7. *Tuzovskiy A.F., Chirikov S.V., Yampolskiy V.Z.* Sistemy upravleniya znaniyami (metody i tekhnologii) [The knowledge management system (methods and techniques)]. Tomsk: Izd-vo NTL, 2005, 260 p.
8. *Rodzin S.I.* Vychislitelnyy intellekt: nemonotonnye logiki i graficheskoe predstavlenie znaniy [Computational intelligence: nonmonotonic logic and graphical representation of knowledge], *Programmnye produkty i sistemy* [Software and Systems], 2002, No. 1, pp. 20-22.
9. *Bova V.V.* Modelirovanie oblasti znaniy v sistemakh podderzhki prinyatiya resheniy dlya nepreryvnogo professionalnogo obucheniya [Modeling the field of knowledge systems, decision support for continuous professional training], *Izvestiya YuFU. Tekhnicheskie nauki* [Izvestiya SFedU. Engineering Sciences], 2009, No. 4 (93), pp. 242-248.
10. *Kureychik V.V., Sorokoletov P.V., Shcheglov S.N.* Analiz sovremennogo sostoyaniya avtomatizirovannykh sistem priobreteniya i predstavleniya znaniy [Analysis of the current state of the automated systems of the acquisition and knowledge representation], *Izvestiya YuFU. Tekhnicheskie nauki* [Izvestiya SFedU. Engineering Sciences], 2008, No. 9 (86), pp. 120-125.
11. *Kravchenko Yu.A.* Sintez raznorodnykh znaniy na osnove ontologiy [The synthesis of heterogeneous knowledge based on ontologies], *Izvestiya YuFU. Tekhnicheskie nauki* [Izvestiya SFedU. Engineering Sciences], 2012, No. 11 (136), pp. 216-221.
12. *Bova V.V., Kureychik V.V., Nuzhnov E.V.* Problemy predstavleniya znaniy v integrirovannykh sistemakh podderzhki upravlencheskikh resheniy [The problem of representation of knowledge in integrated systems to support management decisions], *Izvestiya YuFU. Tekhnicheskie nauki* [Izvestiya SFedU. Engineering Sciences], 2010, No. 7 (108), pp. 107-113.
13. *Kravchenko Yu.A., Markov V.V.* Ontologicheskiy podkhod formirovaniya informatsionnykh resursov na osnove raznorodnykh istochnikov znaniy [Ontological approach of formation of information resources on the basis of different sources of knowledge], *Izvestiya YuFU. Tekhnicheskie nauki* [Izvestiya SFedU. Engineering Sciences], 2013, No. 7 (144), pp. 116-120.
14. *Polkovnikova N.A., Kureychik V.M.* Razrabotka modeli ekspertnoy sistemy na osnove nechyotkoy logiki [Development of model of expert systems based on fuzzy logic], *Izvestiya YuFU. Tekhnicheskie nauki* [Izvestiya SFedU. Engineering Sciences], 2014, No. 1 (150), pp. 83-92.

Статью рекомендовал к опубликованию д.т.н., профессор Ю.А. Гатчин.

Бова Виктория Викторовна – Южный федеральный университет; e-mail: vvbova@yandex.ru; 347928, г. Таганрог, Некрасовский, 44; тел.: 88634371651; кафедра систем автоматизированного проектирования; старший преподаватель.

Bova Victoria Victorovna – Southern Federal University; e-mail: vvbova@yandex.ru; 44, Nekrasovskiy, Taganrog, 347928, Russia; phone: +78634371651; the department of computer aided design; senior teacher.

