ПРОГРАММНАЯ РЕАЛИЗАЦИЯ НЕЙРОННЫХ СЕТЕЙ ДЛЯ ОБУЧЕНИЯ В СИСТЕМАХ РЕАЛЬНОГО ВРЕМЕНИ

Скобцов Ю. А., Ткаченко А. В.
Донецкий национальный технический университет,
кафедра автоматизированных систем управления
E-mail: a.tkachenko@kiev-konti.com

Abstract

Skobtsov Y.A., Tkachenko A.V. Program realization of neural networks for learning in real-time systems. *Problem of program realization of artificial neural networks for real-time system is considered. Best methods for neural network training are overviewed. Resilient backpropagation algorithm is considered as one of the most efficient weight adaptation method. Application of conjugate graph for gradient generation is shown. Problem of weight initialization is considered and Ngueyn-Widrow method is applied to give good first approximation. Program library implementing overviewed methods is developed on C. High speed of the library is shown in comparison with Mathlab.*

Анализ исследований по рассматриваемой проблеме. Существуют два способа реализации искусственных нейронных сетей – аппаратная и программная. Программная реализация, уступая аппаратной по скорости работы и автономности, обладает рядом очевидных преимуществ, связанных с простотой использования и внедрения в информационно-управляющую систему.

Несмотря на то, что работы по программной реализации нейронных сетей осуществляются на протяжении более чем 30 лет, найти библиотеку программ позволяющую моделировать и обучать многослойные сети прямого распространения, пригодную для использования в некоторой разрабатываемой системе довольно сложно. Перечислим некоторые средства моделирования доступные в Internet:

- Stuttgart Neural Network Simulator [7];
- Qnet [8];
- NeuroSolutions [9];
- Fast Artificial Neural Network Library [10];
- Jets Neural Library [11].

Обычно это готовые программные продукты, либо библиотеки программ написанные под Unix, которые имеют трудности при переносе в Windows.

Самым распространенным инструментом для применения нейронных сетей является Matlab Neural Network Toolbox. Нейронные сети в Matlab, в следствие поставленной цели моделирования широкого класс сетей, имеют громоздкую реализацию и невысокую скорость при решении конкретного класса задач. Кроме того, использование Matlab в системе реального времени затруднительно.

В системах реального времени построение математических моделей управляемых процессов должно быть автоматизировано в максимально возможной степени. Поэтому возникает потребность интегрирования нейросетевого комплекса в систему управления, что позволить повысить ее адаптивные свойства.

Очевидно, что при использовании нейронной сети, наиболее эффективна оптимизированная под данную конкретную задачу программная реализация. Когда требования к скорости обучения высоки, разработка конкретной программной реализации нейронной сети становится необходимостью.

Постановка задачи и цели исследований. Цель данного исследования – разработка библиотеки программ, сокращенно называемой OSNN (On-Line System Neural Network), позволяющей моделировать и обучать многослойные нейронные сети прямого распространения в системах реального времени. В качестве языка программирования используется широко известный и легко переносимый на другие платформы язык C++. Для организации обучения будут использоваться методы признанные в настоящее время наиболее эффективными. Под эффективностью алгоритма будем понимать в первую очередь время работы алгоритма, а во вторую объем требуемых ресурсов. В тоже время при одинаковой эффективности предпочтение будет отдаваться алгоритмам более простым в реализации. В качестве возможных функций активации сети будет рассматриваться линейная функция $f(u) = u$ и гиперболический тангенс $f(u) = (e^u - e^{-u})/(e^u + e^{-u})$.

Задача обучения нейронной сети. Задано множество пар векторов *{x,d}* называемое обучающим множеством, где *{x}* входной вектор, а *{d}* желаемый выходной вектор. *{y}* - множество реакций нейронной сети на вход *{x}*. Тогда мера разницы между *{y}* и *{d}* ($E = \|y - d\|$) называется ошибкой обучения. Используя среднюю квадратичную меру ошибки получим

$$E = \frac{1}{SM}\sum_{i=1}^{S}\sum_{j=1}^{M}\left(y_j^i - d_j^i\right)^2, \qquad (1)$$

где S – число обучающих пар, M - размерность выходного вектора.

Задача обучения нейронной сети сводится к поиску таких значений весовых коэффициентов $w_{i,j}^{(k)}$, чтобы ошибка обучения E стала меньше некоторого значения ε ($E < \varepsilon$).

