崔子寒
MG20330012
zihancui@smail.nju.edu.cn

Assignment 4
Neural Network, 19/20

2020 年 11 月 25 日

# 1 环境说明

- 操作系统: MacOS 10.15.7
- Python 版本: 3.6.10 64-bit
- Python 项目依赖 (相近版本应该也可以运行):
  * numpy==1.18.1
  * matplotlib==3.1.3
- 安装依赖: pip install -r requirements.txt
- 运行方法:

```
usage: main.py [-h] [-lr LEARNING_RATE] [--batch-size BATCH_SIZE]
               [--hidden-size HIDDEN_SIZE] [--max-epoch MAX_EPOCH]

Neural Network Training

optional arguments:
  -h, --help            show this help message and exit
  -lr LEARNING_RATE, --learning-rate LEARNING_RATE
                        learning rate, default 0.001
  --batch-size BATCH_SIZE
                        batch size. should be a multiple of 64. default 64
  --hidden-size HIDDEN_SIZE
                        hidden size. default 48
  --max-epoch MAX_EPOCH
                        max epoch. default 1000
```

默认的参数是经过测试的比较好的参数，直接运行 `python main.py` 即可。

# 2 算法设计

## 2.1 问题分析

首先我们分析问题，我们需要对一个三维空间中的二维曲面进行拟合。我们已经知道这个曲面的方程描述:

$$y = sin(x_1) - cos(x_2), x_1 \in [-5, 5], x_2 \in [-5, 5]$$

那么我们首先需要获取训练数据，最简单的方式就是对曲面进行均匀采样，采样出一些数据点作为训练数据。曲面拟合这个任务是一个回归任务，而且我们直接对原始曲面进行采样是可以保证训练数据和真

实数据的分布是完全相同的，因此不需要考虑过拟合的问题，也不需要设置验证集进行验证。我们的目标就是让训练误差最小。

## 2.2 网络设计

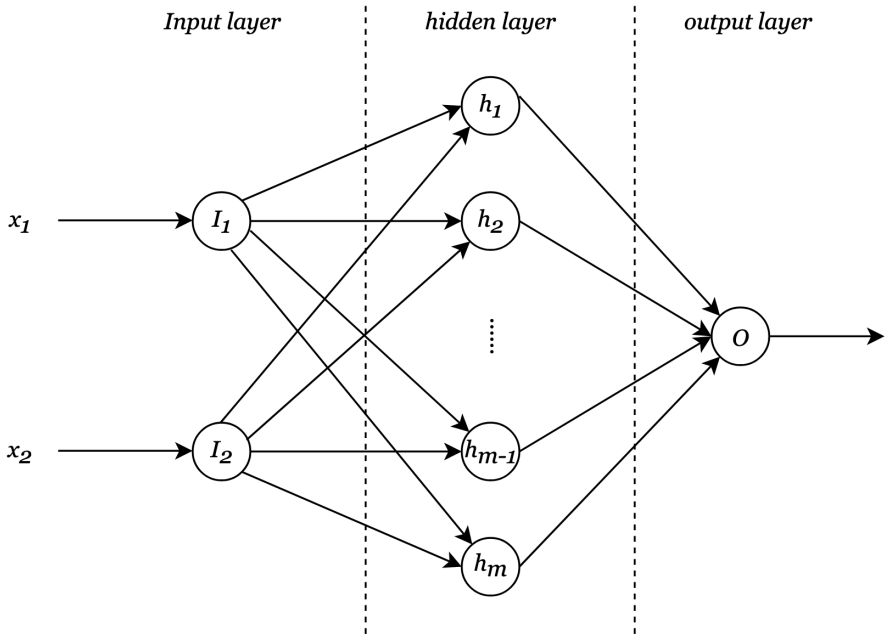


图 1: 网络结构

我的网络中使用了单层隐藏层。考虑到曲面拟合是一个比较简单的任务，因此不需要非常深的网络。虽然加深网络提升模型的性能，但是加深网络也会带来问题。网络的深度太大会带来梯度爆炸和梯度消失的问题，也会增加反向传播的计算开销。使用单层隐藏层可以避免这些问题，而且训练较快。但是这需要增加隐藏层的神经元个数，经过测试，至少需要 30 个以上的神经元才能较好的拟合出给定的曲面，我的实现中使用了 48 个神经元。

隐藏层的激活函数使用 `sigmoid` 函数:

$$f(x) = \frac{1}{1+e^{-x}}$$

使用 `sigmoid` 函数的原因有:

- `sigmoid` 函数非常光滑，便于求导，而且导数的形式简单;
- `sigmoid` 函数中有指数部分，可以引入非线性;

