第五周上机课：神经网络-Matlab深度网络设计器

机器学习

机器学习致力于研究如何通过计算的手段，利用经验来改善系统自身的性能。
这里的经验即为学习的目标，它可以是：数据、现有模型、人类定义的规则集、模型自身。

机器学习的三大要素：1. 模型；2. 策略；3. 算法。

1. 模型：要学习什么样的模型？模型假设空间、参数范围、模型超参数
2. 策略：按何种准则进行学习？损失函数、代价函数
3. 算法：用何种优化算法学习？如何来计算和选择模型参数 / 类别

机器学习模型

机器学习往往被分为传统机器学习和深度学习两类。

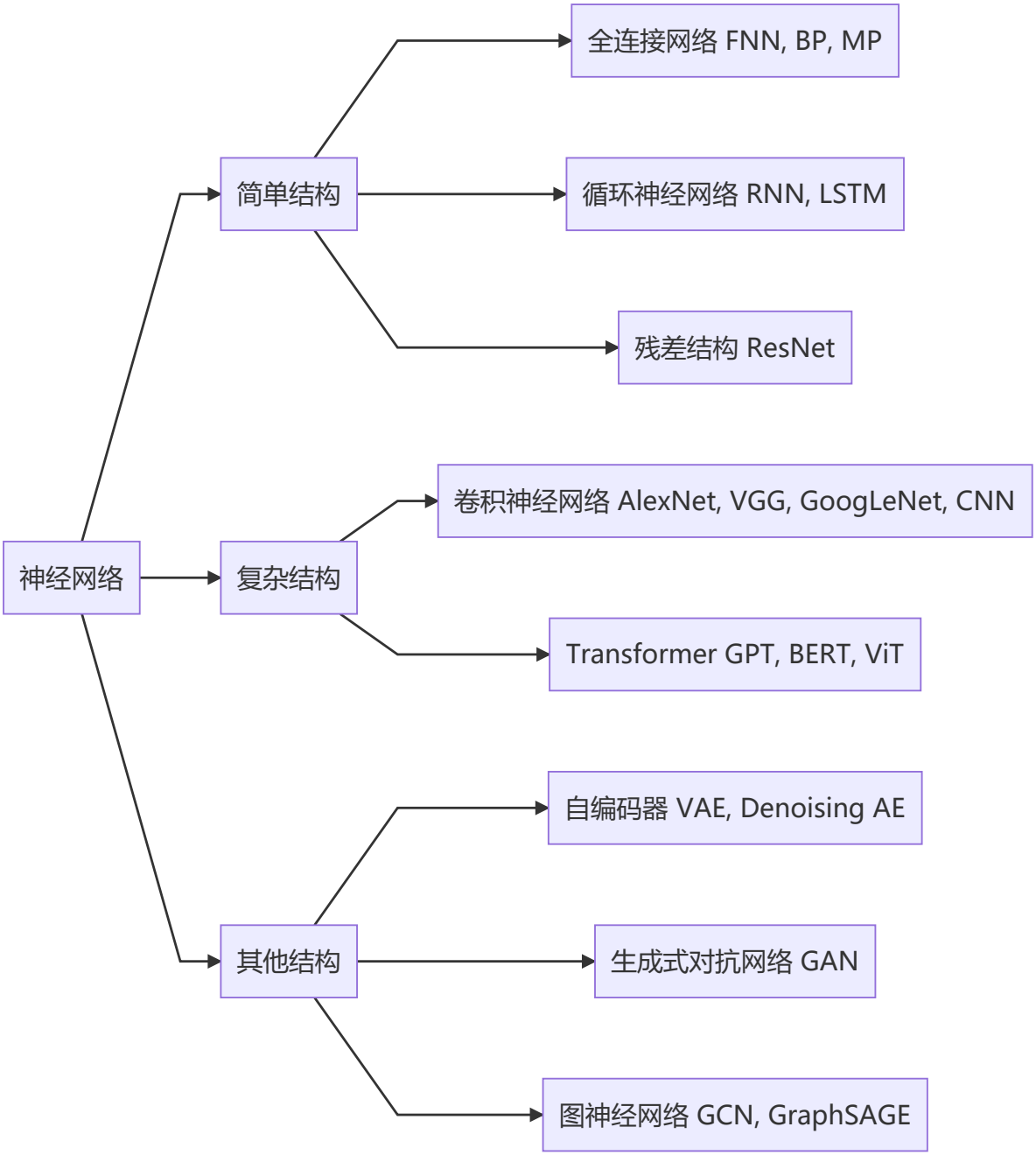
其中，传统机器学习主要是利用概率、统计推断等手段，在有严格数学框架的背景下，学习数据中潜藏的信息。

