Оглавление

[1. Что такое нейронная сеть. 2](#_Toc90812912)

[1.1. Из чего состоит нейронная сеть. 2](#_Toc90812913)

[1.2. Как нейронная сеть учится. 4](#_Toc90812914)

[2. О используемой библиотеке для машинного обучения 6](#_Toc90812915)

[3. О среде разработки – Google Colab 7](#_Toc90812916)

[4. Жизненный цикл модели глубокого обучения (именно такой тип нейросети и был создан) 7](#_Toc90812917)

[3.1 Жизненный цикл 5-ступенчатой модели 7](#_Toc90812918)

[4.2 Sequential Model API (последовательная модель) 11](#_Toc90812919)

[4.3 Functional Model API (Функциональная модель) 12](#_Toc90812920)

[5. Программа для предсказания значения на основе обучающей выборки 13](#_Toc90812921)

[5.1. Алгоритм работы программы для предсказания значения 13](#_Toc90812922)

[5.2. Код и его объяснение 17](#_Toc90812923)

[5.3.1. Результаты обучения и предсказание значений 21](#_Toc90812924)

[5.3.2. Реализация обновления обученной модели на основе только новых данных 25](#_Toc90812925)

[5.3.3. Обновление обученной модели на основе старых и новых данных 29](#_Toc90812926)

[Приложение 1. Дополнительная информация о некоторых модулях из библиотеки tensorflow 30](#_Toc90812927)

[1. Module: tf.keras.optimizers 30](#_Toc90812928)

[2. Module: tf.keras.losses 30](#_Toc90812929)

[3. Module: tf.keras.metrics 32](#_Toc90812930)

[4. Module: tf.keras.layers 34](#_Toc90812931)

# 1. Что такое нейронная сеть.

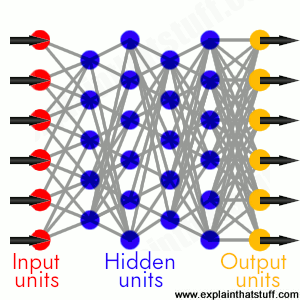
Основная идея нейронной сети состоит в том, чтобы моделировать (копировать упрощенным, но достаточно точным образом) множество плотно связанных между собой «клеток мозга» внутри компьютера, чтобы вы могли научить его, распознавать закономерности и принимать решения по-человечески. Самое удивительное в нейронной сети то, что вам не нужно программировать ее для явного обучения: она обучается сама по себе, как мозг.

Но это не мозг. Важно отметить, что нейронные сети (как правило) представляют собой программную симуляцию: они создаются путем программирования обычных компьютеров, работающих очень традиционным образом со своими обычными транзисторами и последовательно подключенными логическими вентилями, чтобы вести себя так, как будто они построены из миллиардов тесно связанных между собой клеток мозга, работающих параллельно. Никто еще не пытался построить компьютер, соединив транзисторы в плотно параллельную структуру, точно такую ​​же, как человеческий мозг. Другими словами, нейронная сеть отличается от человеческого мозга точно так же, как компьютерная модель погоды отличается от настоящих облаков, снежинок или солнечного света. Компьютерное моделирование — это просто набор алгебраических переменных и математических уравнений, связывающих их вместе (другими словами, числа, хранящиеся в ячейках, значения которых постоянно меняются). Они ничего не значат для компьютеров, внутри которых работают, - только для людей, которые их программируют.

Строго говоря, нейронные сети, созданные образом, описанным выше, называются искусственными нейронными сетями (или ИНС), чтобы отличать их от настоящих нейронных сетей (совокупностей взаимосвязанных клеток мозга), которые мы находим внутри нашего мозга.

## 1.1. Из чего состоит нейронная сеть.

Типичная нейронная сеть имеет от нескольких десятков до сотен, тысяч или даже миллионов искусственных нейронов, называемых **units** (единицами, юнитами), расположенными в серии слоев, каждый из которых соединяется со слоями с обеих сторон. Некоторые из них, известные как **input units** (юниты ввода), предназначены для получения различных форм информации из внешнего мира, которую сеть будет пытаться узнать, распознать или иным образом обработать. Другие устройства находятся на противоположной стороне сети и сигнализируют, как она реагирует на полученную информацию; они известны как **output units** (юниты выходные). Между входными и выходными модулями находится один или несколько слоев **hidden units** (скрытых юнитов), которые вместе составляют большую часть искусственного мозга. Большинство нейронных сетей **fully connected** (полносвязанная), что означает, что каждый скрытые юниты (hidden units) и каждый выходной юнит (output unit) подключены к каждому юниту в слоях с каждой стороны. Связи между одним юнитом и другим представлены числом, называемым **weight** (весом), которое может быть положительным (если один юнит возбуждает другой) или отрицательным (если один юнит подавляет или подавляет другой). Чем больше вес, тем большее влияние оказывает одно устройство на другое. Это соответствует тому, как реальные клетки мозга запускают друг друга через крошечные промежутки, называемые синапсами.

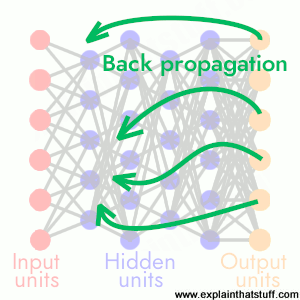


Полносвязанная нейронная сеть состоит из input units (красный), hidden units (синий) и output units (желтый), причем все юниты подключены ко всем юнитам в слоях с каждой стороны. Входы подаются слева, активируют hidden units посередине и отдают выходное воздействие юнитам справа. Сила (weight) связи между любыми двумя устройствами постепенно регулируется по мере обучения сети.

Хотя простая нейронная сеть для решения простых задач может состоять всего из трех слоев, как показано здесь, она также может состоять из множества разных слоев между входом и выходом. Такая более богатая структура называется глубокой нейронной сетью (DNN – deep neural network) и обычно используется для решения гораздо более сложных проблем. Теоретически DNN может сопоставлять любой входной сигнал с любым выходным, но недостатком является то, что он требует значительно большего значения обучающей. Для обучения могут потребоваться миллионы или миллиарды примеров по сравнению с, возможно, сотнями или тысячами, как в более простой сети.

## 1.2. Как нейронная сеть учится.

Информация проходит через нейронную сеть двумя способами. Когда она обучается (до обучения) или работает нормально (после обучения), образцы информации передаются в сеть через input units, которые запускают слои hidden units, а они, в свою очередь, поступают в output units. Этот общий дизайн называется сетью с прямой связью (**feedforward network**). Каждый юнит (unit) получает входные данные от юнитов слева от него, и эти входные данные умножаются на веса (weights) соединений, по которым они перемещаются. Каждый юнит суммирует все входные данные, которые он получает таким образом, и (в простейшем типе сети), *если сумма превышает определенное пороговое значение, определяемое функцией активации*, юнит «срабатывает» и запускает юниты, к которым он подключен (те, что справа).



*Нейронная сеть может обучаться путем обратного распространения ошибки, который представляет собой своего рода процесс обратной связи, который передает корректирующие значения в обратном направлении по сети.*

Чтобы нейронная сеть могла обучаться, должен быть задействован элемент обратной связи – точно так же, как дети учатся, когда им говорят, что они делают правильно или неправильно. Нейронные сети изучают вещи с помощью процесса обратной связи, называемого обратным распространением – **backpropagation** (иногда сокращенно «backprop»). Это включает в себя сравнение выходных данных, которые производит сеть, с выходными данными, которые она должна была производить, и использование разницы между ними для изменения весов соединений между устройствами в сети, работая от output units через hidden units к input units. Другими словами, движение назад (backward). Со временем обратное распространение приводит (backpropagation) к тому, что сеть учится, уменьшая разницу между фактическим и предполагаемым выходом до точки, в которой они точно совпадают, поэтому сеть выясняет вещи именно так, как должна.

Функция активации

Ранее в параграфе 1.2 была фраза «*сумма превышает определенное пороговое значение, определяемое функцией активации*». Объяснение, что такое функция активации представлено ниже.

В искусственных нейронных сетях функция активации нейрона определяет выходной сигнал, который определяется входным сигналом или набором входных сигналов. Стандартная компьютерная микросхема может рассматриваться как цифровая сеть функций активации, которые могут принимать значения «ON» (1) или «OFF» (0) в зависимости от входа.

В биологических нейронных сетях функция активации обычно является абстракцией, представляющей скорость возбуждения потенциала действия в клетке. В наиболее простой форме эта функция является двоичной — то есть нейрон либо возбуждается, либо нет. Функция выглядит как *ϕ(vi)* = *U(vi)* , где U — ступенчатая функция Хевисайда. В этом случае нужно использовать много нейронов для вычислений за пределами линейного разделения категорий. Прямая с положительным угловым коэффициентом может быть использована для отражения увеличения скорости возбуждения по мере увеличения входного сигнала. Такая функция имела бы вид *ϕ(vi)* = *μv*, где μ— наклон прямой. Эта функция активации линейна, а потому имеет те же проблемы, что и двоичная функция. Кроме того, сети, построенные с использованием таковой модели, имеют нестабильную сходимость. Поэтому в нейронных сетях применяют следующие распространённые типы функций:

-сигмоида

-гиперболический тангенс

-ReLU

-и другие

Таблица 1 – Наиболее часто встречающиеся функции активации

|  |  |  |
| --- | --- | --- |
| Название | График | Уравнение |
| Тождественная |  |  |
| Единичная ступенька |  |  |
| Логистическая (сигмоида или Гладкая ступенька) |  |  |
| Линейный выпрямитель (или Полулинейный элемент) (англ. Rectified linear unit, ReLU) |  |  |
| Гауссова |  |  |
| Гиперболический тангенс |  |  |

# 2. О используемой библиотеке для машинного обучения

Подробнее почитать можно тут: <https://www.tensorflow.org/> и, конечно, тут: <https://en.wikipedia.org/wiki/TensorFlow>

**Краткое описание.**

TensorFlow - это бесплатная библиотека программного обеспечения с открытым исходным кодом для машинного обучения и искусственного интеллекта. Его можно использовать для решения ряда задач, но особое внимание уделяется обучению и анализу глубоких нейронных сетей.

TensorFlow был разработан командой Google Brain для внутреннего использования Google в исследованиях и производстве. Первоначальная версия была выпущена под лицензией Apache License 2.0 в 2015 году. Google выпустила обновленную версию TensorFlow под названием TensorFlow 2.0 в сентябре 2019 г.

TensorFlow можно использовать на самых разных языках программирования, в первую очередь на Python, а также на Javascript, C ++ и Java. Эта гибкость позволяет находить множество применений во многих различных секторах.

# 3. О среде разработки – Google Colab

Colaboratory, или сокращенно Colab – продукт компании Google Research. Colab позволяет любому писать и выполнять произвольный код Python через браузер и особенно хорошо подходит для машинного обучения, анализа данных и образования. С технической точки зрения Colab – это размещенная на хосте служба Jupyter для скриптов, которая не требует настройки для использования, но при этом предоставляет бесплатный доступ к вычислительным ресурсам, включая графические процессоры.

Скрипты Colab хранятся на Google Диске или могут быть загружены с GitHub. Скриптами Colab можно делиться так же, как документами или таблицами Google. Код выполняется на виртуальной машине, принадлежащей текущей учетной записи. Виртуальные машины удаляются, если они бездействуют в течение некоторого времени.

