Основы практического использования нейронных сетей.

Лекция 5. Аппроксимация квадратичной функции.

Дмитрий Буряк.
к.ф.-м.н.
dyb04@yandex.ru

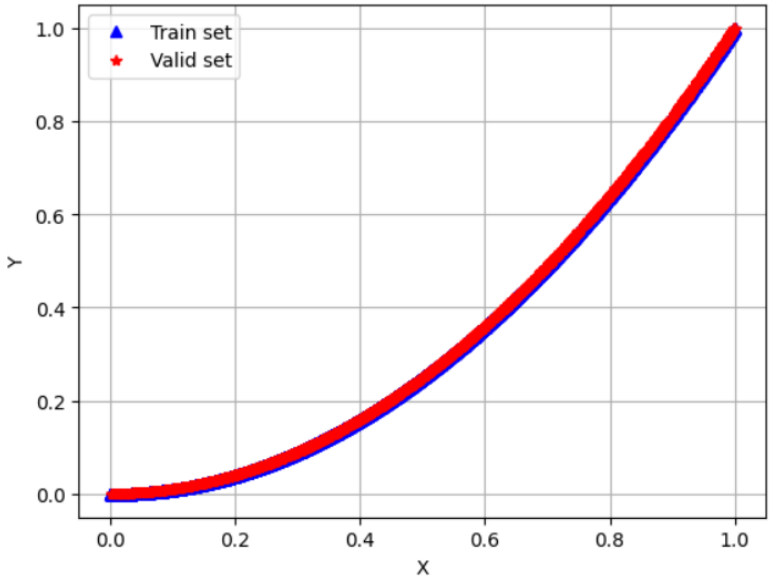
Задача

Обучить НС для аппроксимации квадратичной функции

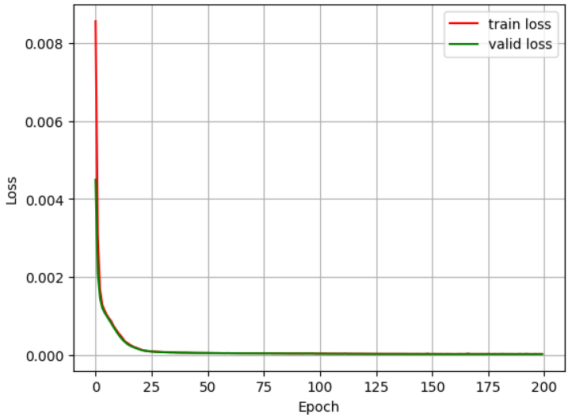
$y = x^2, x \in (0,1)$

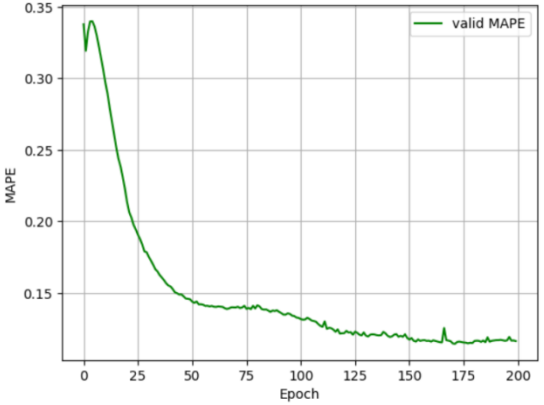
- Обучающая выборка: 8000
- Валидационная выборка: 2000
- Шума нет
- Целевая метрика: MAPE

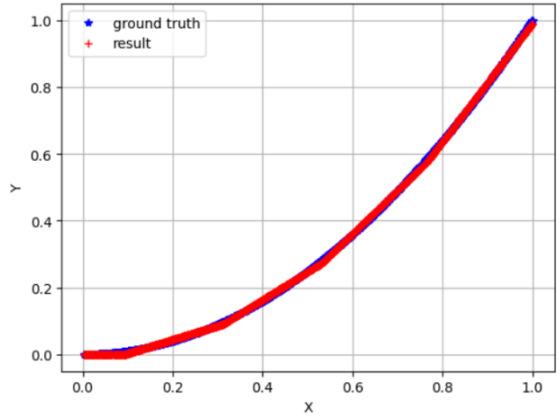
$$MAPE = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{Y_i - \hat{Y}_i}{Y_i} \right|$$

