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 приступать к выполнению **`[0]task8_train_model.ipynb`** для тестирования нейросети на датасетах.

Тестирование

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

In [ ]:
```
import torch
import numpy as np
import unittest
```

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

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

In [ ]:

```python
class TestLayers(unittest.TestCase):
    '''
        Класс для тестирования всех модулей.
    '''

    def _generate_test_data(
        self, shape, dtype=np.float32, mode='uniform', minval=-10, maxval=10
    ):
        '''
        Генерирует тестовые данные для `forward()` и `backward()`
        '''
        if mode == 'uniform':
            rand_array = np.random.uniform(minval, maxval, shape).astype(dtype)
            # иногда имеет смысл нормализовать
            # rand_array /= rand_array.sum(axis=-1, keepdims=True)
            # rand_array = rand_array.clip(1e-5, 1.)
            # rand_array = 1. / rand_array
            return rand_array


    def _custom_forward_backward(
        self,
        layer_input,
        next_layer_grad,
        custom_layer,
        return_params_grad=False
    ):
        '''
        Вычисляет результат `forward()` и `backward()` в слое `layer`.

        Вход:
            `layer_input (np.array)` -- тестовый вход
            `next_layer_grad (np.array)` -- тестовый градиент,
                пришедший от следующего слоя
            `layer` -- слой из нашего мини-фреймворка на NumPy
            `return_params_grad` -- если True, то вернуть еще градиенты параметров слоя
        Выход:
            `custom_layer_output (np.array)` -- выход слоя `layer` после `forward()`
            `custom_layer_grad (np.array)` -- градиент слоя `layer` после `backward()`
            [opt] `custom_params_grad (np.array)` -- градиенты параметров слоя `layer`
        '''
        custom_layer_output = custom_layer.forward(layer_input)
        custom_layer_grad = custom_layer.backward(layer_input, next_layer_grad)
        if return_params_grad:
            custom_layer.update_grad_params(layer_input, next_layer_grad)
            custom_params_grad = custom_layer.get_grad_params()
```

```
            return custom_layer_output, custom_layer_grad, custom_params_grad
        else:
            return custom_layer_output, custom_layer_grad


    def _torch_forward_backward(
        self,
        layer_input,
        next_layer_grad,
        torch_layer,
        return_params_grad=False
    ):
        '''
        Вычисляет результат `forward()` и `backward()` в PyTorch-слое `layer`.

        Вход:
            `layer_input (np.array)` -- тестовый вход
            `next_layer_grad (np.array)` -- тестовый градиент,
                пришедший от следующего слоя
            `torch_layer` -- слой из PyTorch
            `return_params_grad` -- если True, то вернуть еще градиенты параметров слоя
        Выход:
            `torch_layer_output (np.array)` -- выход слоя `layer` после `forward()`
            `torch_layer_grad (np.array)` -- градиент слоя `layer` после `backward()`
            [opt] `torch_params_grad (np.array)` -- градиенты параметров слоя `layer`
        '''
        layer_input_torch = torch.from_numpy(layer_input)
        layer_input_torch.requires_grad = True
        torch_layer_output = torch_layer(layer_input_torch)
        torch_layer_output = torch_layer_output
        next_layer_grad_torch = torch.from_numpy(next_layer_grad)
        torch_layer_output.backward(next_layer_grad_torch)
        torch_layer_grad = layer_input_torch.grad
        if return_params_grad:
            torch_params_grad = torch_layer.parameters()
            return torch_layer_output.data.numpy(), torch_layer_grad.data.numpy(), torch_params_grad
        else:
            return torch_layer_output.data.numpy(), torch_layer_grad.data.numpy()


#     def test_Linear(self):
#         np.random.seed(RANDOM_SEED)
#         torch.manual_seed(RANDOM_SEED)
#         batch_size, n_in, n_out = 2, 3, 4
#         for _ in range(100):
#             # инициализируем слои
```

