Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информатика»

Д. В. Прокопенко

ПРОГРАММИРОВАНИЕ НЕЙРОННЫХ СЕТЕЙ

ПРАКТИКУМ
по выполнению лабораторных работ
для студентов специальности 1-40 04 01
«Информатика и технологии программирования»
дневной формы обучения

Гомель 2024

Библиотека ГГТУ им.П.О.Сухого

УДК 004.9:005.8(075.8)
ББК 32.972я73
П80

Рекомендовано научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 9 от 17.05.2023 г.*)

Рецензент: доц. каф. «Информационные технологии» ГГТУ им. П. О. Сухого
канд. техн. наук, доц. *В. И. Токочаков*

Прокопенко, Д. В.

П80 Программирование нейронных сетей : практикум по выполнению лаборатор. работ для студентов специальности 1-40 04 01 «Информатика и технологии программирования» днев. формы обучения / Д. В. Прокопенко. – Гомель : ГГТУ им. П. О. Сухого, 2024. – 58 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

Изложены основные темы, изучаемые на курсе «Программирование нейронных сетей». Приведены основные сведения о программировании нейронных сетей на языке Python.

Для студентов специальности 1-40 04 01 «Информатика и технологии программирования» дневной формы обучения.

УДК 004.9:005.8(075.8)
ББК 32.972я73

© Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», 2024

Библиотека ГГТУ им. П.О.Сухого

Содержание

Библиотека ГГТУ им. П.О. Сухого

Введение

В настоящее время нейронные сети используются в различных областях, включая компьютерное зрение, обработку естественного языка, игровую индустрию и другие. Нейронные сети представляют собой модели машинного обучения, которые позволяют компьютеру обучаться на основе большого количества данных. Они позволяют решать задачи, которые ранее были трудно решаемыми с использованием традиционных алгоритмов.

Программирование нейронных сетей становится все более популярным среди специалистов в области машинного обучения, а также среди начинающих программистов. Однако, разработка нейронных сетей может быть сложной задачей, требующей знания специализированных библиотек и фреймворков.

В данном контексте, цель данного практикума – рассмотреть основные принципы и инструменты программирования нейронных сетей.

Для достижения этой цели изучим основы машинного обучения и нейронных сетей, рассмотрим различные типы архитектур нейронных сетей и принципы их работы. Также рассмотрим популярные библиотеки и фреймворки для программирования нейронных сетей и рассмотрим примеры использования нейронных сетей в различных областях, таких как компьютерное зрение, обработка естественного языка.

Практикум предназначен для студентов, обучающихся по специальности 1-40 04 01 «Информатика и технологии программирования».

Библиотека ГГТУ им. П.О. Сухого

1 Введение в нейронные сети

1.1 Основные понятия

Нейронная сеть – это алгоритм машинного обучения, который имитирует работу нервной системы человека. Она состоит из множества взаимосвязанных узлов (нейронов), которые передают между собой информацию. Каждый нейрон принимает на вход некоторое значение, которое обрабатывается внутри нейрона, а затем передается на выход. В зависимости от значения на выходе нейрона, он может активироваться или оставаться неактивным.

Нейронные сети используются для решения задач классификации, регрессии, кластеризации, обработки изображений и звука, анализа текста и других задач машинного обучения. Главным преимуществом нейронных сетей является их способность к обучению на больших объемах данных, что позволяет получать точные и надежные прогнозы.

Современные нейронные сети могут иметь глубину до нескольких сотен слоев, и поэтому они называются глубокими нейронными сетями или нейронными сетями глубокого обучения. Такие сети обладают высокой вычислительной мощностью и могут решать самые сложные задачи машинного обучения.

Глубокие нейронные сети стали мощным инструментом в области машинного обучения и искусственного интеллекта. Их применение расширяется на различные области, включая компьютерное зрение, обработку естественного языка, робототехнику, автоматическое управление, биоинформатику и многие другие.

В области компьютерного зрения, глубокие нейронные сети широко используются для распознавания образов, сегментации изображений, детектирования объектов и распознавания лиц. В области обработки естественного языка, они применяются для классификации текстов, анализа тональности, машинного перевода и

Библиотека ГГТУ им. П.О. Сухого

генерации текста. В робототехнике, глубокие нейронные сети используются для управления роботами, планирования движения и определения положения объектов. В автоматическом управлении, они могут использоваться для прогнозирования и оптимизации производственных процессов и управления системами энергетики. В биоинформатике, глубокие нейронные сети применяются для распознавания генов, прогнозирования структуры белков и анализа медицинских изображений.

Применение глубоких нейронных сетей в этих областях уже привело к значительным результатам и открытиям, и они продолжают быть активно исследованы и развиваться. В результате, глубокие нейронные сети становятся все более востребованными в широком спектре задач и приложений, и их применение продолжит расширяться в будущем.

Искусственный нейрон МакКаллока-Питтса

Искусственный нейрон МакКаллока-Питтса (*McCulloch-Pitts neuron*) – это математическая модель, которая имитирует работу биологического нейрона. Он состоит из нескольких входов, каждый из которых имеет свой вес и один выход. На каждый вход подается значение, которое умножается на его вес, а затем суммируется с другими взвешенными входами. Если сумма превышает определенный порог, то нейрон активируется и выдает на выходе сигнал со значением 1, в противном случае выходное значение равно 0.

МакКаллок и Питтс разработали эту модель в 1943 году, чтобы показать, что нейронные сети могут быть использованы для моделирования логических операций. Они представили нейрон как булеву функцию, которая может принимать только два значения: 0 или 1.

Хотя модель МакКаллока-Питтса является упрощенной версией биологических нейронов, она была важным шагом в развитии искусственных нейронных сетей. Она вдохновила множество последующих моделей нейронов, которые используются в современных нейронных сетях.

Библиотека ГГТУ им. П.О. Сухого

Функция активации

Функция активации – это нелинейная математическая функция, которая применяется к входным данным нейрона и определяет, должен ли нейрон активироваться и передавать сигнал далее или нет. Функция активации имитирует способность биологических нейронов обрабатывать информацию и принимать решения на основе сигналов, которые поступают к ним.

В нейронных сетях существует множество различных функций активации, каждая из которых подходит для определенного типа задач и типа данных. Некоторые из самых распространенных функций активации включают в себя:

– сигмоидальная функция: она используется для задач классификации с двумя классами и помогает ограничить выходное значение нейрона в диапазон от 0 до 1;

– гиперболический тангенс: подобно сигмоидальной функции, гиперболический тангенс используется для задач классификации с двумя классами, но вместо диапазона от 0 до 1 он выдает значение от -1 до 1;

– *ReLU* (*Rectified Linear Unit*): это одна из наиболее популярных функций активации, используемых в нейронных сетях. Она имеет простую структуру и применяется для ускорения обучения нейронных сетей;

– *Leaky ReLU*: это модификация *ReLU*, которая предотвращает проблему «мертвых нейронов», когда нейрон не активируется на всем протяжении обучения;

– *Softmax*: функция активации, используемая для многоклассовой классификации. Она преобразует выходные значения нейронов в вероятности каждого класса.

Гиперпараметры

Гиперпараметры – это параметры модели машинного обучения, которые не могут быть оптимизированы в процессе обучения, а должны быть установлены перед обучением. Они влияют на характеристики и поведение модели и могут включать в себя параметры, такие как скорость обучения, количество слоев,

Библиотека ГГТУ им. П.О. Сухого

количество нейронов в слоях, функции активации, регуляризацию и т.д.

Выбор гиперпараметров является важной задачей в машинном обучении, так как правильно выбранные значения могут значительно повысить качество модели, а неправильно выбранные могут привести к переобучению или недообучению.

Выбор оптимальных значений гиперпараметров может осуществляться различными способами, например, перебором значений на сетке (*grid search*), случайным поиском (*random search*), оптимизацией гиперпараметров (*hyperopt*), байесовской оптимизацией и другими методами.

Важно также учитывать, что выбор гиперпараметров может зависеть от конкретной задачи и используемых данных, поэтому их необходимо настраивать и оптимизировать для каждой новой задачи.

Правила Хэбба

Правила Хэбба (*Hebb's rule*) – это одно из ключевых понятий, связанных с искусственными нейронными сетями. Это правило формулирует принцип, по которому укрепление связи между нейронами происходит при активации обоих нейронов одновременно.

Согласно правилам Хэбба, если нейрон *A* активируется одновременно с нейроном *B*, то с течением времени связь между ними укрепляется. Это означает, что при повторной активации нейрона *A* он будет более вероятно активировать нейрон *B*, и наоборот.

Правила Хэбба были сформулированы Дональдом Хэббом в 1949 году. В то время он исследовал процессы запоминания в мозге и предложил, что связи между нейронами укрепляются в процессе обучения.

Сегодня правила Хэбба являются основой для многих моделей искусственных нейронных сетей, включая самые простые, такие как однослойные перцептроны, и более сложные, такие как многослойные перцептроны и сверточные нейронные сети.

Персептрон

Библиотека ГГТУ им. П.О. Сухого

Персептрон (англ. *perceptron*) – простейшая модель искусственной нейронной сети прямого распространения. Она была разработана Фрэнком Розенблаттом в 1957 году.

Персептрон состоит из одного или нескольких искусственных нейронов, каждый из которых получает на вход несколько значений (например, координаты точки на плоскости), взвешивает их и применяет к результату функцию активации (обычно это пороговая функция). В зависимости от значения на выходе нейрона, персептрон выдает 1 или -1, что может интерпретироваться как бинарный ответ на заданную задачу.

Персептроны обучаются методом стохастического градиентного спуска. На каждой итерации алгоритм получает на вход некоторый пример, вычисляет выход персептрона и сравнивает его с желаемым выходом. Если выход нейрона неверен, то производится коррекция весов в соответствии с формулой:

$$w(t+1) = w(t) + \eta(y - y')x\ .$$

где w – вектор весов,

t – номер итерации,

η – скорость обучения,

y – выход персептрона на текущем примере,

y' – желаемый выход,

x – вектор входных значений.

