Санкт-Петербургский государственный университет
Кафедра теории систем управления электрофизической аппаратурой

Московская Маргарита Дмитриева

Выпускная квалификационная работа бакалавра

Разработка самообучающегося робота для игровых мобильных приложений

Направление 02.03.02

«Фундаментальная информатика и информационные технологии»

ООП «Программирование и информационные технологии»

Научный руководитель:
кандидат физ.-мат. наук,
доцент Козынченко
Владимир Александрович

Санкт-Петербург
2019

Оглавление

Введение

Создание бота [1], то есть программы, автоматически выполняющей определенные действия (самостоятельно играть) через тот же интерфейс, что и люди, для мобильной игры довольно интересная, полезная и актуальная задача. Существует очень много видов игр и их реализаций. Для их тестирования, предоставления возможности играть в многопользовательские игры одному игроку, а также для других целей, разрабатываются боты. Научив бота успешно играть в игру, можно автоматизировать ее тестирование, определение сложности уровней и прочее.

Главной целью выпускной квалификационной работы является разработка самообучающегося бота для мобильной игры. Его эффективность оценивается по количеству очков, зарабатываемых ботом в игре. Для реализации программы бота необходимы данные о происходящем в игре: набранные очки, положение игрока, и информация о различных важных объектах на игровом поле (вектор состояния). Вектор состояния используется для определения вознаграждения за выполняемые ботом действия при обучении с подкреплением.

Особенностью задачи является невозможность использовать и модернизировать код игры для получения данных, необходимых при обучении бота. Эта особенность обусловлена тем обстоятельством, что игра, для которой разрабатывается бот, не всегда имеет открытый доступный код. Поэтому все нужные данные следовало получать сторонними средствами с помощью обработки кадра игры.

При решении задачи распознавания на кадре игры распознаются цифры текущего счета, препятствия и графические элементы, обозначающие проигрыш игры. Каждый кадр игры - конкретное состояние бота и описание

его окружения. Изображение кадра необходимо получить. Далее на основе полученных данных необходимо обучить бота, используя выбранные архитектуру нейронной сети и метод ее обучения.

Проблемы, которые решаются при разработке бота, то есть ситуации, с которыми он сталкивается в играх и с которыми необходимо справиться, довольно сильно связаны с реальной жизнью и можно найти в ней их отражения. Обход препятствий, взаимодействие с окружающей средой и враждебными факторами - всё это встречается как и в играх, так и в реальности. Можно сказать, что боты функционируют в игровом мире так же, как и механические роботы в нашем, и имеют схожие принципы обучения.

Самообучающиеся боты используются для достижения более человекоподобного поведения. Можно было бы жестко и явно прописать боту модель поведения, определенные действия в определенных ситуациях. Однако это слишком трудоемко, ведь предугадать все возможные игровые ситуации практически нереально. Одним из способов достижения самообучаемости бота является использование искусственной нейронной сети как основы для принятий решений о действиях, которые будет совершать бот для успешного продолжения игры. Существует несколько типов обучения нейронной сети: обучение с учителем, обучение без учителя и обучение с подкреплением. Для первого типа обучения необходим большой объем специально обработанных обучающих данных, которые не представляется возможным получить в рамках рассматриваемой задачи. Второй вариант обучения используется в задачах другого типа, в частности в задачах классификации. Третий вариант, обучение с подкреплением - наиболее подходящий и часто используемый вариант для тренировки нейросети, на основе которой функционируют боты.

Получение информации для вектора состояния было проведено несколькими способами, из которых были выбраны наиболее эффективные. Среди испытанных вариантов были поиск на кадре игры по шаблону, обучение с учителем и классификаторы на основе метода опорных векторов. Подробнее о использованных в работе методах можно прочитать в главе 1. Для получения различных данных, входящих в вектор состояния, были исследованы и использовались разные методы, что зависело от их возможностей и пригодности получения конкретных данных. Например, состояние бота, то есть жив он или игра окончена, и счет игры определялись отличными друг от друга способами. Подробнее об этом написано в главе 2.

В главе 3 описаны проведенные эксперименты, в частности по обучению бота. Также в ней указаны используемые параметры обучения и достигнутые результаты. Разработка велась на языке Python 3.6.

Обзор литературы

Существует достаточно много статей про создания ботов на основе обучения с подкреплением для различных игр с использованием уже созданных клонов игр для библиотеки PyGame языка python, например, в статье Using Reinforcement Learning techniques to build an AI bot for the game Flappy Bird [2] автор для удобства обучения бота использует разработанную другим человеком копию игры Flappy Bird на том же языке, что и созданный им бот и алгоритм обучения, в дополнение код копии игры изменяется автором статьи. Также можно встретить статьи, где для модификации и обучения используется открытый код игры. Автор статьи How I built an AI to play Dino Run [3] использует открытый код игры Dino Run, появляющейся при отсутствии подключения к интернету в браузере Chrome, модифицирует его и взаимодействует напрямую с игрой для получения данных при обучении и функционировании бота. В модификациях кодов игр в данных статьях прослеживается тенденция убирать задний динамичный фон, что связано с уменьшением поступления и обработки лишней информации об игре для ускорения процесса обучения и уменьшения требований производительности оборудования, на котором запускается игра и осуществляется обучение бота.

