Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Нейроинформатика»

Студент: К. О. Вахрамян
Преподаватель: Н. П. Аносова
Группа: М8О-406Б
Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №3

Многослойные сети. Алгоритм обратного распространения ошибки

Задача: Целью работы является исследование свойств многослойной нейронной сети прямого рас- пространения и алгоритмов ее обучения, применение сети в задачах классификации и аппрокси- мации функции.

Основные этапы работы:

1. Использовать многослойную нейронную сеть для классификации точек в случае, когда классы не являются линейно разделимыми.
2. Использовать многослойную нейронную сеть для аппроксимации функции. Произвести обучение с помощью одного из методов первого порядка.
3. Использовать многослойную нейронную сеть для аппроксимации функции. Произвести обучение с помощью одного из методов второго порядка.

Вариант :2

2.	Эллипс: $a = 0.4$, $b = 0.15$, $\alpha = \pi/6$, $x_0 = 0$, $y_0 = 0$ Эллипс: $a = 0.7$, $b = 0.5$, $\alpha = 0$, $x_0 = 0$, $y_0 = 0$ Эллипс: $a = 1$, $b = 1$, $\alpha = 0$, $x_0 = 0$, $y_0 = 0$

2.	$x = \sin\left(t^2 - 2t + 3\right), \quad t \in [0, 6], \; h = 0.025$	*traincgb, trainlm*

1 Описание

Слоистая сеть представляет собой композицию отображения $y = f(\phi(x))$
Суть процесса обучения:
Пусть имеется НС, зададим каким-либо образом начальные значения синаптических весов W в ней. Подадим на вход НС некоторый вектор

$$X = (X_1, ..., X_N)$$

Распространаяясь по сети, этот набор порождает набор порождает выходной вектор

$$Y = (Y_1, ..., Y_M)$$

Пусть для данного X_i входного вектора X известно, каким должно быть правильное значение Y^* выходного вектора Y.

Найдем уклонение желаемого выхода от выхода сети. Оно для данного вектора X представляет собой **ошибку**, вляющейся функцией от параметра W:

$$\epsilon(W) = ||Y^* - Y(X, W)||$$

Задача обучения сети - это задача минимизации функции ошибки по сниаптическим весам W:

$$\epsilon(W*) = \min_W \epsilon(p, W)$$

Метод обратного распростанения ошибки

1. Весам сети W присваиваются начальные значения.
2. Выбирается очередной обучающий пример $p_k = \langle X_k, Y_k \rangle$ из обучающего набора.
3. Вычисляется выход сети $Y(X_k, W)$.
4. Вычисляется ошибка $\epsilon_k(X_k, W) = ||Y^* - Y(X_k, W)||$
5. Шаги 2,3,4 повторяются для всез обучающих примеров при одном и том же значении W; эти N_p шагов составляют одну эпоху.
6. Вычисляется суммарная ошибка сети $\epsilon^{(r)}$ для данной эпохи:

$$\epsilon^{(r)} = \sum_{k=1}^{N_p} \epsilon_k$$

7. Веса сети W корректируются так, чтобы уменьшить ошибку.

8. Шаги 2-7 повторяются до тех пор, пока не будет выполнено условие $\epsilon_k(W) \leq \epsilon$, где $\epsilon > 0$ - наперд заданное малое число.

Последовательность шагов алгоритма обратного распространения ошибки реализует итерационную процедуру постепенного подбора весов W; для некоторой эпози r эту процедуру можно представить выражением вида:

$$W^{r+1} = W^r - \eta^r \frac{\partial \epsilon}{\partial W}$$

Ошибка сети $\epsilon(\dot{})$ непосредственно связана лишь с выходами сети и весами связей выходного слоя и предшествуюего ему скрытого слоя.

Нужно уметь определять **ошибку на выходах нейронов скрытых слоев** по ошибке в последующем.

Ошибка всей сети **распространяется назад**, от выходного слоя сети к ее входному слою, отсда название алгоритма: **алгоритм обратного распространения ошибки**.

Переход назад сопровождается пересчетом весов данного слоя.

В основе данного метода лежит цепное правило дифференцирования.