Персептроны могут использоваться для решения задач бинарной классификации, т.е. разделения примеров на два класса по определенному признаку. Однако персептроны не могут решать задачи, в которых классы не могут быть разделены гиперплоскостью в пространстве входных признаков. В этом случае может помочь многослойный персептрон или другая модель нейронной сети.

Тренировочный сет

Тренировочный сет или обучающая выборка – это набор данных, на котором обучается нейронная сеть. Обучение нейронной сети заключается в изменении весовых коэффициентов сети таким

Библиотека ГГТУ им.П.О.Сухого

образом, чтобы сеть могла корректно решать поставленную задачу. Для этого используется алгоритм обратного распространения ошибки, в котором сначала происходит прямое распространение сигнала через сеть, затем вычисляются ошибки на выходном слое и распространяются обратно через сеть для корректировки весовых коэффициентов.

Тренировочный сет должен быть достаточно большим и разнообразным, чтобы сеть могла обобщать полученные знания и корректно работать на новых, ранее не встречавшихся данных. Однако, если тренировочный сет слишком большой, это может привести к переобучению сети – сеть начнет максимально точно запоминать все данные из тренировочного сета, но не сможет корректно обрабатывать новые данные. Поэтому необходимо подбирать оптимальный размер и состав тренировочного сета.

В машинном обучении и нейронных сетях, итерация – это один полный проход по всем примерам в тренировочном наборе данных, которые были поделены на мини-пакеты (*mini-batches*). Этот процесс включает в себя передачу каждого примера через сеть, вычисление функции потерь и обновление параметров модели на основе этой функции потерь с использованием выбранного метода оптимизации.

Эпоха – это полный проход по всем примерам в тренировочном наборе данных без разбиения на мини-пакеты. В процессе каждой эпохи модель обучается на каждом примере тренировочного набора данных, вычисляя функцию потерь и обновляя свои параметры. Количество эпох, необходимых для достижения оптимальной производительности модели, зависит от сложности задачи и размера тренировочного набора данных. В общем случае, большое количество эпох может привести к переобучению модели, а малое количество может привести к недообучению.

1.2 Виды нейронных сетей

Нейронные сети с прямым распространением сигнала

Библиотека ГГТУ им. П.О.Сухого

Нейронные сети с прямым распространением сигнала (*feedforward neural networks*) – это класс нейронных сетей, в которых информационный поток распространяется только в одном направлении, от входных нейронов к выходным, без обратных связей. Такая архитектура нейронной сети также называется персептроном.

Первоначально нейронные сети с прямым распространением сигнала были разработаны для решения задач классификации и при обработке естественного языка. Однако, сейчас они широко применяются в различных областях, таких как компьютерное зрение, обработка речи, рекомендательные системы, анализ текстов и многие другие.

Архитектура нейронной сети с прямым распространением состоит из нескольких слоев. Первый слой является входным и принимает на вход данные, которые необходимо обработать. Последующие слои называются скрытыми слоями, так как их выходы не являются наблюдаемыми. Они обрабатывают входные данные и передают их на следующий слой. Последний слой является выходным и выдает результат работы нейронной сети.

Каждый слой состоит из нейронов, которые связаны между собой. Связи между нейронами имеют веса, которые определяют, насколько важен входной сигнал от данного нейрона для работы сети. Входные данные умножаются на веса и передаются на следующий слой. После этого активируется функция активации, которая определяет выходное значение нейрона. Далее выходное значение передается на следующий слой и так далее, пока не достигнут выходной слой.

Обучение нейронной сети с прямым распространением заключается в настройке весов связей между нейронами таким образом, чтобы минимизировать ошибку работы сети на обучающих данных. Это делается с помощью алгоритмов обратного распространения ошибки, которые позволяют вычислить градиент ошибки по весам и обновить их значения.

Рекуррентные нейронные сети

Библиотека ГГТУ им. П.О. Сухого

Рекуррентные нейронные сети (*Recurrent Neural Networks, RNN*) являются разновидностью нейронных сетей, которые способны обрабатывать и анализировать последовательности данных. Они особенно полезны для задач, где входные данные имеют динамический или изменяющийся характер, таких как обработка естественного языка, распознавание речи, прогнозирование временных рядов и многих других.

Основное отличие рекуррентных нейронных сетей от нейронных сетей с прямым распространением сигнала заключается в их способности сохранять информацию о предыдущих входах и использовать ее для обработки текущего входа. Для этого в рекуррентных сетях используются специальные связи между нейронами, называемые рекуррентными связями. Они позволяют передавать информацию о предыдущем состоянии сети на текущий момент времени и использовать ее для принятия решений.

Одним из наиболее популярных типов рекуррентных нейронных сетей является *Long Short-Term Memory* (*LSTM*), которая была предложена в 1997 году Хохрейтером и Шмидхубером. *LSTM* использует специальные блоки памяти, которые могут запоминать информацию на длительное время и контролировать поток данных внутри сети. Благодаря этому *LSTM* может эффективно работать с длинными последовательностями данных и избегать проблемы затухания градиентов, которая возникает в других типах рекуррентных сетей.

В последние годы рекуррентные нейронные сети стали широко применяться в различных областях, связанных с обработкой и анализом последовательностей данных. Они используются для создания генеративных моделей, прогнозирования временных рядов, машинного перевода, распознавания речи и текста, генерации текстов и многих других приложений.

Глубокая нейронная сеть

Глубокая нейронная сеть (ГНС) – это нейронная сеть, имеющая несколько скрытых слоев между входным и выходным слоями. Она представляет собой последовательность преобразований входных

Библиотека ГГТУ им. П.О. Сухого

данных, каждое из которых выполняется на промежуточных слоях сети. ГНС позволяет моделировать сложные зависимости между входными и выходными данными, а также обнаруживать скрытые закономерности в данных.

В отличие от классических нейронных сетей, которые имеют только один скрытый слой, глубокие нейронные сети состоят из нескольких скрытых слоев, каждый из которых обрабатывает данные на все более высоком уровне абстракции. При этом каждый слой может содержать множество нейронов, что позволяет получать многомерное представление данных. Таким образом, глубокая нейронная сеть может обрабатывать более сложные данные, чем классическая нейронная сеть.

Примерами областей, в которых применяются глубокие нейронные сети, являются компьютерное зрение, обработка естественного языка, рекомендательные системы, распознавание речи и другие. Глубокие нейронные сети позволяют решать сложные задачи, которые ранее считались неразрешимыми, и обеспечивают высокую точность работы.

1.3 Способы вычисления ошибок и обучение нейронных сетей

Способы вычисления ошибок нейронных сетей зависят от выбранной функции потерь. Функция потерь (или функция ошибки) определяет, какая ошибка будет минимизироваться во время обучения сети.

Например, для задачи классификации на два класса часто используется бинарная кросс-энтропия (*binary cross-entropy*), которая определяется как:

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

где y_i – истинная метка класса для i-го образца,

$\hat{y}_i$ – предсказанная метка класса.

Библиотека ГГТУ им. П.О. Сухого

Для задачи классификации на несколько классов часто используется категориальная кросс-энтропия (*categorical cross-entropy*), которая определяется как:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j}\log_8(\hat{y}_{i,j})$$

где $y_{i,j}$ – индикаторная переменная, которая равна 1, если *i*-й образец принадлежит к классу *j*, и 0 в противном случае;

$\hat{y}_{i,j}$ – вероятность того, что *i*-й образец принадлежит к классу *j*.

Для задач регрессии часто используется среднеквадратическая ошибка (*mean squared error*), которая определяется как:

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

где y_i – истинное значение для i-го образца;

$\hat{y}_i$ – предсказанное значение.

После вычисления функции потерь необходимо минимизировать ее значение во время обучения сети. Для этого используются алгоритмы оптимизации, такие как стохастический градиентный спуск (*SGD*) или его различные модификации (например, *Adam, RMSprop*).

Нейрон смещения (*bias neuron*) – это нейрон, который всегда имеет значение 1 и не имеет входных связей. Его выходное значение просто передается в следующий слой нейронов, умноженное на соответствующий вес. Таким образом, нейрон смещения служит для управления смещением функции активации нейронов следующего слоя и повышения гибкости модели.

Добавление нейрона смещения в каждый слой является стандартной практикой в нейронных сетях, так как это помогает

Библиотека ГГТУ им.П.О.Сухого

увеличить ее способность к аппроксимации сложных функций и увеличить степень свободы в процессе обучения.

Существует несколько методов обучения нейронных сетей. Рассмотрим наиболее распространенные из них:

1. Метод обратного распространения ошибки (*backpropagation*) – наиболее распространенный метод обучения глубоких нейронных сетей. Он основан на вычислении градиента функции ошибки по весам и последующем их изменении в направлении наискорейшего убывания. В процессе обучения сеть сначала прогнозирует результат, затем сравнивает его с фактическим результатом и вычисляет ошибку. Эта ошибка затем распространяется назад через сеть, с учетом весов каждого нейрона, чтобы скорректировать их значения.

2. Метод оптимизации стохастического градиента (*Stochastic Gradient Descent, SGD*) – в этом методе обучения веса в сети обновляются на каждом примере обучающей выборки. Это позволяет быстрее достичь минимума функции ошибки и ускоряет обучение нейронной сети.

3. Метод адаптивного шага обучения (*Adaptive Learning Rate, Adagrad*) – в этом методе обучения шаг обучения автоматически адаптируется для каждого параметра, в зависимости от частоты его изменения. Это позволяет ускорить сходимость нейронной сети.

4. Метод стохастической дропаут регуляризации (*Stochastic Dropout Regularization*) – в этом методе обучения случайные нейроны отключаются во время обучения. Это помогает предотвратить переобучение и улучшить обобщающую способность нейронной сети.

