УДК 681.3

РАСПАРАЛЛЕЛИВАНИЕ АЛГОРИТМОВ ФУНКЦИОНИРОВАНИЯ КЛАССИФИКАТОРА СО СЛУЧАЙНЫМИ ПОДПРОСТРАНСТВАМИ

Д.В. Жора

Институт программных систем НАН Украины
03187, Киев, проспект Академика Глушкова, 40,
т.: 526 1538; факс: 526 6263, email: dvz73@bigfoot.com

Нейронные сети являются достаточно популярным средством решения многих задач искуственного интеллекта. В то же время, любые аппаратные реализации нейросетевых архитектур очень быстро устаревают благодаря стремительному развитию вычислительной техники. Таким образом, большинство исследователей стремятся использовать в первую очередь программные реализации алгоритмов, что делает актуальным распараллеливание функционирования нейросетевых систем. В работе рассматривается нейросетевой классификатор со случайными подпространствами и предлагаются алгоритмы распараллеливания его основных операций.

Neural networks are quite popular for solving many tasks of artificial intelligence. At the same time, any hardware implementation of some neural architecture becomes obsolete really fast due to rapid development of semiconductor industry. Thus, most researchers first of all tend to use software algorithm implementations, which makes parallelization of neural network algorithms quite attractive. This article analyses neural random subspace classifier and suggests algorithms for parallelization off it's basic operations.

Использование различных подходов для распараллеливания вычислительных процессов

Большинство современных компъютерных систем используют принципы последовательной обработки информации и имплементируют парадигму машины Тьюринга, которая была разработана в начале 20-го века. Данная парадигма допускает наличие только одного процессора, таким образом, обработка большого количества информации осуществляется последовательно такт за тактом. Распределение вычислительной нагрузки во времени приводит к тому, что даже на современных быстродействующих вычислительных устройствах решения серьезных задач требует существенных временных ресурсов.

Значительная часть усилий специалистов в области информационных технологий сейчас сосредоточена на распределении вычислительной нагрузки между несколькими вычислительными устройствами. Самый высокий уровень распараллеливания алгоритмов обработки информации предполагает разделение исходной вычислительной задачи на множество подзадач и распределение полученных подзадач между различными компьютерами (узлами), объединенными между собой в вычислительную сеть. Преимущество данного подхода заключается в использовании вычислительных ресурсов компьютеров, которые в обычной ситуации не используются. Основным недостатком является отсутствие общей физической памяти для большинства процессов распределенной программы.

Промежуточный вариант использования принципа распараллеливания вычислений заключается в создании многопроцессорных компьютерных систем. К преимуществам данного подхода следует отнести возможность использования всего набора операций, которые предусмотрены архитектурой современных процессорных устройств, возможность использования общей памяти и упрощение коммуникаций между процессами. Одним из недостатков данной архитектуры является необходимость корректной синхронизации действий разных процессов и потоков вычислений. Для решения вопросов синхронизации современная теория предлагает следующие механизмы: использование координационных моделей и языков, разделение исходной задачи на непересекающиеся подзадачи, применение различных инструментов синхронизации, таких как очереди, стеки, события, критические сессии, мютексы, семафоры и т.д.

Нижний уровень распараллеливания вычислительных систем представлен архитектурными решениями, где разделение операций производится на уровне элементарных логических и математических операций. Обычно такие системы имеют исключительно параллельную организацию, причем параллельность архитектуры постулируется априори. Полный цикл функционирования системы, или решение некоторой прикладной задачи может проводиться всего за несколько тактов. Одним из классов таких систем являются нейронные сети, которые имеют множество аналогий с функционированием нервной системы человека.

Алгоритмические преимущества нейросетевых систем заключаются в том, что использование данной технологии не обязательно предполагает знание некоторой математической модели прикладной области или правил, которые необходимо выполнять для соблюдения консистентности с теорией. То есть, нейросетевой подход использует принципы индуктивного моделирования и не предполагает наличия возможности

дедуктивного вывода правил для области приложения. Основной способ использования нейронной сети предполагает обучение сети на так называемой обучающей выборке и последующее применение полученной конфигурации сети на тестовой выборке.

Нейросетевые технологии на данный момент применяются для решения следующих прикладных задач: распознавание речи, распознавание лиц, преобразование формы представления документов из сканированной в символьную, прогнозирование загруженности электрических и телекоммуникационных сетей, прогноз погоды, прогнозирование наводнений и других стихийных бедствий, создание имплантантов зрения или слуха, обработка аудио, видео и другой мультимедийной информации, прогнозирования динамики финансовых рынков, моделирование процессов мышления и т.д. Очевидно, большинство из перечисленных задач не поддается полной математической формализации, так как количество факторов, которые влияют на поведение динамической системы, очень большое, кроме того, многие факторы не являются известными.

Нейросетевые алгоритмы могут быть применены для решения достаточно широкого круга проблем искусственного интеллекта, таких как классификация, кластеризация, регрессия и интерполяция, ассоциативный поиск и т.д. Большинство специалистов в данной области полагают, что задачи классификации или распознавания образов являются наиболее распространенными и их решение является существенным для получения многих научных и экономических результатов.

Подробный анализ нейросетевых архитектур приведен, например, в монографии [1]. Достаточно полный и авторитетный обзор методов распознавания образов представлен в [2]. Следует отметить возможность успешного решения задач аппроксимации функций с использованием нейронных сетей, которые являются универсальными аппроксиматорами [1–4]. Для нейронных сетей, используемых для решения задач классификации, как правило, важно, чтобы данная сеть являлась универсальным классификатором.

