Распознавание цифры по базе MNIST

Делаем нейронную сеть, которая будет распознавать цифры, написанные от руки

Обязательно изучите введение в нейронные сети. Чтобы решить данную задачу нужно будет создать многослойный персепрон. В прошлый раз я рассказывал как обучить нейронную сеть операции XOR. Нейронная сеть будет очень похожей.

Решение для PyTorch

Проект доступен на гитхабе.

Демо версия.

Для начала, подключим необходимые библиотеки:

```
import torch, math
import numpy as np
import matplotlib.pyplot as plt

from torch import nn
from torchsummary import summary
from torch.utils.data import DataLoader, TensorDataset

tensor_device = torch.device('cuda') if torch.cuda.is_available() else to
print ("Device:", tensor_device)
```

Датасет будем брать из базы MNIST. MNIST - это база цифр от 0 до 9, нарисованных отруки.

Скачать ее можно следующей командой:

```
wget https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.np
```

Загрузим датасет:

```
data_orig = np.load("mnist.npz", allow_pickle=True)
data_orig = {
  "train": {
    "x": data_orig["x_train"],
    "y": data_orig["y_train"],
  },
```

```
  "test": {
    "x": data_orig["x_test"],
    "y": data_orig["y_test"],
  }
}
```

Дата сет содержит обучающую и контрольную выборку, а также сами изображения и правильные ответы.

Выведите на экран информации о датасете:

```
print ("Train images", data_orig["train"]["x"].shape)
print ("Train answers", data_orig["train"]["y"].shape)
print ("Test images", data_orig["test"]["x"].shape)
print ("Test answers", data_orig["test"]["y"].shape)
```

Должно вывести следующую информацию:

```
Train images (60000, 28, 28)
Train answers (60000,)
Test images (10000, 28, 28)
Test answers (10000,)
```

Давайте убедимся, что там действительно цифры. Загрузим из датасета определенное фото и отобразим его на экране. Должна отобразиться цифра, которая находится на позиции photo_number.

```
photo_number=256
print ("Number:", data_orig["train"]["y"][photo_number])
plt.imshow(data_orig["train"]["x"][photo_number], cmap='gray')
plt.show()
```

Нормализация данных

Перед тем, как обучать нейронную сеть, нужно нормализовать изображения. Фотографии 28x28 нужно преобразовать в вектора. Также нужно все числа вектора поделить на 255, чтобы они были в диапазоне от 0 до 1 формата float32.

А также правильные ответы должны быть в векторе из 10 цифр. Например:

- число 1 нужно преобразовать в вектор [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
- число 5 в [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

```
def get_vector_by_number(count):

    r"""
```

```
    Преобразует число в списке в выходной вектор
    Например:
    1 -> [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
    5 -> [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
    """

    def f(number):
      res = [0.0] * count

      if (number >=0 and number < count):
        res[number] = 1.0

      return res

    return f
```

Объявим две функции, которые будут нормализовать входящие данные и ответы.

```
 def data_normalize_x(data_x):
    r"""
    Нормализация датасета по x
    """
    data_x = torch.from_numpy(data_x)
    data_x_shape_len = len(data_x.shape)

    if data_x_shape_len == 3:
      data_x = data_x.reshape(data_x.shape[0], -1)
    elif data_x_shape_len == 2:
      data_x = data_x.reshape(-1)

    data_x = data_x.to(torch.float32) / 255.0
    return data_x


 def data_normalize_y(data_y):
    r"""
    Нормализация датасета по y
    """
    data_y = list(map(get_vector_by_number(10), data_y))
    data_y = torch.tensor( data_y )
    return data_y
```

Создадим нормализованный датасет:

```
 batch_size = 128
```

```
data = {
  "train": {
    "x": data_normalize_x(data_orig["train"]["x"]),
    "y": data_normalize_y(data_orig["train"]["y"]),
  },
  "test": {
    "x": data_normalize_x(data_orig["test"]["x"]),
    "y": data_normalize_y(data_orig["test"]["y"]),
  }
}

train_dataset = TensorDataset( data["train"]["x"], data["train"]["y"] )
test_dataset = TensorDataset( data["test"]["x"], data["test"]["y"] )

train_count = data["train"]["x"].shape[0]
test_count = data["test"]["x"].shape[0]

train_loader = DataLoader(
  train_dataset,
  batch_size=batch_size,
  drop_last=True,
  shuffle=True
)
test_loader = DataLoader(
  test_dataset,
  batch_size=batch_size,
  drop_last=True,
  shuffle=False
)
```

