Санкт-Петербургский государственный университет

Кафедра математического моделирования

энергетических систем

Байшев Олег Михайлович

Выпускная квалификационная работа бакалавра

Прогнозирование исходов спортивных событий с использованием методов машинного обучения

Направление 01.03.02 «Прикладная математика и информатика»

ООП «Прикладная математика, фундаментальная информатика и программирование»

Профиль «Дискретная математика и математическое

программирование»

Научный руководитель,
доктор физ.-мат. наук,
профессор ММЭС

Крылатов А.Ю.

Санкт-Петербург

2020

Оглавление

Введение

Футбол — это одна из самых популярных игр в мире. Большая часть ставок в сфере беттинга приходится именно на этот вид спорта [1]. В сфере спортивного прогнозирования представляется логичным использовать большое число статистической информации, доступной исследователям: прошлые результаты команды, показатели игроков в различных категориях, расширенные протоколы командных действий, а также дополнительную информацию, такую как количество перелетов команды, погода во время матчей, наем нового персонала и т.д. — все это позволяет различным заинтересованным сторонам оценить шансы клуба на победу в будущих матчах. Эта информация важна из-за финансовой составляющей: букмекеры и ставочные игроки заинтересованы и конкурируют в предварительном приближении шансов.

Глубинные нейронные сети успешно применяются во многих областях науки, бизнеса и промышленности для извлечения информации из больших объемов данных. Однако использование методов глубокого обучения в сферах спорта и спортивной аналитики довольно ограничено. Наиболее широко эти методы применяются в сфере компьютерного спорта по причинам упрощенного доступа к различным показателям и большого их количества. Тем не менее, уже сейчас спортивные организации все чаще обращаются к данным, понимая, что в них содержится огромное количество неиспользованной и неинтерпретированной информации. Интерес к методам использования этих данных неуклонно растет.

Предсказывая исход матча между двумя футбольными командами, человек, как правило, принимает во внимание определенные факторы, такие как результаты команд за определенный период, место проведения матча

(дома или в гостях), составы команд и недавние трансферы игроков и т.д. Проблема предсказания исхода человеком в том, что на его решение различные факторы будут влиять различным образом и помимо его воли добавятся такие факторы, как личные предпочтения: отношение к отдельным игрокам, тренерам или даже цвет формы команд.

Постановка задачи

Имеется набор данных, в котором содержатся результаты футбольных матчей (футбольные матчи — объекты), $X = (x_1, \ldots, x_n)$, имеется l признаков $f_1, \ldots, f_l$.

Признаковое описание объекта — это вектор $f_1(x), \ldots, f_l(x)$, где $x \in X$.

Признаковое описание матчей формирует обучающую выборку - матрицу объектов-признаков F:

$$\begin{pmatrix} f_1(x_1) & \cdots & f_l(x_1) \\ \vdots & \ddots & \vdots \\ f_1(x_n) & \cdots & f_l(x_n) \end{pmatrix}$$

Цель — по объектам обучающей выборки построить модель классификации, основанную на методах глубокого обучения, такую, что: $X \to Y$, где $Y = (y_1, \ldots, y_n)$ — метки верных классов, известные только для обучающей выборки.

Исходя из поставленной цели и темы работы, возникают следующие задачи:

— Обзор существующих методов глубокого обучения.

— Выбор архитектуры нейронной сети, с помощью которой будет производиться прогнозирование.

— Описание набора данных, который будет использоваться для обучения и тестирования работы нейронной сети, и его предобработка.

— Написание модели на языке Python с использованием библиотеки для глубокого обучения Tensorflow.

— Анализ полученных результатов.

Обзор литературы

В прошлом для решения задач прогнозирования исходов футбольных матчей применялись методы, основанные на логистической регрессии [2]. В указанной статье авторами была создана модель, основанная на четырех признаках, содержащихся в наборе данных. На выходе, после применения логистической функции, авторы получали вектор вероятности победы хозяев поля. Если значение было больше 0,5 победителями считались хозяева, в противном случае — гости. Удалось добиться точности предсказания около 69,5%.

В работе [3] статистические методы применяются для анализа матчей Американской футбольной лиги. Автором применяются t-критерий Стьюдента, дисперсионный анализ, коэффициент корреляции Пирсона для нормально распределенного набора данных. U-критерий Манна -Уитни для непараметрических данных.

В 2011 году выпущена работа [4], нацеленная на отбор признаков, которые будут использоваться для прогнозирования, так как во многом сложность прогнозирования состоит в огромном числе признаков, которыми характеризуется современный футбол. Изначальный набор данных содержал 30 признаков, в ходе работы авторами было отобрано 20. Были созданы модели, основанные на наивном байесовском классификаторе, методе К-ближайших соседей, случайных лесах и искусственных нейронных сетях. Наибольшую точность показала нейросетевая модель — порядка 65%.

В 2014 году для прогнозирования применяются методы data mining [5]. Авторы используют информацию, доступную в футбольном симуляторе FIFA, и, выделяя новые признаки, используют их для прогнозирования исходов реальных футбольных матчей. Им удалось добиться до 75% точности, тем самым показав, что информация, накопленная разработчиками видеоигр, может использоваться для решения реальных проблем.