В постановке задачи данной работы указано, что получение информации об игре путем захвата и обработки кадра игры. По теме компьютерного зрения, методов преобразования изображений и поисков на них объектов написано много книг. В частности, в книге Learning OpenCV 3 Computer Vision with Python [4] можно познакомиться как с основными аспектами работы с изображениями при использовании библиотеки OpenCV, поддерживающейся многими языками программирования, так и с самой библиотекой.

Перед разработкой бота для мобильной игры полезно изучить имеющиеся материалы не только о самих ботах и положении дел в этой области, но и о играх, их структуре и ключевых моментах игрового процесса и его разработке. Общие концепции устройства игр, их тестирования и взаимодействия с ними, отличного от пользовательско-игрового, можно прочесть в книге Game Development Essentials Game Qa & Testing[5]. Также в ней рассматриваются вопросы автоматического тестирования, в частности роботами. С существующими концепциями ботов и техниками их взаимодействий с играми, а также целями их применений и разрабоки, можно познакомиться в книге Practical Video Game Bots: Automating Game Processes using C++, Python, and AutoIt [6].

Глава 1. Используемые при реализации бота методы

1.1. Нейронные сети

Нейронная сеть - математическая модель, а также её программная или аппаратная реализация, представляющая собой связную систему из элементов, называемых нейронами, которые можно описать нелинейной функцией, именуемой функцией активации, от линейной комбинации входных сигналов. Нейронная сеть состоит из входного слоя, одного или более скрытых (вычислительных) слоев и выходного слоя.

Нейрон является вычислительным элементом сети, который получает на вход информацию, состоящую из одного или больше сигналов, вычисляет значение функции активации на ее основе, которое является выходным сигналом нейрона. На рисунке 1 представлено схематическое изображение нейрона с поясняющими подписями.

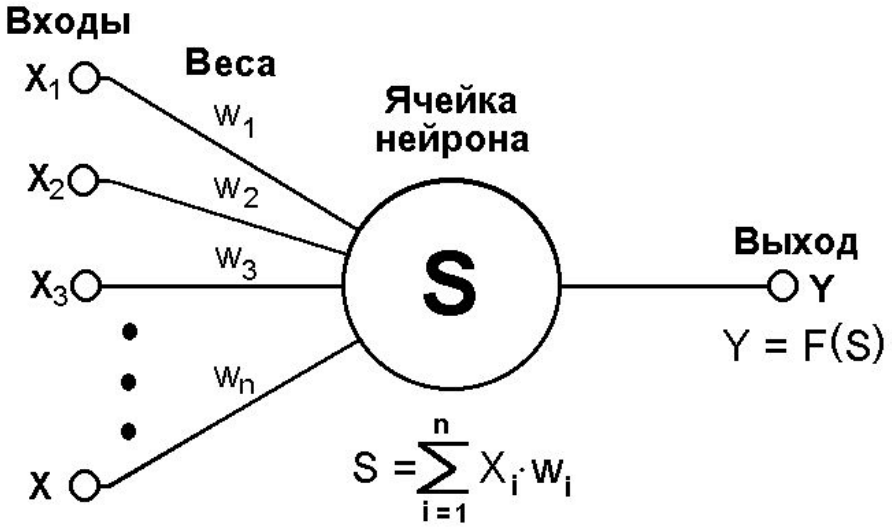


Рисунок 1. Схема нейрона.

Функция активации F рассчитывает выходное значение нейрона в зависимости от взвешенной суммы входных сигналов. В данной работе используется два вида функций активации: сигмоида и ReLU.

Сигмоида - нелинейная функция, хорошо подходящая для задач классификации, имеющая фиксированный диапазон значений, и стремящаяся перевести значения к концам этого диапазона. Формула сигмоидальной функции:

$$F(x) = \frac{1}{1+e^{-x}} \quad .$$

Функция активации ReLu имеет формулу F(x) = max(0, x) и так же является нелинейной. Её главное преимущество в легкости вычисления и разряженной активации нейронов, то есть из-за ее свойства возвращать 0 при отрицательных значениях аргумента большое количество нейронов не будут активированы.

Персептрон - элементарная модель нейронной сети, архитектура которой состоит из последовательного расположения слоев трех типов: первый слой состоит из принимающих входные сигналы элементов, или сенсоров, далее расположен слой нейронов, или несколько слоев, если персептрон многослойный. Последний слой состоит из элементов, считающих линейную сумму полученных входных сигналов для реализации своей пороговой функции.

Сверточная нейронная сеть - нейронная сеть со специальной архитектурой, характеризующейся чередованием сверточных слоев и субдискретизирующих слоев (или англ. pooling layers). Сверточный слой - основной слой данной нейронной сети. В нем осуществляется операция свертки, при которой определенные части входного сигнала умножаются поочередно на матрицу небольшого размера, элементами которой являются

веса, настраиваемые в процессе обучения. Суть этой операции заключается в том, что матрица перемещается по обрабатываемому предыдущему слою, формируя после каждого перемещения часть выходного сигнала в соответствующей позиции. Далее идет слой субдискретизации. В нем осуществляется операция нелинейного уплотнения карты признаков, получившейся как результат работы сверточного слоя, при этом группа пикселей (обычно размера 2×2) заменяются одним пикселя, проходя нелинейное преобразование.