5. Метод сверточных нейронных сетей (*Convolutional Neural Network, CNN*) – этот метод используется в основном для обработки изображений и видео. Он включает в себя несколько слоев, каждый из которых выполняет определенные операции, такие как свертка и подвыборка. Это позволяет сети автоматически выделять важные признаки изображения.

Метод обратного распространения ошибок (*backpropagation*) – это один из наиболее распространенных методов обучения глубоких нейронных сетей. Он основан на алгоритме градиентного спуска,

Библиотека ГГТУ им. П.О. Сухого

который позволяет оптимизировать веса нейронной сети для минимизации ошибки на обучающей выборке.

Алгоритм градиентного спуска состоит в том, чтобы на каждом шаге обновлять значения весов сети в направлении, противоположном градиенту функции ошибки по этим весам. Это позволяет двигаться в сторону минимума функции ошибки и тем самым уменьшать ошибку на обучающей выборке.

В методе обратного распространения ошибок градиент вычисляется по цепному правилу, которое позволяет распространить ошибку на все слои сети, начиная с выходного и двигаясь к входному. В процессе обучения каждый входной пример проходит через сеть, и для каждого нейрона вычисляются его выходное значение и ошибка. Затем градиент ошибки передается обратно через сеть, и веса каждого нейрона обновляются в соответствии с градиентом и шагом обучения.

Метод обратного распространения ошибок позволяет обучать глубокие нейронные сети с множеством скрытых слоев, что делает его наиболее популярным методом обучения глубоких нейронных сетей. Однако этот метод также имеет некоторые ограничения, такие как проблема затухания или взрыва градиента, когда градиент становится очень маленьким или очень большим в процессе обучения глубоких нейронных сетей. Для решения этих проблем были разработаны различные модификации метода обратного распространения ошибок, такие как градиентный спуск со стохастическим градиентом или методы оптимизации, такие как *Adam* или *RMSprop*.

Библиотека ГГТУ им. П.О. Сухого

2 Обучение простейшей нейронной сети на языке python

2.1 Библиотеки *Theano* и *NumPy*. Нейронная сеть с двумя слоями

Библиотека *Theano* является одной из самых популярных библиотек для глубокого обучения. Она позволяет определять, оптимизировать и вычислять математические выражения, в том числе и нейронные сети. Она предоставляет удобный интерфейс для определения вычислительных графов, которые могут быть оптимизированы на ходу для более быстрого выполнения.

Библиотека *NumPy* предоставляет мощный функционал для работы с многомерными массивами. Она широко используется в машинном обучении и глубоком обучении для обработки данных и реализации алгоритмов. Она предоставляет функции для выполнения различных операций над массивами, включая арифметические операции, матричное умножение, транспонирование, срезы и индексирование.

Theano и *NumPy* могут использоваться вместе для реализации нейронных сетей. *Theano* может использовать *NumPy* массивы для хранения и обработки данных, а также для оптимизации вычислений. *Theano* также предоставляет множество встроенных функций для создания и оптимизации нейронных сетей, которые могут быть использованы вместе с *NumPy*.

Пример создания простой нейронной сети с помощью *Theano* и *NumPy*:

```
import numpy as np
import theano
import theano.tensor as T

# Задаем входные данные
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
```


Библиотека ГГТУ им. П.О. Сухого

```
# Задаем выходные данные
Y = np.array([0, 1, 1, 0])

# Задаем параметры модели
n_hidden = 3
n_input = X.shape[1]
n_output = 1
learning_rate = 0.1
n_epochs = 10000

# Создаем переменные для входных и выходных данных
x = T.matrix('x')
y = T.vector('y')

# Инициализируем веса
w1 = theano.shared(np.random.randn(n_input, n_hidden), name='w1')
w2 = theano.shared(np.random.randn(n_hidden, n_output), name='w2')

# Определяем функцию активации (sigmoid)
sigmoid = lambda x: 1 / (1 + T.exp(-x))

# Вычисляем значения скрытого слоя
hidden = sigmoid(T.dot(x, w1))

# Вычисляем значения выходного слоя
output = sigmoid(T.dot(hidden, w2))

# Определяем функцию ошибки (кросс-энтропия)
cost = T.mean(-y * T.log(output) - (1 - y) * T.log(1 - output))

# Определяем градиенты функции ошибки по весам
dw1, dw2 = T.grad(cost, [w1, w2])
```


Библиотека ГГТУ им.П.О.Сухого

```
# Определяем функцию обновления весов (градиентный спуск)
update_w1 = theano.function(inputs=[x, y], outputs=cost, updates=[(w1, w1 - learning_rate * dw1)])
update_w2 = theano.function(inputs=[x, y], outputs=cost, updates=[(w2, w2 - learning_rate * dw2)])

# Обучаем модель
for i in range(n_epochs):
    cost1 = update_w1(X, Y)
    cost2 = update_w2(X, Y)
    if i % 1000 == 0:
        print('Epoch:', i, 'Cost1:', cost1, 'Cost2:', cost2)
# Проверяем результаты
hidden_layer = sigmoid(np.dot(X, w1.get_value()))
output_layer = sigmoid(np.dot(hidden_layer, w2.get_value()))
predictions = output_layer.round()
print('Predictions:', predictions)
```

Данный код реализует двухслойную нейронную сеть с помощью библиотеки *Theano* для решения задачи исключающего или (*XOR*).

Сначала задаются данные для обучения: четыре входных вектора (0, 0), (0, 1), (1, 0) и (1, 1), а также ожидаемые выходы для каждого из входных векторов (0, 1, 1, 0).

Затем задаются гиперпараметры: количество входных, скрытых и выходных нейронов.

Далее создаются символические переменные для входных данных (*x*) и ожидаемых выходов (*y*), а также переменные для хранения весов и смещений (*W1, b1, W2, b2*).

Затем определяется граф вычислений с помощью функций *Theano* для вычисления скрытых и выходных значений, а также функции потерь (*MSE*).

Библиотека ГГТУ им.П.О.Сухого

Для обновления весов и смещений используется метод градиентного спуска с шагом 0.1, определенный в функции *train_model*.

Для проверки результатов обучения используется функция *test_model*, которая принимает на вход входные данные и выдает предсказанные выходы.

Наконец, модель обучается на 10000 эпохах, и результаты проверяются с помощью функции *test_model*, которая выводит предсказанные значения для всех входных данных.

2.2 Простейшая реализация алгоритма градиентного спуска

Алгоритм градиентного спуска является одним из базовых методов оптимизации в машинном обучении, который используется для обновления весов в нейронных сетях. Реализация алгоритма градиентного спуска в «сигмоиде» может выглядеть следующим образом на языке Python:

```
import numpy as np

# Задаем данные для обучения
X_train = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
Y_train = np.array([[0], [1], [1], [0]], dtype=np.float32)

# Задаем гиперпараметры
n_input = 2
n_hidden = 2
n_output = 1
learning_rate = 0.1

# Инициализируем веса
W1 = np.random.randn(n_input, n_hidden)
b1 = np.zeros(n_hidden)
W2 = np.random.randn(n_hidden, n_output)
```


Библиотека ГГТУ им. П.О. Сухого

```
b2 = np.zeros(n_output)

# Задаем функцию «сигмоиды»
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Обучаем модель
for i in range(10000):
    # Прямое распространение
    hidden = sigmoid(np.dot(X_train, W1) + b1)
    output = sigmoid(np.dot(hidden, W2) + b2)

    # Вычисляем ошибку
    error = Y_train - output

    # Обратное распространение
    delta_output = error * output * (1 - output)
    delta_hidden = np.dot(delta_output, W2.T) * hidden * (1 - hidden)
    # Обновляем веса
    W2 += learning_rate * np.dot(hidden.T, delta_output)
    b2 += learning_rate * np.sum(delta_output, axis=0)
    W1 += learning_rate * np.dot(X_train.T, delta_hidden)
    b1 += learning_rate * np.sum(delta_hidden, axis=0)

# Проверяем результаты
print(sigmoid(np.dot(sigmoid(np.dot(X_train, W1) + b1), W2) + b2))
```

В данной реализации мы инициализируем веса случайными значениями и задаем гиперпараметры, такие как количество входных, скрытых и выходных узлов, а также скорость обучения. Затем мы применяем функцию «сигмоиды» для прямого распространения, вычисляем ошибку и применяем алгоритм обратного

Библиотека ГГТУ им. П.О.Сухого

распространения для обновления весов. В конце мы проверяем результаты, применяя прямое распространение с обновленными весами на обучающих данных.

Нейронные сети используются во многих задачах машинного обучения и искусственного интеллекта. Некоторые примеры включают:

1. *Классификация*: нейронные сети могут использоваться для классификации объектов на различные классы. Например, классификация изображений на категории, такие как кошки, собаки, автомобили и т.д.

2. *Принятие решений*: нейронные сети могут использоваться для принятия решений на основе входных данных. Например, нейронные сети могут использоваться для оценки рисков и принятия решений в финансовой индустрии.

3. *Распознавание образов*: нейронные сети могут использоваться для распознавания образов на изображениях, звуках или других типах данных. Например, распознавание рукописного текста или распознавание лиц.

4. *Прогнозирование*: нейронные сети могут использоваться для прогнозирования будущих событий на основе исторических данных. Например, прогнозирование цен на акции или прогнозирование погоды.

Это лишь некоторые примеры использования нейронных сетей. Нейронные сети могут применяться во многих других областях, таких как автоматическое управление, робототехника, медицинские приложения и многое другое.