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

Путем тернистой дороги матанализа поуучаем:

$$\frac{\partial E}{\partial W^{l-1}} = \delta^{l-1} * (a^{l-2})^T$$

где δ_i^m - **чувствительность** E к изменению выхода сумматора i-го элемента из слоя m

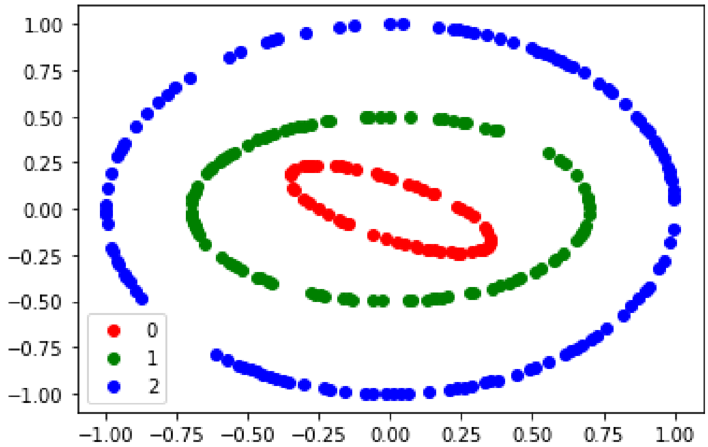
$$\delta^{l-1} = (f^{l-1})' \cdot (W^l)^T \cdot \delta^l$$

$$\delta^l = (f^l)' \cdot \frac{\partial E}{\partial a^l}$$

2 Ход работы

Задание 1

На основе заданных уравнений генерируем обучающее множество, затем делим случайным образом на обучающее, тестовое и контрольное.



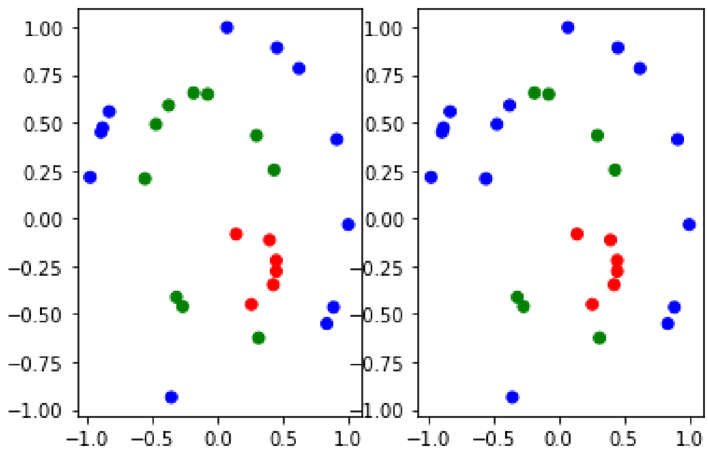

```
# 70-20-10 split
X_0 = np.vstack((x1, y1)).T
Y_0 = np.zeros(shape=(60,))
X_0_train, X_0_test, Y_0_train, Y_0_test = train_test_split(X_0, Y_0, test_size=0.3,
    random_state=13)
X_0_val, X_0_test, Y_0_val, Y_0_test = train_test_split(X_0_test, Y_0_test, test_size
    =0.33, random_state=13)

X_1 = np.vstack((x2, y2)).T
Y_1 = np.ones(shape=(100,))
X_1_train, X_1_test, Y_1_train, Y_1_test = train_test_split(X_1, Y_1, test_size=0.3,
    random_state=13)
X_1_val, X_1_test, Y_1_val, Y_1_test = train_test_split(X_1_test, Y_1_test, test_size
    =0.33, random_state=13)

X_2 = np.vstack((x3, y3)).T
Y_2 = 2 * np.ones(shape=(120,))
X_2_train, X_2_test, Y_2_train, Y_2_test = train_test_split(X_2, Y_2, test_size=0.3,
    random_state=13)
X_2_val, X_2_test, Y_2_val, Y_2_test = train_test_split(X_2_test, Y_2_test, test_size
    =0.33, random_state=13)

X_train = np.vstack((X_0_train, X_1_train, X_2_train))
Y_train = np.concatenate((Y_0_train, Y_1_train, Y_2_train))
X_val = np.vstack((X_0_val, X_1_val, X_2_val))
Y_val = np.concatenate((Y_0_val, Y_1_val, Y_2_val))
```

```
X_test = np.vstack((X_0_test, X_1_test, X_2_test))
Y_test = np.concatenate((Y_0_test, Y_1_test, Y_2_test))
```