Для оценки качества модели используются метрики. В данной работе были выбраны следующие две метрики: accuracy и precision. Accuracy = $\frac{P}{N}$, где P – количество элементов выборки, по которым классификатор принял правильное решение, а N – размер обучающей выборки. Так как эта метрика может плохо работать при несбалансированном количестве элементов в разных классах, то выбранная вторая метрика, precision, оценивает качество работы алгоритма на каждом из классов по отдельности. Метрика precision рассчитывается по формуле: precision = $\frac{TP}{TP+FP}$, где TP - количество элементов, верно отнесенные к рассматриваемому класса, а FP - количество элементов, которые ошибочное определены как принадлежащие этому классу. Можно сказать, что данная метрика считается делением правильно классифицированных элементов одного класса на все элементы, определенные как элементы этого класса.

1.2. Обучение с учителем

Обучение с учителем - один из методов обучения нейронной сети, при котором сеть обучается на специальном наборе данных, называемых обучающими, состоящим из непосредственно входных данных и требуемых результатов работы сети, коротко ярлык, или метка, для этих данных. На

вход нейросети подаются данные с их выходными метками и веса этой сети пересчитываются, чтобы выдавать желаемый результат. Так происходит до тех пока, пока не достигается определенный порог точности ответов сети при проверке на тестовых данных, специальной части обучающих данных, которые не участвуют в процессе обучения, а используются для расчета метрик. Один период обучения нейросети, за который через нее пройдет весь обучающий набор данных, называется эпохой. Метрики считаются по завершению каждой эпохи.

В данной работы задачей обучения будет бинарная классификация изображений на предмет наличия индикаторов смерти игрока, и соответственно бота. Одним из наиболее часто используемых алгоритмов при обучении с учителем является метод обратного распространения ошибки [7]. Его суть состоит в том, что при обновлении весов считается производная функции активации и ошибка сети, на основе чего веса последовательно изменяются, начиная с последнего слоя, двигаясь в сторону входного слоя. При обучении с помощью обратного распространения ошибки можно использовать различные методы градиентного спуска.

1.3. Обучение с подкреплением

Обучение с подкреплением - один из методов обучения нейронной сети, при котором агент взаимодействует с окружением и с помощью функции вознаграждения, являющейся реакцией на результат его действий, вырабатывает модель поведения, соответствующую ожиданием разработчика. Агент - система взаимодействия с внешней средой, способная выполнять действий. Окружение - мир, в котором существует и двигается агент.

Важную роль в обучении с подкреплением играет функция вознаграждения, на основе которой нейросеть получает реакцию на свой текущий результат работы и обучается выдавать тот результат, который наиболее близок к желаемому. Отрицательно вознаграждение назначается за нежелательные или фатальные действия, ведущие непосредственно к проигрышу, а также за сам проигрыш. Положительные действия. ведущие к продвижению по игре и увеличению заработанных в ней очков, ранжируются по степени влияния на конечный результат с помощью разных значений функций, которые тем больше, чем больше влияние.

Основной задачей при обучении с подкреплением является расчет функции полезности Q, которая оценивает ожидаемый выигрыш в зависимости от выбранного действия и состояния системы на данный момент и политики π выбора дальнейших действий [9]. Политика π - стратегия поведения бота, с помощью которой он выбирает следующее действие на основе текущего состояния

Функция полезности рассчитывается по формуле:

$$Q_\pi(s_t,\ a_t) \leftarrow Q\,(s_t,\ a_t) + \alpha \cdot \left(r_t + \gamma \cdot maxQ_\pi(s_{t+1},\ a_t) - Q_\pi(s_t,\ a_t\ \right)$$

При обучении с подкреплением с использованием нейросети, расчет данной функции заменяется нейронной сетью.

1.4. Метод опорных векторов

Суть метода опорных векторов состоит в том, чтобы переместиться из имеющегося пространства в пространство с большим количеством измерений и найти в нем гиперплоскость, разделяющей объекты и находящейся от них максимально далеко. Гиперплоскость - линейное пространство, имеющее

размерность, равную n-1, и являющееся подпространством пространства размерности n.

Пусть имеется обучающая выборка:

$$(x_1, y_1), \ldots, (x_m, y_m), \text{ где } x_i \in R^n, y_i \in \{-1, 1\}.$$

Метод опорных векторов строит классифицирующую функцию F в виде $F(x) = sign(w \bullet x + b)$, где $\bullet$ — скалярное произведение, w — нормальный вектор к разделяющей гиперплоскости, а b — вспомогательный параметр. Функция имеет именно такой вид потому, что любая гиперплоскость может быть задана в виде $w \bullet x + b = 0$ для некоторых w и b. Те объекты, для которых $F(x) = 1$ попадают в один класс, а объекты с $F(x) = -1$ — в другой. Далее, необходимо найти такие w и b, которые максимизируют расстояние до каждого класса. Можно подсчитать, что данное расстояние равно $\frac{1}{||w||}$. Проблема нахождения максимума $\frac{1}{||w||}$ эквивалентна проблеме нахождения минимума $||w||^2$. Можно записать это в виде задачи оптимизации, являющейся стандартной задачей квадратичного программирования и решается с помощью множителей Лагранжа [10].