Библиотека ГГТУ им. П.О. Сухого

3 Глубокое обучение нейронной сети. Классификации изображений

3.1 Глубокое обучение нейронной сети на основе *MNIST*

Keras

Keras – это высокоуровневый фреймворк для создания нейронных сетей, который позволяет легко и быстро создавать модели, обучать их и использовать для решения задач машинного обучения. Вот некоторые основные настройки, которые можно задать при работе с *Keras*:

1. *Архитектура модели*: *Keras* позволяет выбирать из большого количества слоев и комбинировать их в различные конфигурации для создания архитектуры модели.

2. *Функция потерь* (*loss function*): это функция, которую минимизирует модель в процессе обучения. Выбор подходящей функции потерь зависит от задачи машинного обучения, например, для задач классификации можно использовать категориальную кросс-энтропию.

3. *Оптимизатор* (*optimizer*): оптимизатор отвечает за обновление весов модели на каждом шаге обучения с целью минимизации функции потерь. *Keras* поддерживает множество оптимизаторов, например, стохастический градиентный спуск (*SGD*) и *Adam*.

4. *Метрики* (*metrics*): метрики используются для оценки качества модели в процессе обучения и тестирования. Например, для задач классификации можно использовать метрику *accuracy*.

5. *Регуляризация* (*regularization*): это техника, которая помогает бороться с переобучением модели. В *Keras* можно использовать *L1* или *L2* регуляризацию для ограничения весов модели.

6. *Разбиение данных* (*data splitting*): для обучения модели необходимо разделить данные на обучающую, валидационную и

Библиотека ГГТУ им. П.О. Сухого

тестовую выборки. В *Keras* можно использовать функцию *train_test_split* из библиотеки *sklearn* для разбиения данных.

7. *Размер батча* (*batch size*): это количество образцов данных, которое передается модели за одну итерацию обучения. Выбор подходящего размера батча зависит от объема данных и вычислительных ресурсов.

8. *Количество эпох* (*number of epochs*): это количество итераций обучения на всем наборе данных. Выбор оптимального количества эпох также зависит от объема данных и вычислительных ресурсов, а также от того, когда модель начинает переобучаться.

Классический набор рукописных цифр *MNIST* содержит изображения 70,000 рукописных цифр размером 28x28 пикселей, разделенных на 60,000 обучающих и 10,000 тестовых изображений. Каждое изображение представлено в оттенках серого (*grayscale*) и имеет метку класса, соответствующую изображенной на нем цифре от 0 до 9. *MNIST* является одним из самых известных и широко используемых датасетов в области компьютерного зрения и машинного обучения, и часто используется для обучения и тестирования моделей нейронных сетей, а также в качестве базового датасета для сравнения различных алгоритмов классификации изображений.

Препроцессинг данных – это подготовка данных к использованию в модели машинного обучения. Включает в себя обработку, очистку, масштабирование, нормализацию и другие преобразования данных, чтобы сделать их подходящими для обучения модели.

В задачах распознавания рукописных цифр, таких как *MNIST*, препроцессинг данных может включать в себя следующие шаги:

1. *Загрузка данных*: данные могут быть предоставлены в различных форматах, например, *CSV*-файл, база данных или набор файлов изображений.

2. *Разделение данных на обучающую и тестовую выборки*: данные обычно разбиваются на две выборки: обучающую и тестовую, чтобы оценить производительность модели на новых данных.

Библиотека ГГТУ им. П.О.Сухого

3. *Преобразование данных в нужный формат*: в случае рукописных цифр *MNIST* данные могут быть представлены в виде изображений, каждое из которых имеет размерность 28x28 пикселей. Для использования их в нейронной сети, изображения нужно преобразовать в формат, который может быть воспринят моделью, например, массив *NumPy*.

4. *Нормализация данных*: данные могут быть нормализованы для улучшения производительности модели, например, путем масштабирования значений пикселей в диапазон от 0 до 1.

5. *Разработка пайплайна препроцессинга*: разработка пайплайна препроцессинга позволяет автоматизировать обработку данных для более эффективного обучения модели.

В *Keras* для преобразования данных изображений в формат, который может быть использован в нейронной сети, можно использовать готовую функцию *ImageDataGenerator*, которая автоматически преобразует изображения в массивы *NumPy* и масштабирует их. Для нормализации данных можно использовать функцию *normalize* из модуля *numpy*.

MNIST является классическим набором данных для задач распознавания изображений. Он состоит из 60 000 изображений рукописных цифр для обучения и 10 000 изображений для тестирования. Каждое изображение имеет размер 28x28 пикселей и представлено в виде черно-белой матрицы.

Для обучения нейронной сети на основе *MNIST* необходимо выполнить следующие шаги:

– *загрузить данные*: загрузить обучающий и тестовый наборы данных *MNIST*. В *Keras* это можно сделать с помощью функции *load_data()* из модуля *keras.datasets*;

– *предобработка данных*: привести данные к нужному формату, например, преобразовать каждый пиксель в диапазон от 0 до 1;

– *определить архитектуру модели*: определить количество скрытых слоев, количество нейронов в каждом слое и функции активации для каждого слоя;

Библиотека ГГТУ им. П.О. Сухого

– *настроить параметры обучения*: выбрать метод оптимизации, функцию потерь и метрики для оценки качества модели;

– *обучить модель*: обучить модель на обучающем наборе данных с использованием метода *fit()*;

– *оценить качество модели*: оценить качество модели на тестовом наборе данных с использованием метода *evaluate()*;

Пример глубокого обучения нейронной сети на основе *MNIST* в *Keras*:

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

# Загружаем данные MNIST
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Предобрабатываем данные
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# Преобразуем метки классов в бинарные матрицы
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Определяем структуру модели
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
```


Библиотека ГГТУ им.П.О.Сухого

```
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
# Настраиваем параметры обучения
model.compile(loss='categorical_crossentropy',
        optimizer=RMSprop(),
        metrics=['accuracy'])

# Обучаем модель
batch_size = 128
epochs = 20
history = model.fit(x_train, y_train,
            batch_size=batch_size,
            epochs=epochs,
            verbose=1,
            validation_data=(x_test, y_test))

# Оцениваем точность работы модели на тестовых данных
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

В этом примере мы загружаем набор данных *MNIST* и предобрабатываем данные, чтобы они были готовы для использования в модели. Затем мы определяем структуру модели с помощью класса *Sequential* из библиотеки *Keras*, который позволяет добавлять слои друг за другом. Наша модель состоит из трех слоев: два слоя *Dense* с функцией активации *relu* и функцией отсева *Dropout*, а также слой *Dense* с функцией активации *softmax*, который выдает вероятности принадлежности каждому из десяти классов цифр.

Затем мы настраиваем параметры обучения модели, включая функцию потерь (*loss*), оптимизатор (*optimizer*) и метрику (*metrics*) для оценки производительности модели. После этого мы обучаем модель на тренировочных данных, используя метод *fit*.

Библиотека ГГТУ им.П.О.Сухого

Наконец, мы оцениваем точность работы модели на тестовых данных и выводим результаты.

Для анализа качества обучения нейронной сети на основе *MNIST* можно использовать различные метрики, такие как *accuracy, precision, recall, F1-score* и другие.

Accuracy (точность) – это метрика, которая измеряет долю правильных предсказаний, т.е. отношение количества правильных предсказаний к общему количеству предсказаний. В случае классификации *MNIST accuracy* рассчитывается как отношение числа правильно предсказанных изображений к общему числу изображений в тестовой выборке.

Precision (точность) – это метрика, которая показывает, как часто классификатор предсказывает положительный класс правильно. *Precision* рассчитывается как отношение числа верно классифицированных объектов положительного класса к общему числу объектов, которые классификатор отнес к положительному классу.

Recall (полнота) – это метрика, которая показывает, как много объектов положительного класса было правильно предсказано классификатором. *Recall* рассчитывается как отношение числа верно классифицированных объектов положительного класса к общему числу объектов положительного класса в тестовой выборке.

F1-score – это гармоническое среднее между *precision* и *recall*. *F1-score* рассчитывается как 2*(precision*recall)/(precision+recall)*.

Для оценки качества обучения нейронной сети на основе *MNIST* можно использовать функцию *evaluate*, которая рассчитывает значения метрик для заданной модели и тестовых данных. Например, для оценки *accuracy* модели можно использовать следующий код:

```
score = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (score[1]*100))
```


Библиотека ГГТУ им. П.О.Сухого

Здесь *X_test* и *y_test* – это тестовые данные, а *score[1]* содержит значение *accuracy*. Аналогично можно рассчитать и другие метрики, например:

```
score = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (score[1]*100))
print("Precision: %.2f%%" % (score[2]*100))
print("Recall: %.2f%%" % (score[3]*100))
print("F1-score: %.2f%%" % (score[4]*100))
```

Здесь *score[2]*, *score[3]* и *score[4]* содержат значения *precision*, *recall* и *F1-score* соответственно.

Определение гиперпараметров – это важная часть процесса обучения нейронной сети, которая влияет на ее качество и скорость обучения. Некоторые из наиболее важных гиперпараметров включают количество слоев сети, количество эпох обучения, размер мини-выборки и скорость обучения.

Количество слоев сети

Количество слоев нейронной сети зависит от сложности задачи, которую необходимо решить. Если задача является простой, то достаточно одного или двух скрытых слоев, если же задача более сложная, может потребоваться больше слоев. Однако увеличение количества слоев может привести к переобучению сети, поэтому нужно подбирать оптимальное количество.

Количество эпох

Количество эпох обучения также зависит от сложности задачи и размера данных. Если данные большие, то требуется больше эпох для обучения сети. Однако слишком большое количество эпох может также привести к переобучению.

Размер мини-выборки

Размер мини-выборки определяет количество примеров, которые обрабатываются сетью за одну итерацию обучения. Большие значения могут ускорить обучение, но также могут привести к

Библиотека ГГТУ им.П.О.Сухого

ухудшению качества. Размер мини-выборки обычно выбирается из диапазона от 16 до 256 примеров.

Скорость обучения

Скорость обучения определяет, насколько сильно корректируются веса и смещения в процессе обучения. Оптимальная скорость обучения зависит от конкретной задачи и может быть подобрана путем проб и ошибок.

