СПИСОК ЛИТЕРАТУРЫ:

1. *Chaum D.* Security without identification: transaction systems to make big brother obsolete // Communications of the ACM. 1985. № 28 (10). P. 1030–1044.
2. *Camenisch J. and Lysyanskaya A.* Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation // EUROCRYPT 2001. LNCS. Springer Verlag, 2001. P. 93–118.
3. *Camenisch J. and Lysyanskaya A.* Signature schemes and anonymous credentials from bilinear maps // Advances in Cryptology — CRYPTO 2004. Vol. 3152 of LNCS. Springer Verlag, 2004. P. 56–72.
4. *Rabin M.* How to exchange secrets by oblivious transfer. Harvard Aiken Computation Laboratory, 1981.
5. *Camenisch J., Neven G., Shelat A.* Simulatable adaptive oblivious transfer // EUROCRYPT 2007. Vol. 4515 of LNCS. Springer-Verlag, 2007. P. 573–590.
6. *Aiello W., Ishai Y., Reingold O.* Priced oblivious transfer: How to sell digital goods // EUROCRYPT 2001. Vol. 2045 of LNCS. Springer-Verlag, 2001. P. 119–135.
7. *Di Crescenzo G., Ostrovsky R., Rajagopalan S.* Conditional oblivious transfer and timed-release encryption // EUROCRYPT'99. Vol. 1592 of LNCS. Springer-Verlag, 1999. P. 74–89.
8. *Herranz J.* Restricted adaptive oblivious transfer // Cryptology ePrint Archive, Report 2008/182, 2008. URL: http://eprint.iacr.org/.

С. Д. Жилкин

МЕТОДЫ ПОСТРОЕНИЯ МОДЕЛЕЙ ШТАТНОЙ РАБОТЫ ПО И АЛГОРИТМЫ ВЫЯВЛЕНИЯ АНОМАЛЬНОГО ПОВЕДЕНИЯ ПО

Введение

Среди нынешних задач сферы компьютерной безопасности остро стоит проблема выявления событий некорректной работы программного обеспечения. На текущий момент обозначены многие способы решения этой задачи. Данная статья в общих чертах обозначает методы выявления некорректной и аномальной работы ПО с помощью математического аппарата нейронных сетей.

Так как нейронные сети успешно используются для угадывания образов, на которые они были натренированы [1], основная идея предлагаемого решения задачи заключаются в сравнении заранее заданных (эталонных) моделей поведения ПО с их реальным поведением. Результат сравнения двух образов посредством нейронных сетей является вероятностью совпадения этих образов [1]. По величине этой вероятности (а также некоторым дополнительным характеристикам нейронной сети) можно говорить об аномальном поведении программы. Таким образом, решение проблемы выявления аномалии поведения с помощью нейронных сетей подразумевает выполнение следующих шагов: а) построение модели поведения программы, б) постоянное сравнение запускаемых экземпляров программы с эталонной моделью.

1. Построение модели поведения ПО

Для начала опишем, что представляет собой модель поведения ПО с точки зрения нейронной сети. Нейронная сеть — это набор слоев с так называемыми нейронами. Каждый нейронный слой связан с последующим некой активационной функцией и множеством коэффициентов и смещений. Каждый нейрон принимает на вход множество значений (координат), вычисляет свой выход по активационной функции и передает получившееся значение в следующий слой. В зависимости от способов передачи результатов активационной функции по нейронным слоям нейронные сети делятся на различные классы. В данной статье речь будет идти о билинейном персептроне, каждый

слой которого полностью связан с соседними (Рис. 1). Выбор данного класса нейронной сети объясняется тем, что ему посвящено больше всего научных работ, затрагивающих вопросы обучения и оптимизации, а также для него разработано значительное количество готовых программных реализаций математических механизмов [1].

Активационную функцию для нейрона билинейного персептрона можно представить в следующем виде:

$$A(X) = f(x),$$

где X — вектор, поступивший на нейрон, $x = c1*x1 + c2*x2 + ... + cn*xn + b$, xi — i-я координата вектора X, ci — i-й коэффициент из множества коэффициентов для данного нейрона, b — смещение для данного нейрона, $f()$ — чаще всего сигмоидальная функция.

