выработка единых для всех разработчиков стандартов форматирования. Несмотря на это, с высокой вероятностью различные проекты будут иметь отличные стандарты форматирования, что вынудит разработчика всякий раз перестраиваться при работе над очередным проектом.

Альтернативой вышеназванному подходу может служить подход, основанный на использовании ассоциированных с каждым репозиторием распределённой системы контроля версий (будь то локальным для каждого из разработчиков или удалённым) специфичных опций форматирования. В процессе копирования репозитория разработчик может задать предпочитаемый стиль форматирования, отличный от такового для исходного репозитория. Каждая ревизия полученных из исходного репозитория файлов форматируются в соответствии с заданными предпочтениями.

В процессе внесения правок в исходный код, в локальном репозитории разработчик руководствуется индивидуальным стилем форматирования. При публикации правок в удалённый репозиторий, выполняется форматирование файлов в соответствии с настройками удалённого репозитория, и сохранение в таком виде под версионным контролем. Таким образом, семантически эквивалентные файлы с исходным кодом имеют различное форматирование в локальном для каждого из разработчиков репозитории, но единое в централизованном, что позволяет эффективно применять diff-подобные утилиты для выявления изменений в логике работы исходного кода.

Ограничением предложенного подхода можно считать его несовместимость с системами контроля версий, основанными на хранении записей об изменении содержимого файлов (Mercurial), а не самих изменённых файлов (Git).

Список использованных источников:

1. Buse R. P. L., Weimer W. R. A metric for software readability // Proceedings of the 2008 international symposium on Software testing and analysis. – Seattle, WA, USA: ACM, 2008. – С. 121-130.

2. Miara R. J., Musselman J. A., Navarro J. A., Shneiderman B. Program indentation and comprehensibility // Commun. ACM. – 1983. – Т. 26, № 11. – С. 861-867.

3. Jackson D., Ladd D. A. Semantic Diff: a tool for summarizing the effects of modifications // Software Maintenance, 1994. Proceedings., International Conference on – 1994. – С. 243-252.

АВТОМАТИЧЕСКОЕ ПОСТРОЕНИЕ АРХИТЕКТУРЫ НЕЙРОННОЙ СЕТИ ПРИ ПОМОЩИ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Попитич А. Я.

Серебряная Л. В. – к-т. техн. наук, доцент

В ходе решения задач, где необходимо использование нейронной сети, всегда присутствует проблема выбора конкретной архитектуры сети. Эта проблема в большинстве случаев решается простым подбором и подгоном параметров по причине разнообразия анализируемых данных и уникальности решаемых задач. В данном докладе будет предложено применение генетического алгоритма, который позволяет автоматизировать процесс и сгенерировать сеть, которая будет соответствовать анализируемым данным и решаемой задачи.

Основная задача была поставлена следующим образом – обеспечить построение топологии нейронной сети автоматическим образом. В качестве исходных данных была взята обучающая выборка для нейронной сети, которая в дальнейшем будет применяться для обучения каждого, генерируемого генетическим алгоритмом, экземпляра нейронной сети. Все эксперименты для простоты были организованы для решения небольшой задачи XOR, результаты которой могут быть впоследствии применены и для решения более масштабных задач.

Для применения генетического алгоритма в ходе решения поставленной задачи было использовано прямое кодирование связей нейронной сети. На рисунке 1 представлен пример небольшой нейронной сети и кодирования межнейронных связей.

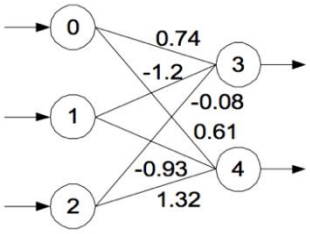

Рис. 1 – Пример топологии нейронной сети

Результат кодирования – геном, который представлен в таблице 1, где каждый ген в геноме содержит информацию об связующих нейронах и значения веса связи.

0	3	1	3	2	3	0	4	1	4	2	4
0.74		-1.2		-0.08		0.61		-0.93		1.32	

При этом были введены следующие правила при формировании новых и удалении имеющихся нейронов в ходе работы генетического алгоритма:

– все нейроны имеют определенный индекс. Удаление нейронов входного или выходного слоев не допускается;

– новые нейроны получают минимальный из возможных номеров;

– при удалении нейрона индексы остальных нейронов корректируются с учетом удаленного.

Оператор скрещивания для генетического алгоритма использует две особи (генома нейронных сетей) и в результате получает два потомка. Общие нейроны и связи наследуются между потомками, а значения связи в сетях потомков формируются при помощи двухточечного кроссовера. Различающиеся элементы разыгрываются между двумя потомками. На рисунке 2 представлен пример процесса скрещивания.

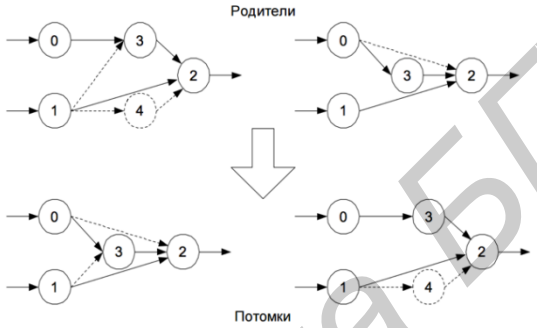

Рис. 2 – Пример процесса скрещивания двух топологий

Оператор мутации представляет собой композицию из нескольких операторов мутации: добавление нейрона в скрытый слой, удаление случайного нейрона вместе с его связями, добавление связей между нейронами, удаление связи между нейронами и изменение веса случайно выбранной связи.

