Министерство образования и науки Российской Федерации
МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(государственный университет)
ФАКУЛЬТЕТ УПРАВЛЕНИЯ И ПРИКЛАДНОЙ МАТЕМАТИКИ
КАФЕДРА ТЕОРЕТИЧЕСКОЙ И ПРИКЛАДНОЙ ИНФОРМАТИКИ

ЧАЩИН АРТЁМ ВАЛЕРЬЕВИЧ

МОДЕЛИРОВАНИЕ СИСТЕМЫ УПРАВЛЕНИЯ РЕСУРСАМИ ЯДРА ОС LINUX НА ОСНОВЕ АППАРАТА ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

Выпускная квалификационная работа бакалавра

Направление подготовки 010900 «Прикладные математика и физика»

Научный руководитель _____ П.В. Емельянов

Студент _____ А.В. Чащин

г. Долгопрудный
2016

Содержание

1 Введение

Многие сталкивались с проблемой медленно работающего компьютера. Согласно исследованиям, проведённым компаниями Sandisk и Micron Technology, в течение года среднестатистический пользователь ПК теряет время, эквивалентное одной рабочей неделе, в ожидании медленной загрузки компьютера и приложений. [13–17] Если для рядовых пользователей трата времени не имеет серьёзных последствий, то для компаний медленная работа системы или приложений может принести серьёзные убытки. Казалось бы, простой выход из ситуации — посмотреть на счётчики системы, понять, какие ресурсы программа потребляет и по возможности добавить их в систему. Проблема заключается в том, что ядро может выделить программе совсем не те ресурсы, которые она хочет потреблять. В этом случае определить причину медленной работы без понимания, чего программа хочет на самом деле, затруднительно.

Существует и обратная задача — по ресурсам, которые программа запрашивает у ядра, понять, какие ресурсы будут выделены ей. Если система ненагружена и программа не слишком ресурсоёмкая, то будет логичным ожидать выделенные ресурсы примерно равными запрашиваемым. Однако в нагруженной системе или при ресурсоёмкой программе это правило нарушается. Так или иначе, нужно уметь конвертировать желаемое в выделяемое и наоборот.

Цель работы — моделирование поведения ядра в вопросах потребления и предоставления ресурсов с помощью нейронных сетей.

Методы исследования:

1. написание тестовых программ, потребляющих заранее заданную долю ресурсов;
2. обучение нейронной сети на тестовой выборке;
3. построение нейронной сети, которая в первом приближении моделирует выделения ресурсов ядром Linux;

Объект исследования — ресурсы ядра Linux.
Предмет исследования — искусственные нейронные сети.

2 Математический аппарат нейронных сетей

2.1 Задача обучения по прецедентам

Для моделирования выделения ресурсов ядром Linux мы решили выбрать нейронную сеть. Для начала рассмотрим общую задачу, которую решает нейронная сеть. Пусть X - множество объектов, Y - множество допустимых ответов, $y^* : X \to Y$ - целевая функция, $X^l = (x_i, y_i)_{i=1}^l$ - обучающая выборка, состоящая из прецедентов (x_i, y_i).

Задача обучения по прецедентам заключается в восстановлении зависимости y^* по выборке X^l. [8] Для этого строится решающая функция $a : X \to Y$, которая приближала бы y^* на всём множестве X. Эту функцию также называют алгоритмом, т.к. она должна допускать эффективную компьютерную реализацию.

Процесс построения алгоритма называется обучением. Методом обучения называется отображение $\mu : (X, Y)^l \to A$, которое сопоставляет алгоритм произвольной конечной выборке. В задачах обучения по прецедентам всегда есть два этапа:

- этап обучения (метод μ по выборке X^l строит алгоритм $a = \mu(X^l)$);
- этап применения (алгоритм a для новых объектов x выдаёт ответы $a(x)$).

Одним из классических методов обучения является минимизация эмпирического риска (empirical risk minimization). Для него вводится функция потерь $\mathscr{L}(a, x)$, показывающая величину ошибки алгоритма a на объекте x. Ответ $a(x)$ называется корректным, если $\mathscr{L}(a, x) = 0$.

Также вводится функционал качества (или эмпирический риск) алгоритма a на выборке X^l:

$$Q(a, X^l) = \frac{1}{l} \sum_{i=1}^{l} \mathscr{L}(a, x_i). \tag{1}$$

Метод заключается в минимизации эмпирического риска на заданной выборке X^l по множеству A алгоритмов заданной модели:

$$\mu(X^l) = arg \min_{a \in A} Q(a, X^l). \tag{2}$$

2.2 Определение и устройство нейронной сети

Перейдём непосредственно к нейронным сетям. Согласно [4], нейронная сеть — это громадный распределённый параллельный процессор, состоящий из элементарных единиц обработки информации, накапливающих экспериментальные знания и предоставляющих их для последующей обработки. Нейронную сеть можно сравнить с мозгом в двух аспектах:

- Знания поступают в нейронную сеть из окружающей среды и используются в процессе обучения;

- Чтобы знания накапливались, применяются синаптические веса - связи между нейронами.

Процедура, которая используется для процесса обучения, называется алгоритмом обучения.

Единицей обработки информации в нейронной сети является нейрон. Его модель показана на рисунке 1.

