

Нейронные сети и их практическое применение.

Лекция 6.
Практические рекомендации для обучения НС.

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

Формирование обучающей выборки

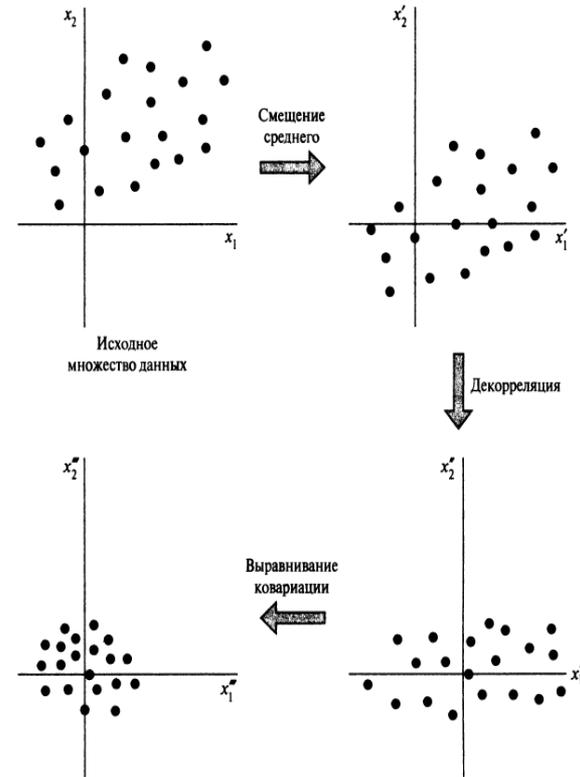
- ❑ Размер обучающей выборки
 - $N=O(|W|)$, W – множество весов. «Чем больше, тем лучше»
- ❑ Состав
 - Репрезентативность
 - Согласованность с валидационной и тестовой выборками
 - Аугментация данных:
 - добавление шумов
 - искажение данных
 - модели для генерации синтетических данных

Формирование пакетов при обучении

- Пакет – часть обучающей выборки, по которой вычисляется коррекция весов на текущем шаге.
- Принцип максимизации информативности
 - использование примеров, вызывающих наибольшие ошибки обучения
 - использование кардинально отличающихся примеров
- Случайный порядок следования примеров
- Схема акцентирования
- Коррекция распределения данных в обучающей выборке

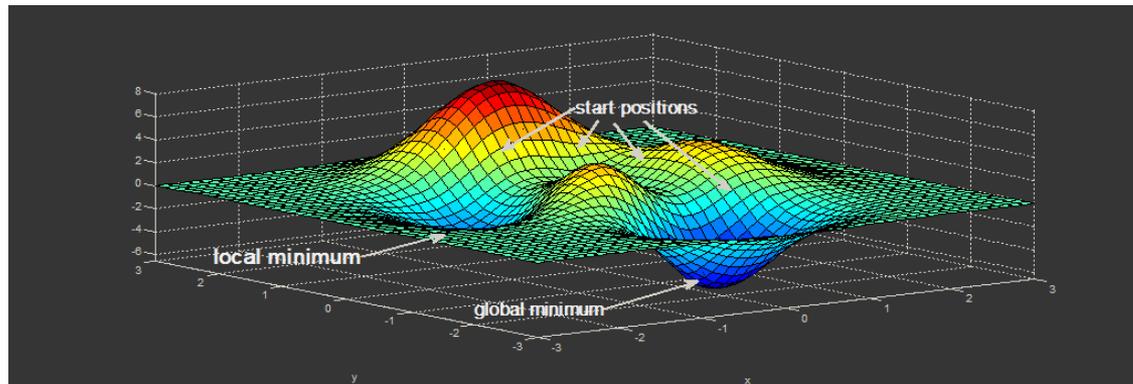
Предобработка входных данных

- Целевые значения должны находится в области значений функции активации
- Нормировка входов
 - нулевое среднее
 - отсутствие корреляции
 - одинаковая ковариация



