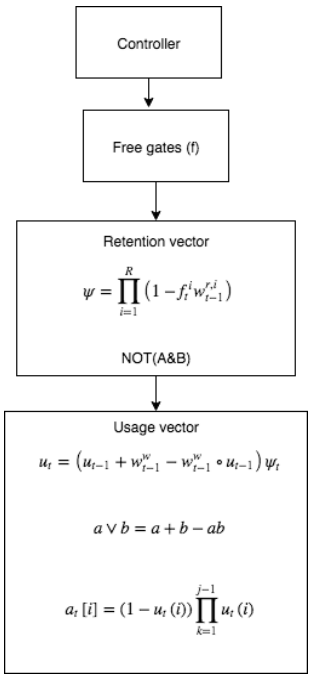


Рис. 5: Динамическое выделение памяти

3.5 Взвешивание (весовые коэффициенты) операции записи

Контроллер может производить запись в недавно выделенные слоты памяти или в слоты, ассоциированные с некоторым контентом, или он может вообще не производить запись. Во-первых, весовые коэффициенты операции записи по контенту $c_t^W = \mathcal{C}\left(M_{t-1}, \mathbf{k}_t^W, \beta_t^W\right)$

c_t^w интерполируется с помощью весовых коэффициентов выделения памяти a_t, определенного с помощью уравнения (50) для определения веса операции записи $w_t^W \in \Delta_N$:

$$w_t^w = g_t^w \left[g_t^a \mathbf{a}_t + (1 - g_t^a c_t^w)\right], \tag{51}$$

где $g_t^a \in [0,1]$ – это вентиль выделения памяти, управляющий интерполяцией $g_t^w \in [0,1]$ – это вентиль записи. Если вентиль записи равен 0, тогда

ничего не будет записываться (независимо от других параметров записи); поэтому его можно использовать для защиты памяти от ненужных изменений.

Мы можем игнорировать механизм выделения памяти, если используем механизм адресации по контенту, т. к. адресация по контенту обычно фокусируется на конкретных слотах в памяти, содержащих необходимую информацию, и используется в основном для перезаписи этой информации. Механизм выделения памяти используется для хранения новой информации.

3.6 Временна́я связь памяти (механизм временно́й связи, механизм временны́х ссылок)

Система распределения памяти, определенная выше, не хранит никакой информации о порядке, в котором записана информация. Однако существует много ситуаций, в которых сохранение этой информации полезно: например, когда последовательность инструкций должна быть записана и восстановлена в некотором порядке. Мы будем использовать временну́ю матрицу ссылок $L_t \in [0,1]^{N \times N}$ для отслеживания последовательно измененных слотов памяти (рис. 6d).

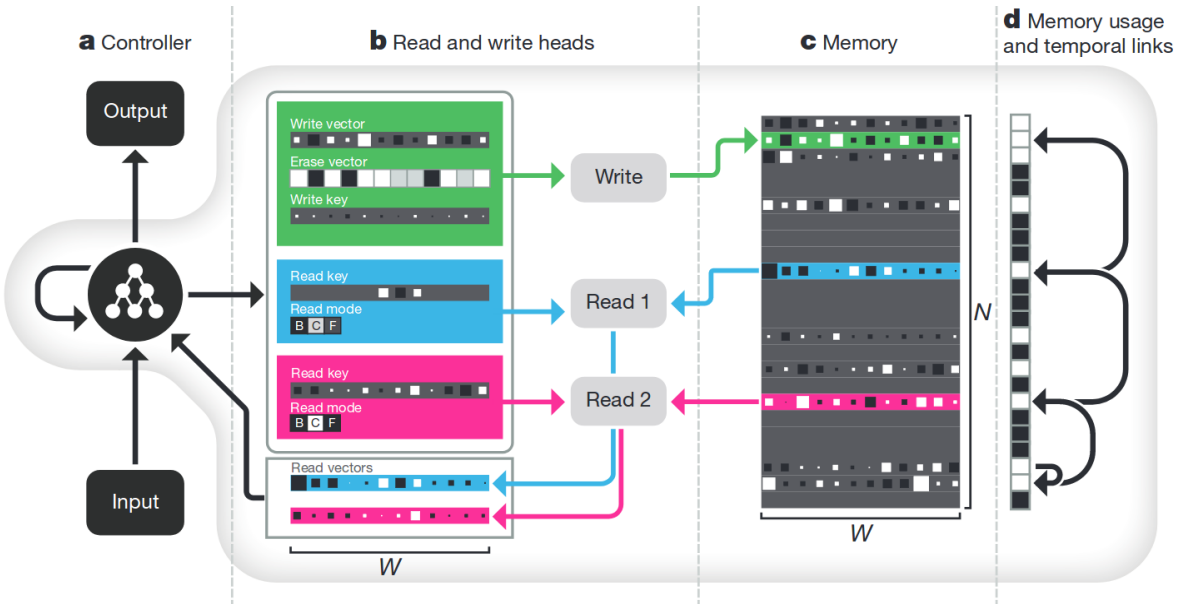


Рис. 6: Общая схема дифференцируемого нейронного компьютера

$L_t[i,j]$ представляет собой степень, в которой ячейка памяти i была записанной в момент времени t после ячейки j в то время как запись в ячейку j была произведена в момент времени $t-1$. Каждая строка и столбец L_t определяет вес между ячейками:

$L_t[i,\cdot] \in \Delta_N$ и $L_t[\cdot,j] \in \Delta_N$ для всех i, j и t. Для определения L_t нам потребуется взвешенный приоритет (весовой коэффициент приоритета) $\mathbf{p}_t \in \Delta_N$, где элемент $\mathbf{p}_t[i]$ представляет степень в которой i-ый слот памяти был записан последним. $\mathbf{p}_t$ определяется рекуррентным соотношением

$$\mathbf{p}_0 = \mathbf{0},$$

$$\mathbf{p}_t = \left(1 - \sum_i \mathbf{w}_t^w[i]\right) p_{t-1} + \mathbf{w}_t^w,$$

где $\mathbf{w}_t^w$ – это взвешивание (вес) записи, определенное в уравнении (51).

Каждый раз, когда слот памяти изменяется, матрица ссылок обновляется, чтобы удалить старые ссылки в эту ячейку и из этой ячейки, а затем создать новые ссылки. Для реализации этой логики мы используем следующее рекуррентное соотношение:

$$L_0[i,j] = 0, \forall i, j$$

$$L_t[i,i] = 0, \forall i$$

$$L_t[i,j] = (1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j])\, L_{t-1}[i,j] + \mathbf{w}_t^w[i]\, p_{t-1}[j].$$

Рассмотрим выражение $(1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j])\, L_{t-1}[i,j]$ более подробно:
Рассмотрим строку i и столбец j в матрице временны́х ссылок.

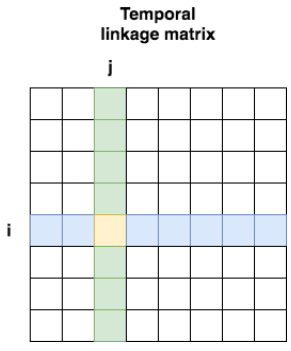


Рис. 7: Матрица временны́х ссылок

Если мы переписываем информацию в матрице внешней памяти в позициях i и j, то все элементы в строке i и столбце j матрицы временны́х

ссылок изменятся. Следовательно, мы должны забыть предыдущее состояние $L_{t-1}[i,j]$, и степень забывания зависит от интенсивности новых весовых коэффициентов $\mathbf{w}_t^w[i]$ и $\mathbf{w}_t^w[j]$. А затем мы должны обновить информацию в строке i и столбце j матрицы временны́х ссылок за счёт слагаемого $\mathbf{w}_t^w[i]\,p_{t-1}[j]$ (т. к. по определению $L_t[i,j]$ – это степень в которой слот i был записан после слота j, то мы должны прибавить к изменённому $L_{t-1}[i,j]$ весовой коэффициент $\mathbf{w}_t^w[i]$, умноженный на степень того, что j-ый слот был записан последним на момент времени $t-1$).

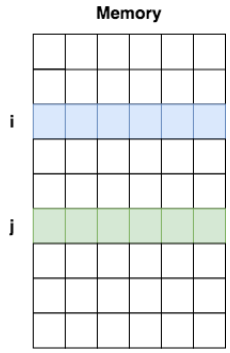


Рис. 8: Матрица внешней памяти

Ссылки-петли исключены (элементы, лежащие на диагонали матрицы ссылок всегда равны нулю) потому что неясно, как следует переход от ячейки к самой себе. Строки и столбцы L_t представляют веса временны́х ссылок, входящих и выходящих из определенных слотов памяти, соответственно. Пусть даны L_t, обратное временное взвешивание (вес) $\mathbf{b}_t^i \in \Delta_N$ и прямое временное взвешивание (вес) $\mathbf{f}_t^i \in \Delta_N$ для считывающей головки i определяются так:

$$\mathbf{f}_t^i = L_t \mathbf{w}_{t-1}^{r,i},$$

$$\mathbf{b}_t^i = L_t^\top \mathbf{w}_{t-1}^{r,i},$$

где $\mathbf{w}_{t-1}^{r,i}$ – это i-ый вес считывания с предыдущего временного шага.

Начнём с описания того, как все будет работать в бинарном представлении, когда запись производится на каждом временном шаге только в один слот, также предположим (ошибочную) гипотезу о том, что операция записи всегда выполняется на каждом временном шаге.

С этими упрощениями L_t будет матрицей из нулей и единиц, в которых если i является последним записанным слотом (в момент времени t), а j является вторым по последнему записанному слоту (в момент времени $t-1$), то $L_t[i,j]=1$; если запись в i-ый слот не производилась на временном шаге t, и в j-ый слот не производилась запись на временном шаге $t-1$, то $L_t[i,j]$ остается неизменным относительно его предыдущего значения на временном шаге $t-1$; в других случаях $L_t[i,j]=0$, что отражает тот факт, что запись в i-ый слот не была произведена после записи в j-ый слот (напомним, что мы предполагаем, что операция записи всегда выполняется на каждом шаге) [22]:

$$L_t[i,j]=\begin{cases}1, \text{если } w_t^w[i]=1 \text{ и } w_{t-1}^w[j]=1\\ 0, \text{если } w_t^w[i]\neq w_{t-1}^w[j]\\ L_{t-1}[i,j], \text{ иначе}\end{cases} \tag{52}$$

В остальных случаях $L_0[i,j]=0$ для всех i и j.

Помимо упрощения, присущего бинарному представлению, предыдущее уравнение игнорирует тот факт, что между одной записью и следующей может быть произвольное количество временных шагов. Чтобы преодолеть эти ограничения, нам нужно более точное средство для записи степени, в которой была произведена запись в слот памяти.

Эта цель достигается путем введения нового взвешивания $p_t \in \Delta_N$, которое называется взвешенном приоритетом. Элемент $p_t[i]$ обозначает степень, в которой последняя запись производилась в i-ый слот, и вычисляется рекурсивно.

Если величина $p_t[i]$ большая, то это может указывать:

– на то, что запись в i-ый слот производилась на временном шаге t с большей степенью (то есть другие слоты вообще не были записаны на временном шаге t или или запись в них производилась в гораздо меньшей степени)
– это также может указывать на то, что запись в i-ый слот была произведена с большей степенью при $t^{'} \leq t$, но что значительная запись не была выполнена в памяти с момента времени $t^{'}$;
– это может также указывать на то, что вес (внимание) записи на i-ом слоте было частично распределено на разных недавних временных шагах в прошлом, а общая сумма накопленного веса (внимания) больше, чем вес

(степень внимания) другим слотам.

Предварительная бинарная формулировка весового приоритета следующая:

$$p_t = \begin{cases} w_t^w, \text{ если } \sum\limits_{i=1}^{N} w_t^w[i] = 1 \\ p_{t-1}, \text{иначе} \end{cases} \tag{53}$$

Заметим, что вектор p_t обновляется только тогда, когда операция записи выполнялась на шаге времени t, и поэтому он мог оставаться нетронутым для длительных периодов в режиме «только для чтения».

Как известно, даже один шаг работы DNC в режиме «только для чтения», является сложным процессом. Чтобы вернуться к реальной модели памяти DNC нужно принять факт, что весовой коэффициент записи крайне редко (почти никогда) будет равен нулю, поэтому введём непрерывную формулировку взвешенного приоритета следующим образом:

$$\mathbf{p}_t = \left(1 - \sum_i \mathbf{w}_t^w[i]\right) p_{t-1} + \mathbf{w}_t^w,$$

где $\mathbf{w}_t^w$ – это взвешивание (вес) записи, определенное в уравнении (51). $\left(1 - \sum\limits_i \mathbf{w}_t^w[i]\right)$ – это вероятность (степень) того, что мы ничего не записываем.

Итак, если мы ничего не записываем, то мы оставляем предыдущий $\mathbf{p}_{t-1}$ (используем его).

Обратите внимание, что предыдущее уравнение вырождалось бы до (53) в предельных случаях полной записи или «только для чтения». В первом случае суммирование всех компонент w_t^w будет иметь наибольшее значение (т. е. равно 1), а p_t будет уменьшено до w_t^w; во втором случае суммирование всех компонент w_t^w будет равно 0, w_t^w также будет равно 0, а p_t будет тогда копией p_{t-1}. Обратите внимание, что после гипотетической полной записи p_t будет перезагружен, а прошлая история (закодированная в p_{t-1}) полностью забыта. Упорядоченный приоритет инициализируется $p_0 = 0$.

Принимая во внимание все это, мы, наконец, готовы изложить окончательную дифференцируемую формулировку $L_t[i,j]$, в которой бинарные ограничения (52) уже не учитываются. Уравнение должно отражать тот факт, что если в слот памяти i была произведена значительная запись на временном шаге t (т. е. $w_t^w[i]$ велико), то в $L_t[i,j]$ должно быть значение, которое

в основном пропорционально степени, в которой слот памяти j был последним слотом, записанным до этого (эта информация представлена в $p_{t-1}[j]$); одним из способов для реализации этого является произведение $w_t^w[i]\,p_{t-1}[j]$. Если $w_t^w[i]$ мало (или, что то же самое, $1-w_t^w[i]$ велико), то $L_t[i,j]$ будет в основном близко к значению $L_{t-1}[i,j]$; тем не менее, существует исключение из последнего правила: если в слот j была произведена значительная запись на шаге времени t (т. е. $w_t^w[j]$ является высоким), тогда эта ситуация должна инициировать не бинарный сброс $L_t[i,j]$ до низкого значения, поскольку ячейка памяти, которая будет записана после j, не будет известна до временного шага $t+1$. Получаем дифференцируемое уравнение, которое объединяет все эти аспекты:

$$L_t[i,j] = (1-\mathbf{w}_t^w[i]-\mathbf{w}_t^w[j])\,L_{t-1}[i,j]+\mathbf{w}_t^w[i]\,p_{t-1}[j].$$

Матрица временных связей определяет порядок, в котором была произведена запись в слоты памяти.

Контроллер может использовать L_t чтобы извлекать записанные данные ранее (до) b_t^i или после f_t^i последней позиции чтения $\mathbf{w}_{t-1}^{r,i}$, позволяя ему двигаться вперед или назад во времени (оперировать данными вперед или назад во времени).

Рассмотрим пример.

Пусть $L_t = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$.

В этом примере единица, обозначенная красным цветом L_t [4,2] представляет предложение «Запись в слот 4 была произведена после записи в слот 2» . Обратите внимание, что строка i хранит обратную информацию (а именно, что было написано до записи в слот i, например, четвертая строка в матрице позволяет нам знать, что слот 2 было записано до местоположения 4), тогда как столбец j содержит прямую информацию (а именно, что было написано после записи в слот j, например, второй столбец в предыдущей матрице позволяет нам знать, что головка записи переместилась в слот 4 после записи в слот 2). Тот факт, что второй столбец имеет ненулевой элемент, но вторая строка состоит из нулей, указывает, что первый слот, в который производилась запись имеет индекс (адрес) 2; тот факт, что и третий ряд, и третий столбец равны нулю, показывает, что в слот 3 еще не производилась

запись.

Принимая это во внимание и учитывая общий вес w_t, можно легко вывести формулы, описывающие, как мы можем двигаться назад или вперед во времени, чтобы сосредотачивать внимание на те слоты памяти, запись в которые производилась до или после тех, которые представлены в векторе w_t; результирующие слоты будут соответственно представлены обратным взвешиванием b_t и прямым взвешиванием f_t, которые вычисляются следующим образом:

$$\mathbf{f}_t^i = \hat{L}_t \mathbf{w}_{t-1}^{r,i},$$

$$\mathbf{b}_t^i = \hat{L}_t^\top \mathbf{w}_{t-1}^{r,i}.$$

Эта матрица действует как летописец, сохраняя историю памяти, записывая в математическом эквиваленте хроники следующим образом: «В начале была произведена запись в слот памяти 2. Затем в слот 4 была произведена запись после записи в слот 2. Затем в слот 1 была произведена запись после записи в слот 4».

Например, учитывая предыдущую матрицу временной связи L_t и бинарный случай взвешивания $w_t = [0; 0; 0; 1]$, обратное взвешивание будет:

$$\mathbf{b}_t = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}^\top w_t = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix},$$

что указывает, что слот, записанный до слота с индексом 4, является слотом с индексом 2. Прямое взвешивание будет аналогичным образом получено как:

$$f_t = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} w_t = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

что слот, записанный после слота с индексом 4, является слотом с индексом 1. Заметим еще раз, что, хотя предыдущий пример относится к упрощенному бинарному случаю, DNC имеют дело с матрицами временной связи и весами, которые в общем случае являются вещественными числами. В

результате этого элемент $L_t[i,j]$ во временно́й матрице ссылок фактически указывает, в какой степени ячейка памяти i была записана после ячейки j; прямые и обратные веса также сосредоточивают свое внимание на каждой ячейке не бинарным образом (компоненты этих векторов являются также вещественными числами).

В частном случае попытки получить обратное взвешивание для второго слота памяти ($w_t = [0;1;0;0]$), которое является первым местом, записанным в соответствии с L_t, мы получим:

$$\mathbf{b}_t = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}^{\top} w_t = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Как указывалось ранее, временная связь слотов памяти является режимом адресации, который предназначен для чтения. Этот режим основан на информации об операциях записи, которая предоставляется матрицей временных связей L_t.

Основная цель данных построений заключается в том, чтобы позволить слотам памяти считываться в том же порядке, в котором производилась в них запись (или в обратном порядке, в зависимости от задачи).

$$b_t^i = L_t^{\top} w_{t-1}^{r,i},$$

$$f_t^i = L_t w_{t-1}^{r,i},$$

$$b_t^i \in \Delta_N,$$

$$f_t^i \in \Delta_N.$$

Обратите внимание, что так как $w_t^w \in \Delta_N$, то невозможно, чтобы вычитания в скобках приводили к отрицательному значению. Для завершения спецификации $L_t[i,j]$ требуются два следующих уравнения:

$$L_0[i,j] = 0,$$

$$L_t[i,i] = 0.$$

Смещение фокуса к следующему моменту (состоянию памяти): Lw

Смещение фокуса к предыдущему моменту (состоянию памяти): $L^\top w$

Это завершает описание того, что весовые значения чтения вычисляются как комбинация контент-адресации и временной связи (временны́х ссылок) в памяти. Рассмотрим более подробно вопрос о том как эффективно хранить и вычислять матрицу временны́х ссылок L_t.

3.7 Матрица разреженных ссылок

Матрица ссылок имеет размер $N \times N$ и, поэтому требует $\mathcal{O}\left(N^2\right)$ ресурсов памяти и времени для её точного вычисления. Эта стоимость быстро становится запредельной по мере увеличения количества слотов во внешней памяти.

К счастью, матрица ссылок, как правило, очень разрежена и может быть аппроксимирована за $\mathcal{O}(NlogN)$ времени и $\mathcal{O}(N)$ памяти без заметных потерь в производительности. Для некоторого фиксированного K мы сначала вычисляем разреженные векторы $\hat{\mathbf{w}}_t^w$ и $\hat{\mathbf{p}}_{t-1}$ путём сортировки $\mathbf{w}_t^w$ и $\mathbf{p}_{t-1}$, устанавливая все, кроме K максимальных значений, равными 0, и делим оставшиеся K на их сумму, чтобы обеспечить их суммирование до 1.