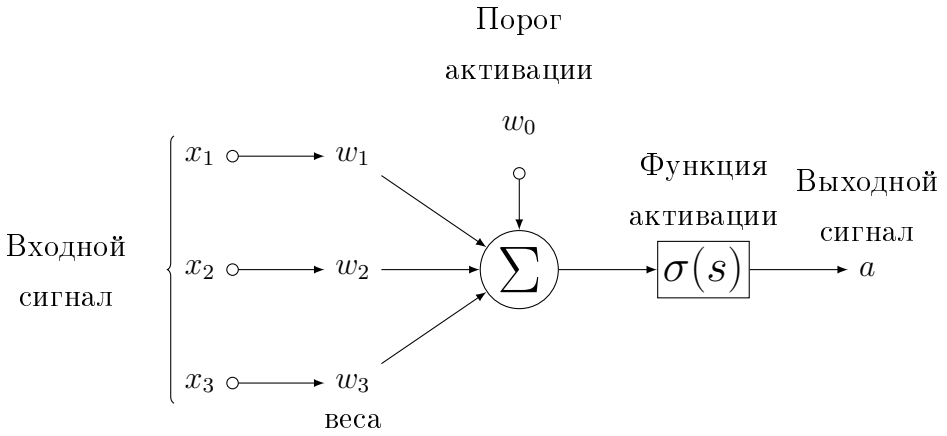

Рис. 1: Структура нейрона

Основными элементами нейрона являются:

1. Набор синапсов, характеризующихся весами. Если синапсу i, связанному с нейроном k на вход подаётся сигнал x_i, то сигнал умножается на вес w_{kj}.

2. Сумматор, который вычисляет линейную комбинацию сигналов путём сложения значений этих сигналов, умноженных на соответствующий вес синапса нейрона.

3. Функция активации, которая позволяет ограничить амплитуду выходного сигнала нейрона. Как правило, после нормировки диапазон амплитуд выхода нейрона находится в интервале [0,1] или [-1,1].

Также в модели нейрона присутствует пороговый элемент (или порог активации), который обозначается w_0. Эта величина отражает увеличение или уменьшение входного сигнала, подаваемого на функцию активации. Использование порога обеспечивает эффект афинного преобразования выхода линейного сумматора.

2.3 Модель МакКаллока-Питтса

Рассмотрим математическую модель нейрона, предложенную МакКаллоком и Питтсом. [8] Это алгоритм, который принимает на вход вектор описаний признаков $x =$

$(x^1, x^2, ..., x^n)$. Для простоты считаем, что признаки бинарные. Значения признаков - импульсы, поступающие на вход нейрона через синапсы. Эти импульсы складываются с весами $w_1, ..., w_n$, причём синапс называется возбуждающим, если соответствующий ему вес $w_i > 0$, и тормозящим, если $w_i < 0$. Если суммарный импульс превышает заданный порог активации w_0, то нейрон подаёт на выход 1, иначе он подаёт 0. Таким образом, нейрон занимается вычислением n-арной булевой функции вида

$$a(x) = \varphi(\sum_{j=1}^{n} w_j x_j - w_0), \tag{3}$$

где $\varphi(z) = [z \geq 0]$ - функция Хэвисайда. Функцию φ ещё называют функцией активации.

Также модель можно обобщить на случай произвольных входов и выходов и произвольной функции активации. Наиболее часто используются следующие функции $\varphi(z)$:

- Функция единичного скачка

$$\varphi(z) = \begin{cases} 1, & \text{если } z \geq 0 \\ 0, & \text{если } z < 0 \end{cases} \tag{4}$$

- Кусочно-линейная функция

$$\varphi(z) = \begin{cases} 1, & z \geq +\frac{1}{2} \\ |z|, & -\frac{1}{2} < z < +\frac{1}{2} \\ 0, & z \leq -\frac{1}{2}, \end{cases} \tag{5}$$

- Сигмоидальная функция

$$\varphi(z) = \frac{1}{1 + exp(-\alpha z)}, \tag{6}$$

  где α - параметр наклона.

2.4 Стохастический градиентный спуск

Исходя из принципа минимизации эмпирического риска, задачу о настройке весов можно свести к задаче минимизации функционала качества:

$$Q(w) = \sum_{i=1}^{l} \mathscr{L}(a(x_i), y_i) \to \min_w, \tag{7}$$

где $\mathscr{L}(a, y)$ - заданная функция потерь, которая характеризует величину ошибки ответа a при правильном ответе y. Воспользуемся методом градиентного спуска. Изменение весов на каждой итерации можно записать следующим образом:

$$w := w - \eta \frac{\partial Q}{\partial w}, \tag{8}$$

где $\eta > 0$ - величина шага в направлении антиградиента. В предположении, что $\mathscr{L}$ и φ дифференцируемы, распишем градиент:

$$w := w - \eta \sum_{i=1}^{l} \mathscr{L}'_a(a(x_i), y_i)\varphi'(\langle w, x_i \rangle)x_i, \tag{9}$$

Вектор весов w можно изменять после предъявления всех l объектов, каждый из которых вносит свой вклад в w, а можно обновлять вектор для каждого объекта. Второй способ называется методом стохастического градиента; при этом объекты перебираются в случайном порядке. Таким образом,

$$w := w - \eta \mathscr{L}'_a(a(x_i), y_i)\varphi'(\langle w, x_i \rangle)x_i, \tag{10}$$

Приведём алгоритм обучения методом стохастического градиента из [8].