Глава 2. Программная реализация бота

2.1. Описание игры

Для программной реализации бота была выбрана игра “Line Runner” [11]. Основная задача игрока в ней заключается в том, чтобы как можно дольше бежать и не сталкиваться с препятствиями. В этой игре препятствия выглядят как черные квадраты и могут располагаться как на высоте, так и на линии, по которой бежит игрок. Препятствия можно перепрыгнуть или нагнуться для прохождения под ними. Очки начисляются за длину пройденной дистанции, остановка игрока не допускается, а при столкновении с препятствием игра прекращается (игрок проигрывает).

Игра запускается на компьютере с помощью android-эмулятора.

Вектор состояния в рассматриваемой игре имеет вид x = (x1, x2, x3), где x1 - это количество препятствий, x2 - счет, x3 - состояние игрока (0 - мерт, 1 - жив).

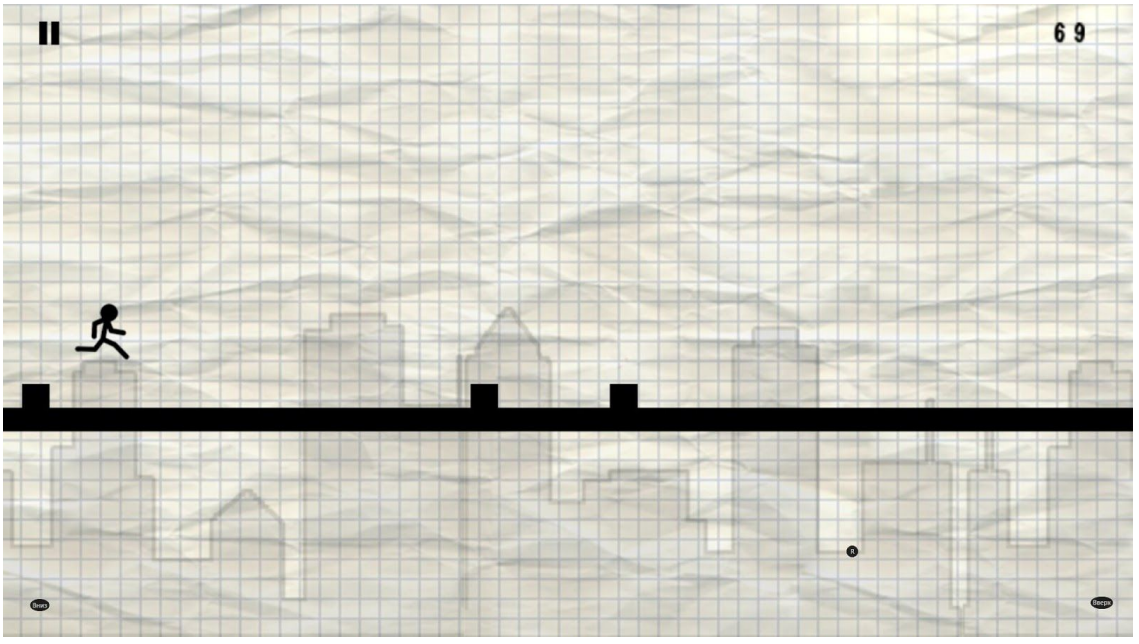


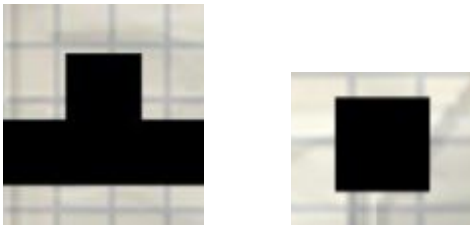
Рисунок 2. Кадр игры

2.2. Поиск по шаблону

В качестве первого способа распознавания и нахождения необходимых элементов на кадре игры был выбран и протестирован поиск по шаблону. Смысл данного способа заключается в следующем: для поиска необходимо иметь изображение с необходимым для распознавания графическим элементом, называемое шаблоном. Этот шаблон сравнивается с другим изображением, для которого требуется определить наличия на нем идентичного или почти идентичного шаблону фрагмента. Сравнение может происходить любым выбранным методом по обработки и сравнению изображений.

Для создания шаблонов были взяты пара кадров игрового процесса и кадр с графическими элементами конца игры, или поражения игрока. Из этих кадров вырезаются объекты, наличие или отсутствие которых на изображении текущего игрового процесса необходимо определять для успешного обучения бота. Пример полученных шаблонов можно увидеть на рисунке 3.

Шаблон конца игры



Шаблоны препятствий

Рисунок 3.Шаблоны.

Для реализации поиска по шаблону была использована библиотека OpenCV[12] для языка Python. Далее приведена основная функция, реализующая поиск по шаблону, и ее параметры:

```
cv2.matchTemplate(img,imgtpl,cv2.TM_CCOEFF_NORMED)
```

Параметр `cv2.TM_CCOEFF_NORMED` [13] определяет метод процесса сопоставления шаблона с изображением, `imgtpl` – шаблон, а `img` – кадр игры.