Этот шаг имеет вычислительную сложность $\mathcal{O}(NlogN + K)$ для учёта стоимости сортировки $\mathcal{O}(N)$ памяти. Затем вы вычисляем разреженное внешнее произведение $\hat{\mathbf{w}}_t^w \hat{\mathbf{p}}_t^\top$, которое требует $\mathcal{O}\left(K^2\right)$ памяти и времени. Предполагая, что разреженная матрица ссылок $\hat{L}_{t-1}$ с предыдущего временного шага имеет не более NK ненулевых элементов, то $\hat{L}_t$ можно обновить за $\mathcal{O}(NK)$ используя

$$\hat{L}_t[i,j] = \left(1 - \hat{\mathbf{w}}_t^w[i] - \hat{\mathbf{w}}_t^w[j]\right)\hat{L}_{t-1}[i,j] + \hat{\mathbf{w}}_t^w[i]\,\hat{\mathbf{p}}_{t-1}[j],$$

а затем присваивая нулю все элементы $\hat{L}_t$, которые меньше $1/K$.

Поскольку каждая строка и столбец $\hat{L}_t$ в сумме даёт не более, чем 1, то эта операция гарантирует, что $\hat{L}_t$ имеет не более K ненулевых элементов в строке и в столбце и, следовательно, не более NK ненулевых элементов. Наконец, прямое и обратное временное взвешивание может быть рассчитано с использованием $\mathcal{O}(NK)$ времени и $\mathcal{O}(N)$ памяти следующим образом:

$$\mathbf{f}_t^i = \hat{L}_t \mathbf{w}_{t-1}^{r,i},$$

$$\mathbf{b}_t^i = \hat{L}_t^\top \mathbf{w}_{t-1}^{r,i}.$$

Поскольку K является константой, не зависящей от N (на практике $K = 8$ оказывается достаточной, независимо от размера памяти), полное разреженное обновление $\mathcal{O}(NlogN)$ при вычислении $\mathcal{O}(N)$ памяти.

3.8 Взвешивание (весовые коэффициенты) операции чтения

Каждая считывающая головка i вычисляет **взвешивание по контенту** $\mathbf{c}_t^{r,i} \in \Delta_N$ используя ключ чтения $\mathbf{k}_t^{r,i} \in \mathbb{R}^w$:

$$\mathbf{c}_t^{r,i} = \mathcal{C}\left(M_t, \mathbf{k}_t^{r,i}, \beta_t^{r,i}\right).$$

Каждая считывающая головка также принимает вектор режима чтения $\pi_t^i \in \mathcal{S}_3$ как параметр, который интерполирует среди обратного временного взвешивания $\mathbf{b}_t^i$, прямого временного взвешивания $\mathbf{f}_t^i$ и взвешивания чтения по контенту (содержимому) $\mathbf{c}_t^{r,i}$, тем самым определяя весовое соотношение $\mathbf{w}_t^{r,i} \in \mathcal{S}_3$:

$$\mathbf{w}_t^{r,i} = \pi_t^i[1]\,\mathbf{b}_t^i + \pi_t^i[2]\,\mathbf{c}_t^{r,i} + \pi_t^i[3]\,\mathbf{f}_t^i,$$

$$\pi_t^i = softmax\left(\hat{\pi}_t^i\right),$$

$$\pi_t^i[j] = \frac{exp\left(\hat{\pi}_t^i[j]\right)}{\sum\limits_{k=1}^{3} exp\left(\hat{\pi}_t^i[k]\right)}, i = 1, \ldots, R, j = 1, 2, 3.$$

Если $\pi_t^i[2]$ доминирует в режиме чтения, тогда взвешивание (weighting) возвращается к поиску содержимого с помощью ключа $\mathbf{k}_t^{r,i}$. Если $\pi_t^i[3]$ доминирует, тогда считывающая головка выполняет проходит по ячейкам памяти в том порядке, в котором они были записаны, игнорируя ключ чтения. Если $\pi_t^i[1]$ доминирует, тогда считывающая головка считывает содержимое ячеек в обратном порядке (по отношению к порядку, в котором они были записаны), также игнорируя ключ чтения.

Для построения весового вектора чтения используется два механизма адресации: адресация (взвешивание) по контенту (по содержимому) и временное связывание.

Коэффициенты π_t^i определяют в какой степени использовать механизмы адресации: по контенту, прямое временное взвешивание и обратное временное взвешивание.

Прямое временное взвешивание означает, что мы должны считывать слоты памяти в порядке, в котором они были записаны.

Обратное временное взвешивание означает, что мы должны считывать слоты памяти в порядке, обратном тому, в котором они были записаны.

Для любого весового вектора $\mathbf{w}$ операция $L\mathbf{w}$ плавно перемещает фокус вперед в места, записанные после тех, что указаны в $\mathbf{w}$, тогда как $L^T\mathbf{w}$ сдвигает фокус назад.

Это дает DNC возможность восстанавливать последовательности информации в том порядке, в котором они были записаны, даже если последовательные записи не возникали в смежных временных шагах. [2].

Механизм распределения памяти не зависит от размера и содержимого памяти, что означает, что DNC можно обучить решению задачи с использованием одного размера памяти, а затем обновлять до большей памяти без переподготовки. В принципе, это позволит использовать неограниченную внешнюю память, автоматически увеличивая количество мест каждый раз, когда минимальное использование любого места проходит определенный порог. И удалять долго неиспользуемые слоты памяти.

Поиск контента позволяет создавать ассоциативные структуры данных. Временные ссылки обеспечивают последовательное извлечение входных последовательностей. Механизм выделения памяти обеспечивает запись в неиспользуемые места.

Раздел 4. Описание вычислительных экспериментов

Модели LSTM, NTM и DNC реализованы на языке Python при помощи библиотеки TensorFlow.

Во всех приведённых далее задачах в процессе обучения при обратном распространении ошибки градиенты весов контроллера LSTM в моделях NTM и DNC были поэлементно обрезаны до диапазона [-10, 10].

4.1 Задача копирования битовых векторов

4.1.1 Сравнение NTM и LSTM

Рассмотрим задачу копирования последовательностей битовых векторов. Эта задача позволяет проверить могут ли NTM и DNC хранить и воспроизводить более длинные последовательности данных, чем LSTM. Хранение и доступ к информации в течение длительных периодов времени всегда были проблематичными для сетей RNN и других динамических архитектур. Обратите внимание, что во время получения целевой последовательности от NTM и DNC входные данные не были представлены в сеть, чтобы гарантировать, что они восстанавливает всю последовательность из своей памяти, а не из обучающей/тестовой выборки. На вход контроллера будем подавать случайные битовые векторы фиксированной длины. Каждая последовательность битовых векторов разделена специальной меткой (дополнительным битом, равным единице).

Последовательности битовых векторов (каждый длиной 8), поступающих на вход, имеют случайную длину в диапазоне от 1 до 20. Целевая последовательность будет совпадать с входной последовательностью (без учёта разделительной метки и с задержкой во времени). Сначала полностью поступает на вход последовательность векторов, а затем требуется полностью восстановить эту последовательность.

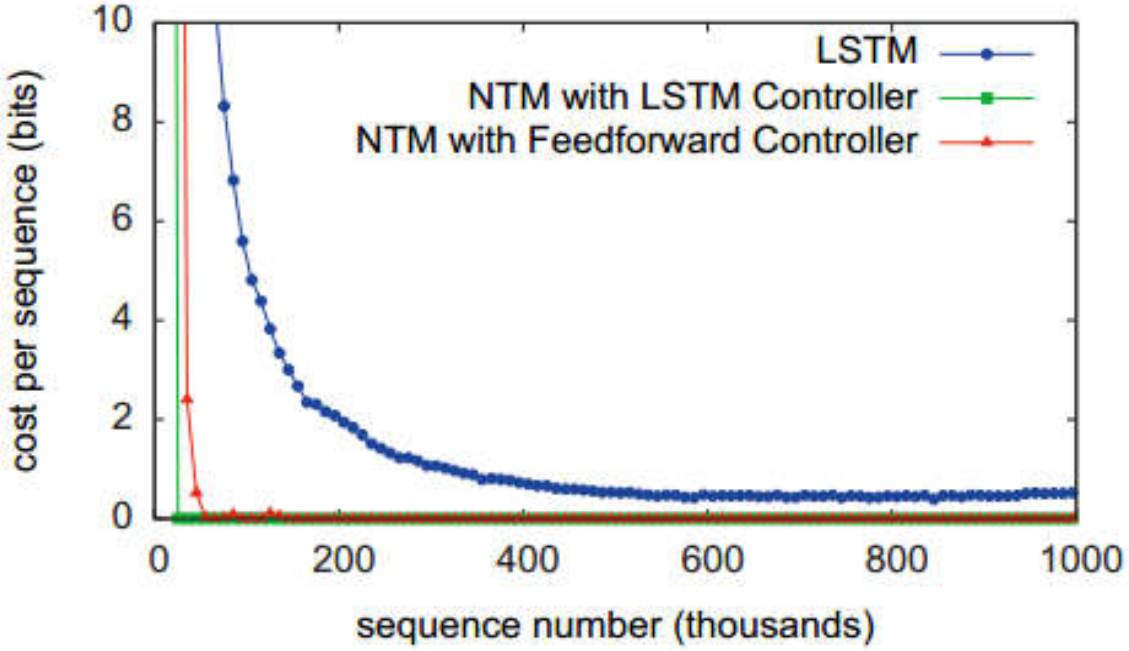


Рис. 9: Изменение функционала качества в процессе обучения

На графике (9) видно, что NTM (с FFNN или LSTM контроллером) обучается намного быстрее, чем LSTM и сходится к более хорошему результату.

Также в эксперименте была изучена способность NTM к обобщению (т.е. решению задачи для более длинных последовательностей, не входящих в обучающую выборку). На рис. 10 показано поведение NTM и LSTM на тестовых данных разной длины. Четыре пары изображений в верхнем ряду соответствуют выходным данным сети и соответствующими целевым последовательностям длиной 10, 20, 30 и 50 соответственно. Изображения в нижнем ряду соответствуют последовательности длиной 120. Сеть обучалась только на последовательностях длиной до 20. Первые четыре последовательности воспроизводятся NTM с высокой достоверностью и с очень небольшим количеством ошибок. Самый длинный имеет несколько локальных ошибок и одну глобальную ошибку: в точке, обозначенной красной стрелкой внизу, дублируется один вектор, сдвигая все последующие векторы на один шаг назад. Несмотря на субъективную близость к правильной копии, это приводит к большим потерям.

Гиперпараметры NTM для задачи Copy Task:

– количество считывающих головок: 1,
– количество слотов памяти: 128,
– размер слота памяти: 20,
– размер контроллера: 1 LSTM-слой размером 100 нейронов.

LSTM допускает серьезные ошибки уже при длине последовательности равной 30.

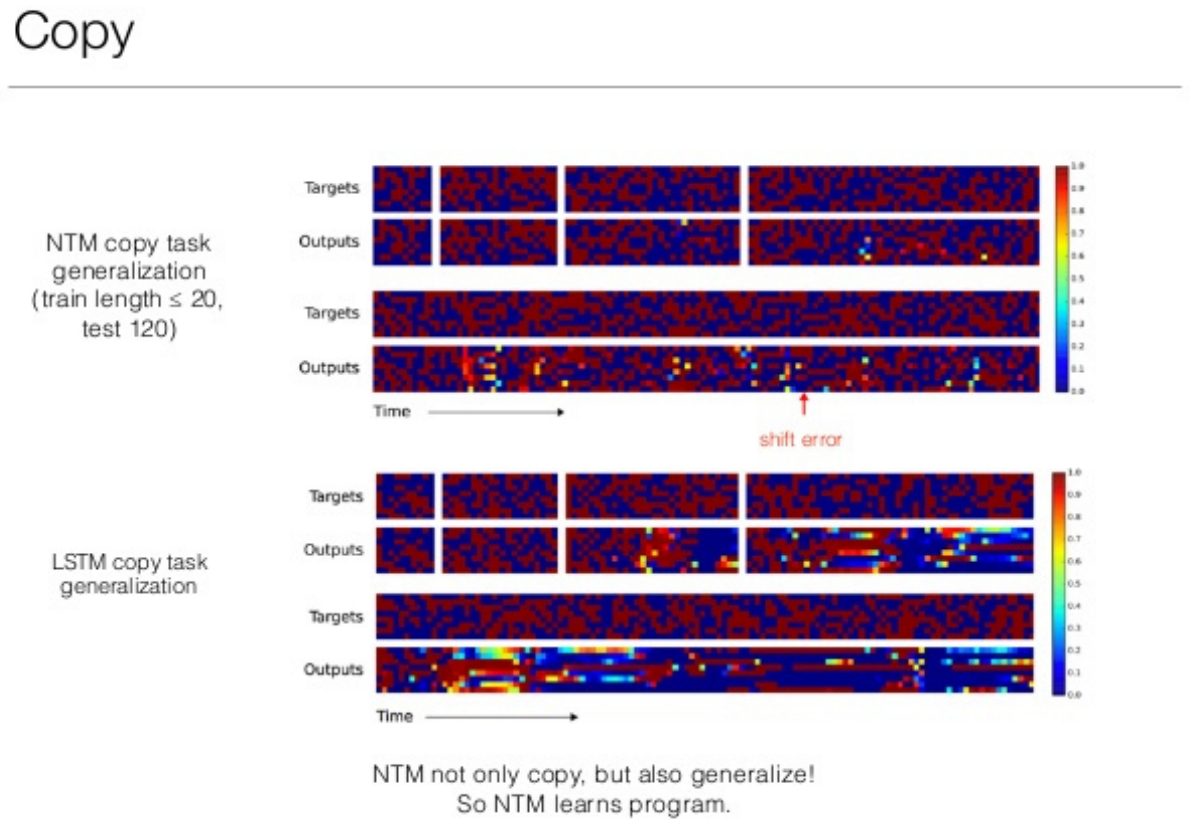


Рис. 10: Результат тестирования в задаче копирования последовательностей битовых векторов

Как и NTM, LSTM учится воспроизводить последовательности длиной до 20 практически идеально. Однако LSTM явно не может строить обобщения на более длинные последовательности. Также обратите внимание, что длина точного префикса уменьшается с увеличением длины последовательности, что указывает на существенные проблемы с сохранением информации в сети в течение длительных периодов.

4.1.2 Сравнение DNC и LSTM

На рис. 11 изображен график кривых функции потерь в процессе обучения моделей LSTM и DNC. На обучение DNC потребовалось в среднем 10000 итераций. В процессе обучения LSTM критерий останова не был выполнен (из-за постоянных скачков функции потерь). Остановка процесса обучения LSTM была произведена вручную.

Гиперпараметры DNC для задачи Copy Task:

– количество считывающих головок: 2,
– размер пакета: 4,
– количество слотов памяти: 20,
– размер слота памяти: 10,
– размер контроллера: 1 LSTM-слой размером 128 нейронов.

Гиперпараметры LSTM для задачи Copy Task:

– размер пакета: 1,
– размер контроллера: 3 LSTM-слоя размером 256 нейронов каждый.

Рис. 11: Графики процесса обучения моделей LSTM и DNC

На рис. 12 изображены графики функционала качества в процессе тестирования LSTM и DNC.

Тестирование проводилось на последовательностях битовых векторов. Длины последовательностей варьировались от 1 до 120 с шагом 10. По каждой длине последовательности генерировалось 1000 объектов.

Мы видим, что при длине последовательностей от 1 до 20 обе модели дают хороший результат. Но при попытке проверить обобщающую способность

моделей сразу видно, что DNC превосходит LSTM по способности запоминать длинные последовательности данных. Разница в качестве более чем в 2 раза.

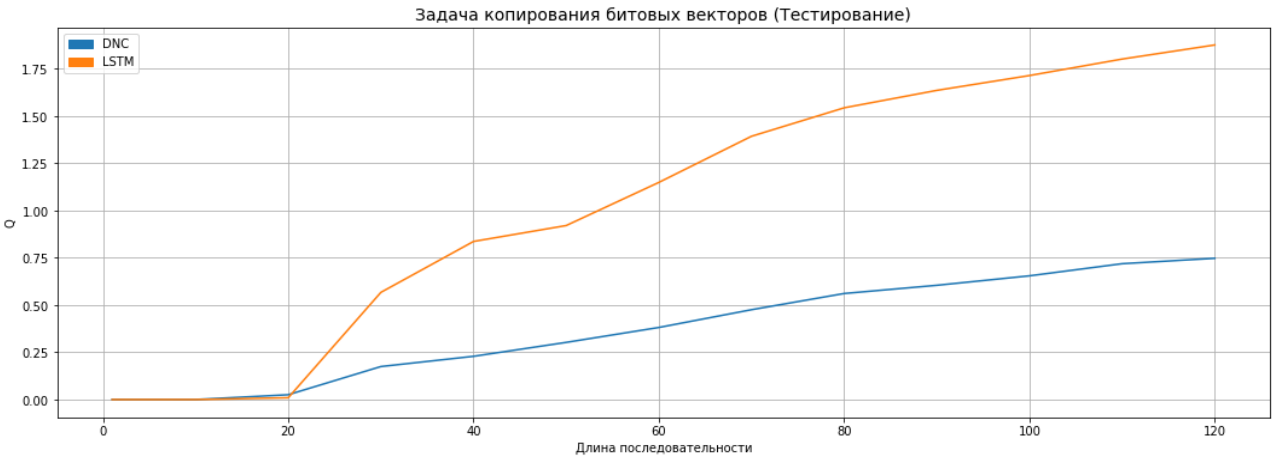


Рис. 12: Графики обобщающей способности моделей LSTM и DNC в задаче копирования битовых векторов

4.2 Описание bAbI задачи

Набор данных bAbI-task содержит набор из 20 искусственно созданных заданий для вопросно-ответных систем (чат-ботов), которые предназначены для проверки различных аспектов логического мышления. Поскольку данные bAbI генерируются программно, а код является общедоступным, можно использовать несколько версий данных. Для наших экспериментов мы использовали подмножество en-10k данных, доступных для загрузки с сайта [24]. Для каждого из 20 заданий данные разбиваются на обучающий набор, состоящий из 10000 вопросов и тестовый набор из 1000 вопросов. Задачи bAbI состоят из историй, которые могут содержать более одного вопроса. Мы рассматривали каждую историю как отдельную последовательность и предоставляли её сети в виде векторов, соответствующих словам, по одному слову за раз. После удаления всех чисел, разделения оставшегося текста на слова и преобразования всех слов в нижний регистр в лексиконе было 156 уникальных слов и три символа пунктуации: «.», «?» и «-», последний из которых мы добавили для указания мест во входной последовательности, где требуются выходы от модели (требуется ответ от сети). Таким образом, каждое слово было представлено как вектор размером 159 в унитарном коде (one hot encoding), а выходы нейронной сети были из распределения $softmax$ размерности 159. Предложения были разделены символами полной остановки, и

все вопросы были разделены знаком вопроса, за которым следовало столько символов тире, сколько было слов в ответе на этот вопрос. Например, рассказ из заданий «Подсчет» и «Списки / множества», содержащие пять вопросов, был представлен в виде следующей последовательности входных токенов из 69 слов:

Мэри отправилась на кухню.
Мэри перешла в спальню.
Джон вернулся в коридор.
Джон взял там молоко.
Что несет Джон?
-
Джон отправился в сад.
Джон отправился в спальню.
Что несет Джон?
-
Мэри отправилась в ванную.
Джон взял яблоко там.
Что несет Джон?
- -

Ответы, соответствующие символам «-», группируются по каждому вопросу в фигурные скобки: {молоко}, {молоко}, {молоко яблоко}

4.3 Описание подзадач

Рассмотрим примеры историй и вопросов по каждой подзадаче.

4.3.1 Связывание двух объектов

1 Джон отправился в прихожую.
2 Мэри отправилась в ванную.
3 Где Джон? прихожая 1

4.3.2 Сопоставление двух фактов

1 Мэри взяла молоко.
2 Джон перешел в спальню.
3 Сандра вернулась на кухню.

4 Мэри отправилась в прихожую.
5 Где молоко? прихожая 1 4

4.3.3 Сопоставление трёх фактов

1 Мэри взяла молоко.
2 Джон перешел в спальню.
3 Даниил отправился в офис.
4 Джон взял там яблоко.
5 Джон взял футбольный мяч.
6 Джон отправился в сад.
7 Мария оставила молоко.
8 Джон оставил футбольный мяч.
9 Даниил перешел в сад.
10 Даниил взял футбольный мяч.
11 Мэри отправилась в коридор.
12 Мария пошла на кухню.
13 Джон положил там яблоко.
14 Джон поднял яблоко.
15 Сандра вышла в коридор.
16 Даниэль оставил там футбольный мяч.
17 Даниил взял футбольный мяч.
18 Джон отправился на кухню.
19 Даниил оставил футбольный мяч.
20 Джон уронил яблоко.
21 Джон схватил яблоко.
22 Джон пошел в офис.
23 Сандра вернулась в спальню.
24 Сандра взяла молоко.
25 Джон отправился в ванную.
26 Джон отправился в офис.
27 Сандра оставила молоко.
28 Мария пошла в спальню.
29 Мэри отправилась в офис.
30 Джон отправился в прихожую.
31 Сандра пошла в сад.