输出层我们不使用激活函数，直接输出结果。使用均方误差 (Mean Squared Error, MSE) 来衡量预测值和真实值的差距。

$$loss = \frac{1}{m}\sum_{i=1}^{m}(\hat{y_i} - y_i)^2$$

其中 $m$ 是一个批次的大小 (batch size)。

## 2.3 算法实现

### 2.3.1 前向传播

- 输入层 $\longrightarrow$ 隐藏层：为了加快训练速度，同时也防止单个数据点的梯度太小，我使用批量梯度下降来进行训练。在批量梯度下降中，假定 batch size 为 $m$，那么实际上一个批是一个 $m \times 2$ 大小的矩阵。矩阵的每一行都是一个数据点的坐标 $(x_1, x_2)$。假定隐藏层有 $l$ 个神经元，那么对于第 $i$ 个神经元需要训练三个参数，包括两个权重 $w_{i1}, w_{i2}$ 和一个偏置 $b_i$，但是实际上，可以考虑为输入矩阵新增一列 1，那么就可以把 $b_i$ 也看作一个权重，它对应的输入总是 1。

  将隐藏层的每个神经元参数当作行向量，构成的矩阵记做 $W_{hidden}$，记一个批次的矩阵为 $I$，那么隐藏层的输出为:

  $$O_{hiddent} = \texttt{sigmoid}(W_{hidden} I^T)$$

- 隐藏层 $\longrightarrow$ 输出层：隐藏层的输出是一个 $l \times m$ 的向量，输出层需要训练的参数是 $l$ 个权值以及一个偏置。记输出层的权值向量为 $W_{output}$,，输出层的偏置为 $B_{output}$，那么输出层的输出为:

  $$O_{output} = W_{output} O_{hiddent}^T + B_{output}$$

### 2.3.2 反向传播

我们使用 MSE 作为损失函数，那么对于一个批，记它的输出为 $\hat{y} = [\hat{y_1}, \hat{y_2}, \cdots \hat{y_m}]$，记真实值为 $y = [y_1, y_2, \cdots y_m]$，那么 loss 为:

$$loss = \frac{1}{m} \sum_{i=1}^{m} (\hat{y_i} - y_i)^2$$

我们根据权值来分别计算 loss 关于 $B_{output}$,$W_{output}$,$W_{hidden}$ 的偏导数。

- 输出层偏置 $B_{output}$:

$$\begin{aligned}
\frac{\partial loss}{\partial B_{output}} =& \frac{1}{m} \sum_{i=1}^{m} \frac{\partial (\hat{y_i} - y_i)^2}{\partial B_{output}} \\
=& \frac{1}{m} \sum_{i=1}^{m} \frac{\partial (\hat{y_i} - y_i)^2}{\partial \hat{y_i}} \frac{\partial \hat{y_i}}{\partial B_{output}} \\
=& \frac{2}{m} \sum_{i=1}^{m} (\hat{y_i} - y_i) \frac{\partial (W_{output} O_{hiddent} + B_{output})}{\partial B_{output}} \\
=& \frac{2}{m} \sum_{i=1}^{m} (\hat{y_i} - y_i)
\end{aligned}$$

- 输出层权值 $W_{output}$:

$$\begin{aligned}\frac{\partial loss}{\partial W_{output}} =& \frac{1}{m}\sum_{i=1}^{m}\frac{\partial(\hat{y}_i - y_i)^2}{\partial W_{output}} \\ =& \frac{1}{m}\sum_{i=1}^{m}\frac{\partial(\hat{y}_i - y_i)^2}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial W_{output}} \\ =& \frac{2}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)\frac{\partial(W_{output}O^T_{hiddent}[:,i] + B_{output})}{\partial W_{output}} \\ =& \frac{2}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)O^T_{hidden}[:,i] \\ =& \frac{2}{m}(\hat{y} - y)O^T_{hidden}\end{aligned}$$

- 隐藏层权值 $W_{hidden}$:

$$\begin{aligned}\frac{\partial loss}{\partial W_{hidden}} =& \frac{1}{m}\sum_{i=1}^{m}\frac{\partial(\hat{y}_i - y_i)^2}{\partial W_{hidden}} \\ =& \frac{1}{m}\sum_{i=1}^{m}\frac{\partial(\hat{y}_i - y_i)^2}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial W_{hidden}} \\ =& \frac{2}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)\frac{\partial(W_{output}O^T_{hiddent}[:,i] + B_{output})}{\partial W_{hidden}} \\ =& \frac{2}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)\frac{\partial(W_{output}O^T_{hiddent}[:,i] + B_{output})}{\partial O^T_{hiddent}[:,i]}\frac{\partial O^T_{hiddent}[:,i]}{\partial W_{hidden}} \\ =& \frac{2}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)W_{output}\frac{\partial \texttt{sigmoid}(W_{hidden}I_i)}{\partial W_{hidden}} \\ =& \frac{2}{m}\texttt{sigmoid}(W_{hidden}I) * (1 - \texttt{sigmoid}(W_{hidden}I)) * (W_{output}(\hat{y} - y)I)\end{aligned}$$