常见的传统机器学习模型有：

类别	核心任务	经典算法
监督学习	从有标签数据中学习预测模型	线性回归、逻辑回归、SVM、决策树、KNN、朴素贝叶斯
无监督学习	从无标签数据中发现模式	K-Means、DBSCAN、PCA、t-SNE
集成学习	组合多个模型提升性能	随机森林 (Bagging), XGBoost (Boosting)

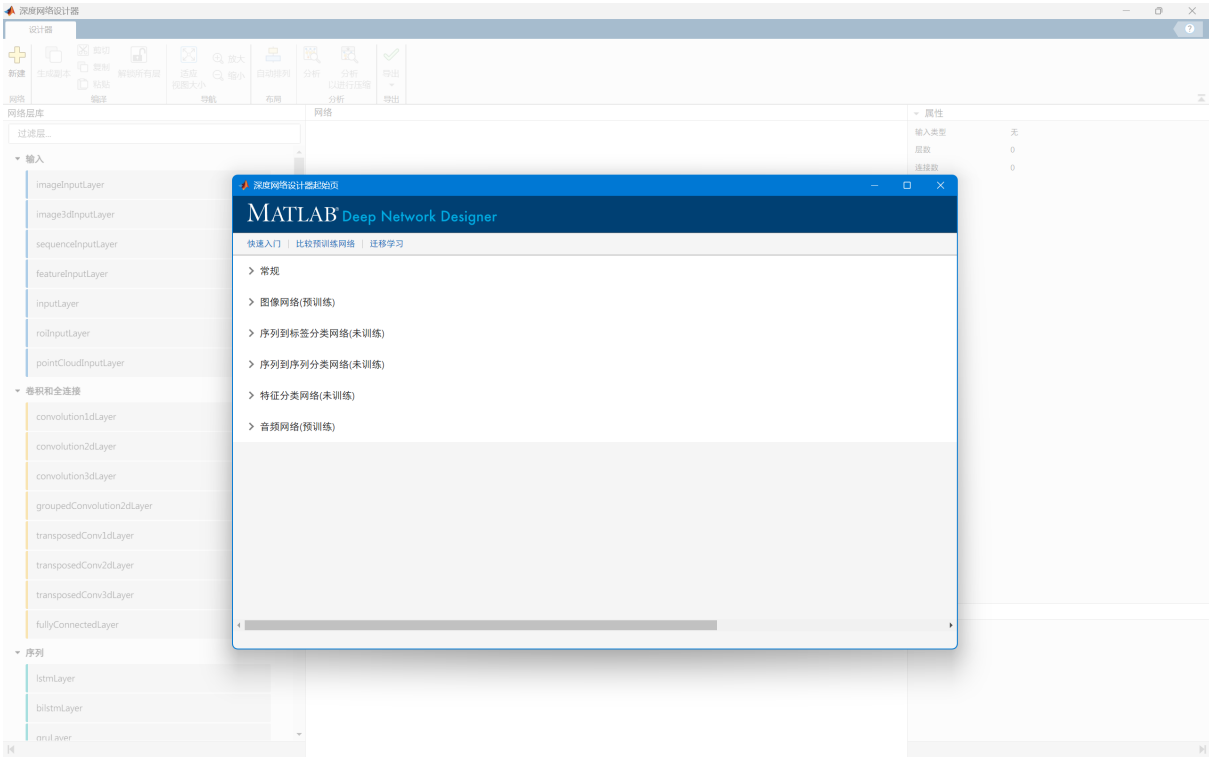
我们在第二周上机课中学到的线性判别方法是机器学习模型。

深度学习模型，使用了神经网络结构，基于广义逼近定理，在没有较好的收敛、稳定性等数学结果的前提下，实现对数据中潜在信息的挖掘和学习。目前神经网络已经可以用于解决分类、拟合、解方程、多模态识别等问题，大致可以分为如下的几类：