```
#             torch_layer = torch.nn.Linear(n_in, n_out)
#             custom_layer = Linear(n_in, n_out)
#             custom_layer.W = torch_layer.weight.data.numpy().T
#             custom_layer.b = torch_layer.bias.data.numpy()
#             # формируем тестовые входные тензоры
#             layer_input = self._generate_test_data((batch_size, n_in))
#             next_layer_grad = self._generate_test_data((batch_size, n_out))
#             # тестируем наш слой
#             result = self._custom_forward_backward(
#                 layer_input,
#                 next_layer_grad,
#                 custom_layer,
#                 return_params_grad=True
#             )
#             custom_layer_output, custom_layer_grad, custom_params_grad = result
#             # тестируем слой на PyTorch
#             result = self._torch_forward_backward(
#                 layer_input,
#                 next_layer_grad,
#                 torch_layer,
#                 return_params_grad=True
#             )
#             torch_layer_output, torch_layer_grad, torch_params_grad = result
#             # сравниваем выходы с точностью atol
#             self.assertTrue(np.allclose(torch_layer_output, custom_layer_output, atol=1e-6))
#             self.assertTrue(np.allclose(torch_layer_grad, custom_layer_grad, atol=1e-6))
#             weight_grad, bias_grad = custom_params_grad
#             torch_weight_grad = torch_layer.weight.grad.data.numpy()
#             torch_bias_grad = torch_layer.bias.grad.data.numpy()
#             self.assertTrue(np.allclose(torch_weight_grad.T, weight_grad, atol=1e-6))
#             self.assertTrue(np.allclose(torch_bias_grad, bias_grad, atol=1e-6))


    def test_SoftMax(self):
        np.random.seed(RANDOM_SEED)
        torch.manual_seed(RANDOM_SEED)
        batch_size, n_in = 2, 4
        for _ in range(100):
            # инициализируем слои
            custom_layer = LogSoftMax()
            torch_layer = torch.nn.LogSoftmax(dim=1)
            # формируем тестовые входные тензоры
            layer_input = self._generate_test_data((batch_size, n_in))
            next_layer_grad = self._generate_test_data((batch_size, n_in))
            # тестируем наш слой
            custom_layer_output, custom_layer_grad = self._custom_forward_backward(
```

```
                layer_input,
                next_layer_grad,
                custom_layer
            )
            # тестируем слой на PyTorch
            torch_layer_output, torch_layer_grad = self._torch_forward_backward(
                layer_input,
                next_layer_grad,
                torch_layer
            )
            # сравниваем выходы с точностью atol
            self.assertTrue(np.allclose(custom_layer_output, torch_layer_output, atol=1e-5))
            self.assertTrue(np.allclose(custom_layer_grad, torch_layer_grad, atol=1e-5))


    def test_LogSoftMax(self):
        np.random.seed(RANDOM_SEED)
        torch.manual_seed(RANDOM_SEED)
        batch_size, n_in = 2, 4
        for _ in range(100):
            # инициализируем слои
            custom_layer = LogSoftMax()
            torch_layer = torch.nn.LogSoftmax(dim=1)
            # формируем тестовые входные тензоры
            layer_input = self._generate_test_data((batch_size, n_in))
            next_layer_grad = self._generate_test_data((batch_size, n_in))
            # тестируем наш слой
            custom_layer_output, custom_layer_grad = self._custom_forward_backward(
                layer_input,
                next_layer_grad,
                custom_layer
            )
            # тестируем слой на PyTorch
            torch_layer_output, torch_layer_grad = self._torch_forward_backward(
                layer_input,
                next_layer_grad,
                torch_layer
            )
            # сравниваем выходы с точностью atol
            self.assertTrue(np.allclose(custom_layer_output, torch_layer_output, atol=1e-5))
            self.assertTrue(np.allclose(custom_layer_grad, torch_layer_grad, atol=1e-5))