2.3. Применение обучения с учителем

Вектор состояния - вектор, содержащий всю необходимую информацию о состоянии объекта. В качестве второго способа определения одной из составляющих вектора состояния, а именно для нахождения на кадре графических элементов, обозначающих проигрыш бота, или конец игры, было использовано обучение с учителем. Для этого была создана база данных, состоящая из 100 кадров, среди которых на 70 кадров присутствуют необходимые элементы, а на 30 кадрах изображен игровой процесс. В файл формата .csv были внесены данные о обучающих изображениях данной выборки: идентификатор изображение и его метка, где 0 соответствует изображению обычного игрового процесса, а 1- окончанию игры. Метка - при обучении с учителем так называется идентификатор входных данных и желаемый ответ нейросети. Полученная база кадров была разделена на обучающие и тестовые выборки. При обучении была использована библиотека Keras совместно с TensorFlow. На рисунке 2 приведены примеры обучающих кадров совместно с их метками.

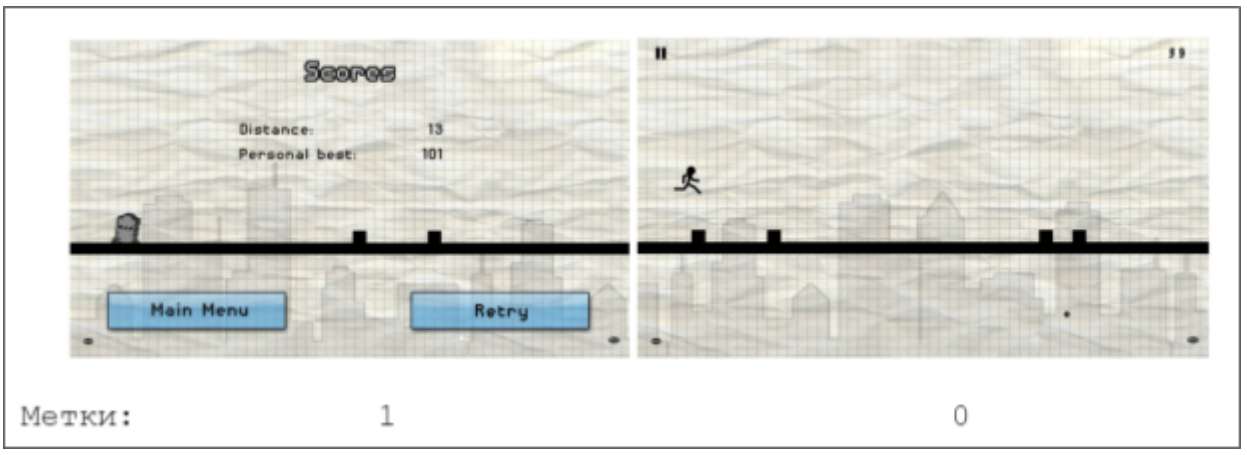


Рисунок 4. Обучающие кадры.

2.4. Определение счета игры

Для определения заработанных очков в текущей игре было решено использовать классификатор на основе метода опорных векторов. Было разработано и протестировано несколько вариантов классификаторов, основанных на разных библиотеках.

В верхнем правом углу кадра игры изображается число, являющееся текущим счетом игры. Для определения этого числа была собрана база кадров со всеми цифрами от 0 до 9. С помощью приложения ImgLab произведена их разметка и получен .xml файл с описывающими эти кадры данными: именем изображения, имеющимися на нем цифр и координат их расположения. Этот файл, необходимый для обучения классификатора на основе метода опорных векторов из библиотеки dlib. Далее приведен соответствующий обучению код:

```
dlib.train_simple_object_detector("training.xml",
                                  "detector.svm")
```

Было проделано два варианта обучения данного классификатора: в первом случае классификатор обучался распознавать все цифры, а во втором - для каждой цифры обучался свой детектор.

Для реализации другого классификатора в качестве основной библиотеки была выбрана scikit-learn [14]. Для его обучения, реализованного аналогично предыдущему варианту на основе метода опорных координат был использован стандартный набор рукописных цифр MNIST, так как шрифт выбранной игры можно описать как достаточно приближенный к человеческому письму.

Далее приведен код тренировки классификатора:

```
features = np.array(dataset.data, 'int16')
labels = np.array(dataset.target, 'int')
for feature in features:
    fd = hog( feature.reshape((28, 28)),
            orientations=9, pixels_per_cell=(14,14),
            cells_per_block=(1,1), visualise=False)
    list_hog_fd.append(fd)
hog_features = np.array(list_hog_fd, 'float64')
pp=preprocessing.StandardScaler().fit(hog_features)
hog_features = pp.transform(hog_features)
clf = SVC()
clf.fit(hog_features, labels)
joblib.dump((clf, pp), "digits_cls.pkl", compress=3)
```

После этого, для использования классификатора обрабатываются кадры игры и передаются как входные данные в функцию `clf.predict()`,

которая возвращает в качестве ответа все цифры, имеющиеся на кадре по ее расчетам.