УДК 004.421.6

Н.И. Витиска, С.К. Буханцева

СИНТЕЗ БИБЛИОТЕКИ КЛАССОВ OBJECTIVE-C_NEURON ДЛЯ ПРОЕКТИРОВАНИЯ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ ПРИ РАСПОЗНАВАНИИ ОБРАЗОВ

Излагается объектно-ориентированный метод создания библиотеки при внедрении искусственных нейронных сетей в мобильных и переносных устройствах, осуществляющих программные реализации моделей распознавания образов. Рассмотрены проблемы моделирования нейронных сетей для решения различных прикладных задач, сложно реализуемых с помощью стандартных методов. Предложена методология проектирования нейронных сетей на платформе iOS, которая позволяет реализовывать прикладные программы при распознавании образов без облачных вычислений, а также методы их создания и обучения с помощью разработан-

ной библиотеки Objective-C_Neuron, которая дает возможность в дальнейшем расширения инструментария с помощью добавления других типов нейронных сетей. Предложенная библиотека дополняет известные, так как возможно расширение алгоритмов обучения нейронных сетей. Особое внимание обращается на увеличение области применения нейронных сетей - такие приложения работают как на мобильных устройствах, так и на персональных компьютерах. Расширение области применения программ при распознавании образов является очень важной и необходимой частью современных технических разработок. Главное достоинство предложенной системы – удобный способ работы с многопоточностью. Рассматривается возможность распараллеливания основных вычислений по созданию связей между нейронами, где расчет весов будет реализовываться на GUP. Задачи графики предполагают независимую параллельную обработку данных, и GPU изначально мультипоточен. Микроархитектура спроектирована так, чтобы эксплуатировать имеющееся в наличии большое количество нитей, требующих исполнения. Таким образом, неграфические вычисления, реализуемые на GPU дают значительное увеличение в производительности по сравнению с традиционными решениями.

Нейронная сеть; методология; метод; библиотека; инструментарий; многопоточность.

N.I. Vitiska, S.K. Bukhantseva

SYNTHESIS CLASS LIBRARY OBJECTIVE-C_NEURON FOR THE ARTIFICIAL NEURAL NETWORKS IN PATTERN RECOGNITION

Describes the object-oriented method for creating libraries in the implementation of artificial neural networks in mobile and portable devices, offering software-based models of pattern recognition. This article addressed the challenges of modeling of neural networks for various applications complexly using by standard methods. Methodology of designing neural networks for iOS platform was proposed. Also methods of creating and learning through the developed library Objective-C_Neuron were created that give ability to expand the toolkit. Particular attention has been paid to increase range of application of neural networks. The main advantage of proposed system is a comfy way of multithreading working. Such applications work on mobile devices and personal computers. The possibility of parallelization basic calculations to establish the connections between neurons, where the calculation of weights will be implemented in GUP. Task graphs assume independent parallel processing, and GPU originally multipotochen. Microarchitecture is designed to exploit the available large number of threads that require execution. Thus, non-graphical calculations implemented on the GPU give a significant increase in performance compared to traditional solutions.

Neural network; methodology; method; library; toolkit; multithreading.

Введение. При решении задач кластеризации, классификации данных, прогнозирования, аппроксимации непрерывных функций, а также прикладных задач, сложно реализуемых с помощью стандартных методов, используются программные реализации математических моделей. Универсальные системы Neural Network Toolbox (пакет расширения MATLAB), NeuroSolutions, Statistica, BrainMaker, OpenCV, PWNLIB, NNGPULIB обеспечивают достаточно полный инструментарий для моделирования и проектирования широкого круга искусственных нейронных сетей. С развитием технологий устройства с операционной системой iOS становятся всё более сложными, их функциональность разрастается, появляются новые приложения. Многие приложения используют нейронные сети, однако на данный момент инструментарии для создания и обучения искусственных нейронных сетей (ИНС) существуют преимущественно для компьютерных приложений на мощных серверах.

Постановка задачи. Для работы с нейронными сетями(НС) на платформах iOS необходимо подключить готовую библиотеку, реализованную на более подходящих для этого платформах Matlab, Python, и обучить нужную сеть. Этот подход относится к симуляционным методам реализации ИНС [1]. Он отличается значительными временными затратами, так как осуществляется воспроизведение на последовательной архитектуре вычислительных машин параллельные нейрофизиологические процессы. Поэтому отдельные исследования направлены на использо-

