УДК 004.032.26

ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ ДЕКОМПОЗИЦИЯ ПРОГРАММНОЙ ЛОГИКИ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

Р.В. Гирин[1], С.П. Орлов[2]

[1] *ООО «Интеллект Софт», Самара, Россия*
romangirin@gmail.com
[2] *Самарский государственный технический университет, Самара, Россия*
orlovsp1946@gmail.com

Аннотация

В статье описана декомпозиция логики программной реализации искусственных нейронных сетей с целью её последующей реализации в виде набора слабосвязанных доменных классов. Рассмотрены структурная и функциональная декомпозиция логики. Методика декомпозиции программной логики иллюстрируется UML-диаграммами. Приведено краткое описание практической реализации программной логики искусственной нейронной сети на языке C#, выполненной на основе объектно-ориентированного подхода. Подобная реализация использована при проектировании нейронных сетей различной конфигурации с целью их последующего обучения, эксплуатации, а также проведения экспериментов. Предлагаемое разделение программной логики между классами и обеспечение слабой связанности между ними упрощает процесс поиска ошибок в программном коде, делает код более управляемым, повышает продуктивность разработчика. Приведённые в статье примеры демонстрируют, как рациональная декомпозиция программной логики нейронной сети на семантически независимые между собой блоки в сочетании с применением концепции «Внедрение зависимостей», способствует большей структурированности кода. Новым результатом является применение объектно-ориентированной декомпозиции к программной логике нейронной сети, что позволило значительно упростить процесс проектирования программного кода. Приведён пример тестирования декомпозированной логики нейронной сети. Предлагаемый подход к декомпозиции программной логики искусственных нейронных сетей можно применить к широкому спектру различных нейронных сетей, таких, как конволюционные и деконволюционные сети, сети, содержащие одновременно конволюционные и полносвязаннные слои нейронов. При этом единообразие реализации сетей различных видов обуславливает упрощение их понимания, снижает трудовые затраты на сопровождение и развитие реализующего их кода.

Ключевые слова: *искусственная нейронная сеть, системный анализ, декомпозиция, объектно-ориентированный анализ.*

Цитирование: *Гирин, Р.В.* Объектно-ориентированная декомпозиция программной логики искусственных нейронных сетей / Р.В. Гирин, С.П. Орлов // Онтология проектирования. – 2018. – Т. 8, №1(27). - С.110-123. – DOI:10.18287/2223-9537-2018-8-1-110-123.

Введение

Использование искусственных нейронных сетей (ИНС) стало прорывом во многих областях машинного обучения и искусственного интеллекта [1-3]. В этой связи актуальна разработка различных подходов к проектированию ИНС. Во многих проектах необходимо наличие программной библиотеки, предоставляющей ИНС таким образом, чтобы можно было гибко варьировать и расширять её функциональность. Это, в свою очередь, достижимо только в случае рационального проектирования программной логики ИНС. В этом плане задачи реализации ИНС не отличаются от какого-либо другого проекта, связанного с разработкой программного обеспечения [4].

Известно, что при выполнении проектирования программного обеспечения значительное упрощение достигается путем декомпозиции всей совокупности реализуемой функциональности на её составляющие. В некоторых источниках подобную выделенную функциональность, подлежащую реализации в виде отдельного слабо связанного компонента, условно называют логикой [4], например, логика доступа к данным (англ. data access logic) и т.п. В данной статье под программной логикой ИНС понимается вся совокупность функциональностей, выполняющих обработку данных на слоях нейронной сети. Следует отметить, что программная логика – это не логическое программирование, парадигма которого основана на автоматическом доказательстве теорем.

В настоящее время разработано большое число программных реализаций нейронных сетей. При этом во многих реализациях получены эффективные решения, которые целесообразно использовать в дальнейших разработках. В тоже время отсутствует методологический подход, который бы опирался на эффективные приёмы объектно-ориентированного проектирования и использовал накопленные знания в области ИНС.

В предлагаемой статье развивается объектно-ориентированный подход к анализу программной логики ИНС [5-7]. Совокупность реализаций ИНС, доступных в исходных кодах (open sours) на различных языках, можно условно разделить на две большие группы. К одной группе относятся реализации специализированной платформы для алгоритмов машинного обучения, в их числе и ИНС. В другую группу входят библиотеки, которые не используют какие-либо сторонние платформы, и именно они рассматриваются в данной публикации. Выполненная авторами имплементация (практическая реализация) ИНС также не использует сторонние платформы для алгоритмов машинного обучения, и выполнение кода осуществляется непосредственно на CLR (*Common Language Runtime* — общеязыковая исполняющая среда) [8].

В силу актуальности тем, связанных с использованием ИНС, в настоящее время существует большое число примеров их исходных кодов. Многие из программных библиотек, доступных в исходных кодах, например [4, 9], выделяют класс нейронной сети, на который возложено хранение перечня слоев. Этот класс инкапсулирует логику взаимодействия данных слоев для обеспечения прямого и обратного прохождения сигнала. Некоторые варианты реализации добавляют в логику нейронной сети функциональность, обеспечивающую сбор и обработку основной статистики, накапливаемой при обучении нейронной сети. Интересно отметить, что с определённой точки зрения подобное совмещение может противоречить принципу единственности ответственности.

Реализация ИНС в программной библиотеке Caffe (*Convolution Architecture For Feature Extraction* - свёрточная архитектура для распознавания признаков) [10], доступная в исходном коде на языке C++, содержит богатую палитру различных имплементаций слоёв нейронов ИНС. Эти слои удобно взаимно заменять, получая новую функциональность ИНС, обеспеченную набором слоёв с заданными характеристиками, такими как: функция активации, принцип, по которому сигнал корректируется весом синоптической связи, и др.

Известно, что нейронные сети широко варьируются по своей структуре, принципу передачи сигнала между слоями, применяемым функциям активации и другим характеристикам [11]. Программная библиотека, описанная в [12], обеспечивает реализации ИНС на языке Java при едином технологическом подходе к сетям разной структуры.