Так как любой вариант и реализация классификатора в качестве результата своей работы возвращает массив найденных цифр, необходимо объединить их в одно число, равное счету, изображенному на кадре.

Ниже представлен код этого процесса:

```
answer=0
for i in scores:
        tmp = i
        if scores.index(i)>0:
            tmp = i*10*scores.index(i)
        answer += tmp
```

Глава 3. Эксперимент

3.1. Обучение с учителем

Для определения на кадре графических элементов, обозначающих завершение игры и, соответственно, проигрыш бота, было использовано обучение с учителем. Архитектура использованной нейросети представлена в таблице 1. При обучении в качестве оптимизатора использовался RMSprop.

№ уровня	Тип уровня	Параметры	Функция активации
1	Convolution	32	Relu
2	Convolution	32	Relu
3	Pooling	(2, 2)	
4	Convolution	64	Relu
5	Convolution	64	Relu
6	Pooling	(2, 2)	
7	Convolution	128	Relu
8	Convolution	128	Relu
9	Pooling	(2, 2)	
10	Convolution	256	Relu
11	Convolution	256	Relu
12	Pooling	(2, 2)	
13	Fully connected	256	Relu
14	Fully connected	256	Relu
15	Fully connected	1	Sigmoid

Таблица 1. Архитектура нейронной сети при обучении с учителем.

Обучение проходило в течение 7 эпох, в каждой из которых было 5 итераций, при этом обучающий набор (batch size) состоял из 3 элементов. Были испробованы разные варианты, в итоге, как самый оптимальный вариант, были выбраны данные параметры. Метриками для контроля процесса обучения были выбраны accuracy и precision. Далее представлены: на рисунке 5 - график зависимости значений метрик от количества пройденных эпох, на рисунке 6 - график изменения ошибки соответственно. Значение метрик стремиться к 1, а ошибка достигает $10e^{-4}$, что свидетельствует о высоких результатах обучения, хорошем качестве модели и соответствует желаемым значениям.

Для реализации нейронной сети и ее обучения была использована библиотека Keras совместно с TensorFlow.

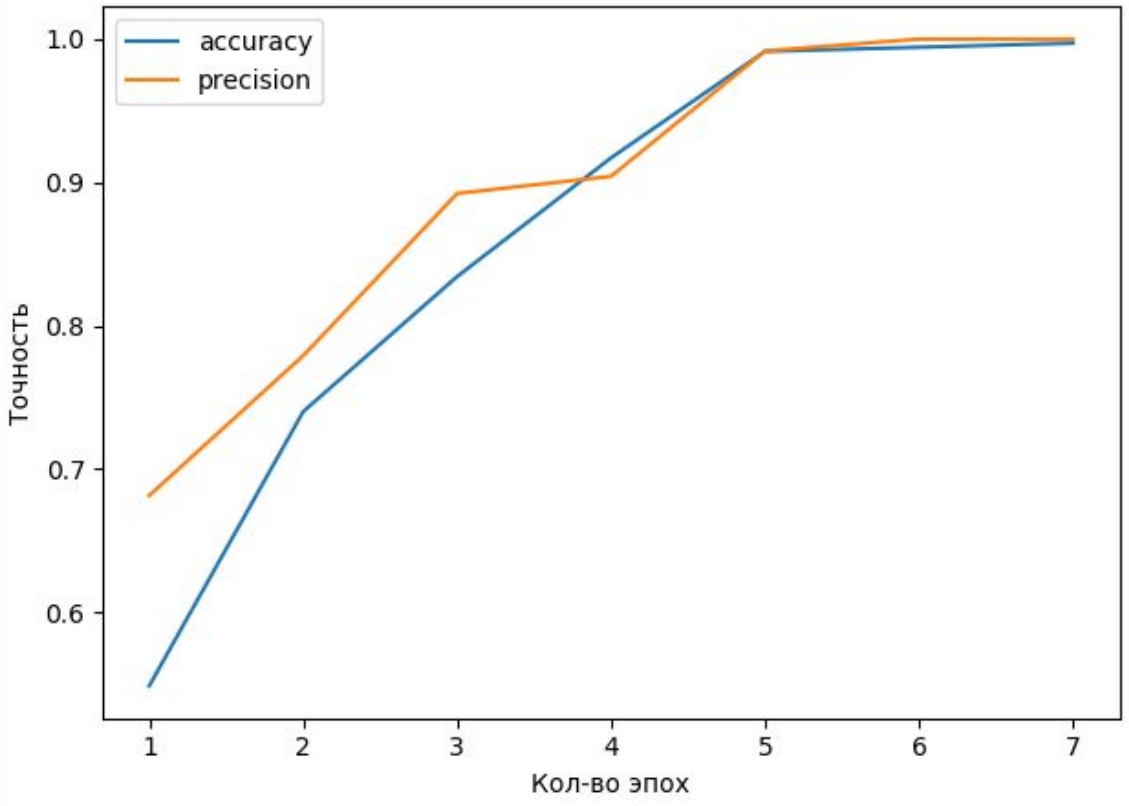


Рисунок 5. Изменение показателей метрик.