32 Мэри пошла на кухню.
33 Даниил взял футбольный мяч.
34 Мария отправилась в спальню.
35 Мария взяла там молоко.
36 Мария оставила молоко.
37 Иоанн пошел в сад.
38 Иоанн выбросил там яблоко.
39 Где было яблоко перед, тем как оказаться в ванной? офис 38 25 22

4.3.4 Отношение с двумя аргументами

1 Коридор к востоку от ванной комнаты.
2 Спальня находится к западу от ванной комнаты.
3 Восточнее какого объекта находится ванная комната? спальня 2

4.3.5 Отношение с тремя аргументами

1 Фред взял футбольный мяч.
2 Фред отдал футбольный мяч Джеффу.
3 Что Фред дал Джеффу? футбольный мяч 2

4.3.6 Вопросы с ответом «да, нет»

1 Мария взяла молоко.
2 Джон перешел в спальню.
3 Джон на кухне? нет 2
4 Мария оставила молоко.
5 Иоанн пошел в сад.
6 Джон на кухне? нет 5
7 Даниил отправился в спальню.
8 Даниил пошел в сад.
9 Джон в саду? да 5

4.3.7 Подсчет объектов

1 Мэри взяла молоко.
2 Джон перешел в спальню.
3 Сколько предметов несет Мэри? один 1

4 Сандра вернулась в ванную.
5 Джон взял там футбольный мяч.
6 Сколько предметов несет Мэри? один 1
7 Мэри отправилась в ванную.
8 Мария дала молоко Сандре.
9 Сколько предметов несет Мэри? нет 1 8
10 Джон вернулся в ванную.
11 Джон покинул футбольный мяч.
12 Сколько предметов несет Джон? нет 5 11
13 Сандра дала молоко Джону.
14 Иоанн отправился в сад.
15 Сколько предметов несет Джон? один 5 11 13
1 Сандра вышла в коридор.
2 Сандра взяла там яблоко.
3 Сколько предметов носит Сандра? один 2
4 Даниил перешел на кухню.
5 Сандра взяла там молоко.
6 Сколько предметов носит Сандра? два 2 5

4.3.8 Списки-множества

1 Мария взяла молоко.
2 Джон перешел в спальню.
3 Что несет Мария? молоко 1
4 Джон взял там футбольный мяч.
5 Джон отправился в ванную.
6 Что несет Джон? футбольный мяч 4
7 Иоанн пошел в сад.
8 Даниил вернулся в коридор.
9 Что несет Джон? футбольный мяч 4
10 Джон вернулся в ванную.
11 Мария пошла в офис.
12 Сандра вернулась на кухню.
13 Мария отправилась в коридор.
14 Джон отправился в спальню.
15 Джон взял там яблоко.

16 Что несет Джон? футбольный мяч, яблоко 4 15
17 Сандра отправилась в спальню.
18 Сандра отправилась на кухню.
19 Что несет Джон? футбольный мяч, яблоко 4 15
1 Даниил взял там молоко.
2 Сандра вернулась в ванную.
3 Что несет Даниил? молоко 1
4 Даниил вернулся в сад.
5 Даниил уронил молоко.
6 Что несет Даниил? ничего 1 5
7 Джон отправился в спальню.
8 Сандра отправилась в спальню.
9 Что несет Даниил? ничего 1 5
10 Даниил отправился в спальню.
11 Джон пошел в ванную.
12 Что несет Даниил? ничего 1 5

4.3.9 Простое отрицание

1 Джон в коридоре.
2 Сандра на кухне.
3 Сандра в спальне? нет 2
4 Сандра отправилась в спальню.
5 Мария отправилась в сад.
6 Сандра в спальне? да 4
7 Мэри вернулась в ванную.
8 Сандры больше нет в спальне.
9 Сандра в спальне? нет 8
10 Джон в офисе.
11 Мэри нет в ванной.
12 Мария в ванной? нет 11

4.3.10 Неопределенные знания

1 Билл на кухне.
2 Джули либо в школе, либо в кино.
3 Билл в спальне? нет 1

4 Фред в спальне.

5 Билл в школе.

6 Джули в спальне? нет 2

7 Джули в спальне или в офисе.

8 Фред в парке.

9 Джулия в спальне? может быть 7

10 Фред либо в школе, либо в спальне.

11 Мария пошла на кухню.

12 Билл в школе? да 5

4.3.11 Базовая кореферентность

1 Джон отправился в прихожую.

2 После этого он отправился в сад.

3 Где Джон? сад 1 2

4.3.12 Конъюнкция

1 Иоанн и Даниил пошли на кухню.

2 Даниил и Иоанн пошли в офис.

3 Где Даниил? офис 2

4 Даниил и Иоанн отправились в коридор.

5 Мария и Иоанн двинулись в сад.

6 Где Джон? сад 5

7 Сандра и Даниэль отправились в офис.

8 Даниил и Иоанн отправились в прихожую.

9 Где Даниил? прихожая 8

10 Мария и Даниил перешли на кухню.

11 Джон и Сандра отправились в сад.

12 Где Сандра? сад 11

13 Иоанн и Даниил отправились на кухню.

14 Даниил и Сандра отправились в коридор.

15 Где Даниил? коридор 14

4.3.13 Сложная связка

1 Даниил и Джон перешли в коридор.

2 После этого они отправились в сад.
3 Где Даниил? сад 1 2
4 Даниил и Иоанн пошли во двор.
5 После этого они вернулись в коридор.
6 Где Джон? коридор 4 5
7 Мария и Даниил вернулись в офис.
8 Затем они переехали в сад.
9 Где Джон? коридор 4 5
10 Джон и Сандра пошли в офис.
11 После этого они отправились в кухню.
12 Где Джон? кухня 10 11
13 Мария и Иоанн вернулись в офис.
14 После этого они пошли в сад.
15 Где Сандра? кухня 10 11

4.3.14 Понимание событий во времени

1 Сегодня утром Мария перешла на кухню.
2 Сегодня днем Мария поехала в кино.
3 Вчера Билл пошел в спальню.
4 Вчера Мария отправилась в школу.
5 Где была Мария перед кинотеатром? кухня 2 1

4.3.15 Базовые навыки дедукции

1 Волки боятся мышей.
2 Овцы боятся мышей.
3 Вайнона это овца.
4 Мыши боятся кошек.
5 Кошки боятся волков.
6 Джессика - мышь.
7 Эмили кошка.
8 Гертруда – это волк.
9 Чего боится Эмили? волк 7 5
10 Чего боится Вайнона? мышь 3 2
11 Чего боится Гертруда? мышь 8 1
12 Чего боится Джессика? кот 6 4

4.3.16 Базовые навыки индуктивного вывода

1 Лили – лебедь.

2 Бернхард – лев.

3 Грег – лебедь.

4 Бернхард – белый.

5 Брайан – лев.

6 Лили – серая.

7 Юлий – носорог.

8 Юлий – серый.

9 Грег – серый.

10 Какого цвета Брайан? белый 5 2 4

4.3.17 Понимание местоположения предметов

1 Розовый прямоугольник находится слева от треугольника.

2 Треугольник находится слева от красного квадрата.

3 Розовый прямоугольник справа от красного квадрата? нет 1 2

4 Розовый прямоугольник слева от красного квадрата? да 1 2

5 Розовый прямоугольник слева от красного квадрата? да 1 2

6 Розовый прямоугольник слева от красного квадрата? да 1 2

7 Розовый прямоугольник справа от красного квадрата? нет 1 2

8 Красный квадрат справа от розового прямоугольника? да 2 1

9 Является ли розовый прямоугольник слева от красного квадрата? да 1 2

10 Розовый прямоугольник слева от красного квадрата? да 1 2

4.3.18 Понимание размеров предметов

1 Коробка помещается внутри сундука.

2 Шоколадная коробка помещается внутри коробки.

3 Контейнер больше шоколадной коробки.

4 Коробка конфет помещается внутри сундука.

5 Чемодан больше контейнера.

6 Шоколад больше ящика? нет 2 1

7 Шоколад больше ящика? нет 2 1

8 Сундук вписывается в шоколадную коробку? нет 2 1

9 Сундук больше шоколадной коробки? да 1 2

10 Чемодан вписывается в шоколадную коробку? нет 3 5

4.3.19 Поиск пути

1 Сад находится к западу от ванной комнаты.

2 Спальня находится к северу от прихожей.

3 Офис к югу от прихожей.

4 Ванная комната находится к северу от спальни.

5 Кухня к востоку от спальни.

6 Как вы идете из ванной в прихожую? с, с 4 2

4.3.20 Мотивация агентов

1 Самит устал.

2 Куда пойдет Самит? спальня 1

3 Джейсон хочет пить.

4 Куда пойдет Джейсон? кухня 3

5 Самит ушёл в спальню.

6 Почему Самит пошел в спальню? устал 1

7 Джейсон перешел на кухню.

8 Почему Джейсон пошел на кухню? пить 3

9 Самит взял пижаму там.

10 Почему Саммит взял пижаму? устал 1

11 Антуан устал.

12 Куда пойдет Антуан? спальня 11

13 Антуан пошел в спальню.

14 Почему Антуан пошел в спальню? устал 11

15 Яну скучно.

16 Куда пойдет Ян? сад 15

17 Ян отправился в сад.

18 Почему Ян пошел в сад? скучно 15

19 Джейсон взял там молоко.

20 Почему Джейсон взял молоко? пить 3

21 Ян взял там футбольный мяч.

22 Почему Ян взял футбольный мяч? скучно 15

4.4 Результаты эксперимента

Сеть была обучена минимизировать кросс-энтропию выходов $softmax$ по отношению к целевым словам; выходы (ответы, полученные от модели) во время шагов, когда целевой вектор не присутствовал (не требовался ответ от модели), игнорировались. Для каждого шага, где присутствовал целевой вектор, в качестве ответа было выбрано наиболее вероятное слово в выходном распределении сети. Считалось, что сеть правильно ответила на вопрос, только если она правильно указала все целевые слова. Оценка работы модели производилась при помощи подсчета доли неправильных ответов.

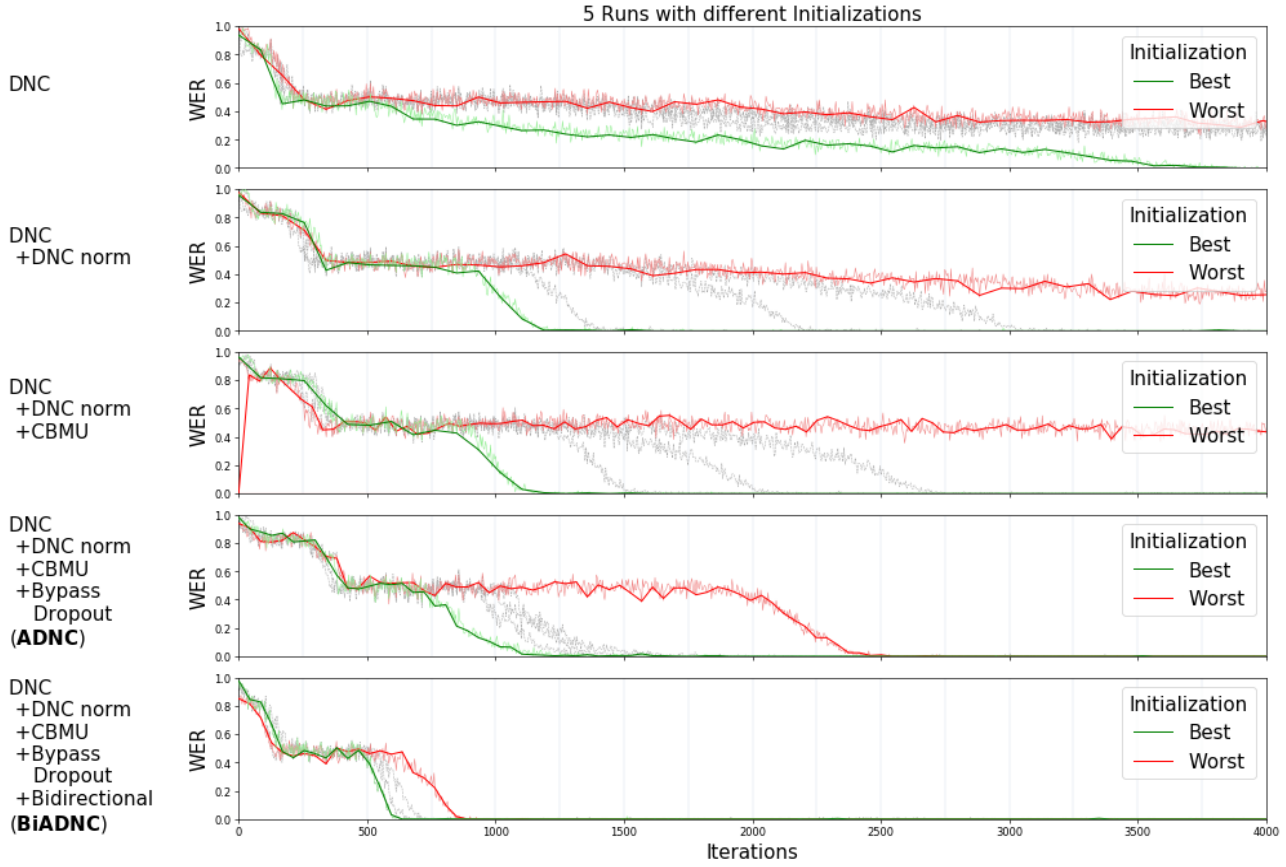


Рис. 13: Графики кривых обучения DNC и его модификаций для задачи bAbI-task

Полные результаты для всех подзадач bAbI для DNC, NTM и LSTM см. в таблицах 3.

Задача	LSTM	NTM	DNC
1: Связывание двух объектов	28.4 ± 1.5	40.6 ± 6.7	**9.0 ± 12.6**
2: Сопоставление трёх событий	56.0 ± 1.5	56.3 ± 1.5	**39.2 ± 20.5**
3: Сопоставление четырёх событий	51.3 ± 1.4	47.8 ± 1.7	**39.6 ± 16.4**
4: Отношение с двумя аргументами	0.8 ± 0.5	0.9 ± 0.7	**0.4 ± 0.7**
5: Отношение с тремя аргументами	3.2 ± 0.5	1.9 ± 0.8	**1.5 ± 1.0**
6: Вопросы с ответом «да, нет»	15.2 ± 1.5	18.4 ± 1.6	**6.9 ± 7.5**
7: Подсчет объектов	16.4 ± 1.4	19.9 ± 2.5	**9.8 ± 7.0**
8: Списки-множества	17.7 ± 1.2	18.5 ± 4.9	**5.5 ± 5.9**
9: Простое отрицание	15.4 ± 1.5	17.9 ± 2.0	**7.7 ± 8.3**
10: Неопределенные знания	28.7 ± 1.7	25.7 ± 7.3	**9.6 ± 11.4**
11: Базовая кореферентность	12.2 ± 3.5	24.4 ± 7.0	**3.3 ± 5.7**
12: Коньюнкция	5.4 ± 0.6	21.9 ± 6.6	**5.0 ± 6.3**
13: Сложная связка	7.2 ± 2.3	8.2 ± 0.8	**3.1 ± 3.6**
14: Понимание событий во времени	55.9 ± 1.2	44.9 ± 13.0	**11.0 ± 7.5**
15: Базовые навыки дедукции	47.0 ± 1.7	46.5 ± 1.6	**27.2 ± 20.1**
16: Базовые навыки индуктивного вывода	**53.3 ± 1.3**	53.8 ± 1.4	53.6 ± 1.9
17: Понимание местоположения предметов	34.8 ± 4.1	**29.9 ± 5.2**	32.4 ± 8
18: Понимание размеров предметов	5.0 ± 1.4	4.5 ± 1.3	**4.2 ± 1.8**
19: Поиск пути	90.9 ± 1.1	86.5 ± 19.4	**64.6 ± 37.4**
20: Мотивация агентов	1.3 ± 0.4	1.4 ± 0.6	**0.0 ± 0.1**

Таблица 3: Величина ошибки на тестовой выборке

Задача	SDNC	ADNC	BiADNC
1: Один опорный факт	0.0 ± 0.0	0.1 ± 0.0	0.0
2: Сопоставление двух фактов	7.1 ± 14.6	0.8 ± 0.5	0.7
3: Сопоставление трех фактов	9.4 ± 16.7	6.5 ± 4.6	2.8
4: Отношение с двумя аргументами	0.1 ± 0.1	0.0 ± 0.0	0.0
5: Отношение с тремя аргументами	0.9 ± 0.3	1.0 ± 0.4	0.7
6: Вопросы с ответом «да, нет»	0.1 ± 0.2	0.0 ± 0.1	0.0
7: Подсчет объектов	1.6 ± 0.9	1.0 ± 0.7	0.1
8: Списки-множества	0.5 ± 0.4	0.2 ± 0.2	0.187
9: Простое отрицание	0.0 ± 0.1	0.0 ± 0.0	0.0
10: Неопределенные знания	0.3 ± 0.2	0.1 ± 0.2	0.0
11: Базовая кореферентность	0.0 ± 0.0	0.0 ± 0.0	0.0
12: Коньюнкция	0.2 ± 0.3	0.0 ± 0.0	0.0
13: Сложная связка	0.1 ± 0.1	0.0 ± 0.0	0.0
14: Понимание событий во времени	5.6 ± 2.9	0.2 ± 0.1	0.0
15: Базовые навыки дедукции	3.6 ± 10.3	0.1 ± 0.1	0.0
16: Базовые навыки индуктивного вывода	53.0 ± 1.3	52.1 ± 0.9	49.8
17: Понимание местоположения предметов	12.4 ± 5.9	18.5 ± 8.8	0.00674
18: Понимание размеров предметов	1.6 ± 1.1	1.1 ± 0.5	0.748
19: Поиск пути	30.8 ± 24.2	43.3 ± 36.7	0.0
20: Мотивация агентов	0.0 ± 0.0	0.1 ± 0.1	0.0

Таблица 4: Величина ошибки на тестовой выборке

Гиперпараметры LSTM для задачи bAbI-task:

– размер пакета: 1,

– размер контроллера: 1 LSTM-слой размером 512 нейронов.

Гиперпараметры NTM для задачи bAbI-task:

– размер пакета: 1,

– количество считывающих головок: 4,

– количество слотов памяти: 256,

– размер слота памяти: 64,

– размер контроллера: 1 LSTM-слой размером 256 нейронов.

Гиперпараметры DNC для задачи bAbI-task:

– размер пакета: 1,

– количество считывающих головок: 4,

– количество слотов памяти: 256,

– размер слота памяти: 64,
– размер контроллера: 1 LSTM-слой размером 256 нейронов.

Из результатов эксперимента видно, что DNC почти всегда даёт лучший результат, чем LSTM и NTM. Также отметим, что DNC показывает наилучшие результаты в задачах определения мотивации агентов и при построении отношения двух аргументов, а наихудшие результаты – в задачах поиска пути и индуктивного вывода.

Раздел 5. Средство для визуализации DNC

В данной работе было разработано приложение на языке Python для визуализации внутренних процессов DNC. Результаты работы программы изображены на рис 14 – 18.