Алгоритм 1: Обучение персептрона методом стохастического градиента

Вход : X^l - обучающая выборка;

η - темп обучения;

Выход: Синаптические веса $w_0, w_1, ..., w_n$;

1 инициализируем веса:

2 $w_j := random(-\frac{1}{2n}, \frac{1}{2n})$;

3 инициализируем текущую оценку функционала:

4 $Q := \sum_{i=1}^{l} \mathscr{L}(a(x_i), y_i)$;

5 **повторять**

6 выбрать объект x_i из X^l случайным образом

7 вычислить выходное значение алгоритма $a(x_i)$ и ошибку:

8 $\varepsilon_i := \mathscr{L}(a(x_i), y_i)$;

9 сделать шаг градиентного спуска:

10 $w := w - \eta \mathscr{L}'_a(a(x_i), y_i)\varphi'(\langle w, x_i \rangle)x_i$;

11 оценить значение функционала:

12 $Q := \frac{l-1}{l}Q + \frac{1}{l}\varepsilon_i^2$;

13 **до тех пор, пока** *значение Q не стабилизируется*;

2.5 Метод обратного распространения ошибок

На практике часто рассматриваются многослойные нейронные сети, которые, как понятно из названия, состоят из нескольких слоёв нейронов: одного входного, одного выходного и нескольких скрытых. Такие сети применяются для большого числа задач. При этом обучение происходит с помощью метода обратного распространения ошибки (error backpropagation algorithm). [4]

Приведём описание алгоритма из [8]. Рассмотрим полную многослойную сеть (т.е. такую сеть, в которой присутствуют все синаптические связи между нейронами предыдущего слоя и следующего слоя) и положим $X = R^n, Y = R^m$.

Пусть выходной слой состоит из M нейронов с функциями активации σ_m и выходами $a^m, m = 1, ..., M$. Перед ним находится скрытый слой из H нейронов с функциями активации σ_h и выходами $u^h, h = 1, ..., H$. Веса связей между двумя слоями будем обозначать w_{hm}. Перед скрытым слоем находится ещё один слой (распределительный или скрытый) с выходами $v^j, j = 1, ..., J$ и весами w_{jh}.

Выходные значения сети на объекте x_i вычисляются как суперпозиция:

$$a^m(x_i) = \sigma_m(\sum_{h=0}^{H} w_{hm}u^h(x_i)); \quad u^h(x_i) = \sigma_h(\sum_{j=0}^{J} w_{jh}v^j(x_i)). \tag{11}$$

Запишем функционал среднеквадратичной ошибки для отдельного объекта x_i:

$$Q(w) = \frac{1}{2}\sum_{m=1}^{M}(a^m(x_i) - y_i^m)^2. \tag{12}$$

Выпишем частные производные Q по выходам нейронов для выходного слоя:

$$\frac{\partial Q(w)}{\partial a^m} = a^m(x_i) - y_i^m = \varepsilon_i^m. \tag{13}$$

Теперь продифференцируем Q по выходам скрытого слоя:

$$\frac{\partial Q(w)}{\partial u^h} = \sum_{m=1}^{M}(a^m(x_i) - y_i^m)\sigma'_m w_{hm} = \sum \varepsilon_i^m \sigma'_m w_{hm} = \varepsilon_i^h. \tag{14}$$

Назовём эту величину ошибкой сети на скрытом слое.

Теперь можно выписать градиент Q по весам:

$$\frac{\partial Q(w)}{\partial w_{hm}} = \frac{\partial Q(w)}{\partial a^m}\frac{\partial a^m}{\partial w_{hm}} = \varepsilon_i^m \sigma'_m u^h, \quad m = 1, ..., M, \quad h = 0, ..., H; \tag{15}$$

$$\frac{\partial Q(w)}{\partial u^h} = \frac{\partial Q(w)}{\partial u^h}\frac{\partial u^h}{\partial w_{jh}} = \varepsilon_i^h \sigma'_h v^j, \quad h = 1, ..., H, \quad j = 0, ..., J; \tag{16}$$

и так далее для каждого слоя. Выпишем алгоритм полностью.

Алгоритм 2: Обучение двухслойной сети методом обратного распространения ошибки

Вход : $X^l = (x_i, y_i)_{i=1}^l$ - обучающая выборка, $x_i \in R^n, y_i \in R^M$;
H - число нейронов в скрытом слое;
η - темп обучения;

Выход: Синаптические веса w_{jh}, w_{hm};