вании программно-аппаратных методов эмуляции НС [2, 3], которые предполагают добавление в классическую архитектуру нейропроцессоров или нейрочипов. Однако в архитектурах переносных и мобильных устройствах отсутствуют нейрокомпьютерные системы , так как они влияют на их стоимость. Поэтому стремятся поставить процессоры средней производительности, например, процессор Apple A6X 1.4 GHz. С обработкой ИНС он может справиться. Но, несмотря на то, что Apple A6X – двуядерный процессор с 4-х ядерным графическим сопроцессором – один из самых мощных процессоров для мобильных устройств в настоящее время, все вычисления, связанные с проектированием сложной нейронной сети, являются облачными. На практике такие вычисления необходимы, но производительность процессоров планшетов уступает производительности процессоров серверов. Для того, чтобы можно было создавать и обучать нейронные сети непосредственно на устройствах с операционной системой iOS необходимо синтезировать библиотеку классов objective-c, что позволит увеличить область применения нейронных сетей.

Задачей данной статьи является разработка инструментария для создания ИНС на платформах iOS. Необходимо получить библиотеку базовых классов, с помощью которой можно воспользоваться уже реализованными типами нейронных сетей и алгоритмами обучения, либо при наследовании от базовых классов создать свою собственную ИНС. Технология должна предоставить удобный способ работы с многопоточностью. Данное требование обусловлено тем, что многопоточность крайне важна для ИНС [4].

Также необходимо создать НС с возможностью изучения свойств выдаваемого нейронной сетью решения для определения адекватности или неадекватности нейронной сети. Ни одна из стандартных нейросетевых программ до сих пор не предоставляет возможности исследования факта скоррелированности ошибки прогноза нейронной сети с ее входными (независимыми) переменными: значимая корреляция является наиболее простой характеристикой неадекватности обученной нейронной сети.

Кроме того, поддержка итеративного процесса над цепочкой этапов “анализ данных – выбор конфигурации модели – адаптация модели – изучение динамики показателей в ходе адаптации модели – изучение статистических свойств решения – изучение результатов влияния различных настроек алгоритмов на свойства получаемых при этом моделей”, возникающие на каждом этапе, показатели могут обрабатываться с помощью одних и тех же стандартных средств статистического анализа и визуализации данных.

Разработанная библиотека должна иметь возможность расширения инструментария с помощью добавления других типов нейронных сетей. Также возможно расширение алгоритмов обучения нейронных сетей.

Структура библиотеки Objective-C_Neuron. Базовым классом для создания нейронной сети является класс NeuralNetwork, от которого должны наследоваться все типы нейронных сетей. В нем представлены все основные функции нейронных сетей. Наследники должны переопределять только инициализацию. Нейронная сеть состоит из слоёв, каждый из которых, в свою очередь, состоит из нейронов. Ниже приведен листинг программы, реализующий инициализацию и создание многослойных прецептронов.

```
@implementation MultiLayerPerceptron
-(MultiLayerPerceptron*) initWithNeuronsInLayers: (NSMutableArray*) neuronsInLayers
{
self = [super init];
if (self)
{
```

```
[self createNetwork: neuronsInLayers];
}
return self;
}

-(void) createNetwork: (NSMutableArray*) neuronsInLayers
{
int neuronsInLayer = [[neuronsInLayers objectAtIndex: 0] intValue];
Layer *layer = [[Layer alloc] initWithLayerNeuronCount: neuronsInLayer];
BOOL useBias = YES;
//check if it is initialized in properties, then use that value
if (useBias)
{
Neuron *n = [[Neuron alloc]init];
[layer addNeuron:n];
}
[self addLayer:layer];
Layer *prevLayer = layer;
for (int layerIndex=1; layerIndex<neuronsInLayers.count; layerIndex++)
{
neuronsInLayer = [[neuronsInLayers objectAtIndex: layerIndex] intValue];
layer = [[Layer alloc] initWithLayerNeuronCount: neuronsInLayer];
if (useBias && (layerIndex<neuronsInLayers.count-1))
{
[layer addNeuron:[[Neuron alloc]init]];
}
[self addLayer:layer];
if (prevLayer)
{
[ConnectionFactory fullConnectPrevLayer: prevLayer andThisLayer: layer];
}
prevLayer=layer;
}
[self setLearningRule:[[BackPropagation alloc]init]];
}

-(void) dealloc
{
 [super dealloc];
}
@end
```

У каждого нейрона есть функция входа (Input Function) и активационная функция (Transfer Function). На основе функции входа мы задаём изначальный вход нейрона. Активационная функция используется при вычислении выхода нейрона и при обучении. InputFunction складывается на основе WeightsFunction и SummingFunction. Для создания связей между нейронами используется класс Connection. Он содержит в себе вес, начальный и конечный нейроны. Сам нейрон хранит в себе два множества:

1. Множество InputConnections – все связи, которые входят в нейрон.
2. Множество OutputConnactions – все связи, которые выходят из нейрона.