Нейронная сеть получает на вход вектор некоторых значений, т. е. набор координат. В общем случае выход нейронной сети также является вектором со множеством координат. При решении поставленной задачи положим размерность выходного вектора равной 1. Таким образом, мы имеем следующий набор переменных: нейронная сеть $N(f(), C, B)$, где C — набор коэффициентов, B — набор смещений, $f()$ — активационная функция; вектор входных значений X; численный результат выхода нейронной сети $N(X)$.

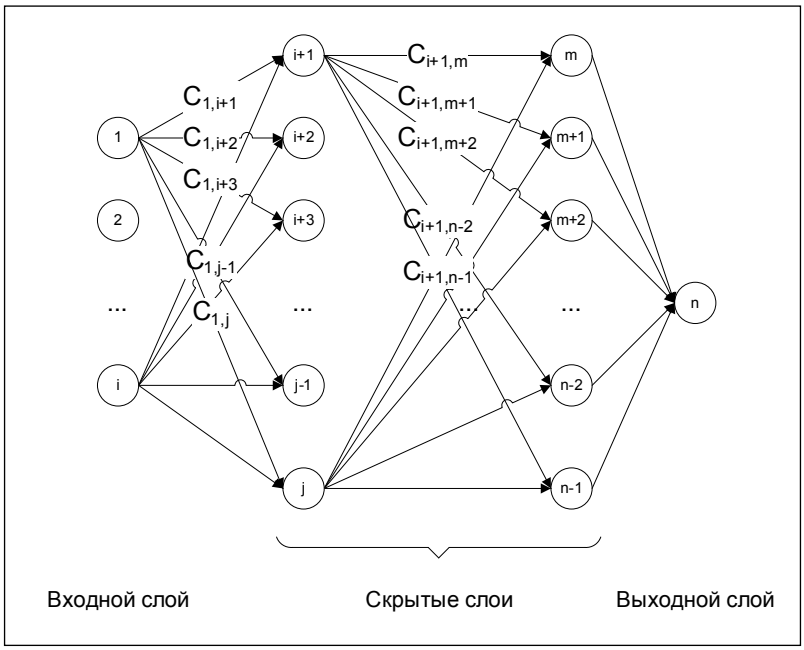

Рис. 1. Схема билинейного персептрона

Связывая имеющиеся математические сущности с поставленной задачей, будем считать входной вектор описанием поведения ПО. Численный результат выхода нейронной сети ограничим диапазоном [0; 1] и будем считать его вероятностью соответствия анализируемого поведения ПО эталонному поведению ПО. Нейронную сеть будем считать моделью корректного поведения программы, на которой входные вектора, описывающие корректное поведение, будут давать результат, очень близкий к 1.

Приняв данный механизм за основу аппарата выявления аномальной работы ПО, необходимо связать работу ПО с входным вектором, описывающим эту работу. Для этого предлагается описывать работу ПО набором количественных (число обращений к различным ресурсам), логических (совершались ли какие-нибудь характерные действия) и статистических (частота выделения ресурсов) характеристик, которые в итоге составят входной вектор. Так, 6-я координата может показывать число сетевых обращений ПО за пределы локальной сети, а 11-я показывает, обращалось ли ПО к системному реестру.

Для реализации такого подхода необходима некоторая программа-агент, которая следит за работой процессов, основываясь, например, на журнальных файлах операционной системы. Размерность вектора зависит от возможностей такой программы-агента. Чем больше размерность вектора, тем точнее будет модель поведения ПО, тем больше будет вероятность нахождения отклонений от созданной модели поведения. Принципиальная схема обучения нейронной сети приведена на рисунке 2.

Обучение нейронной сети представляет собой конечный набор итераций, каждая из которых состоит в получении обучающего вектора и калибровке коэффициентов и смещений нейронной сети таким образом, чтобы на обучающем векторе данной итерации нейронная сеть выдавала результат не ниже некоторого порога, который следует принять близким к 1. При этом выбирается величина предельной ошибки e, которую затем следует учитывать при оценке выхода обученной нейронной сети.

