

实现网络来分类数字 → github 地址.

(通用梯度下降法和 mnist 训练数据)

MNIST 数据集: 6W 训练图像, 1W 测试图像

将训练集分为 1W. 分为 5W 训练神经网络

测试集: 1W. 分为 1W: (验证集) (validation set)

net = Network([2, 3, 1]): 子集 sizes 表明: 第 1 层 (输入层) 第 2 层 第 3 层

<https://github.com/mnielsen/neural-networks-and-deep-learning/blob/master/src/network.py>

Numpy 的 np.random.randn

(生成均值为 0, 标准差为 1 的高斯分布)

随机初始化每个权重 SGD 一个超参数

权重. 权重以 list 存在 numpy 数组

例如 net.weights[1] 表明: 第 2 层 → 第 3 层神经元 numpy 数组

第 3 层输出向量:  $a' = \sigma(wa + b)$

[接收 6 个向量] vectorizing

每个 network 对应一个超参数: 学习率 第 2 层激活函数

为此, 实现一个随机梯度下降 SGD 方法 → github 地址

trainingdata 训练数据 → 期望输出元组 (x, y) 列表

epochs / minibatches: 期望训练迭代次数 / 每次 minibatch 大小

eta 学习率

每次训练, 先随机将训练数据打乱 → 生成随机小训练集

然后对每个 minibatch 应用一次梯度下降 (update minibatch (min eta))

大部分工作 (例如: 0 b, 0 w = self.backprop(x, y)) [backpropagation 反向传播]

```
import mnist_loader
import network
```

```
trainingdata, validationdata, testdata = mnist_loader.load_data_wrapper()
```

```
net = network.Network([784, 30, 10])
```

```
net.SGD(trainingdata, 30, 10, 3.0, testdata)
```

工作就是训练 minibatch

每个训练样本计算梯度

然后更新 self.weights

(快速计算梯度)

无梯度下降法