

Машинное обучение, DS-поток

Домашнее задание 8

См. сначала `[0]task8_train_model.ipynb` , в котором описана структура этого задания.

Перед выполнением этого ноутбука нужно реализовать все слои в `[1]task8_modules.ipynb` .

Если все тесты в этом ноутбуке пройдены, то можно приступить к выполнению `[0]task8_train_model.ipynb` для тестирования нейросети на датасетах.

Тестирование

```
In [ ]: 1 %run [1]task8_modules.ipynb
```

```
In [ ]: 1 import torch
        2 import numpy as np
        3 import unittest
```

```
In [ ]: 1 RANDOM_SEED = 42
```

Примечание: Если тестирование линейного слоя по каким-либо техническим причинам не работает (известны случаи, когда ядро ноутбука "умирает" из-за этого теста), то рекомендуем использовать Google Colab для запуска этого ноутбука.

```

In [ ]: 1 ▾ class TestLayers(unittest.TestCase):
2 ▾     """
3         Класс для тестирования всех модулей.
4     """
5
6 ▾     def _generate_test_data(
7         self, shape, dtype=np.float32, mode='uniform', minval=-10, maxval=10
8     ):
9         """
10        Генерирует тестовые данные для `forward()` и `backward()`
11        """
12 ▾         if mode == 'uniform':
13             rand_array = np.random.uniform(minval, maxval, shape).astype(dtype)
14             # иногда имеет смысл нормализовать
15             # rand_array /= rand_array.sum(axis=-1, keepdims=True)
16             # rand_array = rand_array.clip(1e-5, 1.)
17             # rand_array = 1. / rand_array
18             return rand_array
19
20
21 ▾     def _custom_forward_backward(
22         self,
23         layer_input,
24         next_layer_grad,
25         custom_layer,
26         return_params_grad=False
27     ):
28         """
29        Вычисляет результат `forward()` и `backward()` в слое `layer`.
30
31        Вход:
32        `layer_input` (np.array) -- тестовый вход
33 ▾        `next_layer_grad` (np.array) -- тестовый градиент,
34            пришедший от следующего слоя
35        `layer` -- слой из нашего мини-фреймворка на NumPy
36        `return_params_grad` -- если True, то вернуть еще градиенты параметров слоя
37 ▾        Выход:
38        `custom_layer_output` (np.array) -- выход слоя `layer` после `forward()`
39        `custom_layer_grad` (np.array) -- градиент слоя `layer` после `backward()`
40        [opt] `custom_params_grad` (np.array) -- градиенты параметров слоя `layer`
41        """
42        custom_layer_output = custom_layer.forward(layer_input)
43        custom_layer_grad = custom_layer.backward(layer_input, next_layer_grad)
44 ▾        if return_params_grad:
45            custom_layer.update_grad_params(layer_input, next_layer_grad)
46            custom_params_grad = custom_layer.get_grad_params()

```

```

47         return custom_layer_output, custom_layer_grad, custom_params_grad
48     else:
49         return custom_layer_output, custom_layer_grad
50
51
52     def _torch_forward_backward(
53         self,
54         layer_input,
55         next_layer_grad,
56         torch_layer,
57         return_params_grad=False
58     ):
59         ...
60         Вычисляет результат `forward()` и `backward()` в PyTorch-слое `layer`.
61
62         Вход:
63         `layer_input` (np.array) -- тестовый вход
64         `next_layer_grad` (np.array) -- тестовый градиент,
65             пришедший от следующего слоя
66         `torch_layer` -- слой из PyTorch
67         `return_params_grad` -- если True, то вернуть еще градиенты параметров слоя
68
69         Выход:
70         `torch_layer_output` (np.array) -- выход слоя `layer` после `forward()`
71         `torch_layer_grad` (np.array) -- градиент слоя `layer` после `backward()`
72         [opt] `torch_params_grad` (np.array) -- градиенты параметров слоя `layer`
73         ...
74         layer_input_torch = torch.from_numpy(layer_input)
75         layer_input_torch.requires_grad = True
76         torch_layer_output = torch_layer(layer_input_torch)
77         torch_layer_output = torch_layer_output
78         next_layer_grad_torch = torch.from_numpy(next_layer_grad)
79         torch_layer_output.backward(next_layer_grad_torch)
80         torch_layer_grad = layer_input_torch.grad
81         if return_params_grad:
82             torch_params_grad = torch_layer.parameters()
83             return torch_layer_output.data.numpy(), torch_layer_grad.data.numpy(), torch_params_grad
84         else:
85             return torch_layer_output.data.numpy(), torch_layer_grad.data.numpy()
86
87     # def test_Linear(self):
88     #     np.random.seed(RANDOM_SEED)
89     #     torch.manual_seed(RANDOM_SEED)
90     #     batch_size, n_in, n_out = 2, 3, 4
91     #     for _ in range(100):
92     #         # инициализируем слои