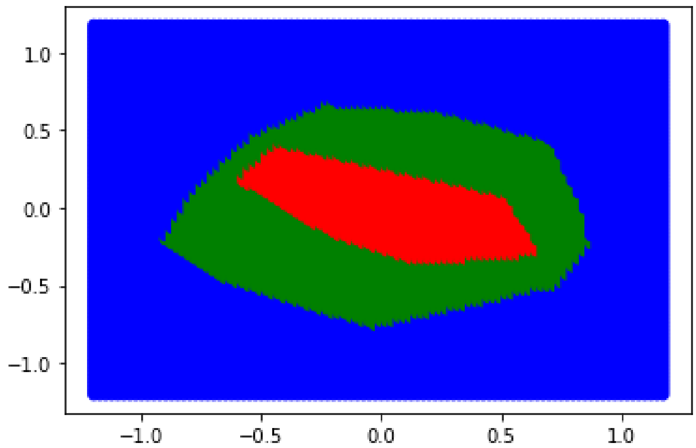
В качестве НС фреймворка я использовал PyTorch. Задаем структуру сети и проводим обучение на 1500 эпохах.

```
class feedforwardnet(nn.Module):
    def __init__(self):
        super().__init__()
        self.hidden1 = nn.Linear(2, 20)
        self.act = nn.ReLU()
        self.output = nn.Linear(in_features=20, out_features=3)

    def forward(self, x):
        x = self.hidden1(x)
        x = self.output(self.act(x))
        return x

model = feedforwardnet()
optimizer = torch.optim.Rprop(model.parameters(), lr=3e-4)
criterion = nn.CrossEntropyLoss()

epochs = 1500
losses = []
total_correct = 0

for epoch in tqdm(range(epochs)):
    for d, l in train_loader:
        loss = criterion(model(d), l)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        losses.append(loss.item())

    for d, l in val_loader:
        Y_pred = torch.argmax(model(d), dim=1)
        is_correct = (Y_pred == l)
        total_correct += is_correct.sum()

    if epoch % 300 == 0:
        print(f'Val accuracy: {total_correct / len(val_loader.dataset)}')
        print(f'Loss: {losses[-1]}\n')

    total_correct = 0
    losses = []

Val accuracy: 0.9642857313156128
Loss: 0.1410696804523468
```

Сделаем предсказание на тестовых данных в виде графика:



Произведем классификацию точек в облачи $[-1.2, 1.2] \times [-1.2, 1.2]$



Для 2-го и 3-го задания был выбран фреймворк neupy.

Генерируем множество и разбиваем его на обучающее и тестовое.

```
h = 0.025
t = np.linspace(0, 6, int(6/0.025), endpoint=True)
x = np.sin(t**2 - 6*t + 3)

train_size = int(t.shape[0] * 0.9)
train_size

X_train = t[:train_size]
y_train = x[:train_size]

X_test = t[train_size:]
```

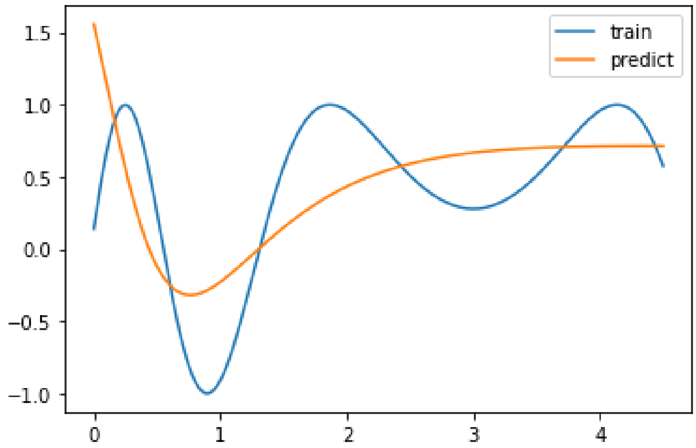
```
y_test = x[train_size:]

scaler_x = StandardScaler()
scaler_y = StandardScaler()
tmp_train_scaled_x = scaler_x.fit_transform(X_train[:, np.newaxis])
tmp_test_scaled_x = scaler_x.transform(X_test[:, np.newaxis])
tmp_train_scaled_y = scaler_y.fit_transform(y_train[:, np.newaxis])
```