    def test_BatchNormalization(self):
        np.random.seed(RANDOM_SEED)
        torch.manual_seed(RANDOM_SEED)
```

```
        batch_size, n_in = 32, 16
        for _ in range(100):
            # инициализируем слои
            slope = np.random.uniform(0.01, 0.05)
            alpha = 0.9
            custom_layer = BatchNormalization(alpha)
            custom_layer.train()
            torch_layer = torch.nn.BatchNorm1d(
                num_features=n_in,
                eps=custom_layer.EPS,
                momentum=1.-alpha,
                affine=False
            )
            custom_layer.moving_mean = torch_layer.running_mean.numpy().copy()
            custom_layer.moving_variance = torch_layer.running_var.numpy().copy()
            # формируем тестовые входные тензоры
            layer_input = self._generate_test_data((batch_size, n_in))
            next_layer_grad = self._generate_test_data((batch_size, n_in))
            # тестируем наш слой
            custom_layer_output, custom_layer_grad = self._custom_forward_backward(
                layer_input,
                next_layer_grad,
                custom_layer,
                return_params_grad=False
            )
            # тестируем слой на PyTorch
            torch_layer_output, torch_layer_grad = self._torch_forward_backward(
                layer_input,
                next_layer_grad,
                torch_layer,
                return_params_grad=False
            )
            # сравниваем выходы с точностью atol
            self.assertTrue(np.allclose(torch_layer_output, custom_layer_output, atol=1e-6))
            self.assertTrue(np.allclose(torch_layer_grad, custom_layer_grad, atol=1e-4))
            # тестируем moving mean
            self.assertTrue(np.allclose(custom_layer.moving_mean, torch_layer.running_mean.numpy()))
            # мы не проверяем moving variance, потому что в PyTorch используется
            # немного другая формула: var * N / (N-1) (несмещенная оценка)
#             self.assertTrue(np.allclose(custom_layer.moving_variance, torch_layer.running_var.numpy()))
            # тестируем BatchNorm на стадии evaluation
            custom_layer.moving_variance = torch_layer.running_var.numpy().copy()
            custom_layer.evaluate()
            torch_layer.eval()
            custom_layer_output, custom_layer_grad = self._custom_forward_backward(
                layer_input,
```

```
                next_layer_grad,
                custom_layer,
                return_params_grad=False
            )
            torch_layer_output, torch_layer_grad = self._torch_forward_backward(
                layer_input,
                next_layer_grad,
                torch_layer,
                return_params_grad=False
            )
            self.assertTrue(np.allclose(torch_layer_output, custom_layer_output, atol=1e-6))


    def test_Sequential(self):
        np.random.seed(RANDOM_SEED)
        torch.manual_seed(RANDOM_SEED)
        batch_size, n_in = 2, 4
        for _ in range(100):
            # инициализируем слои
            alpha = 0.9
            # тестируем `Sequential = [BatchNormalization, Scaling]`,
            # чтобы убедиться, что он равен `BatchNorm1d` из PyTorch
            # torch слой
            torch_layer = torch.nn.BatchNorm1d(
                n_in, eps=BatchNormalization.EPS, momentum=1.-alpha, affine=True
            )
            torch_layer.bias.data = torch.from_numpy(np.random.random(n_in).astype(np.float32))
            # наши слои
            custom_layer = Sequential()
            bn_layer = BatchNormalization(alpha)
            bn_layer.moving_mean = torch_layer.running_mean.numpy().copy()
            bn_layer.moving_variance = torch_layer.running_var.numpy().copy()
            custom_layer.add(bn_layer)
            scaling_layer = Scaling(n_in)
            scaling_layer.gamma = torch_layer.weight.data.numpy()
            scaling_layer.beta = torch_layer.bias.data.numpy()
            custom_layer.add(scaling_layer)
            custom_layer.train()