1 Структурная и функциональная декомпозиция программной логики ИНС

Рассматривая всю совокупность логики ИНС, можно выделить следующие основные структурные компоненты.

- *Функция активации.* Она может быть представлена в виде сущности, инкапсулирующей логику, связанную с функцией активации. Эта сущность содержит данные о параметрах функции активации и логику вычисления значений функции и её производной.
- *Функция ошибки.* Эта сущность применяется при обучении ИНС. Для тренировки ИНС применяет подход, который в области машинного обучения именуется «Обучение с учителем». В ходе этого процесса вместе с входным сигналом доступны сведения об ожидаемом выходном сигнале, который можно сравнить с фактически полученным сигналом. Оценку величины расхождения эталонного и фактического результата оценивают с использованием функции ошибки для определения величины, на которую необходимо скорректировать веса синаптических связей.
- *Слой нейронов ИНС.* Он может быть представлен в виде сущности, инкапсулирующей логику работы совокупности искусственных нейронов, находящихся на одном слое ИНС. Он обеспечивает хранение значений весов синаптических связей и содержит логику вычисления поля индуцирования (передаваемого функции активации) на основании входного сигнала. Поскольку данная логика использует функцию активации, каждый слой содержит хотя бы один объект функции активации.
- *Нейронная сеть.* Сеть может быть представлена в виде совокупности одного или нескольких слоев искусственных нейронов вместе с логикой их взаимодействия (передача сигнала в процессе его прямого и обратного прохождения, хранение значений предыдущих сигналов при необходимости и т.п.). Нейронная сеть для своего функционирования должна содержать хотя бы один слой нейронов.

В то же время, рассматривая функциональную организацию логики ИНС, можно выделить следующие уровни.

- *Логика прямого прохождения сигнала.* Она представляет собой основной режим работы нейронной сети, когда сигнал, поданный на вход ИНС, передаётся на вход функции активации по синаптическим связям с корректировкой на их веса. При этом функция активации, в свою очередь, формирует выходной сигнал отдельно взятого слоя нейронов или ИНС в целом. В конволюционных сетях [13] все нейроны, расположенные на одном слое, используют одинаковые синаптические связи. Это необходимо учитывать и в логике прямого прохождения сигнала, и в логике корректировки весов при обучении.
- *Логика обратного прохождения сигнала ошибки.* Эта функциональность применяется для обучения ИНС [13]. При обратном прохождении сигнала выходные значения ИНС сравниваются с эталонным сигналом (который доступен при обучении с учителем) и их расхождение оценивается с использованием функции ошибки. Полученная ошибка на выходном слое передаётся на предшествующий слой и далее до входного слоя.
- *Логика корректировки весов и сдвигов нейронной сети.* Подобная корректировка необходима для обучения нейронной сети. Удобно, когда эта логика гибкая и взаимозаменяемая. При этом частная производная функции ошибки используется для определения непосредственно величины, на которую корректируется вес связи или сдвиг. При необходимости имплементация логики ИНС должна выполнять замену используемой функции ошибки.
- *Вспомогательная логика*. К данной функциональной части можно отнести логику, выполняющую матричные операции, к которым сводится большинство численных преобразований, выполняемых внутри нейронной сети. Например, при использовании концепции глубокого обучения [14] необходимо осуществить «предобучение» каждого отдельного слоя заданной нейронной сети на базе «Обучение без учителя», с последующим обучением всей ИНС на основе «Обучение с учителем». На стадии предобучения требуется логика, выполняющая анализ структуры заданной нейронной сети, генерацию автоэнкодеров

для предобучения каждого слоя и само предобучение со сбором статистики. С целью соблюдения принципа единственности ответственности эта логика инкапсулирована в отдельном классе.

Дополняя описанную выше функциональную декомпозицию программной логики работы ИНС, рассмотрим UML - диаграммы последовательности, приведённые на рисунках 1 и 2.

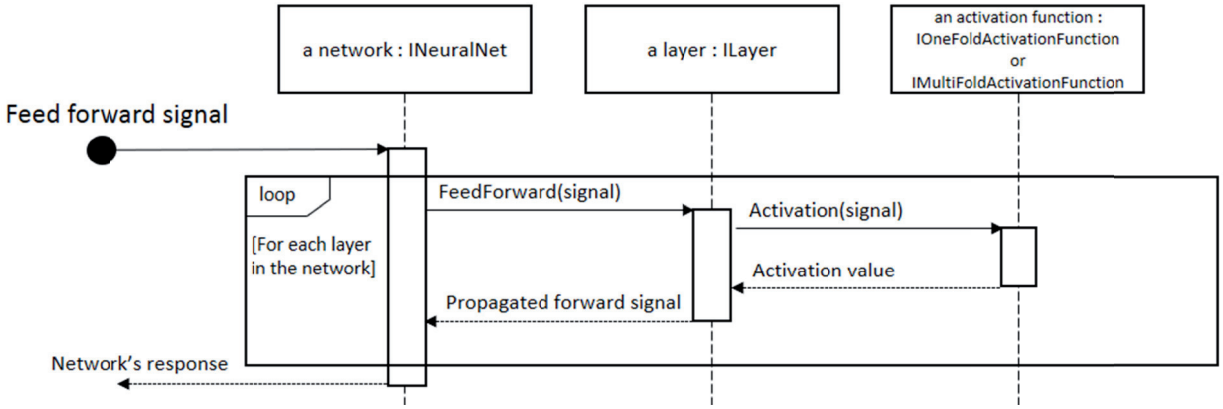

Рисунок 1 - UML-диаграмма последовательности, иллюстрирующая переход управления при прямом прохождении сигнала в ИНС