инициализировать веса небольшими случайными значениями:
$w_{jh} := random(-\frac{1}{2n}, \frac{1}{2n})$;
$w_{hm} := random(-\frac{1}{2H}, \frac{1}{2H})$;
повторять
  выбрать объект x_i случайным образом
  прямой ход:
  $u_i^h := \sigma_h(\sum_{j=0}^{J} w_{jh} v^j(x_i))$, для всех $h = 1, ..., H$;
  $a_i^m := \sigma_m(\sum_{h=0}^{H} w_{hm} u^h(x_i))$, для всех $m = 1, ..., M$;
  $\varepsilon_i^m := a_i^m - y_i^m$, для всех $m = 1, ..., M$;
  $Q_i := \sum_{m=1}^{M} (\varepsilon_i^m)^2$
  обратный ход:
  $\varepsilon_i^h := \sum_{m=1}^{M} \varepsilon_i^m \sigma'_m w_{hm}$, для всех $h = 1, ..., H$;
  Градиентный шаг:
  $w_{hm} := w_{hm} - \eta \epsilon_i^m \sigma'_m u^h$, для всех $h = 0, ..., H, m = 1, ..., M$;
  $w_{jh} := w_{jh} - \eta \epsilon_i^h \sigma'_h x^j$, для всех $j = 0, ..., n, h = 1, ..., H$;
  $Q := \frac{l-1}{l} Q + \frac{1}{l} Q_i$;
до тех пор, пока Q *не стабилизируется*;

2.6 Обобщение и проблема переобучения

Одним из недостатков метода обратного распространения ошибки является то, что сеть способна переобучаться, если чрезмерно увеличивать веса. При таком методе обучения в сеть подаётся обучающая выборка, и с помощью алгоритма вычисляются синаптические веса многослойного персептрона. При этом мы надеемся, что полученная сеть способна к обобщению, т.е. для данных из контрольной выборки, которых она никогда ранее не видела, отображение входа на выход будет корректным. Предполагается, что контрольная выборка взята из той же популяции, что и обучающая.

Однако, если сеть обучается на слишком большом количестве примеров, она может "запомнить"примеры обучения. Например, из-за шума она может найти признаки, свойственные обучающей выборке, но не моделируемой функции. В этом случае говорят о переобучении сети. Такие сети теряют способность к обобщению на схожих входных

сигналах. [4]

Для улучшения качества и сходимости градиентного обучения, а также для борьбы с переобучением пользуются сокращением весов. [8] Идея заключается в добавлении к функционалу $Q(w)$ штрафного слагаемого с целью ограничения роста абсолютных значений весов:

$$Q_\tau(w) = Q(w) + \frac{\tau}{2}\|w\|^2. \tag{17}$$

Пересчитаем градиент:

$$\frac{\partial Q_\tau(w)}{\partial w} = \frac{\partial Q(w)}{\partial w} + \tau w. \tag{18}$$

Веса будут обновляться по формуле

$$w := w(1 - \eta\tau) - \eta\frac{\partial Q(w)}{\partial w} \tag{19}$$

3 Операционная система Linux

3.1 Историческая справка про операционные системы

Операционную систему можно назвать самой главной системной программой, т.к. в её обязанности входит управление всеми системными ресурсами и обеспечение основы для работы прикладных программ. [2] С точки зрения взаимодействия с пользователем операционная система предлагает абстракции, которые удобнее в обращении, чем то, что может предложить основное оборудование. Она предоставляет особый тип команд (системные вызовы), которые устраняют потребность пользователя в непосредственной работе с аппаратным обеспечением. С точки зрения взаимодействия с компьютером операционная система обеспечивает распределение ресурсов между программами.

На ранней стадии развития компьютеров программы писались непосредственно для аппаратуры на языках, понятных ей. Операционных систем ещё не существовало, и пользоваться всеми ресурсами вычислительной машины мог только один пользователь, запускающий одно приложение. Для облегчения жизни разработчикам в 1950-е годы были созданы первые операционные системы. Среди них были такие системы, как General Motors Operating System (GMOS), разработанная для IBM 701, и FORTRAN Monitor System (FMS), созданная North American Aviation для IBM 709. [9]

Операционная система Linux была разработана в 1991 г. Линусом Торвальдсом.

3.2 Устройство ядра Linux

Центральной частью операционной системы является ядро. При этом, несмотря на колоссальные размеры, оно имеет чёткую структурную организацию в виде подсистем и уровней. [9] Типичными компонентами ядра являются обработчики прерываний для обслуживания запросов на прерывание, планировщик для разделения процессорного

времени между несколькими процессами, система управления памятью для управления адресным пространством и системные службы, такие как сети и межпроцессное взаимодействие. [7]

На современных системах с защищёнными блоками управления памятью ядро обычно находится в более привилегированном состоянии по сравнению с остальными приложениями благодаря защищённому пространству памяти и полному доступу к аппаратным средствам. Это состояние системы и пространство памяти вместе называют пространством ядра (kernel-space). С другой стороны, пользовательские приложения выполняются в пользовательском пространстве (user-space). Они видят лишь часть доступных ресурсов машины и могут выполнять некоторые системные функции, напрямую работать с аппаратными средствами, получать доступ к памяти за пределами той, что выделена ядром. При исполнении кода ядра система находится в пространстве ядра и работает в режиме ядра. При исполнении обычного процесса система находится в пользовательском пространстве и работает в пользовательском режиме.