Существует огромное количество алгоритмов обучения нейронных сетей. Для произвольной задачи нельзя заранее сказать какой из них будет наиболее быстрым. Однако для сетей большой размерности (с числом весовых коэффициентов больше 500-1000), лучшие результаты дают методы основанные на применении сопряженных градиентов, и различные эвристические алгоритмы [1,6]. Методы сопряженных градиентов довольно сложны в реализации. Например, метод масштабируемых сопряженных градиентов (scaled conjugate gradient), показывающий лучшие результаты из этого класса методов, требует вычисления производных второго порядка целевой функции [5].

Метод "неунывающего" обратного распространения RPROP. Лучшим из эвристических алгоритмов обучения является алгоритм RPROP (resilient backpropagation) [2]. Его основным преимуществом над методами сопряженных градиентов является простота реализации при высокой скорости сходимости и низких требованиях к погрешности вычисления градиента. Алгоритм RPROP основывается на поведении знаков градиентов. Он не требует сложных вычислений и не зависит от величин производных. Приращение на каждом шаге вычисляется индивидуально для каждого веса. Приращение вычисляется по формуле

$$\Delta_l^{(i)} = \begin{cases} \eta^+ \Delta_l^{(i-1)}, & если \ \dfrac{\partial E(w)}{\partial w_l}^{(i-1)} \dfrac{\partial E(w)}{\partial w_l}^{(i)} > 0 \\ \eta^- \Delta_l^{(i-1)}, & если \ \dfrac{\partial E(w)}{\partial w_l}^{(i-1)} \dfrac{\partial E(w)}{\partial w_l}^{(i)} < 0 \\ \Delta_l^{(i-1)}, & если \ \dfrac{\partial E(w)}{\partial w_l}^{(i-1)} \dfrac{\partial E(w)}{\partial w_l}^{(i)} = 0 \end{cases}, \qquad (2)$$

где $0 < \eta^- (= 0.5) < 1 < \eta^+ (= 1.2)$. Величина приращения усиливается фактором η^+ в том случае, когда алгоритм сходится к минимуму и производная не меняет знак. Это ускоряет процесс на плоских участках и замедляет поиск, в случае пропуска локального минимума.

Значения весов модифицируются в соответствии с направлением убывания градиента

$$\Delta w_l^{(i)} = \begin{cases} \Delta_l^{(i)} \operatorname{sgn}\left[\frac{\partial E(w)}{\partial w_l}^{(i)}\right], & \text{если } \frac{\partial E(w)}{\partial w_l}^{(i-1)} \frac{\partial E(w)}{\partial w_l}^{(i)} \geq 0 \\ -\Delta_l^{(i)}, & \text{если } \frac{\partial E(w)}{\partial w_l}^{(i-1)} \frac{\partial E(w)}{\partial w_l}^{(i)} < 0 \end{cases}, \quad (3)$$

где sgn[*] – функция знака. Когда производная ошибки изменяет знак, показывая, что минимум пропущен, происходит возврат к предыдущему значению веса $w_l^{(i-1)}$. При этом, для того что бы избежать изменения знака и на следующем шаге необходимо обнулить значение производной $\frac{\partial E(w)}{\partial w_l}^{(i)}$. Таким образом, алгоритм имеет вид:

$$if\left(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) > 0\right)$$

{

$$\Delta_{ij}(t) = \min(\Delta_{ij}(t-1) * \eta^+, \Delta_{\max})$$

$$\Delta w_{ij}(t) = -\Delta_{ij}(t) * sign\left[\frac{\partial E}{\partial w_{ij}}(t)\right]$$

$$\Delta w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

} *else* {

$$if\left(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) < 0\right)$$

{

$$\Delta_{ij}(t) = \max(\Delta_{ij}(t-1) * \eta^-, \Delta_{\min})$$

$$\Delta w_{ij}(t+1) = w_{ij}(t) - \Delta w_{ij}(t-1)$$

$$\frac{\partial E}{\partial w_{ij}}(t) = 0$$

} *else* {

$$\Delta w_{ij}(t) = -\Delta_{ij}(t) * sign\left[\frac{\partial E}{\partial w_{ij}}(t)\right]$$

$$\Delta w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

}

}

Однако шаг в обратном направлении эффективен только для сетей небольшой размерности и обычно приводит лишь к ухудшению результата. Поэтому в OSNN, в случае изменения знака шаг назад не делается, а лишь уменьшается приращение:

$$if\left(\frac{\partial E}{\partial w_{ij}}(t-1)*\frac{\partial E}{\partial w_{ij}}(t)<0\right)$$

