*Torch.[data\_type]Tensor(shape)* – создание тензора указанного data\_type и указанной размерности. Примеры: FloatTensor, IntTensor…

*Torch.zeroes(shape, [dtype = torch.data\_type])* – создание нулевого тензора нужной размерности, dtype – возможность указать тип данных для тензора

*Torch.ones(shape)* – аналогичная предыдущей функция, создающая матрицу, заполненную единицами

*Torch.eye(shape)* – диагональная матрица нужной размерности

*Torch.full(shape, value)* – создание матрицы формы shape, заполненной везде value.

*Torch.arange(start, stop, step)* – создание тензора с заданной последовательностью.

*Torch.linspace(start, stop, batches\_number)* – создает последовательность, поделенную линейно на batch\_number отрезков. (1, 5, 2) = [1,5]; (1,5,1) = [1,]; (1,5,3) = [1, 3, 5]

*Torch.rand(shape)* – матрица заданной размерности с рандомными числами из равномерного распределения

*Torch.randn(shape)* – матрица заданной размерности с рандомными числами из нормального распределения со средним равным 0 и стандартным отклонением равным 1.

*Torch.manual\_seed(value)* – стабилизация аутпута генератора рандомных чисел.

**Inplace методы** – подменяют данные в текущем объекте. Обозначаются “\_” на конце:

* x = *torch.Tensor(shape).fill\_(value)* – заполнит тензор x значениями value.
* *x.random\_(start, end)* – заполняет тензор x рандомными int значениями от start до end (не включая). Равномерное распределение.
* *x.uniform\_(start, end)* – заполняет тензор x рандомными float значениями от start до end (не включая). Равномерное распределение.
* *x.normal\_(mean, std\_dev)* - заполняет тензор x рандомными float значениями от start до end (не включая). Нормальное распределение.

**Представления тензоров (tensor view)**

*x = torch.Tensor(27)* – дает нам вектор из 27 элементов.

*y = x.view(3, 3, 3)* – возвратит объект y, с размерностью 3, 3, 3. При этом что x, что y будут ссылаться на один и тот же объект данных. Т.е. при изменении x[0] = 100, y[0,0,0], ссылаясь на тот же объект данных будет равен 100.

Аналогичный .view метод – .reshape. Работает по тем же правилам.

Inplace *метод .resize\_(shape)* позволяет изменить размерность самого объекта данных, т.е. обрезать или расширить его. View при несоответствующем shape выдаст ошибку.

*.ravel()* вытягивает тензор любой размерности в единичный (в вектор). Аналог .reshape(-1) в numpy.

**Работа с осями**

*x.mT* – возвращает транспонированный тензор

*x.permute(axis1, axis2)* – меняет местами оси axis1 и axis2.

*torch.unsqueeze(tensor\_object, dim)* – добавление новой оси. Существует inplace версия unsqueeze\_(…)

*torch.squeeze(tensor\_object, [dim])* – удаление единичных осей, т.е. из shape [3,1,2,1,4] получается [3,2,4]. Если добавить dim, то удалит только ту ось, которая указана: при dim = 1 из shape [3,1,2,1,4] получается [3,2,1,4]