```

```

93 # torch_layer = torch.nn.Linear(n_in, n_out)
94 # custom_layer = Linear(n_in, n_out)
95 # custom_layer.W = torch_layer.weight.data.numpy().T
96 # custom_layer.b = torch_layer.bias.data.numpy()
97 # формируем тестовые входные тензоры
98 # layer_input = self._generate_test_data((batch_size, n_in))
99 # next_layer_grad = self._generate_test_data((batch_size, n_out))
100 # тестируем наш слой
101 # result = self._custom_forward_backward(
102 #     layer_input,
103 #     next_layer_grad,
104 #     custom_layer,
105 #     return_params_grad=True
106 # )
107 # custom_layer_output, custom_layer_grad, custom_params_grad = result
108 # тестируем слой на PyTorch
109 # result = self._torch_forward_backward(
110 #     layer_input,
111 #     next_layer_grad,
112 #     torch_layer,
113 #     return_params_grad=True
114 # )
115 # torch_layer_output, torch_layer_grad, torch_params_grad = result
116 # сравниваем выходы с точностью atol
117 # self.assertTrue(np.allclose(torch_layer_output, custom_layer_output, atol=1e-6))
118 # self.assertTrue(np.allclose(torch_layer_grad, custom_layer_grad, atol=1e-6))
119 # weight_grad, bias_grad = custom_params_grad
120 # torch_weight_grad = torch_layer.weight.grad.data.numpy()
121 # torch_bias_grad = torch_layer.bias.grad.data.numpy()
122 # self.assertTrue(np.allclose(torch_weight_grad.T, weight_grad, atol=1e-6))
123 # self.assertTrue(np.allclose(torch_bias_grad, bias_grad, atol=1e-6))
124
125
126 ▾ def test_SoftMax(self):
127     np.random.seed(RANDOM_SEED)
128     torch.manual_seed(RANDOM_SEED)
129     batch_size, n_in = 2, 4
130     ▾ for _ in range(100):
131         # инициализируем слой
132         custom_layer = LogSoftMax()
133         torch_layer = torch.nn.LogSoftmax(dim=1)
134         # формируем тестовые входные тензоры
135         layer_input = self._generate_test_data((batch_size, n_in))
136         next_layer_grad = self._generate_test_data((batch_size, n_in))
137         # тестируем наш слой
138     ▾ custom_layer_output, custom_layer_grad = self._custom_forward_backward(

```

```

139         layer_input,
140         next_layer_grad,
141         custom_layer
142     )
143     # тестируем слой на PyTorch
144     torch_layer_output, torch_layer_grad = self._torch_forward_backward(
145         layer_input,
146         next_layer_grad,
147         torch_layer
148     )
149     # сравниваем выходы с точностью atol
150     self.assertTrue(np.allclose(custom_layer_output, torch_layer_output, atol=1e-5))
151     self.assertTrue(np.allclose(custom_layer_grad, torch_layer_grad, atol=1e-5))
152
153
154     def test_LogSoftMax(self):
155         np.random.seed(RANDOM_SEED)
156         torch.manual_seed(RANDOM_SEED)
157         batch_size, n_in = 2, 4
158         for _ in range(100):
159             # инициализируем слои
160             custom_layer = LogSoftMax()
161             torch_layer = torch.nn.LogSoftmax(dim=1)
162             # формируем тестовые входные тензоры
163             layer_input = self._generate_test_data((batch_size, n_in))
164             next_layer_grad = self._generate_test_data((batch_size, n_in))
165             # тестируем наш слой
166             custom_layer_output, custom_layer_grad = self._custom_forward_backward(
167                 layer_input,
168                 next_layer_grad,
169                 custom_layer
170             )
171             # тестируем слой на PyTorch
172             torch_layer_output, torch_layer_grad = self._torch_forward_backward(
173                 layer_input,
174                 next_layer_grad,
175                 torch_layer
176             )
177             # сравниваем выходы с точностью atol
178             self.assertTrue(np.allclose(custom_layer_output, torch_layer_output, atol=1e-5))
179             self.assertTrue(np.allclose(custom_layer_grad, torch_layer_grad, atol=1e-5))
180
181
182     def test_BatchNormalization(self):
183         np.random.seed(RANDOM_SEED)
184         torch.manual_seed(RANDOM_SEED)