Переходя к методам, связанным с глубоким обучением, рассмотрим работу [6], в которой авторы рассматривают рекуррентные нейронные сети (в частности, LSTM) для прогнозирования в ряде задач. Эти задачи характеризуются общим свойством: информация в них поступает через определенные промежутки времени, то есть исходные данные для таких задач являются последовательностями. Авторы предлагают улучшение архитектуры LSTM, которая, в свою очередь, является улучшением обычных рекуррентных нейронных сетей.

В статьях [7] содержится подробный разбор принципов функционирования LSTM сетей.

Русскоязычный обзор истории развития, методов и достижений нейросетевой науки представлен в книге [8].

Глава 1. Обзор методов глубокого обучения

Цель этой главы — описать теорию, необходимую для понимания проведенной работы. Она начинается с введения в машинное обучение, нейронные сети и глубокое обучение.

1.1 Машинное обучение

Машинное обучение — это раздел искусственного интеллекта, популярность которого в последние годы лет возросла (особенно в таких областях, как нейронные сети и методы глубокого обучения, которые, в свою очередь, являются подразделами машинного обучения) как в научных исследованиях, так и в промышленности. В отличие от традиционного искусственного интеллекта, когда алгоритм представляет собой список предопределенных правил, машинное обучение пытается использовать большие объемы данных для обучения прогнозированию. Многие проблемы невозможно решить с помощью заранее заданного списка правил из-за огромного количества ограничений или сложности проблемы в целом. Характерной черной машинного обучения является не прямое решение задачи, а обучение на доступных решениях множества сходных задач.

1.2 Нейронные сети

Искусственные нейронные сети (Artificial Neural Networks, ANN) — это подход, основанный на том, как функционируют биологические нейронные сети. Человеческий мозг состоит из нервных клеток, называемых нейронами, которые связаны друг с другом аксонами. ANN состоят из нескольких узлов, которые имитируют биологические нейроны человеческого мозга. Нейроны связаны звеньями, которые имитируют биологические аксоны, и они

взаимодействуют друг с другом. Каждый узел принимает входные данные, выполняет простую операцию и передает результат другим узлам. Как и биологический мозг, ANN способны обучаться, и поэтому могут использоваться в тех областях, где решение трудно найти с помощью традиционных подходов.

Основа искусственных нейронных сетей — это нейроны и функция активации, которая имеет несколько входов и один выход. Нейрон можно рассматривать как композицию других взвешенных нейронов, к нейронам применяется функция активации, а сеть образуется, когда нейроны объединяются в слои. Сеть можно описать функцией

$$\hat{f}(x) = K(\sum_i \omega_i g_i(x))$$

ω_i— весовые коэффициенты, g_i — функции выхода других нейронов, K — нелинейная функция активации, такая как логистическая функция, гиперболический тангенс или ReLU ($K(x) = max(0, x)$).

Предполагая, что между входными данными x и выходными y, существует функциональная зависимость,

$$y = f(x)$$

необходимо построить такую функцию $\hat{f}(x)$, что:

$$|| \hat{f}(x_k) - f(x_k) || \leq \varepsilon_1 \; \forall \, k \qquad (1.1)$$

где $\{(x_k, y_k\}_{k=1}^{N}$ — объекты обучающего множества, а ε_1 задано наперед.

Главной задачей является выполнение условия:

$$|| \hat{f}(x) - f(x) || \leq \varepsilon_2 \, \forall \, x \in X \qquad (1.2)$$

Основные проблемы для исследователей — недообучение и переобучение.

Недообучение — нарушение условия (1.1). Это значит, что сложности построенной сети не хватило для описания набора данных.

Переобучение — нарушение (1.2). В этом случае сеть оказывается слишком сложной. При этом ε_2 либо выбрано слишко малым, либо обучающего множества оказывается недостаточно для построения функциональной зависимости.

Полносвязная многослойная сеть прямого распространения состоит из нескольких слоев нейронов, соединения в которых идут от одного слоя к другому. Первый слой называется входным, последний — выходным, а все промежуточные слои называются скрытыми. Глубинная нейронная сеть может иметь несколько сотен скрытых слоев.

Нейронная сеть с хотя бы одним скрытым слоем с конечным числом нейронов в этом слое может аппроксимировать любую непрерывную функцию. Это известно из теоремы об универсальной аппроксимации [9], и это является одной из причин, почему можно полагать, что нейронные сети могут использоваться для искусственного интеллекта в целом.

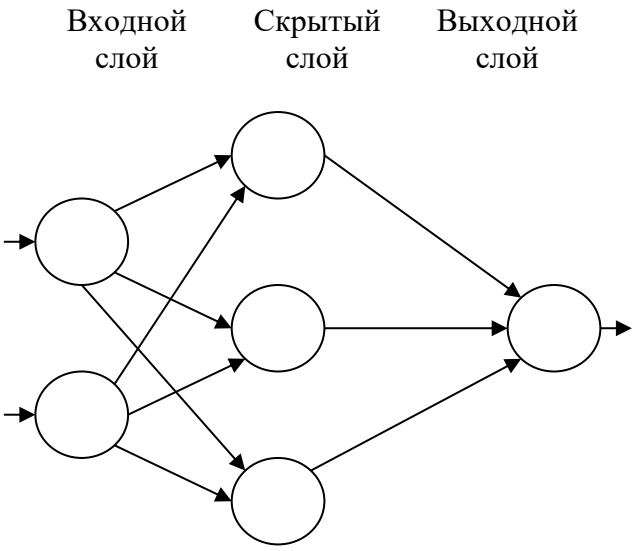

Рисунок 1.1: Пример полносвязной многослойной сети прямого распространения из трех слоев. Два нейрона во входном слое, три на скрытом и один на выходном.

