

# Back-Propagation-Neural-Network

language C++

last commit today

本项目是基于C++实现基础BP神经网络，有助于深入理解BP神经网络原理。

是对项目[GavinTechStudio/bpnn\\_with\\_cpp](https://github.com/GavinTechStudio/bpnn_with_cpp)的代码重构。

## 项目结构

```
1  .
2  ├── CMakeLists.txt
3  ├── README.md
4  ├── data
5  |   ├── testdata.txt
6  |   └── traindata.txt
7  ├── docs
8  |   └── formula.md
9  ├── img
10 |   └── net-info.png
11 ├── lib
12 |   ├── Config.h
13 |   ├── Net.cpp
14 |   ├── Net.h
15 |   ├── Utils.cpp
16 |   └── Utils.h
17 └── main.cpp
18
19 4 directories, 12 files
```

## 主要文件

- Net: 网络具体实现
- Config: 网络参数设置
- Utils: 工具类
  - 数据加载
  - 激活函数
- main: 网络具体应用

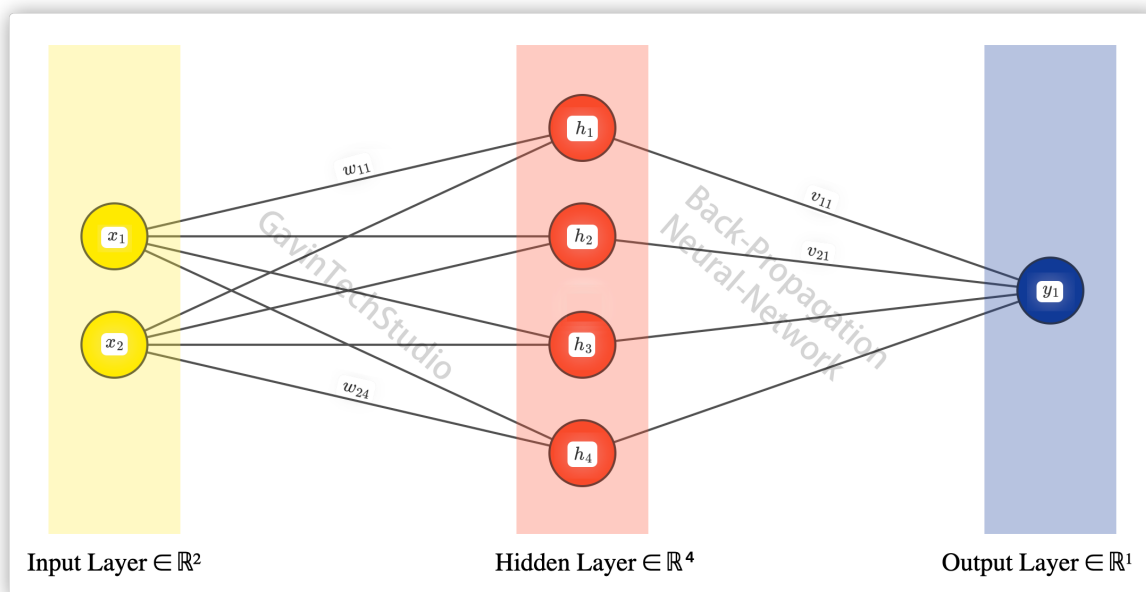
## 训练原理

具体公式推导请看视频讲解[彻底搞懂BP神经网络 理论推导+代码实现 \(C++\)](https://www.bilibili.com/video/BV18t4y1g7tY/) 哔哩哔哩bilibili

本部分文档包含大量数学公式，由于GitHub的README页面不支持数学公式渲染，推荐以下阅读方式：

1. 如果您使用的是Chrome、Edge、Firefox等浏览器，可以安装插件[MathJax Plugin for Github](https://chrome.google.com/webstore/detail/mathjax-plugin-for-github)（需要访问chrome web store）。
2. 使用[PDF](#)的方式进行阅读（**对网络无要求**）。
3. 使用[预渲染的静态网页](#)进行阅读（**对网络要求高，推荐**）。

## 0. 神经网络结构图



## 1. Forward（前向传播）

### 1.1 输入层向隐藏层传播

$$h_j = \sigma\left(\sum_i x_i w_{ij} - \beta_j\right)$$

其中  $h_j$  为第  $j$  个隐藏层节点的值， $x_i$  为第  $i$  个输入层节点的值， $w_{ij}$  为第  $i$  个输入层节点到第  $j$  个隐藏层节点的权重， $\beta_j$  为第  $j$  个隐藏层节点偏置值， $\sigma(x)$  为 **Sigmoid** 激活函数，后续也将继续用这个表达，其表达式如下

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

本项目中的代码实现如下：

```
1  for (size_t j = 0; j < Config::HIDENODE; ++j) {
2      double sum = 0;
3      for (size_t i = 0; i < Config::INNOD; ++i) {
4          sum += inputLayer[i]->value *
5              inputLayer[i]->weight[j];
6      }
7      sum -= hideLayer[j]->bias;
8
9      hideLayer[j]->value = Utils::sigmoid(sum);
10 }
```

### 1.2 隐藏层向输出层传播

$$\hat{y}_k = \sigma\left(\sum_j h_j v_{jk} - \lambda_k\right)$$

其中  $\hat{y}_k$  为第  $k$  个输出层节点的值（预测值）， $h_j$  为第  $j$  个隐藏层节点的值， $v_{jk}$  为第  $j$  个隐藏层节点到第  $k$  个输出层节点的权重， $\lambda_k$  为第  $k$  个输出层节点的偏置值， $\sigma(x)$  为激活函数。