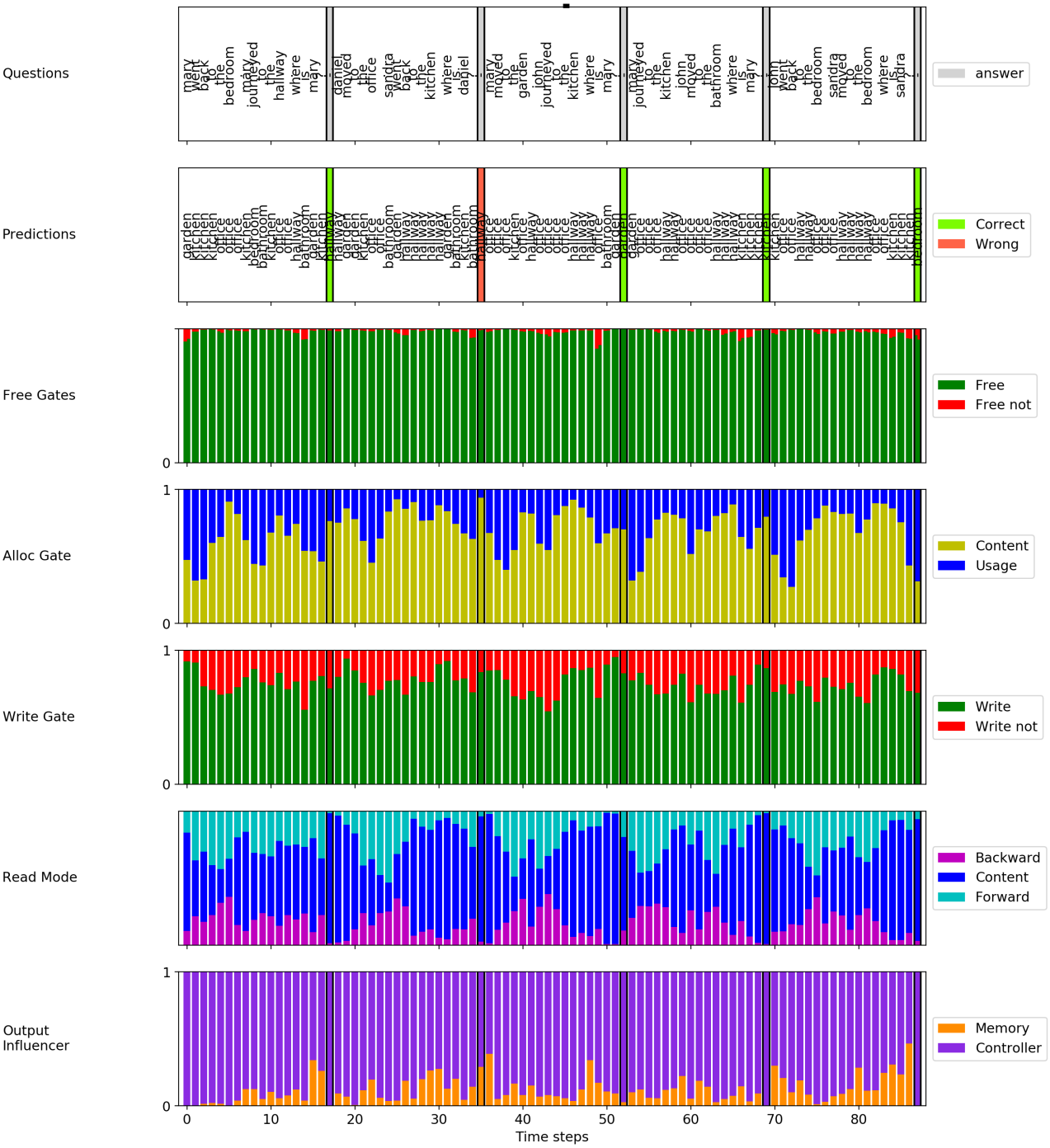


Рис. 14: Визуализация основных параметров DNC в задаче bAbI-task

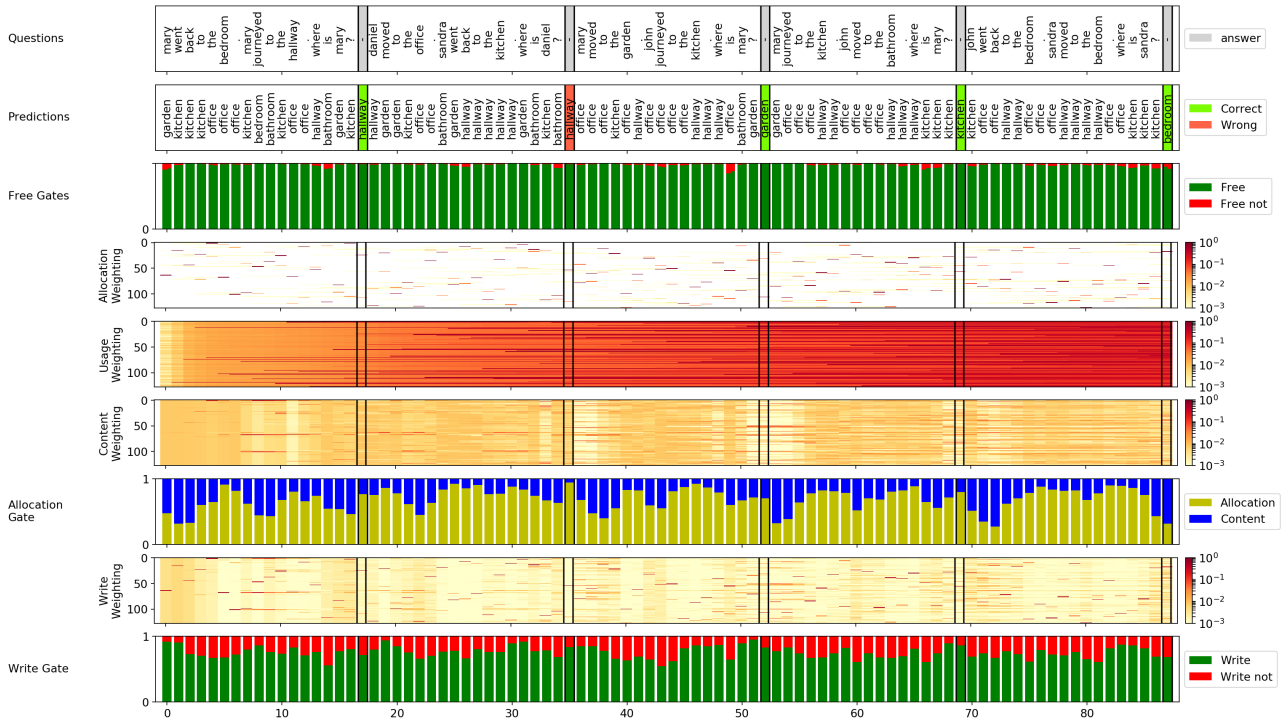


Рис. 15: Детальная визуализация операции записи DNC в задаче bAbI-task

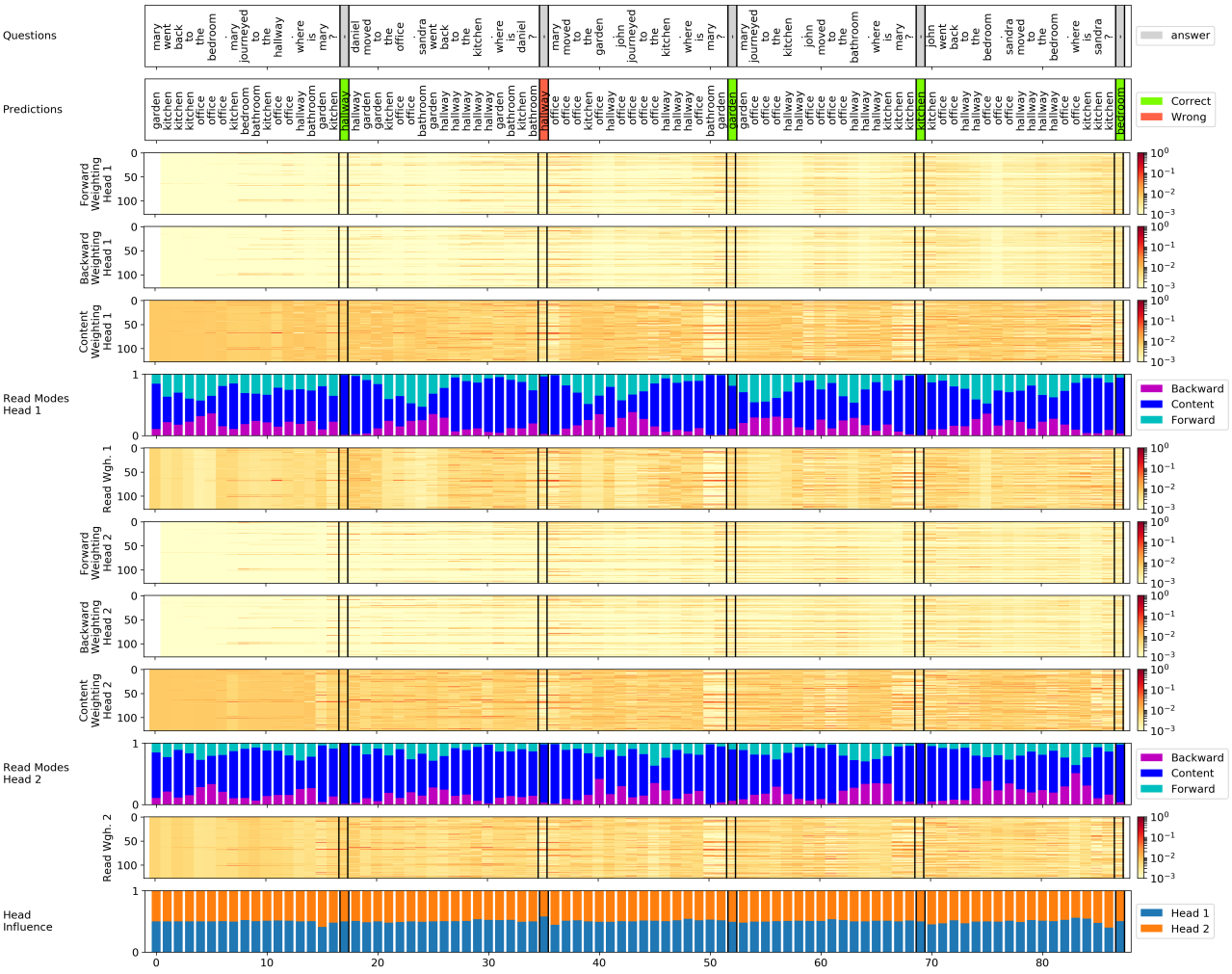


Рис. 16: Детальная визуализация операции чтения DNC в задаче bAbI-task

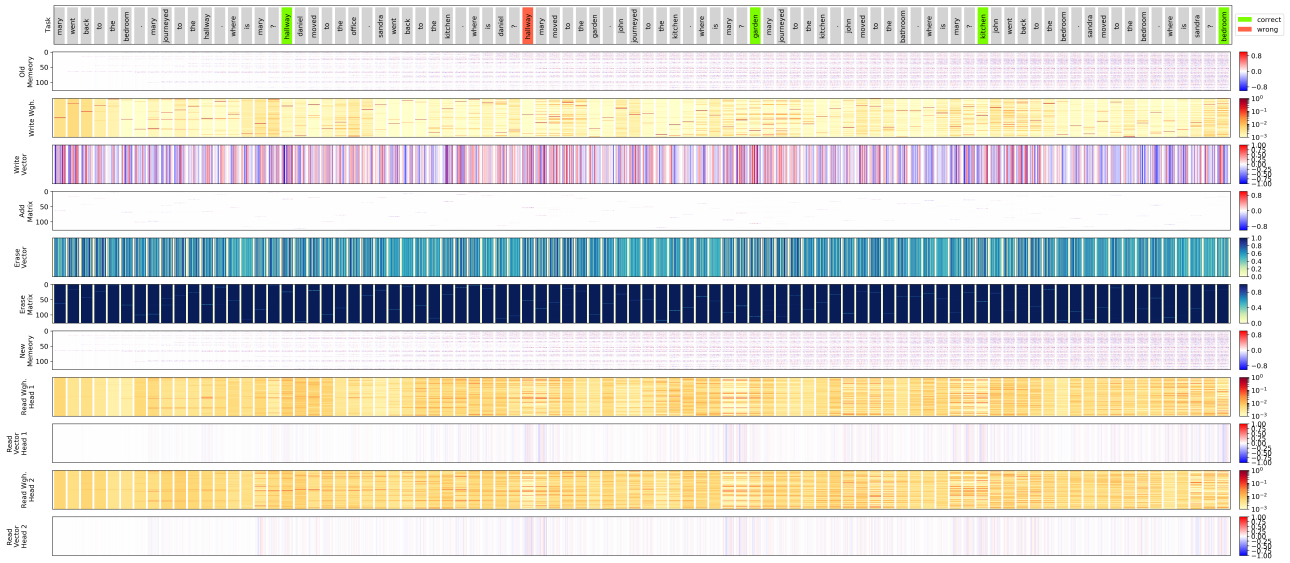


Рис. 17: Детальная визуализация внешней памяти DNC в задаче bAbI-task

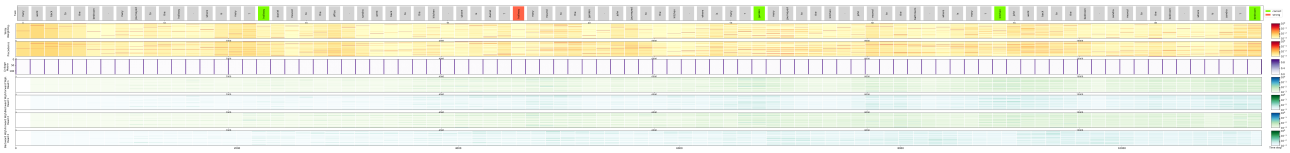


Рис. 18: Детальная визуализация матрицы временны́х ссылок DNC в задаче bAbI-task

Раздел 6. Недостатки DNC

В данной работе мы анализируем DNC в задачах QA. В работе [29] было выявлено четыре основные проблемы:

1. высокое потребление памяти затрудняет эффективное обучение больших моделей;

2. большая разница в эффективности обучения при разных инициализациях;

3. медленная и нестабильная сходимость требует длительного времени обучения;

4. однонаправленная архитектура затрудняет работу с различными видами вопросов в задачах QA.

Раздел 7. Модификации DNC

1. Надежное обучение с упором на раннее использование памяти и нормализацию.

2. Использование эффективной памяти, основанной на адресации по контенту для задач QA.

3. Двунаправленный DNC, который обеспечивает более богатое кодирование входных последовательностей.

В искусственной задаче bAbI-task [24] мы показываем повышение производительности на 80% по сравнению с обычным DNC. Мы также уменьшаем дисперсию до 90% между различными случайными инициализациями.

7.1 Анализ процесса обучения DNC

Анализ процесса обучения в работе [29] показывает, что некоторые модели сходятся, а другие нет. Это зависит только от инициализации. Если процесс обучения модели не сходится, но модель учится решать текущую задачу, значит она переобучается.

Далее проанализируем влияние блока внешней памяти. Выход DNC представляет собой взвешенную сумму выхода контроллера и выхода MU (Memory Unit). В работе [29] во время обучения моделей было обнаружено сильную корреляцию между высоким использованием MU и хорошими характеристиками модели. Когда модель не сходится, блок памяти почти не влияет на поиск правильного ответа. В работе [29] предполагается, что прямой обучающий сигнал через обходное соединение между контроллером и выходом приводит к быстрому успеху в обучении использованию исключительно контроллера. Это может помешать обучению использовать MU. Кроме того, выход MU в начале шумит, что может привести к тому, что контроллер проигнорирует MU. Это может зависеть от инициализации.

7.2 Анализ функциональности

Функциональность DNC может быть проанализирована путем наблюдения за вентилями, которые определяют механизмы записи и чтения содержимого. Проанализировав графики 14 – 18 видно, что DNC в задаче bAbI использует исключительно контентно-ориентированное чтение при ответе на

вопрос. Чтение при помощи механизма временны́х ссылок мало влияет на поиск ответа. Кроме того, использование различных вентилей: для освобождения памяти, определения механизма записи или интенсивности записи не является очень значимым. На рис. 14 показан график вентилей DNC в процессе тестирования на задаче bAbI (подзадача 1).

7.3 Анализ затрат вычислительных ресурсов

Вычислительные ресурсы DNC по сравнению с LSTM требуют в два раза больше памяти во время обучения. Это в основном зависит от объема памяти DNC и длины последовательности. Более тщательный анализ показывает, что матрица временны́х ссылок и механизм временной связи являются основными потребителями памяти. Самое большое влияние оказывает матрица временной связи размера $N \times N$. В нашей реализации время обучения на один объект в четыре раза выше по сравнению с TensorFlow LSTM с вышеупомянутой конфигурацией.

7.4 Эффективный блок внешней памяти

Для эффективного использования модели и масштабируемости для крупномасштабных задач требуется меньшее потребление памяти. Это позволяет иметь дело с большими последовательностями и большими размерами пакетов для более быстрых итераций, например, для настройки гиперпараметров. Потребление памяти DNC очень высоко по сравнению с другими рекуррентными моделями. Как показывает анализ, основной причиной потребления памяти является механизм временного связывания. Но анализ также показывает, что DNC не использует их в задаче bAbI. Это имеет смысл, поскольку восстановление последовательностей едва ли необходимо для поиска правильного ответа в задаче QA. Чтобы обеспечить более эффективное использование памяти в задачах QA, будет использоваться только контентно-ориентированный блок памяти (CBMU). CBMU имеет те же функции, что и MU DNC, но без механизма временного связывания памяти. Следовательно, матрица временно́й связи и все, имеющие отношение к ней, компоненты удаляются. Веса операций чтения основаны только на адресации по контенту. Головка записи, обновление памяти и фактическое чтение памяти остаются прежними. Снижение потребления памяти зависит от гиперпараметров и

длины последовательности, но в статье [29] оно составляет от 30% до 70%. Время вычислений также сокращается на 10-50%.

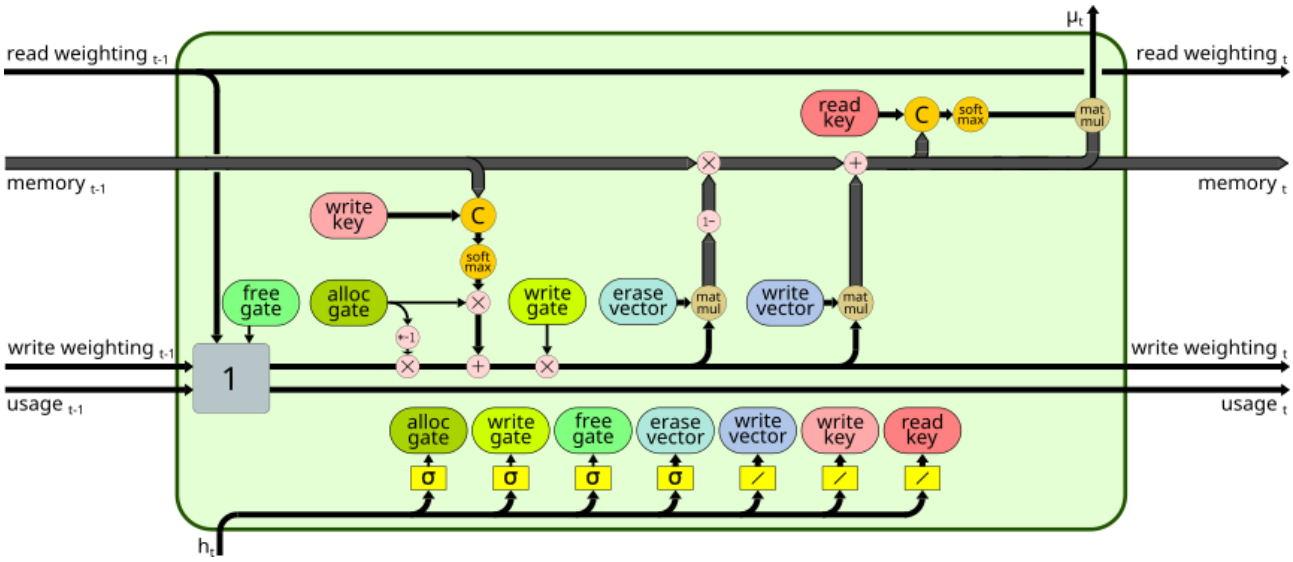


Рис. 19: Внешняя память основана только на адресации по контенту (Content Based Memory Unit)

7.5 Надежное обучение DNC

Модификации DNC позволяют добиться более надежного обучения: уменьшают большую вариацию кривых функционала качества в ходе обучения при разных инициализациях, а также устраняют медленное и нестабильное поведение сходимости. Это делает процесс обучения воспроизводимым и сокращает время обучения. Чтобы добиться этого, была применена нормализация к DNC и использовалась функция Bypass Dropout, чтобы ускорить использование памяти.

7.6 Нормализация DNC

Анализ DNC показывает высокую разницу по качеству и скорости обучения между различными запусками процесса обучения. Применим технику нормализации, чтобы обеспечить устойчивое и гладкое поведение сходимости. В последние годы нормализация слоёв (Layer Normalization) показывает улучшения производительности ANN в нескольких приложениях [30, 31]. В DNC его можно применять как к контроллеру, так и к MU. Пусть μ_t – среднее значение вектора x_t, а σ_t^2 – его дисперсия. Тогда нормализация будет иметь вид

$$LN\left(x_t\right) = g \circ \frac{x_t - \mu_t}{\sqrt{\sigma_t^2 + \epsilon}} + b.$$

Эта нормализация применяется перед каждой функцией активации в контроллере. Текущие и рекуррентные входные сигналы вычисляются совместно. Каждая нормализация слоя (LN) имеет обучаемые переменные, называемые смещением b и усилением g для масштабирования нормализации.