1.3 Глубокое обучение

Глубокое обучение — это метод, основанный на глубоких нейронных сетях (Deep Neural Networks, DNN), виде искусственных нейронных сетей. Различие в том, что глубокие нейронные сети имеют несколько (вплоть до нескольких сотен) скрытых слоев. За последние несколько лет были достигнуты большие успехи в области глубокого обучения. Две наиболее известных статьи: о сети DeepMind, которая играет в игры Atari 2600 [10], и о сети, которая победила чемпиона мира в настольной игре GO [11].

1.4 Рекуррентные нейронные сети

Отличительной особенностью обычных искусственных нейронных сетей является то, что входы и выходы могут иметь различную, но фиксированную длину.

Если на вход нейронной сети подаются изображения в векторной форме, то все они должны иметь одинаковую размерность, а выход, например, при задаче классификации будет вектором длины, равной числу классов, в каждой из компонент которого содержится вероятность отнести изображение к одному из классов. Рекуррентные нейронные сети (Recurrent Neural Network, RNN) решают проблему фиксированных размеров входов и выходов.

В рекуррентных сетях связи между нейронами могут идти не только от предыдущего слоя к следующему, но и от нейрона к предыдущему значению этого же нейрона или других нейронов того же слоя, используя информацию о том, что происходило с ним самим на предыдущих входах, создавая некое подобие работы памяти.

Это позволяет использовать RNN для задач, в которых важно учитывать информацию о прошлых состояниях. Было показано, что этот подход очень хорошо работает для задач обработки естественного языка. [12, 13]

Получая последовательность $x = (x_1, x_2, \ldots, x_r)$, RNN обновляет свое скрытое состояние h_t (см. рис. 1.2):

$$h_t = f(h_{t-1}, x_t),$$

где h_t — новое значение состояние, h_{t-1} — предыдущее значение, x_t — вектор входных значений.

Проблема обыкновенных рекуррентных нейронных сетей — это затухающий или взрывающийся градиент. Градиент рассчитывается с использованием правила вычисления производной сложной функции, и при

многократном умножении малых чисел значения градиента уменьшается экспоненциально. То же самое происходит, когда значения производных слишком велики.

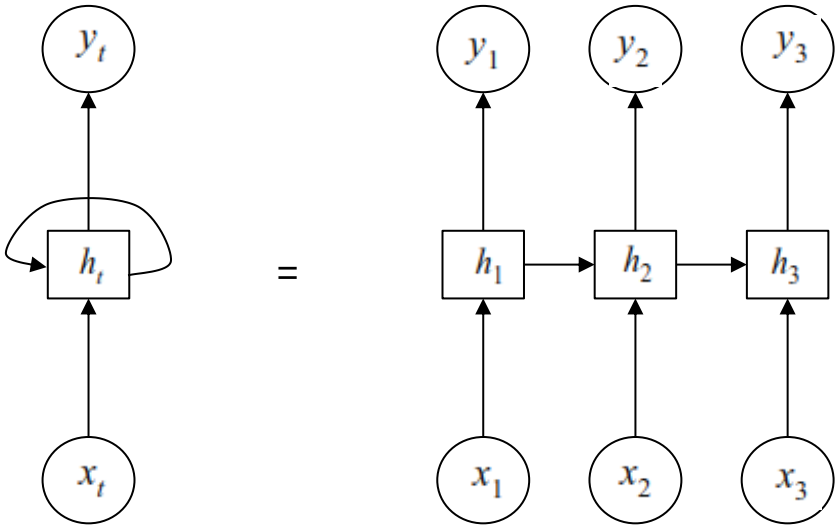

Рисунок 1.2. Пример рекуррентной нейронной сети.

1.5 Long Short-Term Memory

LSTM — это специальная ячейка, отвечающая за “долгую память” и содержащая процессы, отвечающие за запись, чтение из этой ячейки.

Она состоит из трех видов узлов, называемых гейтами (gate): входной (input gate), выходной (output gate) и забывающий (forget gate) и обыкновенная рекуррентная ячейка со скрытым состоянием h_t:

$$h_t{}^j = \sigma_t{}^j tanh(c_t{}^j),$$

где σ_t^j — output gate, который вычисляется следующим образом:

$$\sigma_t^j = \sigma(W_\sigma x_t + U_\sigma h_{t-1} + V_\sigma c_t)^j,$$

где σ — нелинейная функция, V_σ — диагональная матрица, W_σ и U_σ — матрицы весов.

Ячейка памяти $c_t{}^j$(cell) обновляется так:

$$c_t{}^j = f_t{}^j c_{t-1}{}^j + i_t{}^j s_t{}^j,$$

где ${f_t}^j$ — forget gate, ${s_t}^j$ является кандидатом на новое значение ячейки памяти и вычисляется:

$$ {s_t}^j = tanh(W_c x_t + U_c h_{t-1})^j $$

Forget gate вычисляется как

$$ {f_t}^j = \sigma(W_f x_t + U_f h_{t-1} + V_f c_{t-1})^j, $$

a input gate:

$$ {i_t}^j = \sigma(W_i x_t + U_i h_{t-1} + V_i c_{t-1})^j $$

Таким образом, LSTM-ячейка “решает”, какую часть информации из прошлого следует сохранить.

