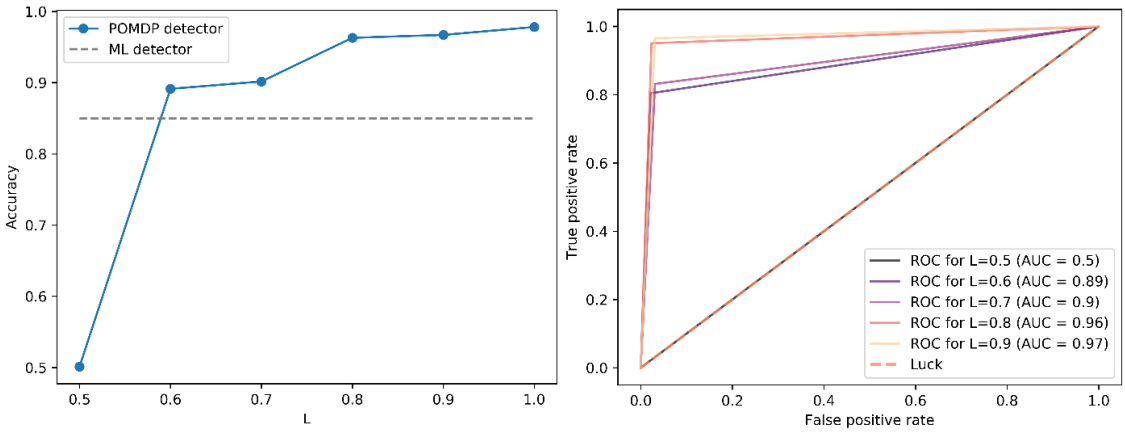

а б

Рисунок 1а – зависимость точности модели от параметра *L*,
1б – ROC-кривые для различных значений параметра *L*

Из полученных результатов видно, что рост параметра L положительно влияет на точность распознавания DNS туннелирования. В то же время, при L < 0,6 качество распознавания резко снижается, и при L = 0,5 носит характер случайного гадания. Стоит отметь, что в этом случае, система производит меньшее количество ложно положительных ошибок, но большее количество ложно отрицательных ошибок. Это объясняется тем, что система пропускает больше подозрительного трафика с целью обеспечения доступности сети в целом.

Таким образом, в поставленном эксперименте, с помощью L-параметризированного POMDP удалось добиться увеличения точности распознавания DNS туннелирования на 10% в сравнении с детектором, представленным в статье [3].

Список использованных источников:

1. Valenzuela, I. Game Changer: Identifying and Defending Against Data Exfiltration Attempts. – Boston : Sans Institute, 2015.
2. Nadler, A. Detection of Malicious and Low Throughput Data Exfiltration Over the DNS protocol / A. Nadler, A. Aminov, A. Shabtai – Negev : Ben Gurion University of the Negev, 2017.
3. Bubnov, Y. DNS Tunneling Detection Using Feedforward Neural Network / Y. Bubnov // European Journal of Engineering Research & Science. – 2018. – Vol. 3, № 11. – P. 16-19.
4. McCarthy, S.Data Exfiltration Detection and Prevention: Virtually Distributed POMDPs for Practically Safer Networks / S. McCarthy, A. Sinha, M. Tambe, P. Manadhata – Los Angeles : University of Southern California, 2016.
5. Iodine [Электронный ресурс]. – Режим доступа: http://code.kryo.se/iodine. – Дата доступа: 03.22.2019.
6. Cassandra, A. Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Process / A. Cassandra, M. Littman, N. Zhang // Proceedings of the Thirteen Conference on Uncertainty in Artificial Intelligence. – 1997. – P. 54-61.

СВЕРТОЧНАЯ НЕЙРОННАЯ СЕТЬ ДЛЯ РАСПОЗНАВАНИЯ РУКОПИСНЫХ СИМВОЛОВ

Гаджиев С. С.

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

Воронов А. А. – к.т.н.,доцент

Рассмотрены вопросы построения сверточной нейронной сети для распознавания рукописных символов. Определена основная модель, приведены примеры слоев нейронной сети.

