1. [Home](https://digitrain.ru/)
2. [Публикации](https://digitrain.ru/articles/)
3. Обнаружение аномалий с помощью автоэнкодеров в TensorFlow 2.0

Руководство по внедрению нейронных сетей в TensorFlow 2.0 для обнаружения аномалий.



**В этом подробном руководстве я объясню, как глубокое обучение можно использовать в области обнаружения аномалий. Кроме того, я объясню, как реализовать модель глубокой нейронной сети для обнаружения аномалий в TensorFlow 2.0. Весь исходный код и соответствующий набор данных, конечно же, доступны для скачивания - хорошо;)**

Таблица содержания

1. **Введение**
2. **Обнаружение аномалий**
3. **Случаи использования для систем обнаружения аномалий**
4. **Пример использования аномалии: финансовое мошенничество**
5. **Как работает автоэнкодер?**
6. **Обнаружение аномалий с помощью AutoEncoder**
7. **Обнаружение мошенничества в TensorFlow 2.0**

1. Введение

Аномалия относится к экземпляру данных, который значительно отличается от других экземпляров в наборе данных. Часто они безвредны. Это могут быть только статистические выбросы или ошибки в данных. Но иногда аномалия в данных может указывать на некоторые потенциально опасные события, которые произошли ранее.

**Такое событие может представлять собой финансовое мошенничество.**

В этой статье я покажу вам, как использовать возможности глубоких нейронных сетей для выявления мошеннических транзакций с финансовыми кредитными картами, которые могут быть идентифицированы как аномалии в наборе данных.

2. Обнаружение аномалий

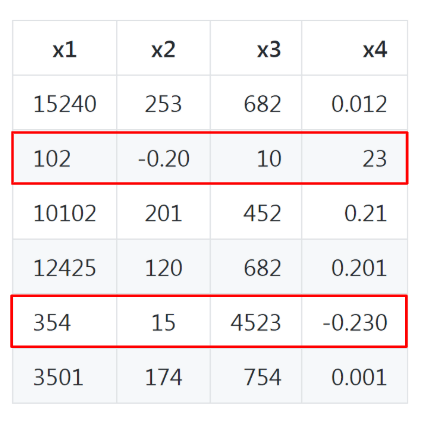
К сожалению, обычные нейронные сети с прямой связью плохо подходят для обнаружения аномалий. По этой причине я представлю новую архитектуру нейронной сети под названием Autoencoder.

Но сначала я хотел бы более подробно объяснить термин «аномалия» и показать, почему системы обнаружения аномалий играют такую ​​важную роль в области прогнозной аналитики.



Обычное наблюдение, которое можно сделать при анализе реальных данных, заключается в том, что некоторые экземпляры или наблюдения в наборе данных могут вызывать подозрения, поскольку они значительно отличаются от большинства данных в наборе данных. Эти конкретные экземпляры данных отличаются тем, что они не соответствуют другим ожидаемым шаблонам или поведению в наборе данных. Такие случаи обычно называют аномалиями.

Пожалуйста, рассмотрите следующий набор данных, который состоит из 4 функций.



При более внимательном рассмотрении в этом наборе данных обнаруживаются некоторые нарушения. Значения во втором и пятом экземплярах данных сильно отличаются от значений в других экземплярах. Значения функций этих двух экземпляров сильно отличаются от соответствующих значений функций других экземпляров.

Ясно, что здесь мы наблюдаем две аномалии. Аномалии в данных могут возникать по нескольким причинам. Иногда аномалии - это ошибки в данных, которые произошли во время сбора данных или предварительной обработки.

Конечно, также велика вероятность того, что аномалии могут принадлежать реальным данным. В этом случае они не более чем статистические выбросы. С другой стороны, аномалии могут иногда указывать на новое, ранее неизвестное основное событие, которое изначально вызвало эти аномалии.

В этом случае аномалии в данных могут указывать на такие события, как мошенничество, злоупотребление или нарушение обслуживания, все из которых представляют опасность для бизнеса или организации. Вопрос, который вы можете задать себе сейчас, заключается в том, почему мы можем рассматривать эти события как аномалии в данных.

Простой ответ заключается в том, что мошенничество, злоупотребление или нарушение обслуживания - это, конечно, события, которые не соответствуют обычному, ожидаемому поведению или процессам в бизнесе или организации. На самом деле это довольно редкие события.

3. Случаи использования для систем обнаружения аномалий

Давайте посмотрим на какую-то конкретную сферу бизнеса, где аномалии в данных могут указывать на потенциальную угрозу или проблему.