В MU мы применяем LN к вентилям, векторам и ключам по отдельности, но это не дало увеличения производительности по сравнению с совместной нормализацией всех сигналов. Таким образом, мы применяем его после взвешивания выходных сигналов (выходов) контроллера h_t и перед тем, как вектор ξ_t будет разделен на разные управляющие сигналы:

$$\xi_t = LN\left(h_t W_\xi\right).$$

Нормализация применяется во время обучения и тестирования. Недостатками являются увеличенное время вычислений и потребность в памяти из-за стандартизации величины x_t и дополнительных переменных: усиления g и смещения b.

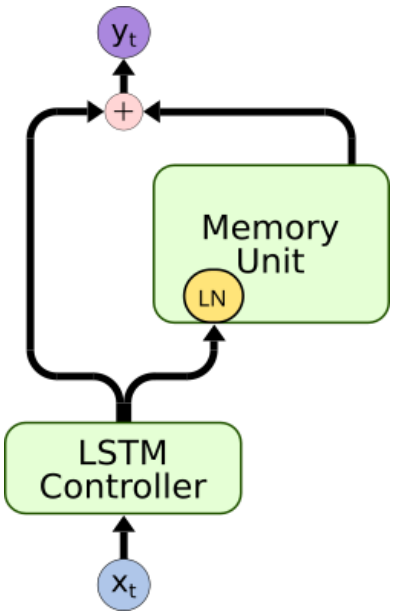


Рис. 20: DNC нормализация (DNC Normalization)

7.7 Bypass Dropout (Обходной Dropout)

Анализ в разделе 7.1 показывает, что поведение сходимости DNC зависит от использования MU. Если MU сильно влияет на производительность системы, модель достигает хорошей производительности. Это понимание требует явного принуждения к использованию MU во время обучения для более быстрого достижения сходимости и повышения производительности. Чтобы усилить влияние MU, влияние контроллера на выход через обходное соединение может быть ограничено. Это может быть достигнуто за счет уменьшения связи между контроллером и выходом.

Уменьшение возможности соединения путем взвешивания или уменьшения функционального пространства будет постоянным и не будет изменяться. При использовании Dropout в обходном соединении между контроллером и выходом возможно контролируемое снижение связности. Dropout – это метод регуляризации, введенный в 2014 году в работе [32], чтобы предотвратить переобучение ИНС. В нашем случае использование dropout позволяет настраивать сокращение обходного соединения. Он может контролироваться с вероятностью удержания и используется только во время обучения. Использование dropout помогает обеспечить более быстрое использование MU. Мы называем эту технику Bypass Dropout (BD).

Вероятность выпадения p точно регулирует поток сигналов без постоянного снижения функциональности контроллера. Bypass Dropout применяется к DNC путем умножения вектора Бернулли на обходное соединение:

$$r_t \sim Bernoulli\,(p)\,,$$

$$\boldsymbol{y}_t = \mathbf{W}_h\,(\boldsymbol{h}_t \circ \boldsymbol{r}_t) + \mathbf{W}_\mu \boldsymbol{\mu}_t.$$

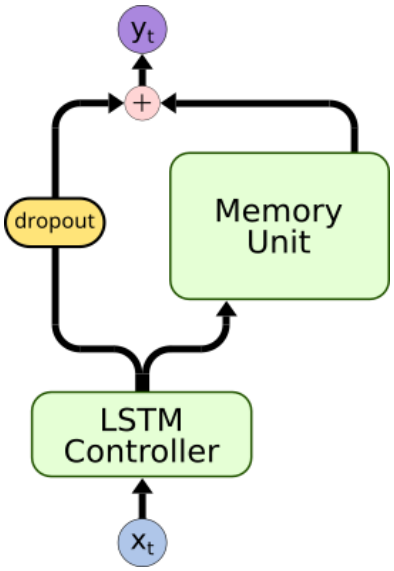


Рис. 21: Bypass Dropout

7.8 Двунаправленный DNC

Однонаправленная архитектура DNC затрудняет работу с переменным вводом, когда, например, вопрос появляется в середине текста, а полный текст актуален. Это также предотвращает извлечение богатой информации в прямом и обратном направлении в любой последовательной задаче. Поэтому в данной работе вводится двунаправленная конфигурация, чтобы обеспечить полную доступность входной последовательности для модели. Больше никакого различия между контекстом и вопросом не требуется. Модель может использовать информацию из более поздней точки входной последовательности, чтобы определить, что хранить. В двунаправленных DNC (BDNC) и rsDNC (BrsDNC) дополнительный RNN в обратной линии связи обеспечивает последовательностное понимание в обоих направлениях.

Из-за рекуррентного соединения между MU и контроллером, инкапсулированный двунаправленный контроллер невозможен, поскольку такая модель не разворачивается во времени. Решение, представленное в этой работе, применяет независимый рекуррентный контроллер с обратной направленностью, который обеспечивает дополнительный входной сигнал для MU и выходного слоя, показанного на рисунке 21. Следовательно, BrsDNC имеет два контроллера, прямой контроллер и обратный контроллер

$$\boldsymbol{h}_t^{fw} = \text{ Forward Controller } \left(\left[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}^{fw}, \boldsymbol{\mu}_{t-1}\right], \theta_{c^{fw}}\right),$$

$$\boldsymbol{h}_t^{bw} = \text{ Backward Controller } \left(\left[\boldsymbol{x}_t, \boldsymbol{h}_{t+1}^{bw}\right], \theta_{c^{bw}}\right)$$

с независимыми весами Θ и рекуррентными связями. MU получает на вход объединение двух выходов контроллера

$$\boldsymbol{\mu}_t = \text{ Memory Unit } \left(\left[\boldsymbol{h}_t^{fw}, \boldsymbol{h}_t^{bw}\right], \theta_{mu}\right).$$

Выход системы BrsDNC представляет собой сумму взвешенного выхода памяти, взвешенного выхода обратного контроллера и взвешенного выхода прямого контроллера:

$$\boldsymbol{y}_t = \mathbf{W}_{\mu}\boldsymbol{\mu}_t + \mathbf{W}_{fwh}\boldsymbol{h}_t^{fw} + \mathbf{W}_{bwh}\boldsymbol{h}_t^{bw}.$$

Эта архитектура позволяет, во-первых, независимое развертывание обратного контроллера и, во-вторых, развертывание прямого контроллера и MU.

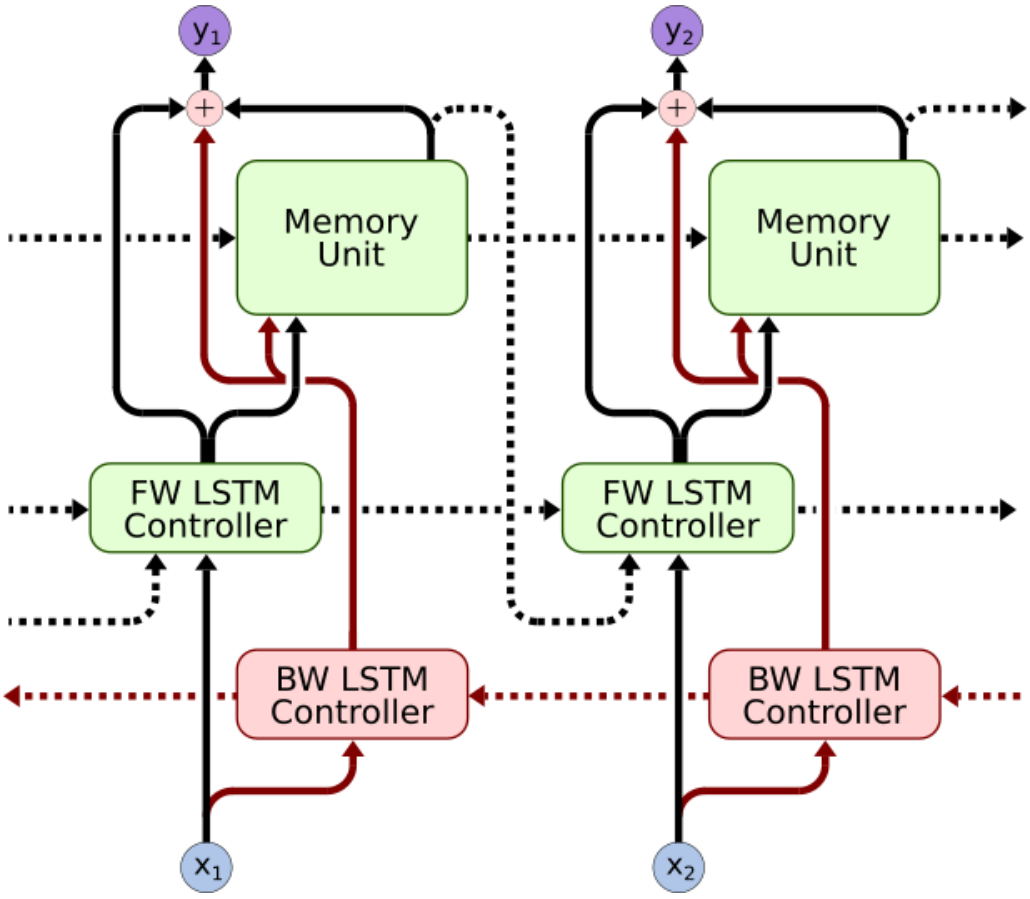


Рис. 22: Двунаправленный контроллер (Bidirectional Controller)

7.9 Детали обучения

Мы используем размер пакета для обучения 32. Контроллер представляет собой LSTM с одним скрытым слоем и размером слоя 512 в однонаправленной конфигурации и 384 каждый в двунаправленной конфигурации. Обе модели имеют матрицу памяти с 256 ячейками, шириной 128 и четырьмя считывающими головками. Dropout применяется с коэффициентом отсева, равным 10. Максимальная длина последовательности во время обучения ограничена 1400 словами. Модель оптимизирована с RMSprop с фиксированной скоростью обучения $3e-05$ и моментом 0,9.

Заключение

В данной книге детально описаны принципы работы нейросетевых моделей, а именно нейронная машина Тьюринга, дифференцируемый нейронный компьютер, а также его модификации. Перечислены сферы применимости этих моделей. Выделены преимущества этих моделей по сравнению с более ранней успешной моделью LSTM. Описаны недостатки этих моделей, а также способы их устранения. Дано теоретическое обоснование того факта, что выше рассмотренные нейронные сети с внешней памятью, обладают бóльшим потенциалом для решения многих задач, чем LSTM.

Проведены вычислительные эксперименты для задач копирования последовательностей битовых векторов, а также для задачи освоения базовых навыков вопросно-ответной системы. Результаты показывают, что нейронные сети с внешней памятью обладают большей «долгосрочной памятью», чем LSTM, имеют бóльшую обобщающую способность. В некоторых случаях они превосходят LSTM по скорости обучения.

Список использованной литературы

[1] Graves A., Wayne G., Danihelka I. / Neural Turing Machines // In: CoRR. — 2014. [Электронный ресурс] / Режим досутупа: http://arxiv.org/abs/1410.5401v2.

[2] Graves A., G. Wayne, Reynolds M. / Hybrid computing using a neural network with dynamic external memory // Nature — 2016. — Vol. 538. — P. 471 – 476. [Электронный ресурс] / Режим досутупа: https://pdfs.semanticscholar.org/7635/78fa9003f6c0f735bc3250fc2116f6100463.pdf.

[3] Unger F. / The Neural Turing Machine: Introduction, Implementation and Experiments. — 2017. [Электронный ресурс] / Режим досутупа: https://github.com/flomlo/ntm_keras/blob/master/The_NTM_-_Introduction_And_Implementation.pdf.

[4] Siegelmann H. T., Sontag E. D / On the computational power of neural nets // Journal of computer and system sciences, — 1995. — Vol. 50, №. 1, P. 132 – 150. [Электронный ресурс] / Режим досутупа: https://binds.cs.umass.edu/papers/1992_Siegelmann_COLT.pdf.

[5] Siegelmann H. T. and Sontag E. D / Turing computability with neural nets // Appl. Math. Lett. — 1991. — Vol. 4. — №. 6. — P. 77 – 80. [Электронный ресурс] / Режим досутупа: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.8383&rep=rep1&type=pdf.

[6] Allen R. J., Baddeley A. D., Hitch G. J. / Working memory and binding in sentence recall // Journal of Memory and Language — 2009. — № 6, P. 438 – 456. [Электронный ресурс] / Режим досутупа: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.603.4316&rep=rep1&type=pdf.

[7] Miller G. A. / The magical number seven, plus or minus two: some limits on out capacity for processing information // Psychological Review. — 1956. — 63(2). — pp. 81-97. [Электронный ресурс] / Режим доступа: http://www.sns.ias.edu/ tlusty/courses/InfoInBio/Papers/Miller1956.pdf.

[8] Hochreiter S. / Long sort-term memory // Neural computation — 1997. — Vol. 9(8). — pp. 1735 – 1780. [Электронный ресурс] / Режим доступа: https://www.bioinf.jku.at/publications/older/2604.pdf.

[9] Buduma N., Locascio N. /Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms // O'Reilly Media. — 1st Edition. — 2017. – P. 224 – 248. [Электронный ресурс] / Режим доступа: http://perso.ens-lyon.fr/jacques.jayez/Cours/Implicite/Fundamentals_of_Deep_Learning.pdf.

[10] Graves A., Jaitly N. / Towards End-To-End Speech Recognition with Recurrent Neural Networks // Proceedings of the 31st International Conference on Machine Learning — 2014. — Vol. 32(2). — P. 1764 – 1772. [Электронный ресурс] / Режим доступа: http://proceedings.mlr.press/v32/graves14.pdf.

[11] Graves A. / Generating Sequences with Recurrent Neural Networks. // ArXiv. — 2013. [Электронный ресурс] / Режим досутупа: https://arxiv.org/abs/1308.0850.

[12] Sutskever I., Vinyals O., Le Q. /Sequence to Sequence Learning with Neural Networks // ArXiv. — 2014. [Электронный ресурс] / Режим досутупа: https://arxiv.org/abs/1409.3215.

[13] Sutskever I., Martens J., Hinton G. / Generating Text with Recurrent Neural Networks // Proceeding ICML'11 Proceeding of the 28th International Conference on International Conference on Machine Learning. — 2011. — P. 1017 – 1024. [Электронный ресурс] / Режим доступа: https://www.cs.utoronto.ca/ ilya/pubs/2011/LANG-RNN.pdf.

[14] Hochreiter S., Bengio Y., Frasconi P., Shmidhuber J. / Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Trem Dependencies // A field guide to dynamical recurrent neural networks. IEEE Press. — 2001. — Vol. 1 [Электронный ресурс] / Режим досутупа: https://pdfs.sematics.org/2e5f/2b57f4c476dd69dc22ccdf547e48f40a994c.pdf.

[15] Graves A. / Generating Sequences With Recurrent Neural Networks // ArXiv. – 2014. [Электронный ресурс] / Режим досутупа: https://arxiv.org/pdf/1308.0850.pdf.

[16] Graves A., Mohamed Abdek-rahman, Hinton G. / Speech recognition with deep recurrent neural networks // IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) – 2013. [Электронный ресурс] / Режим досутупа: https://arxiv.org/pdf/1303.5778.pdf.

[17] Murali S. / Deep Learning papers review — NTM. // Medium. [Электронный ресурс] / Режим досутупа: https://medium.com/deep-dimension/deep-learning-papers-review-i-ntm-d55d2a1a0096.

[18] Азбука ИИ: «Рекуррентные нейросети» // N+1. — 2016. [Электронный ресурс] / Режим досутупа: https://nplus1.ru/material/2016/11/04/recurrent-networks.

[19] Рекуррентная нейронная сеть // Википедия. [Электронный ресурс] / Режим досутупа: https://www.wikipedia.org/wiki/Рекуррентная_нейронная_сеть.

[20] Greydanus S. / Differentiable memory and the brain — 2017. [Электронный ресурс] / Режим досутупа: https://greydanus.github.io/2017/02/27/differentiable-memory-and-the-brain/.

[21] Kahana M.J., Michael J. / Foundations of human memory New York // Oxford University Press. — 2012.

[22] Pérez-Ortiz J. A. / A bit-by-bit guide to the equations governing differentiable neural computers — 2017. [Электронный ресурс] / Режим доступа: https://jaspock.github.io/funicular/dnc.html.

[23] Grefenstette E., Hermann K. M., Suleyman M. / Learning to Transduce with Unbounded Memory // ArXiv. — 2015. [Электронный ресурс] / Режим доступа: https://arxiv.org/pdf/1506.02516.pdf.

[24] [Электронный ресурс] / Режим доступа: http://www.thespermwhale.com/jaseweston/babi/tasks_1-20_v1-2.tar.gz.

[25] Ross S., Gordon G. J. Bagnell J. A. / A reduction of imitation learning and structured prediction to no-regret online learning. // Proc. Fourteenth International Conference on Artificial Intelligence and Statistics. — 2010. — P. 627–635.

[26] Объяснение нейронных машин Тьюринга. — 2017. [Электронный ресурс] / Режим доступа: https://savepearlharbor.com/?p=285721.

[27] Долгая краткосрочная память // Wikipedia [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/Долгая_краткосрочная_память.

[28] Байназаров Н. / Google научил искусственный интеллект ориентироваться в Лондонском метро // Rusbase. — 2016. [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/Долгая_краткосрочная_память.

[29] Franke J., Niehues J., Waibel A. / Robust and Scalable Differentiable Neural Computer for Question Answering. // Proceedings of the Workshop on Machine Reading for Question Answering. — 2018. — P. 47 – 59. [Электронный ресурс] / Режим доступа: https://www.aclweb.org/anthology/W18-2606.

[30] Ba J. L., Kiros J. R., Hinton G. E. / Layer Normalization // ArXiv. — 2016. [Электронный ресурс] / Режим доступа: https://arxiv.org/pdf/1607.06450.pdf.

[31] Klambauer G., Unterthiner T., Mayr A., Hochreiter S. / Self-Normalizing Neural Networks // ArXiv. — 2017. [Электронный ресурс] / Режим доступа: https://arxiv.org/pdf/1706.02515.pdf.

[32] Srivastava N, G. Hinton, A. Krizhevsky, I. Sutskever, Salakhutdinov R. / Dropout: A Simple Way to Prevent Neural Networks from Overfitting // Journal of Machine Learning Research. — vol. 15. — 2014. — P. 1929 – 1958. [Электронный ресурс] / Режим доступа: http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf.

[33] Chan W., Jaitly N., Le Q. V., Vinyals O. / Listen, Attend and Spell // ArXiv. — 2015. [Электронный ресурс] / Режим доступа: https://arxiv.org/pdf/1508.01211.pdf.

[34] Chorowski J., Bahdanau D., Serdyuk D., Cho K., Bengio Y. / Attention-Based Models for Speech Recognition // ArXiv. — 2015. [Электронный ресурс] / Режим доступа: https://arxiv.org/pdf/1506.07503.pdf.

[35] Xu K., Ba J., Kiros R., Cho K., Courville A., Salakhutdinov R., Zemel R., Bengio Y. / Show, Attend and Tell: Neural Image Caption Generation with Visual Attention // ArXiv. — 2015. [Электронный ресурс] / Режим доступа: https://arxiv.org/pdf/1502.03044.pdf.

[36] Bahdanau D., Cho K., Bengio Y. /Neural Machine Translation by Jointly Learning to Align and Translate // ArXiv. — 2014. [Электронный ресурс] / Режим доступа: https://arxiv.org/pdf/1409.0473.pdf.