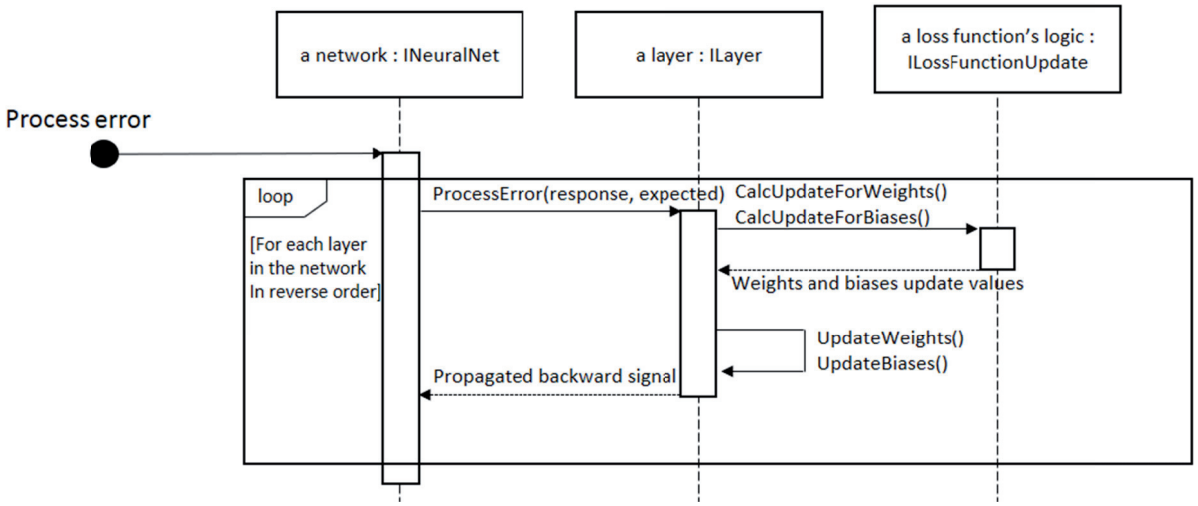

Рисунок 2 - UML-диаграмма последовательности, иллюстрирующая переход управления при обратном прохождении сигнала в ИНС

На приведённых диаграммах легко видеть взаимосвязь между выделенными в ходе функциональной декомпозиции совокупностями функциональностей. При прямом прохождении сигнала (см. рисунок 1) ИНС последовательно делегирует задачу прямой обработки сигнала каждому из входящих в неё слоёв нейронов. Каждый из слоёв нейронов, в свою очередь, делегирует объекту класса функции активации задачу вычисления её значения. При обратном прохождении сигнала (см. рисунок 2) ИНС последовательно делегирует задачу обратной обработки сигнала каждому из входящих в неё слоёв нейронов. Переход управления от слоя к слою в этом случае выполняется, начиная с последнего и заканчивая первым.

В некоторых технических решениях требования таковы, что необходимо использовать уже обученную нейронную сеть, без последующей её корректировки и дообучения. Для обеспечения эксплуатации нейронной сети в таких условиях важно выполнить группировку функционала, составляющего ИНС, в двух вариантах: функционал, необходимый для эксплуатации и функционал, необходимый для обучения.

Рассматривая выполненную выше структурную декомпозицию, легко заметить, что для проектирования и эксплуатации ИНС достаточно иметь в распоряжении функцию активации, один или несколько слоёв нейронной сети и сущность, соответствующую самой нейронной сети. Сущность, соответствующая функции ошибки, требуется только в процессе обучения ИНС.

Функциональная декомпозиция показывает, что для работы ИНС достаточно иметь логику, обеспечивающую прямое прохождение сигнала, в совокупности с некоторой долей вспомогательной логики (например, логики, выполняющей матричные операции). Логика обратного прохождения сигнала и логика корректировки весов синаптических связей требуются только в процессе обучения ИНС.

Используя описанный подход при объектно-ориентированном проектировании, будем разделять логику ИНС на две группы классов:

- классы, инкапсулирующие логику обучаемых ИНС и их компонентов;
- классы, инкапсулирующие логику необучаемых ИНС и их компонентов.

Важно заметить, что обучаемые ИНС в себе содержат логику необучаемых, поэтому эти классы связаны наследованием.

В приводимой декомпозиции выделяется логика нейронов, находящихся на отдельно взятом слое ИНС. Дальнейшая декомпозиция на уровни логики отдельно взятых нейронов не выполняется в силу следующих соображений. На практике в структуре нейронных сетей на отдельно взятом слое используются нейроны схожей конфигурации. Учитывая это, арифметические операции, выполняемые на отдельно взятом слое ИНС, можно выразить и реализовать в коде в виде матричных операций. Некоторые матричные операции, такие как умножение, достаточно ресурсоёмки и обладают большой асимптотической сложностью [15], однако при этом они обладают алгоритмическим параллелизмом. Кроме того, такие вычисления можно выполнять на графических процессорах GPU (*Graphics Processing Unit*) в рамках концепции GPGPU (*General-purpose computing for GPU* - неспециализированные вычисления на графических процессорах) [12, 16].

2 Технология проектирования программного кода логики ИНС

При проектировании описываемой реализации ИНС использовались следующие основные принципы:

- функциональность отдельно взятой выделенной сущности должна представлять собой *логическую целостность* для удобства чтения и понимания кода;
- совокупность выделенных сущностей должна обладать достаточной степенью *слабой связанности* для обеспечения возможности независимо развивать, тестировать и использовать различные сущности;
- каждый отдельно взятый класс, инкапсулирующий какую-либо логику, должен соответствовать принципу *единственной ответственности*, то есть быть ответственным за решение какой-либо отдельно взятой задачи и только её.

Следование этим принципам нацелено на проектирование кода так, чтобы заложенные концепции позволяли удобно компоновать различные ИНС и расширять их функционал (добавление новых функций активации, функций оценки ошибки, поддержка различных конфигураций ИНС и т.п.) [5].