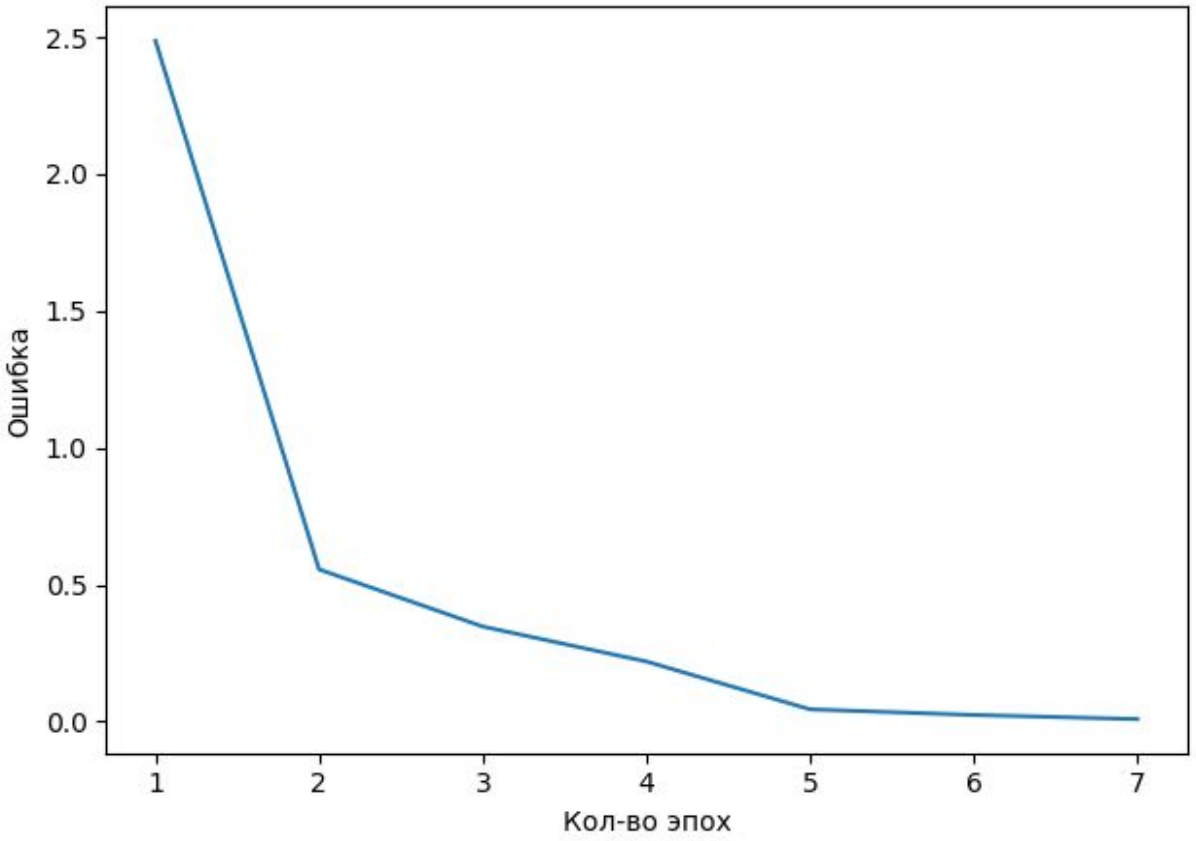


Рисунок 6. Изменение ошибки.

3.2. Обучение бота

Целью обучения бота является обучение его выбору одного из двух возможных в рассматриваемой игре действий (игрок подпрыгивает или игрок наклоняется) для максимизации времени до столкновения с препятствием. Перед непосредственно обучением необходимо запустить эмулятор, а после - выбранную игру. В начале обучения, для инициализации игры бот генерирует и выполняет рандомное действие. Имитация нажатия клавиши, соответствующей выбранному действию, реализована с помощью библиотеки Pyautogui функцией `pyautogui.press()` с использованием следующих функций и вспомогательных переменных, являющихся списком действий:

```
def botAgent(action):
    k = getKey(action)
    pyautogui.press(k)
```

```
def getKey(action):
    curkey = botControl[actionsList.index(action)]
    return curkey

actionsList = ['jump', 'roll']
botControl = ['up', 'down']
```

Далее обрабатывается кадр игры для формирования вектора состояния и расчета функции вознаграждения, чтобы получить информацию об эффективности совершенного действия. Затем начинается основной цикл обучения. В нем бот выполняет действие, предсказанное нейросетью, или, с вероятностью ε=0.1, выбранное рандомно. Следом происходит обработка кадра для получения вектора состояния, значения функции вознаграждения и пересчет весов нейросети. Раз в 1000 итераций веса обучающейся сети сохраняются. При обучении с подкреплением в качестве оптимизатора использовался Adam.

Для функции вознаграждения были взяты следующие значения: -10 - за проигрыш, +1 - за успешное продолжение игры.

При обучении бота используется правило:

```
reward                                                    +
discount*numpy.max(model.predict(state))*alive
```

Результаты численного эксперимента представлены в таблице 2.

№	Время обучения	Очки
1	7 часов	30
2	13 часов	50
3	21 час	90

Таблица 2. Результаты обучения бота.

Из таблицы 2 видно, что чем дольше продолжается обучение, тем лучше результат, но он все еще достаточно небольшой по сравнению с тем, который может достичь человек. Однако уже можно судить об эффективности обучения.