```

```

185 batch_size, n_in = 32, 16
186 for _ in range(100):
187     # инициализируем слои
188     slope = np.random.uniform(0.01, 0.05)
189     alpha = 0.9
190     custom_layer = BatchNormalization(alpha)
191     custom_layer.train()
192     torch_layer = torch.nn.BatchNorm1d(
193         num_features=n_in,
194         eps=custom_layer.EPS,
195         momentum=1.-alpha,
196         affine=False
197     )
198     custom_layer.moving_mean = torch_layer.running_mean.numpy().copy()
199     custom_layer.moving_variance = torch_layer.running_var.numpy().copy()
200     # формируем тестовые входные тензоры
201     layer_input = self._generate_test_data((batch_size, n_in))
202     next_layer_grad = self._generate_test_data((batch_size, n_in))
203     # тестируем наш слой
204     custom_layer_output, custom_layer_grad = self._custom_forward_backward(
205         layer_input,
206         next_layer_grad,
207         custom_layer,
208         return_params_grad=False
209     )
210     # тестируем слой на PyTorch
211     torch_layer_output, torch_layer_grad = self._torch_forward_backward(
212         layer_input,
213         next_layer_grad,
214         torch_layer,
215         return_params_grad=False
216     )
217     # сравниваем выходы с точностью atol
218     self.assertTrue(np.allclose(torch_layer_output, custom_layer_output, atol=1e-6))
219     self.assertTrue(np.allclose(torch_layer_grad, custom_layer_grad, atol=1e-4))
220     # тестируем moving mean
221     self.assertTrue(np.allclose(custom_layer.moving_mean, torch_layer.running_mean.numpy()))
222     # мы не проверяем moving variance, потому что в PyTorch используется
223     # немного другая формула: var * N / (N-1) (несмещенная оценка)
224     self.assertTrue(np.allclose(custom_layer.moving_variance, torch_layer.running_var.numpy()))
225     # тестируем BatchNorm на стадии evaluation
226     custom_layer.moving_variance = torch_layer.running_var.numpy().copy()
227     custom_layer.evaluate()
228     torch_layer.eval()
229     custom_layer_output, custom_layer_grad = self._custom_forward_backward(
230         layer_input,

```

```

231         next_layer_grad,
232         custom_layer,
233         return_params_grad=False
234     )
235     torch_layer_output, torch_layer_grad = self._torch_forward_backward(
236         layer_input,
237         next_layer_grad,
238         torch_layer,
239         return_params_grad=False
240     )
241     self.assertTrue(np.allclose(torch_layer_output, custom_layer_output, atol=1e-6))
242
243
244     def test_Sequential(self):
245         np.random.seed(RANDOM_SEED)
246         torch.manual_seed(RANDOM_SEED)
247         batch_size, n_in = 2, 4
248         for _ in range(100):
249             # инициализируем слои
250             alpha = 0.9
251             # тестируем `Sequential = [BatchNormalization, Scaling]`,
252             # чтобы убедиться, что он равен `BatchNorm1d` из PyTorch
253             # torch слой
254             torch_layer = torch.nn.BatchNorm1d(
255                 n_in, eps=BatchNormalization.EPS, momentum=1.-alpha, affine=True
256             )
257             torch_layer.bias.data = torch.from_numpy(np.random.random(n_in).astype(np.float32))
258             # наши слои
259             custom_layer = Sequential()
260             bn_layer = BatchNormalization(alpha)
261             bn_layer.moving_mean = torch_layer.running_mean.numpy().copy()
262             bn_layer.moving_variance = torch_layer.running_var.numpy().copy()
263             custom_layer.add(bn_layer)
264             scaling_layer = Scaling(n_in)
265             scaling_layer.gamma = torch_layer.weight.data.numpy()
266             scaling_layer.beta = torch_layer.bias.data.numpy()
267             custom_layer.add(scaling_layer)
268             custom_layer.train()
269             # формируем тестовые входные тензоры
270             layer_input = self._generate_test_data((batch_size, n_in), minval=-5, maxval=5)
271             next_layer_grad = self._generate_test_data((batch_size, n_in), minval=-5, maxval=5)
272             # тестируем наш слой
273             result = self._custom_forward_backward(
274                 layer_input,
275                 next_layer_grad,
276                 custom_layer,