На основании проведённых выше анализа и декомпозиции программной логики ИНС была выполнена её имплементация с использованием языка C#. На рисунке 3 приведена UML-диаграмма, которая даёт хорошее представление о перечне и связях, используемых в какой-либо программной библиотеке классов.

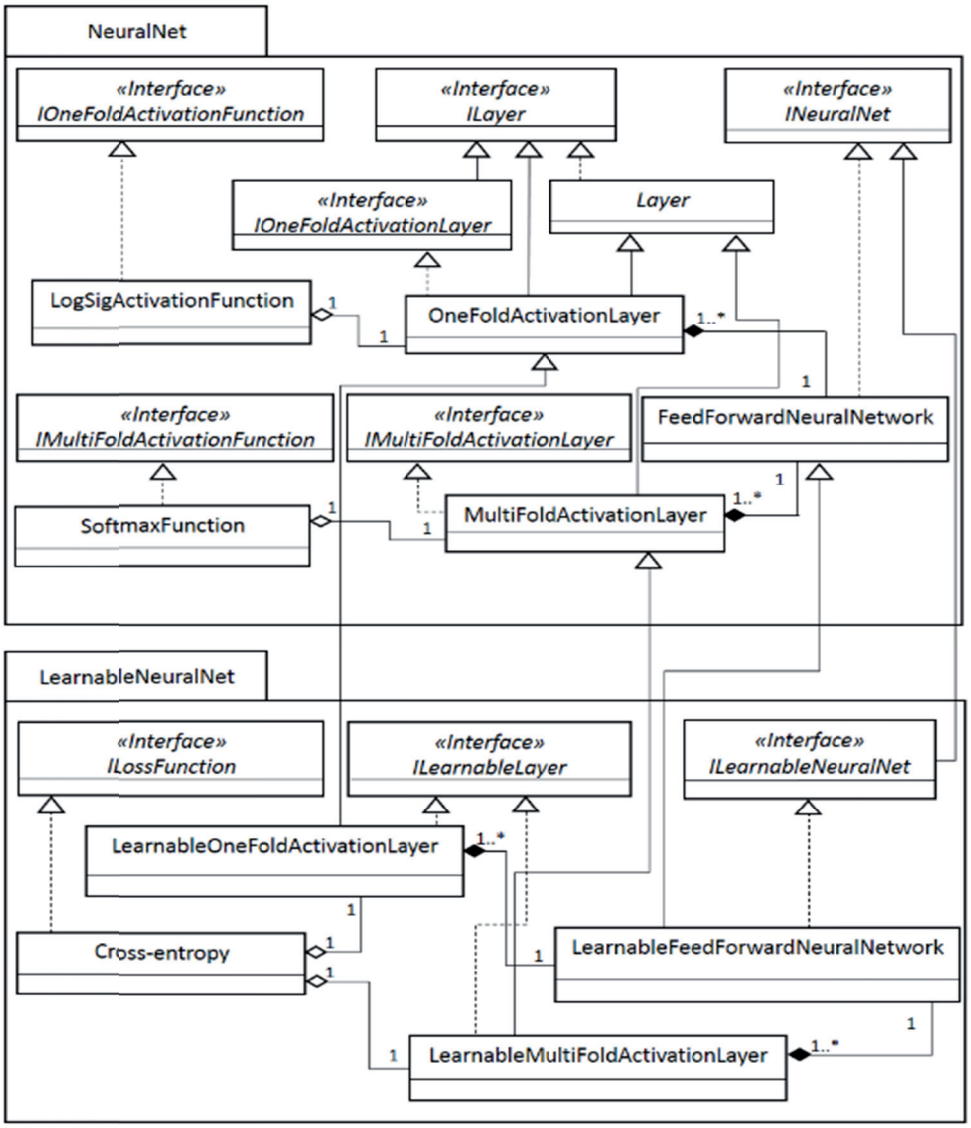

Рисунок 3 - UML-диаграмма классов, иллюстрирующая
объектно-ориентированную декомпозицию программной логики ИНС

На диаграмме легко выделить два пакета. В пакете, расположенном в верхней части изображения, содержатся интерфейсы и классы, обеспечивающие прямое прохождение сигнала в ИНС. В нем расположены интерфейсы, объявляющие специфичные члены для многосвёрточных и односвёрточных функций активации, *IMultiFoldActivationFunction* и *IOneFoldActivationFunction*, соответственно. Для наглядности представлена одна реализация каждого из этих интерфейсов (классы *LogSigActivationFunction* и *SoftmaxFunction*). В действительности их может быть больше в зависимости от числа поддерживаемых функций активации. Интерфейс *ILayer* объявляет перечень членов, специфичных для отдельно взятого слоя нейронов. Его непосредственной реализацией является абстрактный класс *Layer*, который задаёт базовую реализацию, общую как для слоёв содержащий многосвёрточную функцию активации, так и односвёврточную. Наследники класса *Layer*, классы *MultiFoldActivationLayer* и *OneFoldActivationLayer*, содержат реализацию логики многосвёрточных и односвёрточных слоев. Интерфейс *INeuralNet* объявляет члены, присущие непосредственно ИНС. Его реализацией является класс *FeedForwardNeuralNetwork*, название которого говорит само

за себя – он реализует функциональность сети, выполняющей прямое прохождение сигнала. Разумеется, можно добавить реализацию и других нейронных сетей посредством реализации интерфейса *INeuralNet*. Каждый класс ИНС, и в частности, *FeedForwardNeuralNetwork*, образует композицию с одним или более слоями нейронов.