本项目中的代码实现如下：

```

1  for (size_t k = 0; k < Config::OUTNODE; ++k) {
2      double sum = 0;
3      for (size_t j = 0; j < Config::HIDENODE; ++j) {
4          sum += hideLayer[j]->value *
5              hideLayer[j]->weight[k];
6      }
7      sum -= outputLayer[k]->bias;
8
9      outputLayer[k]->value = Utils::sigmoid(sum);
10 }

```

## 2. 计算损失函数 (Loss Function)

损失函数定义如下：

$$Loss = \frac{1}{2} \sum_k (y_k - \hat{y}_k)^2$$

其中 $y_k$ 为第 $k$ 个输出层节点的目标值（真实值）， $\hat{y}_k$ 为第 $k$ 个输出层节点的值（预测值）。

本项目中的代码实现如下：

```

1  double loss = 0.f;
2
3  for (size_t k = 0; k < Config::OUTNODE; ++k) {
4      double tmp = std::fabs(outputLayer[k]->value - out[k]);
5      los += tmp * tmp / 2;
6  }

```

## 3. Backward (反向传播)

利用梯度下降法进行优化。

### 3.1 计算 $\Delta\lambda_k$ (输出层节点偏置值的修正值)

其计算公式如下（激活函数为Sigmoid时）：

$$\Delta\lambda_k = -\eta(y_k - \hat{y}_k)\hat{y}_k(1 - \hat{y}_k)$$

其中 $\eta$ 为学习率（其余变量上方已出现过不再进行标注）。

本项目中的代码实现如下：

```

1  for (size_t k = 0; k < Config::OUTNODE; ++k) {
2      double bias_delta =
3          -(out[k] - outputLayer[k]->value) *
4          outputLayer[k]->value *
5          (1.0 - outputLayer[k]->value);
6
7      outputLayer[k]->bias_delta += bias_delta;
8  }

```

### 3.2 计算 $\Delta v_{jk}$ (隐藏层节点到输出层节点权重的修正值)

其计算公式如下 (激活函数为Sigmoid时) :

$$\Delta v_{jk} = \eta(y_k - \hat{y}_k)\hat{y}_k(1 - \hat{y}_k)h_j$$

其中 $h_j$ 为第 $j$ 个隐藏层节点的值 (其余变量上方已出现过不再进行标注) 。

本项目中的代码实现如下:

```
1  for (size_t j = 0; j < Config::HIDENODE; ++j) {
2      for (size_t k = 0; k < Config::OUTNODE; ++k) {
3          double weight_delta =
4              (out[k] - outputLayer[k]->value) *
5              outputLayer[k]->value *
6              (1.0 - outputLayer[k]->value) *
7              hideLayer[j]->value;
8
9          hideLayer[j]->weight_delta[k] += weight_delta;
10     }
11 }
```

### 3.3 计算 $\Delta \beta_j$ (隐藏层节点偏置值的修正值)

其计算公式如下 (激活函数为Sigmoid时) :

$$\Delta \beta_j = -\eta \sum_k (y_k - \hat{y}_k)\hat{y}_k(1 - \hat{y}_k)v_{jk}h_j(1 - h_j)$$

其中 $v_{jk}$ 为第 $j$ 个隐藏层节点到第 $k$ 个输出层节点的权重 (其余变量上方已出现过不再进行标注) 。

本项目中的代码实现如下:

```
1  for (size_t j = 0; j < Config::HIDENODE; ++j) {
2      double bias_delta = 0.f;
3      for (size_t k = 0; k < Config::OUTNODE; ++k) {
4          bias_delta +=
5              -(out[k] - outputLayer[k]->value) *
6              outputLayer[k]->value *
7              (1.0 - outputLayer[k]->value) *
8              hideLayer[j]->weight[k];
9      }
10     bias_delta *=
11         hideLayer[j]->value *
12         (1.0 - hideLayer[j]->value);
13
14     hideLayer[j]->bias_delta += bias_delta;
15 }
```

### 3.4 计算 $\Delta w_{ij}$ (输入层节点到隐藏层节点权重的修正值)

其计算公式如下 (激活函数为Sigmoid时) :

$$\Delta w_{ij} = \eta \sum_k (y_k - \hat{y}_k)\hat{y}_k(1 - \hat{y}_k)v_{jk}h_j(1 - h_j)x_i$$

其中 $x_i$ 为第 $i$ 个输入层节点的值 (其余变量上方已出现过不再进行标注) 。

本项目中的代码实现如下:

```
1  for (size_t i = 0; i < Config::INNOD; ++i) {
2      for (size_t j = 0; j < Config::HIDENODE; ++j) {
3          double weight_delta = 0.f;
4          for (size_t k = 0; k < Config::OUTNODE; ++k) {
5              weight_delta +=
6                  (out[k] - outputLayer[k]->value) *
7                  outputLayer[k]->value *
8                  (1.0 - outputLayer[k]->value) *
9                  hideLayer[j]->weight[k];
10         }
11         weight_delta *=
12             hideLayer[j]->value *
13             (1.0 - hideLayer[j]->value) *
14             inputLayer[i]->value;
15
16         inputLayer[i]->weight_delta[j] += weight_delta;
17     }
18 }
```