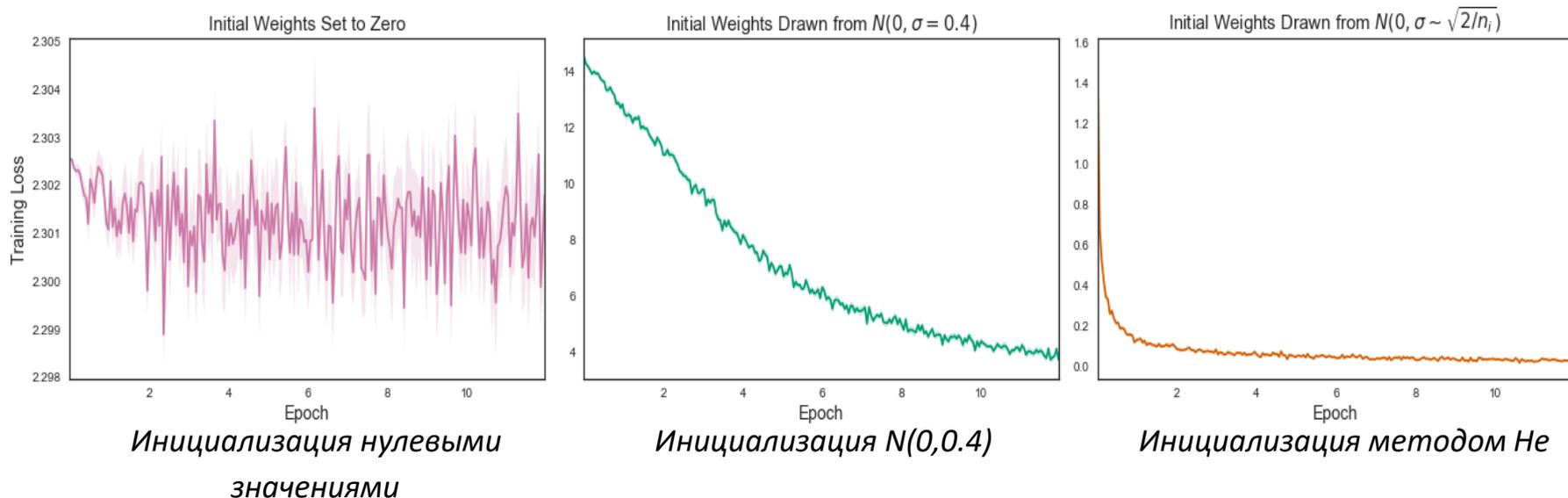
Инициализация весов

- ❑ Инициализация весов – определение начальной точки для процесса оптимизации функции ошибки.
- ❑ Обеспечение ассиметричности выходов нейронов и «динамического» диапазона функции активации.



Влияние на эффективность обучения

Классификация изображений MNIST



Источник: <https://intoli.com/blog/neural-network-initialization/>

Методы инициализации

❑ Инициализировать нулями нельзя.

❑ Небольшими случайными числами.

❑ $w \in N(0, \frac{1}{\sqrt{n}}) \Rightarrow u = \sum_1^n w_i x_i \in N(0,1)$;

❑ $w \in N(0, \frac{1}{\sqrt{n_{in} + n_{out}}})$ (Glorot et al., Understanding the difficulty of training deep feedforward neural networks);

❑ $w \in N(0, \sqrt{\frac{2}{n}})$ для ReLU (He et al., Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification);

❑ Разреженная (sparse) инициализация ;

❑ Инициализация смещений – нулевые значения.