Типы графических процессоров, доступных в Colab, со временем меняются. Это необходимо, чтобы Colab могла бесплатно предоставлять доступ к этим ресурсам. Графические процессоры, доступные в Colab, включают в себя Nvidia K80s, T4s, P4s и P100s. Нельзя выбрать, к какому типу графического процессора можно подключиться, пользователи, которые заинтересованы в более надежном доступе к самым быстрым графическим процессорам Colab, могут использовать Colab Pro.

Объем памяти, доступной в виртуальных машинах Colab, меняется со временем (но остается стабильным в течение всего срока работы виртуальной машины). Иногда объём может автоматически назначаться дополнительная память, когда Colab обнаруживает, что она может понадобиться.

# 4. Жизненный цикл модели глубокого обучения (именно такой тип нейросети и был создан)

В этом разделе вы познакомитесь с жизненным циклом модели глубокого обучения и двумя API tf.keras (библиотека машинного обучения от google tensorflow или сокращённо tf, в особенности модулем keras для работы с нейросетями), которые можно использовать для определения моделей.

## 3.1 Жизненный цикл 5-ступенчатой модели

Модель имеет жизненный цикл, и это очень простое знание обеспечивает основу как для моделирования набора данных, так и для понимания API tf.keras.

Пять этапов жизненного цикла **Deep Learning Model**:

1. Определение модели.

2. Компиляция модели.

3. «Подгонка» модели.

4. Оценка модели.

5. Создание прогнозов.

***Определение модели***

Для определения модели необходимо сначала выбрать нужный тип модели, а затем выбрать архитектуру или топологию сети.

С точки зрения API (application programming interface), это включает в себя определение слоев модели, настройку каждого уровня с несколькими узлами и функцией активации и соединение слоев в единую модель.

Модели можно определять либо с помощью последовательного API, либо с помощью функционального API, и мы рассмотрим это в следующем разделе.

|  |  |
| --- | --- |
|  | ...  # define the model  model = ... |

***Компиляция модели***

Компиляция модели требует, чтобы вы сначала выбрали функцию потерь (loss function), которую хотите оптимизировать, например среднеквадратичную ошибку или перекрестную энтропию (mean squared error и cross-entropy).

Это также требует, чтобы вы выбрали алгоритм для выполнения процедуры оптимизации (optimization procedure), обычно стохастический градиентный спуск или современный вариант, такой как Adam (stochastic gradient descent, Adam). Также может потребоваться, чтобы вы выбрали какие-либо показатели производительности (performance metrics), которые нужно отслеживать в процессе обучения модели.

С точки зрения API, это включает вызов функции для компиляции модели с выбранной конфигурацией, которая подготовит соответствующие структуры данных, необходимые для эффективного использования модели, которую вы определили.

Оптимизатор (optimizer) может быть указан как строка для известного класса оптимизатора, например «Sgd» для стохастического градиентного спуска (gradient descent), или вы можете настроить экземпляр класса оптимизатора и использовать его.

|  |  |
| --- | --- |
|  | ...  # compile the model  opt = SGD (learning\_rate = 0.01, momentum = 0.9)  model.compile(optimizer = opt, loss = 'binary\_crossentropy') |

Три наиболее распространенных функции потерь:

* «*binary\_crossentropy*» для двоичной классификации.
* «*sparse\_categorical\_crossentropy*» для мультиклассовой классификации. «*mse*» (среднеквадратичная ошибка) для регрессии.

|  |  |
| --- | --- |
|  | ...  # compile the model  model.compile(optimizer='sgd', loss='mse') |

Метрики определяются как список строк для известных метрических функций или список функций, которые необходимо вызвать для оценки прогнозов.

|  |  |
| --- | --- |
|  | # compile the model  model.compile(optimizer='sgd', loss='binary\_crossentropy', metrics=['accuracy']) |

***«Подгонка» модели***

*Для подгонки модели (её непосредственного обучения)* необходимо сначала выбрать конфигурацию обучения, такую как количество эпох (epochs, циклы по набору обучающих данных) и размер пакета (batch size, количество выборок в эпоху, используемых для оценки ошибки модели).

При обучении применяется выбранный алгоритм оптимизации, чтобы минимизировать выбранную функцию потерь (loss function), и обновлять модель, используя алгоритм обратного распространения ошибки (backpropagation).

Подгон модели – это самая медленная часть всего процесса создания нейросети, который может занять от нескольких секунд до часов или дней, в зависимости от сложности модели, используемого оборудования и размера обучающего набора данных.

С точки зрения API, это включает вызов функции для выполнения процесса обучения. Эта функция будет заблокирована (не вернется) до тех пор, пока процесс обучения не будет завершен.

|  |  |
| --- | --- |
|  | ...  # fit the model  model.fit(X, y, epochs=100, batch\_size=32) |

При подгоне модели индикатор выполнения будет суммировать статус каждой эпохи и общий процесс обучения. Это можно упростить до простого отчета о производительности модели за каждую эпоху, установив для аргумента «подробный» значение 2. Весь вывод можно отключить во время обучения, установив «подробный» (*verbose*) на 0.

|  |  |
| --- | --- |
|  | ...  # fit the model  model.fit(X, y, epochs=100, batch\_size=32, verbose=0) |

***Оценка модели***

Оценка модели требует, чтобы вы сначала выбрали набор данных удержания, используемый для оценки модели. Это должны быть данные, которые не используются в процессе обучения, чтобы мы могли получить объективную оценку производительности модели при прогнозировании новых данных. Проще говоря, разделить данные на тренировочную и тестовую выборку.

Скорость оценки модели пропорциональна количеству данных, которые вы хотите использовать для оценки, это намного быстрее, чем обучение, поскольку модель не изменяется.

С точки зрения API, это включает вызов функции с набором данных удержания и получение потерь и, возможно, других показателей, о которых можно сообщить.

|  |  |
| --- | --- |
|  | ...  # evaluate the model  loss = model.evaluate(X, y, verbose=0) |

***Создание прогнозов.***

Прогнозирование – это последний шаг в жизненном цикле. Для этого модель строилась и обучалась. Это требует, чтобы у вас были новые данные, для которых требуется прогноз, например где у вас нет целевых значений.

С точки зрения API, вы просто вызываете функцию для прогнозирования метки класса, вероятности или числового значения: чего бы вы ни спроектировали в своей модели для прогнозирования.

Вы можете сохранить модель, а затем загрузить ее, чтобы делать прогнозы. Вы также можете подогнать модель ко всем имеющимся данным, прежде чем начнете ее использовать.

Теперь, когда мы знакомы с жизненным циклом модели, давайте рассмотрим два основных способа использования tf.keras API для построения моделей: последовательный и функциональный.

|  |  |
| --- | --- |
|  | ...  # make a prediction  yhat = model.predict(X) |

## 4.2 Sequential Model API (последовательная модель)

API последовательной модели – самый простой и тот API, который я рекомендую, особенно когда вы только начинаете.

Он называется «последовательным», потому что он включает определение класса Sequential и добавление слоев к модели один за другим линейным образом, от входа к выходу.

В приведенном ниже примере определяется модель последовательного MLP, которая принимает восемь входных данных, имеет один скрытый слой (hidden layer) с 10 узлами, а затем выходной слой (output layer) с одним узлом для прогнозирования числового значения.

|  |  |
| --- | --- |
|  | # пример модели, определенной как последовательная  from tensorflow.keras import Sequential  from tensorflow.keras.layers import Dense  # Определение модели  model = Sequential()  model.add(Dense(10, input\_shape=(8,)))  model.add(Dense(1)) |

Обратите внимание, что видимый слой сети определяется аргументом input\_shape на первом скрытом слое (hidden layer). Это означает, что в приведенном выше примере модель ожидает, что вход для одной выборки будет вектором из восьми чисел.

Последовательный API прост в использовании, потому что вы продолжаете вызывать model.add (), пока не добавите все свои слои. Например, вот глубокий MLP с пятью скрытыми слоями.

|  |  |
| --- | --- |
|  | # пример модели, определенной как последовательная  from tensorflow.keras import Sequential  from tensorflow.keras.layers import Dense  # определение модели  model = Sequential()  model.add(Dense(100, input\_shape=(8,)))  model.add(Dense(80))  model.add(Dense(30))  model.add(Dense(10))  model.add(Dense(5))  model.add(Dense(1)) |

## 4.3 Functional Model API (Функциональная модель)

Функциональная модель API более сложный, но при этом более гибкий. Она включает явное соединение вывода одного слоя со входом другого слоя.

Во-первых, входной слой должен быть определен через класс Input, и должна быть указана форма входного образца. Мы должны сохранить ссылку на входной слой при определении модели.

|  |  |
| --- | --- |
|  | ...  # define the layers  x\_in = Input(shape=(8,)) |

Затем полностью подключенный слой можно подключить к входу путем вызова слоя и передачи входного слоя. Это вернет ссылку на выходное соединение в этом новом слое.

|  |  |
| --- | --- |
|  | ...  x = Dense(10)(x\_in) |

Затем мы можем таким же образом подключить это к выходному слою (output layer).

|  |  |
| --- | --- |
|  | ...  x\_out = Dense(1)(x) |

После подключения мы определяем объект модели и указываем уровни ввода и вывода. Полный пример приведен ниже.

|  |  |
| --- | --- |
|  | # пример функциональной модели  from tensorflow.keras import Model  from tensorflow.keras import Input  from tensorflow.keras.layers import Dense  # определение слоёв  x\_in = Input(shape=(8,))  x = Dense(10)(x\_in)  x\_out = Dense(1)(x)  # define the model  model = Model(inputs=x\_in, outputs=x\_out) |

Таким образом, она позволяет создавать более сложные модели, такие, которые могут иметь несколько входных путей (отдельные векторы), и модели, которые имеют несколько выходных путей (например, слово и число). Функциональный API может доставить массу удовольствия, когда вы к нему привыкнете.

# 5. Программа для предсказания значения на основе обучающей выборки

Всё, что будет написано ниже базируется на сказаном ранее, подробное описание используемы в работе модулей библиотеки приведено в конце, в приложении 1 и в документации для библиотеки: <https://www.tensorflow.org/api_docs/python/tf>. Поэтому максимально подробного описания не будет, а будут некоторые уточнения, связанные с нюансами для конкретного набора данных (dataset) и модели. В процессе исследоввния было выснено, что *рекуррентная нейросеть показала результат намного хуже, чем обычная нейросеть глубокого изучения*. В коде приведены и та и другая сети, но критичных изменений приводить для работы того или иного типа сети не нужно. Внимательным нужно быть только когда мы выбираем каакую создать модель и обучать.

## 5.1. Алгоритм работы программы для предсказания значения

Самое важное и сложное в работе с нейросетями создать хороший, чистый и размеченый датасет для обучения сети, если есть готовый датасет, то это примерно 70% успеха. Создавать сеть без датасета абсолюно бесполезно, так как гиперпараметры сети, принцип предобработки и обратокти индивидуален для каждого типа данных (фото, видео, изображение, текст, числа) и всегда имеют разные размеры как самого датасета, так и элементов внутри него. Поэтому первоначально создадим датасет. Делать это можно и в обычном экселе. Первые варианты датасета опустим, так как они использовались чисто для отладки и перейдём к последней версии.

Ниже приведён фрагмент исходных чисел для датасета (рисунок 5.1.). Где t – температура воздуха, p – плотность воздуха, V – скорость ветра, P – мощность, которую теоритически можно снять с ветроустановки , Angle – угол, S – ометаемая площадь.