При обучении сети основная задача заключается в выборе ее характеристик и параметров: будет ли сеть сужающейся или расширяющейся, сколько нейронов должен содержать скрытый слой, сколько итераций обучения следует провести и прочее. Не существует жестких формул, по которым можно вычислять эти параметры. В нашем распоряжении лишь набор рекомендаций, которыми можно руководствоваться при выборе параметров, однако при всем при этом не исключается использование метода проб и ошибок [2].

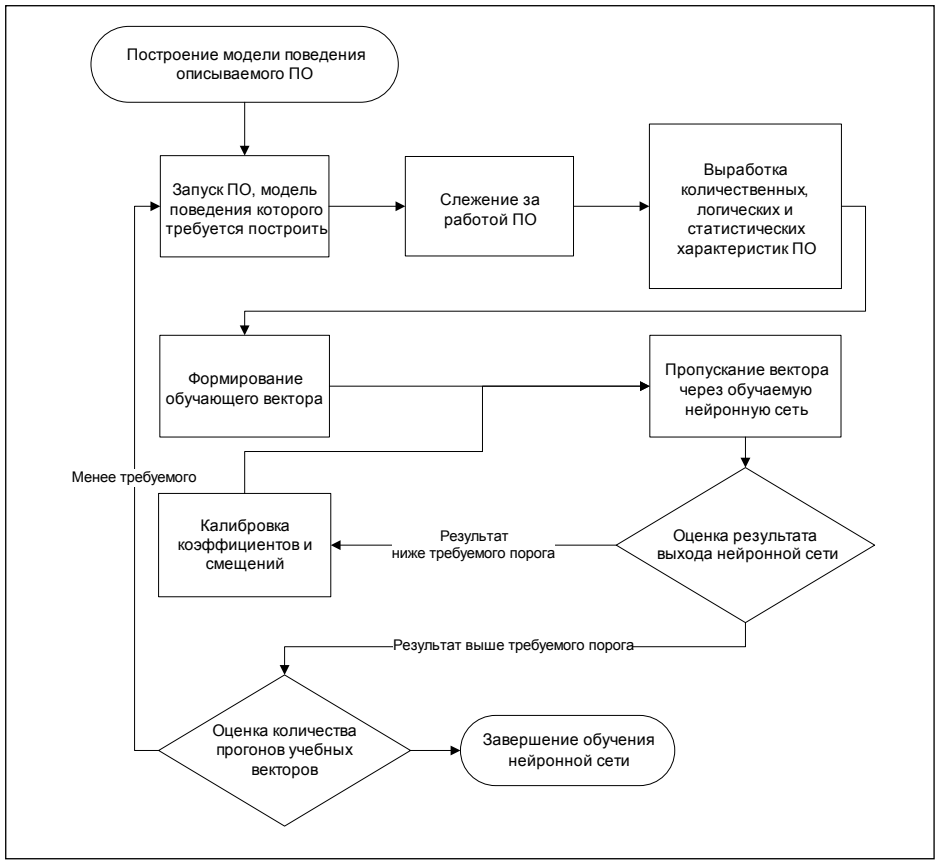

Рис. 2. Обучение нейронной сети

Как показала практика, наиболее эффективным и легко программируемым методом обучения (хоть и не самым быстрым [3]) является метод обратного распространения ошибки [4]. Основная идея этого метода состоит в распространении сигналов ошибки от выходов сети к ее входам в направлении, обратном прямому распространению результата в обычном режиме работы. В данном

методе, как показывает практика, очень хорошие результаты показывают сети, состоящие из двух уровней нейронов (помимо входного): одного скрытого уровня и одного выходного [5].

Таким образом, выбор метода обучения определяет структуру сети: всего один скрытый слой. Для определения числа нейронов этого слоя не существует точной формулы, однако есть соотношение Хечт–Нэльсона [4], использующее теорему Колмогорова, которая утверждает, что любая функция n переменных может быть представлена как суперпозиция $2n + 1$ одномерных функций. Т. е. нет никакого смысла выбирать количество скрытых элементов большим, чем удвоенное число входных элементов:

$$h \leq 2i + 1. \qquad (2.1)$$

Существует также простейшая оценка Баума и Хаусслера [6] для использования на практике, которая учитывает величину предельной ошибки при обучении сети. Утверждается, что количество тренировочных примеров t должно быть приблизительно равным количеству весов w сети, умноженному на обратную величину ошибки e:

$$t \geq w \cdot e^{-1}. \qquad (2.2)$$

Учитывая, что в нашем случае количество весов w примерно равно числу входов i, умноженному на размерность скрытого слоя h, получаем соотношение:

$$t \geq h \cdot i \cdot e^{-1}. \qquad (2.3)$$

На основании предполагаемого числа тренировочных прогонов ПО t, которые позволяется осуществить, и величины e можно сделать вывод о числе нейронов скрытого слоя.