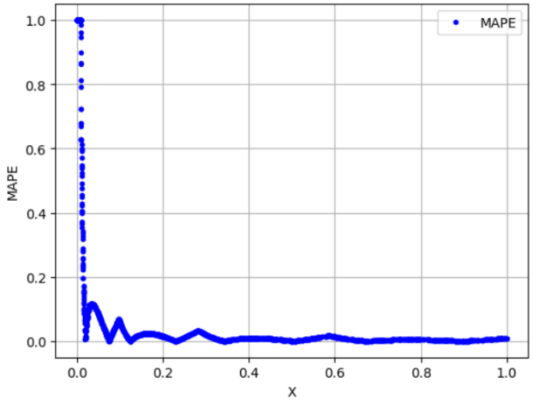


Стандартное обучение (2 слоя)







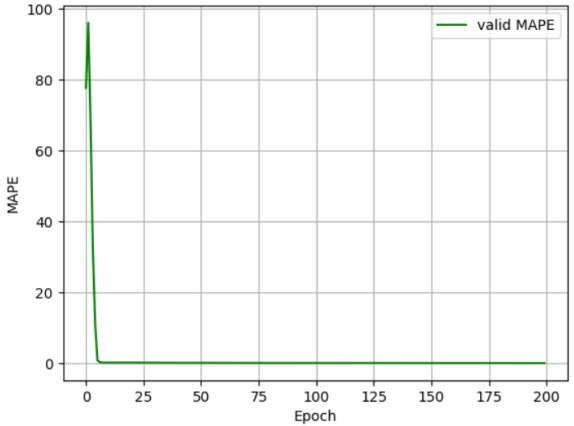


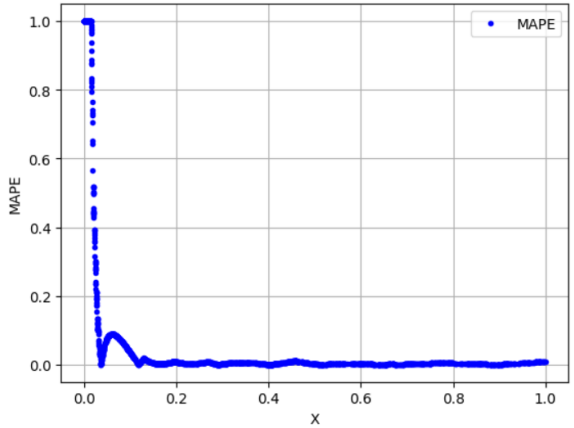
- 2 слоя: 1 – 50 – 1
- Relu

- batch=128
- epoch=200
- Adam (lr=0.001)
- без изменения lr
- Loss: MSE

mse: 1.5783e-05
rmse: 0.0040
MAPE: 0.1143

Стандартное обучение (2слоя)



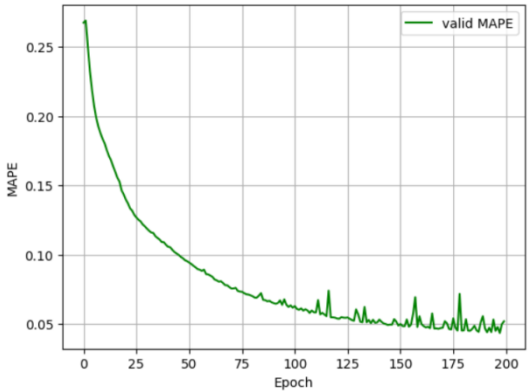


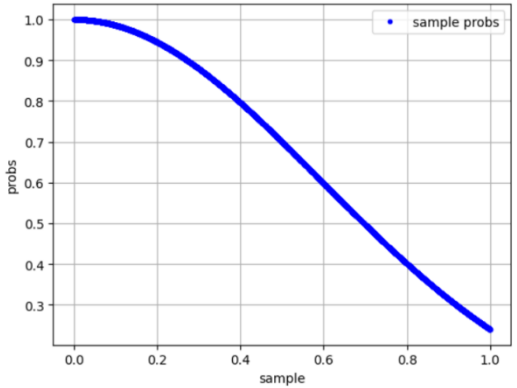
- **batch=512**
- epoch=200
- Adam (lr=0.001)
- без изменения lr
- Loss: MSE