Приложения

Приложение 1. Схемы нейросетевых моделей и примеры выполнения операций, связанных с адресацией

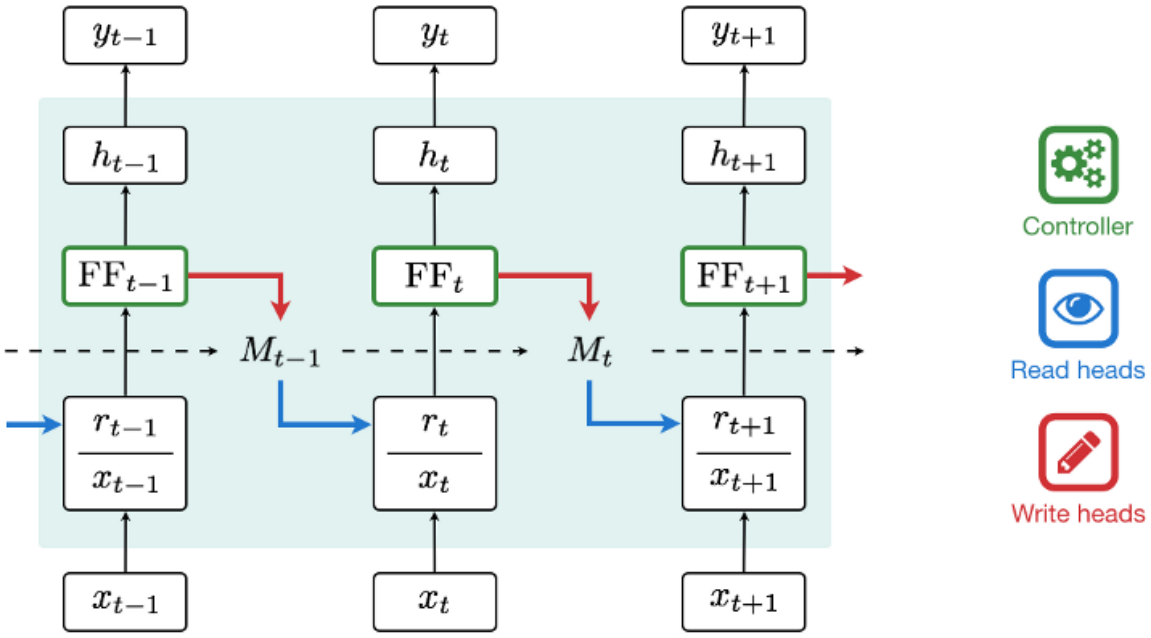


Рис. 1: Нейронная машина Тьюринга с контроллером FFNN (пошаговая схема работы).

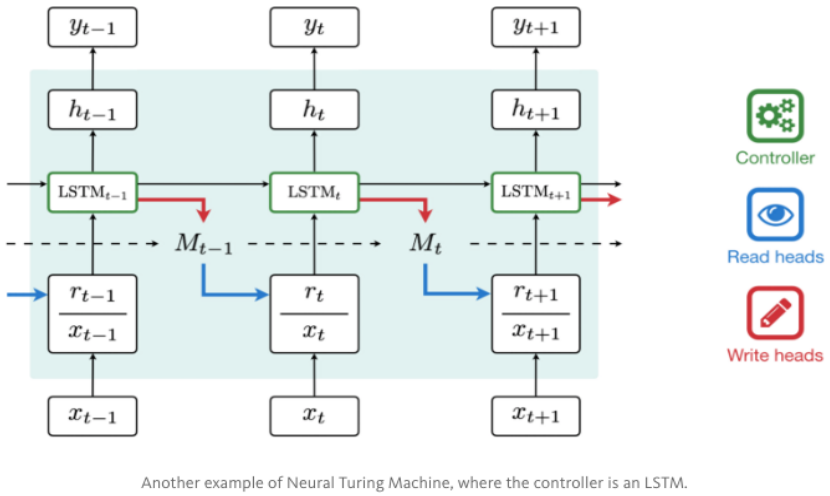


Рис. 2: Нейронная машина Тьюринга с контроллером LSTM (пошаговая схема работы).

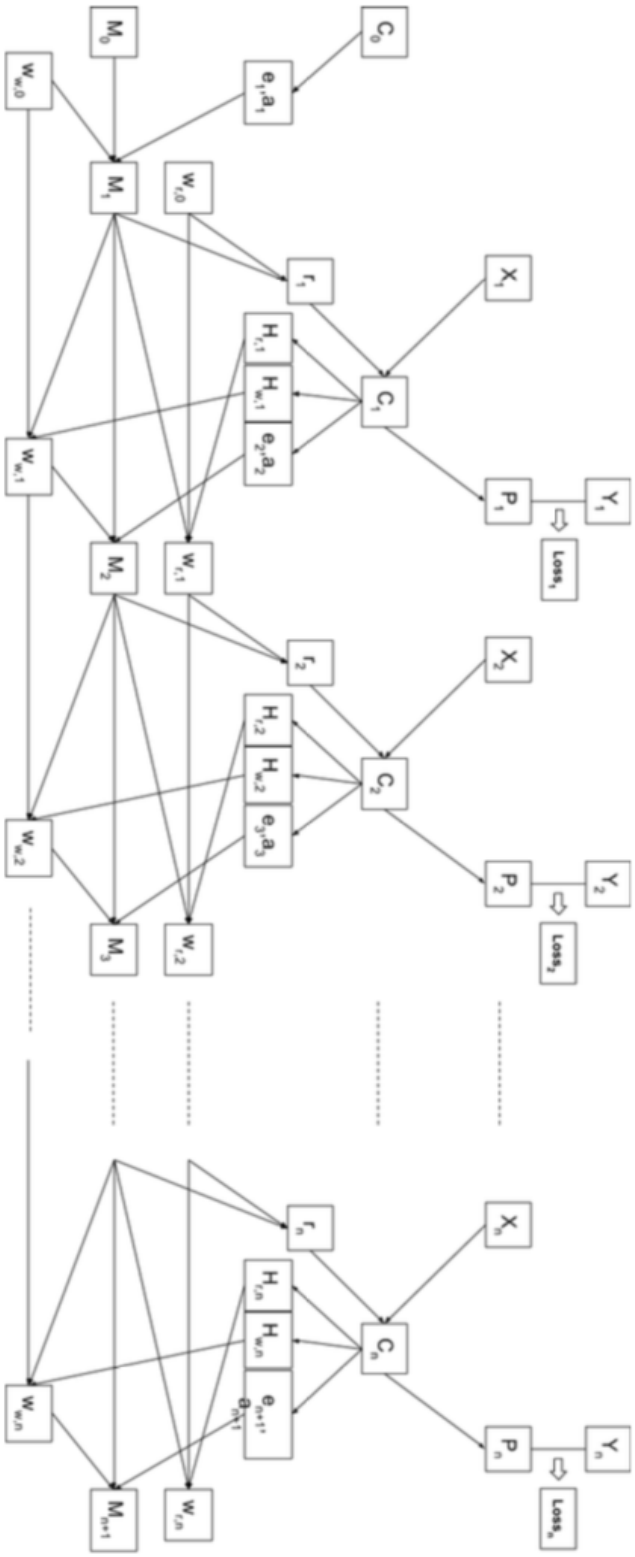


Рис. 3: Процесс обучения нейронной машины Тьюринга

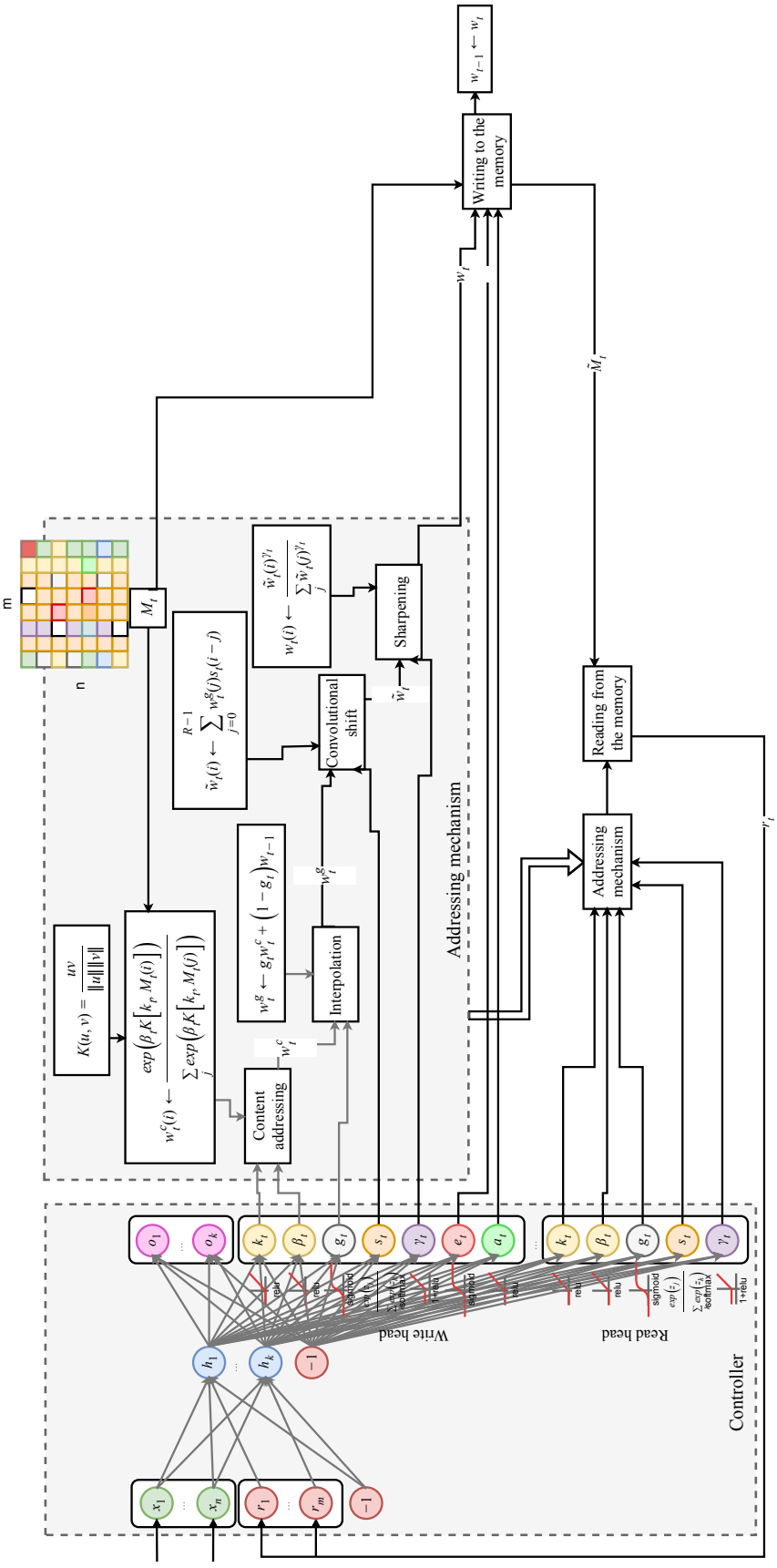


Рис. 4: Нейронная машина Тьюринга

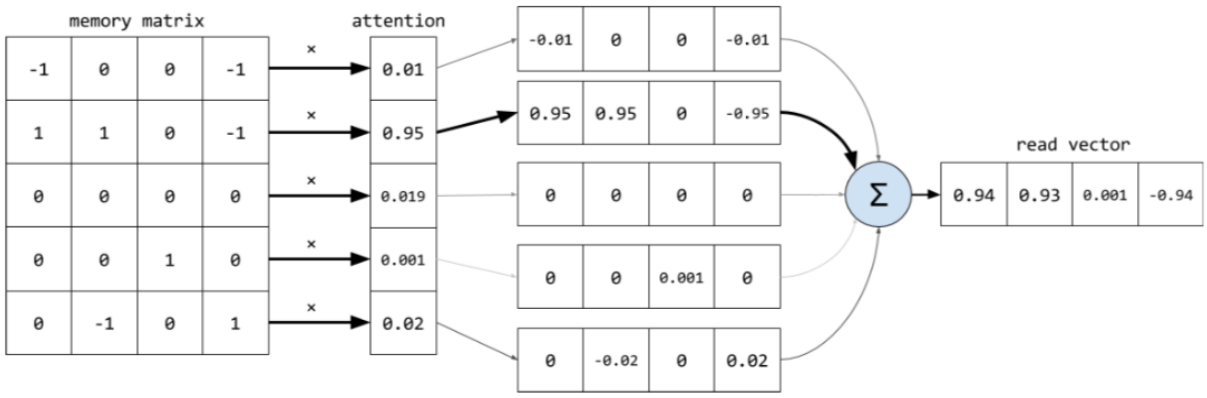


Рис. 5: Пример вычисления вектора чтения

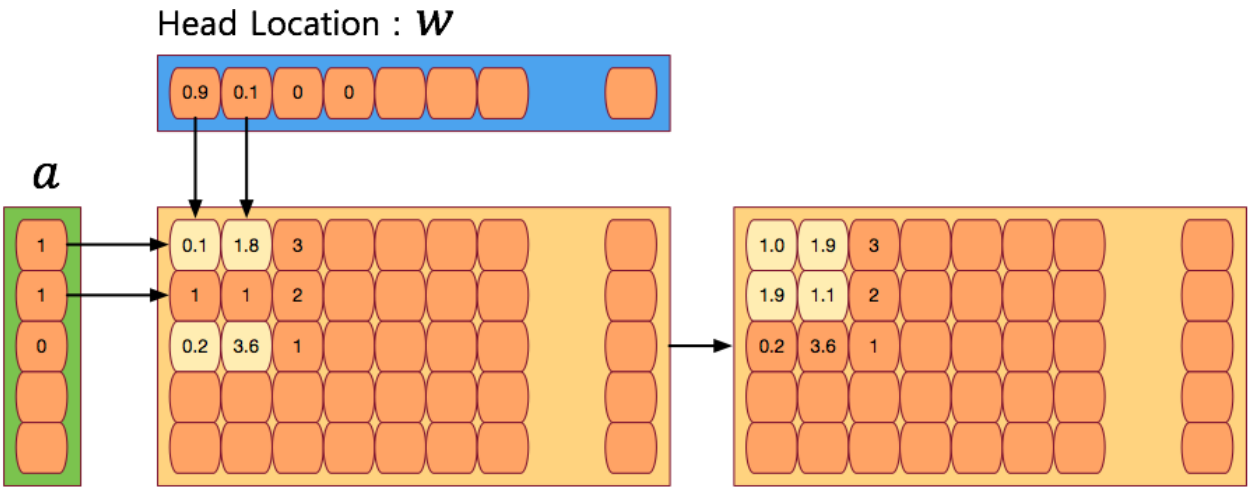


Рис. 6: Пример обновления матрицы внешней памяти при помощи вектора добавления

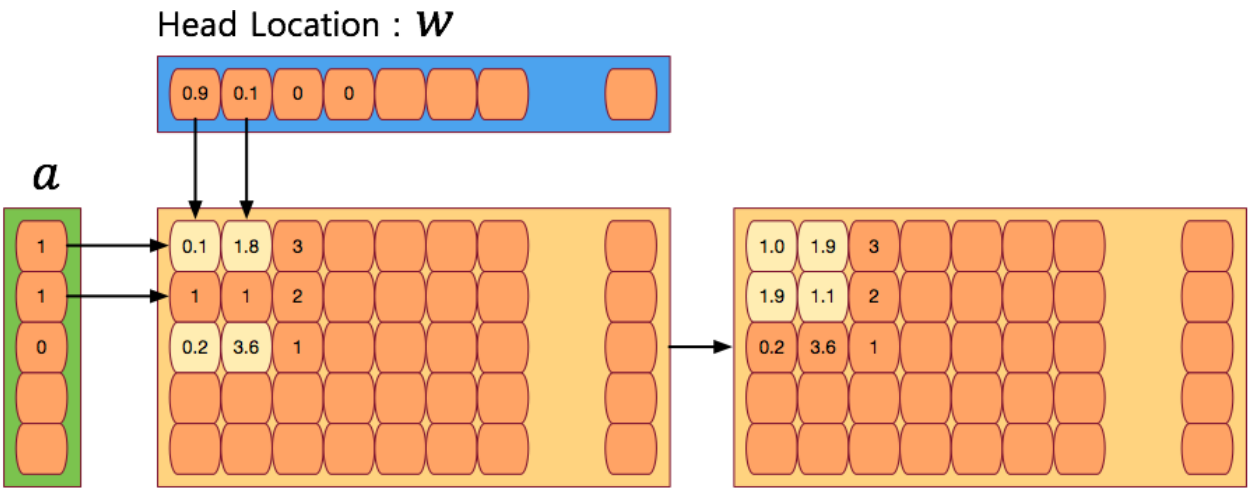


Рис. 7: Пример обновления матрицы внешней памяти при помощи стирающего вектора

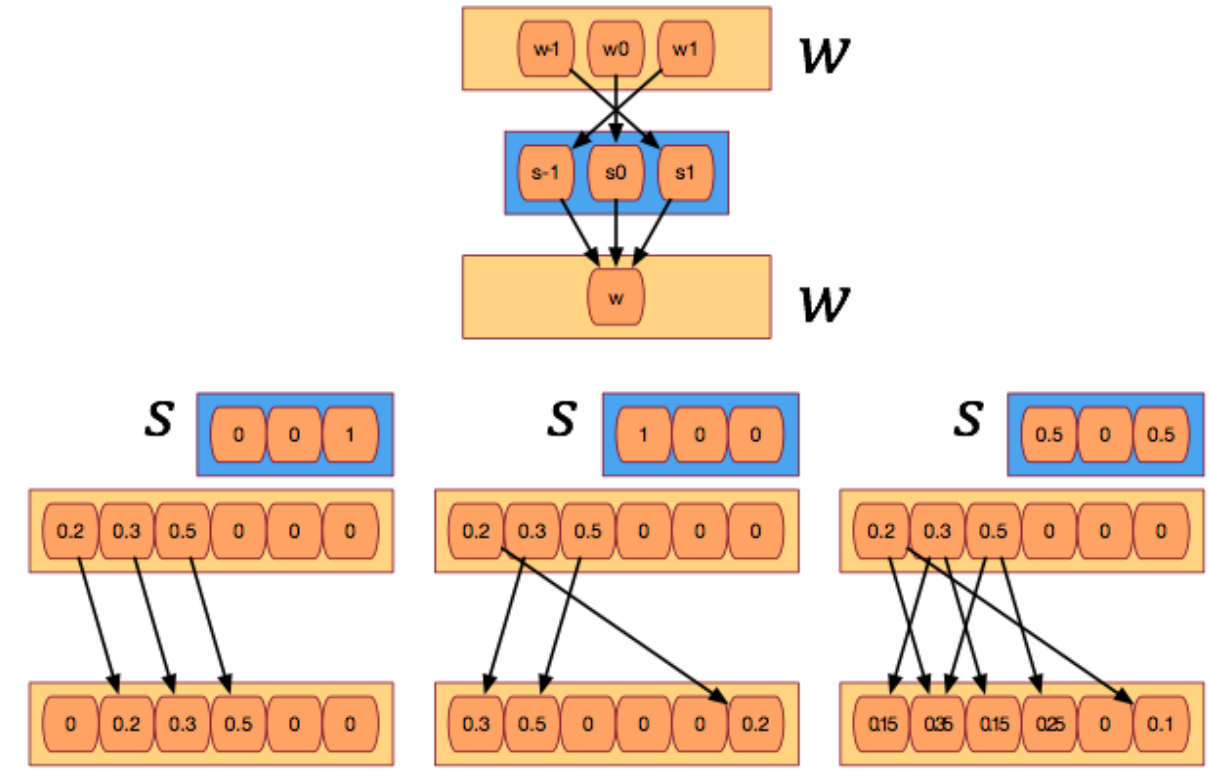


Рис. 8: Пример вычисления свёрточного сдвига

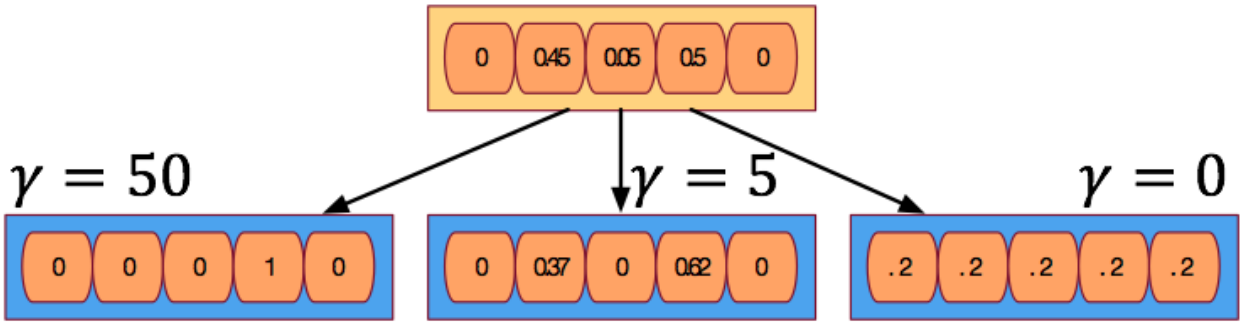


Рис. 9: Пример вычисления операции «уточнение»