            # формируем тестовые входные тензоры
            layer_input = self._generate_test_data((batch_size, n_in), minval=-5, maxval=5)
            next_layer_grad = self._generate_test_data((batch_size, n_in), minval=-5, maxval=5)
            # тестируем наш слой
            result = self._custom_forward_backward(
                layer_input,
                next_layer_grad,
                custom_layer,
```

```
                return_params_grad=True
            )
            custom_layer_output, custom_layer_grad, custom_params_grad = result
            # тестируем слой на PyTorch
            result = self._torch_forward_backward(
                layer_input,
                next_layer_grad,
                torch_layer,
                return_params_grad=True
            )
            torch_layer_output, torch_layer_grad, torch_params_grad = result
            # сравниваем выходы с точностью atol
            self.assertTrue(np.allclose(torch_layer_output, custom_layer_output, atol=1e-6))
            self.assertTrue(np.allclose(torch_layer_grad, custom_layer_grad, atol=1e-4))
            weight_grad, bias_grad = custom_params_grad[1]
            torch_weight_grad = torch_layer.weight.grad.data.numpy()
            torch_bias_grad = torch_layer.bias.grad.data.numpy()
            self.assertTrue(np.allclose(torch_weight_grad, weight_grad, atol=1e-6))
            self.assertTrue(np.allclose(torch_bias_grad, bias_grad, atol=1e-6))


    def test_Dropout(self):
        np.random.seed(RANDOM_SEED)
        batch_size, n_in = 2, 4
        for _ in range(100):
            # инициализируем слой
            p = np.random.uniform(0.3, 0.7)
            layer = Dropout(p)
            layer.train()
            # формируем тестовые входные тензоры
            layer_input = self._generate_test_data((batch_size, n_in), minval=-5, maxval=5)
            next_layer_grad = self._generate_test_data((batch_size, n_in), minval=-5, maxval=5)
            # тестируем `update_output()`
            layer_output = layer.update_output(layer_input)
            self.assertTrue(np.all(np.logical_or(
                np.isclose(layer_output, 0),
                np.isclose(layer_output*(1.-p), layer_input)
            )))
            # тестируем `update_grad_input()`
            layer_grad = layer.update_grad_input(layer_input, next_layer_grad)
            self.assertTrue(np.all(np.logical_or(
                np.isclose(layer_grad, 0),
                np.isclose(layer_grad*(1.-p), next_layer_grad)
            )))
            # тестируем evaluation режим
            layer.evaluate()
```

```
            layer_output = layer.update_output(layer_input)
            self.assertTrue(np.allclose(layer_output, layer_input))
            # тетсируем маску при нескольких значениях `p`
            p = 0.0
            layer = Dropout(p)
            layer.train()
            layer_output = layer.update_output(layer_input)
            self.assertTrue(np.allclose(layer_output, layer_input))
            p = 0.5
            layer = Dropout(p)
            layer.train()
            layer_input = self._generate_test_data((batch_size, n_in), minval=-5, maxval=5)
            next_layer_grad = self._generate_test_data((batch_size, n_in), minval=-5, maxval=5)
            layer_output = layer.update_output(layer_input)
            zeroed_elem_mask = np.isclose(layer_output, 0)
            layer_grad = layer.update_grad_input(layer_input, next_layer_grad)
            self.assertTrue(np.all(zeroed_elem_mask == np.isclose(layer_grad, 0)))
            # тестируем тот факт, что маска должна генерироваться независимо
            # для каждого элемента входной матрицы, а не для строки/столбца
            batch_size, n_in = 1000, 1
            p = 0.8
            layer = Dropout(p)
            layer.train()
            layer_input = self._generate_test_data((batch_size, n_in), minval=5, maxval=10)
            layer_output = layer.update_output(layer_input)
            self.assertTrue(np.sum(np.isclose(layer_output, 0)) != layer_input.size)
            layer_input = layer_input.T