В пакете, расположенном в нижней части диаграммы, находятся интерфейсы и классы, обеспечивающие обучение ИНС. Интерфейс *ILossFunction* объявляет члены, присущие функциям потерь. Приведена одна из реализаций данного интерфейса – класс *Cross-Entropy*. Интерфейсы *ILearnableLayer* и *ILearnableNeuralNet* объявляют перечень членов, необходимых для обучения отдельно взятого слоя и сети в целом, соответственно. Классы *LearnableMultiFoldActivationLayer* и *LearnableOneFoldActivationLayer* – потомки *MultiFoldActivationLayer* и *OneFoldActivationLayer*, соответственно. Они добавляют своим предкам реализацию *ILearnableLayer*, а класс *LearnableFeedForwardNeuralNetwork* добавляет своему предку *FeedForwardNeuralNetwork* реализацию *ILearnableNeuralNet*.

Приведённая диаграмма более детально отражает описанную выше структурную декомпозицию логики ИНС. Использованы два интерфейса, описывающих классы, содержащие логику вычисления значений функции активации, *IOneFoldActivationFunction* и *IMultiFoldActivationFunction* (см. рисунок 4).

Совокупность одного или нескольких слоёв нейронов образует собой ИНС. Интерфейс, соответствующей ей, представлен на рисунке 5.

```
public interface IOneFoldActivationFunction
{
   double Activation(double parameter);
   double Derivative(double parameter);
}
public interface IMultiFoldActivationFunction
{
   double[] Activation(double[] parameters);
   double[,] Derivative(double[] parameters);
}
```

Рисунок 4 – Интерфейсы *IOneFoldActivationFunction* и *IMultiFoldActivationFunction*

```
public interface INeuralNet
{
   double[] ProcessInput(double[] input);
   IEnumerable<ILayer> Layers { get; }
}
```

Рисунок 5 - Интерфейс *INeuralNet*

Поскольку слой нейронов ИНС обладает значительной частью логики, которая является общей для многих разновидностей слоёв нейронных сетей, она вынесена в базовый интерфейс *ILayer.* Такая практика выделения базовых сущностей на основании анализа признаков общности и различности необходима, чтобы не дублировать код [6, 18]. Наследниками данного базового интерфейса являются классы *OneFoldActivationLayer* и *MultiFoldActivationLayer.*

Как видно из названий этих классов, они декларируют только те члены, которые необходимы для реализации логики слоя нейронной сети в той части, где они различны в случае применения односвёрточной и многосвёрточной функции активации. Интерфейсы *ILayer, IOneFoldActivationLayer* и *IMultiFoldActivationLayer* приведены на рисунке 6.

```
public interface ILayer
{
   double[] Biases { get; }
   double[,] Weights { get; }
   int NumberOfInputs { get; }
   int NumberOfNeurons { get; }
   double[] FeedForward(double[] inputVector);
   //некоторые члены интерфейса опущены для краткости
}
public interface IOneFoldActivationLayer : ILayer
{
   IOneFoldActivationFunction OneFoldActivationFunction { get; }
}
public interface IMultiFoldActivationLayer : ILayer
{
   IMultiFoldActivationFunction MultiFoldActivationFunction { get; }
}
```

Рисунок 6 – Интерфейсы ILayer, IOneFoldActivationLayer и IMultiFoldActivationLayer

Следование принципу разделения объявления и реализации программных сущностей позволяет в дальнейшем гибко изменять функциональность путём использования разных реализаций заданных интерфейсов. Имплементация сформированного набора интерфейсов выполнена следующим образом.

Одна из реализаций интерфейса *IOneFoldActivationFunction* – это класс *LogSigActivationFunction*, который представлен на рисунке 7.

```
   public class LogSigActivationFunction :
IOneFoldActivationFunction
   {
      public double Activation(double parameter)
      {
         return 1.0 / (1.0 + Math.Pow(Math.E, (-1.0 *
parameter)));
      }
      public double Derivative(double parameter)
      {
         double e = 1.0 / (1.0 + Math.Pow(Math.E, (-1.0 *
parameter)));
         return e * (1.0 - e);
      }
   }
```

Рисунок 7 - Класс *LogSigActivationFunction*

Логика этого класса достаточно проста, поэтому паттерн «Внедрение зависимостей» (*Dependency injection*) [19, 20] в данном случае не применяется. Однако этот класс служит примером, образно говоря, начального элемента цепочки зависимостей между доменными классами рассматриваемой программной библиотеки.

В классах, имплементирующих интерфейсы *IOneFoldActivationFunction* и *IMultiFoldActivationFunction*, применённая декомпозиция позволяет сосредоточиться только на логике вычисления значений функции активации в зависимости от переданных парамет-

ров и вычислении значений производной данной функции. Это полностью соответствует принципу единственной ответственности для отдельного взятого класса.

Следующим логическим блоком в рассматриваемой логике является слой нейронов ИНС. Класс *OneFoldActivationLayer* показан на рисунке 8.

```
   public class OneFoldActivationLayer : Layer,
IOneFoldActivationLayer
   {
      public OneFoldActivationLayer(double[,] weights,
double[] biases, IOneFoldActivationFunction af)
         : base(weights, biases)
      {
         //имплементация конструкторов не приведена
для краткости
      }
      public IOneFoldActivationFunction
OneFoldActivationFunction
      {
         get
         {
            return oneFoldActivationFunction;
         }
      }
      //другие члены класса не приведены для
краткости
   }
```

Рисунок 8 - Класс *OneFoldActivationLayer*

Видно, что один из параметров конструкторов данного класса – это реализация *IOneFoldActivationFunction.* Реализация этого класса выполняет делегирование соответствующих вычислений переданной реализации. Для краткости имплементация класса *Layer* не приводится, но в ней содержится общая базовая функциональность, специфичная для слоя нейронов ИНС.

Аналогичным образом зависимость внедряется и для связки нейронной сети в целом, и составляющих её слоёв нейронов. Конструктор класса принимает в качестве параметра перечень слоёв, входящих в состав нейронной сети, а впоследствии делегирует выполнение прямой передачи сигнала от слоя к слою, обеспечивая прямую передачу сигнала по сети в целом.

Таким образом, вся логика ИНС, декомпозированная описанным способом, объединена в общую функциональность посредством концепции «Внедрение зависимостей» (рисунок 9).