Neural Turing Machine

- Real version

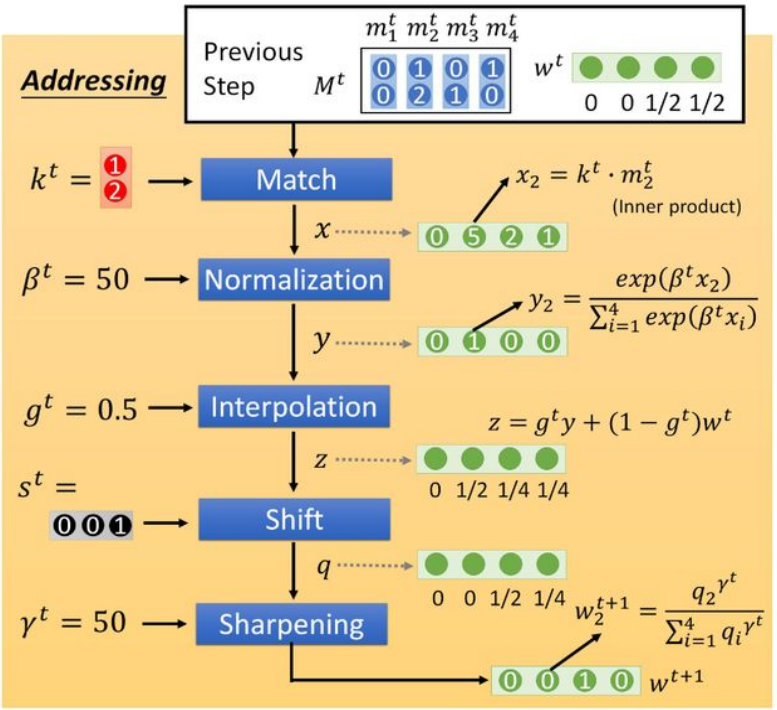


Рис. 10: Пример последовательного вычисления весового вектора

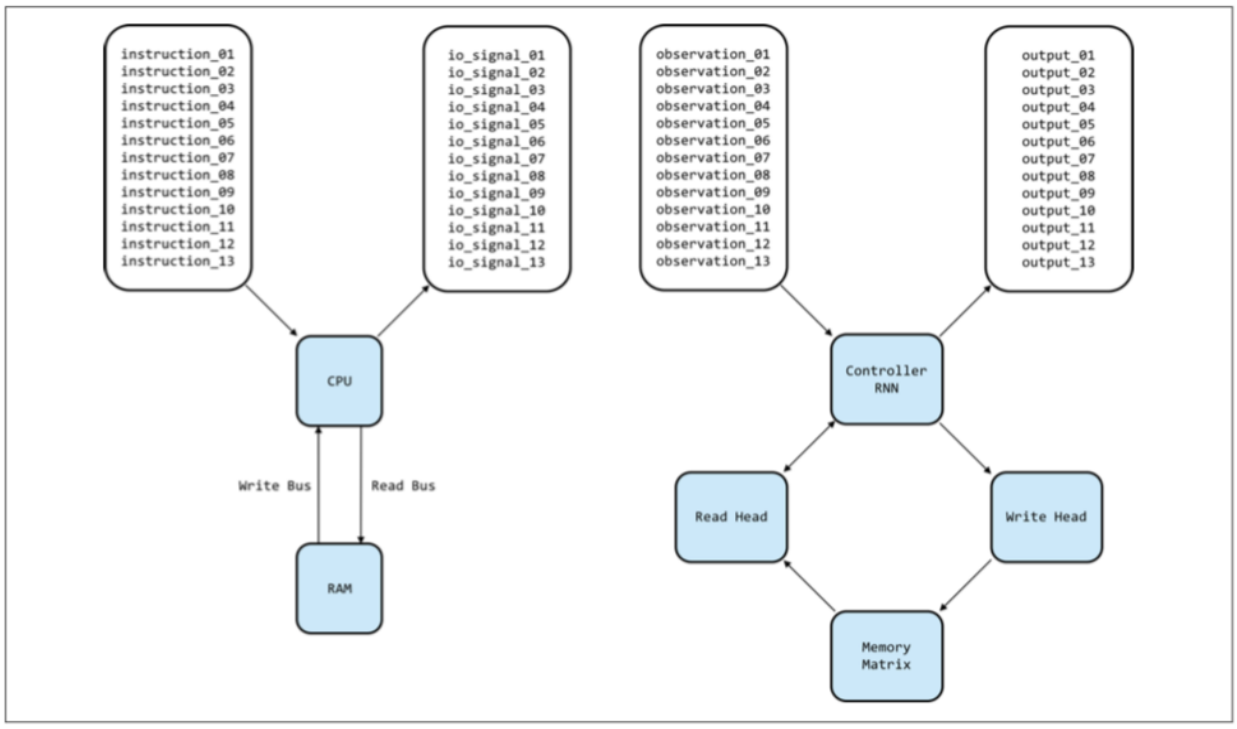


Рис. 11: Пример аналогии NTM с компьютером

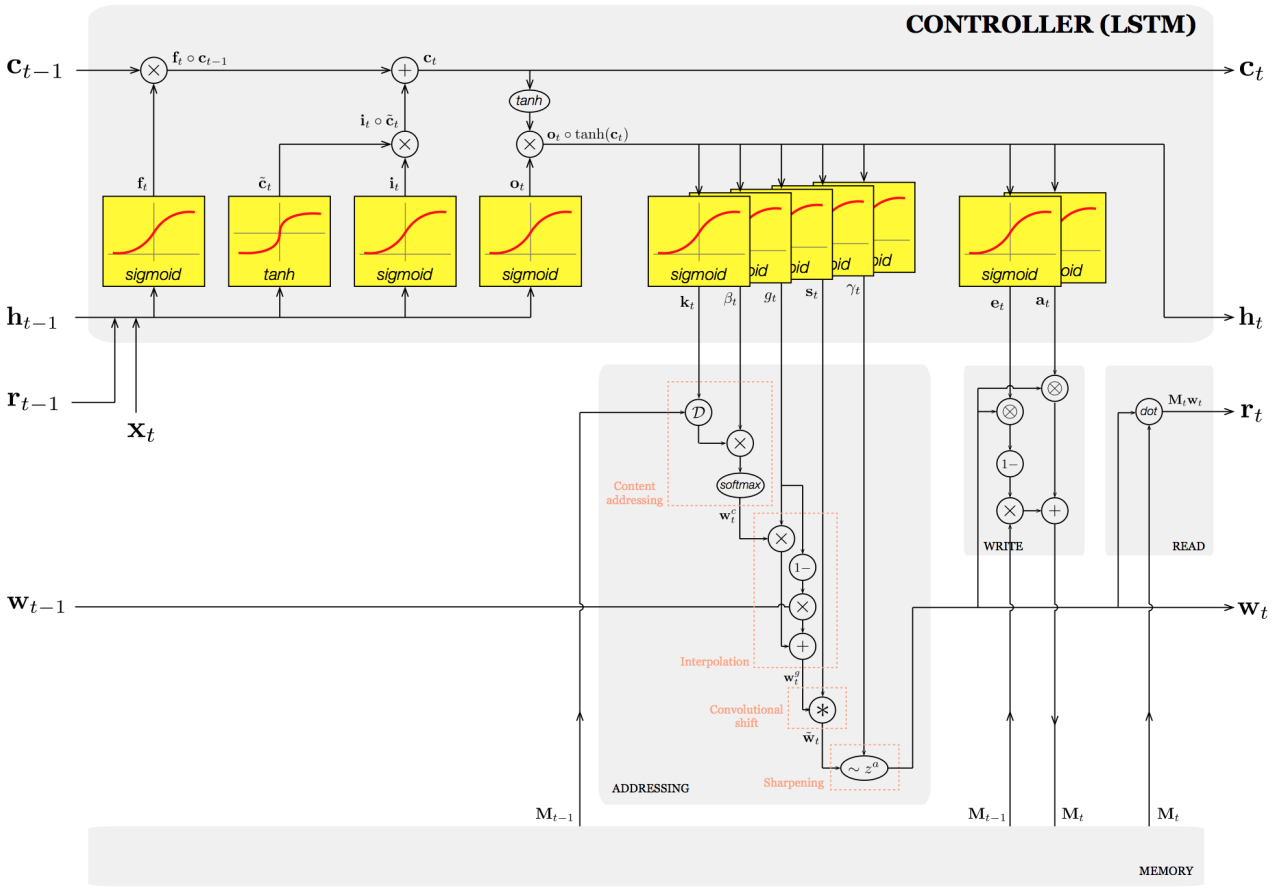


Рис. 12: Схема NTM с контроллером LSTM

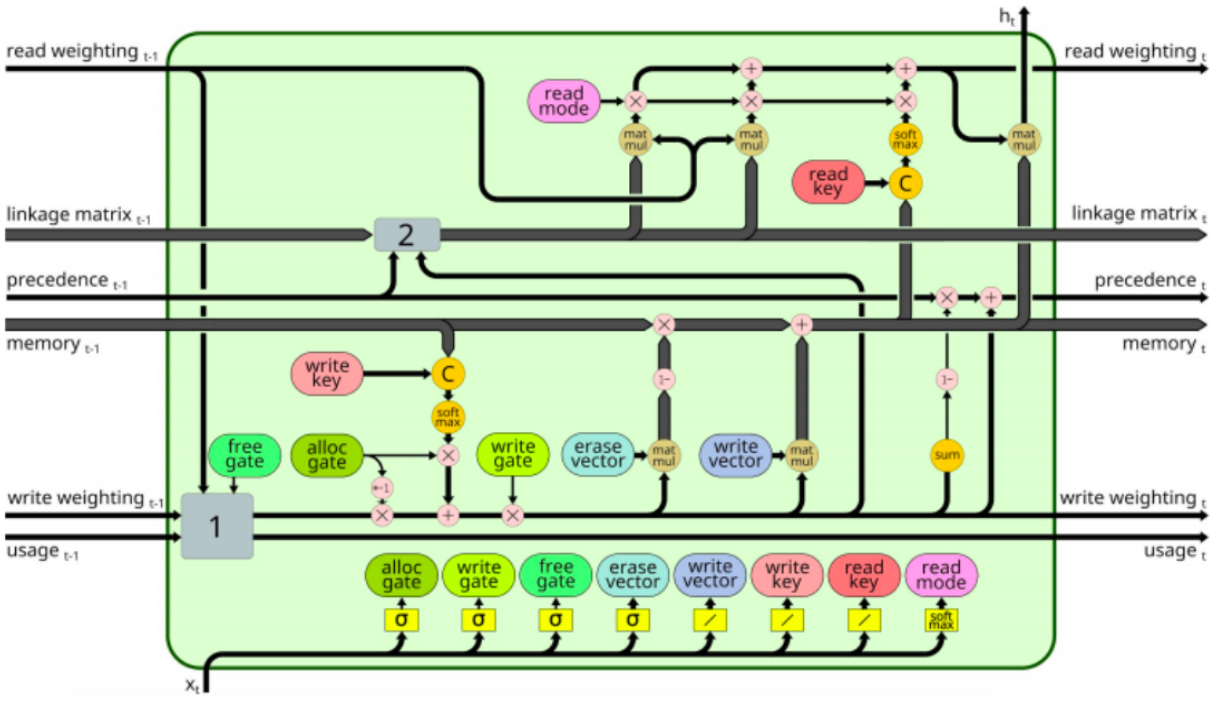


Рис. 13: Схема адресации DNC

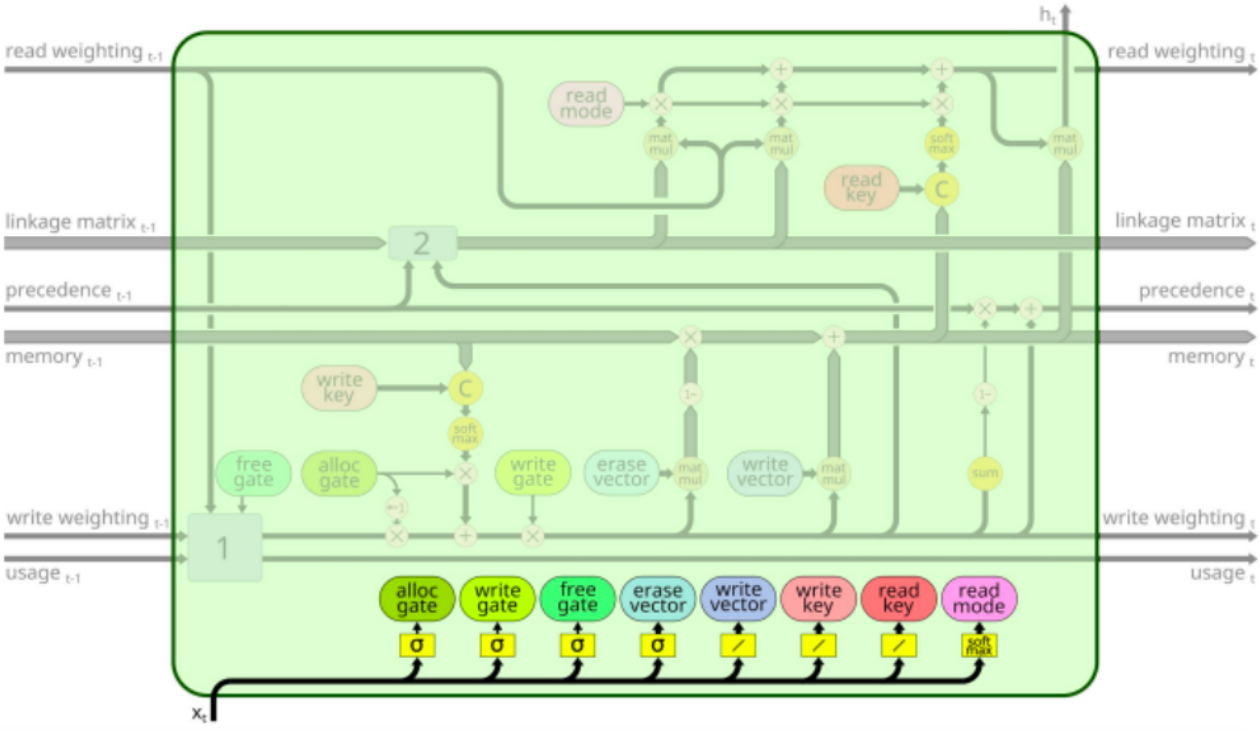


Рис. 14: Параметры адресации, получаемые от контроллера

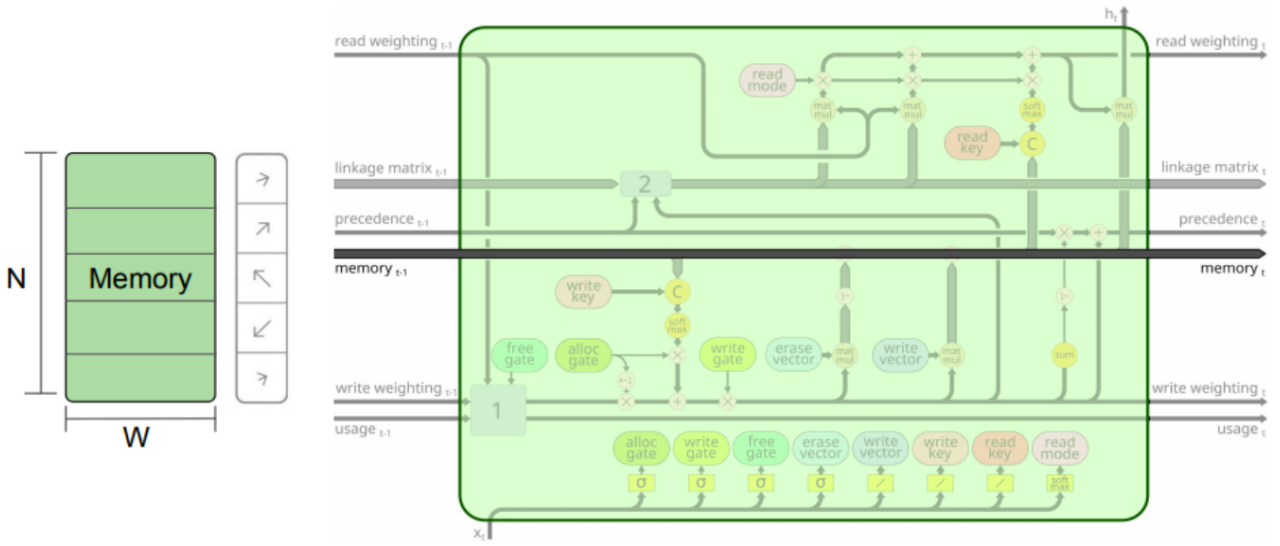


Рис. 15: Использование матрицы внешней памяти в механизме адресации DNC

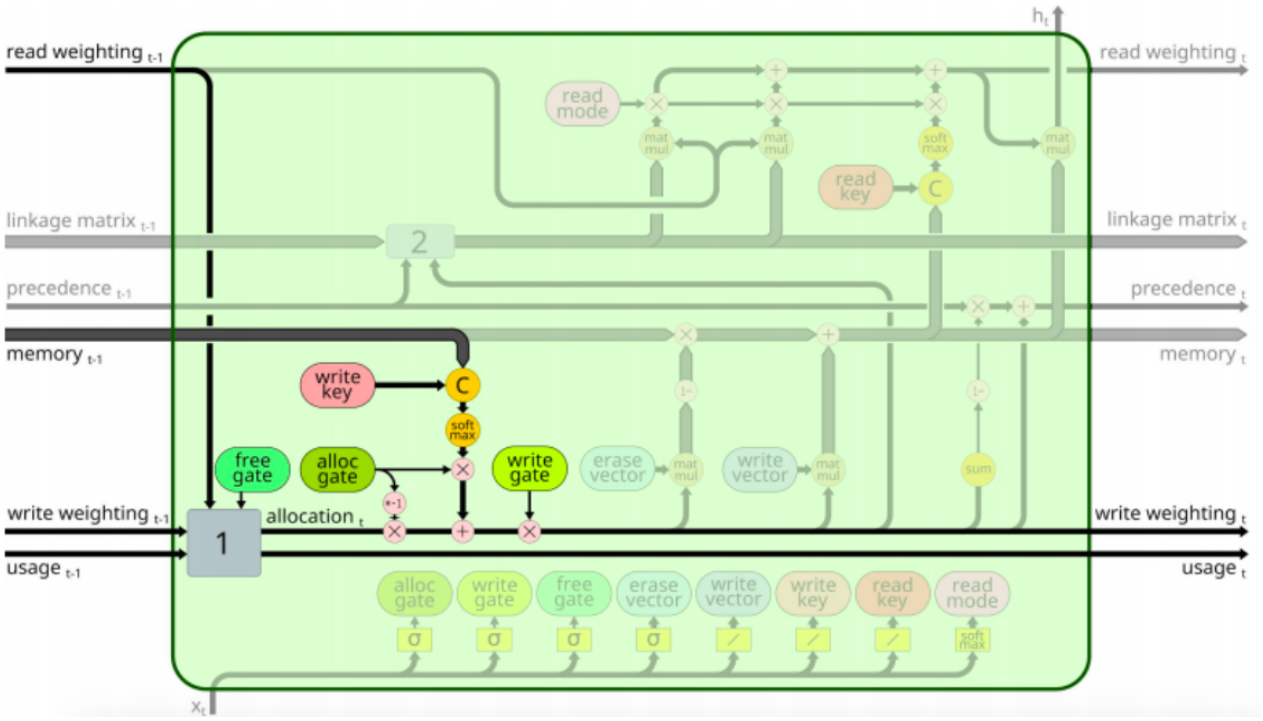