У каждой нейронной сети можно задать обучающее правило – LearningRule. Данный класс также является базовым, от которого могут наследоваться другие классы, реализующие тот или иной алгоритм обучения. У каждого обучающего правила (LearningRule) есть обучающее множество (TrainingSet), которое, в свою очередь, состоит из обучающих элементов (TrainingElement). TrainingElement создаётся на основе обучающих данных. Ниже приводится листинг программы, реализующий обучающие правила, множество на основе входного вектора.

```
@implementation LearningRule

-(void) setLearningRate:(double) rate
{
learningRate = rate;
}

-(void) setMaxIterations:(int)iterations
{
maxIterations = iterations;
iterationsLimited = YES;
}

-(void) setNeuralNetwork: (NeuralNetwork*) aNeuralNetwork
{
neuralNetwork = aNeuralNetwork;
}

-(void) run
{
stopLearning = NO;
[self learn:trainingSet];
}
-(void) learn:(TrainingSet *)aTrainingSet
{
[self reset];
while (!stopLearning)
{
[self doLearningEpoch: aTrainingSet];
currentIteration++;
if (iterationsLimited && currentIteration==maxIterations)
{
[self stopLearning];
}
else if (!iterationsLimited && (currentIteration == INT_MAX))
{
currentIteration = 1;
}
}
}

-(void) doLearningEpoch:(TrainingSet *)aTrainingSet
{
previousEpochError = totalNetworkError;
totalNetworkError = 0;
NSEnumerator * enumerator = [aTrainingSet enumerator];
TrainingElement *element ;
while ((element = [enumerator nextObject]) && !stopLearning)
{
if ([element isKindOfClass:[SupervisedTrainingElement class]])
{
SupervisedTrainingElement *supervisedTrainingElement
= (SupervisedTrainingElement*)element;
[self learnPattern:supervisedTrainingElement];
}
}
if ([self hasReachedStopCondition])
{
[self stopLearning];
}
```

```
}

-(void) learnPattern: (SupervisedTrainingElement*) trainingElement
{
NSMutableArray *input = [trainingElement input];
[neuralNetwork setInput: input];
[neuralNetwork calculate];
NSMutableArray *output = [neuralNetwork getOutput];
NSMutableArray *desiredOutput = [trainingElement desiredOutput];
NSMutableArray *patternError = [self getPatternErrorFromOutput:output
andDesiredOutput: desiredOutput];
[self updateTotalNetworkError: patternError];
[self updateNetworkWeights: patternError];
}

-(void) stopLearning
{
stopLearning = YES;
}

-(BOOL) hasReachedStopCondition
{
return ((totalNetworkError<maxError) || [self errorChangeStalled]);
}

-(BOOL) errorChangeStalled
{
double absErrorChange = fabs(previousEpochError - totalNetworkError);
if (absErrorChange <= minErrorChange)
{
minErrorChangeIterationsCount++;
if (minErrorChangeIterationsCount >= minErrorChangeIterationsLimit)
{
return YES;
}
} else
{
minErrorChangeIterationsCount = 0;
}
return NO;
}

-(void) setMaxError:(double) error
{
maxError = error;
}

-(void) reset
{
currentIteration = 0;
}

-(NSMutableArray*)getPatternErrorFromOutput:(NSMutableArray*)output
andDesiredOutput: (NSMutableArray*) desiredOutput
{
NSMutableArray *patternError = [[NSMutableArray alloc] init];
double outputError = 0;
for(int i = 0; i < [output count]; i++)
{
outputError = [[desiredOutput objectAtIndex:(NSUInteger)i] doubleValue] –
[[output objectAtIndex:(NSUInteger)i] doubleValue];
[patternError addObject: [NSNumber numberWithDouble:outputError]] ;
```

```
}
return patternError;
}

-(void) updateTotalNetworkError: (NSMutableArray*) patternError
{
double sqrErrorSum = 0;
double error = 0;
int patternErrorCount = [patternError count];
for (int i=0; i<patternErrorCount; i++)
{
error = [[patternError objectAtIndex:(NSUInteger)i] doubleValue];
sqrErrorSum += (error * error);
}
totalNetworkError += sqrErrorSum / (2*patternErrorCount);
}
@end
```