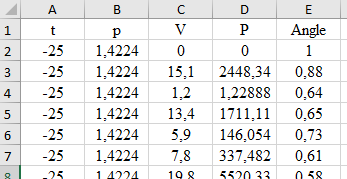


Рисунок 5.1 – Составление датасета (исходные данные)

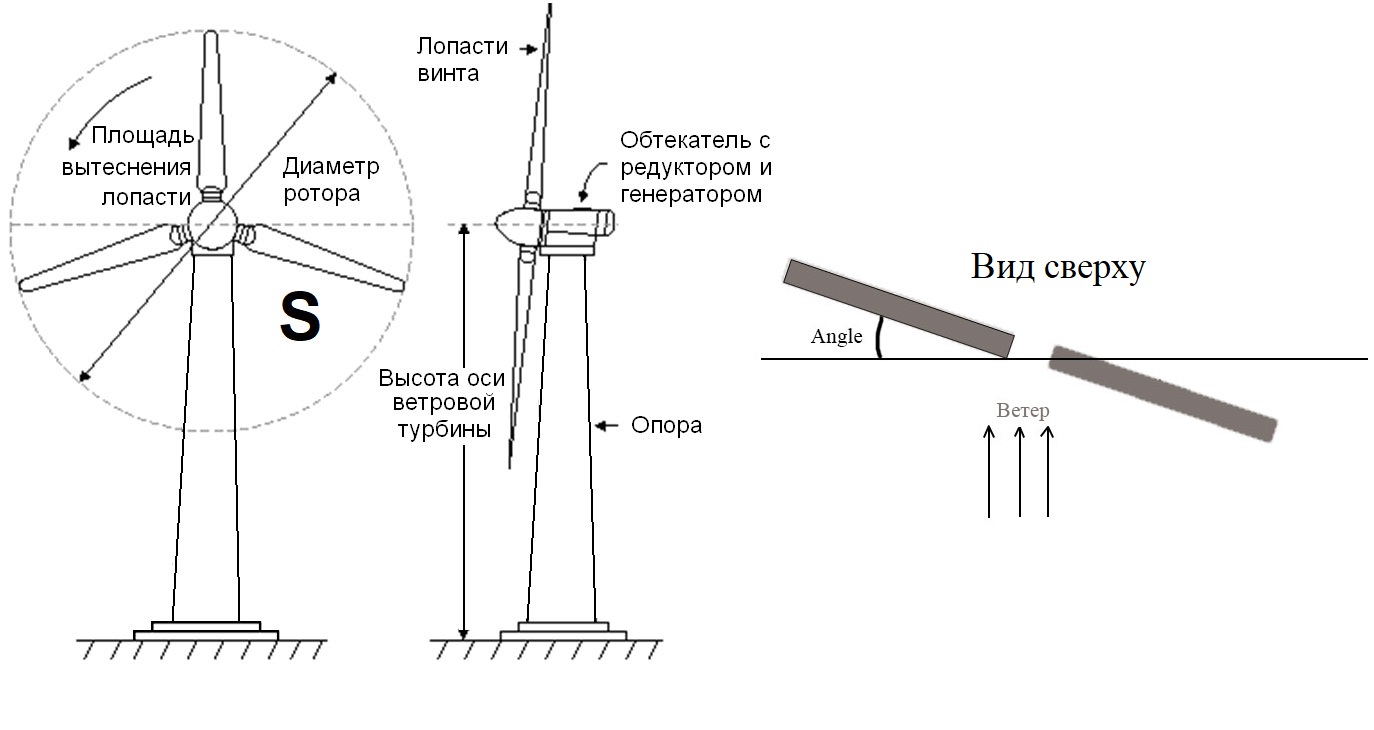


Рисунок 5.2. – Параметры ветроустановки

На основе данных, созданим их смаштабированные значения, относительно их максимальной величины, коэ-ты расчитываются автоматически вот здесь:

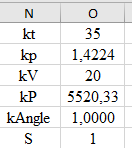


Рисунок 5.4. – Масштабные коэффициенты

И в итоге получается датасет смасштабированных значений, которые будут отправлены в нейросеть.

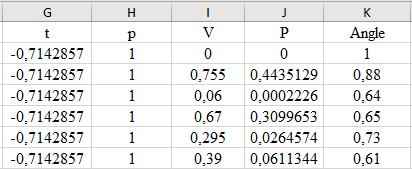


Рисунок 5.5 – Смасштабированные значения

Далее датасет сохраняется в формате csv (рисунок 5.6), потом слегка редактируется в WordPad (рисунок 5.7), запятые как разделитель дробной и целой части меняется на точку, точка с запятой меняется на запятую (это всё происходит из-за того, что эксель русскоязычный) и объвляются заголовки переменных.

После всех манипуляций и загружаю этот датасет себе на гитхаб, чтобы потом в коде обратиться по адресу к нему, можно и просто на Гугл диск скинуть, это не так важно.

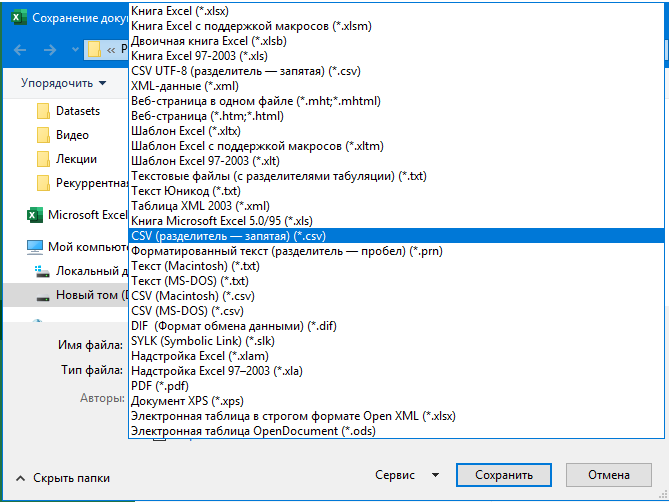


Рисунок 5.6. – Сохранение датасета

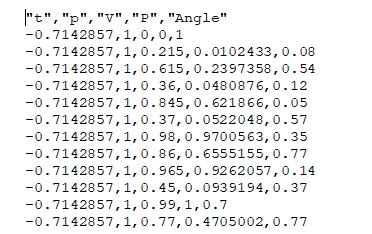
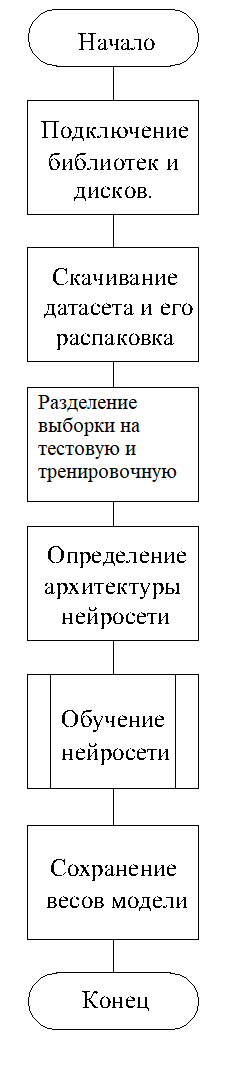


Рисунок 5.7. – Отредактированный датасет

Алгоритм работы программы представлен ниже. Подробно останавливаться не буду, так работа сети была описана выше, в теоритической части.

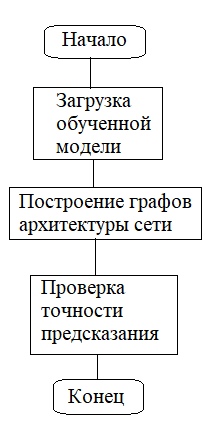


Рисунок 5.8 – Алгоритм работы программы

## 5.2. Код и его объяснение

Подключение гугл диска, определение номера итерации и подстройка готовой сети, а также внесение масштабных коэ-в из экселя.

from google.colab import drive

drive.mount('/content/drive')

number\_of\_interations = 8

Adjusting\_the\_prediction\_result = 0.02 # Подстройка результата предсказания, спользуется для более точного подгона нейросети

kt = 35 #Поправочный коэ-т для температуры

kp = 1.4224 #Поправочный коэ-т для плотности воздуха при нормальных условиях

kV = 20 #Поправочный коэ-т для скорости ветра

kP = 5265.19  #Поправочный коэ-т для мощности

kAngle = 1 #Поправочный коэ-т для угла поворота

number\_of\_param = 4 #Количество аргументов для функции (переменных)

print('Текущий номер для сохранения модели, чекпоинтов, а также загрузки существующей модели с облака: ', number\_of\_interations)

Додавим несколько библиотек и модулей из этих библиотек

from pandas import read\_csv

import numpy as np

import math

import matplotlib.pyplot as plt

from sklearn.metrics import mean\_squared\_error, accuracy\_score

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from keras.models import Sequential

from keras.layers import Dense, SimpleRNN, LSTM, Dropout, Masking, Embedding

from sklearn.preprocessing import MinMaxScaler

from numpy import vstack

from numpy import hstack

from numpy import mean

from keras.models import load\_model

Напишем функцию, которая будет создавать пустую модель рекуррентной нейросети. Делать она это будет только при вызове и передаче ей аргументов, этом момент наступает, когда происходит обучение сети.

def create\_RNN(hidden\_units, dense\_units, input\_shape, activation):

    model = Sequential()

    model.add(SimpleRNN(hidden\_units, input\_shape=input\_shape, activation = activation[0]))

    model.add(Dense(units=hidden\_units, activation=activation[1]))

    model.add(Dense(units=dense\_units, activation=activation[1]))

    # Dropout for regularization

    #model.add(Dropout(0.5))

    model.compile(loss='mean\_squared\_error', optimizer="sgd", metrics=["accuracy"])

    return model

demo\_model = create\_RNN(128, 1, (3,1), activation=['relu', 'softmax']) # Пример создания модели

Загрузим датасет с гитхаба и разделим данные на тестовую выборку и тренировочную. Соотношение 85% – тренировка, 15% – для теста.

# Parameter split\_percent defines the ratio of training examples 85% - тренировочная выборка

def get\_train\_test(url, split\_percent=0.85):

    df = read\_csv(url, usecols=[1], engine='python')

    data = np.array(df.values.astype('float32'))

    scaler = MinMaxScaler(feature\_range=(0, 1))

    data = scaler.fit\_transform(data).flatten()

    n = len(data)

    # Point for splitting data into train and test

    split = int(n\*split\_percent)

    train\_data = data[range(split)]

    test\_data = data[split:]

    return train\_data, test\_data, data

dataset\_url = 'https://raw.githubusercontent.com/AnLiMan/RNN/main/Datasets/WindDatasetGen4.csv'

train\_data, test\_data, data = get\_train\_test(dataset\_url)

Создадим массив с метками из этих данных.