Matlab深度网络设计器

在命令行输入 `deepNetworkDesigner` 或者点击 “App-deepNetworkDesigner” 均可打开Matlab深度网络设计器。



起始页将包含不同的网络模板。其中，“预训练”标识代表着网络的参数有部分已经训练完成，将可以直接用于下游任务。

全连接神经网络

拟合问题

拟合问题，又称为回归问题，即根据已知的$\{x_1,\cdots,x_n\}$和其对应的象$\{y_1,\cdots,y_n\}$，找到最合适的对应法则 $y=f(x)$，使得$y_i=f(x_i)$近似成立。但是在现实中很难找到对应的f，或者说找到最接近真实的f，主要原因为真实测量得到的x_i,y_i往往有大量噪声，这也就导致其偏离了真实值，所以问题就转化为了一个最小化函数问题：

$$\min_f \sum_{i=1}^{n} \|y_i - f(x_i)\|_2^2$$

$$\min_f \sum_{i=1}^{n} L(y_i, f(x_i))$$

此时由于f的取值很复杂，所以根据f的表达式进一步将拟合分为了线性拟合、多项式拟合、样条拟合等。

f 所处的空间即为模型的假设空间。

L 为损失函数、代价函数 Loss function，criterion

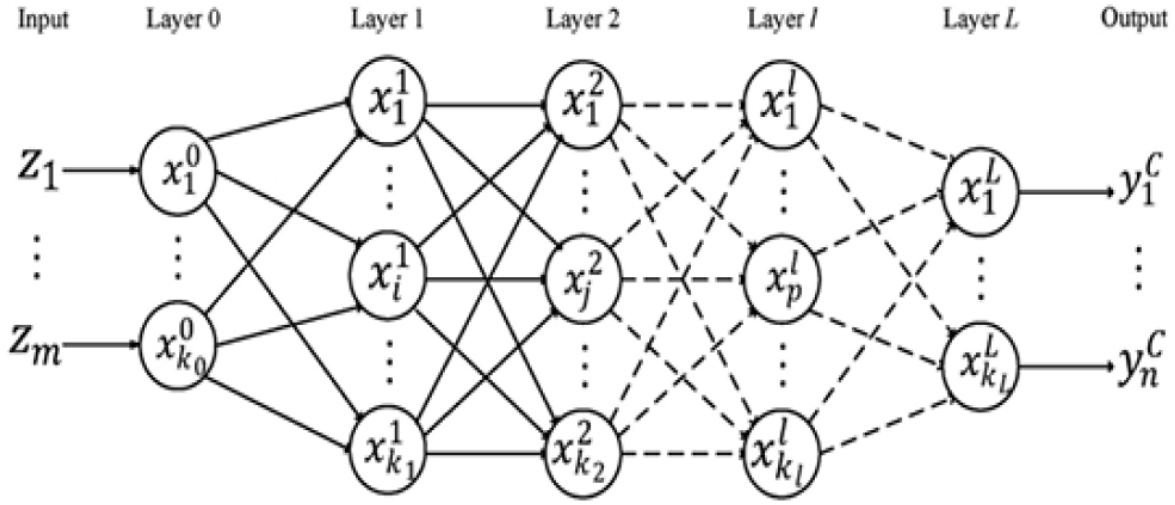
x_i 为输入特征，features，可以是多维的、图片、音频等

y_i 为标签，labels，targets，可以是多维的

全连接网络

大部分神经网络类似于一个黑箱子，给神经网络一个输入，并告诉他对应的输出，就可以让神经网络进行训练，最终实现给神经网络一个输入，就可以给定一个想要的输出。例如分类问题中，如果给一个神经网络200个照片，并告诉每一个照片对应的类别(猫或者狗)，就可以让神经网络进行训练，最终实现，给神经网络一个新的图片，可以识别出照片中是猫还是狗。为了让识别效果更好，神经网络往往需要特殊的架构和大量需要训练的参数，这也就导致了神经网络的可解释性很差。

然而对于全连接神经网络而言，基于其简单的架构，可以很容易了解其训练过程和更新本质。对于一个全连接神经网络，具有如图架构。



神经网络中第0层为输入层，第L层为输出层，中间层被称为隐藏层。对于图片分类问题而言，输入为一张图片，输出为分类结果，输入层的每一个神经元代表图片的一个像素输入，输入层的神经元个数为图片像素个数；而对于一元函数拟合问题，输入层输入的是已知的自变量值，输出层为因变量的值，神经元均各为1个。神经网络的正向传播模式为逐层转播，即首先从输入层输入样本，然后通过将样本的不同特征做线性组合后传入第一隐藏层，通过激活函数作用后，再将新样本的特征线性组会传入第二隐藏层，以此类推。