Для ассоциативного поиска информации, как правило, используются сети на основе сети Хопфилда или ассоциативно-проективные нейронные структуры [5–6]. Для решения задач регрессии или аппроксимации функций часто используются следующие нейронные сети прямого распространения: многослойный персептрон, сеть на основе радиально-базисных функций, вариации перечисленных архитектур с использованием других базисных функций. Многие исследователи отмечают существенную особенность нейронных сетей, которая состоит в том, что нейронные сети являются нелинейными преобразователями. Последнее обуславливает возможность решения сложных задач распознавания образов и нетривиальное поведение сети как динамической системы. В частности, однослойный персептрон является линейным дискриминатором и его когнитивные возможности ограничены. Двухслойный персептрон является универсальным аппроксиматором в классе непрерывных функций [3]. Персептрон, который имеет три или более слоев может аппроксимировать более сложные и разрывные функции, однако данная архитектура является более сложной для обучения. Сеть на основе радиально-базисных функций является универсальным аппроксиматором в классе непрерывных функций. Эти же системы можно успешно использовать для решения задач классификации или распознавания образов.

Наиболее популярной идеей решения сложных задач классификации и регрессии является использование нелинейного преобразования исходного пространства, размерность которого обычно мала, в пространство характеристик с существенно большей размерностью. При этом вероятность того, что полученная задача окажется линейно разрешимой в новом пространстве характеристик, становится существенно большей. Этот же принцип используется в методе опорных векторов и теории статистического обучения [7]. Фактически, увеличение размерности пространства позволяет увеличить число степеней свободы рассматриваемой системы индуктивного моделирования по отношению к количеству ограничений, накладываемых обучающей выборкой.

В работах [8–11] рассматривается классификатор со случайными подпространствами, который является высокопроизводительным нейросетевым классификатором и может быть эффективно реализован как программно так и аппаратно. Классификатор использует вычислительно эффективную схему грубого кодирования для формирования многомерного бинарного представления входных вещественнозначных векторов. Сравнительный анализ бинарных схем грубого кодирования представлен в [9]. Большинство нейросетевых парадигм также могут рассматриваться как методы грубого кодирования, в частности, следует отметить использование радиально-базисных функций.

Современное развитие вычислительной техники является достаточно стремительным. В то же время, данный классификатор может быть легко адаптирован как к системам с ограниченными вычислительными ресурсами, так и к современным мощным компьютерам. Это может быть достигнуто благодаря регулированию количества нейронов скрытого слоя и оптимизации различных параметров структуры классификатора. Математический анализ модели нейронной сети, проведенный в [8], позволяет это сделать с учетом конкретного распределения входных данных. Сравнение данной модели с другими классификационными алгоритмами позволяет сделать вывод, что классификатор со случайными подпространствами является достаточно конкурентоспособным, и может быть применен для решения широкого круга практических задач.

В 80-х годах прошлого века достаточно популярной была аппаратная реализация нейросетевых архитектур. В то же время, развитие полупроводниковых технологий было столь стремительным, что со временем, даже сторонники такого подхода стремились использовать программные реализации для исследовательских целей и большинства прикладных задач. Экономически гораздо более целесообразно использовать параллельную программную реализацию для многопроцессорной компьютерной системы, так как некоторая фиксированная аппаратная реализация требует специальной разработки. В частности, высокое

быстродействие может понадобиться для следующих задач: краткосрочное внутридневное прогнозирование финансовых рынков, распознавание образов в реальном времени.

Описание классификатора со случайными подпространствами

Классификатор предназначен для классификации точек, обычно рассматриваемых в n-мерном евклидовом пространстве. Для удобства будем считать областью классификации n-мерный единичный куб. В частности, любая задача классификации может быть сведена к классификации внутри единичного гиперкуба с помощью, например, линейного преобразования. Классификатор, схема которого показана далее на рис. 1, состоит из четырех слоев нейронов. К первому слою относятся нейроны $\mathbf{l}$ и $\mathbf{h}$, последущие три слоя представлены нейронами $\mathbf{A}$, $\mathbf{B}$ и $\mathbf{C}$. Первые три слоя производят нелинейное преобразование входного вещественного вектора в некоторый двоичный вектор большой размерности. Последние два слоя нейронов $\mathbf{B}$ и $\mathbf{C}$ представляют собой обычный персептрон с обучаемой матрицей связей $\mathbf{W}$.