```

```

277         return_params_grad=True
278     )
279     custom_layer_output, custom_layer_grad, custom_params_grad = result
280     # тестируем слой на PyTorch
281     result = self._torch_forward_backward(
282         layer_input,
283         next_layer_grad,
284         torch_layer,
285         return_params_grad=True
286     )
287     torch_layer_output, torch_layer_grad, torch_params_grad = result
288     # сравниваем выходы с точностью atol
289     self.assertTrue(np.allclose(torch_layer_output, custom_layer_output, atol=1e-6))
290     self.assertTrue(np.allclose(torch_layer_grad, custom_layer_grad, atol=1e-4))
291     weight_grad, bias_grad = custom_params_grad[1]
292     torch_weight_grad = torch_layer.weight.grad.data.numpy()
293     torch_bias_grad = torch_layer.bias.grad.data.numpy()
294     self.assertTrue(np.allclose(torch_weight_grad, weight_grad, atol=1e-6))
295     self.assertTrue(np.allclose(torch_bias_grad, bias_grad, atol=1e-6))
296
297
298     def test_Dropout(self):
299         np.random.seed(RANDOM_SEED)
300         batch_size, n_in = 2, 4
301         for _ in range(100):
302             # инициализируем слой
303             p = np.random.uniform(0.3, 0.7)
304             layer = Dropout(p)
305             layer.train()
306             # формируем тестовые входные тензоры
307             layer_input = self._generate_test_data((batch_size, n_in), minval=-5, maxval=5)
308             next_layer_grad = self._generate_test_data((batch_size, n_in), minval=-5, maxval=5)
309             # тестируем `update_output()`
310             layer_output = layer.update_output(layer_input)
311             self.assertTrue(np.all(np.logical_or(
312                 np.isclose(layer_output, 0),
313                 np.isclose(layer_output*(1.-p), layer_input)
314             )))
315             # тестируем `update_grad_input()`
316             layer_grad = layer.update_grad_input(layer_input, next_layer_grad)
317             self.assertTrue(np.all(np.logical_or(
318                 np.isclose(layer_grad, 0),
319                 np.isclose(layer_grad*(1.-p), next_layer_grad)
320             )))
321             # тестируем evaluation режим
322             layer.evaluate()

```



```

323 layer_output = layer.update_output(layer_input)
324 self.assertTrue(np.allclose(layer_output, layer_input))
325 # тестируем маску при нескольких значениях `p`
326 p = 0.0
327 layer = Dropout(p)
328 layer.train()
329 layer_output = layer.update_output(layer_input)
330 self.assertTrue(np.allclose(layer_output, layer_input))
331 p = 0.5
332 layer = Dropout(p)
333 layer.train()
334 layer_input = self._generate_test_data((batch_size, n_in), minval=-5, maxval=5)
335 next_layer_grad = self._generate_test_data((batch_size, n_in), minval=-5, maxval=5)
336 layer_output = layer.update_output(layer_input)
337 zeroed_elem_mask = np.isclose(layer_output, 0)
338 layer_grad = layer.update_grad_input(layer_input, next_layer_grad)
339 self.assertTrue(np.all(zeroed_elem_mask == np.isclose(layer_grad, 0)))
340 # тестируем тот факт, что маска должна генерироваться независимо
341 # для каждого элемента входной матрицы, а не для строки/столбца
342 batch_size, n_in = 1000, 1
343 p = 0.8
344 layer = Dropout(p)
345 layer.train()
346 layer_input = self._generate_test_data((batch_size, n_in), minval=5, maxval=10)
347 layer_output = layer.update_output(layer_input)
348 self.assertTrue(np.sum(np.isclose(layer_output, 0)) != layer_input.size)
349 layer_input = layer_input.T
350 layer_output = layer.update_output(layer_input)
351 self.assertTrue(np.sum(np.isclose(layer_output, 0)) != layer_input.size)
352
353
354 ▼ def test_ReLU(self):
355     np.random.seed(RANDOM_SEED)
356     torch.manual_seed(RANDOM_SEED)
357     batch_size, n_in = 2, 4
358     ▼ for _ in range(100):
359         # инициализируем слои
360         custom_layer = ReLU()
361         torch_layer = torch.nn.ReLU()
362         # формируем тестовые входные тензоры
363         layer_input = self._generate_test_data((batch_size, n_in))
364         next_layer_grad = self._generate_test_data((batch_size, n_in))
365         # тестируем наш слой
366     ▼ custom_layer_output, custom_layer_grad = self._custom_forward_backward(
367         layer_input,
368         next_layer_grad,