Изображение выглядит как текст, логотип, Шрифт, Графика

Автоматически созданное описание

**Банковское дело**. В банковской сфере аномалии могут быть связаны с необычно высокими покупками / депозитами или кибер-вторжениями.



**Здравоохранение**. В сфере здравоохранения мошенничество с претензиями и платежами может выделяться как аномалия. Но что еще более важно, отклонения в данных о здоровье пациента могут указывать на заболевание или ухудшение состояния здоровья.



**Производство**: при производстве ненормальное поведение машины может быть зарегистрировано как аномалия в данных, производимых машиной.



**Финансы**: как уже упоминалось в предыдущем примере, в области финансов (которые также могут относиться к банковскому делу и наоборот) мошеннические финансовые транзакции могут быть зарегистрированы как экземпляры аномальных данных.



**Умный дом**: в случае умного дома утечка энергии может вызвать неожиданные наблюдения в данных.



**Телекоммуникации**: злоупотребление роумингом, мошенничество с доходами, сбои в предоставлении услуг могут быть признаны в данных ненормальными случаями.

Как видно из этих примеров, обнаружение аномалий является важной частью множества сфер бизнеса. Модели обнаружения аномалий могут защитить компании и учреждения от финансового и личного ущерба. И даже стать потенциальным спасателем в сфере здравоохранения.

4. Случай аномального использования: финансовое мошенничество.

Возьмем, к примеру, мошенничество. Представьте, что вы снимаете деньги со своего банковского счета. Вы делаете это один раз в неделю, каждый раз в банкомате дома. И вы снимаете деньги каждый раз, когда сумма в диапазоне, скажем, от 100 до 250 долларов. Конечно, каждый раз, когда вы снимаете деньги, ваш банк собирает данные, связанные со снятием денег, такие как время, место, сумма денег и т. Д.



Пока вы придерживаетесь своего обычного шаблона, экземпляры собранных данных выглядят почти так же. А теперь представьте себе случай, когда ваша карта была украдена, и воры узнали ваш персональный идентификационный номер банкомата. Воры используют эту возможность и снимают крупную сумму денег, намного превышающую вашу обычную сумму, в банкомате, который находится не в вашем городе.

Как вы понимаете, экземпляр данных, содержащий соответствующую информацию для этого конкретного снятия наличных, значительно отличается от предыдущих экземпляров с точки зрения суммы денег и места снятия.

В этом случае экземпляр данных привлек бы внимание как аномалия.

Конечно, аномалия и вид угрозы, которую она может предполагать, зависят от отрасли и соответствующего типа данных. В любом случае целью моделей обнаружения аномалий является обнаружение аномальных данных, чтобы можно было предпринять шаги для дальнейшего исследования обнаруженных аномалий и предотвращения возможных угроз или проблем для компании или ее клиентов.

5. Как работает автоэнкодер?

В оставшейся части статьи я сосредоточусь исключительно на финансовом мошенничестве как на примере выявления аномалий.

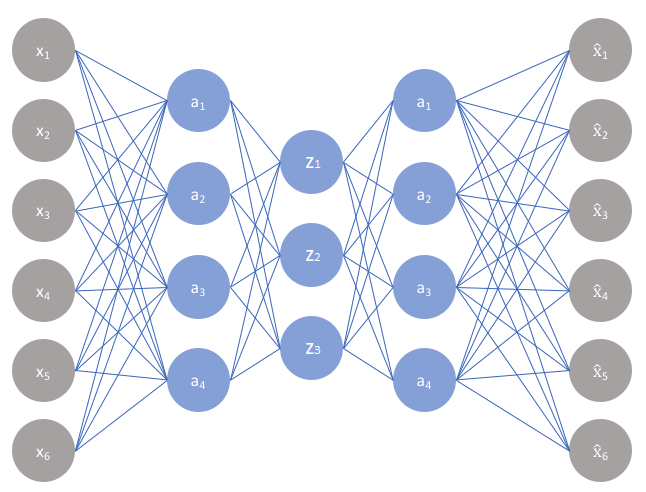
Это время, когда мы можем использовать глубокие нейронные сети в наших интересах. Нейронные сети - это универсальные системы распознавания образов, которые могут идентифицировать закономерности и связывать их с мошенническим поведением, которое мы, люди, никогда бы не рассмотрели.

Например, шаблоны, такие как время, проведенное над определенной кнопкой или областью экрана во время подготовки транзакции.

Человек может никогда не узнать, связана ли эта модель с мошенничеством или нет. Однако для нейронных сетей это совсем не проблема. Архитектура нейронной сети, которая хорошо подходит для идентификации таких шаблонов, называется автоэнкодером.