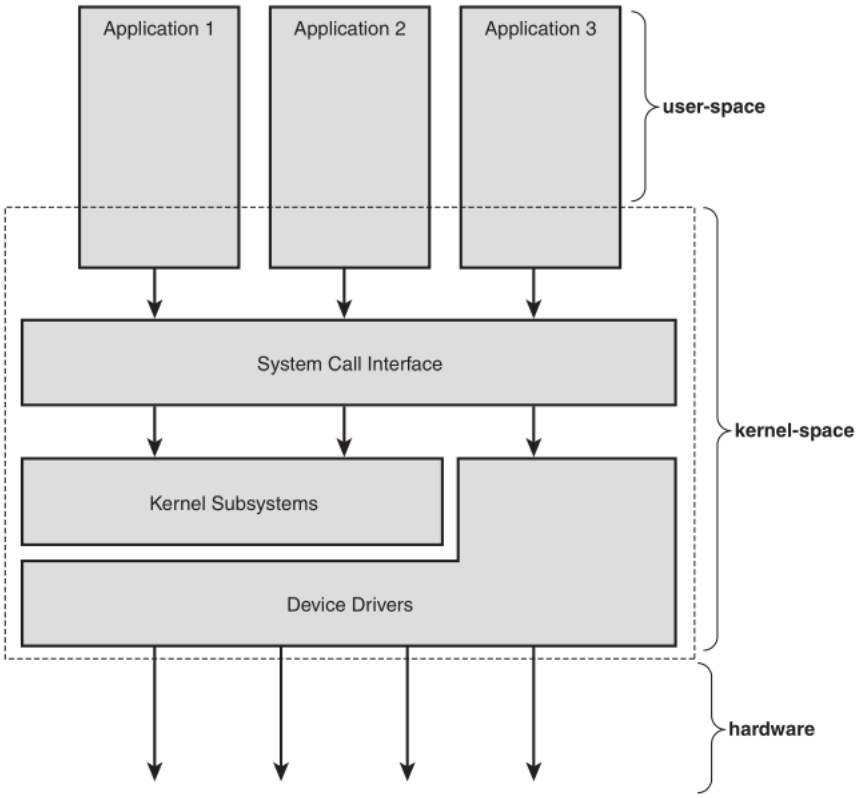

Рис. 2: Фундаментальная архитектура операционной системы Linux

Приложения, работающие в системе, взаимодействуют с ядром с помощью системных вызовов (см. рис. 2). Приложение обычно вызывает функции из библиотеки, которые, в свою очередь, опираются на интерфейс системных вызовов, чтобы поручить ядру выполнение задач от имени приложения. Некоторые библиотечные вызовы предоставляют множество функций, которых нет в системных вызовах, и, таким образом,

обращение к ядру может быть небольшой частью работы большой функции. Например, известная функция printf() предоставляет форматирование и буферизацию данных; вызов write() для записи данных в консоль является лишь одним из шагов её работы. С другой стороны, существуют библиотечные вызовы, которые помимо обращения к ядру почти ничего не делают. Например, библиотечная функция open() основном исполняет системный вызов open(). Тем не менее, такие функции библиотеки C, как strcpy(), вообще не должны напрямую обращаться к ядру. Когда приложение выполняет системный вызов, говорят, что ядро работает от имени приложения, приложение выполняет системный вызов в kernel-space, ядро работает в контексте процесса. Такое взаимодействие, когда приложения обращаются к ядру с помощью интерфейса системных вызовов, является основным принципом работы приложений.

4 Практическая часть

4.1 Описание модели

Для начала вкратце вспомним о таких ресурсах системы, как процессорное время, память и диск.

О работе приложений в user space и kernel space подробно говорилось в главе 3. Процессор работает в режиме пользователя, если он исполняет код в user space, и в режиме ядра, если он исполняет код в kernel space. В первом случае время работы процессора называется user time, во втором - system time.

Иерархию памяти часто изображают в виде пирамиды (рис. 3). [3] Если двигаться по ней сверху вниз, то будут увеличиваться время доступа к памяти и её объём, а также уменьшаться стоимость мегабайта памяти. На самом верху расположена память регистров. Доступ к ней осуществляется быстрее всего и составляет несколько наносекунд. Под регистрами находится кэш-память объёмом до нескольких мегабайт, время доступа к которой немного больше. Затем идёт основная память объёмом от одного до сотен гигабайт. Дальше идут магнитные диски, ленточные носители и оптические диски.

Большинство дисков представляют собой цилиндр: несколько пластин круглой формы располагаются друг под другом. Каждая поверхность снабжена кронштейном и головкой, которая может перемещаться на разное расстояние от оси цилиндра. Данные на диске содержатся в секторах, которые расположены на концентрических кругах, называемых зонами. Обычно размер данных в секторе составляет 512 байт.

Для считывания сектора или записи в него головка перемещается на нужное расстояние от оси, а диск вращается, пока требуемый сектор не окажется под головкой. Время поворота диска называют временем ожидания сектора.

Из-за особенностей устройства диска скорость передачи данных может сильно разниться. Она зависит от расположения секторов, к которым обращается головка. На-

Рис. 3: Иерархическая организация памяти

пример, считывание данных последовательно в одной зоне происходит быстрее, чем считывание произвольно разбросанных по пластинам секторов. Поэтому существуют различные методики измерения скорости чтения с диска и записи на него.