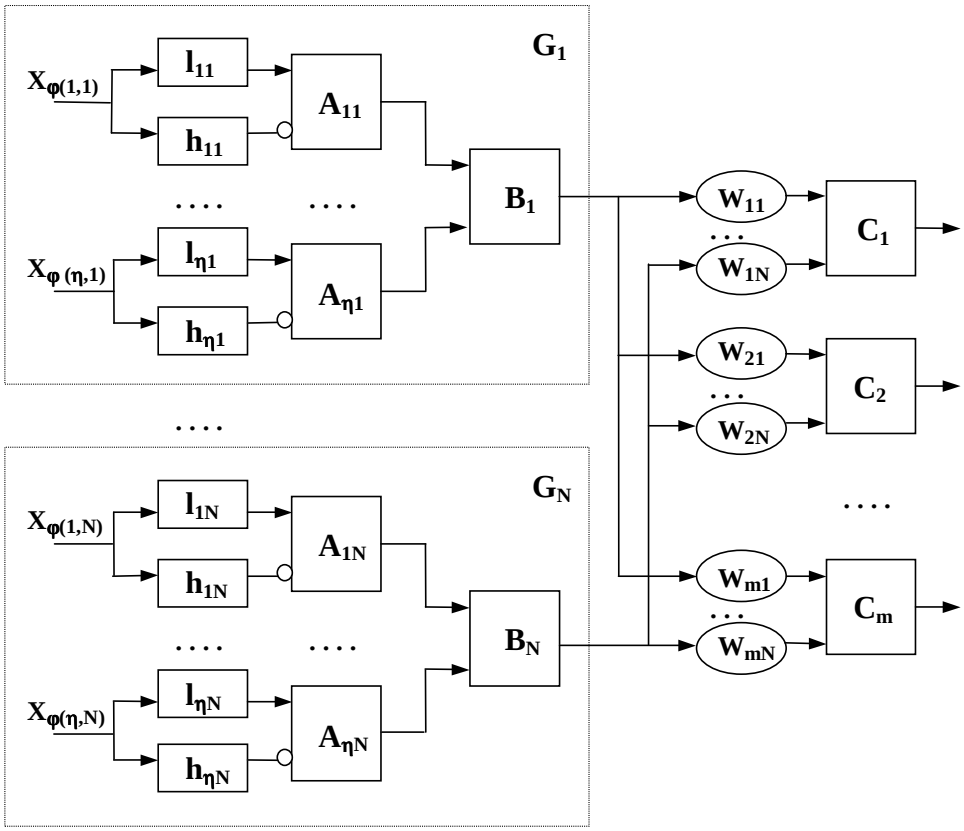


Рис. 1. Схема классификатора со случайными подпространствами

⟶ – возбуждающее соединение ⟶○ – тормозящее

Обозначим количество нейронов слоя $\mathbf{B}$ как N. Каждому нейрону этого слоя соответствует рецептивная группа нейронов (receptive group, box), предшествующих ему по связям. Каждая такая группа использует в качестве входов некоторые η различных компонент входного вектора, здесь $1 \le \eta \le n$. Выбор компонент задает индекс $\varphi(i, j)$, где функция φ принимает значения от 1 до n. Здесь i – номер используемой компоненты внутри группы, изменяющийся от 1 до η, а j – номер группы, изменяющийся от 1 до N.

Пусть на вход классификатора подан некоторый вектор $\mathbf{x} = (x_1, \ldots, x_n)$. Каждая компонента входного вектора с номером $\varphi(i, j)$ подается на два пороговых нейрона $\mathbf{l}_{ij}$ и $\mathbf{h}_{ij}$. Выходной сигнал нейрона $\mathbf{l}_{ij}$ принимает единичное значение, если подаваемый на него сигнал превышает пороговое значение l_{ij}. В остальных случаях выход нейрона $\mathbf{l}_{ij}$ равен нулю. Аналогично, выходной сигнал нейрона $\mathbf{h}_{ij}$ принимает единичное значение только при превышении входным сигналом значения h_{ij}. Далее, выходной сигнал нейрона $\mathbf{A}_{ij}$ принимает единичное значение когда $\mathbf{l}_{ij}$ возбужден и $\mathbf{h}_{ij}$ выдает нулевой сигнал, во всех остальных случаях выход нейрона $\mathbf{A}_{ij}$ принимает нулевое значение. Другими словами, нейрон $\mathbf{A}_{ij}$ возбуждается при соблюдении неравенства $l_{ij} < x_{\varphi(i,j)} < h_{ij}$. Нейроны слоя $\mathbf{B}$ также могут иметь на выходе только значения

равные нулю или единице. Нейрон $\mathbf{B}_j$ возбужден только в случае, когда возбуждены все нейроны $\mathbf{A}_{ij}$, от которых он получает сигнал, т.е. при выполнении условия $\forall i \in \overline{1,\eta}:\ l_{ij} < x_{\varphi(i,j)} < h_{ij}$.

Фактически это означает, что нейрон $\mathbf{B}_j$ реагирует на попадание подаваемой на вход классификатора точки в рецептивное поле – область пространства, ограниченную гиперплоскостями $x_{\varphi(i,j)} = l_{ij}$ и $x_{\varphi(i,j)} = h_{ij}$, где $i = \overline{1,\eta}$.

После того, как определено возбуждение нейронов слоя $\mathbf{B}$, производится вычисление входных напряжений (постсинаптических потенциалов) для нейронов слоя $\mathbf{C}$ по формуле $u_k = \sum_{j=1}^{N} W_{kj} B_j$. Синаптические веса W_{kj}, и следовательно напряжения u_k, принимают целочисленные значения. После этого возбуждается только тот нейрон слоя $\mathbf{C}$, который имеет максимальное входное напряжение. Номер возбужденного нейрона слоя $\mathbf{C}$ и есть номер класса, к которому классификатор относит подаваемый входной вектор.

Перед обучением классификатора необходимо произвести генерацию структуры классификатора, которая включает индекс подпространств $\varphi(i, j)$ и пороговые значения нейронов первого слоя l_{ij} и h_{ij}. Описываемая здесь модель называется классификатором со случайными подпространствами (random subspace classifier, RSC) благодаря случайному выбору индекса $\varphi(i, j)$. Изначально классификатор со случайными порогами (random threshold classifier, RTC) был предложен как частный случай последней при $\eta = n$ и $\varphi(i, j) = i$. Обобщение модели было произведено для того, чтобы при увеличении размерности классифицируемого пространства вероятность возбуждения нейрона слоя $\mathbf{B}$ не была слишком малой.

Каждое значение $\varphi(i, j)$ – случайная величина, принимающая с равной вероятностью значения $\overline{1,n}$. Накладывается только одно ограничение, что значения $\varphi(i, j)$ должны быть различны в пределах одной рецептивной группы, т.е. при фиксированном j. Такой выбор индексов полезен на практике и легко программируется. Если выполняется условие $\eta << n$, то индексы подпространств можно считать независимыми величинами.