Для определения оптимальных гиперпараметров можно использовать методы оптимизации, такие как *Grid Search* или *Random Search*. *Grid Search* позволяет перебрать все возможные значения гиперпараметров в заданных диапазонах, а *Random Search* случайным образом выбирает значения гиперпараметров в заданных диапазонах.

Наборы данных для обучения являются одним из наиболее важных компонентов в задачах машинного обучения. Обычно данные разбивают на три группы: обучающую выборку, проверочную выборку и тестовую выборку.

1. *Обучающая выборка*: это набор данных, на котором модель будет обучаться. Этот набор данных содержит примеры, которые модель будет использовать для определения внутренних параметров. Обычно обучающая выборка составляет от 60% до 80% от общего набора данных.

2. *Проверочная выборка*: это набор данных, который используется для оптимизации модели в процессе обучения. Проверочный набор данных используется для настройки гиперпараметров модели, таких как количество эпох обучения, размер мини-батча, скорость обучения и т.д. Проверочная выборка составляет обычно около 10-20% от общего набора данных.

3. *Тестовая выборка*: это набор данных, который используется для оценки качества работы модели после ее обучения. Этот набор данных должен быть независимым от обучающей и проверочной выборки. Тестовая выборка позволяет оценить, насколько хорошо модель работает на новых данных, которые она ранее не видела. Обычно тестовая выборка составляет 20-30% от общего набора данных.

Библиотека ГГТУ им. П.О. Сухого

Разбиение данных на три выборки обеспечивает оценку качества модели на нескольких этапах ее развития и обеспечивает возможность выявления переобучения. Переобучение возникает, когда модель слишком точно подстроилась под обучающий набор данных и не может обобщать на новые данные. Выявление переобучения можно произвести, сравнивая качество модели на проверочной и тестовой выборках. Если качество модели на проверочной выборке высокое, а на тестовой – низкое, то это является признаком переобучения.

Проблема переобучения (*overfitting*) возникает, когда модель слишком хорошо подстраивается под обучающую выборку и начинает «запоминать» ее вместо обобщения закономерностей в данных. В результате модель начинает показывать плохие результаты на новых данных, не видимых ей в процессе обучения. Это может произойти, например, когда модель имеет слишком большую сложность или когда количество обучающих данных слишком мало.

Для борьбы с переобучением можно использовать различные методы. Некоторые из них:

– регуляризация – добавление дополнительных слагаемых в функцию потерь, чтобы ограничить значения весов и снизить сложность модели;

– *Dropout* – случайное выключение некоторых нейронов в процессе обучения, чтобы сделать модель более устойчивой и предотвратить запоминание шумовых зависимостей;

– увеличение размера обучающей выборки – более обширная выборка поможет модели уловить более широкий диапазон закономерностей в данных и снизить вероятность переобучения;

– *Early stopping* – остановка обучения, когда производительность модели на проверочных данных перестает улучшаться, чтобы избежать переобучения;

– использование ансамблей моделей – объединение нескольких моделей для улучшения качества предсказаний и снижения вероятности переобучения.

Библиотека ГГТУ им. П.О. Сухого

3.2 Использование обученной нейронной сети для классификации изображений

Сохранение обученной нейронной сети в *Keras* может быть достигнуто несколькими способами. Один из самых простых способов – использовать метод *save()* модели. Этот метод сохраняет все параметры модели в единый файл, включая веса нейронной сети, оптимизатор и конфигурацию модели.

Пример сохранения модели в *Keras*:

```
from keras.models import Sequential
from keras.layers import Dense

# Создаем нейронную сеть
model = Sequential()
model.add(Dense(10, input_dim=8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Компилируем модель
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Обучаем модель
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))

# Сохраняем модель в файл
model.save('my_model.h5')
```

В этом примере мы создали простую нейронную сеть с двумя слоями *Dense* и обучили ее на данных, разделенных на обучающую и проверочную выборки. После этого мы вызвали метод *save()* модели, передав имя файла для сохранения.

Библиотека ГГТУ им.П.О.Сухого

Теперь мы можем загрузить обученную модель с помощью метода *load_model()* из модуля *keras.models*.

```
from keras.models import load_model

# Загружаем модель из файла
loaded_model = load_model('my_model.h5')

# Используем модель для предсказания
y_pred = loaded_model.predict(X_test)
```

Таким образом, сохранение обученной нейронной сети в *Keras* позволяет легко сохранять модели после обучения и загружать их позже для использования в других приложениях.

Формат *JSON* для сохранения архитектуры нейронной сети в *Keras* представляет собой сериализованный словарь, содержащий информацию об архитектуре сети. Ключи и значения этого словаря включают в себя следующие элементы:

– *class_name*: строковое значение, указывающее на класс соответствующей модели (например, *Sequential* или *Model*);

– *config*: словарь, содержащий конфигурационные параметры модели, такие как количество слоев, типы слоев и их параметры (например, *units, activation, kernel_initializer* и т.д.), а также другие параметры, такие как функции активации и функции потерь;

– *weights*: список массивов весов и смещений, содержащий значения весовых коэффициентов и смещений всех слоев модели.

Формат *HDF5* (*Hierarchical Data Format version 5*) используется в *Keras* для сохранения весов нейронной сети после обучения. Файлы в формате *HDF5* могут хранить большие объемы данных и позволяют эффективно обрабатывать многомерные массивы.

В *Keras* веса модели можно сохранить в формате *HDF5* с помощью метода *save_weights* класса *Model*. Пример:

```
from keras.models import Model
```


Библиотека ГГТУ им.П.О.Сухого

```
model = ... # ваша модель
model.save_weights('model_weights.h5')
```

После сохранения весов модели, их можно загрузить с помощью метода *load_weights*:

```
from keras.models import Model
model = ... # ваша модель
model.load_weights('model_weights.h5')
```

Также, при использовании метода save класса *Model*, сохраняется как архитектура модели, так и веса в формате *HDF5*:

```
from keras.models import Model
model = ... # ваша модель
model.save('model.h5')
```

Загрузить модель целиком можно с помощью метода *load_model*:

```
from keras.models import load_model
model = load_model('model.h5')
```

Для загрузки и компиляции сохраненной нейронной сети из файлов мы можем использовать модуль *models* библиотеки *Keras*.

Пример загрузки модели из файла *model_architecture.json* и весов из файла *model_weights.h5* и их компиляции:

```
from keras.models import model_from_json
from keras.optimizers import SGD

# Загрузка модели из файла JSON
with open('model_architecture.json', 'r') as f:
    model = model_from_json(f.read())
```


Библиотека ГГТУ им.П.О.Сухого

```
# Загрузка весов из файла HDF5
model.load_weights('model_weights.h5')

# Компиляция модели с оптимизатором SGD
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
```

В этом примере мы загружаем модель из файла *model_architecture.json*, который содержит архитектуру нашей нейронной сети в формате *JSON*. Затем мы загружаем веса модели из файла *model_weights.h5*, который содержит сохраненные веса модели в формате *HDF5*.

Далее мы компилируем модель с помощью оптимизатора *SGD* и указываем функцию потерь (*categorical_crossentropy*), оптимизатор (*sgd*) и метрики (*accuracy*).

После компиляции мы можем использовать модель для прогнозирования на новых данных.

Для использования обученной нейронной сети для классификации своих изображений необходимо выполнить следующие шаги:

1. Подготовить изображения в соответствии с форматом, на котором была обучена нейронная сеть. Например, если нейронная сеть была обучена на изображениях размером 28x28 пикселей в оттенках серого, то и входные изображения должны быть в таком же формате.
2. Загрузить сохраненную модель нейронной сети из файлов, содержащих информацию об архитектуре сети и весах модели.
3. Скомпилировать модель с помощью функции *compile()* и передать в нее параметры, необходимые для выполнения классификации (например, оптимизатор и функцию потерь).

Библиотека ГГТУ им. П.О. Сухого

4. Подать на вход обученной нейронной сети новые изображения и получить предсказания классов для этих изображений с помощью функции *predict()*.

Пример кода для использования обученной нейронной сети для классификации своих изображений:

```
from keras.models import model_from_json
import numpy as np
from PIL import Image

# Загрузка сохраненной модели из файлов
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
loaded_model.load_weights("model.h5")
print("Загрузка модели из файлов завершена")

# Компиляция модели
loaded_model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])

# Подготовка входных данных
img = Image.open('my_image.png')
img = img.convert('L') # преобразование в оттенки серого
img = img.resize((28, 28)) # изменение размера до 28x28 пикселей
x = np.asarray(img, dtype=np.float32)
x = x.reshape(1, 28, 28, 1)
x /= 255

# Классификация изображения
prediction = loaded_model.predict(x)
class_index = np.argmax(prediction)
```


Библиотека ГГТУ им.П.О.Сухого

print("Предсказанный класс:", class_index)

В этом примере мы загружаем сохраненную модель нейронной сети из файлов *model.json* и *model.h5*, компилируем ее, подаем на вход новое изображение, преобразуем его в формат, необходимый для входных данных модели, и получаем предсказание класса с помощью функции *predict()*.

В задаче классификации изображений каждое изображение может относиться к одному из нескольких классов. Для того чтобы представить каждый класс в виде числа, можно использовать категориальное представление (*one-hot encoding*).

Категориальное представление заключается в том, что каждый класс представляется в виде вектора, состоящего из нулей и одной единицы. Например, если в задаче классификации изображений есть 10 классов, то каждый класс может быть представлен в виде вектора длины 10, где на *i*-й позиции стоит единица, если это *i*-й класс, и ноль в остальных позициях. Например, класс 3 может быть представлен в виде вектора [0, 0, 1, 0, 0, 0, 0, 0, 0, 0].