Основной проблемой в случае применения мутации являлось бесконтрольное добавление и удаление связей, что в большинстве случает приводило к нарастанию большого числа связей или наоборот – “обеднению” связей между нейронами. Для преодоления такого негативного эффекта были использованы специальные метрики, которые оценивали текущее состояние топологии и помогали алгоритму выбрать нужный оператор мутации.

Первый коэффициент – коэффициент связности нейронов. Чем больше данный коэффициент, тем более насыщенной связями является топология. Данный коэффициент рассчитывается формулой 1:

$$f_c = \frac{N_c}{2^{FB-1}[N_n\ N_n - 1\ - N_i\ N_i - 1\ -\ 1 - FB\ N_o\ N_o - 1\ ]} \quad (1)$$

где Nc – количество связей в сети, Ni,No,Nn – количество входных, выходных и общее количество нейронов в сети

Второй коэффициент fn вычисляется на предположении, чем больше нейронов присутствует на входных и выходных слоях в сумме, тем более сложная сеть (с большим количеством нейронов) необходима сеть (формула 2):

$$f_n = (N_i + N_o)/N_n \quad (2)$$

Чем больше нейронов в сети, тем меньше будет коэффициент и тем меньше вероятность выбора мутации по добавлению нового нейрона.

Полученные в результате скрещивания и мутации потомки оцениваются при помощи фитнес-функции. В качестве фитнес функции взята формула вычисления среднеквадратичной ошибки в ходе обучения сети на обучаемых данных.

По результатом данной функции можно определить именно те нейронные сети, которые наиболее приспособлены к данным для обучения.

Для тестирования разработанного алгоритма была решена задача XOR, которая является частным случаем n-битной задачи четности, когда для последовательности из n-бит необходимо определить четность единичных разрядов (1 – нечетно, 0 - четно). Изначально все особи представляли собой одинаковые сети с входным и выходным слоем, без нейронов промежуточного слоя.

В результате применение генетического алгоритма была получена следующая топология сети (рисунок 3).

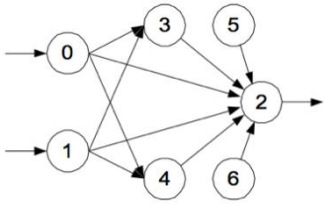

Рис. 3 – Пример автоматически сгенерированной топологии сети

Минимальная сеть, которая способна решать поставленную задачу, состоит из 4-х нейронов и 5-и связей, однако стоит отметить, что полученная сеть была сгенерирована автоматическим образом, что является неплохим результатом

Для дальнейшего улучшения результатов автоматического построения можно провести дополнительные эксперименты по выбору способа кодирования топологии сети. Для этого можно применить косвенный способ кодирования, где строятся определенные грамматики топологических структур или кодируются отдельные блоки.

Список использованных источников:

1. N.J. Radcliffe, Genetic Neural Networks on MIMD computers / Radcliffe N.J., Davis L // University of Edinburgh, Edinburgh, England, 1990 – 65p.
2. K. O. Stanley, Evolving Neural Topologies through Augmenting Topologies / Stanley K. O., Miikkulainen R. // Evolutionary Computation, The MIT Press, 2002. – № 10(2). – p. 99-127.
3. J. R. Koza, Genetic generation of both the weight and architecture for a neural network / Koza J. R., Rice J. P. // International Joint Conference on Neural Networks. 1991. – Vol. 2, P. 397–404.

ЭВОЛЮЦИОННЫЙ АЛГОРИТМ ГЕНЕРАЦИИ ТЕСТОВЫХ СЦЕНАРИЕВ JSON ВЕБ-СЕРВИСОВ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Проведенцев Е. С.

Скобцов В. Ю. – кандидат технических наук

В практике современных информационных технологий и программной инженерии широкое распространение получили веб-сервисы, принимающие в качестве параметров данные в JSON формате. Запросы и ответы подобных сервисов могут быть описаны с помощью JSON-schema.

Для функционального тестирования JSON-сервисов зачастую применяется ручная разработка тестовых сценариев. Автоматизация разработки тестового сценария для JSON-сервиса осложнена, большим количеством входных и выходных параметров, а также возможными зависимостями логики срабатывания одного параметра от другого [1]. Для решения трудно формализуемых задач большой размерности широкое применение получили эволюционные алгоритмы, являющиеся одним из перспективных направлений исследований в области интеллектуальных систем, задач поиска, интеллектуальной обработки данных [2]. В частности, эволюционные алгоритмы эффективно используются в области автоматизации тестирования программного обеспечения [3]. Поэтому в данном исследовании предлагается эволюционный алгоритм генерации тестовых сценариев для подобных веб-сервисов.

В данной работе предлагается алгоритм, использующий JSON-schema для автоматизированного создания тестового сценария JSON-сервиса. Целью данного алгоритма является генерация тестового сценария, обнаруживающего максимальное количество ошибок и, в то же время, содержащего минимальное количество тестовых случаев (test case). В основе данного алгоритма лежит представление тестового случая в качестве экземпляра, каждое поле запроса, которого представляет собой отдельный ген.

Алгоритм работает следующим образом:

1. Генерируется множество-популяция случайных запросов, заполненных в соответствии с предоставленной JSON-схемой. Наполнение запроса должно быть максимальным, то есть каждому полю, по возможности, должно быть присвоено значение. В зависимости от типа поля выбирается стратегия заполнения. Для числовых типов могут выбираться граничные значения, для строковых – значения из предопределенного списка, который составляется в зависимости от специфики программного кода сервиса. Цель данного запроса – задействовать как можно больше ветвей кода, повысив вероятность выявления ошибки. В случае, если данный пункт выполняется не первый раз, то необходимо убедиться, что запрос не