Структура классификатора также зависит от определяемых заранее двух параметров δ_1 и δ_2, удовлетворяющих условиям $0 < \delta_2$, $0 \le \delta_1 \le \delta_2$. Значения порогов вычисляются по формулам $l_{ij} = \xi_{ij} - \zeta_{ij}$ и $h_{ij} = \xi_{ij} + \zeta_{ij}$, где центр ξ_{ij} – случайная величина, равномерно распределенная на отрезке $[-\delta_2, 1+\delta_2]$, а полуширина ζ_{ij} – случайная величина, равномерно распределенная на $[\delta_1, \delta_2]$. Все величины ξ_{ij} и ζ_{ij} независимы, что обусловливает независимость случайных величин l_{ij}, а также h_{ij}. Величины l_{ij} и h_{ij} с одинаковыми индексами i и j, понятно, зависимы. Таким образом, структура классификатора зависит от параметров η, N, δ_1 и δ_2. Вообще, можно рассмотреть и другие распределения для ζ_{ij} на отрезке $[0, \delta_2]$. Представляет интерес сравнение различных способов вычисления пороговых значений.

Предложенный способ генерации пороговых значений обеспечивает для всех точек отрезка $[0,1]$ равновероятное попадание в интервал (l_{ij}, h_{ij}). Следовательно, чувствительность классификатора одинакова для всех точек единичного гиперкуба. Кроме того, попадание пороговых значений вне отрезка $[0,1]$ обусловливает возможность классификации точек, не принадлежащих единичному гиперкубу, что может оказаться полезным при использовании классификатора на практике.

Изначально все синаптические веса W_{ij} имеют нулевое значение. Для каждого вектора из обучающей выборки определяется класс, к которому классификатор относит данный вектор. Модификация весов W_{ij} производится при ошибочной классификации. Пусть t – истинный номер класса, а f – номер класса, определенный классификатором. Тогда для всех значений индекса $j = \overline{1,N}$ веса пересчитываются по формулам $W'_{tj} = W_{tj} + B_j$ и $W'_{fj} = \max(W_{fj} - B_j, 0)$. Иногда используется правило с наказанием, когда $W'_{fj} = W_{fj} - B_j$. Как показывает опыт, второе правило обеспечивает более быструю сходимость обучения, но при этом качество распознавания становится хуже. Цикл обучения обычно повторяется до сходимости, либо заданное число раз. В любом случае результат обучения (множество значений весов синаптической матрицы) зависит от порядка предъявления векторов.

Основные свойства классификатора со случайными подпространствами

Первые эксперименты по сравнению базовой модели классификатора с такими известными алгоритмами, как метод потенциальных функций, ближайшего соседа и обратного распространения ошибки, приведены в [11]. Для большинства искусственно сгенерированных задач классификатор со случайными подространствами превосходит остальные классификаторы по таким критериям, как время обучения, время распознавания и количество ошибок. Причем сравнительные характеристики классификатора улучшаются при увеличении количества точек обучающего набора и размерности входного пространства.

Следует выделить следующие характеристики, определяющие качество функционирования сети: качество распознавания, скорость распознавания и скорость обучения. Также существенной характеристикой является объем памяти, необходимый для хранения порогов, индексов подпространств и синаптической матрицы. При программной реализации скорость распознавания обратно пропорциональна количеству групп нейронов N. При аппаратной реализации распознавание производится за несколько тактов, а количество групп нейронов влияет на затраты при исполнении классификатора. Размерность слоя **B** целесообразно выбирать как можно больше, при этом не превышая требований по скорости распознавания, выдвигаемых прикладной задачей. Скорость обучения классификатора непосредственно зависит от скорости распознавания. Кроме того, время обучения определяется длиной и структурой обучающей выборки, т.е. количеством проходов цикла обучения (эпох), необходимых для сходимости. Качество распознавания, вообще, может оцениваться только с точки зрения потребностей прикладной задачи.

В работе [8], для классификатора со случайными подпространствами получены аналитические зависимости, которые позволяют сравнивать чувствительность классификатора с другими известными моделями. В частности, получена функциональная зависимость расстояния Хемминга между двоичными образами двух разделяемых вещественных векторов на предпоследнем слое классификатора. Также дана оценка минимального различимого расстояния. Исследования, проведенные в [9], показывают, что рассматриваемая схема грубого кодирования, как правило, обеспечивает наименьшее минимальное различимое расстояние. Аналитически получено, что оптимальное расстояние между порогами для соответствующих пар нейронов первого слоя должна составлять не менее единицы. Кроме того, использование переменной ширины накрывающего отрезка обычно не дает улучшения качества работы классификатора. Результаты экспериментов показывают, что оптимальное значение параметра подпространств η обычно составляет от 3 до 5. При большем значении размерности подпространства каждой рецептивной группы вероятность возбуждения нейронов слоя **B** становится очень малой, что обуславливает плохую обобщающую способность сети. Таким образом, во многих случаях выбор параметров структуры классификатора можно произвести без эксперимента. Такая возможность необходима в случае если распределение для выборки входных векторов неизвестно, сложно или не принимается к рассмотрению. Кроме того, в [8] приведен способ преобразования входного пространства для адаптации сети к вероятностному распределению входных данных. При использовании последнего метода скорость обучения классификатора обычно увеличивается в несколько раз.

Как правило, классификатор со случайными подпространствами интерпретирует обучащую выборку без ошибок. Такие сети обычно рассматриваются как переученные (overfitted). В то же время, именно распределение входных данных, а не степень соответствия обучающей выборке, определяет полезность дальнейшего обучения и возможность получения большей вероятности успеха на тестовой выборке. Классификатор со случайными подпространствами является универсальным классификатором, т.е. имеет возможность безошибочной интерпретации произвольной обучающей выборки. Таким образом, данное свойство обеспечивает, что данный классификатор способен обеспечивать тонкую, сложную и многомерную структуру входного пространства. Другими словами, произвольно сложная непротиворечивая обучающая выборка может быть проинтерпретирована классификатором без ошибок. В частности, под сложностью следует понимать длину обучающей выборки, а также пространственную конфигурацию представителей соответствующих классов.

