Вопрос Инфо

Вопрос 1

Выполнен

Баллов: 1,00 из 1,00

В нашем примере мы хотим по координате x предсказать y. При этом у нас есть выборка, которая была сгенерирована следующим образом: $y = sin(x) + noise$

Отметьте верные утверждения:

Выберите один или несколько ответов:

- [x] a. Задача регрессии – предсказание вещественного числа
- [x] b. Метод unsqueeze_ добавляет тензору еще одну размерность
- [x] c. Мы применили метод unsqueeze_, так как хотим, чтобы каждый элемент был вектором (пусть и из одного числа)
- [ ] d. Задача регрессии – предсказание класса объекта
- [ ] e. Мы применили метод unsqueeze_, чтобы поместить x и y в единый тензор

Ваш ответ верный.

Вопрос Инфо

Вопрос 2

Выполнен

Баллов: 1,00 из 1,00

В нашем примере мы поделили датасет на 2 части train и validation, но правильнее использовать деление на 3 части: train, validation, test.

Для чего нужно такое деление?

PS: если вопрос вам кажется непонятным, пересмотрите лекцию из второго урока

PPS: гиперпараметрами в ответах называется то, что не настраивается непосредсвенно градиентным спуском, а, напротив, выбирается человеком. Это архитектура нейронной сети, выбор loss-функции, метод градиентного спуска, learning rate, количество эпох в обучении и прочее.

Выберите один или несколько ответов:

- a. Test нужен для финальной оценки качества работы сети
- b. Validation нужна, чтобы подобрать гиперпараметры сети
- c. Train нужен для обучения
- d. В процессе экспериментов с сетью мы могли "подогнать" гиперпараметры под данные в validation. Значит, чтобы получить "честное" качество, нам нужен test
- e. Если бы мы обучались на полном датасете и на нем же валидировались, то не заметили бы переобучения (когда сетка "запомнила" выборку)
- f. Validation нужна, чтобы посмотреть качество работы сетки на данных, которых не было в обучении

Ваш ответ верный.

Вопрос 3

Верно

Баллов: 1,00 из 1,00

Давайте попрактикуемся с SineNet:

1) Добавим еще один fc-слой

2) Заменим активацию между слоями на гиперболический тангенс

Ответ: (штрафной режим: 0 %)

Сброс ответа

```
import torch


class SineNet(torch.nn.Module):
    def __init__(self, n_hidden_neurons):
        super().__init__()
        self.fc1 = torch.nn.Linear(1, n_hidden_neurons)
        self.act1 = torch.nn.Tanh()
        self.fc2 = torch.nn.Linear(n_hidden_neurons, n_hidden_neurons)
        self.act2 = torch.nn.Tanh()
        self.fc3 = torch.nn.Linear(n_hidden_neurons, 1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act1(x)
        x = self.fc2(x)
        x = self.act2(x)
        x = self.fc3(x)
        return x


sine_net = SineNet(int(input()))
sine_net.forward(torch.Tensor([1.]))

print(sine_net)

```

Прошли все тесты! ✔

Верно

Баллы за эту попытку: 1,00/1,00.

Вопрос 4

Выполнен

Баллов: 1,00 из 1,00

Сопоставьте конец строки в соответствии с началом:

sine_net =	SineNet(50)
optimizer =	torch.optim.Adam(sine_net.parameters(), lr=0.01)
optimizer	.zero_grad()
y_pred =	sine_net.forward(x_train)
loss_val = loss	(y_pred, y_train)
loss_val	.backward()
optimizer.	step()

Ваш ответ верный.

Вопрос 5

Выполнен

Баллов: 1,00 из 1,00

Что такое "эпоха"?

Выберите один ответ:

- a. Итерация обучения на всем датасете
- b. Обучение на одном объекте
- c. Итерация градиентного спуска

Ваш ответ верный.