Инициализация весов. Обоснование

- Входы: нулевое среднее, единичная дисперсия, некоррелированы:

$$E[y_i y_k] = \begin{cases} 1 & \text{для } k = i \\ 0 & \text{для } k \neq i \end{cases}$$

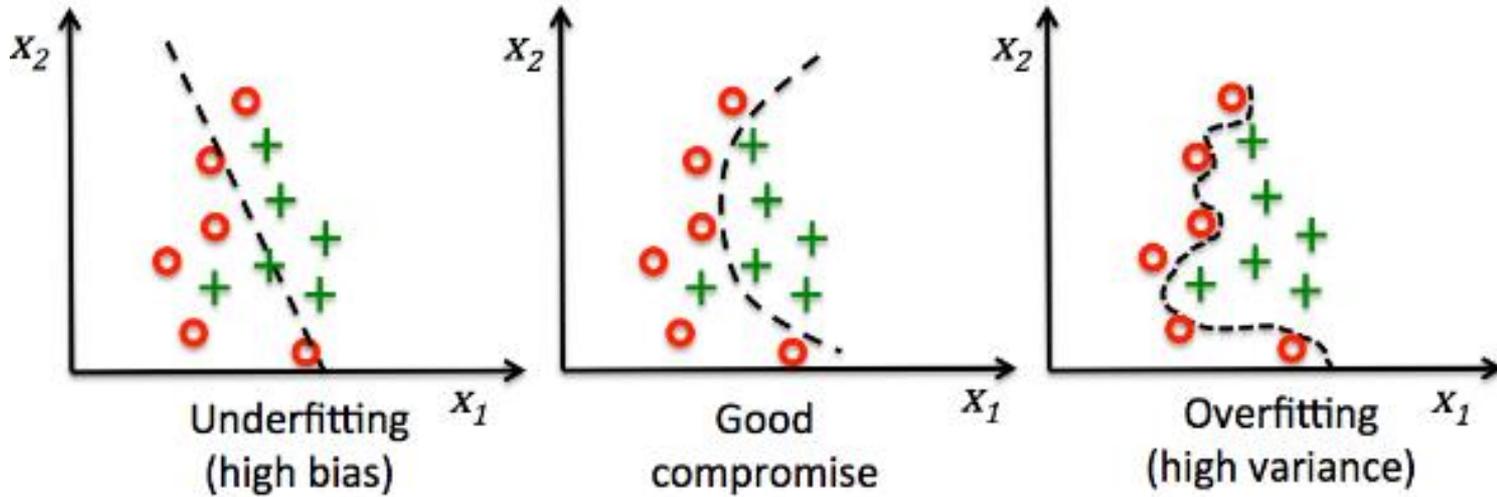
- Начальные синаптические веса: равномерно распределены, нулевое среднее. Определить дисперсию.
- Среднее и дисперсия линейной комбинации нейронов

$$\mu_v = E[v_j] = E\left[\sum_{i=1}^m w_{ji} y_i\right] = \sum_{i=1}^m E[w_{ji}] E[y_i] = 0,$$

$$\begin{aligned} \sigma_v^2 &= E[(v_j - \mu_v)^2] = E[v_j^2] = E\left[\sum_{i=1}^m \sum_{k=1}^m w_{ji} w_{jk} y_i y_k\right] = \\ &= \sum_{i=1}^m \sum_{k=1}^m E[w_{ji} w_{jk}] E[y_i y_k] = \sum_{i=1}^m E[w_{ji}^2] = m \sigma_w^2, \end{aligned}$$

- Если $\sigma_w = m^{-1/2}$ то $\sigma_v = 1$

Проблема переобучения НС



Необходимо контролировать процесс обучения.