С одной точки зрения, использование классификатора аналогично применению метода опорных векторов [7]. В этом случае оба подхода используют нелинейное отображение входного вектора в многомерное пространство характеристик. Если исходная задача классификации не является линейно разделимой в исходном пространстве, то вероятность того, что задача станет линейно разделимой в пространстве характеристик увеличивается при увеличении его размерности. В отличие от метода опорных векторов данный классификатор специфицирует тип нелинейного преобразования, но не использует оптимизационного подхода для формирования “хорошей” линейной разделяющей поверхности. Такое решение обеспечивает достаточно высокую вычислительную эффективность. В то же время, классификатор со случайными подпространствами может имплементировать решающее правило, полученное с использованием метода опорных векторов.

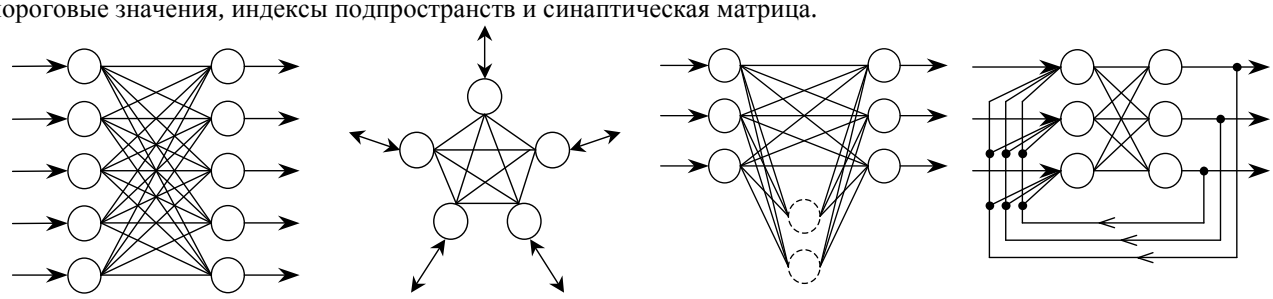
С другой стороны, классификатор со случайными подпространствами аналогичен нейронной сети на основе радиально-базисных функций [1]. Различие состоит в том, что все операции являются дискретными, а форма функции активации нейронов скрытого слоя не является радиальной. Так как архитектура классификатора использует элементы наиболее популярных и современных нейросетевых решений, эффективность данной модели должна быть достаточно высокой при решении реальных прикладных задач.

Успешность использования той или иной модели классификатора, как правило, обусловлена конкретной прикладной задачей. Многие авторы подчеркивают что не существует классификационной модели,

которая заведомо превосходила бы все остальные. Классификатор со случайными подпространствами показывает достаточно хорошие результаты для задач, где области принадлежности классов имеют четкие границы. В частности, при тестировании классификатора на базе данных ELENA [12–13], именно на таких задачах были получены хорошие результаты по сравнению с другими алгоритмами классификации [12]. Если плотности распределения классов пересекаются, классификатор все равно способен интерпретировать обучающую выборку без ошибок, благодаря свойству универсальности. Однако, в этом случае полученное решение не является оптимальным и существенно отличается от байесовского для искусственных задач с априорно известными плотностями распределения. Лучшая обобщающая способность может быть получена при использовании алгоритма локального усреднения синаптической матрицы. Данный метод усредняет синаптические веса, соответствующие соседним рецептивным полям и имеет некоторую аналогию с пространственной фильтрацией. Параметры алгоритма усреднения позволяют варьировать степень соответствия обучающей выборке.

Распараллеливание алгоритмов функционирования классификатора

По аналогии с биологическими системами нейросетевые архитектуры имеют, как правило, параллельную организацию. Этот факт существенно упрощает создание параллельных алгоритмов функционирования нейронных сетей для использования на многопроцессорных компьютерах. Далее на рис. 2 показаны наиболее популярные и используемые архитектурные решения. Условно, данные схемы соединения нейронов можно разделить на легко- и трудно-параллелизируемые. Соответственно первые два типа сетей относятся к первой категории, а последние два – ко второй. В частности, классификатор со случайными подпространствами можно отнести к многослойным сетям прямого распространения. Большинство нейросетевых архитектур использует схемы соединения “каждый с каждым”, таким образом, возможность использования общей памяти является существенной для обеспечения оперативного доступа к требуемой информации. Данному условию вполне удовлетворяют многопроцессорные компьютерные системы. В рассматриваемой модели классификатора существенный объем памяти занимает следующая информация: пороговые значения, индексы подпространств и синаптическая матрица.



Многослойные сети прямого распространения | Автоассоциативные полносвязные сети | Каскадно-корреляционные сети | Многослойные реккурентные сети

Рис. 2. Наиболее используемые нейросетевые архитектуры, стрелками показаны входные и выходные сигналы сети

Для классификатора со случайными подпространствами следующие “макро”-операции являются основными: генерация структуры классификатора, обучение сети, распознавание или использование сети в режиме экзамена. Следует выделить следующие базовые операции классификатора, имплементация которых обеспечит вычислительно сложную работу сети: *генерация структуры классификатора*, *вычисление выходных сигналов рецептивных групп*, *вычисление постсинаптических потенциалов нейронов последнего слоя*, *модификация синаптической матрицы*. Именно для этих задач целесообразно построить методы распараллеливания.