```

```

369         custom_layer
370     )
371     # тестируем слой на PyTorch
372     torch_layer_output, torch_layer_grad = self._torch_forward_backward(
373         layer_input,
374         next_layer_grad,
375         torch_layer
376     )
377     # сравниваем выходы с точностью atol
378     self.assertTrue(np.allclose(custom_layer_output, torch_layer_output, atol=1e-6))
379     self.assertTrue(np.allclose(custom_layer_grad, torch_layer_grad, atol=1e-6))
380
381
382     def test_LeakyReLU(self):
383         np.random.seed(RANDOM_SEED)
384         torch.manual_seed(RANDOM_SEED)
385         batch_size, n_in = 2, 4
386         for _ in range(100):
387             # инициализируем слои
388             slope = np.random.uniform(0.01, 0.05)
389             torch_layer = torch.nn.LeakyReLU(slope)
390             custom_layer = LeakyReLU(slope)
391             # формируем тестовые входные тензоры
392             layer_input = self._generate_test_data((batch_size, n_in))
393             next_layer_grad = self._generate_test_data((batch_size, n_in))
394             # тестируем наш слой
395             custom_layer_output, custom_layer_grad = self._custom_forward_backward(
396                 layer_input,
397                 next_layer_grad,
398                 custom_layer
399             )
400             # тестируем слой на PyTorch
401             torch_layer_output, torch_layer_grad = self._torch_forward_backward(
402                 layer_input,
403                 next_layer_grad,
404                 torch_layer
405             )
406             # сравниваем выходы с точностью atol
407             self.assertTrue(np.allclose(custom_layer_output, torch_layer_output, atol=1e-6))
408             self.assertTrue(np.allclose(custom_layer_grad, torch_layer_grad, atol=1e-6))
409
410
411     def test_ELU(self):
412         np.random.seed(RANDOM_SEED)
413         torch.manual_seed(RANDOM_SEED)
414         batch_size, n_in = 2, 4

```

```

415     for _ in range(100):
416         # инициализируем слои
417         alpha = 1.0
418         torch_layer = torch.nn.ELU(alpha)
419         custom_layer = ELU(alpha)
420         # формируем тестовые входные тензоры
421         layer_input = self._generate_test_data((batch_size, n_in))
422         next_layer_grad = self._generate_test_data((batch_size, n_in))
423         # тестируем наш слой
424         custom_layer_output, custom_layer_grad = self._custom_forward_backward(
425             layer_input,
426             next_layer_grad,
427             custom_layer
428         )
429         # тестируем слой на PyTorch
430         torch_layer_output, torch_layer_grad = self._torch_forward_backward(
431             layer_input,
432             next_layer_grad,
433             torch_layer
434         )
435         # сравниваем выходы с точностью atol
436         self.assertTrue(np.allclose(custom_layer_output, torch_layer_output, atol=1e-6))
437         self.assertTrue(np.allclose(custom_layer_grad, torch_layer_grad, atol=1e-6))
438
439
440     def test_SoftPlus(self):
441         np.random.seed(RANDOM_SEED)
442         torch.manual_seed(RANDOM_SEED)
443         batch_size, n_in = 2, 4
444         for _ in range(100):
445             # инициализируем слои
446             custom_layer = SoftPlus()
447             torch_layer = torch.nn.Softplus()
448             # формируем тестовые входные тензоры
449             layer_input = self._generate_test_data((batch_size, n_in))
450             next_layer_grad = self._generate_test_data((batch_size, n_in))
451             # тестируем наш слой
452             custom_layer_output, custom_layer_grad = self._custom_forward_backward(
453                 layer_input,
454                 next_layer_grad,
455                 custom_layer
456             )
457             # тестируем слой на PyTorch
458             torch_layer_output, torch_layer_grad = self._torch_forward_backward(
459                 layer_input,
460                 next_layer_grad,