# Подготовка входных данных X и целевых Y

def get\_XY(dat, time\_steps):

    # Индексы целевого массива

    Y\_ind = np.arange(time\_steps, len(dat), time\_steps)

    Y = dat[Y\_ind]

    # Подготовка X

    rows\_x = len(Y)

    X = dat[range(time\_steps\*rows\_x)]

    X = np.reshape(X, (rows\_x, time\_steps, 1))

    return X, Y

time\_steps = number\_of\_param #Количество входных данных для X

trainX, trainY = get\_XY(train\_data, time\_steps)

testX, testY = get\_XY(test\_data, time\_steps)

Далее идёт код с обучением рекуррентной нейросети. Ещё раз. Мы выбираем тип сети и проводим обучение, то есть либо рекуррентная, либо обычная нейросеть. Модель, которая получится в итоге и будет использоваться для тестов. Ячейки не запускаются одна за другой, а выбирается только одна конкретная. В конце модель сохраняется на гугл Диск.

model = create\_RNN(hidden\_units=256, dense\_units=1, input\_shape=(time\_steps, 1), activation=['relu', 'tanh'])

model.fit(trainX, trainY, epochs=20, batch\_size=1, verbose=2, shuffle = True)

model.summary()

# Сохраним всю модель в  HDF5 файл

model.save("/content/drive/My Drive/Files for colab/RNN Models/RNN\_Model" + str(number\_of\_interations) + ".h5")

В этом блоке кода проходи обучение обычной нейросети глубокого изучения. В конце модель сохраняется на гугл Диск.

model = tf.keras.models.Sequential([

  tf.keras.layers.Flatten(input\_shape=(number\_of\_param, 1)),

  tf.keras.layers.Dense(512, activation='linear'),

  tf.keras.layers.Dense(1024, activation='linear'),

  tf.keras.layers.Dropout(0.8),

  tf.keras.layers.Dense(1)])

model.compile(loss='mean\_squared\_error', optimizer="sgd", metrics=['accuracy'])

model.fit(trainX, trainY, epochs=20, batch\_size=2, verbose=2, shuffle = True)

model.summary()

# Сохраним всю модель в  HDF5 файл

model.save("/content/drive/My Drive/Files for colab/RNN Models/RNN\_Model" + str(number\_of\_interations) + ".h5")

Загрузка обученной модели

# Восстановим в точности ту же модель, включая веса и оптимизатор

Loading\_model = keras.models.load\_model("/content/drive/My Drive/Files for colab/RNN Models/RNN\_Model" + str(number\_of\_interations) + ".h5")

Loading\_model.summary()

Начертим загруженную модель в виде графа (рисунок 5.9). Из архитектуры видно, что есть входные значения (flatten input) для входного слоя (flatten), которые передаются на два скрытых слоя (dence\_2, dence\_3), потом на слой исключения – dropout (Слой исключения случайным образом устанавливает для входных единиц значение 0 с частотой на каждом шаге во время обучения, что помогает предотвратить переобучение. Входы, для которых не установлено значение 0, масштабируются на 1 / (1 - скорость), так что сумма по всем входам не изменяется), а в конце на выходной слой (dence\_4)

keras.utils.plot\_model(Loading\_model, 'RNN\_model.png')

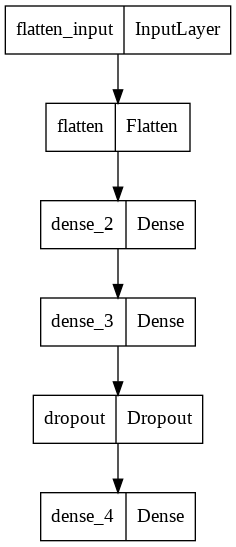


Рисунок 5.9 – Архитектура сети с глубоким изучением

Выведем размерности входа и выхода каждого слоя на построенном графе (рисунок 5.10).

keras.utils.plot\_model(model, 'model\_with\_shape\_info.png', show\_shapes=True)

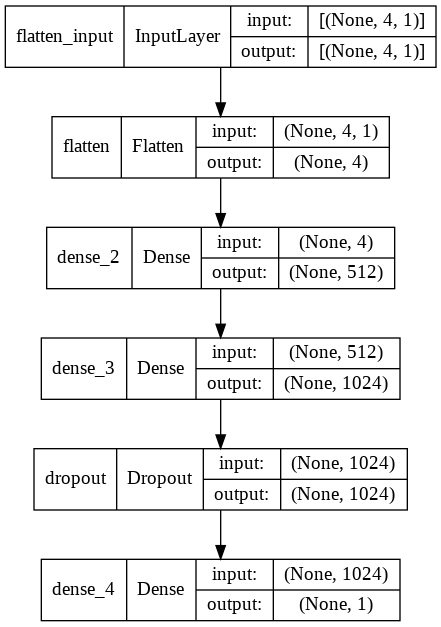


Рисунок 5.10 – Более подробная архитектура сети

## 5.3.1. Результаты обучения и предсказание значений

Проверка точности предсказания

def print\_error(trainY, testY, train\_predict, test\_predict):

    # Ошибка предсказания

    train\_rmse = math.sqrt(mean\_squared\_error(trainY, train\_predict))

    test\_rmse = math.sqrt(mean\_squared\_error(testY, test\_predict))

    # Печать значений

    print('RMSE для тренировочной выборки: %.3f RMSE' % (train\_rmse))

    print('RMSE для тестовой выборки: %.3f RMSE' % (test\_rmse))

# Сделаем предсказания

train\_predict = Loading\_model.predict(trainX) + Adjusting\_the\_prediction\_result

test\_predict = Loading\_model.predict(testX) + Adjusting\_the\_prediction\_result

# Среднеквадратичное отклонение

print\_error(trainY, testY, train\_predict, test\_predict)

Результаты выполнения блока выше для обычной нейросети

RMSE для тренировочной выборки: 0.016 RMSE

RMSE для тестовой выборки: 0.046 RMSE

Результаты выполнения блока выше для рекуррентной нейросети



Построим графики.

# График конечного результата

def plot\_result(trainY, testY, train\_predict, test\_predict):

    actual = np.append(trainY, testY)

    predictions = np.append(train\_predict, test\_predict)

    rows = len(actual)

    plt.figure(figsize=(15, 6), dpi=90)

    plt.plot(range(rows), actual)

    plt.plot(range(rows), predictions)

    plt.axvline(x=len(trainY), color='r')

    plt.legend(['Реальное', 'Предсказанное'])

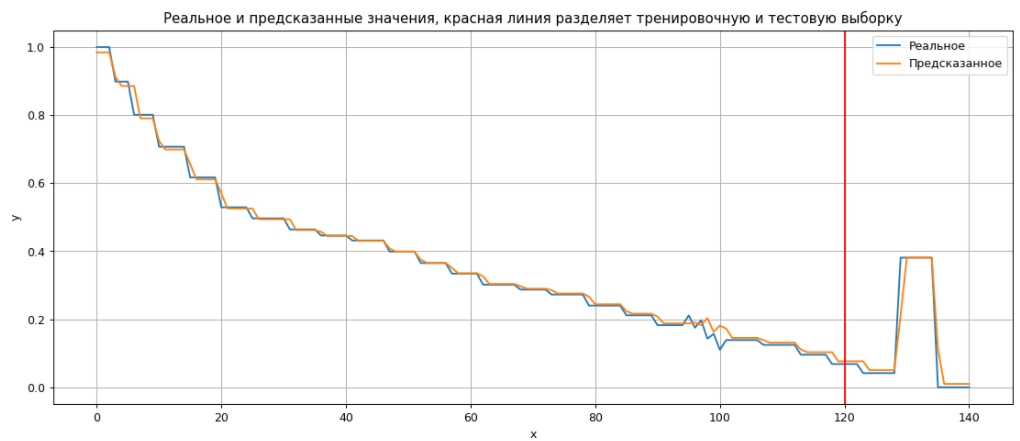
    plt.xlabel('x')

    plt.ylabel('y')

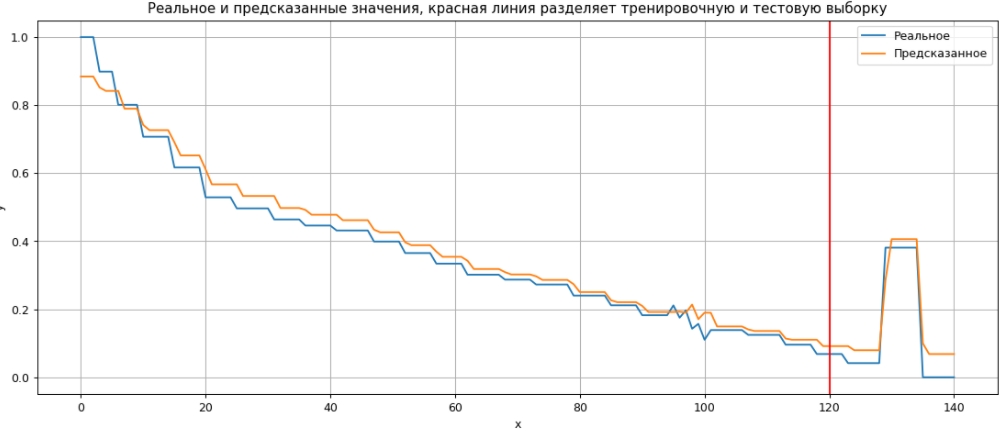
    plt.title('Реальное и предсказанные значения, красная линия разделяет тренировочную и тестовую выборку')

    plt.grid()

plot\_result(trainY, testY, train\_predict, test\_predict)



Предсказания обучающей и тестовой выборки нейросети с глубоким изучением



Предсказания обучающей и тестовой выборки рекуррентной нейросети

График только для тестовой выборки

# График конечного результата

def plot\_result(testY , test\_predict):

    plt.figure(figsize=(15, 6), dpi=90)

    plt.plot(testY)

    plt.plot(test\_predict)

    plt.legend(['Реальное', 'Предсказанное'])

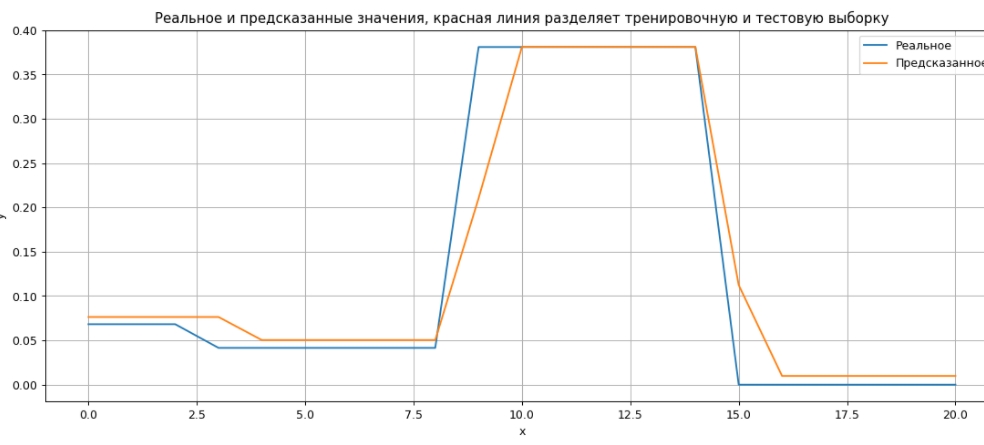
    plt.xlabel('x')

    plt.ylabel('y')