Совокупность функциональностей, соответствующая логике обучения ИНС, реализована следующим образом. Выделен интерфейс *ILossFunction.* Различные имплементации данного интерфейса позволяют привнести в описываемую библиотеку использование различных функций потерь. Посредством объекта, имплементирующего данный интерфейс, определяются значения, на которые корректируются значения весов и сдвигов.

Независимо от того, какая имплементация слоя нейронов используется (одна из описанных выше или добавленная в ходе доработки библиотеки), слой нейронов можно использовать в обучаемой ИНС при выполнении всего одного условия: класс, описывающий этот слой, должен реализовывать интерфейс *ILearnableLayer.* В логике этого класса выполняется делегирование задачи вычисления величины корректировки весов и сдвигов объекту, реализующему *ILossFunction*.

```
public class FeedForwardNeuralNetwork : INeuralNet
{
    protected List<ILayer> layers = new
List<ILayer>();
    public FeedForwardNeuralNetwork(params
ILayer[] layers)
    {
      //имплементация конструктора  не приводится
для краткости
    }
    public IEnumerable<ILayer> Layers
    {
      get
      {
        return layers;
      }
    }
    public double[] ProcessInput(double[] input)
    {
      for (int i = 0; i < layers.Count; i++)
        input = layers[i].FeedForward(input);
      return input;
    }
}
```

Рисунок 9 - Класс *FeedForwardNeuralNetwork* как одна из реализаций интерфейса *INeuralNet*

Интерфейс *ILearnableNeuralNet* расширяет интерфейс *INeuralNet* перечнем членов, необходимых для выполнения обучения ИНС.

Таким образом, для получения функциональности обучаемой ИНС достаточно выполнить реализацию интерфейсов *ILossFunction*, *ILearnableLayer* и *ILearnableNeuralNet.* В качестве примера на рисунке 10 показана одна из реализаций обучаемой ИНС в виде класса *LearnableFeedForwardNeuralNetwork*.

Важно отметить, что такая реализация является наследником класса *FeedForwardNeuralNetwork* для предотвращения дублирования кода, описывающего логику прямого прохождения сигнала. Как было подробно описано выше, *FeedForwardNeuralNetwork* инкапсулирует логику и все необходимые ссылки на объекты для обеспечения работы логики прямого прохождения сигнала.

3 Модульное тестирование

Предлагаемая методика реализации ИНС обеспечивает приемлемую производительность и точность нейронных сетей, обученных на массивах данных [21], а также даёт возможность выполнять эксперименты, связанные с апробацией различных конфигураций ИНС [22]. Повышение качества существующего кода достигается посредством рефакторинга [5, 23], технология которого предполагает применение модульного тестирования с использованием юнит-тестов (*unittesting*). Рассмотрим предлагаемую декомпозицию логики ИНС с точки зрения удобства реализации юнит-тестов.

Для примера проведём тестирование логики класса *OneFoldActivationLayer.* Один из юнит-тестов, выполняющий проверку корректности метода *FeedForward* данного класса, приведён на рисунке 11.

```
   public class LearnableFeedForwardNeuralNetwork :
FeedForwardNeuralNetwork, ILearnableNeuralNet
   {
      private List<ILearnableLayer> learnableLayers = new
List<ILearnableLayer>();
      public LearnableFeedForwardNeuralNetwork(params
ILearnableLayer[] layers) : base(layers)
      {
         learnableLayers.AddRange(layers);
      }
      public IEnumerable<ILearnableLayer> LearnableLayers
      {
         get
         {
            return learnableLayers;
         }
      }
      public void ProcessError(double[] target, double alpha,
double momentum)
      {
         double[] error =
learnableLayers.Last().ErrorVectorFrom(target);
         for (int i = learnableLayers.Count - 1; i >= 0; i--)
            error = learnableLayers[i].ProcessError(target,
error, alpha, momentum);
         foreach (var layer in learnableLayers)
         {
            layer.UpdateWeights();
            layer.UpdateBiases();
         }
      }
      //некоторые другие члены не приведены для
краткости
   }
```

Рисунок 10 - Класс *LearnableFeedForwardNeuralNetwork*

Легко видеть, что подменой логики вычисления функции активации (имплементация интерфейса *IOneFoldActivationFunction*) в ходе тестирования можно проверить логику работы метода *FeedForward* класса *OneFoldActivationLayer* независимо от других классов. Это очень удобно, т.к. позволяет легко локализовать ошибки в коде и вносить корректировки.

Таким же образом можно проверить корректность использования хранимых данным классом весов, логику вычисления поля индуцирования входных значений для функции активации. При этом достаточно использовать в качестве, как их часто называют на практике, тестового дублёра достаточно простую условную реализацию интерфейса *IOneFoldActivationFunction.*

Аналогичным образом упрощается проверка реализации интерфейса *INeuralNet* класса *FeedForwardNeuralNetwork.* Путём внедрения зависимости в виде тестовых дублёров, представленных в виде значительно упрощённых имплементаций интерфейса *ILayer*, легко проверяется логика работы класса *FeedForwardNeuralNetwork.* Подобное разделение позволило декомпозировать достаточно сложную логику работы ИНС в несколько относительно независимых блоков. При этом каждый из них имеет свою семантическую целостность в сочетании с умеренной сложностью.

```
[TestMethod]
public void OneFoldActivationLayerFeedForward()
{
    //Arrange
    double[,] weightMatrix = new double[,] { { 1, 1, 1, 1
}, { 1, 1, 1, 1 } };
    double[] biasVector = new double[] { 1, 1 };
    OneFoldActivationLayer layer = new
OneFoldActivationLayer(weightMatrix, biasVector, new
DummyOneFoldActivationFunction());
    //Act
    double[] output = layer.FeedForward(new double[] {
1, 2, 3, 4 });
    //Assert
    Assert.AreEqual(11, output[0]);
    Assert.AreEqual(11, output[1]);
}
```

Рисунок 11 - Юнит-тест, проверяющий корректность метода *FeedForward* класса *OneFoldActivationLayer*

Заключение

Выполненная реализация декомпозиции программной логики ИНС опирается на использование объектно-ориентированной парадигмы. Предложенное разделение программной логики между классами и обеспечение слабой связанности между ними упрощает процесс поиска ошибок в коде, делает код более управляемым, повышает продуктивность разработчика при проведении рефакторинга. Приведённые примеры демонстрируют, как рациональная декомпозиция логики на семантически независимые между собой блоки в сочетании с применением концепции «Внедрение зависимостей» способствует большей структурированности кода.