Метка класса – это просто число, которое указывает, к какому классу относится изображение. Например, если в задаче классификации изображений есть 10 классов, то метка класса может быть числом от 0 до 9, где 0 соответствует первому классу, 1 – второму классу и т.д.

В *Keras* для работы с категориальным представлением и метками классов используются соответственно функции *to_categorical()* и *argmax()*. Функция *to_categorical(y, num_classes=None)* принимает на вход метки классов y и возвращает их категориальное представление. Аргумент *num_classes* определяет количество классов. Функция *argmax(a, axis=None)* принимает на вход массив *a* и возвращает индекс элемента с максимальным значением вдоль заданной оси *axis*. В задаче классификации изображений, ось *axis* обычно равна 1, так как необходимо выбрать класс с максимальной вероятностью для каждого изображения.

Библиотека ГГТУ им. П.О. Сухого

4 Нейросети для анализа текстов

4.1 Представление текста в цифровом виде

Для того чтобы обработать текст нейронной сетью, необходимо представить его в цифровом виде. Существует несколько подходов для представления текста в виде чисел:

1. Мешок слов (*bag of words*) – каждое слово из текста представляется отдельным числом, а сам текст – вектором, в котором на каждой позиции стоит количество вхождений соответствующего слова в тексте.

2. *N*-граммы (*n-grams*) – это последовательности из *N* подряд идущих слов. Каждая *N*-грамма представляется отдельным числом, а текст – вектором, в котором на каждой позиции стоит количество вхождений соответствующей *N*-граммы в тексте.

3. *TF-IDF* (*term frequency – inverse document frequency*) – этот метод также основан на мешке слов, но в отличие от него учитывает частотность слова во всех документах. Таким образом, слова, которые встречаются во многих документах, получают меньший вес, чем слова, которые встречаются редко.

4. Эмбеддинги слов (*word embeddings*) – это метод, который позволяет представить слова в виде векторов в некотором *n*-мерном пространстве, в котором семантически близкие слова расположены близко друг к другу. Векторы эмбеддингов могут быть получены с помощью нейронных сетей, таких как *Word2Vec* или *GloVe*.

5. Сверточные нейронные сети (*CNN*) – этот метод используется для обработки изображений, но также может быть применен для обработки текста. В сверточной нейронной сети каждое слово представляется вектором, который затем проходит через сверточный слой, после чего используется для классификации текста.

Выбор метода для представления текста зависит от задачи, которую необходимо решить, и от особенностей данных. Например, для задачи классификации текстов по темам может быть эффективен

Библиотека ГГТУ им. П.О. Сухого

метод *TF-IDF*, а для задачи определения тональности текста – метод эмбеддингов слов.

One-hot encoding – это процесс преобразования категориальных данных в числовой формат, который может быть использован для обучения моделей машинного обучения. В контексте обработки текста, *one-hot encoding* используется для преобразования слов или символов в векторы с нулями и единицами.

Для примера, рассмотрим следующий текст: "*The quick brown fox jumped over the lazy dog*". Первый шаг заключается в создании словаря уникальных слов, который может быть использован для преобразования каждого слова в соответствующий вектор. Для данного текста, словарь может выглядеть так:

```
{
   "the": 0,
   "quick": 1,
   "brown": 2,
   "fox": 3,
   "jumped": 4,
   "over": 5,
   "lazy": 6,
   "dog": 7
}
```

Затем, каждое слово может быть преобразовано в вектор с помощью *one-hot encoding*. Векторы имеют размерность, равную количеству уникальных слов в словаре. Если слово есть в тексте, то соответствующий элемент вектора равен 1, в противном случае - 0. Например, вектор для слова "*fox*" будет выглядеть следующим образом:

[0, 0, 0, 1, 0, 0, 0, 0]

Библиотека ГГТУ им. П.О. Сухого

Таким образом, *one-hot encoding* позволяет представить каждое слово в тексте в виде вектора фиксированной длины, что может быть использовано для обучения моделей нейронных сетей.

Представление текста плотными векторами (*embeddings*) является более компактным и информативным способом представления текста в цифровом виде для обработки нейронной сетью. В этом подходе каждое слово представляется вектором фиксированной длины, называемым эмбеддингом, который хранит семантическую информацию о слове.

При использовании предобученных эмбеддингов (например, *GloVe, FastText, Word2Vec*), можно использовать готовые векторы слов, которые уже были обучены на больших корпусах текста. Такие эмбеддинги могут помочь улучшить качество модели, особенно если для обучения нейронной сети доступно небольшое количество данных.

Например, в *Keras* можно использовать слой *Embedding*, который позволяет преобразовать последовательность индексов слов в последовательность эмбеддингов. В качестве входных данных слой *Embedding* принимает последовательность целых чисел, соответствующих индексам слов в словаре, и преобразует их в плотные векторы. Каждый плотный вектор имеет размерность, определяемую параметром *output_dim* слоя *Embedding*, и каждому индексу соответствует свой уникальный вектор.

Также, важным параметром слоя *Embedding* является *input_length*, который определяет длину входной последовательности. Если последовательности различной длины, то ее необходимо привести к одной длине, например, путем добавления паддингов (заполнение последовательности до максимальной длины) или обрезания.

Пример использования слоя *Embedding* в *Keras*:

```
from keras.layers import Embedding

vocab_size = 10000 # размер словаря
```


Библиотека ГГТУ им. П.О. Сухого

```
max_len = 100 # максимальная длина последовательности
embedding_dim = 50 # размерность эмбеддинга

model = Sequential()
model.add(Embedding(input_dim=vocab_size,
output_dim=embedding_dim, input_length=max_len))
```

Это создаст модель, которая принимает на вход последовательность индексов слов в словаре размера *vocab_size* и длины *max_len*, и возвращает последовательность плотных векторов размерности *embedding_dim*.

4.2 Нейросети для анализа текстов

Рекуррентные нейронные сети (*RNN*) являются классом нейронных сетей, которые используются для анализа последовательностей данных, таких как текст, речь, временные ряды и другие. Основная идея *RNN* заключается в том, чтобы передавать информацию из предыдущих шагов последовательности в следующие, создавая возможность учитывать контекст при обработке данных.

Одной из особенностей *RNN* является наличие обратной связи, которая позволяет информации передаваться между шагами и, таким образом, создавать связь между данными. Благодаря этому *RNN* позволяют обрабатывать данные переменной длины и учитывать зависимости между элементами последовательности.

Одним из наиболее распространенных типов *RNN* является *Long Short-Term Memory (LSTM)*, который был разработан для решения проблемы затухания градиента, которая может возникать при обучении *RNN* на длинных последовательностях. *LSTM* позволяет сохранять информацию в памяти и извлекать ее при необходимости, что улучшает производительность сети.

Еще одним распространенным типом *RNN* является *Gated Recurrent Unit (GRU)*, который является упрощенной версией LSTM и

Библиотека ГГТУ им. П.О. Сухого

имеет меньше параметров, что позволяет ускорить обучение и улучшить производительность в задачах средней сложности.

RNN и их модификации, такие как *LSTM* и *GRU*, широко используются в задачах, связанных с обработкой текстов и речи, в частности в задачах машинного перевода, генерации текста и распознавания речи.

Анализ тональности текста (*Sentiment Analysis*) – это задача классификации, где тексты необходимо отнести к положительному или отрицательному классу в зависимости от их эмоциональной окраски. Рекуррентные нейронные сети (*RNN*) являются мощным инструментом для анализа тональности текста, поскольку они могут учитывать контекст и последовательность слов в предложении.

Для обучения рекуррентной нейронной сети для анализа тональности текста нам необходимо:

1. Получить набор данных, состоящий из положительных и отрицательных текстов.
2. Предобработать данные, преобразовав текст в формат, который может быть использован в нейронной сети. Мы можем использовать *one hot encoding* или плотные векторы (*embeddings*).
3. Разделить данные на обучающую и тестовую выборки.
4. Определить структуру рекуррентной нейронной сети. Обычно используются *LSTM (Long Short-Term Memory)* или *GRU (Gated Recurrent Unit)* слои.
5. Настроить гиперпараметры модели, такие как количество слоев, количество нейронов, длину последовательности и скорость обучения.
6. Обучить модель на обучающей выборке и оценить ее качество на тестовой выборке.
7. Протестировать модель на новых текстах и оценить ее точность.

LSTM (Long Short-Term Memory) и *GRU (Gated Recurrent Unit)* – это два типа рекуррентных нейронных сетей, которые были разработаны для решения проблемы затухания градиентов в обычных рекуррентных сетях.

Библиотека ГГТУ им. П.О.Сухого

В обычной рекуррентной сети информация передается от одного шага времени к другому через скрытое состояние. Когда сеть обучается на долгосрочной зависимости, происходит затухание градиентов: градиенты становятся слишком маленькими или слишком большими, и в результате сеть перестает обучаться.

LSTM и *GRU* решают эту проблему, используя специальные структуры внутри каждой ячейки, которые позволяют им хранить информацию в течение более длительного времени. *LSTM* использует три внутренних структуры: «ворота забывания», «входной ворот» и «выходной ворот». *GRU* использует две внутренние структуры: «восстановление» и «обновление». Оба типа сетей также используют нелинейную функцию активации для моделирования нелинейных зависимостей в данных.

В общем, *LSTM* и *GRU* лучше подходят для обработки последовательных данных, таких как тексты, временные ряды и аудио-сигналы, поскольку они могут учитывать контекст в течение длительного времени.

Рассмотрим как можно использовать сеть *LSTM* для анализа тональности отзывов *IMDB*.

Для начала нужно загрузить данные и выполнить их предварительную обработку. Мы будем использовать набор данных *IMDB*, который уже включен в *Keras*. Этот набор данных содержит 50000 обучающих и 50 000 тестовых отзывов, разбитых на положительные и отрицательные, так что задача будет состоять в классификации двух классов. Данные уже токенизированы и предварительно обработаны.