1.6 Дропаут

Одним из важнейших методов регуляризации искусственных нейронных сетей является дропаут (dropout) [14]. Для нейронных сетей с большим числом параметров существует проблема переобучения, когда алгоритм “подстраивается” под обучающую выборку, при этом показываю плохую обобщающую способность на тестовых данных.

Идея дропаута состоит в следующем: для каждого нейрона существует вероятность p, с которой его выход будет приравнен к нулю — это приведет к тому, что алгоритмы прямого и обратного распространения ошибки остановятся на этом нейроне, то есть, фактически, он будет удален из сети.

На рисунке 1.2(б) изображена трехслойная нейронная сеть 1.2(а), к которой был применен дропаут. На выходном слое дропаут не применяется, потому что размерность выхода должна быть фиксирована. Таким образом, на каждой эпохе обучения исходная нейронная сеть немного изменяется — происходит усреднение 2^N(N — число удаленных или сохраненных нейронов) архитектур нейронных сетей.

В [15] показано, что применение дропаута серьезно улучшает различные нейросетевые модели.

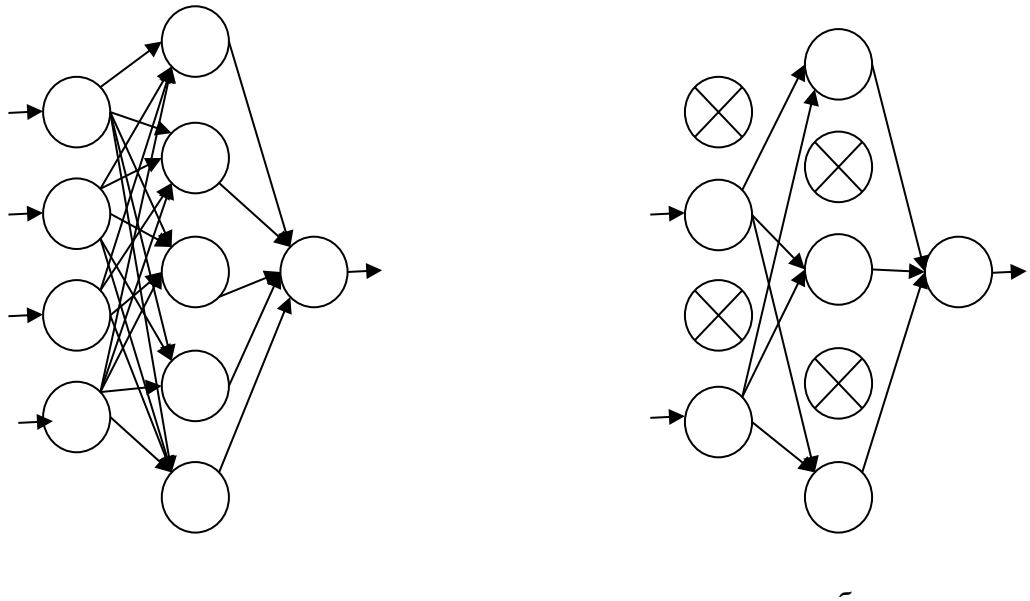

Рис. 1.2. Пример нейронной сети: а — без дропаута, б — с дропаутом.

1.7 Softmax

Softmax — это многомерное обобщение логистической функции. Результатом применения функции к вектору размерности K станет вектор той же размерности K, но значения его координат будут находиться в диапазоне $[0,1]$, а их сумма равняться 1.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}},$$

где $i = 1, \ldots, K$.

Функция softmax используется для различных задач классификации, когда число классов больше двух. Координаты σ_i при этом интерпретируются как вероятности принадлежности классифицируемого объекта к i-му классу

В классификаторах, основанных на нейронных сетях, softmax часто используется на последнем слое. В этом случае в качестве функции потерь используется перекрестная энтропия.

1.8 Перекрестная энтропия

Одна из возможных функций ошибки, которую следует оптимизировать во время обучения нейронных сетей, это перекрестная энтропия. Она описывает энтропию распределения y' относительно распределения y и меру того, как много бит нужно, чтобы закодировать события из “истинного” распределения y, основываясь на имеющемся распределении y'.

$$H(y, y') = -\sum_i y_i \, log \, y'_i$$

1.9 Стохастический градиентный спуск (Adam)

Adam (Adaptive Moment Estimate) — это улучшение алгоритма стохастического градиентного спуска. Этот метод объединяет в себе идеи AdaGrad [16] и RMSprop [17]. Для каждого параметра шаг градиента масштабируется отдельно, а пересчет градиента происходит с учетом масштабирования на некоторый гиперпараметр инерции. Оценки первого и второго моментов задаются нулями. Таким образом, при $\gamma_1, \gamma_2, \lambda, \eta, m_0 = 0, g_0 = 0$ пересчет происходит следующим образом:

$$m_{t+1} = \gamma_1 m_t + (1-\gamma_1) grad(f_i(\theta_t))$$

$$g_{t+1} = \gamma_2 g_t + (1-\gamma_2) grad(f_i(\theta_t))^2$$

$$\overline{m_{t+1}} = \frac{m_{t+1}}{1-{\gamma_1}^{t+1}}$$

$$\overline{g_{t+1}} = \frac{g_{t+1}}{1-{\gamma_2}^{t+1}}$$

$$\theta_{t+1} = \theta_t - \frac{\eta \overline{m_{t+1}}}{\sqrt{\overline{g_{t+1}} + \varepsilon}}$$