При последовательной обработке информации основная вычислительная сложность состоит в необходимости пересчета выходных сигналов для достаточно большого количества нейронов. В частности, для многих практических приложений количество рецептивных групп классификатора N составляет десятки тысяч. С другой стороны, при малом значении параметра N распараллеливание может не являться целесообразным, так как вычислительная сложность обработки малого подмножества рецептивных групп будет сравнима с затратами, необходимыми для создания нового рабочего потока. Пусть используемая многопроцессорная вычислительная система содержит P процессоров. Основная идея распараллеливания базовых алгоритмов последовательной программной реализации классификатора со случайными подпространствами состоит в разделении множества всех рецептивных групп на P непересекающихся подмножеств или блоков. Для удобства будем считать, что N кратно P.

Программирование параллельного алгоритма достаточно описать в терминах многопотоковой программы, так как задача распределения вычислительных потоков по процессорам выполняется операционной системой. Необходимость непосредственного доступа к общим массивам памяти обуславливает, что

параллельная программа должна быть реализована как один многопотоковый процесс. Далее в работе будут использоваться инструменты среды программирования Visual C++ 6.0. Изложение основных принципов разработки многопотоковых приложений, использование функций Win32 API и системы объектов MFC приведено, например, в [14]. Наиболее авторитетным описанием языка программирования C++ является [15].

Использование нескольких процессов не является эффективным, в частности, по следующим причинам: в современных процессорах компании Intel в защищенном режиме аппаратно запрещен доступ одного процесса в область памяти другого, средства коммуникации между процессами сложны в использовании и ресурсоемки для больших массивов памяти. В терминах операционной системы, например, Windows 2000, адресное пространство процесса вообще является виртуальным и может быть представлено несколькими фрагментами физического адресного пространства с нарушением принципа линейной адресации. Аналогично, мало целесообразным является использование распределенных систем вычислений, таких как MPI, PVM или GRID-платформ. Необходимость передачи значительного объема данных между узлами кластера обуславливает временные задержки и значительно усложняет исходный алгоритм функционирования нейронной сети. В то же время, возможно успешное применение кластеров для моделирования, например, модульных нейронных сетей [16], либо для задач, которые допускают разделение на подзадачи на более высоком концептуальном уровне.

Далее представлен шаблон кода C++ для генерации структуры классификатора. Сравнительная простота распараллеливания достигается благодаря полной независимости вычислений, производимых для каждой j-й группы нейронов. Пусть $\alpha = \overline{1,P}$ – индекс блока структуры сети, $N_P = N/P$ – количество групп внутри одного блока. Для каждого блока, который соответствует индексам групп j, изменяющимся от $(\alpha-1)\cdot N_P+1$ до $\alpha \cdot N_P$, запускается отдельный рабочий поток. Функция **`CreateThread`** принимает в том числе следующие аргументы: указатель на функцию, с которой должно начинаться исполнение потока, указатель на структуру данных, которая содержит информацию для конкретного экземпляра рабочего потока. В частности, данная структура содержит начальный и конечный индексы рецептивных групп соответствующего блока структуры классификатора. После того, как все требуемые потоки запущены, для продолжения дальнейших вычислений вызывающий поток исходной программы обязан ждать завершения каждого из них. Это достигается с использованием функции **`WaitForMultipleObjects`**, которая ожидает пока все синхронизационные объекты полученного массива не окажутся в сигнальном состоянии. В частности, рабочий поток переходит в сигнальное состояние после его завершения.

```
struct StructureOptions {
    /* ... */
};
struct StructureInfo {
    int m_iStart; // starting index of the structure block
    int m_iFinish; // starting index of the next block
    StructureOptions m_options; // structure parameters
};
/* ... */
int iNumberOfGroups,iNumberOfProcessors;
// function that generates structure
// for one particular receptive group
void StructureForGroup(int,StructureOptions&);
// function declaration in order to use
// the thread entry function pointer
DWORD WINAPI StructureBlockThread(LPVOID);
/* ... */
// assume the variables iNumberOfGroups and
// iNumberOfProcessors are initialized
int iBlockSize=iNumberOfGroups/iNumberOfProcessors;
void GenerateStructure(StructureOptions& options) {
    /* ... */
    StructureInfo* pInfo=new StructureInfo[iNumberOfProcessors];
    HANDLE* pThreadHandle=new HANDLE[iNumberOfProcessors];
    for (int i=0;i<iNumberOfProcessors;i++) {
        /* copy required option attributes */
        pInfo[i].m_iStart=i*iBlockSize;
        pInfo[i].m_iFinish=(i+1)*iBlockSize;
        /* ... */
        DWORD dwThreadID;
```

```
            // start worker thread for each structure block
            pThreadHandle[i]=::CreateThread(NULL,0,
                StructureBlockThread,&(pInfo[i]),0,&dwThreadID);
            ASSERT(pThreadHandle[i]!=NULL);
        }
        // before current thread can continue it's
        // needed to wait for each worker thread to exit
        ::WaitForMultipleObjects(iNumberOfProcessors,
            pThreadHandle,TRUE,INFINITE);
        for (i=0;i<iNumberOfProcessors;i++) {
            // close thread handle so that operating
            // system can destroy the object
            VERIFY(::CloseHandle(pThreadHandle[i]));
        }
        delete[iNumberOfProcessors] pThreadHandle;
        delete[iNumberOfProcessors] pInfo;
        /* ... */
    }
    DWORD WINAPI StructureBlockThread(LPVOID pInfo) {
        StructureInfo& info=*((StructureInfo*) pInfo);
        for (int j=info.m_iStart;j<info.m_iFinish;j++) {
            StructureForGroup(j,info.m_options);
        }
        return 0;
    }
    void StructureForGroup(int iGroupIndex,
        StructureOptions& options) {
        /* generate subspace index values, */
        /* low and high threshold values */
        /* for particular receptive group */
        /* ... */
    }
```

Следует отметить, что в приведенном примере отсутствует использование инструментов синхронизации доступа к объектам данных. Это достигается благодаря логической организации алгоритма. В частности, каждый рабочий поток оперирует с непересекающимися подмножествами массивов памяти.