Вот пример кода для обучения сети *LSTM* на наборе данных *IMDB*:

```
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding
from keras.optimizers import Adam
```


Библиотека ГГТУ им.П.О.Сухого

```
# Загружаем данные и задаем гиперпараметры
max_features = 10000  # максимальное количество слов для словаря
maxlen = 500  # максимальная длина отзыва в словах
batch_size = 32  # размер мини-пакета
embedding_size = 32  # размерность пространства слов

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Ограничиваем длину отзывов и заполняем недостающие значения
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

# Создаем модель LSTM
model = Sequential()
model.add(Embedding(max_features, embedding_size, input_length=maxlen))
model.add(LSTM(64, return_sequences=True))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0001), metrics=['accuracy'])

# Обучаем модель
model.fit(x_train, y_train, batch_size=batch_size, epochs=10, validation_data=(x_test, y_test))

# Оцениваем качество модели
scores = model.evaluate(x_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```


Библиотека ГГТУ им.П.О.Сухого

В этом примере мы использовали набор данных *IMDB* для определения тональности отзывов. Мы загрузили данные и ограничили длину отзывов, чтобы привести их к одному размеру. Затем мы создали модель *LSTM*, которая состоит из слоя *Embedding* для представления слов в плотном векторном пространстве, двух слоев *LSTM* для обработки последовательности слов и одного полносвязного слоя с функцией активации *sigmoid* для определения тональности. Мы обучили модель на тренировочном наборе данных и оценили ее качество на тестовом наборе данных.

Одномерные сверточные нейросети

Одномерные сверточные нейросети (*1D CNN*) – это нейросети, которые используют свертку для обработки одномерных последовательностей, например, временных рядов, текстовых последовательностей и сигналов.

Одномерные сверточные нейросети в основном используются для задач классификации, распознавания образов и анализа временных рядов. Они работают на основе свертки, максимального пулинга и слоев активации, таких как *ReLU*, для изучения локальных шаблонов входных данных.

1D CNN также могут использоваться в сочетании с рекуррентными слоями, чтобы создать более мощные модели для обработки последовательностей данных.

Примером использования *1D CNN* является задача классификации текста, когда тексты представлены в виде последовательности слов. В этом случае *1D CNN* может изучать локальные шаблоны в представлении последовательности слов и использовать их для классификации текста.

Еще одним примером использования *1D CNN* является задача анализа временных рядов. В этом случае *1D CNN* может изучать локальные шаблоны во временных рядах и использовать их для прогнозирования будущих значений временных рядов.

В целом, одномерные сверточные нейросети являются мощным инструментом для обработки различных типов данных и позволяют

Библиотека ГГТУ им. П.О. Сухого

создавать более точные модели для различных задач машинного обучения.

Классификация текстов – это задача определения категории, к которой относится заданный текст. Это может быть положительный или отрицательный отзыв, категория новостей или класс товаров.

Одним из методов классификации текстов является использование нейронных сетей. Одним из типов нейросетей, которые могут быть использованы для классификации текстов, являются одномерные сверточные нейронные сети.

Одномерные сверточные нейросети – это тип нейронных сетей, которые используют сверточные слои для обработки входных данных. Они могут быть использованы для анализа текстовой информации, так как они могут обнаруживать локальные структуры в последовательностях, таких как предложения или отдельные слова.

Примером задачи классификации текстов с использованием одномерных сверточных нейронных сетей является классификация отзывов на фильмы по их тональности. В этой задаче, каждый отзыв представляется в виде последовательности слов, которые подаются на вход сети. Сеть состоит из нескольких слоев, включая сверточные, max-pooling и полносвязные слои.

Для классификации текстов с помощью одномерных сверточных нейронных сетей, тексты сначала преобразуются в числовые векторы. Это может быть достигнуто путем применения техник, таких как one-*hot encoding* или *word embeddings*. Затем эти числовые векторы используются как входные данные для одномерной сверточной нейронной сети.

При обучении сети, веса настраиваются таким образом, чтобы минимизировать функцию потерь между предсказаниями сети и правильными метками классов. После обучения, сеть может быть использована для классификации новых текстовых данных.

Многозначная классификация

Многозначная классификация текстов представляет собой задачу, в которой необходимо классифицировать тексты по нескольким категориям. Например, при анализе тональности отзывов

Библиотека ГГТУ им. П.О. Сухого

на товары, можно выделить несколько категорий, таких как «положительный», «отрицательный», «нейтральный».

Для решения этой задачи можно использовать многоклассовую классификацию, где каждый текст относится только к одной категории. Однако, в таком подходе теряется информация о том, что текст может одновременно относиться к нескольким категориям.

В этом случае можно использовать многозначную классификацию, где каждый текст может относиться к нескольким категориям. Например, если текст содержит отзыв на несколько товаров, то он может быть классифицирован как «положительный» для одного товара и «отрицательный» для другого.

Для решения задачи многозначной классификации можно использовать различные архитектуры нейронных сетей, включая рекуррентные и сверточные нейронные сети. В зависимости от задачи и характера данных, можно выбрать подходящую архитектуру и соответствующие методы обучения.

Для обучения многозначной классификации текстов, необходимо иметь размеченный набор данных с метками для каждой категории. Кроме того, необходимо определить метрики качества для оценки результатов классификации, такие как точность, полнота и *F*-мера.

Библиотека ПГУ им. Ю. Янки

Лабораторные работы

Лабораторная работа №1

Создание простой нейронной сети

Цель работы: научиться создавать простые нейронные сети, освоить основные принципы работы нейронной сети.

Задание.

1. Определить предметную область для нейронной сети.
2. Создать схему нейронной сети. В нейронной сети должно быть:
   - на входном слое не менее *5* нейронов;
   - на скрытом слое не менее *2* нейронов;
   - на выходном слое *1* нейрон.
3. Исходя из задачи расставить веса для каждой связи нейронной сети.
4. Записать полученную нейронную сеть на языке программирования *Python*.
5. Сделать выводы по проделанной работе.

Структура отчета

1. Титульный лист.
2. Цель работы.
3. Постановка задачи.
4. Схема нейронной сети с расставленными весами.
5. Листинг кода нейронной сети.
6. Скриншот выполнения кода работы.
7. Выводы по работе.

Лабораторная работа №2

Библиотека ГГТУ им.П.О.Сухого

Обучение нейронной сети методом обратного распределения ошибки и градиентного спуска

Цель работы: научиться обучать нейронные сети, освоить методы обучения нейронных сетей: метод обратного распределения ошибки и градиентного спуска.

Задание

1. Нейронную сеть, полученную в предыдущей лабораторной работе, обучить методом обратного распределения ошибки и градиентного спуска.
2. Вывести численное значение полученных весов для каждой связи и сравнить с весами, полученными в предыдущей работе.
3. Сделать выводы по проделанной работе.

Структура отчета

1. Титульный лист.
2. Цель работы.
3. Постановка задачи.
4. Листинг кода нейронной сети.
5. При использовании облачной платформы *Google*, в отчете сделать ссылку на созданный проект.
6. Скриншот выполнения кода работы.
7. Результаты исследований.
8. Выводы по работе.

Лабораторная работа №3

Исследование полносвязной нейронной сети для классификации изображений цифр БД MNIST

Цель работы: научиться строить многослойную, полносвязную нейронную сеть, а также подготавливать обучающие, проверочные и тестовые данные для обучения многослойной сети.

Задания на лабораторную работу

Библиотека ГГТУ им. П.О. Сухого

Для *БД MNIST* – изображений рукописных цифр создать архитектуру НС с наименьшим числом нейронов, достаточных для правильной классификации изображений тестовой выборки на уровне не менее 97%. Обучить НС с контролем эффекта переобучения путем использования выборки валидации.

Дополнительно выполнить задания по вариантам из таблицы 1.

Таблица 1 Варианты заданий

№	Параметр
1	Вывести структуру НС с помощью метода summary() и пояснить число весовых коэффициентов для каждого слоя (почему именно столько)
2	Самостоятельно выполнить разбиение обучающей выборки на собственно обучающую и выборку валидации. В выборке валидации должно быть 10 000 наблюдений, отобранных случайным образом из обучающей выборки. Обучить НС с полученными выборками
3	Сократить обучающую выборку до размеров 20 000 наблюдений, отобранных случайным образом из исходной обучающей выборки. Обучить НС с новой выборкой и сравнить качество обучения с предыдущим случаем (когда использовалась вся обучающая выборка)
4	Провести анализ качества работы НС при разных функциях активации нейронов скрытых слоев: линейная, логистическая, гиперболический тангенс, ReLU
5	Провести анализ качества работы НС, построенной без использования смещений (bias – биасов)
6	Провести анализ качества обучения НС для разных оптимизаторов: Adam, RMSProp, моменты Нестерова. Для каждого оптимизатора подобрать наилучшие параметры для решения задачи классификации рукописных цифр
7	Провести анализ качества обучения НС при разных функциях потерь: – бинарная кросс-энтропия (binary crossentropy) – категориальная кросс-энтропия (categorical crossentropy) – средний квадрат ошибок (mean squared error)
8	Провести анализ качества обучения НС при разных размерах мини-батчей. Определить размер мини-батчей для обеспечения быстрого и качественного обучения сети
9	Вывести первые пять неверно распознанных изображений обученной НС. Объяснить результаты неверной классификации.