Глава 2. LSTM-модель прогнозирования

2.1 Описание модели

Для экспериментов была выбрана рекуррентная нейронная сеть с LSTM-ячейками. Для прогнозирования результатов футбольных матчей, которые проводятся на каком-то временном отрезке, представляется важным, чтобы нейросеть с выбранной архитектурой могла во время обучения использовать данные о прошлых матчах и результатах соревнующихся команд. Этой особенностью обладают, как показано выше, рекуррентные нейронные сети. Минусом классической рекуррентной архитектуры является то, что сеть не способна “запомнить” данные о предыдущих состояниях надолго. Так выбранный набор данных охватывает период в несколько лет, за который были сыграны тысячи матчей, это может негативно влиять на качество прогнозирования, поэтому в архитектуру были добавлены LSTM-ячейки, решающие эту проблему. Информация о принципах функционирования рекуррентный нейронных сетей и LSTM-ячеек представлена в главе 2.

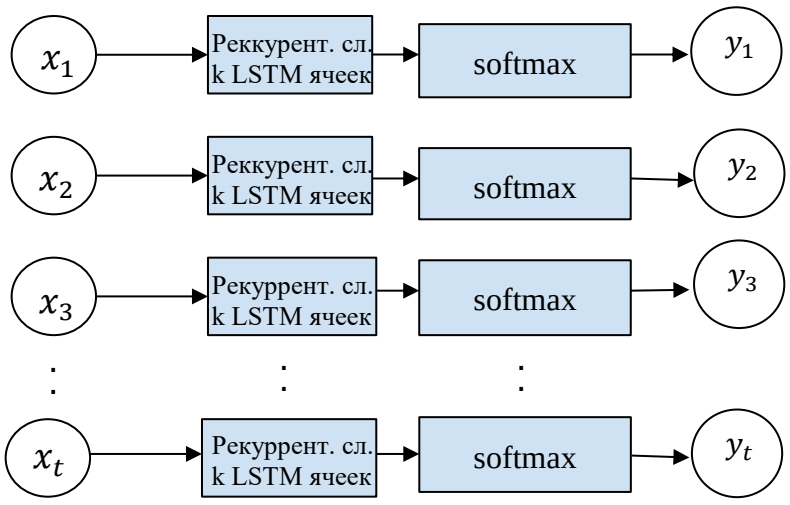

Рис. 2.1. Выбранная архитектура нейронной сети. Рекуррентная нейронная сеть с k LSTM ячейками в рекуррентном слое.

На последнем слое к выходам рекуррентного слоя применяется softmax функция.

В качестве функции потерь была выбрана перекрестная энтропия, используемый метод оптимизации — Adam.

Входными значениями для нейронной сети являются вектора, с компонентами, содержащими закодированные значения заранее определенных признаков.

Выходные значения — вектор размерности 2, в компонентах которого содержатся вероятности победы хозяев поля и иного исхода (ничья или победа гостей).

2.2 Настраиваемые параметры

Гипер-параметрами модели являются:

— Размер батча — количество строк данных, подающихся на вход модели за одну итерацию. Этот параметр определяет количество входных данных (подмножества обучающих данных), по которым будет рассчитываться градиент, необходимый для обратного распространения ошибки.

— Число эпох. Одна эпоха — это полное прохождение всех обучающих данных через нейронную сеть один раз. Недостаток эпох приводит к недообучению (слабая обобщающая способность модели как на тренировочных, так и на тестовых данных), переизбыток — к переобучению (модель слишком сильно “подстроилась” под обучающие данные, при этом показывая плохую эффективность на тестовых). На вопрос о единственно

верном количестве эпох ответить заранее невозможно, этот параметр подбирается экспериментально.

— Длина последовательности. Так как входные данные являются вектором, в компонентах которого содержатся признаки, то длина этого вектора также является настраиваемым параметром.

— Число LSTM-ячеек. Количество Long Short-Term Memory блоков, расположенных в скрытом слое рекуррентной сети.

Глава 3. Модель прогнозирования исходов футбольных матчей

3.1 План работы

По набору данных, содержащим различную статистическую информацию о прошедших футбольных матчах, необходимо с помощью нейросети глубинного обучения выбранной архитектуры составить систему прогнозирования результатов будущих матчей.

План работы представлен на рисунке 2.1.