Предлагаемый подход к декомпозиции программной логики ИНС можно применить к широкому спектру различных нейронных сетей: конволюционные и деконволюционные сети; сети, содержащие одновременно конволюционные и полносвязанные слои нейронов; и т.п. При этом единообразие реализации различного рода сетей обуславливает упрощение их понимания, снижает трудозатраты на сопровождение и развитие реализующего их кода.

Список источников

[1] ***Хайкин, С.*** Нейронные сети: полный курс, 2-е изд. – М.: Издательский дом «Вильямс», 2006. – 1104 с.

[2] ***Rassell, S.*** Artificial Intelligence: A Modern Approach, 3rd Edition / S. Rassell, P. Norvig. - Pearson Education, 2010. - 1095 p.

[3] ***Мак-Каллок, У.С***. Логическое исчисление идей, относящихся к нервной активности / У.С. Мак-Каллок, У. Питтс // В сб. «Автоматы» подред. К.Э. Шеннона и Дж. Маккарти. — М.: Изд-во иностр. лит., 1956. — С. 363—384.

[4] ***Fowler, M.*** Patterns of Enterprise Application Architecture / M. Fowler. - Addison-Wesley, 2002. – 560 p.

[5] ***Jacobsen, I.*** Object Oriented Software Engineering: A Use Case Driven Approach / I. Jacobsen, M. Christerson, P. Jonsson, G. Overgaard. - Addison-Wesley / ACM Press, 1992. – 56 p.

[6] ***Gamma, E.*** Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johanson, J. Vlissides. – Addison-Wesley Professional, 1994. - 395 p.

[7] ***Буч, Г.*** Объектно-ориентированный анализ и проектирование с примерами приложений. 3-е изд. / Г. Буч. - М.: ООО «И.Д. Вильямс», 2008. - 720 с.

[8] ***Richter, J.*** CLR via C#, 4th Edition / J. Richter. - Microsoft Press, 2012. – 896 p.

[9] OpenNN, Сайт проекта. – http://www.opennn.net/

[10] ***Yangqing, J.*** Caffe, Сайт проекта. – http://caffe.berkeleyvision.org/

[11] ***Nielsen, M.*** Neural Networks and Deep Learning, free online book / M. Nielsen. – http://neuralnetworksanddeeplearning.com/.
[12] ***Vasilev, I.*** Java deep learning library, Репозиторий проекта. – https://github.com/ivan-vasilev/neuralnetworks/
[13] ***Rumelhart, D.E.*** Learning Internal Representations by Error Propagation / D.E. Rumelhart, G.E. Hinton, R.J. Williams // Parallel Distributed Processing: explorations in the microstructure of cognition. Cambridge, MA: MIT Press.– 1986. - Vol. 1. – P. 318-362.
[14] ***Goodfellow, I.*** Deep learning / I. Goodfellow, Y. Bengio, Y. Courville. – http://www.deeplearningbook.org/
[15] ***Sedgewick, R.*** Algorithms. 4th Edition / R. Sedgewick, K. Wayne. - Addison-Wesley Professional, 2011. – 992 p.
[16] ***Mittal, S***. A Survey of CPU-GPU Heterogeneous Computing Techniques / S. Mittal, J. Vetter // ACM Computing Surveys (CSUR). - 2015. – Vol.47(4). - P. 1–35.
[17] ***Буч, Г.*** Язык UML. Руководство пользователя / Г. Буч, Дж. Рамбо, И. Якобсон. – М.: ДМК Пресс, 2006. – 496 с.
[18] ***Troelsen, A.*** Pro C# 7: With .NET and .NET Core. 8th Edition / A. Troelsen, P. Japikse. - Apress, 2017. – 1410 p.
[19] ***Seemann, M.*** Dependence Injection in .Net. - Manning, 2011. - 584 p.
[20] ***Fowler, M.*** Inversion of Control Containers and the Dependency Injection pattern / M. Fowler - https://www.martinfowler.com/articles/injection.html/.
[21] ***Grother, P.*** NIST Special Database 19 Handprinted Forms and Characters, 2nd Edition / P. Grother, K. Hanaoka. - National Institute of Standards and Technology, 2016. – 30 p.
[22] ***Гирин, Р.В.*** Двухстадийная нормализация выходных сигналов искусственных нейронных сетей / Р.В. Гирин, С.П. Орлов // Вестник СамГТУ. Серия «Технические науки». – 2017. - №4(56). - С. 7-16.
[23] ***Fowler, M.*** Refactoring: Improving the Design of Existing Code, 1st Edition / M. Fowler. - Addison-Wesley Professional, 1999. – 431 p.