Рис. 16: Создание новых весовых векторов записи

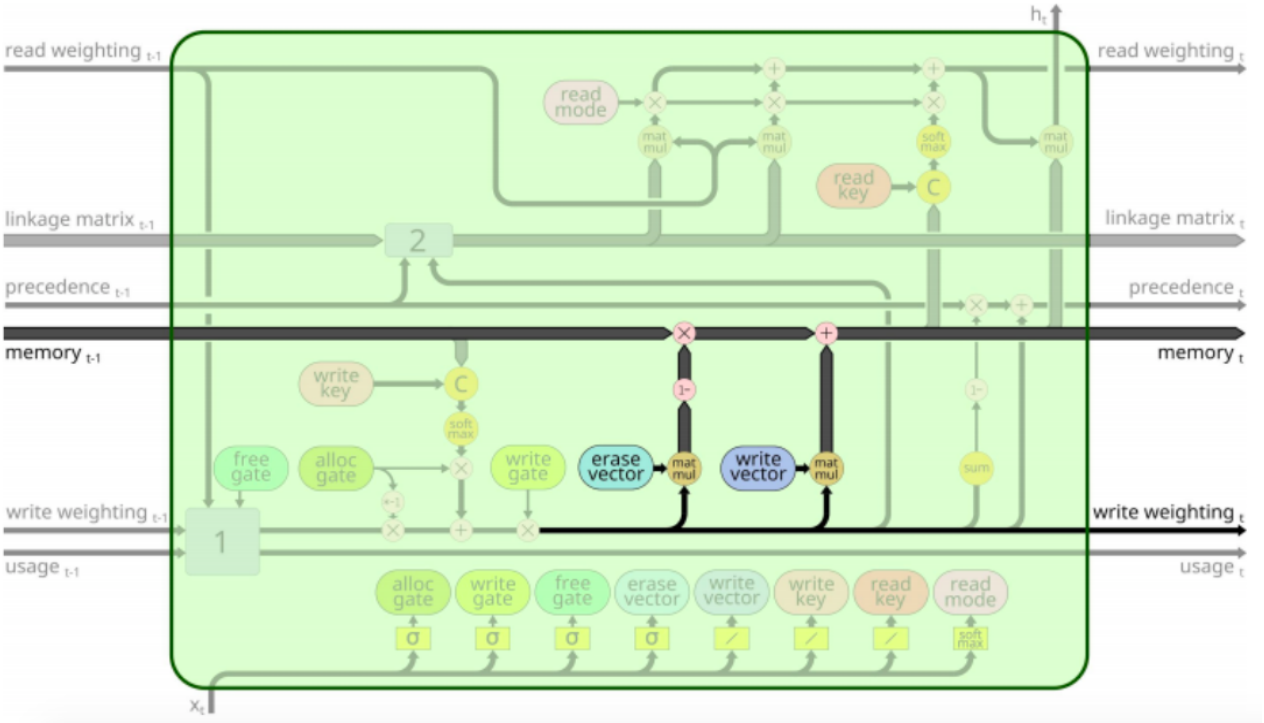


Рис. 17: Очистка памяти и запись новой информации в память

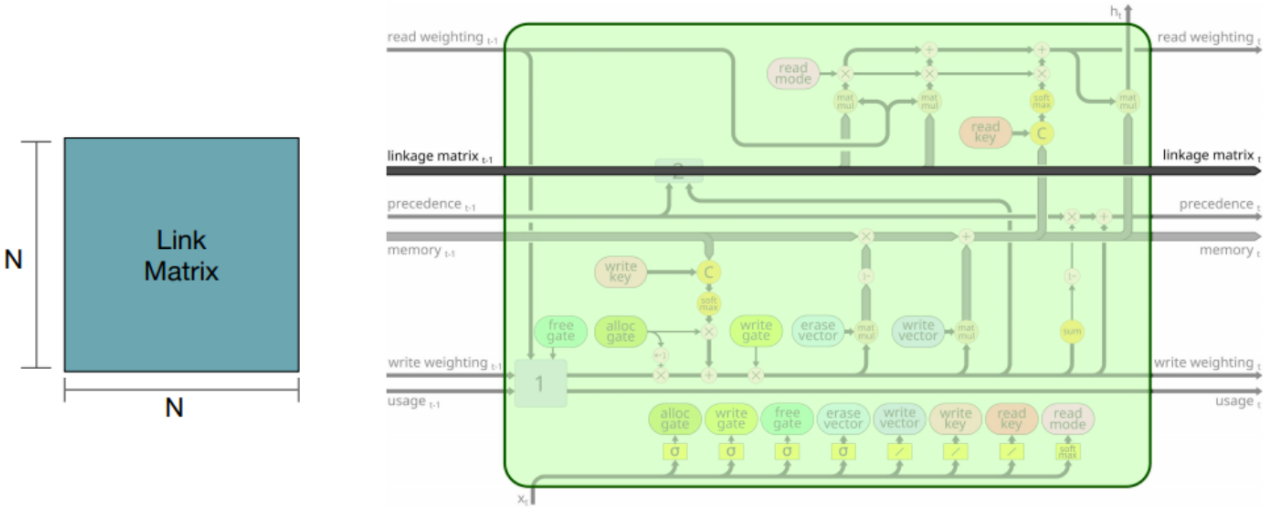


Рис. 18: Использование матрицы временных ссылок в механизме адресации DNC

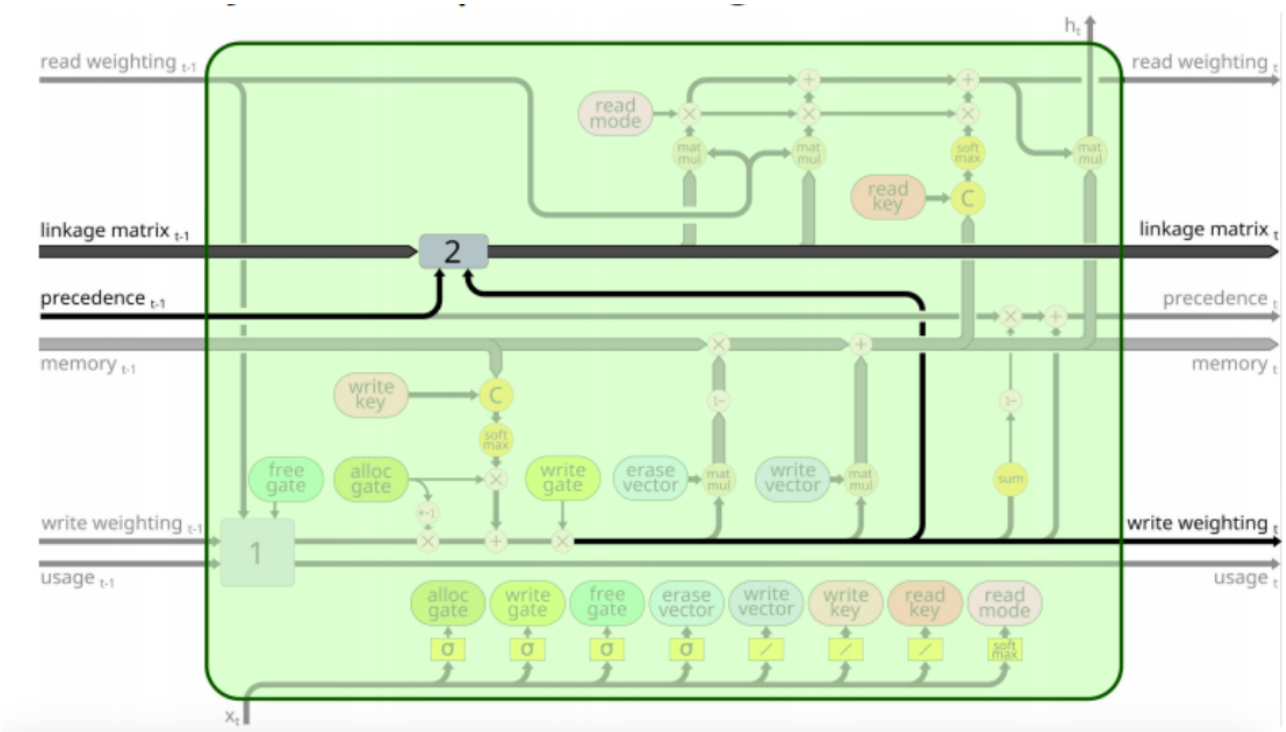


Рис. 19: Обновление матрицы временных ссылок

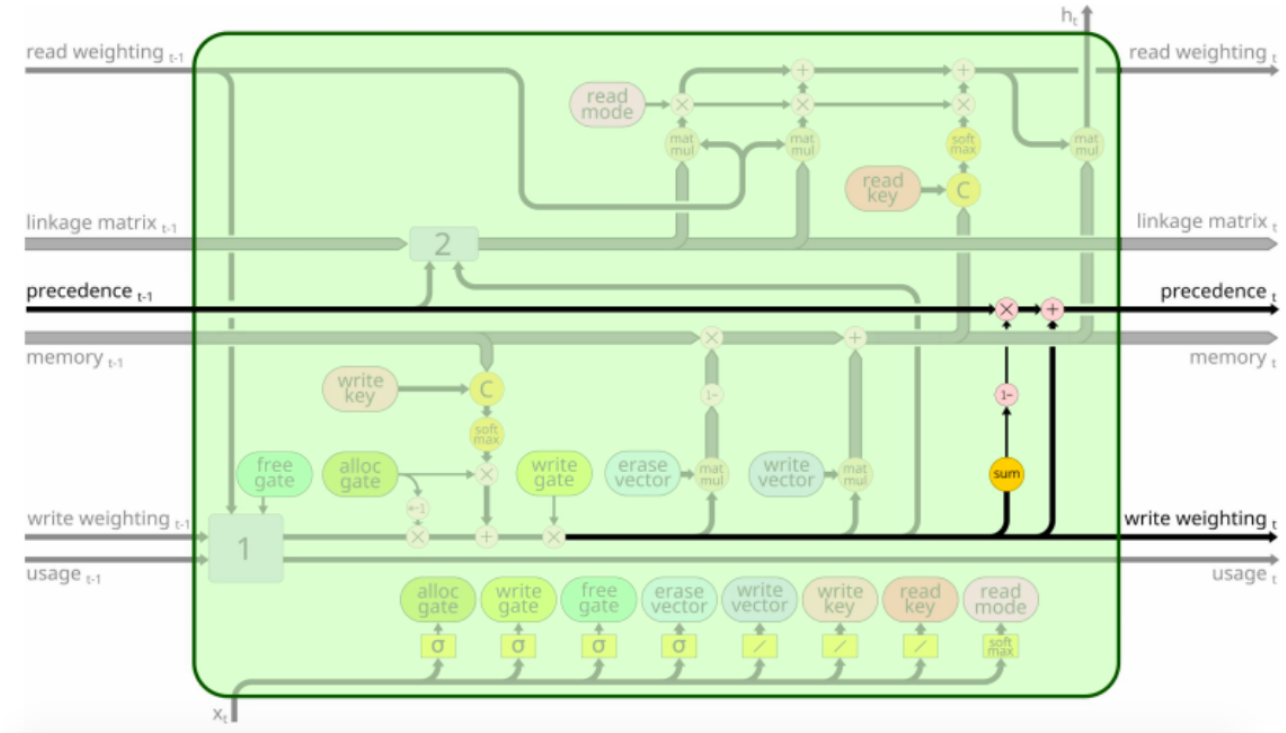


Рис. 20: Обновление вектора приоритета

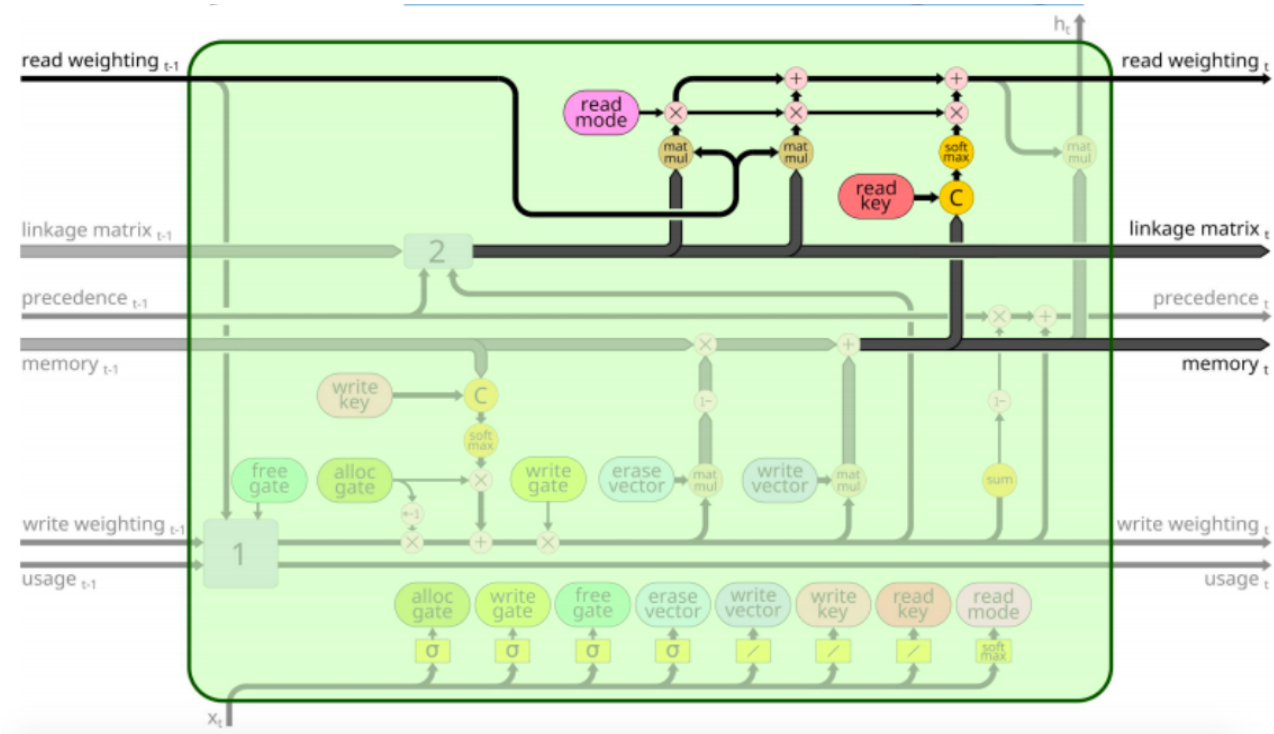


Рис. 21: Создание нового весового вектора для операции чтения

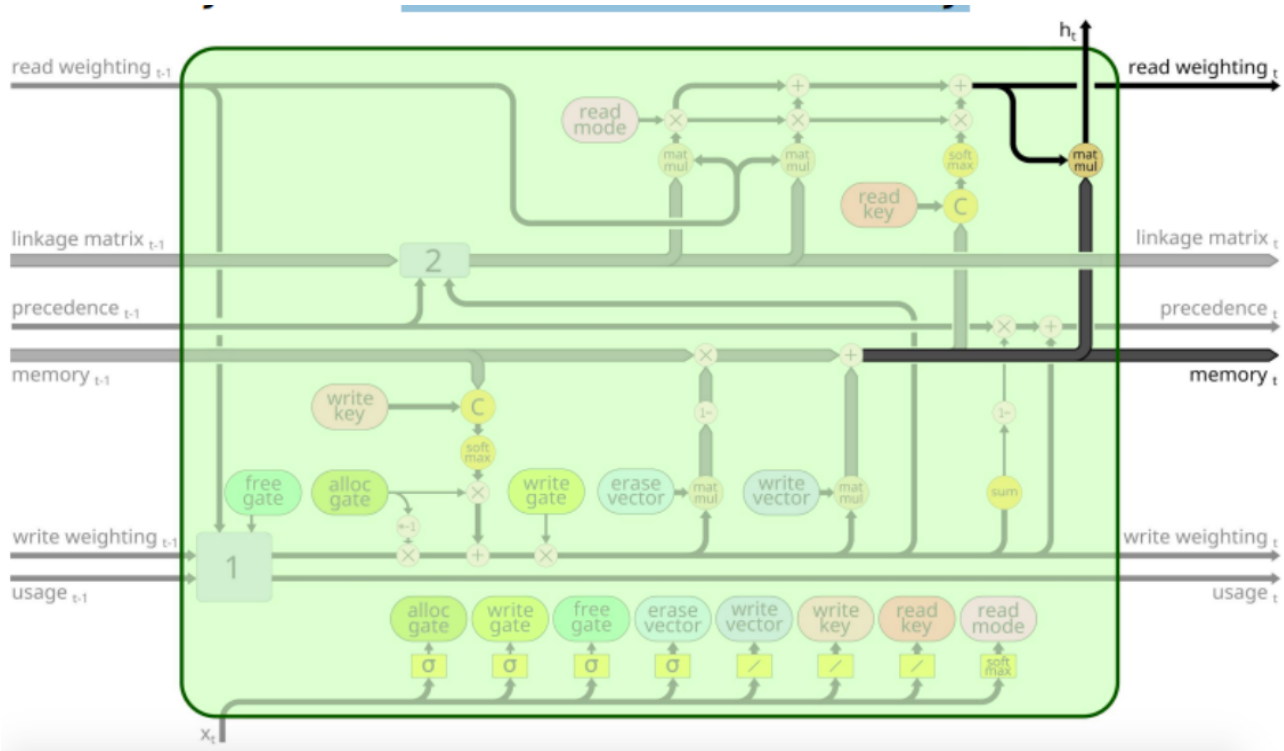


Рис. 22: Чтение из памяти

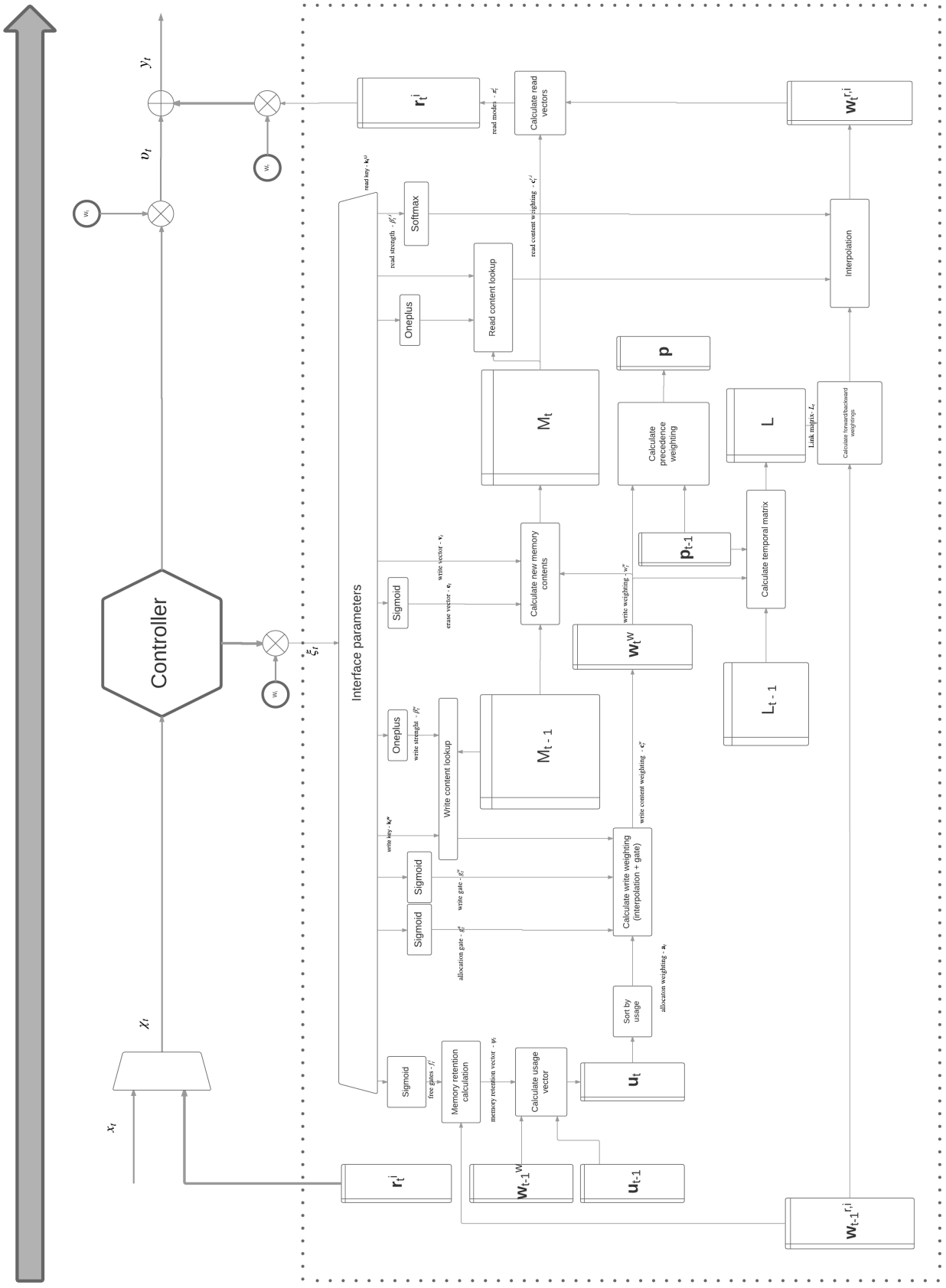


Рис. 23: Детальная схема DNC