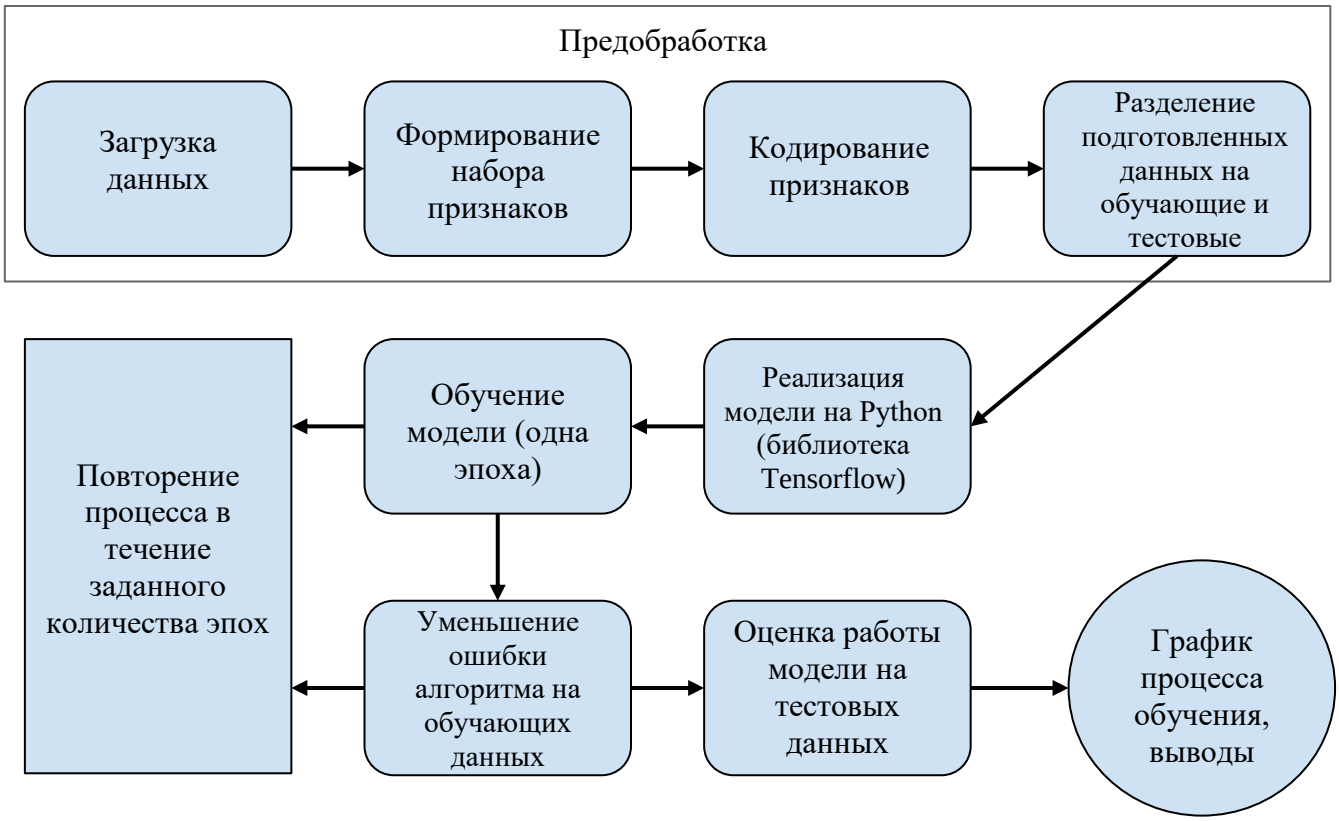

Рис. 2.1. Схема выполнения практической части исследования.

3.2 Данные

Набор данных для прогнозирования результата футбольных матчей был взят с ресурса [18]. В нем содержится информация о матчах английской Премьер-лиги с сезона 2010-11 по сезон 2018-19. Большое преимущество этого набора данных в том, что число матчей, проведенных за весь турнир, и число матчей, сыгранных каждой командой, являются фиксированными. Это облегчает подготовку данных и удаление лишней информации. Набор данных за каждый сезон содержит 380 строк, соответствующих сыгранным матчам, и 65 признаков; информация о матчах расположена в хронологическом порядке. Из-за вышеуказанной особенности существенно облегчается задача получения новых признаков и удаления ненужной информации, путем работы напрямую со строками и столбцами исходного набора данных.

Следующим шагом было формирование набора признаков, на которых будет обучаться глубинная нейронная сеть. В загруженных данных не содержалось исторических данных, они были рассчитаны в процессе предобработки.

Как итог, были сформированы следующие признаки:

— результат матча

— забито голов хозяевами

— забито голов гостями

— пропущено голов хозяевами

— пропущено голов гостями

— число очков у хозяев

— число очков у гостей

— результаты прошлых матчей хозяев (вплоть до пятого)

— результаты прошлых матчей гостей (вплоть до пятого)

— наличие серии из 3-х побед у хозяев

— наличие серии из 5-и побед у хозяев

— наличие серии из 3-х побед у гостей

— наличие серии из 3-х побед у гостей

— разница забитых и пропущенных голов хозяев

— разница забитых и пропущенных голов гостей

— разница очков гостей и хозяев

— разница числа очков, заработанных за последние пять матчей хозяевами и гостями

— разница позиций играющих команд в рейтинге

Следующим шагом была нормализация входных значений. Установлено, что это ускоряет процесс обучения нейронных сетей [19].

Далее необходимо было произвести кодирование строковых и категориальных признаков. Для этого использовались методы OneHotEncoder и LabelEncoder библиотеки sklearn.

Разделение данных на обучающие и тестовые необходимо для проверки качества реализованной модели. Стоит отметить, что перемешивание данных в случае прогнозирования результатов футбольных матчей невозможно из-за того, что временная структура в этом случае имеет значение. Таким образом, обучающие данные соответствуют “исторической информации”, результат матчей в которой нам уже известен, а тестовые — условному “будущему”. На момент начала матча из “будущего” мы владеем информацией о всех признаках, но не знаем, чем он закончится.