OBJECT-ORIENTED DECOMPOSITION OF ARTIFICIAL NEURAL NETWORK'S PROGRAM LOGIC

R.V.Girin[1], S.P. Orlov[2]

[1]*OOO "Intellect Soft", Samara, Russia*
romangirin@gmail.com
[2]*Samara State Technical University, Samara, Russia*
orlovsp1946@gmail.com

Abstract

The article presents a decomposition of neural networks logic for its implementation as a set of loosely coupled domain classes. The structural and functional decomposition of logic are considered. The technique of program logic decomposition is illustrated by UML diagrams. The implementation an artificial neural network's logic in the C # language based on object-oriented approach is briefly described. A similar implementation was used in the design of neural networks configurations for subsequent training, operation, and experimentation. The proposed separation of program logic between classes, as well as providing weak connectivity between them simplifies the process of errors localization in the program code, makes the code more controllable, besides increased developer productivity. The examples demonstrated in the article show how the rational decomposition for neural network program logic with semantically mutually independent units in combination with "Dependency injection" concept, contributes to a more structured code. The new result is the application of object-oriented decomposition for neural network logic, which allows to significantly simplify the process of designing code. An example of testing the neural network decomposed logic is presented. The proposed approach to decomposition of artificial neural networks program logic can be applied to a wide range of different neural networks, such as convolutional, deconvolutional networks, the networks containing fully connective layers of neurons. In this case, the uniformity of implementation of neural networks to simplify their understanding of the causes reduces labor costs for maintenance and development of implementing their code.

Key words: *artificial neural network, system analysis, decomposition, object-oriented analysis.*

Citation: *Girin RV, Orlov SP.* Object-oriented decomposition of artificial neural network's program logic [In Russian]. Ontology of designing. 2018; 8(1): 110-123. DOI: 10.18287/2223-9537-2018-8-1-110-123.

References

[1] ***Hykin S.*** Neural Networks. A Comprehensive Foundation. 2nd Edition. Prentice Hall; 1999.
[2] ***Rassell S, Norvig P.*** Artificial Intelligence: A Modern Approach, 3rd Edition. Pearson Education; 2010.
[3] ***McCulloch WS, Pitts W.*** A logical calculus of ideas immanent in nervous activity // Bull. Math. Biophis., - 1943. – v.5. – p.115–133.
[4] ***Fowler M.*** Patterns of Enterprise Application Architecture. Addison-Wesley; 2002.
[5] ***Jacobsen I, Christerson M, Jonsson P, Overgaard G.*** Object Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley/ACM Press; 1992.
[6] ***Gamma E, Helm R, Johanson R, Vlissides J.*** Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley Professional; 1994.
[7] ***Booch G.*** Object-oriented analysis and design with applications. 3rd Edition [InRussian]. - Moscow: I.D. Williams; 2008.
[8] ***Richter J.*** CLR via C#, 4th edition. Microsoft Press; 2012.
[9] OpenNN, Project Site. - http://www.opennn.net
[10] ***Yangqing J.*** Caffe, Project Site. -http://caffe.berkeleyvision.org
[11] ***Nielsen M.*** Neural Networks and Deep Learning, free online book. - http://neuralnetworksanddeeplearning.com
[12] ***Vasilev I.*** Java deep learning library, project repository. - https://github.com/ivan-vasilev/neuralnetworks
[13] ***Rumelhart DE, Hinton GE, Williams RJ.*** Learning Internal Representations by Error Propagation. Parallel Distributed Processing:explorations in the microstructure of cognition.Cambridge, MA: MIT Press 1986; 1: 318-362.
[14] ***Goodfellow I, Bengio Y, Courville Y.*** Deep learning. - http://www.deeplearningbook.org
[15] ***Sedgewick R, Wayne K.*** Algorithms. 4th Edition. Addison-Wesley Professional; 2011.
[16] ***Mittal S, Vetter J.*** A Survey of CPU-GPU Heterogeneous Computing Techniques. ACM Computing Surveys (CSUR) 2015; 47 (4): 1–35.
[17] ***Booch G, Rambo J, Jacobson I.*** The Unified Modeling Language. Users Guide. Addison-Wesley; 2005.
[18] ***Troelsen A, Japikse P.*** Pro C# 7: With .NET and .NET Core. 8th Edition. Apress; 2017.
[19] ***Seemann M.*** Dependence Injection in .Net. Manning; 2011.
[20] ***Fowler M.*** Inversion of Control Containers and the Dependency Injection pattern. - https://www.martinfowler.com/articles/injection.html
[21] ***Grother P, Hanaoka K.*** NIST Special Database 19 Handprinted Forms and Characters, 2nd Edition. National Institute of Standards and Technology; 2016.
[22] ***Girin RV, Orlov SP.*** Two-stage normalization of output signals of artificial neural networks [In Russian]. Bulletin of Samara State Technical University. “Technical Sciences” 2017; 4(56): 7-16.
[23] ***Fowler M.*** Refactoring: Improving the Design of Existing Code, 1st Edition. Addison-Wesley Professional; 1999.

Сведения об авторах

Гирин Роман Викторович,1979 г. рождения. Окончил Самарский государственный технический университет в 2003 г. Разработчик проектов в компании ООО «Интеллект Софт», Самара. Соискатель ученой степени кандидата технических наук при Самарском государственном техническом университете. Область научных интересов – нейронные сети и их приложения.

Roman Victorovich Girin (b. 1979) graduated Samara Technical State University in 2003.He is project developerof OOO “Intellect Soft”, Samara. He is grandaunt to the degree of Cand.Tech.Sci. at the Samara State Technical University. The field of scientific interests is neural networks and their applications.

Орлов Сергей Павлович, 1946 г. рождения. Окончил Куйбышевский политехнический институт в 1969 г., д.т.н.(1991), профессор. Заведующий кафедрой «Вычислительная техника» Самарского государственного технического университета. Член Международной академии информатизации, член международного общества IEEE. В списке научных трудов более 180 работ в области компьютерных технологий, моделирования сложных систем, создания интеллектуальных систем поддержки принятия решений в технических и социальных системах.

Sergey Pavlovich Orlov (b. 1946) graduated from the Kuibyshev Polytechnic Institute in 1969, D. Sc. Eng. (1991), professor. He is Head of Computer Technology Department of Samara State Technical University. He is member of International Informatization Academy and IEEE member. He is co-author of more than 180 publications in the field of computer technology, complex systems simulation, knowledge based decision support systems in technical and social systems.