Обучающая и подтверждающая выборки

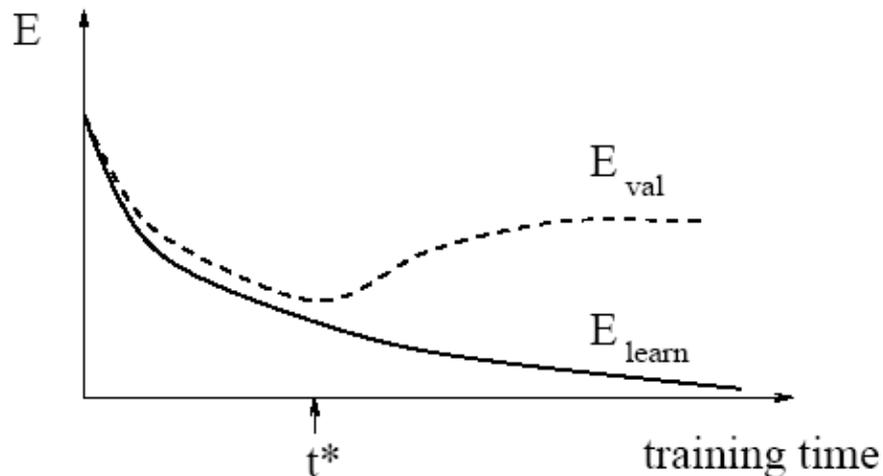
S_L – обучающая выборка;

E_{learn} – ошибка на обучающей выборке;

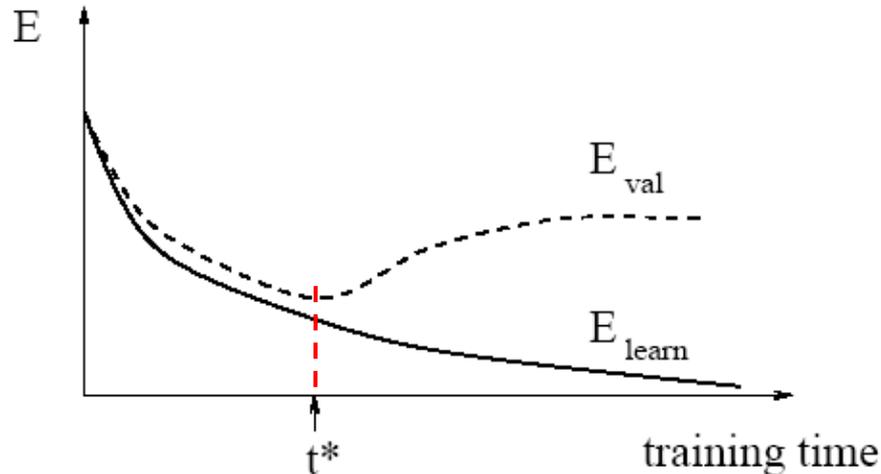
S_V – подтверждающая выборка;

E_{val} – ошибка на подтверждающей выборке;

Необходимо, чтобы E_{learn} и E_{val} в конце обучения достигли минимума



Переобучение. Обучение с ранним остановом



Точка раннего останова

❑ Если размер обучающей выборки много больше числа весов НС, эффективность применения обучения с ранним остановом падает.

Переобучение. Регуляризация

❑ Регуляризация - метод предотвращения переобучения НС.

❑ Введение штрафа для больших весов.

❑ λ - коэффициент регуляризации.

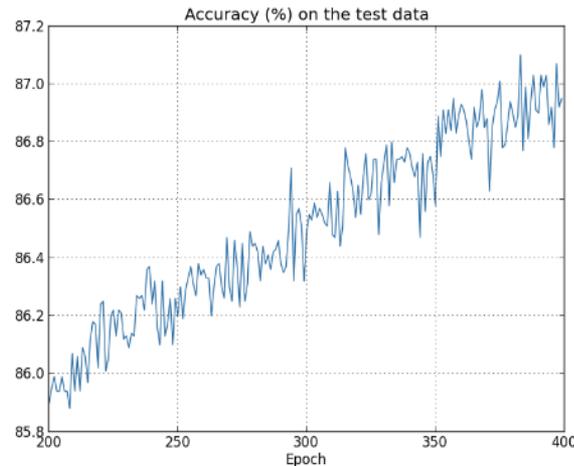
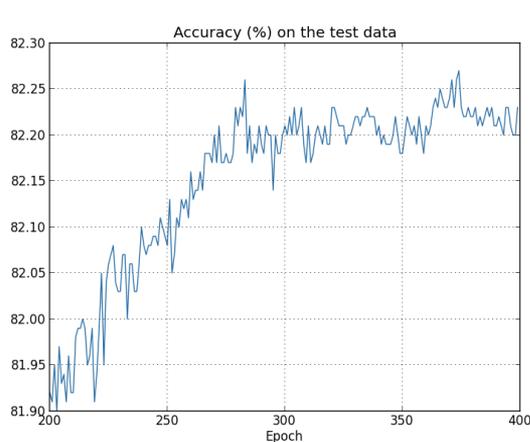
❑ Для смещений регуляризация также может быть применена