Полностью аналогично производится распараллеливание следующих базовых операций классификатора: вычисление выходных сигналов рецептивных групп, модификация синаптической матрицы. В самом деле, данные операции выполняются независимо для каждого значения индекса рецептивной группы j.

Несколько более сложным является алгоритм распараллеливания вычислений постсинаптических потенциалов для нейронов последнего слоя **C**. Поскольку два последних слоя классификатора соединены друг с другом по принципу "каждый с каждым", критерием распараллеливания может является как множество нейронов слоя **B**, так и множество нейронов слоя **C**. С другой стороны, для большинства задач классификации количество используемых классов m будет сравнимо с количеством процессоров P. Таким образом, если $m \neq P$ и критерием распараллеливания является множество нейронов слоя **C**, то распределение вычислительной нагрузки по процессорам будет неравномерным. Более целесообразно, как и ранее, распараллеливать алгоритм по отношению к множеству значений индекса j. В этом случае, следует вычислять постсинаптический потенциал нейронов слоя **C**, создаваемый соответствующим подмножеством нейронов слоя **B**, т.е. блоком рецептивных групп. Данный подход аналогичен модифицированной каскадной схеме для вычисления частных сумм некоторого массива числовых данных [17,18]. Ниже приведен шаблон кода C++ для рассматриваемого алгоритма.

```
    struct SynapticOptions {
        int* m_pPotentials; // array of postsynaptic potentials
        /* ... */
    };
    struct SynapticInfo {
        int m_iStart; // starting index of the structure block
        int m_iFinish; // starting index of the next block
        SynapticOptions m_options; // required parameters
```

```
};
/* ... */
int iNumberOfGroups,iNumberOfClasses,iNumberOfProcessors;
// function that calculates partial postsynaptic potentials
// provided by one particular receptive group
void SynapticForGroup(int,SynapticOptions&);
// function declaration in order to use
// the thread entry function pointer
DWORD WINAPI SynapticBlockThread(LPVOID);
/* ... */
// assume the variables iNumberOfGroups, iNumberOfClasses
// and iNumberOfProcessors are initialized
int iBlockSize=iNumberOfGroups/iNumberOfProcessors;
void SynapticPotentials(SynapticOptions& options) {
    /* ... */
    SynapticInfo* pInfo=new SynapticInfo[iNumberOfProcessors];
    HANDLE* pThreadHandle=new HANDLE[iNumberOfProcessors];
    for (int i=0;i<iNumberOfProcessors;i++) {
        /* copy required option attributes */
        // allocate partial postsynaptic potentials
        pInfo[i].m_options.m_pPotentials=new int[iNumberOfClasses];
        pInfo[i].m_iStart=i*iBlockSize;
        pInfo[i].m_iFinish=(i+1)*iBlockSize;
        /* ... */
        DWORD dwThreadID;
        // start worker thread for each structure block
        pThreadHandle[i]=::CreateThread(NULL,0,
            SynapticBlockThread,&(pInfo[i]),0,&dwThreadID);
        ASSERT(pThreadHandle[i]!=NULL);
    }
    // before current thread can continue it's
    // needed to wait for each worker thread to exit
    ::WaitForMultipleObjects(iNumberOfProcessors,
        pThreadHandle,TRUE,INFINITE);
    // calculating total postsynaptic potentials
    // as soon as all worker threads finished execution
    for (int k=0;k<iNumberOfClasses;k++) {
        options.m_pPotentials[k]=0;
        for (i=0;i<iNumberOfProcessors;i++) {
            options.m_pPotentials[k]+=
                pInfo[i].m_options.m_pPotentials[k];
        }
    }
    for (i=0;i<iNumberOfProcessors;i++) {
        // close thread handle so that operating
        // system can destroy the object
        VERIFY(::CloseHandle(pThreadHandle[i]));
        // deallocate partial postsynaptic potentials
        delete[iNumberOfClasses]
            (pInfo[i].m_options.m_pPotentials);
    }
    delete[iNumberOfProcessors] pThreadHandle;
    delete[iNumberOfProcessors] pInfo;
    /* ... */
}

DWORD WINAPI SynapticBlockThread(LPVOID pInfo) {
    SynapticInfo& info=*((SynapticInfo*) pInfo);
```

```
        for (int k=0;k<iNumberOfClasses;k++) {
            info.m_options.m_pPotentials[k]=0;
        }
        for (int j=info.m_iStart;j<info.m_iFinish;j++) {
            SynapticForGroup(j,info.m_options);
        }
        return 0;
    }
    void SynapticForGroup(int iGroupIndex,
        SynapticOptions& options) {
        /* increment corresponding partial synaptic */
        /* potentials by the value that corresponds */
        /* to particular receptive group */
        /* ... */
    }
```

Заключение и анализ

В современном мире требования по быстродействию вычислительных систем постоянно растут. Многие разработчики аппаратных средств сталкиваются с проблемой дальнейшего повышения тактовой частоты процессорных устройств. Например, процессоры компании Intel с частотой в 3 гигагерца были разработаны еще в 2001 году. Промышленный выпуск таких процессоров был освоен только в 2005 году, а ресурсы архитектурного совершенствования отдельно взятого процессора оказались в основном исчерпанными. Также в 2005 году компании Intel и AMD одновременно приступили к продаже двухъядерных процессоров для персональных компьютеров. Фактически это означает, что дальнейшее повышение быстродействия приложений становится возможным только благодаря разработке эффективных параллельных алгоритмов для многопроцессорных компьютерных систем [19].