Подробное описание архитектуры нейронной сети, безусловно, выходит за рамки данной статьи. Вместо этого я хотел бы дать краткий обзор этой сети. Для более подробного объяснения Autoencoder, пожалуйста, прочтите статью [Deep Autoencoder in TensorFlow 2.0](https://www.deeplearning-academy.com/p/ai-wiki-deep-autoencoder).

Самая простая форма автоэнкодера - это нейронная сеть прямого распространения, с которой вы уже знакомы. Как и в случае нейронных сетей с прямой связью, автоэнкодер имеет входной слой, выходной слой и один или несколько скрытых слоев. На следующем изображении показана архитектура автоэнкодера. Как мы видим, входной и выходной уровни автоэнкодера имеют одинаковое количество нейронов.



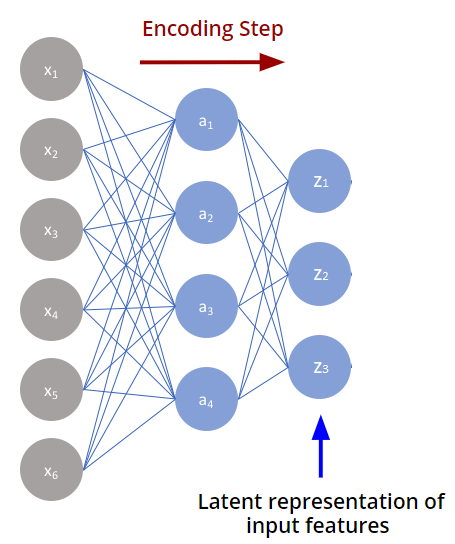
Автоэнкодер можно разделить на две части:

* Кодировщик
* Декодер

Кодировщик

Кодировщик относится к первой половине автокодировщика, где количество скрытых нейронов уменьшается по мере того, как сеть становится глубже.

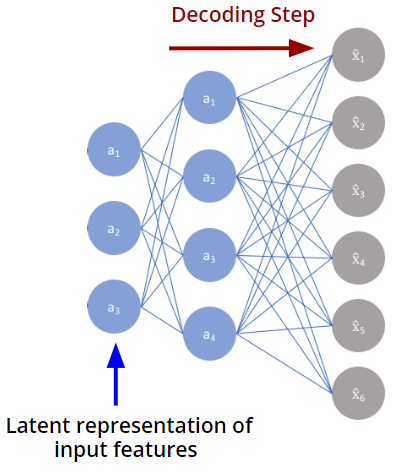
Уменьшение количества нейронов вынуждает «кодировать» или сжатие входных функций *x* в более короткое представление, которое можно найти в среднем скрытом слое. Назовем это представление ***x*** скрытым вектором ***z***.



Декодер

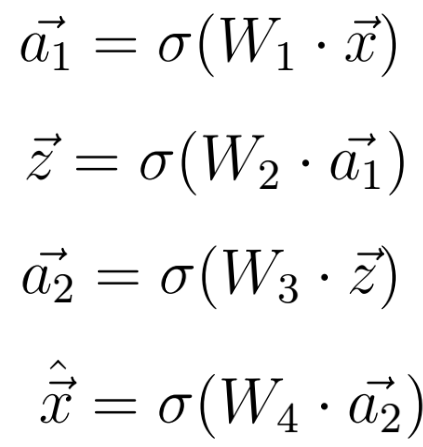
Декодер представляет собой часть автоэнкодера, где количество нейронов в скрытых слоях снова увеличивается.

Более короткое скрытое представление входных элементов, закодированных в средний скрытый слой, используется декодером для восстановления исходных входных элементов ***x***. Назовем реконструированный вход ***x\_hat***.



Если декодер может восстановить входные характеристики ***x*** из этого гораздо более короткого вектора ***z***, то это означает, что ***x*** содержит много нерелевантной информации, от которой можно отказаться.

Уравнения, описывающие весь процесс кодирования и декодирования ***x***, выглядят следующим образом:



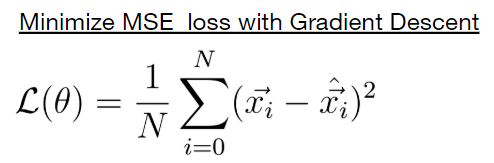
Здесь **W1**, **W2**, **W3** и **W4** представляют матрицы весов, которые соединяют уровни автоэнкодера. сигма представляет собой произвольную нелинейную функцию активации.

Это скрытое представление ***z*** очень важно, поскольку оно может использоваться для множества различных целей, например для обнаружения мошенничества при финансовой транзакции, как мы увидим через несколько минут. Сначала мы обсудим, как обучить автокодировщик, чтобы получить точное скрытое представление входных данных.