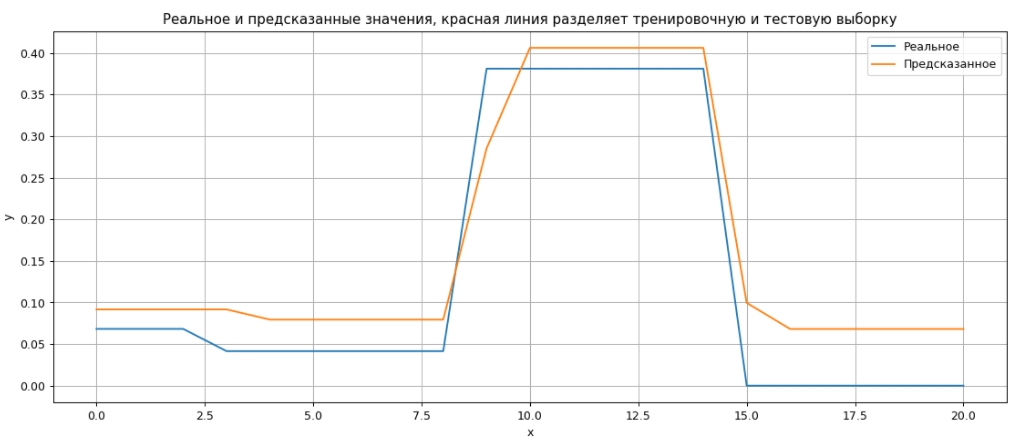
    plt.title('Реальное и предсказанные значения, красная линия разделяет тренировочную и тестовую выборку')

    plt.grid()

plot\_result(testY, test\_predict)



Предсказания тестовой выборки нейросети с глубоким изучением



Предсказания тестовой выборки рекуррентной нейросети

Проверка предсказания. Код для предсказания угла на основе 4-х несмаштабированных переменных

print ("Введите переменные")

list\_of\_value = []

for i in range (0, 4):

  value = float(input())

  list\_of\_value.append(value)

print("Для значений: " + str(list\_of\_value))

data\_for\_predict =[[[list\_of\_value[0]/kt],

  [list\_of\_value[1]/kp],

  [list\_of\_value[2]/kV],

  [list\_of\_value[3]/kP]]]

data\_predict = Loading\_model.predict(data\_for\_predict) + Adjusting\_the\_prediction\_result

Angle = data\_predict[0] \* kAngle

print("Получим: ")

print("Angle = " + str (Angle) + " градусов")

print("Расчётная мощность: ")

print(list\_of\_value[1]\*list\_of\_value[2]\*list\_of\_value[2]\*list\_of\_value[2]/2\*math.cos(Angle/57.2958))

Проверка предсказания. Код для предсказания угла на основе 4-х смаштабированных переменных на нём и проведём проверку. Возьмём следующий массив значений: -0.7142857, 1.0, 0.085, 0.000633, результат расчёта должен быть равен: 0.1. На деле для нейросети с глубоким изучением получили 0,131, что неплохо для предсказания значения из выборки, не основанной на реальных значениях и носящих случайный характер. Для рекуррентной нейросети получили 0.202, что очень далеко от нужного значения. Тесты проводились и на других числах и результаты всегда похожи – сеть глубоко обучения делает предсказания близко или точно в соответствии, рекуррентная ошибается всегда и всегда очень сильно.

print ("Введите переменные")

list\_of\_value = []

for i in range (0, 4):

  value = float(input())

  list\_of\_value.append(value)

print("Для значений: " + str(list\_of\_value))

data\_for\_predict =[[[list\_of\_value[0]],

  [list\_of\_value[1]],

  [list\_of\_value[2]],

  [list\_of\_value[3]]]]

data\_predict = Loading\_model.predict(data\_for\_predict) + Adjusting\_the\_prediction\_result

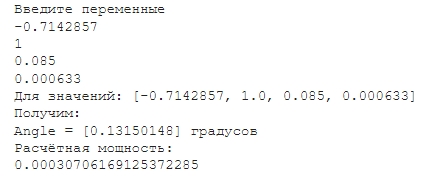
Angle = data\_predict[0] \* kAngle

print("Получим: ")

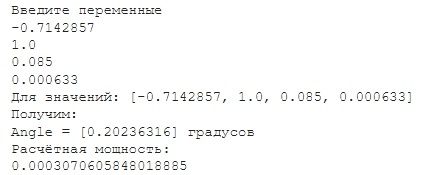
print("Angle = " + str (Angle) + " градусов")

print("Расчётная мощность: ")

print(list\_of\_value[1]\*list\_of\_value[2]\*list\_of\_value[2]\*list\_of\_value[2]/2\*math.cos(Angle/57.2958))



Результат предсказания нейросети с глубоким изучением



Результат предсказания рекуррентной нейросети

## 5.3.2. Реализация обновления обученной модели на основе только новых данных

Далее идёт реализация обновления сети, сначала загрузим новые данные (в нашем случае те же самые) и также разделим на тестовую и тренировочную выборки.

# Parameter split\_percent defines the ratio of training examples 80% - тренировочная выборка

def get\_train\_test(url, split\_percent = 0.8):

    df = read\_csv(url, usecols=[1], engine='python')

    data = np.array(df.values.astype('float32'))

    scaler = MinMaxScaler(feature\_range=(0, 1))

    data = scaler.fit\_transform(data).flatten()

    n = len(data)

    # Point for splitting data into train and test

    split = int(n\*split\_percent)

    train\_data = data[range(split)]

    test\_data = data[split:]

    return train\_data, test\_data, data

dataset\_url = 'https://raw.githubusercontent.com/AnLiMan/RNN/main/Datasets/WindDatasetGen4.csv'

new\_train\_data, new\_test\_data, new\_data = get\_train\_test(dataset\_url)

# Подготовка входных данных X и целевых Y

def get\_XY(dat, time\_steps):

    # Indices of target array

    Y\_ind = np.arange(time\_steps, len(dat), time\_steps)

    Y = dat[Y\_ind]

    # Prepare X

    rows\_x = len(Y)

    X = dat[range(time\_steps\*rows\_x)]

    X = np.reshape(X, (rows\_x, time\_steps, 1))

    return X, Y

time\_steps = number\_of\_param #Количество входных данных для X

new\_trainX, new\_trainY = get\_XY(new\_train\_data, time\_steps)

new\_testX, new\_testY = get\_XY(new\_test\_data, time\_steps)

Блок кода для обновления модели рекуррентной нейросети

# compile the model

Loading\_model.compile(loss='mean\_squared\_error', optimizer = 'sgd', metrics=["accuracy"])

# fit the model on new data

Loading\_model.fit(new\_trainX, new\_trainY, epochs=20, batch\_size=1, verbose=2, shuffle = True)

Loading\_model.summary()

# Сохраним всю модель в  HDF5 файл

#Loading\_model.save("/content/drive/My Drive/Files for colab/RNN\_Model\_Updated" + str(number\_of\_interations) + ".h5")

Блок кода для обновления модели нейросети глубокого изучения.

# compile the model

Loading\_model.compile(loss='mean\_squared\_error', optimizer = 'sgd', metrics=["accuracy"])

# fit the model on new data

Loading\_model.fit(trainX, trainY, epochs=20, batch\_size=2, verbose=1, shuffle = True)

Loading\_model.summary()

Тест. Проводился также как, поменялась только модель и данные для теста.

def print\_error(new\_trainY, new\_testY, new\_train\_predict, new\_test\_predict):

    # Ошибка предсказания

    train\_rmse = math.sqrt(mean\_squared\_error(new\_trainY, new\_train\_predict))

    test\_rmse = math.sqrt(mean\_squared\_error(new\_testY, new\_test\_predict))

    # Печать отклонения

    print('RMSE для тренировочной выборки: %.3f RMSE' % (train\_rmse))

    print('RMSE для тестовой выборки: %.3f RMSE' % (test\_rmse))

# Сделаем предсказания

new\_train\_predict = Loading\_model.predict(new\_trainX)

new\_test\_predict = Loading\_model.predict(new\_testX)

# Среднеквадратичное отклонение

print\_error(new\_trainY, new\_testY, new\_train\_predict, new\_test\_predict)

Для нейросети глубоко обучения получили:

RMSE для тренировочной выборки: 0.024 RMSE

RMSE для тестовой выборки: 0.042 RMSE

Для рекуррентной:

RMSE для тренировочной выборки: 0.027 RMSE

RMSE для тестовой выборки: 0.038 RMS

# График готового результата

def plot\_result(new\_trainY, new\_testY, new\_train\_predict, new\_test\_predict):

    actual = np.append(new\_trainY, new\_testY)

    predictions = np.append(new\_train\_predict, new\_test\_predict)

    rows = len(actual)

    plt.figure(figsize=(15, 6), dpi=90)

    plt.plot(range(rows), actual)

    plt.plot(range(rows), predictions)

    plt.axvline(x = len(new\_trainY), color = 'r')

    plt.legend(['Реальное', 'Предсказанное'])

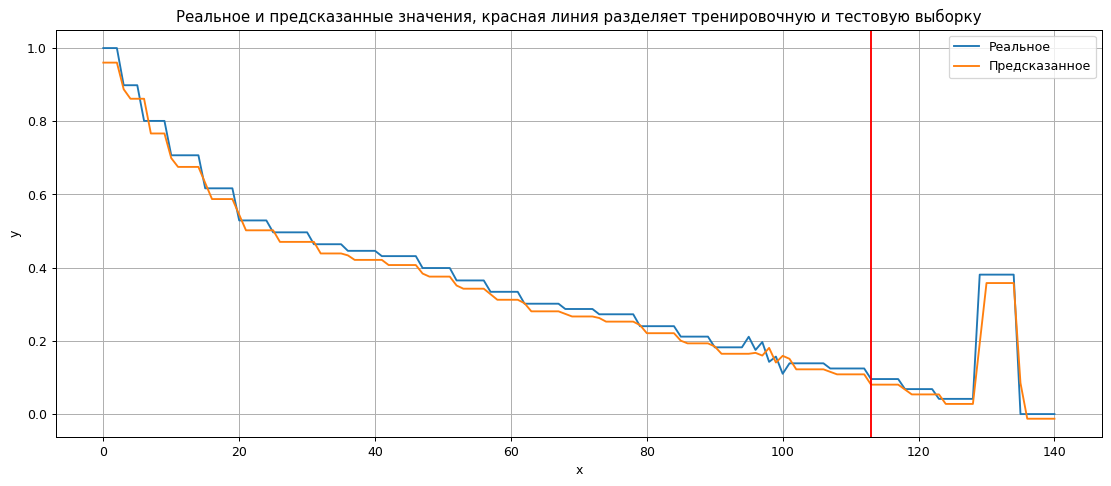
    plt.xlabel('x')

    plt.ylabel('y')

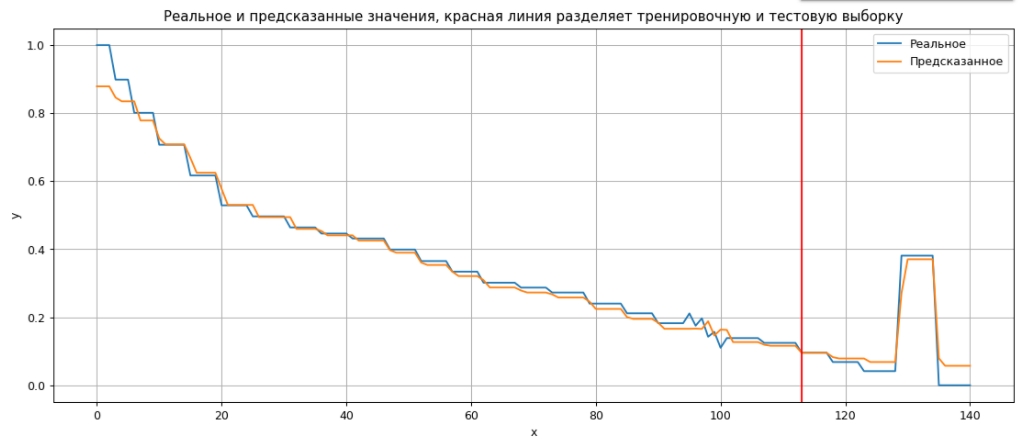
    plt.title('Реальное и предсказанные значения, красная линия разделяет тренировочную и тестовую выборку')

    plt.grid()

plot\_result(new\_trainY, new\_testY, new\_train\_predict, new\_test\_predict)