Создание и обучение нейронной сети

Архитектура модели:

```
def create_model():
  input_shape = 784
  output_shape = 10

  model = nn.Sequential(
    nn.Linear(input_shape, 128),
    nn.ReLU(),
    nn.Linear(128, output_shape),
    #nn.Softmax()
  )
```

```
  # Adam optimizer
  optimizer = torch.optim.Adam(model.parameters(), lr=1e-3, betas=(0.9, 0

  # mean squared error
  loss = nn.MSELoss()

  return {
    "input_shape": input_shape,
    "output_shape": output_shape,
    "model": model,
    "optimizer": optimizer,
    "loss": loss,
  }
```

Выведем информацию о модели на экран:

```
model_info = create_model()

# Show model info
summary(model_info["model"], (model_info["input_shape"],))
```

Будет выведено:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 128]         100,480
              ReLU-2                  [-1, 128]               0
            Linear-3                   [-1, 10]           1,290
================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.39
Estimated Total Size (MB): 0.39
----------------------------------------------------------------
```

Обучение модели

```
epochs = 20

model_info = create_model()
```

```
model = model_info["model"]
optimizer = model_info["optimizer"]
loss = model_info["loss"]

model = model.to(tensor_device)

history = {
  "loss_train": [],
  "loss_test": [],
}

for step_index in range(epochs):

  loss_train = 0
  loss_test = 0

  batch_iter = 0

  # Обучение
  for batch_x, batch_y in train_loader:

    batch_x = batch_x.to(tensor_device)
    batch_y = batch_y.to(tensor_device)

    # Вычислим результат модели
    model_res = model(batch_x)

    # Найдем значение ошибки между ответом модели и правильными ответами
    loss_value = loss(model_res, batch_y)
    loss_train = loss_value.item()

    # Вычислим градиент
    optimizer.zero_grad()
    loss_value.backward()

    # Оптимизируем
    optimizer.step()

    # Очистим кэш CUDA
    if torch.cuda.is_available():
      torch.cuda.empty_cache()

    del batch_x, batch_y

    batch_iter = batch_iter + batch_size
    batch_iter_value = round(batch_iter / train_count * 100)
```

```
    print (f"\rStep {step_index+1}, {batch_iter_value}%", end='')


  # Вычислим ошибку на тестовом датасете
  for batch_x, batch_y in test_loader:

    batch_x = batch_x.to(tensor_device)
    batch_y = batch_y.to(tensor_device)

    # Вычислим результат модели
    model_res = model(batch_x)

    # Найдем значение ошибки между ответом модели и правильными ответами
    loss_value = loss(model_res, batch_y)
    loss_test = loss_value.item()


  # Отладочная информация
  #if i % 10 == 0:
  print ("\r", end='')
  print (f"Step {step_index+1}, loss: {loss_train},\tloss_test: {loss_tes

  # Остановим обучение, если ошибка меньше чем 0.01
  if loss_test < 0.015 and step_index > 5:
    break

  # Добавим значение ошибки в историю, для дальнейшего отображения на гра
  history["loss_train"].append(loss_train)
  history["loss_test"].append(loss_test)
```

Запустим обучение, и выведем его на экран:

```
Step 1, loss: 0.007191469427198172,  loss_test: 0.020943233743309975
Step 2, loss: 0.005624017678201199,  loss_test: 0.016437651589512825
Step 3, loss: 0.004931427538394928,  loss_test: 0.014247491955757141
Step 4, loss: 0.0045996420085430145, loss_test: 0.013283359818160534
Step 5, loss: 0.004507290665060282,  loss_test: 0.012732605449855328
Step 6, loss: 0.004471090622246265,  loss_test: 0.01236715167760849
Step 7, loss: 0.004237602464854717,  loss_test: 0.011990693397819996
```

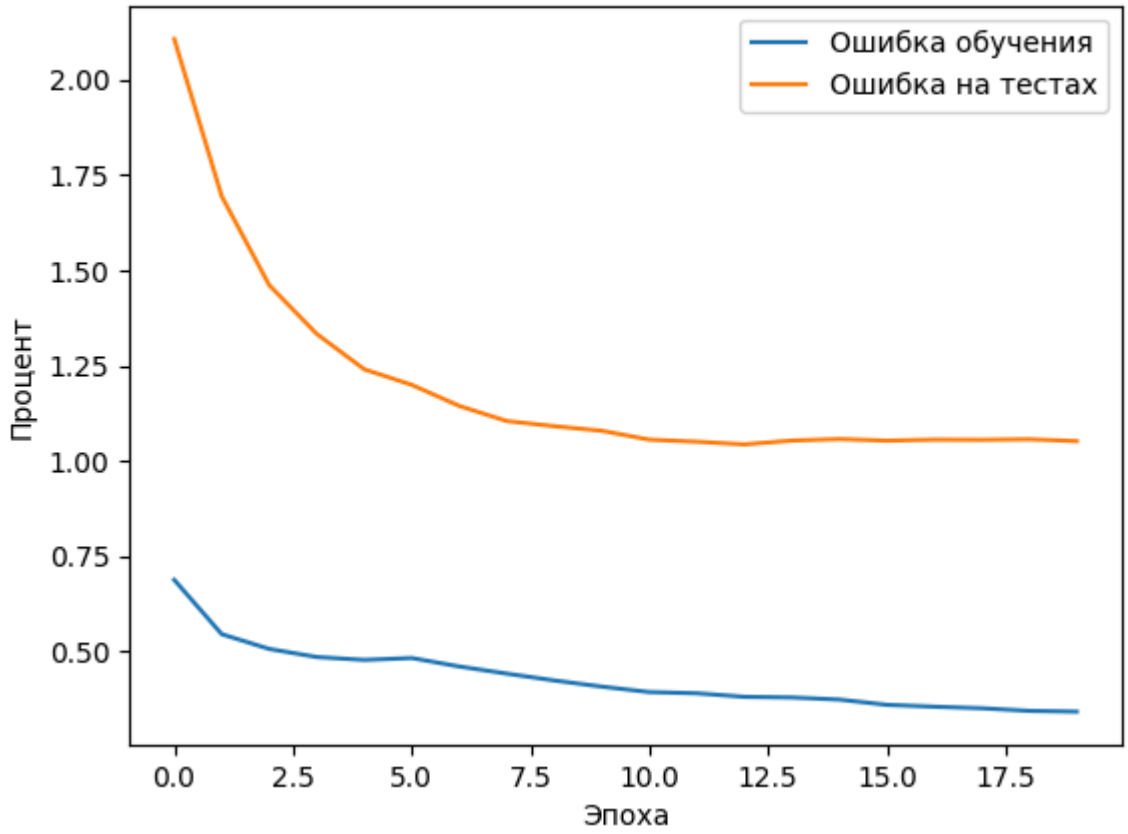
Покажем график обучения:

```
import matplotlib.pyplot as plt

plt.plot( np.multiply(history['loss_train'], 100), label='Ошибка обучения
```

```
plt.plot( np.multiply(history['loss_test'], 100), label='Ошибка на тестах')
plt.ylabel('Процент')
plt.xlabel('Эпоха')
plt.legend()
plt.show()
```

Результат обучения. Как видно из графика процент ошибки стремится к нуль, а правильные ответы к 100%. Делаем вывод, что нейронная сеть обучилась корректно.



Проверка модели

Напишем функцию, которая по вектору, которая отвечает модель, будет возвращаться ответ ввиде числа, а не вектора.

```
def get_answer_from_vector(vector):
  r"""
  Returns answer from vector
  """
  value_max = -math.inf
  value_index = 0
  for i in range(0, len(vector)):
    value = vector[i]
```

```
    if value_max < value:
      value_index = i
      value_max = value

  return value_index
```

Проверим как правильно отвечает модель:

```
photo_number = 200
photo = data_orig["test"]["x"][photo_number]
correct_answer = data_orig["test"]["y"][photo_number]

tensor_x = data_normalize_x(photo)
tensor_x = tensor_x[None, :]
tensor_y = model(tensor_x)

model_answer = get_answer_from_vector(tensor_y[0].tolist())

print ("Model answer", model_answer)
print ("Correct answer", correct_answer)

plt.imshow(photo, cmap='gray')
plt.show()
```

Литература

1. Deep Learning (семестр 1, весна 2022): продвинутый поток
2. Введение в библиотеку Pytorch
3. Нейронная сеть на Pytorch