3.3 Эксперименты и результаты

3.3.1 Библиотеки глубокого обучения

Существует несколько наиболее популярных библиотек для упрощения процесса реализации алгоритмов глубокого обучения на языке Python — Tensorflow, PyTorch, Keras. Они предоставляют инструменты для создания, обучения и настройки различных нейросетевых архитектур. Для работы была использована библиотека Tensorflow. Библиотека автоматического дифференцирования Tensorflow была выбрана, потому что она хорошо документирована и популярна, что облегчает процесс написания программной реализации.

3.3.2 Настройка гипер-параметров

Наилучшие значения параметров находились путем перебора: сравнения качества модели для различных их комбинаций. В итоге были выбраны:

— размер батча — 1

— число эпох — 10

— длина последовательности — 27

— число LSTM-ячеек — 64

3.3.3 Результаты

С выбранными гипер-параметрами модель показала следующий показатель качества (accuracy, точность предсказания, отношение числа верно классифицированных объектов к их количеству).

Точность на обучающих данных: 0.9811, точность на тестовых данных: 0.8075

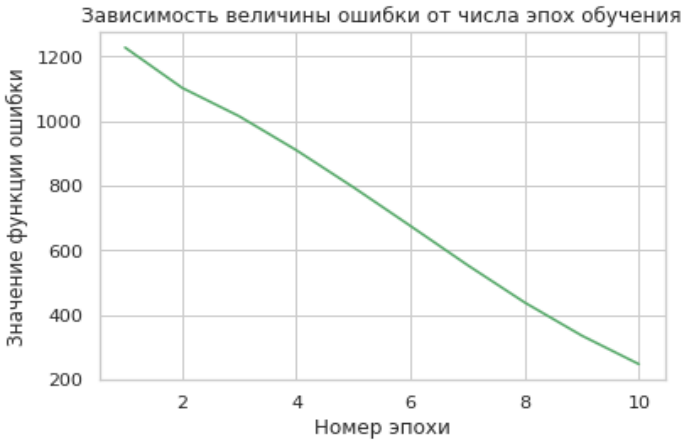

Рис 2.2. График зависимости значения функции ошибки от числа эпох обучения.

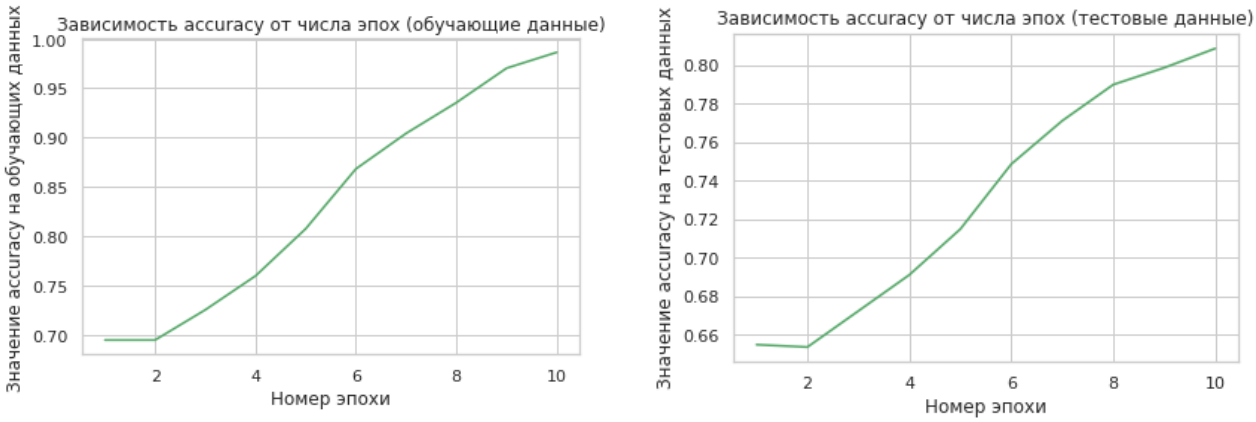

Рис. 2.3. Графики точности прогнозирования в зависимости от числа эпох обучения.

Таким образом, показатель качества модели на тестовых данных выше, чем у моделей из представленного в данной работе обзора литературы. Можно сделать вывод, что рекуррентные нейронные сети и их LSTM модификации имеют существенное преимущество перед остальными моделями прогнозирования исходов футбольных матчей.

Заключение

В данной работе была исследована возможность прогнозирования исходов футбольных матчей с использованием методов глубокого обучения.

В ходе работы:

— рассмотрены некоторые методы глубокого обучения

— рассмотрена архитектура нейронной сети, выбранная для прогнозирования

— описаны и предобработаны данные о футбольных матчах, необходимые для обучения нейронной сети

— на языке Python с использованием библиотеки глубокого обучения Tensorflow была реализована нейросеть выбранной архитектуры и выполнено прогнозирование исходов матчей из набора данных

Таким образом, были выполнены цели, поставленные в начале данной работы.