{

$$\Delta_{ij}(t)=\max(\Delta_{ij}(t-1)*\eta^{-},\Delta_{\min})$$

$$\Delta w_{ij}(t)=-\Delta_{ij}(t)*sign\left[\frac{\partial E}{\partial w_{ij}}(t)\right]$$

$$\Delta w_{ij}(t+1)=w_{ij}(t)+\Delta w_{ij}(t)$$

}

Начальные величины приращения $\Delta_l^{(0)}$ выбираются случайно из промежутка (0,1). Параметры $\Delta_{\min}$ и $\Delta_{\max}$ оказывают заметное влияние на работу алгоритма. Автором метода предложены значения $\Delta_{\max}=50$ и $\Delta_{\min}=1e^{-6}$. Величина $\Delta_{\min}=1e^{-6}$ в условиях конкретной задачи может оказаться слишком большой это не позволит алгоритму точно двигаться в пространстве весовых коэффициентов и внесет дополнительные возмущающие воздействия, ограничивая минимальный шаг. Выбор значения $\Delta_{\min}=0$ также имеет свои минусы, поскольку в этом случае некоторые веса после нескольких тысяч итераций могут быть навсегда исключены из процесса обучения. Наилучшим вариантом, реализованным в OSNN, является выбор минимального, отличного от нуля, возможного для типа double значения, например $\Delta_{\min}=1e^{-300}$.

Метод потоковых графов для генерации градиента. Важным и трудоёмким этапом обучения является нахождения градиента целевой функции. Формулы расчета градиента, взятые в явном виде от целевой функции очень сложны и неудобны для практического применения, особенно в том случае если сеть содержит больше одного скрытого слоя. Поэтому представляется удобным на основе метода потоковых графов построить простые правила формирования компонентов градиента, которые имеют постоянную структуру, не зависящую от сложности сети [4].

Сопряженный граф $\hat{G}$ определяется как исходный граф G, в котором направленность всех дуг изменена на противоположную. Линейная дуга графа G и соответствующая ей дуга сопряженного графа $\hat{G}$ имеют идентичные описания. Способ формирования сопряженного графа и методика его возбуждения для автоматического формирования вектора градиента на основе двух графов G и $\hat{G}$ представлены на рис. 1.

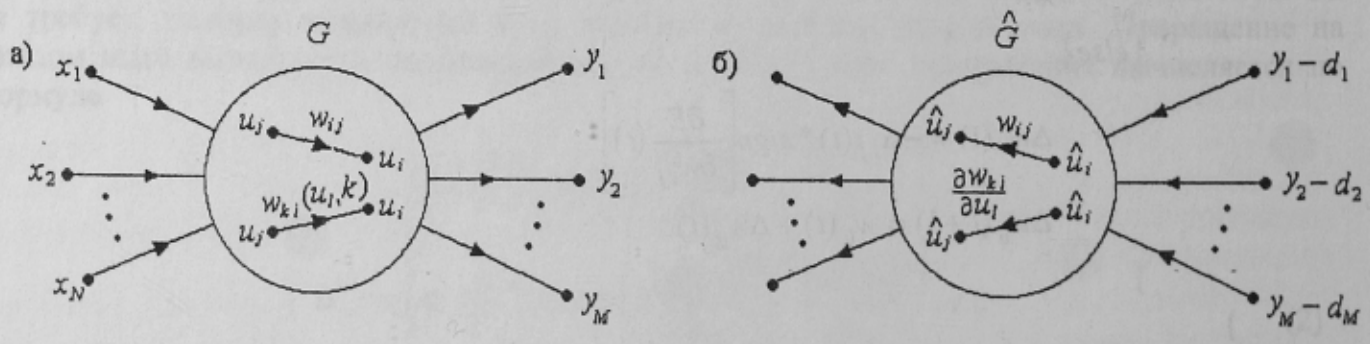

Рисунок 1 - Иллюстрация применения способа формирования и возбуждения сопряженного графа: а) исходный граф G; б) сопряженный граф $\hat{G}$.