Архитектура нейросети, используемой при обучении бота представлена в таблице 3.

№ уровня	Тип уровня	Размерность выходного вектора	Функция активации
1	Convolution	32	Relu
2	Convolution	64	Relu
3	Convolution	64	Relu
4	Fully connected	512	Relu
5	Fully connected	количество возможных действий бота	Sigmoid

Таблица 3. Архитектура нейронной сети при обучении бота.

Выводы

Так как в рамках выпускной квалификационной работы были реализованы несколько вариантов для решения одних и тех же задач, далее приведено их сравнение и обоснования почему были реализованы эти варианты и каким из них было отдано предпочтение. Поиск по шаблону показал свою работоспособность и пригодность, но из-за возникших проблем с производительность, в связи с которыми время расчета вектора состояния необходимо было сократить, было решено воспользоваться иным способом, для достижения такого же конечного результата. Этим способ было выбрано обучение с учителем, а полученная после обучения модель нейронной сети применялась при создании векторов состояния в процессе обучения бота.

При определении счета также было рассмотрено несколько вариантов решения задачи. Первый подход при разработке классификатора, основанного на библиотеке dlib и методе опорных векторов, и обучении распознавать его все цифры оказался менее точным за счет похожести некоторых цифр и не таким удобным для получения финального результата в виде числа очков. Второй способ его обучения и получении в результате отдельных детекторов для каждой цифры был точнее, и более удобным для объединения цифр в нужное число. Так как оба способа отрабатывали за приемлемо, но существенное время и так как классификатор на основе метода опорных векторов показал свою точность, то было решено найти другой вариант реализации. Для дальнейшей работы был взята библиотека scikit-learn. Классификатор, реализованный и обученный на ее основе, использовался при определении счета игры и обучении бота.

Обучение бота на основе захвата кадра игры и его обработки выбранными методами, для получения векторов состояния, успешно реализовано и были получены удовлетворительные результаты.

Заключение

В ходе работы были изучены теоретические основы разработки ботов и применяемых методов, в частности нейронные сети и обучение с подкреплением. Были реализованы поиск по шаблону, два классификатора с применением метода опорных векторов, обучение с учителем и обучение с подкреплением. Среди этих методов были выбраны наиболее подходящие для формирования вектора состояния в процессе обучения бота. Далее было реализовано непосредственно обучение бота, во время которого из кадров игры получалась информация с использованием ранее реализованных и отобранных для этого способов. Проведенный эксперимент показал, что тренировка бота, основанная не на адаптации исходного кода и непосредственной работе с игрой, а с использованием захвата кадра игры, работоспособна и эффективна.

В дальнейшем можно рассмотреть другие вариант получения кадра игры, не захватом экрана запущенной через эмулятор игры, а напрямую с телефона.

Список литературы

1. М. Тим Джонс, Программирование искусственного интеллекта в приложениях. 2 изд. ДМК Пресс, 2015. 312 с.
2. Using Reinforcement Learning techniques to build an AI bot for the game Flappy Bird [Электронный ресурс]: URL: https://medium.com/@videshsuman/using-reinforcement-learning-techniques-to-build-an-ai-bot-for-the-game-flappy-bird-30e0fd22f990
3. How I built an AI to play Dino Run [Электронный ресурс]: URL: https://medium.com/acing-ai/how-i-build-an-ai-to-play-dino-run-e37f37bdf153
4. Minichino J., Howse J., Learning OpenCV 3 Computer Vision with Python. Packt Publishing, 2015. 268 с.
5. Levy L., Novak J., Game Development Essentials Game Qa & Testing. 1 изд. Cengage Learning, 2009. 320 с.
6. Shpigor I., Practical Video Game Bots: Automating Game Processes using C++. 1 изд. Apress, 2018. 344 с.
7. Хайкин С., Нейронные сети. Полный курс. 2 изд. Вильямс, 2018. 1104 с.
8. Richard S. Sutton, Andrew G. Barto, Reinforcement Learning: An Introduction. 2 изд. A Bradford Book, 2018. 522 с.
9. A Beginner's Guide to Deep Reinforcement Learning [Электронный ресурс]: URL: https://skymind.ai/wiki/deep-reinforcement-learning (дата обращения: 21.10.18).

10. Воронцов К. В. Лекции по методу опорных векторов [Электронный ресурс]: URL: http://www.ccas.ru/voron/download/SVM.pdf (дата обращения: 25.11.18).

11. Line Runner on Google Play [Электронный ресурс]: URL: https://play.google.com/store/apps/details?id=com.djinnworks.linerunnerfree (дата обращения: 21.01.19).

12. Rosebrock A., Practical Python and OpenCV: An Introductory, Example Driven Guide to Image Processing and Computer Vision. 2 изд. PyImageSearch, 2014. 160 с.

13. OpenCV: Object Detection [Электронный ресурс]: URL: https://docs.opencv.org/3.4.0/df/dfb/group__imgproc__object.html (дата обращения: 21.10.18).

14. Geron A., Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools and Techniques to Build Intelligent Systems. 1 изд. O'Reilly Media, 2017. 574 с.