Предсказания обучающей и тестовой выборки нейросети с глубоким изучением



Предсказания обучающей и тестовой выборки рекуррентной нейросети

График только для тестовой выборки

# График конечного результата

def plot\_result(new\_testY , new\_test\_predict):

    plt.figure(figsize=(15, 6), dpi=90)

    plt.plot(testY)

    plt.plot(test\_predict)

    plt.legend(['Реальное', 'Предсказанное'])

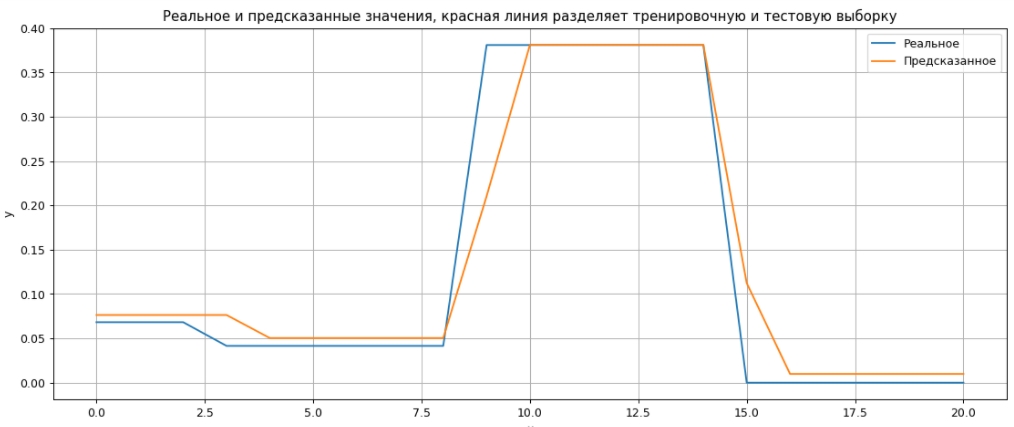
    plt.xlabel('x')

    plt.ylabel('y')

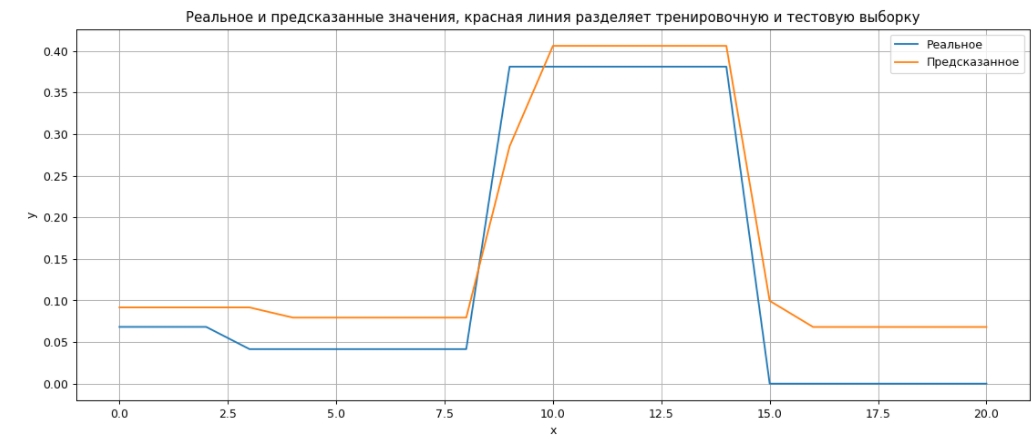
    plt.title('Реальное и предсказанные значения, красная линия разделяет тренировочную и тестовую выборку')

    plt.grid()

plot\_result(new\_testY, new\_test\_predict)



Предсказания тестовой выборки нейросети с глубоким изучением



Предсказания тестовой выборки рекуррентной нейросети

## 5.3.3. Обновление обученной модели на основе старых и новых данных

Можно объединить старые данные и новые. Но данный метод имеет очевидные недостатки – экспоненциальное увеличение времени обучения с каждой итерацией, а также переобучение модели при достижении определенного количества данных, поэтому данный вариант следует применять крайне осторожно. Код также присутствует и критическое отличие от обновления модели заключается в этих блоках кода:

# create a composite dataset of old and new data

X\_both, y\_both = vstack((trainX, new\_trainX)), hstack((trainY, new\_trainY))

# fit the model on old data

Loading\_model.fit(X\_both, y\_both, epochs=150, batch\_size=32, verbose=2)

# make predictions with both models

yhat1 = Loading\_model.predict(new\_testX)

yhat2 = model.predict(new\_testX)

# combine predictions into single array

combined = hstack((yhat1, yhat2))

# calculate outcome as mean of predictions

yhat = mean(combined, axis=-1)

Сначала создаем модель на новых и старых данных, потом объединяем её предсказания со старой, предсказания усредняются и выводится результат.

# Приложение 1. Дополнительная информация о некоторых модулях из библиотеки tensorflow

## 1. Module: tf.keras.optimizers

Public API for tf.keras.optimizers namespace.

Modules

[schedules](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/schedules) module: Public API for tf.keras.optimizers.schedules namespace.

Classes

[class Adadelta](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adadelta): Optimizer that implements the Adadelta algorithm.

[class Adagrad](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adagrad): Optimizer that implements the Adagrad algorithm.

[class Adam](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam): Optimizer that implements the Adam algorithm.

[class Adamax](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adamax): Optimizer that implements the Adamax algorithm.

[class Ftrl](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Ftrl): Optimizer that implements the FTRL algorithm.

[class Nadam](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Nadam): Optimizer that implements the NAdam algorithm.

[class Optimizer](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Optimizer): Base class for Keras optimizers.

[class RMSprop](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/RMSprop): Optimizer that implements the RMSprop algorithm.

[class SGD](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD): Gradient descent (with momentum) optimizer.

Functions

[deserialize(...)](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/deserialize): Inverse of the serialize function.

[get(...)](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/get): Retrieves a Keras Optimizer instance.

[serialize(...)](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/serialize): Serialize the optimizer configuration to JSON compatible python dict.

## 2. Module: tf.keras.losses

Public API for tf.keras.losses namespace.

Classes

[class BinaryCrossentropy](https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy): Computes the cross-entropy loss between true labels and predicted labels.

[class CategoricalCrossentropy](https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy): Computes the crossentropy loss between the labels and predictions.

[class CategoricalHinge](https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalHinge): Computes the categorical hinge loss between y\_true and y\_pred.

[class CosineSimilarity](https://www.tensorflow.org/api_docs/python/tf/keras/losses/CosineSimilarity): Computes the cosine similarity between labels and predictions.

[class Hinge](https://www.tensorflow.org/api_docs/python/tf/keras/losses/Hinge): Computes the hinge loss between y\_true and y\_pred.

[class Huber](https://www.tensorflow.org/api_docs/python/tf/keras/losses/Huber): Computes the Huber loss between y\_true and y\_pred.

[class KLDivergence](https://www.tensorflow.org/api_docs/python/tf/keras/losses/KLDivergence): Computes Kullback-Leibler divergence loss between y\_true and y\_pred.

[class LogCosh](https://www.tensorflow.org/api_docs/python/tf/keras/losses/LogCosh): Computes the logarithm of the hyperbolic cosine of the prediction error.

[class Loss](https://www.tensorflow.org/api_docs/python/tf/keras/losses/Loss): Loss base class.

[class MeanAbsoluteError](https://www.tensorflow.org/api_docs/python/tf/keras/losses/MeanAbsoluteError): Computes the mean of absolute difference between labels and predictions.

[class MeanAbsolutePercentageError](https://www.tensorflow.org/api_docs/python/tf/keras/losses/MeanAbsolutePercentageError): Computes the mean absolute percentage error between y\_true and y\_pred.

[class MeanSquaredError](https://www.tensorflow.org/api_docs/python/tf/keras/losses/MeanSquaredError): Computes the mean of squares of errors between labels and predictions.

[class MeanSquaredLogarithmicError](https://www.tensorflow.org/api_docs/python/tf/keras/losses/MeanSquaredLogarithmicError): Computes the mean squared logarithmic error between y\_true and y\_pred.

[class Poisson](https://www.tensorflow.org/api_docs/python/tf/keras/losses/Poisson): Computes the Poisson loss between y\_true and y\_pred.

[class Reduction](https://www.tensorflow.org/api_docs/python/tf/keras/losses/Reduction): Types of loss reduction.

[class SparseCategoricalCrossentropy](https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy): Computes the crossentropy loss between the labels and predictions.

[class SquaredHinge](https://www.tensorflow.org/api_docs/python/tf/keras/losses/SquaredHinge): Computes the squared hinge loss between y\_true and y\_pred.

Functions

[KLD(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/kl_divergence): Computes Kullback-Leibler divergence loss between y\_true and y\_pred.

[MAE(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_absolute_error): Computes the mean absolute error between labels and predictions.

[MAPE(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_absolute_percentage_error): Computes the mean absolute percentage error between y\_true and y\_pred.

[MSE(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_squared_error): Computes the mean squared error between labels and predictions.

[MSLE(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_squared_logarithmic_error): Computes the mean squared logarithmic error between y\_true and y\_pred.

[binary\_crossentropy(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/binary_crossentropy): Computes the binary crossentropy loss.

[categorical\_crossentropy(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/categorical_crossentropy): Computes the categorical crossentropy loss.

[categorical\_hinge(...)](https://www.tensorflow.org/api_docs/python/tf/keras/losses/categorical_hinge): Computes the categorical hinge loss between y\_true and y\_pred.

[cosine\_similarity(...)](https://www.tensorflow.org/api_docs/python/tf/keras/losses/cosine_similarity): Computes the cosine similarity between labels and predictions.

[deserialize(...)](https://www.tensorflow.org/api_docs/python/tf/keras/losses/deserialize): Deserializes a serialized loss class/function instance.

[get(...)](https://www.tensorflow.org/api_docs/python/tf/keras/losses/get): Retrieves a Keras loss as a function/Loss class instance.

[hinge(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/hinge): Computes the hinge loss between y\_true and y\_pred.

[huber(...)](https://www.tensorflow.org/api_docs/python/tf/keras/losses/huber): Computes Huber loss value.

[kl\_divergence(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/kl_divergence): Computes Kullback-Leibler divergence loss between y\_true and y\_pred.

[kld(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/kl_divergence): Computes Kullback-Leibler divergence loss between y\_true and y\_pred.

[kullback\_leibler\_divergence(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/kl_divergence): Computes Kullback-Leibler divergence loss between y\_true and y\_pred.

[log\_cosh(...)](https://www.tensorflow.org/api_docs/python/tf/keras/losses/log_cosh): Logarithm of the hyperbolic cosine of the prediction error.

[logcosh(...)](https://www.tensorflow.org/api_docs/python/tf/keras/losses/log_cosh): Logarithm of the hyperbolic cosine of the prediction error.

[mae(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_absolute_error): Computes the mean absolute error between labels and predictions.

[mape(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_absolute_percentage_error): Computes the mean absolute percentage error between y\_true and y\_pred.

[mean\_absolute\_error(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_absolute_error): Computes the mean absolute error between labels and predictions.