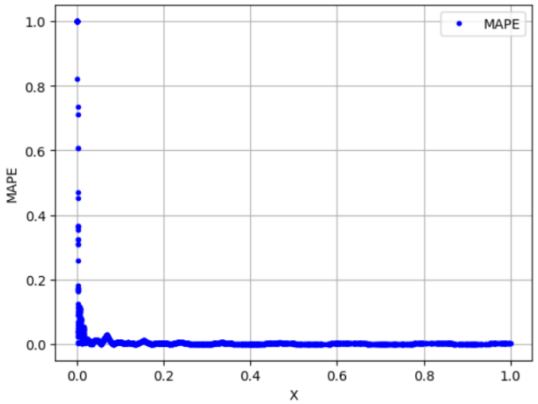
mse: 1.5955e-05
rmse: 0.0040
MAPE: 0.1415

Увеличилась скорость сходимости, но MAPE вырос

Стандартное обучение (2слоя)





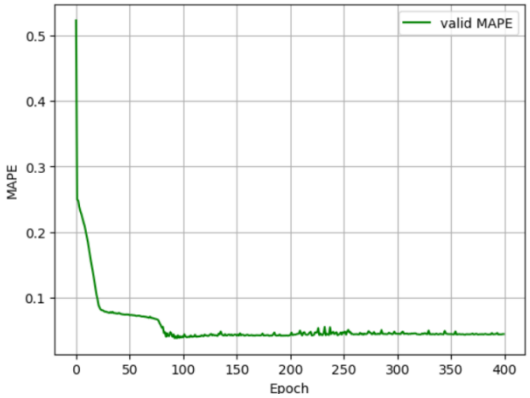


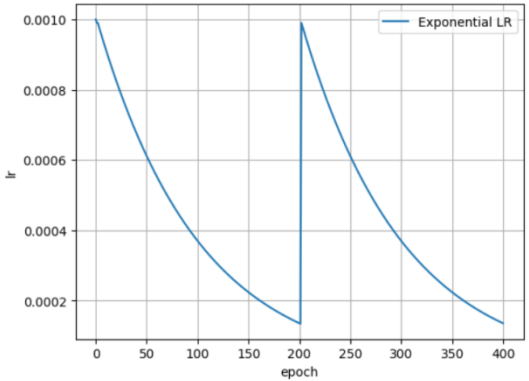
- **Сэмплирование пакетов**
- batch=128
- epoch=200
- Adam (lr=0.001)
- без изменения lr
- Loss: MSE

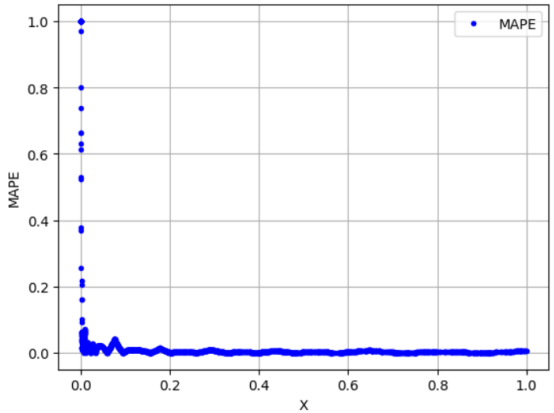
mse: 8.3055e-07
rmse: 0.0009
MAPE: 0.0434

MAPE уменьшилось в ~3 раза

Стандартное обучение (2слоя)