Вопрос 6

Верно

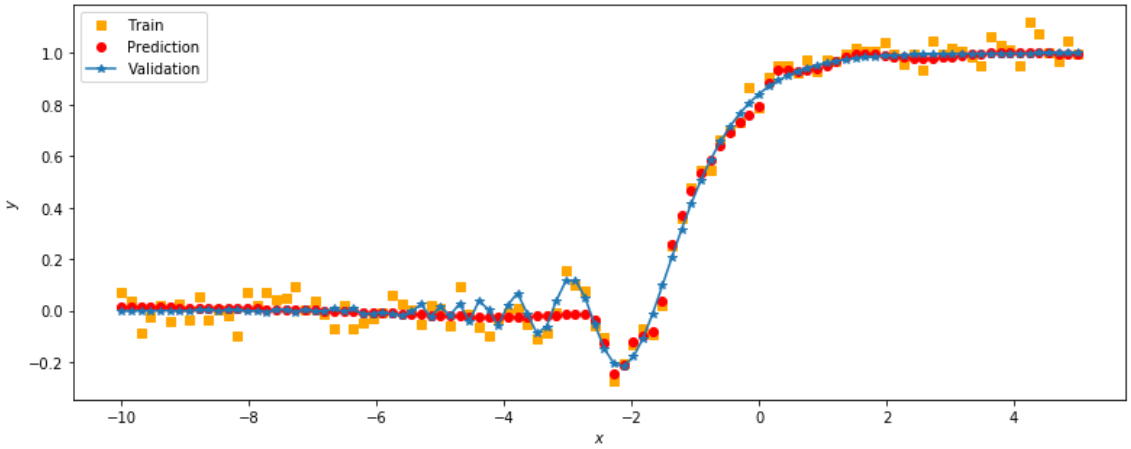
Баллов: 4,00 из 4,00

Обучим нейронную сеть для задачи регрессии:

Возьмем более сложную функцию в качестве таргета: $y = 2^x sin(2^{-x})$.

Кроме того, мы хотим получить хорошую метрику MAE на валидации: $MAE = \frac{1}{l}\sum_{i=1}^{l}|y\_pred_i - y\_target_i|$, тогда как знакомая нам MSE выглядит как $MSE = \frac{1}{l}\sum_{i=1}^{l}(y\_pred_i - y\_target_i)^2$

Вот пример того, как нейросеть может отрабатывать на данной функции:



Данный пример показывает MAE на валидации ~0.021 . Получите метрику не хуже 0.03

Что можно варьировать:

1) Архитектуру сети

2) loss-функцию

3) lr оптимизатора

4) Количество эпох в обучении

Удачи!

Ответ: (штрафной режим: 0 %)

Сброс ответа

```
import torch

def target_function(x):
    return 2**x * torch.sin(2**-x)

class RegressionNet(torch.nn.Module):
    def __init__(self, n_hidden_neurons):
        super(RegressionNet, self).__init__()
        self.fc1 = torch.nn.Linear(1, n_hidden_neurons)
        self.act1 = torch.nn.Sigmoid()
        self.fc2 = torch.nn.Linear(n_hidden_neurons, 1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.act1(x)
        x = self.fc2(x)
        return x

net = RegressionNet(20)

# ------Dataset preparation start--------:
x_train =  torch.linspace(-10, 5, 100)
y_train = target_function(x_train)
noise = torch.randn(y_train.shape) / 20.
y_train = y_train + noise
x_train.unsqueeze_(1)
y_train.unsqueeze_(1)

x_validation = torch.linspace(-10, 5, 100)
y_validation = target_function(x_validation)
x_validation.unsqueeze_(1)
```

```
x_validation.unsqueeze_(1)
y_validation.unsqueeze_(1)
# ------Dataset preparation end--------:


optimizer = torch.optim.Adam(net.parameters(), lr=0.01)

def loss(pred, target):
    squares = (pred - target) ** 2
    return squares.mean()

for epoch_index in range(2000):
    optimizer.zero_grad()

    y_pred = net.forward(x_train)
    loss_value = loss(y_pred, y_train)
    loss_value.backward()
    optimizer.step()

# Проверка осуществляется вызовом кода:
# def metric(pred, target):
#    return (pred - target).abs().mean()
```

Прошли все тесты! ✔

Верно

Баллы за эту попытку: 4,00/4,00.

◄ 3.4 Задачи по теме: Функции потерь

Перейти на...

3.6 Семинар: Классификация в PyTorch ►