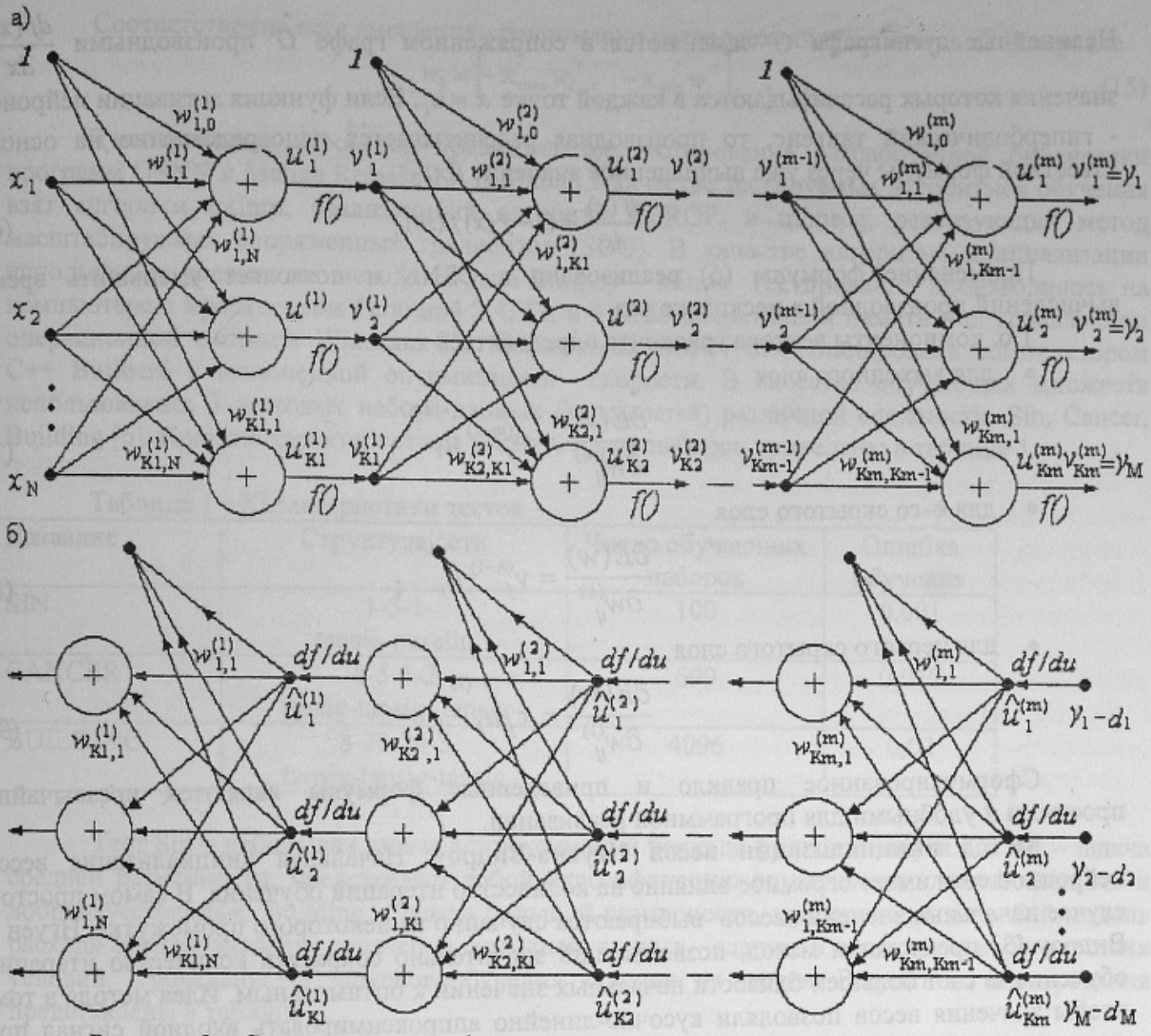

Рисунок 2 - Сопряженные графы для генерации вектора градиента однонаправленной многослойной сети: а) выходной граф сети б) сопряженный граф сети.

Теперь можно рассчитать любой компонент вектора градиента ошибки. Для линейной дуги графа, описываемой весом w_{ij}, формула имеет вид:

$$\frac{\partial E(w)}{\partial w_{ij}} = v_j \hat{v}_i . \qquad (4)$$

Для нелинейной дуги графа G, описываемой функцией $w_{kl}(v_l, K)$, получаем:

$$\frac{\partial E(w)}{\partial K} = \hat{v}_l \frac{\partial w_{kl}(v_l, K)}{\partial K} . \qquad (5)$$

Представленные выражения применимы для любых систем (линейных, нелинейных, рекуррентных и т.п.). Рассмотрим изображенною на рис. 2а типовую многослойную сеть (состоящую их m слоев) с произвольной непрерывной функцией активации нейронов. Количество нейронов в каждом слое обозначим K_j (j=1, 2, …m), причем последний слой является выходным слоем сети, содержащим $K_j = M$ нейронов. Выходные сигналы нейронов в конкретных слоях обозначим $v_j^{(k)}$. На рис. 2б представлен сопряженный граф: Сопряженный граф возбуждается разностями между фактическими y_i и ожидаемыми значениями d_i .