При выборе числа нейронов скрытого слоя следует принимать во внимание результаты практических экспериментов, которые свидетельствуют, что лучшему обучению поддаются расширяющиеся сети [7]. Т. е. для обучения предпочтительны сети, в которых размер скрытого слоя больше, чем входного. В нашем случае это означает, что число нейронов скрытого слоя должно быть больше размерности вектора, описывающего поведение ПО.

Большое число обучающих итераций необходимо по той причине, что, несмотря на жесткое алгоритмическое поведение программы, почти каждый новый запуск порождает свой уникальный вектор. Это связано прежде всего с тем, что операционные системы имеют ряд оптимизационных механизмов, которые позволяют приложениям запускаться во второй раз значительно быстрее, чем в первый. Во время второго запуска может происходить меньше обращений к жесткому диску, оперативной памяти и прочим ресурсам системы. Запуски других программ, использующих смежные с моделируемым ПО ресурсы, также влияют на некоторые составляющие обучающих векторов ПО. Таким образом, набор итераций должен содержать как можно больше запусков моделируемого ПО, различных по своим условиям.

После обучения сети выбранный порог с вычетом выбранной предельной ошибки является значением, относительно которого для данной модели определяются аномалии поведения. Если вектор реальной работы ПО дал результат меньше 0,9 на нейронной сети, обученной на это ПО с порогом 1 предельной ошибкой 0.1, то считается, что поведение ПО некорректно.

Полученная сеть для моделируемого ПО с высокой степенью вероятности выявляет ПО, на которое она была обучена. Однако, если моделируемое ПО являлось частью некоего программного пакета, состоящего из схожего ПО (наиболее тривиальный пример — пакет Microsoft Office), обученная сеть может принимать это самое схожее ПО за то ПО, на которое она была обучена. Иными словами, сеть, обученная на PowerPoint, может дать высокий результат на векторе от Excel. Во избежание таких ошибок используется второй дополнительный этап обучения сети [2]. Суть дополнительного обучения состоит в обучении нейронной сети на обратный результат. Схема дополнительного обучения принципиально не отличается от схемы основного этапа обучения. Вместо моделируемого ПО запускается схожее по поведению ПО. Калибровка

сети осуществляется в том случае, если вектор схожего ПО дал результат на обученной сети выше некоторого порога, который определяется на основании предельной ошибки обученной сети. Калибровка осуществляется до тех пор, пока нейронная сеть не станет выдавать результат меньше порога. Как и в случае с первым этапом, таких калибровок не должно быть слишком много, чтобы не довести сеть до состояния «переобучения», когда она перестанет распознавать ПО, на которое была изначально натренирована. Обучение нейронной сети должно осуществляться для некоторого интервала времени работы ПО. Такой интервал называется активационным временем нейронной сети — время от запуска ПО, которое накапливались обучающие вектора.

2. Расширенная модель поведения ПО

В случае с масштабным ПО (например, программные пакеты для офисной работы) этап запуска позволяет очень точно обучить нейронные сети на распознавание данного ПО. Это связано прежде всего с тем, что при загрузке больших приложений происходит очень много событий работы с системными ресурсами и чем больше число обрабатываемых при обучении событий, тем точнее сеть будет распознавать свой образ. Поэтому для такого ПО вполне хватит одной сети, обученной только на запуск, для того, чтобы точно распознавать это ПО. Тем не менее предлагается расширение модели поведения до трех нейронных сетей на разные периоды работы ПО по следующим причинам:

• аномальное поведение может проявиться уже после запуска по истечении какого-то определенного периода времени либо как реакция на некоторые действия пользователя; в таком случае одной сети, натренированной на запуск, будет недостаточно для выявления аномального поведения;

• если ПО не относится к разряду масштабного, не имеет своего характерного «почерка» и при запуске не использует ничего, помимо стандартных библиотек, такое ПО будет очень плохо различимо. Например, с точки зрения использования системных ресурсов многие стандартные приложения системы Microsoft Windows неразличимы между собой. Тем не менее действия, совершаемые программой по команде пользователя после запуска, позволяют отличить одно ПО от другого.