Сохранение и загрузка нейронных сетей. Так как процесс обучения нейронной сети является трудоёмким и занимает долгое время, имеет смысл не совершать его каждый раз, когда нужна сеть, а сделать один раз и запомнить результаты. В качестве средства создания электронного формата описания моделей нейронных сетей был выбран язык XML [5], что позволяет легко переносить и реализовывать архитектуру на других технологиях. Нотация XML позволяет гибко описывать структурные данные и является универсальной основой для разработки специализированных языков описания объектов различной природы [6].

Для реализации сохранения был взят класс XMLTag, который позволяет генерировать xml файлы. Его реализация проста: создается тэг с определенным именем, к нему можно добавить атрибуты с помощью метода addAttribute WithName и внутренний текст через свойство (property) innerText. Внутренние тэги добавляются с помощью метода addChild.

Сохраняется нейронная сеть при вызове метода save у базового класса NeuralNetwork. На текущий момент используется одна общая архитектура для хранения всех типов нейронных сетей. Загрузка написана с использованием стандартного SAX-парсера NSXMLParser в классе NeuralNetworkXMLReader, который оказался очень удобным и быстрым. Работает, как на iphone, так и на Mac OS.

```
<net>
<layer id = ”0” size = ”2”>
        <neuron id =”0” />
        <neuron id =”1” />
</layer>
<layer id = ”1” connectedTo = ”0” size = ”1”>
        <neuron id = ”0”>
                <connections>
                        <connection from = ”0”>0,5</connection>
                        <connection from = ”1”>-0,5</connection>
                </connections>
        </neuron>
</layer>
</net>
```

Выше представлен пример хранения нейронной сети. Корневым тэгом является тэг net – нейронная сеть. В нем хранится несколько слоев с именем layer, у каждого слоя есть атрибуты id (идентификатор), size (количество нейронов) и connectedTo (указывает на идентификатор предыдущего слоя, с которым он связан). В layer лежат нейроны с уникальным идентификатором id для своего слоя. Связи же хранятся только у нейронов, в которых есть входные связи в форме тэга connection. Каждый

connection имеет атрибут from, где указан уникальный идентификатор начального нейрона для этой связи из предыдущего слоя, а внутри тэга лежит вес. Данная архитектура универсальна для хранения различных типов нейронных сетей.

Создание сверточных нейронных сетей с помощью библиотеки Objective-C_Neuron. Идея сверточных нейронных сетей заключается в чередовании сверточных слоев (Convolution-layers), субдискретизирующих слоев (Subsampling-layers) и наличии полносвязных (Fullyconnected-layers) слоев на выходе. В основе сверточных сетей лежат три механизма, используемые для достижения инвариантности к переносу, масштабированию, незначительным искажениям:

1. Локальное извлечение признаков. Каждый нейрон получает входной сигнал от локального рецептивного поля в предыдущем слое, извлекая, таким образом, его локальные признаки. Как только признак извлечен – его точное расположение уже не имеет значение, поскольку установлено его местонахождение относительно других признаков.
2. Формирование слоев в виде набора карт признаков. Каждый вычислительный слой состоит из множества карт-признаков – плоскостей, на которых все нейроны должны использовать одно и то же множество синаптических весов. Такая форма усложняет структуру сети, однако имеет два важных преимущества: инвариантность к смещению, которое достигается с помощью свертки с ядром небольшого размера, и сокращение числа свободных параметров, которое достигается за счет совместного использования синаптических весов нейронами одной и той же карты.
3. Подвыборка. За каждым слоем свертки следует вычислительный слой, осуществляющий локальное усреднение и подвыборку. За счет этого, достигается уменьшение разрешения для карт признаков. Такая операция приводит к понижению чувствительности выходного сигнала оператора отображения признаков к незначительному смещению и прочим формам деформации. В качестве такого оператора выступает одна из сигмоидальных функций, используемых при построении нейронных сетей, например гиперболический тангенс.