            layer_output = layer.update_output(layer_input)
            self.assertTrue(np.sum(np.isclose(layer_output, 0)) != layer_input.size)


    def test_ReLU(self):
        np.random.seed(RANDOM_SEED)
        torch.manual_seed(RANDOM_SEED)
        batch_size, n_in = 2, 4
        for _ in range(100):
            # инициализируем слои
            custom_layer = ReLU()
            torch_layer = torch.nn.ReLU()
            # формируем тестовые входные тензоры
            layer_input = self._generate_test_data((batch_size, n_in))
            next_layer_grad = self._generate_test_data((batch_size, n_in))
            # тестируем наш слой
            custom_layer_output, custom_layer_grad = self._custom_forward_backward(
                layer_input,
                next_layer_grad,
```

```
                custom_layer
            )
            # тестируем слой на PyTorch
            torch_layer_output, torch_layer_grad = self._torch_forward_backward(
                layer_input,
                next_layer_grad,
                torch_layer
            )
            # сравниваем выходы с точностью atol
            self.assertTrue(np.allclose(custom_layer_output, torch_layer_output, atol=1e-6))
            self.assertTrue(np.allclose(custom_layer_grad, torch_layer_grad, atol=1e-6))


    def test_LeakyReLU(self):
        np.random.seed(RANDOM_SEED)
        torch.manual_seed(RANDOM_SEED)
        batch_size, n_in = 2, 4
        for _ in range(100):
            # инициализируем слои
            slope = np.random.uniform(0.01, 0.05)
            torch_layer = torch.nn.LeakyReLU(slope)
            custom_layer = LeakyReLU(slope)
            # формируем тестовые входные тензоры
            layer_input = self._generate_test_data((batch_size, n_in))
            next_layer_grad = self._generate_test_data((batch_size, n_in))
            # тестируем наш слой
            custom_layer_output, custom_layer_grad = self._custom_forward_backward(
                layer_input,
                next_layer_grad,
                custom_layer
            )
            # тестируем слой на PyTorch
            torch_layer_output, torch_layer_grad = self._torch_forward_backward(
                layer_input,
                next_layer_grad,
                torch_layer
            )
            # сравниваем выходы с точностью atol
            self.assertTrue(np.allclose(custom_layer_output, torch_layer_output, atol=1e-6))
            self.assertTrue(np.allclose(custom_layer_grad, torch_layer_grad, atol=1e-6))


    def test_ELU(self):
        np.random.seed(RANDOM_SEED)
        torch.manual_seed(RANDOM_SEED)
        batch_size, n_in = 2, 4
```

```
        for _ in range(100):
            # инициализируем слои
            alpha = 1.0
            torch_layer = torch.nn.ELU(alpha)
            custom_layer = ELU(alpha)
            # формируем тестовые входные тензоры
            layer_input = self._generate_test_data((batch_size, n_in))
            next_layer_grad = self._generate_test_data((batch_size, n_in))
            # тестируем наш слой
            custom_layer_output, custom_layer_grad = self._custom_forward_backward(
                layer_input,
                next_layer_grad,
                custom_layer
            )
            # тестируем слой на PyTorch
            torch_layer_output, torch_layer_grad = self._torch_forward_backward(
                layer_input,
                next_layer_grad,
                torch_layer
            )
            # сравниваем выходы с точностью atol
            self.assertTrue(np.allclose(custom_layer_output, torch_layer_output, atol=1e-6))
            self.assertTrue(np.allclose(custom_layer_grad, torch_layer_grad, atol=1e-6))


    def test_SoftPlus(self):
        np.random.seed(RANDOM_SEED)
        torch.manual_seed(RANDOM_SEED)
        batch_size, n_in = 2, 4
        for _ in range(100):
            # инициализируем слои
            custom_layer = SoftPlus()
            torch_layer = torch.nn.Softplus()
            # формируем тестовые входные тензоры
            layer_input = self._generate_test_data((batch_size, n_in))