[mean\_absolute\_percentage\_error(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_absolute_percentage_error): Computes the mean absolute percentage error between y\_true and y\_pred.

[mean\_squared\_error(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_squared_error): Computes the mean squared error between labels and predictions.

[mean\_squared\_logarithmic\_error(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_squared_logarithmic_error): Computes the mean squared logarithmic error between y\_true and y\_pred.

[mse(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_squared_error): Computes the mean squared error between labels and predictions.

[msle(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_squared_logarithmic_error): Computes the mean squared logarithmic error between y\_true and y\_pred.

[poisson(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/poisson): Computes the Poisson loss between y\_true and y\_pred.

[serialize(...)](https://www.tensorflow.org/api_docs/python/tf/keras/losses/serialize): Serializes loss function or Loss instance.

[sparse\_categorical\_crossentropy(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/sparse_categorical_crossentropy): Computes the sparse categorical crossentropy loss.

[squared\_hinge(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/squared_hinge): Computes the squared hinge loss between y\_true and y\_pred.

## 3. Module: tf.keras.metrics

Public API for tf.keras.metrics namespace.

Classes

[class AUC](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/AUC): Approximates the AUC (Area under the curve) of the ROC or PR curves.

[class Accuracy](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Accuracy): Calculates how often predictions equal labels.

[class BinaryAccuracy](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/BinaryAccuracy): Calculates how often predictions match binary labels.

[class BinaryCrossentropy](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/BinaryCrossentropy): Computes the crossentropy metric between the labels and predictions.

[class CategoricalAccuracy](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/CategoricalAccuracy): Calculates how often predictions match one-hot labels.

[class CategoricalCrossentropy](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/CategoricalCrossentropy): Computes the crossentropy metric between the labels and predictions.

[class CategoricalHinge](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/CategoricalHinge): Computes the categorical hinge metric between y\_true and y\_pred.

[class CosineSimilarity](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/CosineSimilarity): Computes the cosine similarity between the labels and predictions.

[class FalseNegatives](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/FalseNegatives): Calculates the number of false negatives.

[class FalsePositives](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/FalsePositives): Calculates the number of false positives.

[class Hinge](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Hinge): Computes the hinge metric between y\_true and y\_pred.

[class KLDivergence](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/KLDivergence): Computes Kullback-Leibler divergence metric between y\_true and y\_pred.

[class LogCoshError](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/LogCoshError): Computes the logarithm of the hyperbolic cosine of the prediction error.

[class Mean](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Mean): Computes the (weighted) mean of the given values.

[class MeanAbsoluteError](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/MeanAbsoluteError): Computes the mean absolute error between the labels and predictions.

[class MeanAbsolutePercentageError](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/MeanAbsolutePercentageError): Computes the mean absolute percentage error between y\_true and y\_pred.

[class MeanIoU](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/MeanIoU): Computes the mean Intersection-Over-Union metric.

[class MeanMetricWrapper](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/MeanMetricWrapper): Wraps a stateless metric function with the Mean metric.

[class MeanRelativeError](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/MeanRelativeError): Computes the mean relative error by normalizing with the given values.

[class MeanSquaredError](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/MeanSquaredError): Computes the mean squared error between y\_true and y\_pred.

[class MeanSquaredLogarithmicError](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/MeanSquaredLogarithmicError): Computes the mean squared logarithmic error between y\_true and y\_pred.

[class MeanTensor](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/MeanTensor): Computes the element-wise (weighted) mean of the given tensors.

[class Metric](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Metric): Encapsulates metric logic and state.

[class Poisson](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Poisson): Computes the Poisson metric between y\_true and y\_pred.

[class Precision](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Precision): Computes the precision of the predictions with respect to the labels.

[class PrecisionAtRecall](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/PrecisionAtRecall): Computes best precision where recall is >= specified value.

[class Recall](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Recall): Computes the recall of the predictions with respect to the labels.

[class RecallAtPrecision](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/RecallAtPrecision): Computes best recall where precision is >= specified value.

[class RootMeanSquaredError](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/RootMeanSquaredError): Computes root mean squared error metric between y\_true and y\_pred.

[class SensitivityAtSpecificity](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SensitivityAtSpecificity): Computes best sensitivity where specificity is >= specified value.

[class SparseCategoricalAccuracy](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SparseCategoricalAccuracy): Calculates how often predictions match integer labels.

[class SparseCategoricalCrossentropy](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SparseCategoricalCrossentropy): Computes the crossentropy metric between the labels and predictions.

[class SparseTopKCategoricalAccuracy](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SparseTopKCategoricalAccuracy): Computes how often integer targets are in the top K predictions.

[class SpecificityAtSensitivity](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SpecificityAtSensitivity): Computes best specificity where sensitivity is >= specified value.

[class SquaredHinge](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SquaredHinge): Computes the squared hinge metric between y\_true and y\_pred.

[class Sum](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Sum): Computes the (weighted) sum of the given values.

[class TopKCategoricalAccuracy](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/TopKCategoricalAccuracy): Computes how often targets are in the top K predictions.

[class TrueNegatives](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/TrueNegatives): Calculates the number of true negatives.

[class TruePositives](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/TruePositives): Calculates the number of true positives.

Functions

[KLD(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/kl_divergence): Computes Kullback-Leibler divergence loss between y\_true and y\_pred.

[MAE(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_absolute_error): Computes the mean absolute error between labels and predictions.

[MAPE(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_absolute_percentage_error): Computes the mean absolute percentage error between y\_true and y\_pred.

[MSE(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_squared_error): Computes the mean squared error between labels and predictions.

[MSLE(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_squared_logarithmic_error): Computes the mean squared logarithmic error between y\_true and y\_pred.

[binary\_accuracy(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/binary_accuracy): Calculates how often predictions match binary labels.

[binary\_crossentropy(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/binary_crossentropy): Computes the binary crossentropy loss.

[categorical\_accuracy(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/categorical_accuracy): Calculates how often predictions match one-hot labels.

[categorical\_crossentropy(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/categorical_crossentropy): Computes the categorical crossentropy loss.

[deserialize(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/deserialize): Deserializes a serialized metric class/function instance.

[get(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/get): Retrieves a Keras metric as a function/Metric class instance.

[hinge(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/hinge): Computes the hinge loss between y\_true and y\_pred.

[kl\_divergence(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/kl_divergence): Computes Kullback-Leibler divergence loss between y\_true and y\_pred.

[kld(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/kl_divergence): Computes Kullback-Leibler divergence loss between y\_true and y\_pred.

[kullback\_leibler\_divergence(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/kl_divergence): Computes Kullback-Leibler divergence loss between y\_true and y\_pred.

[log\_cosh(...)](https://www.tensorflow.org/api_docs/python/tf/keras/losses/log_cosh): Logarithm of the hyperbolic cosine of the prediction error.

[logcosh(...)](https://www.tensorflow.org/api_docs/python/tf/keras/losses/log_cosh): Logarithm of the hyperbolic cosine of the prediction error.

[mae(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_absolute_error): Computes the mean absolute error between labels and predictions.

[mape(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_absolute_percentage_error): Computes the mean absolute percentage error between y\_true and y\_pred.

[mean\_absolute\_error(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_absolute_error): Computes the mean absolute error between labels and predictions.

[mean\_absolute\_percentage\_error(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_absolute_percentage_error): Computes the mean absolute percentage error between y\_true and y\_pred.

[mean\_squared\_error(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_squared_error): Computes the mean squared error between labels and predictions.

[mean\_squared\_logarithmic\_error(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_squared_logarithmic_error): Computes the mean squared logarithmic error between y\_true and y\_pred.

[mse(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_squared_error): Computes the mean squared error between labels and predictions.

[msle(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/mean_squared_logarithmic_error): Computes the mean squared logarithmic error between y\_true and y\_pred.

[poisson(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/poisson): Computes the Poisson loss between y\_true and y\_pred.

[serialize(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/serialize): Serializes metric function or Metric instance.

[sparse\_categorical\_accuracy(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/sparse_categorical_accuracy): Calculates how often predictions match integer labels.

[sparse\_categorical\_crossentropy(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/sparse_categorical_crossentropy): Computes the sparse categorical crossentropy loss.

[sparse\_top\_k\_categorical\_accuracy(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/sparse_top_k_categorical_accuracy): Computes how often integer targets are in the top K predictions.

[squared\_hinge(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/squared_hinge): Computes the squared hinge loss between y\_true and y\_pred.

[top\_k\_categorical\_accuracy(...)](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/top_k_categorical_accuracy): Computes how often targets are in the top K predictions.

## 4. Module: tf.keras.layers

Public API for tf.keras.layers namespace.

Modules

[experimental](https://www.tensorflow.org/api_docs/python/tf/keras/layers/experimental) module: Public API for tf.keras.layers.experimental namespace.

Classes

[class AbstractRNNCell](https://www.tensorflow.org/api_docs/python/tf/keras/layers/AbstractRNNCell): Abstract object representing an RNN cell.

[class Activation](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Activation): Applies an activation function to an output.

[class ActivityRegularization](https://www.tensorflow.org/api_docs/python/tf/keras/layers/ActivityRegularization): Layer that applies an update to the cost function based input activity.

[class Add](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Add): Layer that adds a list of inputs.

[class AdditiveAttention](https://www.tensorflow.org/api_docs/python/tf/keras/layers/AdditiveAttention): Additive attention layer, a.k.a. Bahdanau-style attention.

[class AlphaDropout](https://www.tensorflow.org/api_docs/python/tf/keras/layers/AlphaDropout): Applies Alpha Dropout to the input.

[class Attention](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Attention): Dot-product attention layer, a.k.a. Luong-style attention.

[class Average](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Average): Layer that averages a list of inputs element-wise.

[class AveragePooling1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/AveragePooling1D): Average pooling for temporal data.

[class AveragePooling2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/AveragePooling2D): Average pooling operation for spatial data.

[class AveragePooling3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/AveragePooling3D): Average pooling operation for 3D data (spatial or spatio-temporal).

[class AvgPool1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/AveragePooling1D): Average pooling for temporal data.

[class AvgPool2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/AveragePooling2D): Average pooling operation for spatial data.

[class AvgPool3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/AveragePooling3D): Average pooling operation for 3D data (spatial or spatio-temporal).

[class BatchNormalization](https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization): Layer that normalizes its inputs.

[class Bidirectional](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional): Bidirectional wrapper for RNNs.

[class CategoryEncoding](https://www.tensorflow.org/api_docs/python/tf/keras/layers/CategoryEncoding): A preprocessing layer which encodes integer features.

[class CenterCrop](https://www.tensorflow.org/api_docs/python/tf/keras/layers/CenterCrop): A preprocessing layer which crops images.

[class Concatenate](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Concatenate): Layer that concatenates a list of inputs.

[class Conv1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1D): 1D convolution layer (e.g. temporal convolution).

[class Conv1DTranspose](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1DTranspose): Transposed convolution layer (sometimes called Deconvolution).

[class Conv2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D): 2D convolution layer (e.g. spatial convolution over images).

[class Conv2DTranspose](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2DTranspose): Transposed convolution layer (sometimes called Deconvolution).

[class Conv3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv3D): 3D convolution layer (e.g. spatial convolution over volumes).

[class Conv3DTranspose](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv3DTranspose): Transposed convolution layer (sometimes called Deconvolution).

[class ConvLSTM1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/ConvLSTM1D): 1D Convolutional LSTM.

[class ConvLSTM2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/ConvLSTM2D): 2D Convolutional LSTM.