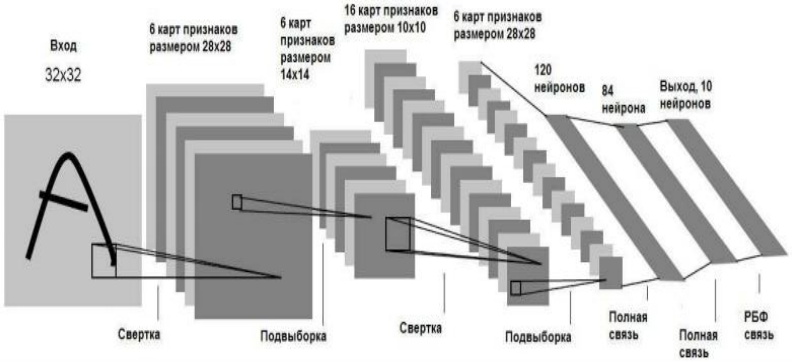

Рис. 1. Сеть свертки, реализующая распознавание изображений

Во входной слой поступает центрированное изображение символа. Такая операция делается для того, чтоб характерные признаки рисунка (дуги, концевые точки) находились в центре рецептивного поля при извлечении признаков высокого порядка. В вышеописанной сети, стандартные символы базы MNIST размером 28x28 пикселов позиционировались в центр изображения 32x32 пикселя. Значения входных пикселов затем нормализируется для ускорения сходимости обучения. Точные нормализированные значения пикселов вычисляется в зависимости от ти-

па используемой активационной функции, реализуемой в файле TransferFunction, в данном случае фоновому пикселю соответствует значение – 0.1, а пикселю, формирующему символ – 1.175, согласно [7].

Первый скрытый слой является слоем свертки. Он состоит из 6 карт признаков размером 28x28. Рассмотрим процесс формирования этого слоя, поскольку остальные сверточные слои формируются подобным образом. Каждый элемент карты признаков соединен с областью размером 5x5 на входном изображении. Следовательно, каждый элемент карты имеет 25 обучаемых коэффициентов и обучаемый сдвиг. Значение элемента карты вычисляется по формуле 1.

$$X_i^{h,l} = f\left(\sum_{k=1}^{n_i-1}\sum_{j=-\infty}^{\infty} X_{i=j}^{k,l} W_{i-j,i}^{h,k,l} + B_i^{h,l}\right). \quad (1)$$

Исходя из полученных результатов можно вычислить рецептивные поля соседних элементов, а также количество связей и обучаемых параметров последующих слоев удобным для программиста способом.

Заключение. Программы, написанные при помощи iOS SDK, работают как на мобильных устройствах, так и на персональных компьютерах, что позволит увеличить область применения нейронных сетей. Использование наиболее эффективных в вычислительном плане алгоритмов и также качественное их программирование дают максимальную скорость работы программ, что позволяет обрабатывать большие объемы данных в наиболее сложных современных задачах.

Была разработана библиотека для создания и обучения искусственных нейронных сетей для iOS. Главным ее достоинством является возможность в дальнейшем расширения инструментария с помощью добавления других типов нейронных сетей. Так же возможно расширение алгоритмов обучения нейронных сетей.

С помощью разработанной библиотеки можно воспользоваться уже реализованными типами нейронных сетей и алгоритмами обучения, либо создать собственную ИНС. Также система предоставляет удобный способ работы с многопоточностью, где подход с многопоточным обучением одной нейронной сети более масштабируем, чем несколько параллельных однопоточных обучений.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Че*рнухин Ю.В.* Искусственный интеллект и нейрокомпьютеры. – Таганрог: Изд-во ТРТУ, 1997. – 273 с.
2. *Кирсанов Э.Ю.* Нейрокомпьютеры с параллельной архитектурой. Кн. 16. – М.: Радиотехника, 2004. – 496 с.
3. *Злобин В.К., Ручкин В.Н.* Нейросети и нейрокомпьютеры: Учеб. пособие. – СПб.: БХИ – Петербург, 20011. – 256 с.
4. *Галушкин А.И.* Теория нейронных сетей. – М.: ИПРЖ «Радиотехника», 2000. – 416 с.
5. Extensible Markup Language. (XML) 1.0. W3C Recommendation. URL: //http://www.w3.org/TR.
6. The SGML/XML Web Page-Extensible Markup Language (XML). URL: //http://www.oasis-open.org/cover/xml.html.
7. *Simard P.Y.* Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis, Microsoft. – P. 23-24.