$$E = E_0 + \frac{\lambda}{2n} \sum_w w^2 \quad \text{Регуляризация L2}$$

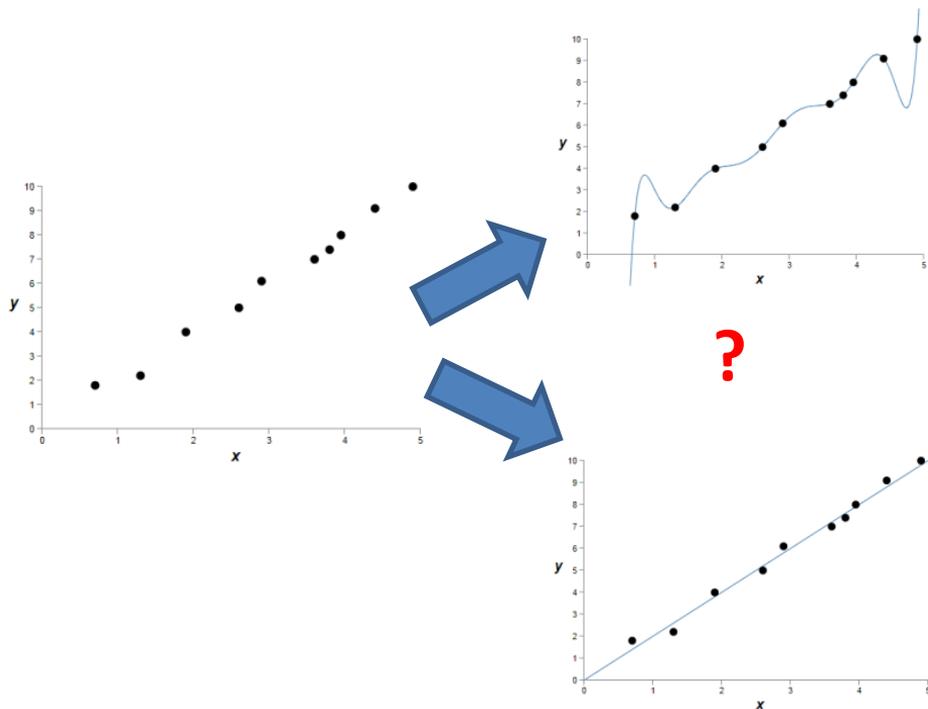
❑ Классификация MNIST

❑ Сеть 784x30x10, 1000

обучающих примеров



Регуляризация → снижение переобучения



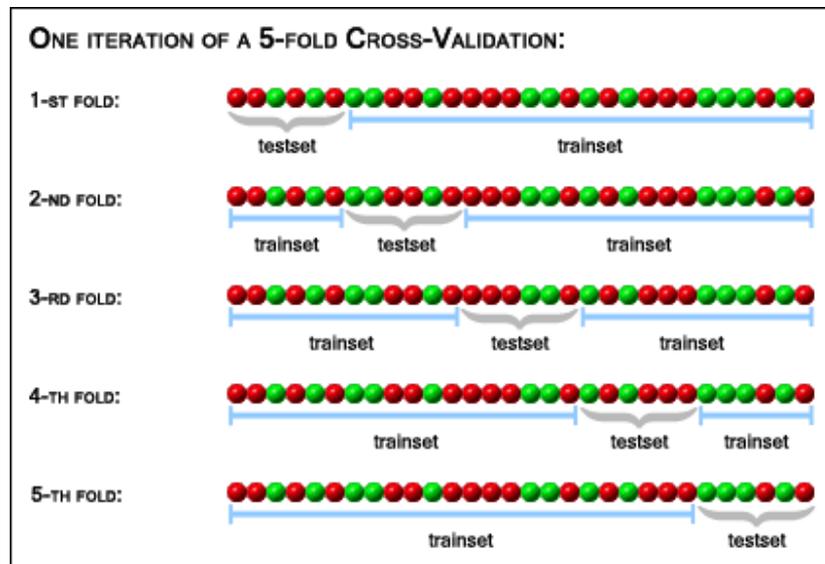
- ❑ Нет однозначного решения без дополнительной информации.
- ❑ Большие значения параметров → увеличение чувствительности к шуму.