全连接神经网络看似简单，但是其有拟合任意函数的能力。万能近似定理（Universal Approximation Theorem）说的就是这样的事情。如果一个神经网络的激活函数是有界连续且单增的（如Sigmoid），则一个两层的全连接网络可以拟合任意紧集上的连续函数（可以参考Universal Approximation Bounds for Superpositions of a Sigmoidal Function ）。

但是这要求神经网络的宽度无限宽，而且对于如何学习到这样的参数没有指明。一般来说，深度的增加可以有效的降低宽度的要求，从而使得极端宽网络的近似能力能在有限宽但相对深的网络上体现出来。

不过随着网络深度的增长，将很快面临梯度爆炸和梯度消失问题，为此出现了残差结构。图像问题上超高维数的特征带来了难以承受的参数增长需求，这催生了卷积神经网络。

令$z \in \mathbb{R}^m, x^l \in \mathbb{R}^{k_l}, y^C \in \mathbb{R}^n$分别为样本在输入层，隐藏层，输出层的取值，为了方便，令$x^0 = z, k_0 = m$,$x^L = y^C$,$k_L = n$. 为了模拟神经网络正向传播过程，首先令

$$t_j^{l+1} = \sum_{i=1}^{k_l} w_{ji}^{l+1} x_i^l + b_j^{l+1}, \quad j = 1, \cdots, k_{l+1}, \quad l = 0, \cdots, L-1$$

其中w_{ji}^{l+1}为神经元x_i^l到x_j^{l+1}的连接权重，b_j^{l+1}为神经元x_j^{l+1}的偏置，x_i^l为向量x^l的第i个分量。进而在利用激活函数σ_l作用一次即可得到第$l+1$层神经元的取值，即为

$$x_j^{l+1} = \sigma_l(t_j^{l+1}), \quad j = 1, \cdots, k_{l+1}, \quad l = 0, \cdots, L-1,$$

其中激活函数主要类型为以下几种：

- sigmoid:

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

- Tanh:

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

- ReLU:

$$\sigma(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases};$$

- Leaky ReLU:

$$\sigma(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases}, \gamma > 0$$

这些激活函数的本质作用都是为神经网络提供一些非线性用于更好拟合非线性函数。

这种逐个神经元更新的方式比较繁琐，计算时也需要写多个循环。所以可以将其改写为矩阵格式，即假设有

$$
\begin{aligned}
&x^l = (x_1^l, \cdots, x_{k_l}^l)^T \in \mathbb{R}^{k_l},\\
&w^{l+1} = \{w_{ji}^{l+1}\}_{i=1,\cdots,k_l}^{j=1,\cdots,k_{l+1}} \in \mathbb{R}^{k_{l+1}\times k_l},\\
&b^{l+1} = (b_1^{l+1}, \cdots, b_{k_{l+1}}^{l+1})^T \in \mathbb{R}^{k_{l+1}},\\
&t^{l+1} = (t_1^{l+1}, \cdots, t_{k_{l+1}}^{l+1})^T \in \mathbb{R}^{k_{l+1}},
\end{aligned}
$$

此时全连接神经网络的正向传播过程可以写为

$$
\begin{cases}
t^{l+1} = w^{l+1}x^l + b^{l+1}\\
x^{l+1} = \sigma_l(t^{l+1})
\end{cases},
$$

其中$l = 0, \cdots, L-1$。

因此数学上，全连接神经网络就是线性变换和激活函数的多重复合。

反向传播

全连接神经网络的反向传播指的是从最后一层$L-1$层到L层的权重与偏置参数更新，再逐层回到输入层到隐藏层的参数更新。其更新方法需要用到梯度下降和矩阵微分工具。神经网络的拟合问题是指对于已知的输入输出$\{(z_l, y_l)\}_{l=1}^{n_1}$，其中$z_l \in \mathbb{R}^m, y_l \in \mathbb{R}^n$，令神经网络对于输入$z_l$时的输出为$y_l^C$。令

$$
Y = (y_1, \cdots, y_n) \in \mathbb{R}^{n\times n_1}, Y^C = (y_1^C, \cdots, y_n^C) \in \mathbb{R}^{n\times n_1}
$$