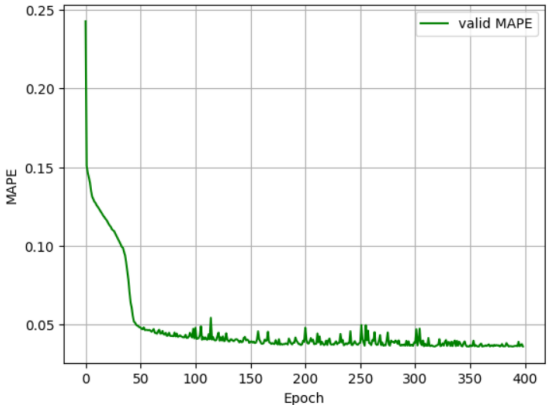


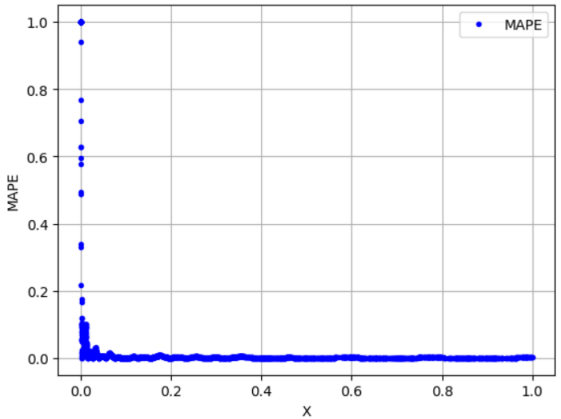
- **Изменение LR**
- Сэмплирование пакетов
- batch=128
- **epoch=400**
- Adam (начальный lr=0.001)
- Loss: MSE

mse: 2.0879e-06
rmse: 0.0014
MAPE: 0.0377

MAPE уменьшилось 0.006

Стандартное обучение (3 слоя)





- **3 слоя: 1 – 40 – 10 – 1**
- Relu
- Изменение LR
- Сэмплирование пакетов
- batch=128
- epoch=400
- Adam (начальный lr=0.001)
- Loss: MSE

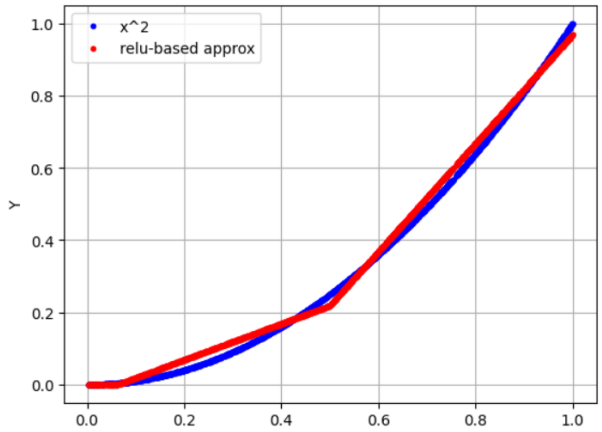
mse: 5.5170e-07
rmse: 0.0007
MAPE: 0.0362

MAPE уменьшилось на 0.001

Аналитическое решение

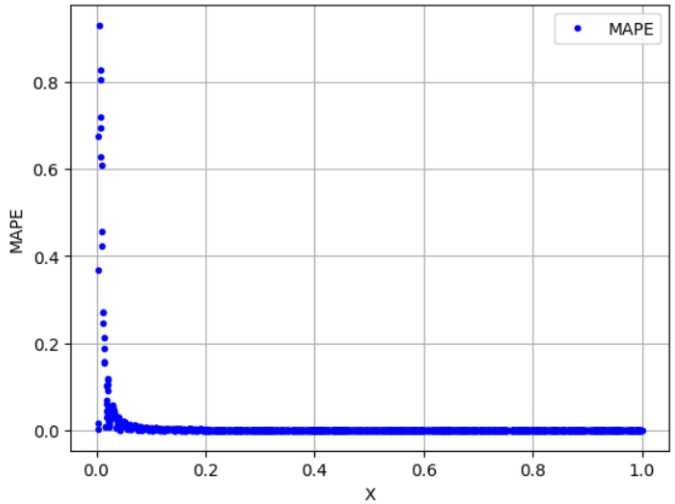
```
self.xvalues = np.arange(0,1+0.0001,0.5) # узлы 0, 0.5, 1
self.l1 = nn.Linear(1, len(self.xvalues )-1)
self.l2 = nn.Linear(len(self.xvalues )-1, 1)
a_prev, b_prev = 0, 0
for i, x in enumerate(self.xvalues[:-1]):
        dx = self.xvalues[i+1] - self.xvalues[i]
        dy = yvalues[i+1] - yvalues[i]
        a = dy/dx
        b = yvalues[i] - a*x
        err_shift = (yvalues[i+1] + yvalues[i]) / 2 - yfunc((self.xvalues[i+1] + self.xvalues[i]) / 2)
        b = b - err_shift / 2
        with torch.no_grad():
           self.l1.weight[i, 0] = a - a_prev
           self.l1.bias[i] = b - b_prev
        a_prev = a
        b_prev = b
with torch.no_grad():
        self.l2.weight[0, :] = 1
        self.l2.bias[0] = 0
```

```
def forward(self, x):
      x = self.l1(x)
      x = nn.ReLU()(x)
      x = self.l2(x)
      return x
```