Теперь перейдём к нашей модели. Она строилась в базовом предположении, что с точки зрения выделения ресурсов ядро ведёт себя одинаково для любой программы. Для первого приближения мы решили рассматривать три основных ресурса системы: cpu (процессорное время), память и диск. Каждый из них характеризуется своими параметрами. Программе, потребляющей эти ресурсы, мы решили сопоставить три числа - доли потребления (в процентах) каждого ресурса. Для cpu это доля загруженности одного ядра, для памяти — процент от размера оперативной памяти (без учёта swap), для диска — процент от его пропускной способности при линейном чтении. Наше предположение относительно результатов заключается в том, что на небольших нагрузках ядро выделяет столько же ресурсов, сколько у него запрашивают. Для обнаружения связей между желаемой и получаемой нагрузками надо построить нейронную сеть. Возможны два варианта сети:

- Определение выделяемых ядром ресурсов по известным требуемым программой ресурсам.

- Определение ресурсов, которые программа запрашивает у ядра, по известным выделяемым ресурсам.

Мы рассматриваем второй вариант.

Для получения практических результатов нам нужно измерять ресурсы, потребляемые процессами. Для этого мы пользовались системными мониторами процессов top и atop.

4.2 Описание тестовых программ

Будем подавать сети на вход три числа (выделяемые системой cpu, память и диск), на выходе также будем получать три числа (требуемые cpu, память и диск). Для построения зависимостей между ресурсами необходимо обучить нейронную сеть. Нам нужно получить набор данных, показывающий соответствие между требуемыми и получаемыми нагрузками, т.е. нужно нагружать систему программами, у которых известны оба типа ресурсов. Для этого мы написали следующие программы:

1. Программа, потребляющая cpu. Итерация программы длится порядка $\frac{1}{10}$ секунды. В течение времени итерации программа вызывает функцию gettimeofday, пока доля времени работы не составит требуемое количество секунд. Оставшееся время итерации она спит.

2. Программа, потребляющая оперативную память. С помощью mmap она аллоцировала требуемую долю физической памяти (без учёта swap), а затем спала.

3. Программа, потребляющая диск. Она совершает линейное (последовательное) чтение файла. Для прямого чтения без кэширования использовался флаг O_DIRECT.

Далее программы комбинировались для изучения потребления сразу нескольких ресурсов. Все они запрашивали у ядра заранее заданную долю ресурсов системы. Мы запускали их по очереди, во время их работы смотрели на системные системные мониторы процессов и записывали выделяемые каждой программе ресурсы. Для исключения влияния других программ работа велась на "чистой"системе Ubuntu с остановленными сервисами и демонами и без дополнительно установленных программ. В результате была сформирована выборка из 70 элементов. 70% от неё пошло на обучающую выборку, 30% - на контрольную. Деление происходило случайным образом.

4.3 Описание нейронной сети и полученные результаты

После получения данных нам нужно построить и обучить нейронную сеть на наших данных. В качестве функции активации мы взяли линейную. Поскольку между ресурсами возможны сложные зависимости, была построена двухслойная сеть. Обучение производится методом обратного распространения ошибки. Начальный вектор весов задаётся случайным малым значением в диапазоне $(-\frac{1}{2n}, \frac{1}{2n})$, где n - количество связей. Коэффициент обучения равен 0.5. Для обучения использовались три вида сетей:

- Простая двухслойная сеть (три нейрона входного слоя связаны с каждым из трёх нейронов скрытого слоя; каждый нейрон скрытого слоя связан только с одним соответствующими ему нейроном выходного слоя);

- Полная двухслойная нейронная сеть (все связи между нейронами входного и скрытого, а также скрытого и выходного слоёв);

- Прореженная двухслойная нейронная сеть (часть связей из полной сети отсутствует, т.к. на основании наших предположений о предметной области соответствующие ресурсы не должны влиять друг на друга);

Эти модели обучились для прямой задачи, поэтому мы предполагаем, что для нашей задачи инвертированные модели также обучатся.

В результате обучения для нейронных сетей получились следующие веса (рис. 4–9). Модели показывают высокую точность на контрольных данных. Для оценки качества решения задачи использовался функционал суммарных потерь.

