Вопрос 1

Выполнен

Баллов: 1,00 из 1,00

Отметьте правильные утверждения:

Выберите один или несколько ответов:

- [x] a. Строчка torch.backends.cudnn.deterministic = True может замедлить вычисления на GPU
- [x] b. Объект из датасета MNIST представляет из себя тензор размерности 28 на 28 элементов
- [x] c. Для воспроизводимости результатов обучения нейронной сети нужно каждый раз инициализировать random seed заново

Ваш ответ верный.

Вопрос Инфо

Вопрос 2

Выполнен

Баллов: 1,00 из 1,00

Попрактикуемся с методом reshape. У нас есть трехмерный тензор размерности (6000, 28, 28) . Сопоставьте операцию над этим тензором и её результатат:

x.reshape(-1,14,32,7).shape	(1500, 14, 32, 7)
x.shape	(6000, 28, 28)
x.reshape(-1,9).shape	ValueError
x.reshape(len(x[1]), len(x), len(x[2])).shape	(28, 6000, 28)
x.reshape(-1).shape	(4704000,)
x.reshape(-1,1,1).shape	(4704000,1,1)
x.reshape(-1,6000).shape	(784, 6000)

Ваш ответ верный.

Вопрос Инфо

Вопрос Инфо

Вопрос 3

Выполнен

Баллов: 1,00 из 1,00

Для того, чтобы переместить тензор на видеокарту мы можем воспользоваться методом .to('cuda:0'). Для того, чтобы вернуть тензор обратно на cpu можно воспользоваться тем же методом .to('cpu'). Кроме того, существует метод .cuda() , который аналогичен .to('cuda:0'). И .cpu() , который аналогичен .to('cpu').

Предположим, у нас есть тензор **a**, который находится на видеокарте и тензор **b**, который находится на cpu.

Чтобы произвести арифметическую операцию с этими тензорами, мы можем:

Выберите один или несколько ответов:

- [x] 1. перевести b на gpu
- [ ] 2. оставить все как есть
- [x] 3. перевести a на cpu

Ваш ответ верный.

Вопрос 4

Выполнен

Баллов: 1,00 из 1,00

Всегда ли верно, что увеличение accuracy на трейне всегда ведет за собой увеличение accuracy на тесте?

Выберите один ответ:

a. Да

b. Нет

Ваш ответ верный.

Вопрос Инфо

Вопрос 5

Верно

Баллов: 1,00 из 1,00

Как было сказано в предыдущем уроке, полносвязный слой может быть представлен как матричное умножение матрицы входов (**X**) и матрицы весов нейронов слоя (**W**), плюс вектор bias'ов слоя (**b**).

В документации к классу torch.nn.Linear (полносвязному слою) написано следующее: Applies a linear transformation to the incoming data: $y = xA^T + b$. **A** здесь – это то, как PyTorch хранит веса слоя. Но чтобы эта матрица совпала с **W** из предыдущего урока, нужно её сперва транспонировать.

Давайте реализуем функциональность torch.nn.Linear и сверим с оригиналом!

Пусть у нас будет 1 объект **x** на входе с двумя компонентами. Его мы передадим в полносвязный слой с 3-мя нейронами и получим, соотсветственно, 3 выхода. После напишем эту же функциональность с помощью матричного умножения.

Для примера:

Тест	Ввод	Результат
`print(fc_out == fc_out_alternative)`	`anything`	`tensor([[True, True, True]])`

Ответ: (штрафной режим: 0 %)

Сброс ответа

```
import torch

# Сперва создадим тензор x:
x = torch.tensor([[10., 20.]])

# Оригинальный полносвязный слой с 2-мя входами и 3-мя нейронами (выходами):
fc = torch.nn.Linear(2, 3)

# Веса fc-слоя хранятся в fc.weight, а bias'ы соответственно в fc.bias
# fc.weight и fc.bias по умолчанию инициализируются случайными числами

# Давайте проставим свои значения в веса и bias'ы:
w = torch.tensor([[11., 12.], [21., 22.], [31., 32]])
fc.weight.data = w

b = torch.tensor([[31., 32., 33.]])
fc.bias.data = b

# Получим выход fc-слоя:
fc_out = fc(x)

# Попробуем теперь получить аналогичные выходы с помощью матричного перемножения:
fc_out_alternative = x @ torch.t(fc.weight.data) + b

# Проверка осуществляется автоматически вызовом функции
#print(fc_out == fc_out_alternative)
# (раскомментируйте, если решаете задачу локально)
```

Прошли все тесты! ✔

Верно

Баллы за эту попытку: 1,00/1,00.