Обучение автоэнкодеру

Автоэнкодер обучается так же, как нейронная сеть с прямой связью. Мы должны просто минимизировать расстояние между входными объектами и их реконструированным аналогом ***x\_hat***.

Расстояние между ***x*** и ***x\_hat*** можно правильно измерить с помощью функции потерь среднеквадратичной ошибки. Чтобы минимизировать расстояние или значение функции потерь, мы должны использовать метод обычного градиентного спуска:



Минимизируя функцию потерь MSE, мы уменьшаем разницу между ***x*** и ***x\_hat***. Это автоматически приводит к лучшему скрытому представлению ***z*** входных функций ***x***.

6. Обнаружение аномалий с помощью AutoEncoder

Теперь, когда мы узнали, что такое AutoEncoder и как его можно обучить, давайте обсудим, как мы можем использовать эту архитектуру нейронной сети для обнаружения аномалий в данных.

Мы должны четко осознавать тот факт, что аномалии в наборе данных - очень редкие события.

Это означает, что наборы данных, которые следует проверять на предмет аномалий любого рода, очень несбалансированы. Подавляющее большинство экземпляров в наборе данных будут абсолютно нормальными - в отличие от очень немногих экземпляров аномальных данных.

Для сравнения: в следующем примере обнаружения аномалии набор данных, который мы собираемся использовать, будет содержать более 280000 экземпляров данных транзакций по кредитным картам, из которых только 492 являются мошенническими. *Это всего 0,17% экземпляров данных, которые можно рассматривать как аномалии.* И этот дисбаланс - серьезная проблема.

Упомянутый набор данных является знаменитым [Набором данных о мошенничестве с кредитными картами](https://www.kaggle.com/mlg-ulb/creditcardfraud). Он содержит транзакции, совершенные с помощью кредитных карт в сентябре 2013 года держателями карт из Европы. В этом наборе данных представлены транзакции, которые произошли за два дня, из которых у нас 492 мошенничества из 284 807 транзакций.

Набор данных выглядит следующим образом:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

* Функции **V1**,…, **V28** - это основные компоненты, полученные с помощью PCA (анализа главных компонентов).
* **«Время**»: секунды, прошедшие между каждой транзакцией и самой первой транзакцией.
* **«Сумма**»: сумма транзакции.
* **«Класс**»: это наш ярлык. 1 представляет собой мошенническую транзакцию, 0 в противном случае

Обучить автоэнкодер для обнаружения аномалий

У нас, как и раньше, возникнут трудности с контролируемым обучением нейронных сетей на основе меток. Потому что на этот раз у нас просто недостаточно экземпляров данных, описывающих аномалии, которые мы хотим обнаружить - в нашем случае мошеннические транзакции.

Следовательно, нашей модели может быть очень сложно изучить общую концепцию аномалии или мошеннической транзакции просто потому, что модель почти никогда не видит таких экземпляров. Добавьте к этому тот факт, что нам нужно разделить набор данных на набор для обучения, тестирования и проверки, что еще больше уменьшит количество аномалий в соответствующих наборах данных.

Общее следствие очень ограниченного числа экземпляров аномальных данных состоит в том, что наша модель будет очень неэффективной при классификации этих событий, поскольку в подавляющем большинстве случаев модель учится только на нормальных экземплярах данных. К счастью, это время, когда мы можем использовать автоэнкодер, чтобы помочь нам там. Мы можем использовать уникальные свойства этой нейронной сети, чтобы решить проблему очень несбалансированного набора данных.

Это означает, что вместо предоставления меток, которые классифицируют входные функции как мошеннические или нет, мы сравниваем предсказание автоэнкодера с исходными входными функциями.

Кроме того, в случае обнаружения мошенничества мы должны обучать автокодировщик только на экземплярах данных, не являющихся мошенническими. Во время обучения Autoencoder будет видеть миллионы только не мошеннических транзакций по кредитным картам.

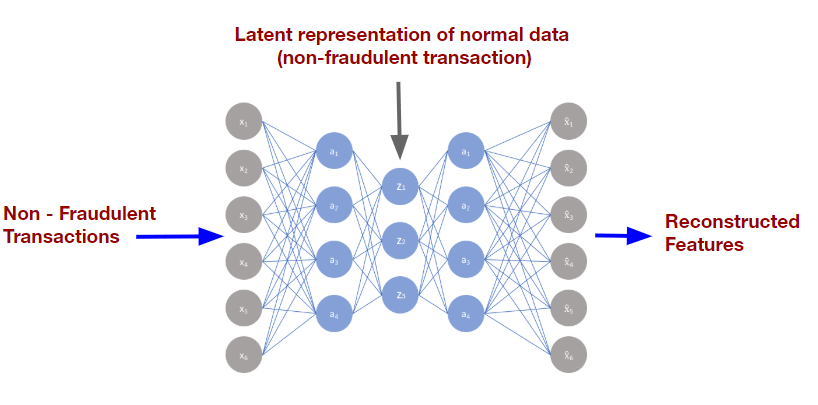
Изображение выглядит как текст, снимок экрана, диаграмма, круг