Библиотека ГГТУ им. П.О.Сухого

Структура отчета

1. Титульный лист.
2. Цель работы.
3. Постановка задачи.
4. Листинг кода нейронной сети.
5. При использовании облачной платформы *Google*, в отчете сделать ссылку на созданный проект.
6. Скриншот выполнения кода работы.
7. Результаты исследований.
8. Выводы по работе.

Лабораторная работа №4

Обучение нейронной сети на примере распознавания рукописных цифр на языке Python

Цель работы: научиться создавать нейронные сети для распознавания рукописных цифр, освоить обучения нейронных сетей и повышение качества обучения за счет изменения гиперпараметров нейронной сети.

Задание 1.

1. Создать нейронную сеть на языке программирования *Python*, которая распознает цифры по картинке.
2. В нейронной сети должно быть:

– На входном слое *800* нейронов

– На выходном слое *10* нейронов

– Предусмотреть как минимум один скрытый слой.

3. При обучении нейронной сети использовать метод обратного распространения ошибки в сочетании с градиентным спуском.
4. Для обучения и тестирования нейронной сети использовать базу данных *MNIST*.

Библиотека ГГТУ им.П.О.Сухого

5. В модель *fit* добавить проверочную выборку. Построить зависимость доли правильных ответов нейронной сети от эпох обучения на обучающей и проверочной выборках.

6. Повысить качество обучения нейронной сети за счет изменения гиперпараметров сети. Выбор гиперпараметров определяется согласно своему варианту из таблицы 2.

7. Построить график зависимости точности обучения нейронной сети от изменяемого гиперпараметра нейронной сети.

8. Сделать расширенные выводы о влиянии гиперпараметра на обучение сети, скорость обучения и переобучения сети, если таковое присутствует.

9. Сохранить обученную сеть в файл.

Задание 2.

1. Скачать обученную нейронную сеть из файла (задание1).

2. Применить скачанную нейронную сеть для распознавания своих рукописных цифр (использовать не менее 5 различных вариантов написания цифр).

3. Сделать выводы по результатам распознавания.

Таблица 2 Варианты заданий

№	**Варьируемый гиперпараметр**
1	Количество эпох обучения
2	Количество нейронов входного слоя
3	Размер мини-выборки
4	Количество скрытых слоев
5	Количество нейронов скрытого слоя
6	Количество скрытых слоев и нейронов на них
7	Размер проверочной выборки

Библиотека ГГТУ им. П.О.Сухого

№	Варьируемый гиперпараметр
8	Активационная функция

Структура отчета

1. Титульный лист.
2. Цель работы.
3. Постановка задачи.
4. Листинг кода нейронной сети.
5. Скриншот выполнения кода работы.
6. График зависимости точности обучения нейронной сети от изменяемого гиперпараметра сети.
7. График зависимости доли правильных ответов нейронной сети от эпох обучения на обучающей и проверочной выборках.
8. Выводы о влиянии гиперпараметра на обучение нейронной сети.
9. При использовании облачной платформы *Google*, в отчете сделать ссылку на созданный проект.
10. Выводы по работе.

Лабораторная работа №5

Исследование сверточной нейронной сети для классификации полноцветных изображений из БД CIFAR-10

Цель работы: научиться строить и обучать многослойную сверточную нейронную сеть правильно классифицировать полноцветные изображения.

Задание на лабораторную работу

Для *БД CIFAR-10* – полноцветных изображений реальных объектов создать архитектуру сверточной НС с наименьшим числом нейронов, достаточных для правильной классификации изображений

Библиотека ГГТУ им. П.О.Сухого

тестовой выборки на уровне не менее 92%. Обучить НС с контролем эффекта переобучения путем использования выборки валидации. Дополнительно выполнить задания по вариантам.

Таблица 3 Варианты заданий

№	Параметр
1	Провести анализ качества обучения НС при разных функциях потерь: – бинарная кросс-энтропия (binary crossentropy) – категориальная кросс-энтропия (categorical crossentropy) – средний квадрат ошибок (mean squared error)
2	Провести анализ качества работы НС при разных функциях активации нейронов скрытых слоев: линейная, логистическая, гиперболический тангенс, ReLU.
3	Самостоятельно выполнить разбиение обучающей выборки на собственно обучающую и выборку валидации. В выборке валидации должно быть 15 000 наблюдений, отобранных случайным образом из обучающей выборки. Обучить НС с полученными выборками.
4	Провести анализ качества обучения НС при разных размерах мини-батчей. Определить размер мини-батчей для обеспечения быстрого и качественного обучения сети.
5	Провести анализ качества обучения НС для разных оптимизаторов: Adam, RMSProp, моменты Нестерова. Для каждого оптимизатора подобрать наилучшие параметры для решения задачи классификации рукописных цифр
6	Добавить (или убрать, если они были в исходной сети) слои Dropout и сравнить качество классификации до и после внесенных изменений. Подобрать параметр слоя Dropout для получения лучшей классификации тестовой выборки. Объяснить полученные результаты
7	Добавить (или убрать, если они были в исходной сети) слои BatchNormalization и сравнить скорость обучения и качество классификации до и после внесенных изменений.

Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Постановка задачи.
4. Листинг кода нейронной сети.
5. Скриншот выполнения кода работы.
6. Результаты исследований.
7. Выводы по работе.

Библиотека ГГТУ им. П.О. Сухого

Лабораторная работа №6

Сентимент-анализ высказываний с помощью рекуррентных LSTM и GRU сетей

Цель работы: изучить теорию рекуррентных нейронных сетей, способы предобработки текстовой информации для задачи сентимент-анализа, а также научиться строить и обучать рекуррентные сети на базе ячеек *LSTM* и *GRU*.

Задание на лабораторную работу

1. Изучить теорию по рекуррентным нейронным сетям.

2. Каждый студент группы должен самостоятельно написать 200 высказываний (100 – положительных, 100 – отрицательных). В итоге размер обучающей выборки должен составить:

$200 \cdot N$ наблюдений,

где N – число студентов в группе.

Примечание: текст ниоткуда не копировать (высказывания повторяться не должны), писать самостоятельно, используя литературный язык, подойти ответственно.

3. Каждому студенту подготовить по 50 высказываний для тестовой выборки (25 – положительных, 25 – отрицательных). Размер тестовой выборки составит:

$50 \cdot N$ наблюдений.

4. Реализовать РНС по модели *Many to One* с использованием LSTM ячейки (число нейронов в ячейках студент определяет самостоятельно) с входным *Embedding* слоем. На выходе последнего рекуррентного слоя должна идти полносвязная НС (структура

Библиотека ГГТУ им. П.О. Сухого

определяется также самостоятельно) с числом выходных нейронов, равных двум положительное или отрицательное высказывание).

5. В процессе обучения РНС на выходе должна давать оценку окраски текста (положительный или отрицательный).

6. При обучении РНС контролировать процесс переобучения и устранять его (при необходимости) путем изменения структуры сети, а также (возможно) с помощью *Dropout* и *Batch Normalization*.

7. Оценить результаты работы полученной РНС на тестовой выборке.

8. Сделать то же самое обучение РНС, но на базе ячейки *GRU*. Сравнить скорость обучения и качество работы с предыдущей реализацией на базе *LSTM*.

8. В ходе работы добиться как можно лучших результатов сентимент-анализа текстовых высказываний.

Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Постановка задачи.
4. Листинг кода нейронной сети.
5. Скриншот выполнения кода работы.
6. Результаты исследований.
7. Выводы по работе.

Библиотека ГГТУ им. П.О. Сухого

Заключение

За последние несколько лет нейронные сети и глубокое обучение стали одной из самых актуальных тем в машинном обучении. Они показали потрясающие результаты во многих областях, таких как компьютерное зрение, обработка естественного языка, распознавание речи, рекомендательные системы и т.д.

В работе было рассмотрено несколько примеров применения нейронных сетей в задачах классификации и анализа текстов. Начали с изучения основных компонентов нейронных сетей и принципов их работы, затем перешли к реализации некоторых задач с помощью библиотеки *Keras*.

Изучили проблему переобучения и способы ее решения, а также рассмотрели важность разделения данных на обучающую, проверочную и тестовую выборки.

Были рассмотрены различные методы представления текста в числовом виде, такие как *one hot encoding* и *embeddings*. Мы также изучили рекуррентные нейронные сети и применили их для анализа тональности текстов.

Наконец, рассмотрели примеры использования одномерных сверточных нейросетей для классификации текстов, в том числе многозначной классификации.

Нейронные сети – это мощный инструмент, который позволяет извлекать сложные закономерности из данных. Они широко используются в индустрии и научных исследованиях и будут продолжать развиваться в будущем.

Библиотека ГГТУ им. П.О. Сухого

Литература

1. Барский, А. Б. Нейронные сети: распознавание, управление, принятие решений / А. Б. Барский. – М. : Финансы и статистика, 2004. – 176 с.

2. Селянкин, В. В. Решение задач компьютерного зрения : учебное пособие / В. В. Селянкин. – Таганрог : Юж. федер. ун-т, 2016. – 93 с. – Режим доступа: http://biblio-club.ru/index.php?page=book&id=493304. – Дата доступа: 08.04.2022.

3. Советов, Б. Я. Моделирование систем : учеб. / Б. Я. Советов. – Изд. 5-е, стер. – М. : Высш. шк., 2007. – 343 с.

4. Хайкин, С. Нейронные сети: полный курс / С. Хайкин. – Изд. 2-е. – М. ; СПб. ; Киев : Вильямс. 2017. – 1103 с.

5. Шелудько, В. М. Язык программирования высокого уровня Python: функции, структуры данных, дополнительные модули : учебное пособие / В. М. Шелудько. – Ростов н/Д ; Таганрог : Юж. федер. ун-т. – 2017. – 108 с. – Режим доступа: http://biblio-club.ru/index.php?page=book&id=500060. – Дата доступа: 08.04.2022.

Библиотека ГГТУ им. Р.Е. Алексеева

Прокопенко Дмитрий Викторович

ПРОГРАММИРОВАНИЕ НЕЙРОННЫХ СЕТЕЙ

Практикум
по выполнению лабораторных работ
для студентов специальности 1-40 04 01
«Информатика и технологии программирования»
дневной формы обучения

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 11.08.24.

Рег. № 136Е.
http://www.gstu.by

Библиотека ГГТУ им. П.О.Сухого