Нейросетевые технологии представляют собой мощное средство для решения многих задач искусственного интеллекта. Большинство топологий нейронных сетей допускает достаточно эффективное распараллеливание. Таким образом, при параллельной программной реализации нейросетевые алгоритмы становятся одним из наиболее удобных способов решения вычислительно сложных задач.

В данной работе рассмотрен нейросетевой классификатор со случайными подпространствами. Все операции расчета данного классификатора являются либо бинарными, либо дискретными. Таким образом, алгоритм прямого распространения информации в сети является достаточно вычислительно эффективным. Следует также отметить, что по сравнению с другими моделями нейронных сетей, в данном случае отсутствует необходимость расчета различных вещественнозначных функций, градиентов функции ошибок и т.д. В работе представлены методы распараллеливания базовых операций классификатора с использованием многопотоковых алгоритмов.

Существует немного работ, где анализируются экспериментальные результаты распараллеливания нейронных сетей. В частности, в [20] на двухъядерном процессоре достигается повышение производительности до 60%. В работе [21] проводится сравнение скорости вычислений для многопроцессорного компьютера, гомогенного кластера Beowulf и гетерогенного кластера. Наибольшее ускорение достигается для многопроцессорной вычислительной системы, причем для каждого подхода существует оптимальное количество процессоров или узлов кластера.

1. *Haykin S.* Neural networks a comprehensive foundation. 2-nd edition, Upper Saddle River, NJ, Prentice Hall, 1999. – 842 p.
2. *Duda R.O., Hart P.E., Stork D.G.* Pattern Classification. 2-nd edition, Wiley Interscience, 2000. – 654 p.
3. *Cybenko G.V.* Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems, 1989. – 2. – P. 303–314.
4. *Hornik K., Stinchcombe M., White H.* Multilayer feedforward networks are universal approximators. Neural Networks, 1989. – 2. – P. 359–366.
5. Нейрокомпьютеры и интеллектуальные роботы / Под ред. Н.М. Амосова, Киев: Наук. думка, 1991. – 272 с.
6. *Куссуль Э.М.* Ассоциативные нейроподобные структуры. – Киев: Наук. думка, 1992. – 210 с.
7. *Vapnik V.N.* Statistical Learning Theory. Wiley Interscience, 1998. – 736 p.
8. *Жора Д.В.* Анализ функционирования классификатора со случайными порогами // Кибернетика и системный анализ. – К.: 2003, – 3, – С. 72–91.
9. *Kussul E.M., Rachkovskij D.A., Wunsch D.C.* The random subspace coarse coding scheme for real-valued vectors. International Joint Conference on Neural Networks, 1999, – 1, – P. 450–455.
10. *Zhora D.V.* Evaluating Performance of Random Subspace Classifier on ELENA Classification Database. International Conference on Artificial Neural Networks. - 2005, LNCS 3697. – P. 343–349.
11. *Kussul E.M., Baidyk T.N., Lukovich V.V., Rachkovskij D.A.* Adaptive high performance classifier based on random threshold neurons. Cybernetics and Systems'94, Ed. R.Trappl, Singapore, World Scientific Publishing Co. Pte. Ltd., 1994. – P. 1687–1695.
12. *Aviles-Cruz C., Guérin-Dugué A., Voz J.L., Van Cappel D.* Databases, Enhanced Learning for Evolutive Neural Architecture. Tech. Rep. R3-B1-P, INPG, UCL, TSA, 1995. – 47 p., – http://www.dice.ucl.ac.be/neural-nets/Research/Projects/ELENA/elena.htm.
13. *Blayo F., Cheneval Y., Guérin-Dugué A., Chentouf R., Aviles-Cruz C., Madrenas J., Moreno M., Voz J.L.* Benchmarks, Enhanced Learning for Evolutive Neural Architecture. Tech. Rep. R3-B4-P, INPG, EERIE, EPFL, UPC, UCL, 1995. – 114 p.
14. *Klander L.* Core Visual C++ 6. Upper Saddle River, NJ, Prentice-Hall PTR, 2000. – 638 p.

15. *Stroustrup B.* The C++ Programming Language. 3-rd edition, Upper Saddle River, NJ, Addison-Wesley, 2001. – 1020 p.
16. *Резник А.М., Куссуль М.Э, Сычов А.С., Садовая Е.Г., Калина Е.А.* Система автоматизированного проектирования модульных нейронных сетей CAD MNN. // Математические машины и системы, 2002. – № 3. – С. 28–36.
17. *Гергель В.П., Стронгин Р.Г.* Основы параллельных вычислений для многопроцессорных вычислительных систем. – Н.Новгород, ННГУ, 2001.
18. *Воеводин В.В., Воеводин В.В.* Параллельные вычисления. БХВ-Петербург. – 2002. – 609 с.
19. *Зюбин В.* Многоядерные процессоры и программирование // Открытые системы, 2005.–№7–8.–http://www.osp.ru/os/2005/07-08/012.htm.
20. *Tsaregorodtsev V.G.* Parallel Implementation of Back-Propagation Neural Network Software on SMP Computers. PaCT 2005, Springer-Verlag, Berlin Heidelberg, LNCS 3606. 2005. – P. 186–192.
21. *Seiffert U.* Artificial Neural Networks on Massively Parallel Computer Hardware // European Symposium on Artificial Neural Networks 2002, Bruges, Belgium, ISBN 2-930307-02-1. – P. 319–330.
22. *Чистяков В.П.* Курс теории вероятностей. – М.: Наука, 1987. – 256 с.