Автоматически созданное описание

Таким образом, мы используем автоэнкодер для кодирования основной или наиболее релевантной информации входных функций в более короткое скрытое представление. Интуитивно мы можем сказать, что таким образом автоэнкодер изучает только концепцию абсолютно нормальной транзакции по кредитной карте.

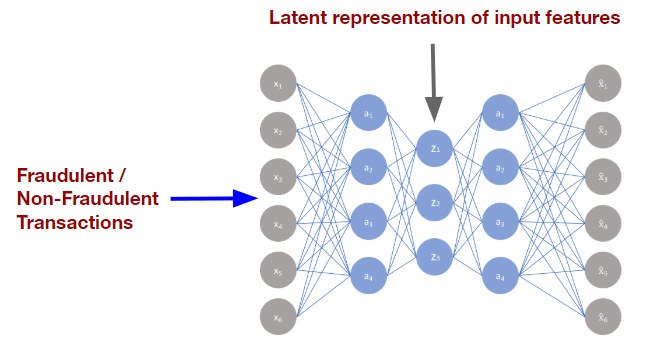
И эту изученную концепцию обычной транзакции по кредитной карте можно найти как скрытое представление на среднем уровне, которое используется для воссоздания исходных функций, используемых в качестве входных данных для автоэнкодера.

Используйте обученный автоэнкодер для обнаружения аномалий



После обучения с обычными экземплярами данных мы, наконец, можем использовать нейронную сеть для обнаружения аномалий.

На этот раз мы можем показать сетевые данные обоих типов - аномалии, а также нормальные данные. Как и раньше, входные функции кодируются автоэнкодером в скрытое представление, которое используется для восстановления входных данных.



Как уже упоминалось, AutoEncoder использует функцию среднеквадратичной ошибки в качестве функции потерь для измерения разницы или ошибки между восстановленным и исходным входными данными.

**Обратите внимание на следующее:** если автокодировщик обучен правильно, мы ожидаем, что ошибка между выходом и входом будет довольно небольшой для нормальных данных.

***Однако это не относится к аномалиям или, в нашем случае, мошенническим транзакциям с данными.***

Помните, что во время обучения автоэнкодер изучил только концепции и характеристики обычного экземпляра данных. Это означает, что веса и смещения автоэнкодера были скорректированы только для кодирования и восстановления обычных данных - в нашем случае не мошеннических транзакций.

Если теперь мы попытаемся закодировать мошенническую транзакцию, скрытое представление этой транзакции будет значительно отличаться от скрытого представления нормальной транзакции.

**В результате восстановленные входные данные будут еще больше отличаться от исходных, что приведет к гораздо большей ошибке, чем в случае обычных транзакций.**

Знание того факта, что мошеннические входные данные приводят к более высоким значениям потерь функции среднеквадратичной ошибки, может быть использовано в наших интересах. Все, что нам нужно сделать, это найти **порог потерь**, который позволяет отличить обычные данные от мошеннических.

**На практике это будет означать, что экземпляры данных, для которых мы получаем значение потерь, превышающее этот порог, классифицируют этот экземпляр как аномалию или, в нашем случае, как мошенническую транзакцию.**

С другой стороны, экземпляры данных со значениями потерь ниже этого порога можно рассматривать как обычные данные или как нечестные транзакции.

* **Случай 1.** Значение потерь MSE для входной функции **выше порога потерь** → входная функция является аномалией (здесь: мошенническая транзакция)
* **Случай 2:** значение потерь MSE для входной функции **ниже порогового значения** → входная функция является нормальной (здесь: не мошенническая транзакция)

Резюме: Обнаружение аномалий с помощью автоэнкодера

Подведем итог тому, что мы узнали о том, как обнаруживать аномалии в наборе данных.

1. Во-первых, если ваш набор данных, который необходимо проверить на наличие аномалий, несбалансирован (что почти всегда так), вы должны использовать автокодировщик.
2. Обучите автокодировщик, используя функцию потерь среднеквадратичной ошибки, только на нормальных данных, не используйте экземпляры аномальных данных во время обучения.
3. После обучения значение потерь для аномалий должно быть намного выше, чем для обычных экземпляров данных.
4. Теперь вам нужно определить порог значения потерь, который лучше всего отличает аномалии от нормальных данных.