В этой статье опишем структуру сверточной нейронной сети для распознавания рукописных символов. Классическая архитектура CHC представлена на рисунке 1:

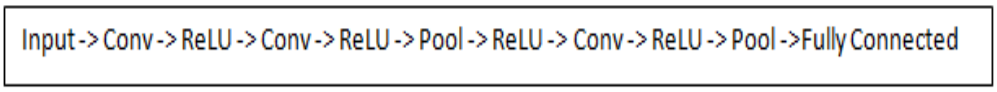

Рис. 1- Архитектура CHC

Рассмотрим, какие действия сеть будет выполнять на высоком уровне. Все фильтры можем рассмотреть, как идентификаторы свойства. Свойством могут быть кривые, прямые границы,

значение цвета. Например, первый фильтр будет иметь размер 9 X 9, и он будет являться детектором прямых границ. Фильтр имеет пиксельную структуру, численные значения которой выше вдоль области, идентифицирующей форму прямой линии. Когда в левом верхнем углу начального изображения начинает проходить фильтр, он умножает значение фильтра на значения пикселей выделенной зоны.

Выше было упомянуто, что умеют делать фильтры первого свёрточного слоя. Они идентифицируют такие свойства базового уровня, как границы и кривые. Как можно себе предположить, чтобы представить какой тип объекта присутствует на картинке, нам нужна нейронная сеть, которая может распознать свойства более высокого уровня, как к примеру глаза, лицо или уши. Нужно подумать, какая картинка будет приходить на первый слой. К примеру его размер будет 32 X 32. Когда рисунок пройдет через первый свёрточный слой, выход этого слоя станет входным параметром второго слоя. Сейчас представить это немного сложнее в визуальном виде. Когда описывали первый слой, входом были только данные исходного изображения. Но когда мы перешли ко второму слою, входным параметром для него будет одна или же несколько карт свойств — исход работы первого слоя. Все наборы входных данных определяют места, где на исходном образе находятся требуемые базовые признаки.

Сейчас, когда вы используете набор фильтров поверх этого на выходе активизируются фильтры, которые имеют более высокий уровень. Чем больше количество свёрточных слоёв обрабатывает картинку и чем глубже оно движется по сети, тем более сложные свойства будут входить в карты активации. В конце сети могут быть фильтры, которые активизируются, когда на рисунке есть рукописный текст, есть розовые объекты и т.д.

Еще один увлекательный момент. Когда вы передвигаетесь дальше внутрь сети, фильтры отрабатывают со все большим полем восприятия, и значит, они могут обрабатывать данные с большей площадью начального рисунка.

Теперь, когда мы имеем возможность обнаружить высокоуровневые свойства, мы можем начинать привязывать полносвязный слой в конец сети. Полносвязный слой берёт входную информацию и выводит N-пространственный вектор, где N — количество классов, из которых будет выбран нужный. К примеру, если требуется нейронная сеть по распознаванию символов латинского алфавита, и N будет иметь значение 12, то есть возьмем для примера первые 12 символов из алфавита. Все символы в этом N-пространственном векторе будут представлять собой вероятность определенного класса. К примеру, если результирующим вектором для сети, которая распознает символы является вектор (0 0,2 0,15 0,8 0 0 0,02, 0,01 0 0 0,02), это означает что есть 20% вероятность, что на рисунке "B", 15% вероятность, что на рисунке "C", 80% вероятность — "D", и 2% вероятность — "I".

Метод, благодаря которому работает полносвязный слой — это обращение к выходу предыдущего слоя и нахождение параметров, которые более характерезуют определенный класс. К примеру, если сеть идентифицирует на каком-либо образе кота, у набора свойств, которые отражают высокоуровневые характеристики, такие как лапы или морду кота, должны иметь высокие параметры. Точно так же, если сеть идентифицирует, что на изображении черепаха — у неё будут высокие значения в наборах свойств, представленных высокоуровневыми характеристиками такими как размер и форма панциря или лапы.

Полносвязный слой ссылается на то, что функции более высокого уровня сильно привязаны к определенному классу и имеют некоторые веса, в таком случае, когда высчитываются произведения весов с предыдущим слоем, то получается верные вероятности для различных классов.

Обучение – это один из аспектов нейронных сетей, о котором до сих пор не упоминалось. Вероятно, это самая важная часть. Сейчас мы задаемся вопросами. Как фильтры первого свёрточного слоя узнают, что нужно найти границы и кривые? Как полносвязный слой узнает, что должна найти набор свойств? Как фильтры каждого слоя узнают, что именно хранить? Метод, с которым машина способна подбирать значения фильтра — это обучающий процесс, который известен как метод обратного распространения ошибки.

Метод обратного распространения ошибки можно поделить на 4 различных блока:

прямое распространение;

функцию потери;

обновление веса;

обратное распространение.