Следовательно, необходимы дополнения модели поведения еще одной нейронной сетью, описывающей действия ПО после запуска. Однако в общем случае заранее невозможно предугадать, сколько продлится работа ПО, а все возможные варианты длительности работы из-за их количества невозможно промоделировать при обучении. Следовательно, в общем случае невозможно обучить нейронную сеть на работу ПО. Поэтому предлагается выделить некоторое характерное для программы действие и обучить нейронную сеть на это действие. В дополнение к двум сетям, обученным на запуск и характерную черту работы ПО, предлагается третья сеть, обученная на завершение работы ПО, так как аномальное поведение в целях маскировки может проявляться лишь под завершение работы ПО.

Таким образом, полная модель поведения ПО должна состоять из нейронных сетей трех родов: сеть первого рода описывает запуск ПО, сеть второго рода описывает некую характерную черту работы ПО, сеть третьего рода описывает завершение работы ПО.

В зависимости от сложности ПО может потребоваться несколько сетей второго рода, если ПО обладает несколькими специфическими чертами.

3. Автоматизированное построение моделей поведения ПО

Процесс обучения является наиболее сложным и трудоемким этапом в выполнении общей задачи выявления аномального поведения. При попытке построить модель заранее неопределенного ПО появляются следующие проблемы:

• Без знаний о ПО заранее не известно, какой должен быть активационный период каждой из трех сетей модели. Не известно, сколько длится запуск программы, сколько по времени она выполняет специфичные для себя действия. Если не знать этого и выбирать в качестве этих

параметров величины по умолчанию, есть риск ошибиться и не полностью захватить события, происходящие в упомянутых трех фазах работы ПО. Так возникает необходимость алгоритма нахождения нужных значений.

• Еще более сложной проблемой является создание сетей второго рода. При обучении не известны специфические действия программы (так, ПО может циклически совершать некоторые операции без ведома пользователя). Обучать вручную нейронные сети второго рода крайне трудоемко. Необходим механизм, способный в общем потоке действий ПО выявлять некоторые характерные моменты и обучать нейронные сети на распознавание таких моментов.

Для разрешения обоих проблем предлагается следующий подход при отслеживании действий ПО: создавать и хранить в памяти вектор, накопленный за каждую секунду работы ПО. Если ПО проработало в общей сложности *m* секунд, мы будем располагать *m* векторами, что позволит отслеживать активность ПО в каждую секунду, что, в свою очередь, поможет разграничить фазы работы ПО.

Приведем пример представления работы ПО набором таких односекундных векторов. Для наглядности будем считать, что вектор содержит всего 5 координат, описывающих работу ПО. Например, 1-я координата показывает число событий чтения системных файлов, 2-я — число событий записи данных на жесткий диск и т. д. Предположим, что ПО проработало 16 секунд. Таким образом, мы имеем 16 односекундных векторов. Выстраивая эти вектора в хронологическом порядке, получаем детальную картину работы ПО, представленную на рисунке 3.