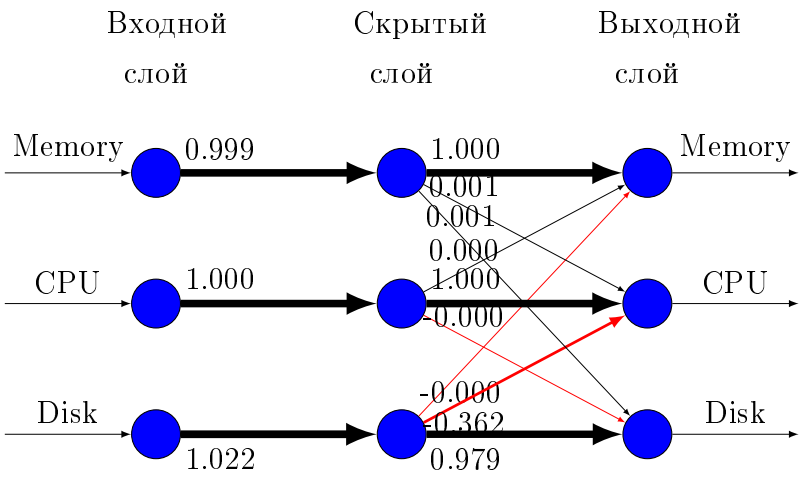

Рис. 4: Простая двухслойная нейронная сеть

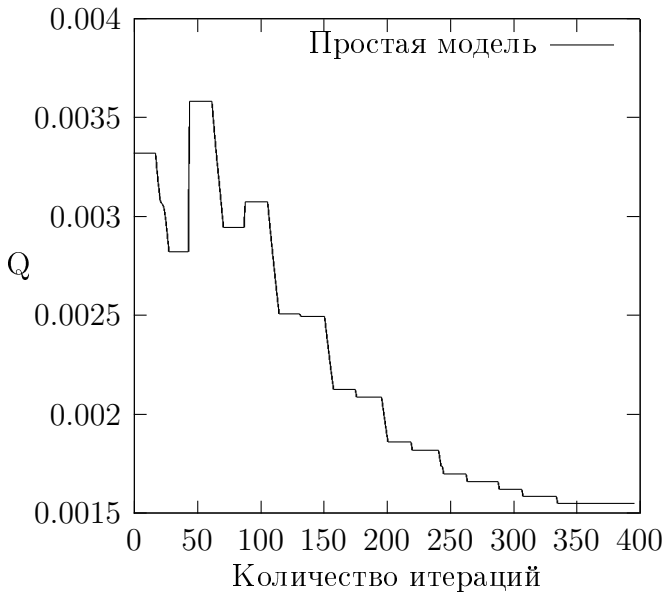

Рис. 5: Функция потерь для простой двухслойной нейронной сети

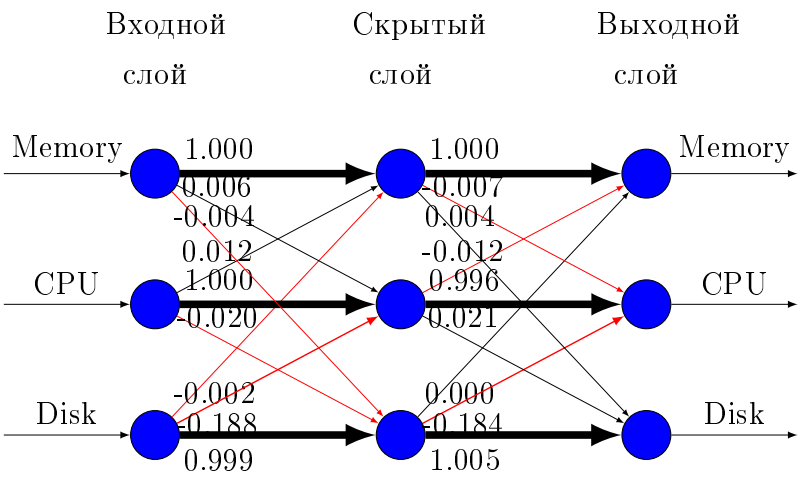

Рис. 6: Полная двухслойная нейронная сеть

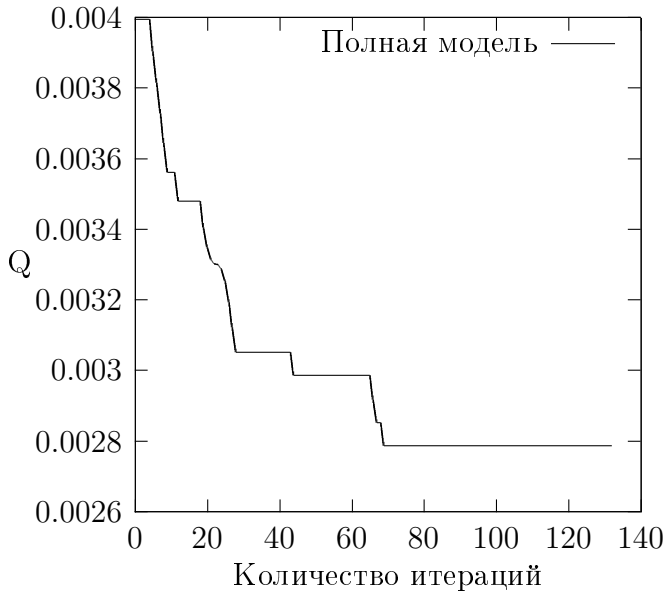

Рис. 7: Функция потерь для полной двухслойной нейронной сети

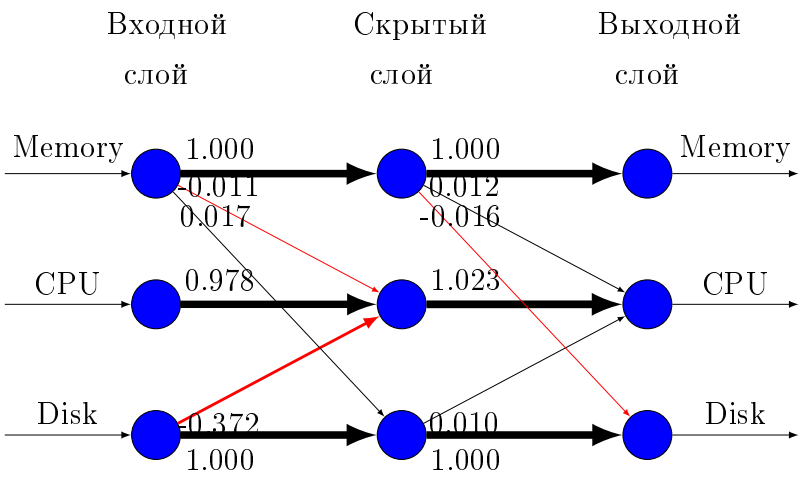

Рис. 8: Прореженная двухслойная нейронная сеть

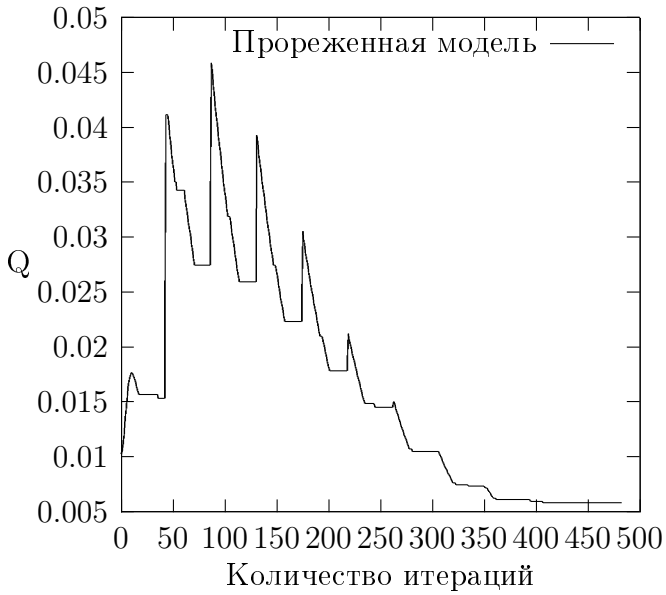

Рис. 9: Функция потерь для прореженной двухслойной нейронной сети

Также мы измерили время ответа нейронной сети на объекте. Оно оказалось примерно равным 10^{-6} с.

Таким образом, искусственная нейронная сеть способна решать поставленную задачу с высокой точностью.

5 Заключение

В ходе работы мы:

- написали тестовые программы, потребляющие заранее заданную долю ресурсов;

- обучили нейронные сети на выборке, составленной из показаний системных мониторов процессов для тестовых программ;
- получили нейронные сети, в первом приближении моделирующие выделение ресурсов ядром Linux.

Таким образом, цель работы достигнута.

В дальнейшем мы планируем расширение нейронной сети. Каждый из ресурсов характеризуется нескольким числом параметров. Например, процессорное время бывает user-time и system-time, есть различные виды памяти, и чтение с диска/запись на него можно выполнять разными способами. За счёт этих параметров можно увеличить количество входов, выходов и слоёв нейронной сети. Такая улучшенная сеть позволит выявлять больше типов связей, в том числе нелинейные, между ресурсами. Также планируются исследование выделяемых ресурсов в загруженной системе и посещение курса по устройству ядра Linux для более глубокого понимания процесса выделения ресурсов ядром.

Список литературы

[1] Бовет, Д. Ядро Linux, 3-е изд.: Пер. с англ. / Бовет Д., Чезати М. — СПб.: БХВ-Петербург, 2007. — 1104 с: ил.

