Лабораторная работа 1.

Один нейрон на PyTorch

Разберем как писать свои нейросети на фреймворке PyTorch:

- рассмотрим, как написать свой нейрон с разными функциями потерь
- рассмотрим работу нейрона на конкретных датасетах.

Задания

1. Загрузить данные из csv-файла, содержащего набор данных «Яблоки-груши»
2. Построить изображение набор данных «Яблоки-груши» в виде точек на плоскости

```
plt.figure(figsize=(10, 8))
plt.scatter(data.iloc[:, 0], data.iloc[:, 1], c=data['target'], cmap='rainbow')
plt.title('Яблоки и груши', fontsize=15)
plt.xlabel('симметричность', fontsize=14)
plt.ylabel('желтизна', fontsize=14)
plt.show();
```

3. Выделить из набора данных матрицу признаков (X) и вектор ответов (y)
4. Импортировать необходимые библиотеки:

```
import torch
from torch.nn import Linear, Sigmoid
```

5. Есть **два пути объявления нейросетей** в PyTorch:

- функциональный (Functional);
- последовательный (Sequential);

Будем использовать второй путь (он чуть более user-friendly) и построим таким способом один нейрон (точно такой же, который мы реализовывали раньше с помощью Numpy):

```
num_features = X.shape[1]

neuron = torch.nn.Sequential(
Linear(num_features, out_features=1),
Sigmoid()
)
```

6. Мы просто создали объект класса Sequential, который состоит из одного линейного слоя размерности (num_features, 1) и последующего применения сигмоиды. Но уже сейчас его можно применить к объекту (тензору), просто веса в начале инициализирутся случайно и при forward_pass'е мы получим какой-то ответ пока что необученного нейрона:

```
neuron(torch.autograd.Variable(torch.FloatTensor([1, 1])))
```

7. Получим результат классификации пока ещё необученным нейроном:

```
proba_pred = neuron(torch.autograd.Variable(torch.FloatTensor(X)))
```

```
y_pred = proba_pred > 0.5
y_pred = y_pred.data.numpy().reshape(-1)

plt.figure(figsize=(10, 8))
plt.scatter(data.iloc[:, 0], data.iloc[:, 1], c=y_pred, cmap='spring')
plt.title('Яблоки и груши', fontsize=15)
plt.xlabel('симметричность', fontsize=14)
plt.ylabel('желтизна', fontsize=14)
plt.show();
```

Как можно заметить, ничего полезного пока не происходит.

8. Обучим нейрон. Обернём данные в torch.Tensor'ы, а тензоры в torch.Variable'ы, чтобы можно было вычислять градиенты по весам:

```
X = torch.autograd.Variable(torch.FloatTensor(X))
y = torch.autograd.Variable(torch.FloatTensor(y))
```

9. Код обучения одного нейрона на PyTorch:

```
# квадратичная функция потерь (можно сделать другую, например, LogLoss)
loss_fn = torch.nn.MSELoss(reduction='none')

# шаг градиентного спуска (точнее -- метода оптимизации)
learning_rate = 0.001                # == 1e-3

# сам метод оптимизации нейросети (обычно лучше всего по-умолчанию работает Adam)
optimizer = torch.optim.SGD(neuron.parameters(), lr=learning_rate)

# количество итераций в градиентном спуске равно num_epochs, здесь 500
for t in range(500):
    # foward_pass() -- применение нейросети (этот шаг ещё называют inference)
    y_pred = neuron(X)

    # выведем loss
    loss = loss_fn(y_pred, y)
    print('{} {}'.format(t, loss.data))

    # обнуляем градиенты перед backard_pass'ом (обязательно!)
    optimizer.zero_grad()

    # backward_pass() -- вычисляем градиенты loss'а по параметрам (весам) нейросети
    # этой командой мы только вычисляем градиенты, но ещё не обновляем веса
    loss.backward()

    # а тут уже обновляем веса
    optimizer.step()
```

10. Получим результат классификации обученным нейроном:

```
proba_pred = neuron(X)
y_pred = proba_pred > 0.5
y_pred = y_pred.data.numpy().reshape(-1)

plt.figure(figsize=(10, 8))
plt.scatter(data.iloc[:, 0], data.iloc[:, 1], c=y_pred, cmap='spring')
plt.title('Яблоки и груши', fontsize=15)
plt.xlabel('симметричность', fontsize=14)
```

```
plt.ylabel('желтизна', fontsize=14)
plt.show();
```

11. Можно добиться лучшего качества работы путём изменения learning_rate и количества итераций (возможно, ещё функции потерь и функции активации). Проведите вычислительный эксперимент, подобрав такие значения learning_rate и количество итераций, при которых качество работы нейрона будет наилучшим (сохраните все свои попытки и их результаты), также попробуйте использовать функцию потерь torch.nn.NLLLoss() или torch.nn.CrossEntropy().