Определение правильного порога стоимости убытков

Чтобы найти это пороговое значение, которое наилучшим образом отличает аномалии от нормальных данных, вам просто нужно попробовать множество различных пороговых значений.

Для этой процедуры я рекомендую взять несколько тысяч экземпляров данных, где примерно 5–10% этих экземпляров являются аномалиями. Затем вы просто вычисляете для каждого экземпляра значение потерь, сравниваете его с порогом и классифицируете его как аномалию или нет.

В конце концов, вы должны проверить, насколько хороши эти классификации, используя настоящие ярлыки. Для количественной оценки результата классификации я рекомендую использовать метрики оценки, которые мы рассмотрели в предыдущей статье [Метрики оценки в науке о данных и машинном обучении](https://www.deeplearning-academy.com/p/ai-wiki-evaluation-metrics-in-data-science) «.

Имея результаты показателей оценки, вы можете решить, был ли этот порог хорошим, и при необходимости скорректировать порог.

7. Обнаружение мошенничества в TensorFlow 2.0

Как вы, возможно, уже догадались, моделью обнаружения аномалий будет автоматический кодировщик, который будет определять мошеннические финансовые транзакции в ранее представленном наборе данных.

Весь исходный код и используемые наборы данных доступны [в моем репозитории GitHub этого проекта](https://github.com/artem-oppermann/Anomaly-Detection-in-TensorFlow-2.0). Не стесняйтесь, загрузите код и попробуйте сами.

К сожалению, я не могу подробно рассказать обо всех используемых функциях и классах, потому что это выходит за рамки курса. Скорее я хотел бы сосредоточиться исключительно на реализации модели нейронной сети. Все остальные используемые классы и методы (в частности, предварительную обработку данных можно посмотреть в репозитории GitHub). Весь исходный код хорошо документирован, поэтому у вас не возникнет трудностей с пониманием кода.

На этой ноте давайте начнем ...

Сначала мне нравится определять класс **BaseModel**, который содержит методы для инициализации весов и смещений и вычисления прямого прохода.

Эти методы позже будут унаследованы классу, который будет определять фактическую модель обнаружения аномалий. Таким образом, позже мы сможем определить несколько моделей автоэнкодеров (например, с разными гиперпараметрами), которые будут наследовать методы от класса **BaseModel**. Делая это, мы можем сэкономить довольно много кода, поскольку разные версии модели имеют одни и те же функции.

В **init\_variables (self)** мы определяем вес и смещения. Как видите, в автоэнкодере всего три скрытых слоя. Количество нейронов в этих слоях составляет [20, 8, 20].

|  |  |
| --- | --- |
|  | import tensorflow as tf |
|  | from tensorflow.keras import Model |
|  |  |
|  | class BaseModel(Model): |
|  | '''This is the base model class that inherits some sharable functions to the the models for the training and inference. |
|  | This class also inherits from the tensorflow.keras.Model. |
|  |  |
|  | ''' |
|  | def \_\_init\_\_(self): |
|  |  |
|  | super(BaseModel, self).\_\_init\_\_() |
|  |  |
|  | self.weight\_initializer=tf.random\_normal\_initializer(mean=0.0, stddev=0.25) |
|  | self.bias\_initializer=tf.zeros\_initializer() |
|  |  |
|  | def init\_variables(self): |
|  | '''Initialize the parameters of the neural network. ''' |
|  |  |
|  | self.W1=tf.compat.v1.get\_variable('W1',shape=[29,20], initializer=self.weight\_initializer, dtype=tf.float32) |
|  | self.W2=tf.compat.v1.get\_variable('W2',shape=[20,8], initializer=self.weight\_initializer, dtype=tf.float32) |
|  | self.W3=tf.compat.v1.get\_variable('W3',shape=[8,20], initializer=self.weight\_initializer, dtype=tf.float32) |
|  | self.W4=tf.compat.v1.get\_variable('W3',shape=[20,29], initializer=self.weight\_initializer, dtype=tf.float32) |
|  |  |
|  | self.b1=tf.compat.v1.get\_variable('b1',shape=[20], initializer=self.bias\_initializer, dtype=tf.float32) |
|  | self.b2=tf.compat.v1.get\_variable('b2',shape=[8], initializer=self.bias\_initializer, dtype=tf.float32) |
|  | self.b3=tf.compat.v1.get\_variable('b3',shape=[20], initializer=self.bias\_initializer, dtype=tf.float32) |
|  |  |
|  |  |
|  | def forward\_propagation(self, x): |
|  | '''Compute the forward pass given the input features x. |
|  |  |
|  | @param x: input features x |
|  |  |
|  | @return prediction: the reconstructed input features x |
|  | ''' |
|  |  |
|  | with tf.name\_scope('feed\_forward'): |
|  |  |
|  | # First hidden layer |
|  | z1=tf.linalg.matmul(x, self.W1)+self.b1 |
|  | a1=tf.nn.relu(z1) |
|  |  |
|  | # Second hidden layer |
|  | z2=tf.linalg.matmul(a1,self.W2)+self.b2 |
|  | a2=tf.nn.relu(z2) |
|  |  |
|  | # Third hidden layer |
|  | z3=tf.linalg.matmul(a2,self.W3)+self.b3 |
|  | a3=tf.nn.relu(z3) |
|  |  |
|  | prediction=tf.linalg.matmul(a3,self.W4) |
|  |  |
|  | return prediction |