  其中 * 代表矩阵对应位置元素做乘法。

## 2.4 代码实现

当推导出如何计算梯度之后，只需要将参数减去梯度乘上学习率，就可以进行参数更新，具体当代码实现如下：

```
for epoch in range(self.max_epochs):
        total_error = 0
        if epoch in self.schedule_point:
            print("Decrease lr from {} to {}".format(self.learning_rate, self.gamma *
                self.learning_rate))
            self.learning_rate *= self.gamma
        for batch in range(len(data_loader)):
            item_X, item_y = data_loader[batch]
            item_X = np.c_[item_X, np.ones(shape=(self.batch_size,1))]
            # forward
            hidden_output = np.dot(item_X, self.hidden_layer.T) # (batch_size, hidden_size)
            output_input = sigmoid(hidden_output)# (batch_size, hidden_size)
```

```
            output = np.dot(output_input, self.output_layer).T + self.output_bias# (1, batch_size)
            np.reshape(output, (self.batch_size,)) #(1, batch_size)
            mse_error = np.sum((item_y - output)**2) / self.batch_size
            total_error += mse_error

            # backward
            # 输出层偏置
            gradient_output_bias = 2 / self.batch_size
            gradient_output_bias *= np.sum(output-item_y)
            self.output_bias -= self.learning_rate * gradient_output_bias

            # 输出层权值
            gradient_output_weight = 2 / self.batch_size
            gradient_output_weight *= np.dot((output-item_y),output_input).T #(hidden_size, 1)
            self.output_layer -= self.learning_rate * gradient_output_weight

            # 隐藏层权值
            gradient_hidden_weight = 2 / self.batch_size
            gradient_hidden_weight *= np.dot(self.output_layer, output-item_y) #(hidden_size,
                batch_size)
            derivative_activate = (1-output_input) * output_input #(batch_size, hidden_size)
            gradient_hidden_weight *= derivative_activate.T #(hidden_size, batch_size)
            gradient_hidden_weight = np.dot(gradient_hidden_weight, item_X) #(hidden, input_length
                + 1)
            self.hidden_layer -= self.learning_rate * gradient_hidden_weight

        training_loss.append(total_error / len(data_loader))
        print("[Epoch {:>3d}/{}]: MSE Training Loss: {:.8f}.".format(epoch + 1, self.max_epochs,
            total_error / len(data_loader)))
    return training_loss
```

# 3 拟合效果

## 3.1 效果展示

我们使用的训练数据来自对 $x_1 \in [-6,6], x_2 \in [-6,6]$ 的均匀采样。测试数据来自于 $x_1 \in [-5,5], x_2 \in [-5,5]$ 的均匀采样。

```
import numpy as np
X_train1 = np.linspace(start=-6, stop=6, num=1024)
X_train2 = np.linspace(start=-6, stop=6, num=1024)
X_test1 = np.linspace(start=-5, stop=5, num=1000)
X_test2 = np.linspace(start=-5, stop=5, num=1000)
```