$$y = a_0x^9 + a_1x^8 + \dots$$

$$y = a_0x + a_1$$

Многократная перекрестная проверка

- Перекрестная проверка – выделение проверочного множества из обучающего. N – размер обучающего множества $\rightarrow r*N$ – размер проверочного подмножества ($r=0.2$)
- Делим N примеров на K подмножеств
- Обучаем на $K-1$ подмножестве, тестируем на оставшемся. Повторяем K раз.
- Вычисляем среднюю ошибку по всем циклам
- Если N мало, то $K=N$.
- Применение: выбор оптимальной архитектуры и параметров обучения.
- Финальная сеть обучается на всем обучающем множестве с выбранными параметрами.



Основные причины низкой эффективности НС

- Низкая эффективность = большая ошибка на тестовых данных.
- Проблемы с данными;
- Несоответствие архитектуры НС сложности задачи;
- Неоптимальные значения гиперпараметров;
- Переобучение;
- Ошибки в реализации.

Обозначения

- ❑ S_{train} – обучающая выборка; S_{test} – тестовая выборка;
- ❑ E_{train} – ошибка на обучающей выборке, E_{test} – ошибка на тестовой выборке, E_{goal} – целевое значение ошибки.

- ❑ $E_{test} > E_{goal}$

Анализ ошибки на обучающей выборке

$E_{train} > E_{goal}$

- Увеличить размер НС;
- Улучшить алгоритм обучения
- Оптимизировать значения гиперпараметров алгоритма обучения
- Анализ качества исходных данных
 - низкое значение сигнал-шум;
 - ошибки алгоритма предобработки;
 - недостоверные референсные значения;
 - несбалансированная выборка.

Анализ ошибки на тестовой выборке

$E_{train} < E_{goal}$ и $E_{test} > E_{goal}$

Увеличить размер S_{train} ;

Уменьшить размер НС;

Оптимизировать значения гиперпараметров НС (регуляризация);

Подбор алгоритма обучения;

Несоответствие S_{train} и S_{test} .

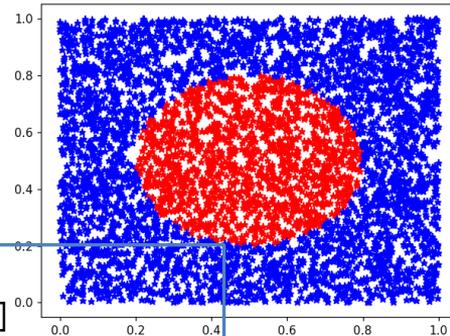
Классификация данных при помощи MLP (1)

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.autograd import Variable
from matplotlib import pyplot as plt
import numpy as np
```

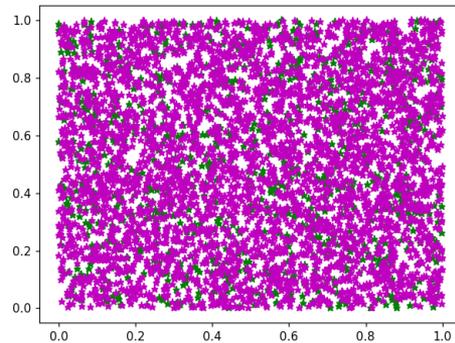
```
data_size = 5000
x = torch.rand(data_size)
y = torch.rand(data_size)
center = (0.5,0.5)
radius = 0.3
distance = torch.sqrt( (center[0]-x)**2+(center[1]-y)**2 )
pos_labels = distance <= radius
neg_labels = distance > radius
labels = torch.zeros(data_size).long()
labels[pos_labels] = 1
```

MLP (1)