[2] Таненбаум Э., Вудхалл А. Операционные системы. Разработка и реализация (+CD). Классика CS. 3-е изд. — СПб.: Питер, 2007. — 704 с: ил.

[3] Таненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. — СПб.: Питер, 2013. — 816 с.: ил.

[4] Хайкин, Саймон. Нейронные сети: полный курс, 2-е изд., испр. : Пер. с англ. — М. : 000 "И.Д. Вильяме 2006. — 1104 с. : ил. — Парал. тит. англ.

[5] Bishop C. M. Pattern Recognition and Machine Learning: Springer, 2006. - 749 c.

[6] Corbet J. Linux Device Drivers, Third Edition / Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman: O'Reilly Media, 2005. - 640 c.

[7] Love, R. Linux kernel development / Robert Love. — 3rd ed.:Addison-Wesley Professional, 2010 - 440 c.

[8] Воронцов К. В. Лекции по искусственным нейронным сетям
http://www.ccas.ru/voron/download/NeuralNets.pdf

[9] Тим Джонс М. Анатомия ядра Linux
https://www.ibm.com/developerworks/ru/library/l-linux-kernel/

[10] Nielsen M. Neural Networks and Deep Learning
http://neuralnetworksanddeeplearning.com/

[11] top(1) - Linux manual page
http://man7.org/linux/man-pages/man1/top.1.html

[12] atop(1) - Linux man page
http://linux.die.net/man/1/atop

[13] Drawing a Blank: Americans Have "No Clue" What Computer Memory Is
http://www.prnewswire.com/news-releases/drawing-a-blank-americans-have-no-clue-what-computer-memory-is-300134544.html

[14] Study Reveals Time Lost To Slow Computers May Be A Barrier To Better Health
http://crucial-com.pr.co/67884-study-reveals-time-lost-to-slow-computers-may-be-a-barrier-to-better-health

[15] Drawing a Blank: Americans Have "No Clue"What Computer Memory Is http://investors.micron.com/releasedetail.cfm?releaseid=929414

[16] Computer Loading Times Cost You Five-And-Half Days A Year http://www.huffingtonpost.co.uk/2013/10/08/computer-loading-times_n_4061470.html

[17] The time we lose to slow computers http://techdataukinfo.co.uk/microsites/public-sector/docs/sandisk/Infographic_time_we_lose.pdf