REFERENCES

1. *Chernukhin Yu.V.* Iskusstvennyy intellekt i neyrokompyutery [Artificial intelligence and neural computers]. Taganrog: Izd-vo TRTU, 1997, 273 p.
2. *Kirsanov Eh.Yu.* Neyrokompyutery s parallelnoy arkhitekturoy [Neural computers with parallel architecture]. Moscow: Radio-tekhnika, 2004, 496 p.
3. *Zlobin V.K., Ruchkin V.N.* Neyroseti i neyrokompyutery [Neural networks and neurocomputers]: Ucheb. posobie. Saint-Petersburg: BKhI – Peterburg, 20011, 256 p.

4. *Galushkin A.I.* Teoriya neyronnykh setey [The theory of neural networks]. Moscow: IPRZh «Radiotekhnika», 2000, 416 p.
5. Extensible Markup Language. (XML) 1.0. W3C Recommendation. Available at: //http://www.w3.org/TR.
6. The SGML/XML Web Page-Extensible Markup Language (XML). Available at: //http://www.oasis-open.org/cover/xml.html.
7. *Simard P.Y.* Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis, Microsoft, pp. 23-24.

Статью рекомендовал к опубликованию д.т.н., профессор В.Ф. Гузик.

Витиска Николай Иванович – Таганрогский государственный педагогический институт; e-mail: vit614294@rambler.ru; 347900, г. Таганрог, ул. Инициативная, 48; тел.: 88634601892; кафедра информатики; профессор.

Буханцева София Константиновна – e-mail: sophia.btv@gmail.com; кафедра информатики; аспирантка.

Vitiska Nikolay Ivanovich – Taganrog Pedagogical University; e-mail: vit614294@rambler.ru; 48, Initsiativnaya street, Taganrog, 347900, Russia; phone: +78634601892; the department of information science; professor.

Bukhantseva Sophia Konstantinovna – e-mail: Sophia.btv@gmail.com; the department of information science; postgraduate student.

УДК 681.3

Ю.О. Чернышев, Н.Н. Венцов, П.А. Панасенко

АЛГОРИТМ ПРИНЯТИЯ ПРОЕКТНЫХ РЕШЕНИЙ НА ОСНОВЕ НЕЧЕТКИХ КОМАНД*

Описан процесс формулирования нечетких команд на основе перечислительного и аналитического представления функций принадлежности. Нечеткая команда может быть сформулирована, как на основе одного, так и на основе двух частично противоречивых условий, заданных функциями принадлежности. На основе нечетких команд разработан алгоритм управления поиском проектных решений. Задание степени соответствия перечислением кортежей позволяет строить графики функций принадлежности произвольной формы, но при этом требуемые ресурсы памяти возрастают пропорционально частоте дискретизации. Использование аналитического способа задания функции принадлежности чисел приблизительно близких к x, за счет изменения параметра θ, дает возможность получать графики симметричные относительно x. Аналитическая запись функции принадлежности позволяет минимизировать зависимость от частоты дискретизации предметной области. В качестве способа остановки алгоритма предлагается использовать автомат адаптации. Изменение глубины памяти автомата адаптации позволяет корректировать инерционность процесса поиска приемлемого решения. Применение нечетких команд ускоряет управление вычислительным процессом, использование автоматов адаптации позволяет корректировать получаемые результаты.

Нечеткие данные; адаптация; принятие решений; интеллектуальные системы; оптимизация.

* Работа выполнена при финансовой поддержке РФФИ (проекты: № 12-01-00474, 13-01-00343).