**forward\_propagation (self, x)** просто вычисляет выходные данные автокодировщика, то есть реконструированные входные функции.

Далее мы можем определить фактический класс, в котором будет обучаться автокодировщик:

|  |  |
| --- | --- |
|  | class AnomalyDetector(BaseModel): |
|  | '''This class represents the class for training of the neural network for anomaly detection. |
|  | In particular this class is used for training only. The learned weights and biases will be used later |
|  | by the inference model to make the actual anomaly detection in production environment. |
|  | ''' |
|  |  |
|  | def \_\_init\_\_(self): |
|  |  |
|  | super(AnomalyDetector, self).\_\_init\_\_() |
|  | self.init\_variables() |
|  |  |
|  | self.optimizer = tf.keras.optimizers.Adam(learning\_rate=0.001) |
|  |  |
|  |  |
|  | def compute\_loss(self, x\_train): |
|  | '''Compute MSE loss function. |
|  |  |
|  | @param x\_train: input features |
|  | ''' |
|  |  |
|  | mse = tf.keras.losses.MeanSquaredError() |
|  | loss = mse(x\_train, self.forward\_propagation(x\_train)) |
|  |  |
|  | return loss |
|  |  |
|  |  |
|  | def train(self, x\_train): |
|  | '''Train the autoencoder. |
|  |  |
|  | @parameter x\_train: training input features |
|  | ''' |
|  |  |
|  | # Compute the gradients and apply the gradient descent step |
|  | with tf.GradientTape() as tape: |
|  | gradients = tape.gradient(self.compute\_loss(x\_train), self.trainable\_variables) |
|  | gradient\_variables = zip(gradients, self.trainable\_variables) |
|  | self.optimizer.apply\_gradients(gradient\_variables) |

[view raw](https://gist.github.com/artem-oppermann/03225691441091d6750a80bb914633a8/raw/1989e152d6446fe7995023b98c2a8bd4b7e3bce7/anomaly_detection.py)[anomaly\_detection.py](https://gist.github.com/artem-oppermann/03225691441091d6750a80bb914633a8#file-anomaly_detection-py)hosted with ❤ by [GitHub](https://github.com/)

Этот класс содержит методы для вычисления значения потери среднеквадратичной ошибки и шаг градиентного спуска.

Тренировочный процесс должен быть понятным:

|  |  |
| --- | --- |
|  | ### Define some hyperparameters ### |
|  |  |
|  | # Number of training samples |
|  | n\_training\_samples=250000 |
|  | # Batch size |
|  | batch\_size=64 |
|  | # Learning rate |
|  | learning\_rate=0.001 |
|  | # Number of test data samples |
|  | n\_test\_samples=34806 |
|  | # Number of epochs |
|  | num\_epoch=50 |
|  | # number of batches |
|  | n\_batches=int(n\_training\_samples/batch\_size) |
|  | # Evaluate model after number of steps |
|  | eval\_after=1000 |
|  | # Path of the TF Records datasets for the training |
|  | train\_path=os.path.abspath(os.path.join(os.path.dirname("\_\_file\_\_"), '..', 'data/tf\_records/train/')) |
|  | # Path of the TF Records datasets for the testing |
|  | test\_path=os.path.abspath(os.path.join(os.path.dirname("\_\_file\_\_"), '..', 'data/tf\_records/test/')) |
|  |  |
|  | # Loss value threshold |
|  | THRESHOLD=10 |
|  |  |
|  | # Initialize the instance of the class for anomaly detection |
|  | model=AnomalyDetector() |
|  |  |
|  | # Initialihe the instance of the class for the perfromance measurements |
|  | performance=Performance(THRESHOLD) |
|  |  |
|  | # Get the training and test datasets |
|  | training\_dataset=get\_training\_data(train\_path) |
|  | test\_dataset=get\_test\_data(test\_path) |
|  |  |
|  | #Iterate over the epochs |
|  | for epoch in range(num\_epoch): |
|  |  |
|  | temp\_loss=0 |
|  |  |
|  | #Iterate over the batches |
|  | for step, x\_train in enumerate(training\_dataset): |
|  |  |
|  | # Extract features and labels |
|  | features, labels=x\_train |
|  |  |
|  | # Train the model |
|  | model.train(features) |
|  |  |
|  | # Compute the loss |
|  | loss\_values=model.compute\_loss(features) |
|  | temp\_loss+=loss\_values |
|  |  |
|  | # Evalute the model on the test set |
|  | if step>0 and step%eval\_after==0: |
|  |  |
|  | # Iteratre over the test dataset |
|  | for step\_test, x\_test in enumerate(test\_dataset): |
|  |  |
|  | # Extract the features and labels |
|  | features, label=x\_test |
|  |  |
|  | label=label.numpy() |
|  |  |
|  | # Compute the loss |
|  | test\_loss=model.compute\_loss(features) |
|  |  |
|  | # Evaluate the test data sample |
|  | performance.eval\_prediciton(test\_loss, label) |
|  |  |
|  |  |
|  | print('epoch\_nr: %d, batch: %d/%d, mse\_loss: %.3f'%(epoch, step, n\_batches, (temp\_loss/step))) |
|  |  |
|  | # Plot ROC curce and show other evaluation metrics |
|  | performance.evaluate\_model() |
|  | performance.reset() |

[view raw](https://gist.github.com/artem-oppermann/351360f9d48aa2e82c5e5f362251f696/raw/50c3ef85365360df5d089b82e1f2150b3b259b52/training.py)[training.py](https://gist.github.com/artem-oppermann/351360f9d48aa2e82c5e5f362251f696#file-training-py)hosted with ❤ by [GitHub](https://github.com/)

Во-первых, мы определяем несколько гиперпараметров, таких как размер пакета, скорость обучения и т. Д.

Очень важным гиперпараметром является введенный порог величины потерь, который я здесь называю *THRESHOLD = 10*. Это значение функции потерь MSE будет отличать аномалии фазы тестирования от нормальных экземпляров данных.

Затем мы определяем экземпляр классов **AnomalyDetector ()**, который является фактической моделью автоэнкодера, и **Производительность (THRESHOLD)**, где некоторые показатели оценки (точность, отзыв, F1-Score ) будет рассчитан. Если вы не знакомы с этими метриками, ознакомьтесь со статьей [Метрики оценки в глубоком обучении](https://www.deeplearning-academy.com/p/ai-wiki-evaluation-metrics-in-data-science).

Наборы данных для обучения и оценки были ранее преобразованы в формат данных TensorFlow Records. В методах *get\_training\_data ()* и **get\_test\_data ()** я использую **tf.data API** для извлечения, загрузки, сделать мини-пакеты и перемешать наборы данных.

На следующем этапе мы перебираем обучающий набор данных, вычисляем для каждого мини-пакета функцию потерь среднеквадратичной ошибки и применяем градиентный спуск.

После обработки 1000 минипакетов я использую класс **Performance ()** вместе с порогом значения потерь для оценки модели детектора аномалий на прибл. 34000 экземпляров в тестовом наборе данных. Набор тестовых данных содержит все 480 экземпляров мошеннических данных (аномалий), а остальные являются обычными экземплярами данных.

Сразу после 5 эпох мы получаем следующие результаты:

|  |  |
| --- | --- |
|  | epoch\_nr: 5, batch: 1000/3906, mse\_loss: 0.363 |
|  |  |
|  | Precision: 0.930 |
|  | Recall: 0.433 |
|  | F1-Score: 0.591 |
|  |  |
|  | True neg rate: 1.000 |
|  | Accuracy: 0.992 |
|  |  |
|  |  |
|  | Non-fraud loss, mean: 0.385, var: 1.870 |
|  | Fraud loss, mean: 19.518, var: 23.489 |

* Детектор аномалий имеет очень хорошее значение точности - 93. Это означает, что в 93% случаев модель правильно классифицирует аномалию.
* Значение отзыва 43,3 говорит нам, что автоэнкодер обнаружил 43,3% аномалий в наборе данных. К сожалению, более половины аномалий остаются невидимыми для модели.

Интересно отметить, что для экземпляров данных без мошенничества среднее значение абсолютной потери ошибок составляет всего 0,385. С другой стороны, значение потерь MSE для экземпляров мошеннических данных со значением 19,518 намного выше. Так и должно быть, и именно то, что мы ожидаем.