```
pos_data_x,pos_data_y = x[pos_labels],y[pos_labels]
neg_data_x,neg_data_y = x[neg_labels],y[neg_labels]
plt.figure("All data")
plt.plot(pos_data_x.numpy(),pos_data_y.numpy(),"r",marker="*",lw=0)
plt.plot(neg_data_x.numpy(),neg_data_y.numpy(),"b",marker="*",lw=0)
```



```
N_train = int(data_size *0.8)
train_data_x = x[:N_train]
train_data_y = y[:N_train]
train_labels = labels[:N_train]
test_data_x = x[N_train:]
test_data_y = y[N_train:]
test_labels = labels[N_train:]
plt.figure("Train and test data")
plt.plot(test_data_x.numpy(),test_data_y.numpy(),"g",marker="*",lw=0)
plt.plot(train_data_x.numpy(),train_data_y.numpy(),"m",marker="*",lw=0)
plt.show()
```



Классификация данных при помощи MLP (2)

```
class MLP(nn.Module):
    def __init__(self,size,out_size):
        super(MLP, self).__init__()
        self.mlp1 = nn.Linear(size, 15)
        self.mlp2 = nn.Linear(15,out_size)
        self.func = nn.ReLU()

    def forward(self, x):
        x = self.func(self.mlp1(x))
        return self.mlp2(x)

    def output_to_label(output,softmax):
        with torch.no_grad():
            output2=softmax(output.detach())
            rez=torch.sort(output2.cpu(),1,True)
            ret_labels = []
            for i in range(rez[1].size(0)):
                ind=rez[1].data[i][0]
                lab=int(ind)
                ret_labels+=[lab]
            return ret_labels
```

```
train_data_all = torch.cat([train_data_x.unsqueeze(1),train_data_y.unsqueeze(1)],dim=1)
test_data_all = torch.cat([test_data_x.unsqueeze(1),test_data_y.unsqueeze(1)],dim=1)

model = MLP(2,2)

lr = 0.001
# Defines a optimizer to update the parameters
optimizer = optim.Adam(model.parameters(), lr=lr)
model_loss = nn.CrossEntropyLoss()
softmax = nn.Softmax(-1)

batch_size = 32
n_iterations = train_data_all.size(0) // batch_size
print("n iter:",n_iterations)
train_errors = []
test_errors = []
```

Классификация данных при помощи MLP (3)

```
for epoch in range(50):
    print("### epoch:", epoch)
    epoch_loss = 0
    error = 0
    for iter in range(n_iterations):
        batch_idx = (torch.rand(batch_size)*train_data_all.size(0)).long()
        batch = Variable(train_data_all[batch_idx].contiguous(), requires_grad = True)
        #batch.requires_grad = True
        ref = train_labels[batch_idx]

        out = model(batch)