Нелинейные дуги графа G заменяются в сопряженном графе $\hat{G}$ производными $\frac{df(x)}{dx}$, значения которых рассчитываются в каждой точке $x = u_i$. Если функция активации нейронов - гиперболический тангенс, то производная рассчитывается непосредственно на основе известной формулы через уже вычисленное значение *f(x)*

$$\frac{df(x)}{dx} = (1 - f(x)f(x)). \quad (6)$$

Применение формулы (6) реализовано в OSNN и позволяет уменьшить время вычислений производной в несколько раз.

Т.о. компоненты вектора градиента определяться как:

- для выходного слоя

$$\frac{\partial E(w)}{\partial w_{ij}^{(m)}} = v_j^{(m-1)} \hat{u}_i^{(m)} \quad ; \quad (7)$$

- для k–го скрытого слоя

$$\frac{\partial E(w)}{\partial w_{ij}^{(k)}} = v_j^{(k-1)} \hat{u}_i^{(k)} \quad ; \quad (8)$$

- для первого скрытого слоя

$$\frac{\partial E(w)}{\partial w_{ij}^{(1)}} = x_j \hat{u}_i^{(1)} \quad ; \quad (9)$$

Сформулированное правило и приведенные формулы являются чрезвычайно простыми и удобными для программной реализации.

Метод инициализации весов Нгуена-Видроу. Начальная инициализация весов нейронной сети имеет огромное влияние на количество итераций обучения. В самом простом случае начальные значения весов выбираются случайно из некоторого промежутка. Нгуен и Видроу [3] предложили метод, позволяющий значительно сократить количество итераций обучения за счет большей близости начальных значений к оптимальным. Идея метода в том, чтобы значения весов позволяли кусочно-линейно аппроксимировать входной сигнал при помощи линейных участков функции активации. Для сигмоидальной функции активации линейная область расположена на интервале аргумента (-1,1). Следовательно, для случая слоя с одним входом и H выходами необходимо чтобы выполнялось условие

$$-1 < w_i x + w_b < 1, \quad (10)$$

где x – равномерно распределенная величина из интервала $[x_{min}, x_{max}]$. Приняв ширину интервала для вех нейронов одинаковой и в сумме заполняющей входной диапазон

$$w_i = \frac{1}{(x_{max} - x_{min})} H \quad (11)$$

получим значения весов смещения как равномерно распределенную величину на интервале

$$w_b \in \left[-x_{max} w_i, \quad -x_{min} w_i\right]. \quad (12)$$

Аналогично получаем формулу для слоя с I входами и H выходами. При этом желательно слегка расширить интервалы “интереса” весов, оставив распределения весов смещения прежним.

$$w_i^* = \frac{1}{(x_{max} - x_{min})} H^{\frac{1}{I}}, \quad (13)$$

$$w_i = 0.7 \frac{1}{(x_{max} - x_{min})} H^{\frac{1}{I}}. \quad (14)$$

Соответственно веса смещения - равномерно распределенная величина на интервале

$$w_b \in \left[-x_{\max} w_i^*, \quad -x_{\min} w_i^*\right]. \tag{15}$$

Тестирование. Проведем сравнительное тестирование разработанной библиотеки программ OSNN и Matlab Release 13. В Matlab в качестве тестируемых алгоритмов обучения взят алгоритм trainrp, реализующий алгоритм RPROP, и trainscg, реализующий метод масштабируемых сопряженных градиентов (SCG). В качестве алгоритма инициализации использована реализация метода Нгуена-Видроу - initnw. Тестирование производилось на компьютере с процессором Pentium4 2,4ГГц и 512Мб оперативной памяти под управлением операционной системы Windows XP. Библиотека OSNN откомпилирована компилятором C++ Builder6 с включенной оптимизацией скорости. В качестве обучающих множеств использовались 3 тестовых набора данных (benchmarks) различной сложности: Sin, Cancer, Building [6]. Краткие характеристики этих тестовых наборов приведены в таблице 1.

Таблица 1 - Характеристики тестов.