Говоря о возможностях улучшения созданной системы прогнозирования, следует отметить:

— использование набора данных с большим числом признаков, таких как статистические показатели отдельных игроков; это поможет отслеживать форму команды в целом, опираясь на индивидуальную форму игроков состава

— использование большего объема исходных данных (информации о большем числе сыгранных матчей) также может положительно сказаться на качестве модели

Список литературы

1. Football betting - the global gambling industry worth billions. URL: URL: https://www.bbc.com/sport/football/24354124
2. Prasetio, D. Predicting football match results with logistic regression // In Advanced Informatics: Concepts, Theory, And Application (ICAICTA), 2016. IEEE.
3. Bailey, M.J. Predicting Sporting Outcomes: A Statistical Approach, 2005 // Swinburne University of Technology: Faculty of Life and Social Sciences.
4. Hucaljuk J., Rakipovi A. Predicting football scores using machine learning techniques, 2011 // Proc. of the 34th International Convention. P. 1623-1627.
5. Shin J., Gasparyan R. A Novel Way to Soccer Match Prediction, 2014 // Stanford University Department of Computer Science.
6. Gerc F.A., Schraudolph N.N., Schmidhuber J. Learning precise timing with LSTM recurrent networks // Journal of machine learning research, 2002. P. 115-143.
7. Understanding LSTMs. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
8. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. СПб.: Питер, 2018. 480 с.
9. Горбань А.Н. Обобщенная аппроксимационная теорема и вычислительные возможности нейронных сетей // Сибирский журнал вычислительной математики. 1998. 1, 1. 11–24.
10. Volodymyr Mnih. Playing atari with deep reinforcement learning. // arXiv, 2013.
11. David Silver. Mastering the game of Go with deep neural networks and tree search. // Nature vol. 529, 2016. P. 484–489.
12. Kyunghyun Cho. Learning Phrase Representations using RNN Encoder Decoder for Statistical Machine Translation. // arXiv, 2014.

13. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. // arXiv, 2014.
14. G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. // Journal of Machine Learning Research, 2014. P. 1929-1958.
15. G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. R. Salakhutdinov. Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors // arXiv, 2012.
16. Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization."The Journal of Machine Learning Research 12 (2011): 2121-2159.
17. Tieleman, Tijmen, and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude."COURSERA: Neural Networks for Machine Learning 4 (2012): 2.
18. Football data. URL: https://www.football-data.co.uk/
19. LeCun, Yann A. Efficient backprop. Neural networks: Tricks of the trade. // Springer Berlin Heidelberg, 2012. P. 9-48

Приложение

Программный код на языке Python

```
hm_epochs=20

n_classes = 2

batch_size = 1

chunk_size=27

n_chunks=1

rnn_size=64


xx = tf.placeholder('float', [None, n_chunks,chunk_size])

y = tf.placeholder('float')

def recurrent_neural_model(x):

    layer =
{'weights':tf.Variable(tf.random_normal([rnn_size,n_classes])),

             'biases':tf.Variable(tf.random_normal([n_classes]))}

   x=tf.transpose(x,[1,0,2])

    print("transpose",x)

    x=tf.reshape(x,[-1,chunk_size])

    print("reshape",x)

    x=tf.split(x,n_chunks)

    print("split",x)

    lstm_cell = rnn.BasicLSTMCell(rnn_size)

    outputs, states = rnn.static_rnn(lstm_cell, x, dtype=tf.float32)

   output = tf.matmul(outputs[-1],layer['weights']) + layer['biases']

  return output
```

```
      def train_neural_network(x):

          prediction = recurrent_neural_model(x)

          cost = tf.reduce_mean(
tf.nn.softmax_cross_entropy_with_logits(logits=prediction,labels=y) )

          optimizer = tf.train.AdamOptimizer().minimize(cost)

         with tf.Session() as sess:

              sess.run(tf.global_variables_initializer())

              hm_epochs=10

              losses = []

              accuracies = []

              accuraciesT = []

              for epoch in range(hm_epochs):

                  epoch_loss = 0

                  for i in range(0,data.shape[0],batch_size):

                      epoch_x, epoch_y =
data.iloc[i:i+batch_size,1:28].values,data.iloc[i:i+batch_size,28:].values

                      epoch_x=epoch_x.reshape((batch_size,n_chunks,chunk_size))

                      _, c = sess.run([optimizer, cost], feed_dict={x: epoch_x,
y: epoch_y})

                      epoch_loss += c

                 losses.append(epoch_loss)

                  print('Epoch', epoch, 'completed out
of',hm_epochs,'loss:',epoch_loss)



                  correct = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))



                accuracy = tf.reduce_mean(tf.cast(correct, 'float'))

                  print('Accuracy
Train:',accuracy.eval({x:data.iloc[:,1:28].values.reshape((-
1,n_chunks,chunk_size)),
```

```
y:data.iloc[:,28:].values}))


accuracies.append(accuracy.eval({x:data.iloc[:,1:28].values.reshape((-
1,n_chunks,chunk_size)),


y:data.iloc[:,28:].values}))

                print('Accuracy
Test:',accuracy.eval({x:dataT.iloc[:,1:28].values.reshape((-
1,n_chunks,chunk_size)),


y:dataT.iloc[:,28:].values}))


accuraciesT.append(accuracy.eval({x:dataT.iloc[:,1:28].values.reshape((-
1,n_chunks,chunk_size)),


y:dataT.iloc[:,28:].values}))

            return losses, accuracies, accuraciesT, tf.argmax(prediction, 1)



      k = train_neural_network(x=xx)
```