则神经网络拟合问题的目标函数为

$$
Loss(Y, Y^C) = \frac{1}{2n_1}\sum_{l=1}^{n_1}\|y_l - y_l^C\|^2,
$$

那么神经网络目标就是通过求解目标函数

$$
\min_{\{w_l\}_{l=1}^L, \{b_l\}_{l=1}^L} Loss(Y, Y^C)
$$

更新参数$\{w_l\}_{l=1}^L, \{b_l\}_{l=1}^L$。

此时可以利用梯度下降的方法更新参数，那么首先就要先计算损失函数关于$\{w_l\}_{l=1}^L, \{b_l\}_{l=1}^L$的梯度。此时需要用到矩阵求导工具，首先回忆矩阵微分满足的性质：

对于矩阵$X, Y, Z \in \mathbb{R}^{m\times n}, f: \mathbb{R}^m \to \mathbb{R}$，$\odot$代表矩阵对应元素相乘。 则矩阵微分满足

$$
\begin{aligned}
&d(X \pm Y) = dX \pm dY \\
&d(XY) = dXY + XdY \\
&df(X) = \mathrm{Tr}\left[\left(\frac{\partial f}{\partial X}\right)^T dX\right] \\
&\mathrm{Tr}(X \pm Y) = \mathrm{Tr}(X) \pm \mathrm{Tr}(Y) \\
&\mathrm{Tr}(X^T(Y \bigodot Z)) = \mathrm{Tr}((X \bigodot Y)^T Z)
\end{aligned}
$$

此时可以开始计算梯度，首先计算损失函数关于x^L的导数，则有

$$
\begin{aligned}
dLoss &= \mathrm{Tr}\left[\left(\frac{\partial f}{\partial x^L}\right)^T dx^L\right] \\
&= \mathrm{Tr}\left[\left(\frac{\partial f}{\partial x^L}\right)^T \sigma_L'(t^L) \bigodot dt^L\right] \\
&= \mathrm{Tr}\left[\left(\frac{\partial f}{\partial x^L} \bigodot \sigma_L'(t^L)\right)^T dt^L\right]
\end{aligned}
$$

此时有

$$
\delta^L = \frac{\partial Loss}{\partial t^L} = \frac{\partial f}{\partial x^L} \bigodot \sigma_L'(t^L)
$$

进一步对于$l = L-1, \cdots 1$，可以推导

$$
\begin{aligned}
dLoss &= \mathrm{Tr}\left[\left(\frac{\partial f}{\partial t^l}\right)^T dt^l\right] \\
&= \mathrm{Tr}\left[\left(\frac{\partial f}{\partial t^l}\right)^T d\left(w^l x^{l-1} + b^l\right)\right] \\
&= \mathrm{Tr}\left[\left(\delta^l\right)^T d\left(w^l\right) x^{l-1}\right] + \mathrm{Tr}\left[\left(\delta^l\right)^T w^l d\left(x^{l-1}\right)\right] + \mathrm{Tr}\left[\left(\delta^l\right)^T d\left(b^l\right)\right] \\
&= \mathrm{Tr}\left[\left(\delta^l\right)^T d\left(w^l\right) x^{l-1}\right] + \mathrm{Tr}\left[\left(\delta^l\right)^T w^l \left(\sigma_{l-1}'(t^{l-1}) \bigodot dt^{l-1}\right)\right] + \mathrm{Tr}\left[\left(\delta^l\right)^T d\left(b^l\right)\right] \\
&= \mathrm{Tr}\left[\left(\delta^l\right)^T d\left(w^l\right) x^{l-1}\right] + \mathrm{Tr}\left\{\left[\left(\left((w^l)^T \delta^l\right) \bigodot \sigma_{l-1}'(t^{l-1})\right)^T dt^{l-1}\right]\right\} + \mathrm{Tr}\left[\left(\delta^l\right)^T d\left(b^l\right.\right.
\end{aligned}
$$

即根据矩阵微分有

$$
\begin{aligned}
&\delta^{l-1} = \frac{\partial Loss}{\partial t^{l-1}} = \left((w^l)^T \delta^l\right) \bigodot \sigma_{l-1}'(t^{l-1}), \\
&\frac{\partial Loss}{\partial w^l} = \delta^l (x^{l-1})^T, \\
&\frac{\partial Loss}{\partial b^l} = \delta^l,
\end{aligned}
$$