Задание 2

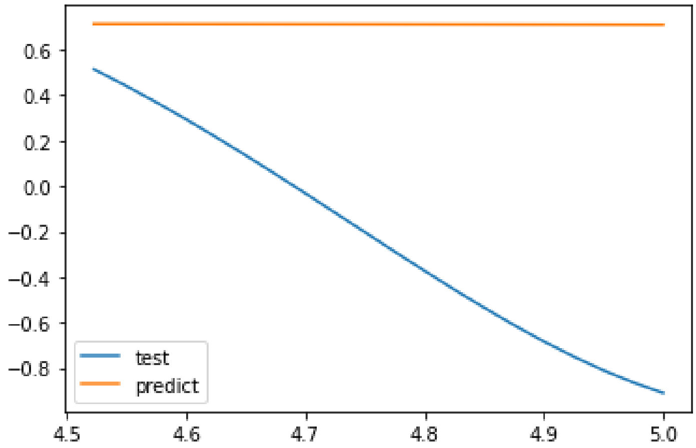
Метод сопряженных градиентов:

```

lmnet = algorithms.ConjugateGradient((Input(1), Tanh(20), Linear(1)), verbose=True)
lmnet.train(X_train, y_train, epochs=60)
```

Результат аппроксимации после обучения:
$RMSE = 0.4687237539182378$



Предсказание на тестовой выборке:
$RMSE = 1.0425874397569055$



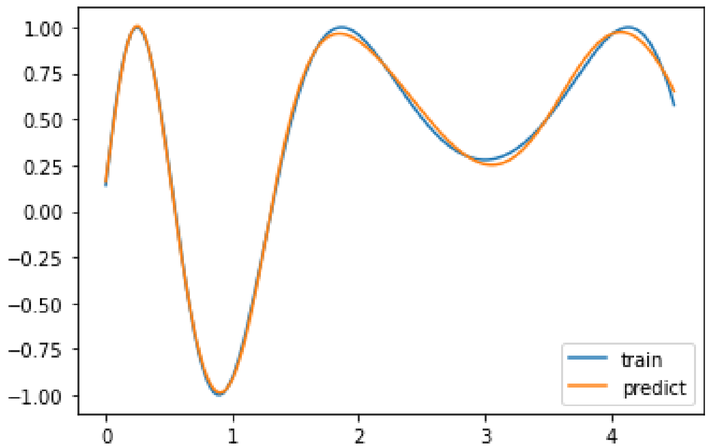
Задание 3

Метод Левенберга-Марквардта:

```

lmnet2 = algorithms.LevenbergMarquardt((Input(1), Tanh(20), Linear(1)), mu = 0.1,
    verbose=True)
lmnet2.train(X_train, y_train, epochs=60)
```

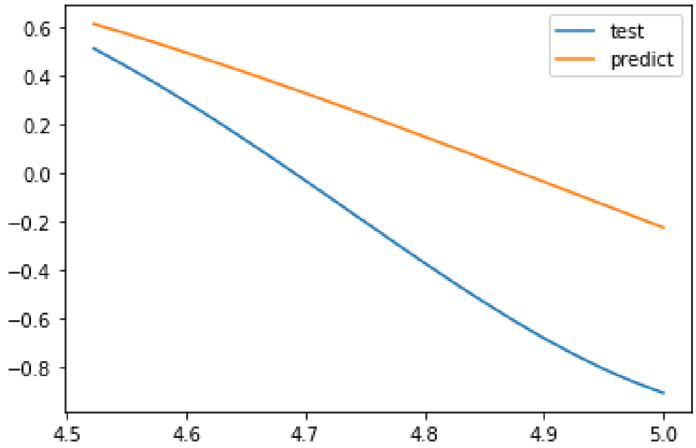
Результат аппроксимации после обучения:

$RMSE = 0.02509954806350635$



Предсказание на тестовой выборке:

$RMSE = 0.48057248246774226$


test
predict
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
4.5
4.6
4.7
4.8
4.9
5.0

3 Выводы

В рамках третьей лабораторной работы я познакомился с многослойными нейронными сетями -мощнейшим аппаратом для решения разнообразных задач. Например, как в данной работе, задач классификации линейно неразделимых классов и аппроксимации функций.