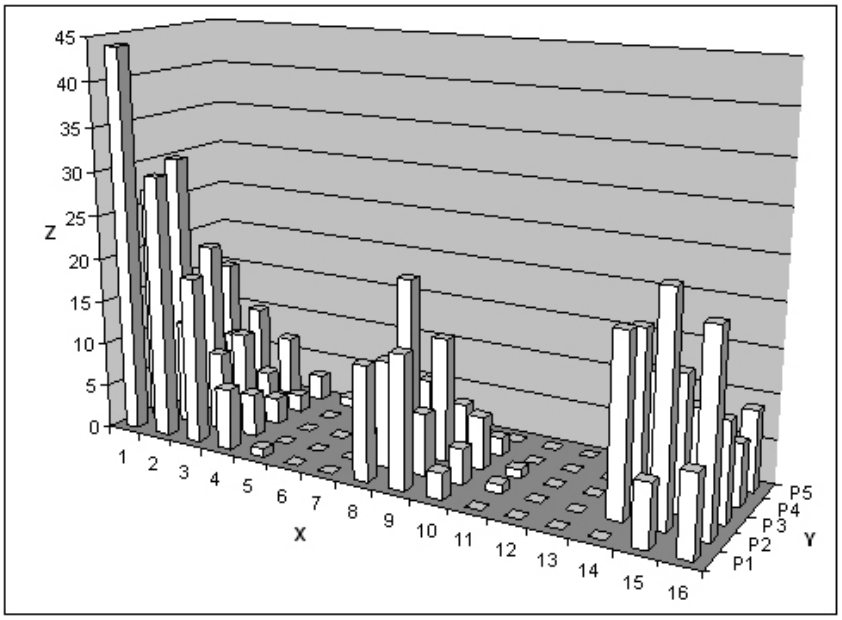

Рис. 3. Представление работы ПО набором односекундных векторов

По оси X отложено время работы ПО в секундах. По оси Y представлены координаты вектора. Ось Z показывает значения координаты. Представление работы ПО в таком графическом виде наглядно разделяет весь жизненный цикл ПО на три фазы. С помощью математических алгоритмов можно выделять эти фазы автоматически. В примере фаза запуска длится 4—5 секунд, фаза работы — 3—4 секунды, фаза завершения работы — 3 секунды. Следовательно, именно такие активационные периоды нужно выбирать при обучении нейронных сетей.

Также подобным разложением жизненного цикла ПО на односекундные вектора решается задача автоматического выявления специфических действий ПО. В приведенном примере очевидно, что такое действие ПО было совершено на 8—11-й секундах работы. Для обучения

нейронной сети второго рода на данное действие ПО необходимо построить обучающий вектор, сложив 4 односекундных вектора за 8—11-ю секунды работы ПО.

В общем случае может обнаружиться несколько таких специфических черт работы ПО. На каждое из них должна быть обучена нейронная сеть второго рода. Отличать два повторения одного специфического действия от двух различных специфических действий можно следующим образом: обучающий вектор для каждого нового найденного специфического действия ПО должен пропускаться через множество нейронных сетей, обученных на найденные ранее специфические действия. Если хотя бы на одной сети новый обучающий вектор даст результат, близкий к 1, считается, что найденное специфическое действие является повторением предыдущего, для которого уже есть обученная нейронная сеть.

4. Выявление аномального поведения программы

Процесс выявления аномального поведения ПО использует методы, применяемые при обучении сети. При запуске ПО специальная программа, выявляющая аномальное поведение, начинает накапливать односекундные вектора работы, чтобы в любой момент иметь возможность получить общий вектор за некоторый период работы ПО (например, вектор для фазы запуска). Такие общие вектора пропускаются через соответствующую нейронную сеть модели процесса. Результаты, близкие к 1, на всех нейронных сетях модели говорят о соответствии работы ПО его модели. Если выход на какой-нибудь сети оказался меньше $(1 - \rho)$, где ρ — точность, выбранная при обучении, значит, в той фазе, которую данная сеть описывает, обнаружено аномальное поведение.

Так как есть вероятность наличия в модели ПО нескольких сетей второго рода, необходим механизм, реализующий логические условия прохождения этих сетей. ПО может иметь несколько характерных действий во время работы, однако не все эти характерные действия могут проявляться одновременно за один жизненный цикл. В таком случае эксперт должен описать с помощью логических операторов ИЛИ/И возможные комбинации проявления характерных действий программы. Если данные об этом недоступны, по умолчанию выбирается логический оператор ИЛИ для всех характерных действий.

Вектор работы каждого процесса в системе сопоставляется со всеми имеющимися моделями. Так, если в системе запущено 20 процессов, а в базе имеются 40 моделей ПО, будет произведено 800 (20 * 40) анализов соответствия работы процессов какой-нибудь имеющейся модели. Среди этого множества результатов нас интересуют события только двух типов: несоответствие процесса своей модели и соответствие процесса чужой модели. Первый вид событий говорит об изменении ПО. Так как модель ПО разбита на фазы работы, можно говорить о том, в какой именно фазе произошло отклонение. Второй вид событий сигнализирует о попытке замаскировать один процесс под другой.