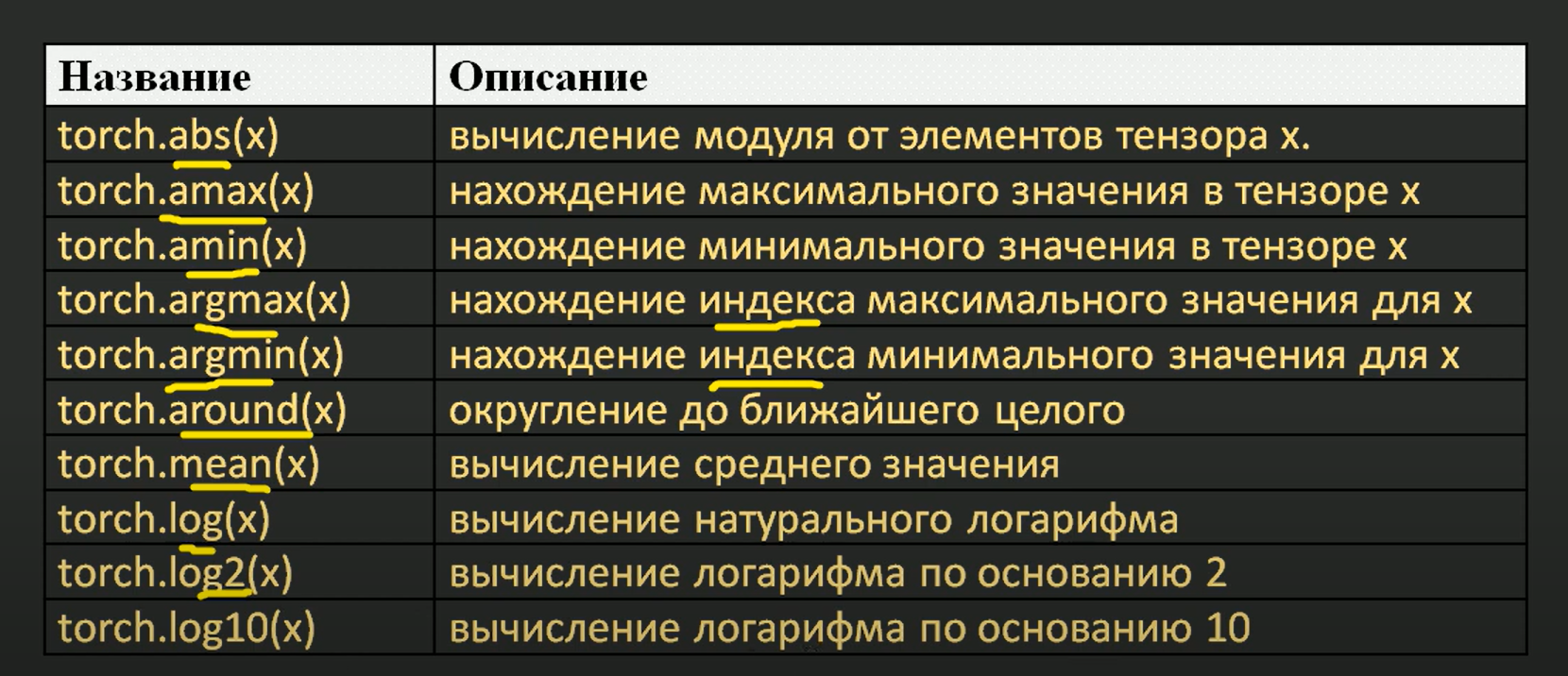
**Индексация**

Срезы и индексация работает аналогично numpy. При этом, если мы возвращаем один элемент из тензора, возвращается тензор 1x1, чтобы получить значение, нужно применить метод item() к аутпуту:

A screenshot of a computer code

Description automatically generated

**Аггрегация и другие трансформации**

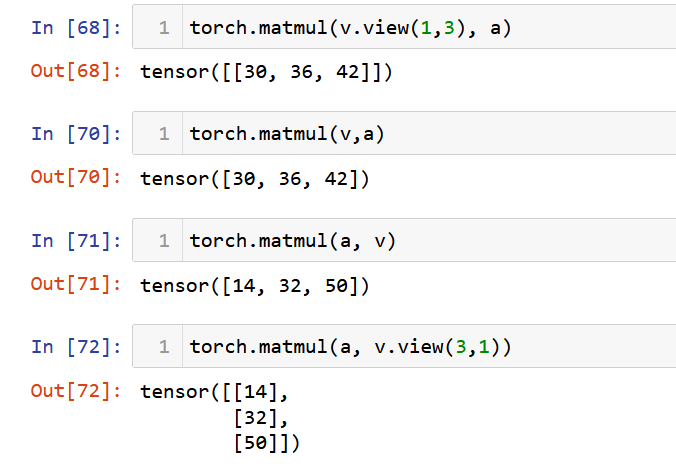


**Матричные операции**

*Torch.matmul(mat1, mat2)* – матричное умножение. Аналог np.dot(). Для этой же операции существует дополнительная функция (и метод) torch.mm(mat1, mat2). Различие в matmul и mm в том, что первая функция позволяет делать broadcast, т.е. автоматом преобразует вектор в матрицу вида 1xn или nx1.

a = torch.arange(1,10).reshape(3,3)

v = torch.LongTensor([1,2,3])



A screenshot of a computer program

Description automatically generated

**Перемножение матриц размерности 3**

*Torch.bmm(mat1, mat2)* – batch перемножение. Mat[n] должны быть размерности 3, так чтобы mat1.shape[0] == mat2.shape[0] and mat1.shape[2] = mat2.shape[1]. Т.е. чтобы размеры первой оси совпадали, а размер осей 2 и 3 позволяли производить матричное умножение.

A screenshot of a computer code

Description automatically generated

**Torch.nn**

*nn.Linear(in\_features = n, out\_features = m)*– создание полносвязного линейного слоя нейросети.

Torch.nn.Module – шаблон для создания нейросетки в форме класса. Используется следующим образом:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

from torch import nn

class MyNN(nn.Module):

def \_\_init\_\_(self, ...):

super().\_\_init\_\_() #Инициализируем базовый класс

... #Логика инициализации нашей нейросетки

def forward(self, x):

... #forward проход

return ... #Возврат аутпута нашей нейросети

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*model = MyNN([input\_params])*

*model.parameters()* – возвращает генератор со всем списком параметров модели, включая значения весов.