Название	Структура сети	Число обучающих наборов	Ошибка обучения
SIN	1-5-1 tansig-purelin	100	0,001
CANCER	9-5-5-2 tansig-tansig- tansig	699	0,012
BUILDING	8-20-20-3 tansig-tansig-tansig	4096	0,02

Тест Sin – небольшая задача аппроксимации периода функции синуса. Canser – задача средней размерности, представляет собой классификацию опухолей на злокачественные и доброкачественные. Building – задача большой размерности, представляющая собой модель расхода электроэнергии, горячей и холодной воды в здании в зависимости от внешних условий. Проведены 30 запусков обучения каждого из тестов. Результаты тестирования представлены в таблице 2.

Таблица 2 - Результаты тестирования OSNN и Mathlab.

Программа	Среднее время обучения, с.	С.к.о времени обучения, с.	Среднее число итераций	Среднее время одной итерации, с.
SIN				
OSNN	0,18	0,29	329	0,00050
Mathlab trainrp	2,18	2,6	599	0,00482
Mathlab trainscg	0,74	0,7	110	
CANCER				
OSNN	30,04	28,62	3069	0,0098
Mathlab trainrp	31.27	32.30	2398	0,0131
Mathlab trainscg	30,98	48,67	962	
BUILDING				
OSNN	174,75	134,81	691	0,252
Mathlab trainrp	382.42	165.31	1342	0.285
Mathlab trainscg	1081.67	251.97	1994	

Основным критерием сравнения программ, который интересует нас при разработки информационно-управляющих систем реального времени, является время обучения. Кроме того, сравнение OSNN и trainrp по количеству итераций позволяет определить лучшую

реализацию алгоритма RPROP, в смысле выбора начальных значений и коэффициентов. Сравнение trainrp и trainscg позволит показать эффективность методов для задач различной размерности.

На тестовом наборе SIN OSNN показала значительно лучшие результаты по всем характеристикам: общему времени обучения, времени одной итерации и их количеству. На задаче малой размерности заметно преимущество метода SCG, использующего вторые производные для более быстрой сходимости.

В тесте Cancer время обучения всех программ приблизительно одинаково. Реализация метода RPROP в Mathlab показала результаты по количеству итераций лучше чем OSNN. Это можно объяснить отличной от OSNN реализацией алгоритма начальной инициализация сети, дающее более высокие результаты для слоев с малым количеством нейронов.

В следующем тесте Building OSNN полностью оправдала свою разработку, сократив время обучения более чем в два раза, причем как за счет большей скорости вычислений, так и за счет изменений в реализации RPROP. В этом тесте, как на задаче большой размерности, RPROP наконец, показал свое значительное превосходство над методом SCG.

Выводы. Рассмотрена проблема программной реализации нейронных сетей в системах реального времени. Сделан вывод о необходимости разработки интегрируемой библиотеки программ, позволяющей моделировать и обучать нейронные сети прямого распространения. Рассмотрен алгоритм неунывающего обратного распространения и предложена его эффективная реализация. Изучен способ формирования градиента на основе метода потоковых графов. Рассмотрен алгоритм начальной инициализации весов Нгуена-Видроу. Разработана программная реализация описанных методов в виде библиотеки программ под названием OSNN. Проведено сравнительное тестирование OSNN с реализациями алгоритмов RPROP и SCG на Mathlab.

Литература

1. Осовский С. Нейронные сети для обработки информации: Пер. с польского. М: Финансы и статистика, 344 с., 2002.

2. M. Riedmiller, “A direct method for faster backpropagation learning”, Proceedings of the 1993 IEEE International Conference on Neural Networks (ICNN ’93), Vol. 1, San Francisco, 586-591.

3. Nguyen, D., and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," *Proceedings of the International Joint Conference on Neural Networks*, vol 3, pp. 21-26, 1990.

4. Rumelhart D.E., Hinton G.E., Williams R.J. Learning internal representations by error propagation. // Parallel Distributed Processing: Exploration in the Microstructure of Cognition, D.E.Rumelhart and J.L.McClelland (Eds.), vol. 1, Cambridge, MA: MIT Press, 1986. PP. 318 - 362.

5. Moller, M. F. ‘A scaled conjugate gradient algorithm for fast supervised learning’, Neural Networks, vol. 6, pp. 525-533, 1993.

6. Speed and memory comparison: backpropagation (Neural Network Toolbox) http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/backpr14.html.

7. http://www-ra.informatik.uni-tuebingen.de/SNNS/.

8. www.qnetv2k.com.

9. www.neurosolutions.com.

10. http://fann.sourceforge.net/.

11. http://www.voltar.org/jneural.