Следует отметить, что возможны случаи прерванной работы ПО, например, на стадии запуска. В таком случае вектор, описывающий прерванный запуск, даст низкую вероятность соответствия нейронной сети для запуска. Это будет расценено как аномальное поведение, тогда как в реальности не было никаких аномалий. Для таких случаев можно ввести переменную, хранящую время работы ПО. Если значение этой переменной меньше активационного периода нейронной сети, то мы не можем доверять выходу этой сети.

Заключение

Предложенный механизм обнаружения аномалий с помощью нейронных сетей имеет ряд преимуществ и недостатков. Преимуществом является то, что средства моделирования процесса не зависят от операционной системы или платформы, на которой запускается ПО. Тем самым, построив профиль поведения некоторого ПО один раз, его можно использовать для любых СВТ, на которых работает смоделированное ПО. Также с применением некоторой экспертной оценки для любого ПО может быть подобран свой набор координат, учитывающий особенности

данного ПО, и эти особенности будут заложены в эталонную модель поведения, что значительно повысит ее эффективность. Кроме того, предоставленный механизм построения эталонных профилей поведения учитывает работу ПО на протяжении всего жизненного цикла. Немаловажно, что размеры получаемых сетей относительно небольшие (до 150 нейронов), поэтому обработка вектора реального поведения ПО при современных вычислительных мощностях будет производиться почти моментально. Т. е. обнаружение аномалии происходит почти сразу же после того, как она произошла. Учитывая, что нейронных сетей в одном профиле и самих профилей может быть много, при необходимости возможно дополнительно ускорить вычислительные возможности сетей [8].

Основным недостатком на данный момент является процесс обучения, а именно стадия сбора обучающих векторов. При размерности вектора около 40 координат число обучающих векторов для нейронной сети любого рода согласно формуле 2.3 должно быть порядка 10000, однако такое количество учебных прогонов ПО не всегда возможно (и нужно) осуществить. Как показала практика, число обучающих векторов может быть на порядки меньше [7], так как основная задача этого набора заключается в том, чтобы описать как можно больше различных вариантов запуска (например, с различными вариантами аргументов запуска), а не один и тот же вариант запуска много раз. Тем не менее автоматизированное построение профиля поведения ПО потребует некоторого времени (до нескольких часов), а также экспертной оценки и, желательно, дополнительных знаний о функциональности ПО.

СПИСОК ЛИТЕРАТУРЫ:

1. *Ясницкий Л. Н.* Введение в искусственный интеллект. М.: Издательский центр «Академия», 2005.
2. *Swingler K.* Applying Neural Networks. A Practical Guide. Morgan Kaufmann; Pap/Dsk edition, 1996.
3. *Дубровин В. И., Субботин С. А.* Алгоритм ускоренного обучения персептронов // Сборник трудов IV Всероссийской научно-технической конференции «Нейроинформатика-2002». М.: МИФИ, 2002.
4. *Harris Drucker, Yann Le Cun.* Improving Generalization Performance Using Backpropagation // IEEE Transactions on Neural Networks. 1992. Vol. 3. № 5.
5. *Malki H. A., Moghaddamjoo A.* Using the Karhunen-Loe`ve Transformation in the Back-Propagation Training Algorithm // IEEE Transactions on Neural Networks. 1991. Vol. 2. № 1.
6. *Paul J. Werbos.* Backpropagation Through Time: What It Does and How to Do It // Artificial Neural Networks: Concepts and Theory. IEEE Computer Society Press, 1992.
7. *Alain Petrowski, Gerard Dreyfus, Claude Girault.* Performance Analysis of a Pipelined Backpropagation Parallel Algorithm // IEEE Transactions on Neural Networks. 1993. Vol. 4. № 6.
8. *Крисилов В. А., Олешко Д. Н., Лобода А. В.* Методы ускорения нейронных сетей // Вестник СевГТУ. Информатика, электроника, связь. Одесса, 2001. Вып. 32. С. 19.