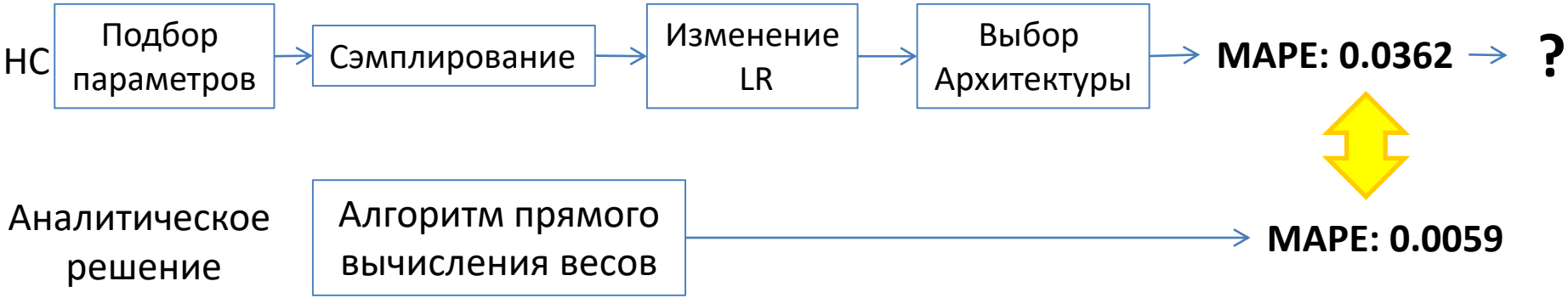
Узлы: 0, 0.5, 1

- **2 слоя: 1 – 50 – 1**
- Relu



MAPE: 0.0059

НС vs Аналитическое решение



Код аналитического решения

```
class XFuncModule(nn.Module):
    def __init__(self, xvalues, yfunc):
        super().__init__()
        assert all(xvalues == sorted(xvalues)), 'xvalues must be sorted'
        self.xvalues = xvalues
        self.l1 = nn.Linear(1, self.steps-1)
        self.l2 = nn.Linear(self.steps-1, 1)
        self.set_weights(yfunc)
    def set_weights(self, yfunc):
        yvalues = yfunc(self.xvalues)
        a_prev, b_prev = 0, 0
        for i, x in enumerate(self.xvalues[:-1]):
            dx = self.xvalues[i+1] - self.xvalues[i]
            dy = yvalues[i+1] - yvalues[i]
            a = dy/dx
            b = yvalues[i] - a*x
            err_shift = (yvalues[i+1] + yvalues[i]) / 2 - yfunc((self.xvalues[i+1] + self.xvalues[i]) / 2)
            b = b - err_shift / 2
            with torch.no_grad():
                self.l1.weight[i, 0] = a - a_prev
                self.l1.bias[i] = b - b_prev
            a_prev = a
            b_prev = b
        with torch.no_grad():
            self.l2.weight[0, :] = 1
            self.l2.bias[0] = 0
```

```
    @property
    def steps(self):
        return len(self.xvalues)
    def forward(self, x):
        x = self.l1(x)
        x = nn.ReLU()(x)
        x = self.l2(x)
        return x
```

```
yfunc = lambda x: x**2
interp_range = np.arange(0,1+0.0001,0.02)

x2m = XFuncModule(interp_range , yfunc)

x_val = torch.rand(2000, 1)
y_val=x_val*x_val

res_model = x2m(x_val).detach().numpy().flatten()
res_true=y_val.numpy()[:,0]

mape_val=np.mean(np.abs((res_true - res_model) / res_true))

print('MAPE:',mape_val)
```