Вопрос 6

Верно

Баллов: 1,00 из 1,00

В предыдущем шаге мы написали функцию, эмулирующую fc-слой. Проверим, что по ней правильно считается градиент.

Функцию backward() в PyTorch можно посчитать только от *скалярной* функции (выход из такой функции – одно число). Это логично, так как loss-функция выдает всегда одно число. Но fc-слой, который мы проэмулировали, имел 3 выхода. Предлагаем их просуммировать, чтобы получить в итоге скалярную функцию. Заметим, впрочем, что можно было бы выбрать любую агрегирующую операцию, например умножение.

Дополните код так, чтобы градиент по весам и смещениям (bias) совпадал с аналогичным градиентом в вашей фунции.

Чем обусловлен полученный градиент? Изменится ли он, если мы подадим другие входы или другую инициализацию весов?

Для примера:

Тест	Ввод	Результат

`print('fc_weight_grad:', weight_grad)`
`print('our_weight_grad:', w.grad)`
`print('fc_bias_grad:', bias_grad)`
`print('out_bias_grad:', b.grad)`
	anything	`fc_weight_grad: tensor([[10., 20.],`
		`[10., 20.],`
		`[10., 20.]])`
		`our_weight_grad: tensor([[10., 20.],`
		`[10., 20.],`
		`[10., 20.]])`
		`fc_bias_grad: tensor([[1., 1., 1.]])`
		`out_bias_grad: tensor([[1., 1., 1.]])`

Ответ: (штрафной режим: 0 %)

Сброс ответа

```
import torch

# Сперва создадим тензор x:
x = torch.tensor([[10., 20.]])

# Оригинальный полносвязный слой с 2-мя входами и 3-мя нейронами (выходами):
fc = torch.nn.Linear(2, 3)

# Веса fc-слоя хранятся в fc.weight, а bias'ы соответственно в fc.bias
# fc.weight и fc.bias по умолчанию инициализируются случайными числами

# Давайте проставим свои значения в веса и bias'ы:
w = torch.tensor([[11., 12.], [21., 22.], [31., 32]])
fc.weight.data = w

b = torch.tensor([[31., 32., 33.]])
fc.bias.data = b

# Получим выход fc-слоя:
fc_out = fc(x)
# Просуммируем выход fc-слоя, чтобы получить скаляр:
fc_out_summed = fc_out.sum()

# Посчитаем градиенты формулы fc_out_summed:
fc_out_summed.backward()
weight_grad = fc.weight.grad
bias_grad = fc.bias.grad

# Ok, теперь воспроизведем вычисления выше но без fc-слоя:
# Проставим, что у "w" и "b" нужно вычислять градиенты (для fc-слоя это произошло автоматически):
w.requires_grad_(True)
b.requires_grad_(True)

# Получим выход нашей формулы:
our_formula = torch.matmul(x, w.t()) + b # SUM{x * w^T + b}

# Сделайте backward для нашей формулы:
our_formula.sum().backward()

# Проверка осуществляется автоматически, вызовом функций:
#print('fc_weight_grad:', weight_grad)
#print('our_weight_grad:', w.grad)
#print('fc_bias_grad:', bias_grad)
#print('out_bias_grad:', b.grad)
```

```
# (раскомментируйте, если работаете над задачей локально)
```

Прошли все тесты! ✔

Верно

Баллы за эту попытку: 1,00/1,00.

◄ 4.3 Задачи по теме: Понимаем SGD с momentum

Перейти на...

5.1 Свёртка, каскад свёрток ►