Когда происходит прямое распространение, берётся тренировочный образ — к примеру, это матрица 28 X 28 — и образ пропускают через всю сеть. В первом обучающем примере, так как все веса или значения фильтра были инициализированы случайным образом, выходным значением будет что-то вроде [.11 .15 .01 .1 .07 .09 .13 .11 .11 .07], то есть такое значение, которое не даст предпочтения какому-то определённому числу. Нейронная сеть с такими весами не сможет найти параметры базового уровня и не может обоснованно идентифицировать класс образа. Это приводит к функции потери. Для обучения требуются обучающие данные. У таких данных есть и изображение, и

ярлык. Допустим, первое обучающее изображение — это цифра 2. Ярлыком изображения будет [0 0 1 0 0 0 0 0 0 0]. Функция потери может быть выражена по-разному. В нашем случае применяется среднеквадратическая ошибка, это 0.5 умножить на (реальность — предсказание) в квадрате.

Примем этот параметр за переменную *L*. Как можно догадаться, потеря будет иметь очень высокое значение для первых двух обучающих образов. Теперь давайте подумаем об этом интуитивно. Мы хотим добиться того, чтобы предсказанный класс был таким же, как класс обучающего изображения. Чтобы добиться этого, нужно привести к минимуму количество потерь, которое имеется.

Скорость обучения — это параметр, который выбирается при разработке сети. Высокая скорость обучения означает, что в обновлениях весов выполнялись большие шаги, поэтому образу может понадобиться меньше времени, для того что бы набрался верный набор весов. Но слишком высокая скорость обучения может привести к очень крупным и неточным шагам, которые могут помешают набрать оптимальные показатели. Процесс прямого распространения, функцию потери, обратное распространение и обновление весов, обычно называют одной эпохой. Программа будет повторять эпоху определенное количество раз для каждого обучающего образа. После того, как завершится обновление свойств на последнем обучающем образце, сеть предположительно должна быть достаточно хорошо обучена и веса слоёв настроены верно.

Для построения модели нейронной сети было использована библиотека Tensor Flow. Было протестировано на базе данных MNIST. Результат зависел от количества эпох для обучения нейронной сети. К примеру, если использовать 10 эпох, то процент распознавания будет 60%. С увеличением количества эпох до 100 к примеру процент распознавания поднялось до 85%. Так же стоит учитывать количество классов для обучения. Чем больше классов, тем меньше процент распознавания.

Список использованных источников

1. Шапиро, Л. Компьютерное зрение / Л. Шапиро, Дж. Стокман. – М. // БИНОМ. Лаборатория знаний, 2006. – 762 с.
2. Гонсалес, Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. // М. : Техносфера, 2005. – 1073 с.

ПРИНЦИПЫ РАБОТЫ ДИНАМИЧЕСКОЙ ДИСПЕТЧЕРИЗАЦИИ МЕТОДОВ В РАЗЛИЧНЫХ ЯЗЫКАХ ПРОГРАММИРОВАНИЯ

Ганнусенко Н.Н.

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

Иванов Н.Н. – к.ф.-м.н., доцент

При разработке программного обеспечения можно использовать различные парадигмы программирования. И часто для их применения используются полиморфные объекты – это объекты, которые меняют своё поведение в зависимости от обстоятельств. И динамическая диспетчеризация методов – один из способов употребления таких объектов. Динамическая диспетчеризация методов – это механизм выбора реализации полиморфной операции на этапе времени выполнения программы. А механизмы их реализации уже представлены по-разному в каждом из языков программирования.

В ходе изучения способов реализации динамической диспетчеризации методов в разных языках программирования эти механизмы можно разделить на подходы:

- применение виртуальных таблиц методов;
- применение толстых указателей;
- обращение через таблицы поиска;
- поиск посредством пересылки сообщений;
- переходы через указатели на функции.

В таких языках программирования, как C++, Java, C#, Swift, используются виртуальные таблицы методов. Хоть их реализации и различаются в зависимости от наличия множественного наследования [1], от наличия интерфейсов [2] или протоколов [3, 4], но у реализаций есть общее – выделение вызовов динамического поведения в отдельные виртуальные таблицы методов [5], а доступ к ним осуществляется посредством обращения к указателям на связанные таблицы, ссылки которых хранятся у каждого объекта класса. Пример виртуальной таблицы методов представлен на рисунке 1.

В таких языках программирования, как Rust или Go, используется концепция толстых указателей. Принцип её работы схож с подходом использования виртуальных таблиц методов, но имеется разница в том, где конкретно хранится указатель на виртуальную таблицу методов. В толстых указателях ссылки на соответствующие виртуальные таблицы методов хранятся вместе