[class ConvLSTM3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/ConvLSTM3D): 3D Convolutional LSTM.

[class Convolution1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1D): 1D convolution layer (e.g. temporal convolution).

[class Convolution1DTranspose](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1DTranspose): Transposed convolution layer (sometimes called Deconvolution).

[class Convolution2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D): 2D convolution layer (e.g. spatial convolution over images).

[class Convolution2DTranspose](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2DTranspose): Transposed convolution layer (sometimes called Deconvolution).

[class Convolution3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv3D): 3D convolution layer (e.g. spatial convolution over volumes).

[class Convolution3DTranspose](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv3DTranspose): Transposed convolution layer (sometimes called Deconvolution).

[class Cropping1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Cropping1D): Cropping layer for 1D input (e.g. temporal sequence).

[class Cropping2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Cropping2D): Cropping layer for 2D input (e.g. picture).

[class Cropping3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Cropping3D): Cropping layer for 3D data (e.g. spatial or spatio-temporal).

[class Dense](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense): Just your regular densely-connected NN layer.

[class DenseFeatures](https://www.tensorflow.org/api_docs/python/tf/keras/layers/DenseFeatures): A layer that produces a dense Tensor based on given feature\_columns.

[class DepthwiseConv1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/DepthwiseConv1D): Depthwise 1D convolution.

[class DepthwiseConv2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/DepthwiseConv2D): Depthwise 2D convolution.

[class Discretization](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Discretization): A preprocessing layer which buckets continuous features by ranges.

[class Dot](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dot): Layer that computes a dot product between samples in two tensors.

[class Dropout](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout): Applies Dropout to the input.

[class ELU](https://www.tensorflow.org/api_docs/python/tf/keras/layers/ELU): Exponential Linear Unit.

[class Embedding](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding): Turns positive integers (indexes) into dense vectors of fixed size.

[class Flatten](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Flatten): Flattens the input. Does not affect the batch size.

[class GRU](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GRU): Gated Recurrent Unit - Cho et al. 2014.

[class GRUCell](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GRUCell): Cell class for the GRU layer.

[class GaussianDropout](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GaussianDropout): Apply multiplicative 1-centered Gaussian noise.

[class GaussianNoise](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GaussianNoise): Apply additive zero-centered Gaussian noise.

[class GlobalAveragePooling1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalAveragePooling1D): Global average pooling operation for temporal data.

[class GlobalAveragePooling2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalAveragePooling2D): Global average pooling operation for spatial data.

[class GlobalAveragePooling3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalAveragePooling3D): Global Average pooling operation for 3D data.

[class GlobalAvgPool1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalAveragePooling1D): Global average pooling operation for temporal data.

[class GlobalAvgPool2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalAveragePooling2D): Global average pooling operation for spatial data.

[class GlobalAvgPool3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalAveragePooling3D): Global Average pooling operation for 3D data.

[class GlobalMaxPool1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalMaxPool1D): Global max pooling operation for 1D temporal data.

[class GlobalMaxPool2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalMaxPool2D): Global max pooling operation for spatial data.

[class GlobalMaxPool3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalMaxPool3D): Global Max pooling operation for 3D data.

[class GlobalMaxPooling1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalMaxPool1D): Global max pooling operation for 1D temporal data.

[class GlobalMaxPooling2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalMaxPool2D): Global max pooling operation for spatial data.

[class GlobalMaxPooling3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalMaxPool3D): Global Max pooling operation for 3D data.

[class Hashing](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Hashing): A preprocessing layer which hashes and bins categorical features.

[class InputLayer](https://www.tensorflow.org/api_docs/python/tf/keras/layers/InputLayer): Layer to be used as an entry point into a Network (a graph of layers).

[class InputSpec](https://www.tensorflow.org/api_docs/python/tf/keras/layers/InputSpec): Specifies the rank, dtype and shape of every input to a layer.

[class IntegerLookup](https://www.tensorflow.org/api_docs/python/tf/keras/layers/IntegerLookup): A preprocessing layer which maps integer features to contiguous ranges.

[class LSTM](https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM): Long Short-Term Memory layer - Hochreiter 1997.

[class LSTMCell](https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTMCell): Cell class for the LSTM layer.

[class Lambda](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Lambda): Wraps arbitrary expressions as a Layer object.

[class Layer](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Layer): This is the class from which all layers inherit.

[class LayerNormalization](https://www.tensorflow.org/api_docs/python/tf/keras/layers/LayerNormalization): Layer normalization layer (Ba et al., 2016).

[class LeakyReLU](https://www.tensorflow.org/api_docs/python/tf/keras/layers/LeakyReLU): Leaky version of a Rectified Linear Unit.

[class LocallyConnected1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/LocallyConnected1D): Locally-connected layer for 1D inputs.

[class LocallyConnected2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/LocallyConnected2D): Locally-connected layer for 2D inputs.

[class Masking](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Masking): Masks a sequence by using a mask value to skip timesteps.

[class MaxPool1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool1D): Max pooling operation for 1D temporal data.

[class MaxPool2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D): Max pooling operation for 2D spatial data.

[class MaxPool3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool3D): Max pooling operation for 3D data (spatial or spatio-temporal).

[class MaxPooling1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool1D): Max pooling operation for 1D temporal data.

[class MaxPooling2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D): Max pooling operation for 2D spatial data.

[class MaxPooling3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool3D): Max pooling operation for 3D data (spatial or spatio-temporal).

[class Maximum](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Maximum): Layer that computes the maximum (element-wise) a list of inputs.

[class Minimum](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Minimum): Layer that computes the minimum (element-wise) a list of inputs.

[class MultiHeadAttention](https://www.tensorflow.org/api_docs/python/tf/keras/layers/MultiHeadAttention): MultiHeadAttention layer.

[class Multiply](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Multiply): Layer that multiplies (element-wise) a list of inputs.

[class Normalization](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Normalization): A preprocessing layer which normalizes continuous features.

[class PReLU](https://www.tensorflow.org/api_docs/python/tf/keras/layers/PReLU): Parametric Rectified Linear Unit.

[class Permute](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Permute): Permutes the dimensions of the input according to a given pattern.

[class RNN](https://www.tensorflow.org/api_docs/python/tf/keras/layers/RNN): Base class for recurrent layers.

[class RandomContrast](https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomContrast): A preprocessing layer which randomly adjusts contrast during training.

[class RandomCrop](https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomCrop): A preprocessing layer which randomly crops images during training.

[class RandomFlip](https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomFlip): A preprocessing layer which randomly flips images during training.

[class RandomHeight](https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomHeight): A preprocessing layer which randomly varies image height during training.

[class RandomRotation](https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomRotation): A preprocessing layer which randomly rotates images during training.

[class RandomTranslation](https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomTranslation): A preprocessing layer which randomly translates images during training.

[class RandomWidth](https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomWidth): A preprocessing layer which randomly varies image width during training.

[class RandomZoom](https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomZoom): A preprocessing layer which randomly zooms images during training.

[class ReLU](https://www.tensorflow.org/api_docs/python/tf/keras/layers/ReLU): Rectified Linear Unit activation function.

[class RepeatVector](https://www.tensorflow.org/api_docs/python/tf/keras/layers/RepeatVector): Repeats the input n times.

[class Rescaling](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Rescaling): A preprocessing layer which rescales input values to a new range.

[class Reshape](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Reshape): Layer that reshapes inputs into the given shape.

[class Resizing](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Resizing): A preprocessing layer which resizes images.

[class SeparableConv1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/SeparableConv1D): Depthwise separable 1D convolution.

[class SeparableConv2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/SeparableConv2D): Depthwise separable 2D convolution.

[class SeparableConvolution1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/SeparableConv1D): Depthwise separable 1D convolution.

[class SeparableConvolution2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/SeparableConv2D): Depthwise separable 2D convolution.

[class SimpleRNN](https://www.tensorflow.org/api_docs/python/tf/keras/layers/SimpleRNN): Fully-connected RNN where the output is to be fed back to input.

[class SimpleRNNCell](https://www.tensorflow.org/api_docs/python/tf/keras/layers/SimpleRNNCell): Cell class for SimpleRNN.

[class Softmax](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Softmax): Softmax activation function.

[class SpatialDropout1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/SpatialDropout1D): Spatial 1D version of Dropout.

[class SpatialDropout2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/SpatialDropout2D): Spatial 2D version of Dropout.

[class SpatialDropout3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/SpatialDropout3D): Spatial 3D version of Dropout.

[class StackedRNNCells](https://www.tensorflow.org/api_docs/python/tf/keras/layers/StackedRNNCells): Wrapper allowing a stack of RNN cells to behave as a single cell.

[class StringLookup](https://www.tensorflow.org/api_docs/python/tf/keras/layers/StringLookup): A preprocessing layer which maps string features to integer indices.

[class Subtract](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Subtract): Layer that subtracts two inputs.

[class TextVectorization](https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization): A preprocessing layer which maps text features to integer sequences.

[class ThresholdedReLU](https://www.tensorflow.org/api_docs/python/tf/keras/layers/ThresholdedReLU): Thresholded Rectified Linear Unit.

[class TimeDistributed](https://www.tensorflow.org/api_docs/python/tf/keras/layers/TimeDistributed): This wrapper allows to apply a layer to every temporal slice of an input.

[class UpSampling1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/UpSampling1D): Upsampling layer for 1D inputs.

[class UpSampling2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/UpSampling2D): Upsampling layer for 2D inputs.

[class UpSampling3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/UpSampling3D): Upsampling layer for 3D inputs.

[class Wrapper](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Wrapper): Abstract wrapper base class.

[class ZeroPadding1D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/ZeroPadding1D): Zero-padding layer for 1D input (e.g. temporal sequence).

[class ZeroPadding2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/ZeroPadding2D): Zero-padding layer for 2D input (e.g. picture).

[class ZeroPadding3D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/ZeroPadding3D): Zero-padding layer for 3D data (spatial or spatio-temporal).

Functions

[Input(...)](https://www.tensorflow.org/api_docs/python/tf/keras/Input): Input() is used to instantiate a Keras tensor.

[add(...)](https://www.tensorflow.org/api_docs/python/tf/keras/layers/add): Functional interface to the [tf.keras.layers.Add](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Add) layer.

[average(...)](https://www.tensorflow.org/api_docs/python/tf/keras/layers/average): Functional interface to the [tf.keras.layers.Average](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Average) layer.

[concatenate(...)](https://www.tensorflow.org/api_docs/python/tf/keras/layers/concatenate): Functional interface to the Concatenate layer.

[deserialize(...)](https://www.tensorflow.org/api_docs/python/tf/keras/layers/deserialize): Instantiates a layer from a config dictionary.

[dot(...)](https://www.tensorflow.org/api_docs/python/tf/keras/layers/dot): Functional interface to the Dot layer.

[maximum(...)](https://www.tensorflow.org/api_docs/python/tf/keras/layers/maximum): Functional interface to compute maximum (element-wise) list of inputs.

[minimum(...)](https://www.tensorflow.org/api_docs/python/tf/keras/layers/minimum): Functional interface to the Minimum layer.

[multiply(...)](https://www.tensorflow.org/api_docs/python/tf/keras/layers/multiply): Functional interface to the Multiply layer.

[serialize(...)](https://www.tensorflow.org/api_docs/python/tf/keras/layers/serialize): Serializes a Layer object into a JSON-compatible representation.

[subtract(...)](https://www.tensorflow.org/api_docs/python/tf/keras/layers/subtract): Functional interface to the Subtract layer.