所以求梯度时按照如下顺序求解即可：

$$\delta^L \to \delta^{L-1} \to \cdots \to \delta^1,$$

进而通过δ求解$\frac{\partial Loss}{\partial w^l}$和$\frac{\partial Loss}{\partial b^l}$。

随着梯度求出，就可以更新参数，第s次迭代更新公式为：

$$w^{l,(s)} = w^{l,(s-1)} - \eta_s \frac{\partial Loss}{\partial w^{l,(s)}}$$
$$b^{l,(s)} = b^{l,(s-1)} - \eta_s \frac{\partial Loss}{\partial b^{l,(s)}}$$

代码流程

首先给定神经网络每一层的权重初值，例如可以使用标准正态分布生成随机参数，然后根据输入值和神经网络
的运算法则得到输出$Y^{C,(1)}$, 进而利用梯度下降法更新权重与偏置，在新的权重与偏置下重新计算输出$Y^{C,(s)}$, 以此
类推，直到达到迭代次数或达到收敛性要求。

Example

实现神经网络并用于拟合，其中神经网络参数为输入层1 个神经元，隐藏层20 个神经元，输出层1 个神经元，
只有一个隐藏层，迭代停止法则为迭代1000 次，网络参数初始值为由标准正态分布生成，输入数据为随机生成，即x = rand(1,1000), 输出为
$y = x + sin(x) + 1e^{-6} * randn(1, 1000)$. 最终结果输出为神经网络训练结果和
真实结果的误差，即输出一个数字

```
% 初始化参数和权重
n = 100;
x = rand(1,n);
x = sort(x);
y = x + sin(x) + 1e-2 * randn(1, n);
w1 = randn(20, 1);
b1 = randn(20, 1);
w2 = randn(1, 20);
b2 = randn(1, 1);
eta = 1e-3;
error_set = zeros(1, 1000);

% 训练循环
for iter = 1:1000
    % 前向传播
    t1 = w1 * x + repmat(b1, 1, n);
```

```
    x1 = t1 .* (t1 > 0);
    t2 = w2 * x1 + repmat(b2, 1, n);
    x2 = t2;
    error_set(iter) = norm(x2 - y);

    % 反向传播和权重更新
    delta2 = (x2 - y);
    delta1 = (w2' * delta2) .* (t1 > 0);
    b2 = b2 - eta * mean(delta2, 2);
    w2 = w2 - eta * mean(delta2 * x1', 2);
    b1 = b1 - eta * mean(delta1, 2);
    w1 = w1 - eta * mean(delta1 * x', 2);
end

% 显示最终误差
disp(error_set(end));
plot(x,x2,'r');
hold on
plot(x,y,'b')
```

作业

实现神经网络并用于拟合，其中神经网络参数为输入层2个神经元，第一个隐藏层20 个神经元，第二个隐藏层20 个神经元，输出层 1 个神经元，迭代停止法则为迭代1000 次，网络参数初始值为由标准正态分布生成，输入数据为随机生成，服从$[-8,8]^2$中的均匀分布，数据量N不超过10000，输出为

$$Y = sinc(X) = sin(r)/r$$

最终结果输出为神经网络训练结果和真实结果的相对误差。

$$err = \frac{1}{N}\sum_{i=1...N}\frac{\|Y_i^{pred} - Y_i^{true}\|_2}{\|Y_i^{true}\|_2}$$

考虑到可能出现的除0错误，你应该在生成数据时避免采样0附近$1e^{-6}$附近的点。

你可以自定义除了输入维数、隐藏层个数**>=2**之外其他的网络超参数，以获得更好的结果。

应提交：

- 报告pdf/doc：包含相对误差，真实函数值二维图、预测函数值二维图、绝对误差二维图。

- 必要的代码文件

```
% 创建网格数据
[x, y] = meshgrid(-8:0.25:8);  % 从-8到8，步长0.25

% 计算到原点的距离
r = sqrt(x.^2 + y.^2);

% 计算二维sinc函数值
z = sinc(r / pi);  % 等价于 sin(r) / r

% 绘制三维曲面
figure;
surf(x, y, z);
xlabel('X');
ylabel('Y');
zlabel('Z');
title('二维Sinc函数 - 曲面图');
colorbar;  % 显示颜色条
shading interp;  % 平滑着色
```