            next_layer_grad = self._generate_test_data((batch_size, n_in))
            # тестируем наш слой
            custom_layer_output, custom_layer_grad = self._custom_forward_backward(
                layer_input,
                next_layer_grad,
                custom_layer
            )
            # тестируем слой на PyTorch
            torch_layer_output, torch_layer_grad = self._torch_forward_backward(
                layer_input,
                next_layer_grad,
```

```
                torch_layer
            )
            # сравниваем выходы с точностью atol
            self.assertTrue(np.allclose(custom_layer_output, torch_layer_output, atol=1e-6))
            self.assertTrue(np.allclose(custom_layer_grad, torch_layer_grad, atol=1e-6))


    def test_NLLCriterionUnstable(self):
        np.random.seed(RANDOM_SEED)
        torch.manual_seed(RANDOM_SEED)
        batch_size, n_in = 2, 4
        for _ in range(100):
            # инициализируем слои
            torch_layer = torch.nn.NLLLoss()
            custom_layer = NLLCriterionUnstable()
            # формируем тестовые данные
            layer_input = np.random.uniform(0, 1, (batch_size, n_in)).astype(np.float32)
            layer_input /= layer_input.sum(axis=-1, keepdims=True)
            layer_input = layer_input.clip(custom_layer.EPS, 1.-custom_layer.EPS)
            target_labels = np.random.choice(n_in, batch_size)
            target = np.zeros((batch_size, n_in), np.float32)
            target[np.arange(batch_size), target_labels] = 1
            # тестируем `update_output()`
            custom_layer_output = custom_layer.update_output(layer_input, target)
            layer_input_var = torch.from_numpy(layer_input)
            layer_input_var.requires_grad = True
            torch_layer_output_var = torch_layer(
                torch.log(layer_input_var), torch.from_numpy(target_labels).long())
            self.assertTrue(np.allclose(
                torch_layer_output_var.data.numpy(), custom_layer_output, atol=1e-6
            ))
            # тестируем `update_grad_input()`
            custom_layer_grad = custom_layer.update_grad_input(layer_input, target)
            torch_layer_output_var.backward()
            torch_layer_grad_var = layer_input_var.grad
            self.assertTrue(np.allclose(torch_layer_grad_var.data.numpy(), custom_layer_grad, atol=1e-6))

    def test_NLLCriterion(self):
        np.random.seed(RANDOM_SEED)
        torch.manual_seed(RANDOM_SEED)
        batch_size, n_in = 2, 4
        for _ in range(100):
            # инициализируем слои
            torch_layer = torch.nn.NLLLoss()
            custom_layer = NLLCriterion()
            # формируем тестовые данные
```

```
            layer_input = np.random.uniform(-5, 5, (batch_size, n_in)).astype(np.float32)
            layer_input = torch.nn.LogSoftmax(dim=1)(torch.from_numpy(layer_input)).data.numpy()
            target_labels = np.random.choice(n_in, batch_size)
            target = np.zeros((batch_size, n_in), np.float32)
            target[np.arange(batch_size), target_labels] = 1  # one-hot encoding
            # тестируем `update_output()`
            custom_layer_output = custom_layer.update_output(layer_input, target)
            layer_input_var = torch.from_numpy(layer_input)
            layer_input_var.requires_grad = True
            torch_layer_output_var = torch_layer(layer_input_var, torch.from_numpy(target_labels).long())
            self.assertTrue(np.allclose(
                torch_layer_output_var.data.numpy(), custom_layer_output, atol=1e-6
            ))
            # тестируем `update_grad_input()`
            custom_layer_grad = custom_layer.update_grad_input(layer_input, target)
            torch_layer_output_var.backward()
            torch_layer_grad_var = layer_input_var.grad
            self.assertTrue(np.allclose(torch_layer_grad_var.data.numpy(), custom_layer_grad, atol=1e-6))


suite = unittest.TestLoader().loadTestsFromTestCase(TestLayers)
unittest.TextTestRunner(verbosity=2).run(suite)
```