```

```

461         torch_layer
462     )
463     # сравниваем выходы с точностью atol
464     self.assertTrue(np.allclose(custom_layer_output, torch_layer_output, atol=1e-6))
465     self.assertTrue(np.allclose(custom_layer_grad, torch_layer_grad, atol=1e-6))
466
467
468     def test_NLLCriterionUnstable(self):
469         np.random.seed(RANDOM_SEED)
470         torch.manual_seed(RANDOM_SEED)
471         batch_size, n_in = 2, 4
472         for _ in range(100):
473             # инициализируем слои
474             torch_layer = torch.nn.NLLLoss()
475             custom_layer = NLLCriterionUnstable()
476             # формируем тестовые данные
477             layer_input = np.random.uniform(0, 1, (batch_size, n_in)).astype(np.float32)
478             layer_input /= layer_input.sum(axis=-1, keepdims=True)
479             layer_input = layer_input.clip(custom_layer.EPS, 1.-custom_layer.EPS)
480             target_labels = np.random.choice(n_in, batch_size)
481             target = np.zeros((batch_size, n_in), np.float32)
482             target[np.arange(batch_size), target_labels] = 1
483             # тестируем `update_output()`
484             custom_layer_output = custom_layer.update_output(layer_input, target)
485             layer_input_var = torch.from_numpy(layer_input)
486             layer_input_var.requires_grad = True
487             torch_layer_output_var = torch_layer(
488                 torch.log(layer_input_var), torch.from_numpy(target_labels).long())
489             self.assertTrue(np.allclose(
490                 torch_layer_output_var.data.numpy(), custom_layer_output, atol=1e-6
491             ))
492             # тестируем `update_grad_input()`
493             custom_layer_grad = custom_layer.update_grad_input(layer_input, target)
494             torch_layer_output_var.backward()
495             torch_layer_grad_var = layer_input_var.grad
496             self.assertTrue(np.allclose(torch_layer_grad_var.data.numpy(), custom_layer_grad, atol=1e-6))
497
498     def test_NLLCriterion(self):
499         np.random.seed(RANDOM_SEED)
500         torch.manual_seed(RANDOM_SEED)
501         batch_size, n_in = 2, 4
502         for _ in range(100):
503             # инициализируем слои
504             torch_layer = torch.nn.NLLLoss()
505             custom_layer = NLLCriterion()
506             # формируем тестовые данные

```

```

507 layer_input = np.random.uniform(-5, 5, (batch_size, n_in)).astype(np.float32)
508 layer_input = torch.nn.LogSoftmax(dim=1)(torch.from_numpy(layer_input)).data.numpy()
509 target_labels = np.random.choice(n_in, batch_size)
510 target = np.zeros((batch_size, n_in), np.float32)
511 target[np.arange(batch_size), target_labels] = 1 # one-hot encoding
512 # тестировать `update_output()`
513 custom_layer_output = custom_layer.update_output(layer_input, target)
514 layer_input_var = torch.from_numpy(layer_input)
515 layer_input_var.requires_grad = True
516 torch_layer_output_var = torch_layer(layer_input_var, torch.from_numpy(target_labels).long())
517 self.assertTrue(np.allclose(
518     torch_layer_output_var.data.numpy(), custom_layer_output, atol=1e-6
519 ))
520 # тестировать `update_grad_input()`
521 custom_layer_grad = custom_layer.update_grad_input(layer_input, target)
522 torch_layer_output_var.backward()
523 torch_layer_grad_var = layer_input_var.grad
524 self.assertTrue(np.allclose(torch_layer_grad_var.data.numpy(), custom_layer_grad, atol=1e-6))
525
526
527 suite = unittest.TestLoader().loadTestsFromTestCase(TestLayers)
528 unittest.TextTestRunner(verbosity=2).run(suite)

```