我们使用的隐藏层大小为 48，Batch size 为 64，学习率为 0.001，在 1000 个 epoch 上，训练的效果如下：

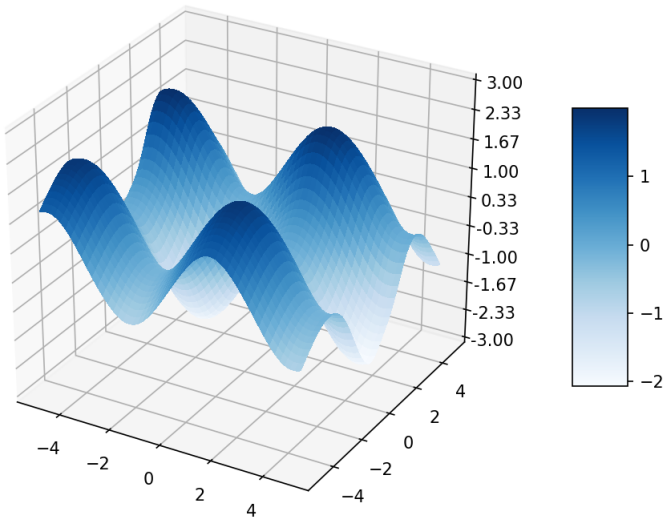


图 2: 拟合曲面

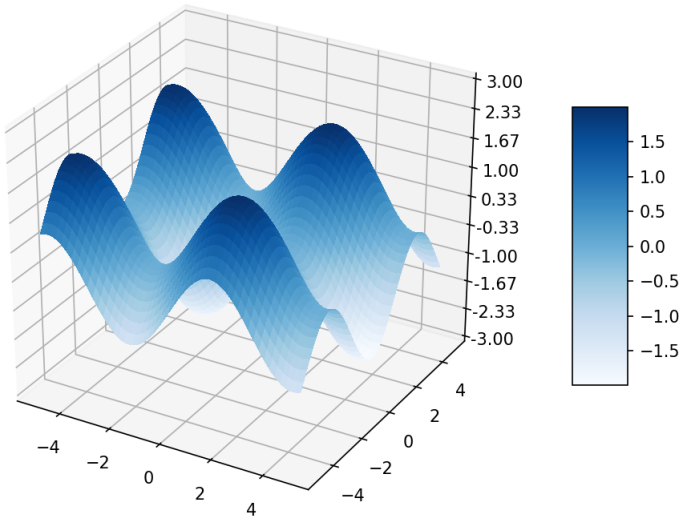


图 3: 真实曲面

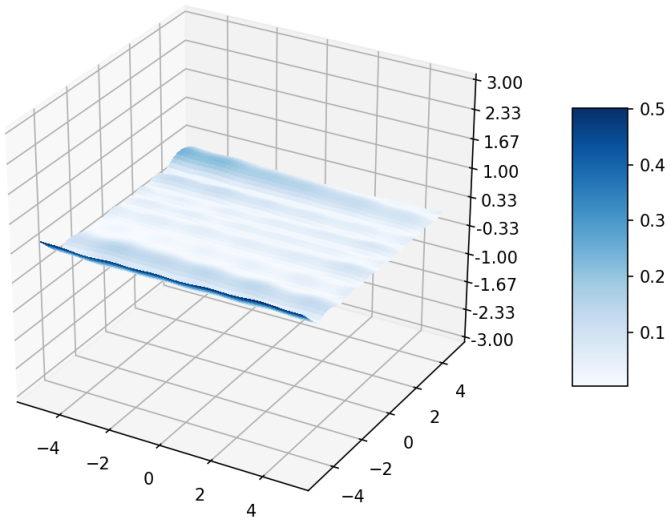


图 4: 误差曲面

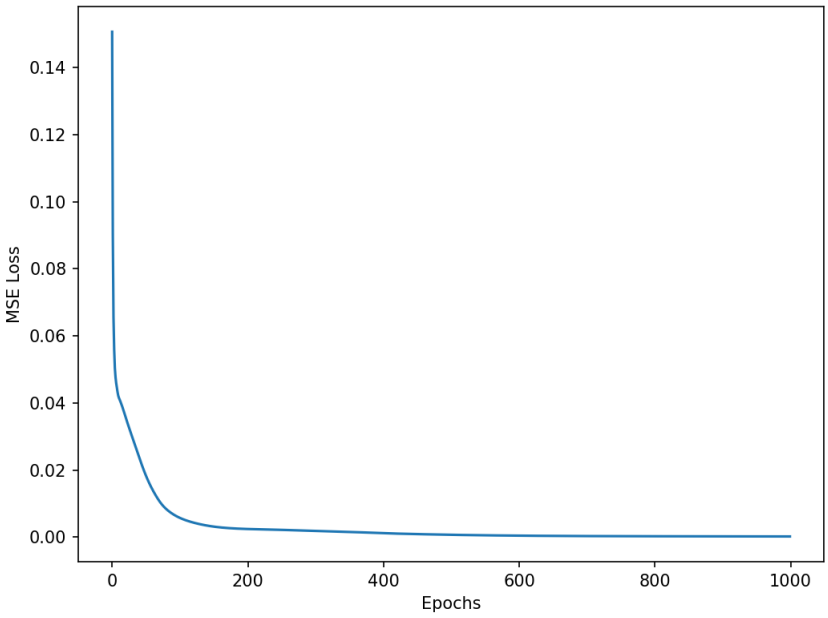


图 5: 训练 Loss 曲线

### 3.2 效果分析

- 从 Loss 上看，网络的训练过程总体稳定，Loss 稳步下降，但是由于 `sigmoid` 函数会带来的梯度消失的问题，导致训练后期 Loss 的很慢。
- 拟合的曲面较好的反应了真实曲面的特点，最大误差约为 0.5。绝大部分数据点的误差在 0.1 左右。

## 4 参考

周志华. (2016). 机器学习.