12. Попробуем более сложную выборку, которая уже не разделяется линейно:

```
N = 100
D = 2
K = 3
X = np.zeros((N * K, D))
y = np.zeros(N * K, dtype='uint8')

for j in range(K):
  ix = range(N * j,N * (j + 1))
  r = np.linspace(0.0, 1, N)
  t = np.linspace(j * 4, (j + 1) * 4,N) + np.random.randn(N) * 0.2 # theta
  X[ix] = np.c_[r * np.sin(t), r * np.cos(t)]
  y[ix] = j

plt.figure(figsize=(10, 8))
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.rainbow)
plt.title('Спираль', fontsize=15)
plt.xlabel('$x$', fontsize=14)
plt.ylabel('$y$', fontsize=14)
plt.show();
```

13. Преобразуем данные в Variable():

```
X = torch.autograd.Variable(torch.FloatTensor(X))
y = torch.autograd.Variable(torch.LongTensor(y.astype(np.int64)))

print(X.data.shape, y.data.shape)
```

14. Попробуем нейрон с сигмоидой на линейно неразделимой выборке (точнее в данном случае это уже 3 нейрона с софтмаксом):

```
# N - размер батча (batch_size, нужно для метода оптимизации)
# D_in - размерность входа (количество признаков у объекта)
# D_out - размерность выходного слоя (суть -- количество классов)
N, D_in, D_out = 64, 2, 3

neuron = torch.nn.Sequential(
  torch.nn.Linear(D_in, D_out),
)
loss_fn = torch.nn.CrossEntropyLoss(reduction='none')

learning_rate = 1e-4
optimizer = torch.optim.SGD(neuron.parameters(), lr=learning_rate)
for t in range(500):
  # forward
  y_pred = neuron(X)
```

```
    # loss
    loss = loss_fn(y_pred, y)
    print('{} {}'.format(t, loss.data))

    # зануляем градиенты с предыдущего шага
    optimizer.zero_grad()

    # backward
    loss.backward()

    # обновляем веса
    optimizer.step()
```

15. Нарисуем результат:

```
# Обратно в Numpy для отрисовки
X = X.data.numpy()
y = y.data.numpy()

h = 0.02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
grid_tensor = torch.FloatTensor(np.c_[xx.ravel(), yy.ravel()])

Z = neuron(torch.autograd.Variable(grid_tensor))
Z = Z.data.numpy()
Z = np.argmax(Z, axis=1)
Z = Z.reshape(xx.shape)

plt.figure(figsize=(10, 8))

plt.contourf(xx, yy, Z, cmap=plt.cm.rainbow, alpha=0.3)
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.rainbow)

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())

plt.title('Спираль', fontsize=15)
plt.xlabel('$x$', fontsize=14)
plt.ylabel('$y$', fontsize=14)
plt.show();
```

16. Оцените полученный результат

Теория

Компоненты нейросети

Вспомним, какие вещи играют принципиальную роль в построении любой ***нейронной сети*** (все их мы задаём *вручную*, самостоятельно):

- непосредственно, сама **архитектура** нейросети (сюда входят типы функций активации у каждого нейрона);
- начальная **инициализация** весов каждого слоя;

- метод **оптимизации** нейросети (сюда ещё входит метод изменения learning_rate);
- размер **батчей** (batch_size);
- количество **эпох** обучения (num_epochs);
- **функция потерь** (loss);
- тип **регуляризации** нейросети (для каждого слоя можно свой);

То, что связано с ***данными и задачей***:

- само **качество** выборки (непротиворечивость, чистота, корректность постановки задачи);
- **размер** выборки;

В модуле torch.nn находится все необходимое для конструирования нейросетей, а в модуле torch.optim находится все необходимое для выбора метода оптимизации нейросети.