        loss = model_loss(out, ref)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

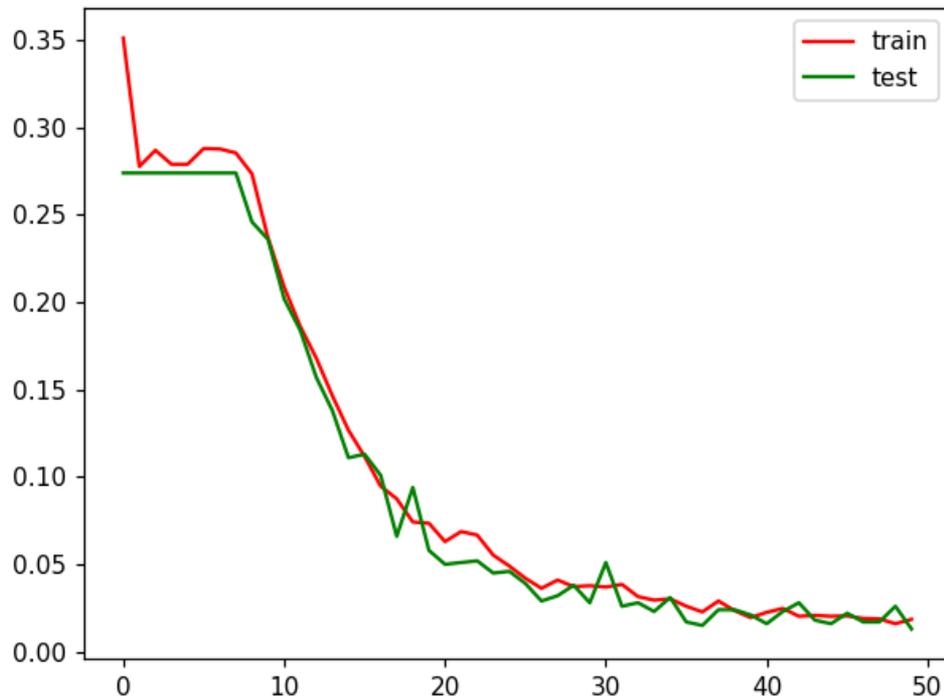
        lab = torch.Tensor(output_to_label(out, softmax)).long()
        error += torch.sum(torch.abs(ref-lab))
        epoch_loss += loss.item()
```

```
# Continue for (epoch)
train_errors += [ [epoch, error.item()/(n_iterations*batch_size)] ]
print("train error:", error, 'of', (n_iterations*batch_size))
#tests
if (epoch%1 == 0):
    test_error = 0
    with torch.no_grad():
        batch = test_data_all
        out = model(batch)
        lab = torch.Tensor(output_to_label(out, softmax)).long()
        test_error = torch.sum(torch.abs(test_labels.long()-lab))
        test_errors += [ [epoch, test_error.item()/test_labels.size(0)] ]
        print("test error:", test_error, " of ", test_labels.size(0))
    print("epoch loss:", epoch_loss/n_iterations)

plt.figure("Errors")
train_errors = np.array(train_errors)
test_errors = np.array(test_errors)
plt.plot(train_errors[:,0], train_errors[:,1], "r-", label = "train")
plt.plot(test_errors[:,0], test_errors[:,1], "g-", label = "test")
plt.legend()
plt.show()
```

Классификация данных при помощи MLP (4)

```
...  
### epoch: 46  
train error: tensor(76) of 4000  
test error: tensor(17) of 1000  
epoch loss: 0.12735117292404174  
### epoch: 47  
train error: tensor(75) of 4000  
test error: tensor(17) of 1000  
epoch loss: 0.11666139790415764  
### epoch: 48  
train error: tensor(64) of 4000  
test error: tensor(26) of 1000  
epoch loss: 0.11834380742907524  
### epoch: 49  
train error: tensor(74) of 4000  
test error: tensor(13) of 1000  
epoch loss: 0.1205455330312252
```



Вопросы

1. Основные принципы формирования обучающей выборки.
2. Основные действия для предобработки данных
3. Для чего используют многократную перекрестную проверку?
4. [Практика]
 - Реализовать однослойную сеть для решения задачи из примера. Сравнить графики сходимости и ошибку классификации для двуслойной и однослойной сетей
 - Задача классификации на 3 класса:
 - . Пространство входных данных: квадрат $[0,1] \times [0,1]$
 - . Класс 1: точки внутри окружности с центром $(0.2, 0.2)$ и радиусом 0.15
 - . Класс 2: точки внутри окружности с центром $(0.7, 0.7)$ и радиусом 0.2
 - . Класс 3: все остальное
 - Реализовать НС для решения задачи классификации на 3 класса:
 - . Попробовать разное количество внутренних слоев, сравнить точность
 - . Сравнить графики сходимости для разных оптимизаторов