第八周上机课

Matlab中的数据

dlarray 是matlab神经网络应用中所使用的数据格式，类似于Pytorch中的 Tensor 张量。

创建一个dlarray对象是非常简单的：

```
x_dl = dlarray([1, 2, 3, 4]);
```

可以接收向量或者矩阵作为输入。

dlarray 对象和matlab自带的矩阵数据一样，支持很多运算。例如下面的计算：

```
y_dl = sin(x_dl) + x_dl.^2;
fprintf('y = sin(x) + x² = '); disp(extractdata(y_dl));
```

其中sin可以接收dlarray作为输入，就像直接接收矩阵或者向量作为输入一样。函数是逐点进行计算的。

extractdata函数接收一个dlarray 对象作为输入，输出这个dlarray内部的数据，返回值是矩阵或者向量。

dlarray 对象和matlab自带的矩阵数据结构最大的**区别**在于它拥有非常完备的高维表示技术。即**维度标签**：

当创建 dlarray对象的时候，可以指定维度标签：

```
% 理解维度标签
A = dlarray(rand(3,4), 'CB');  % C:通道维度, B:批处理维度
```

数据格式，指定为字符串标量或字符向量。字符串中的每个字符必须为以下维度标签之一：

- `"S"` - 空间
- `"C"` - 通道
- `"B"` - 批量
- `"T"` - 时间
- `"U"` - 未指定

可以指定多个标注为 `"S"` 或 `"U"` 的维度。每个 `"C"`、`"B"` 和 `"T"` 标签最多可以使用一次。

当您创建格式化的 `dlarray` 对象时，软件会自动置换维度，使格式的维度采用以下顺序：

- `"S"`
- `"C"`
- `"B"`
- `"T"`
- `"U"`

例如，如果您指定 `"TCB"` 格式（时间、通道、批量），则软件会自动置换维度，使其具有 `"CBT"` 格式（通道、批量、时间）。

`fmt` 包含的维度标签数必须至少与输入数据的维数相同。如果您指定的维度标签数超过该维数，则 `dlarray` 会为额外的标签创建空（单一）维度。

创建一个简单的神经网络

```
% 网络结构：输入1维 -> 隐藏层30神经元 -> 隐藏层30神经元 -> 输出1维
layers = [
    featureInputLayer(1, 'Name', 'input')  % 输入层
    fullyConnectedLayer(30, 'Name', 'fc1') % 全连接层
    tanhLayer('Name', 'tanh1')             % 激活函数
    fullyConnectedLayer(30, 'Name', 'fc2')
    tanhLayer('Name', 'tanh2')
    fullyConnectedLayer(1, 'Name', 'output'), % 输出层
];

% 创建dlnetwork
net = dlnetwork(layers);
fprintf('神经网络构建完成！\n');
fprintf('可学习参数数量: %d\n', numel(net.Learnables.Value));

% 显示网络结构
analyzeNetwork(net);
```

除了这种简单的layer方式创建神经网络，还可以使用深度神经网络工具箱进行搭建。

训练一个神经网络

以拟合 sin 函数为例。

首先需要创建训练数据集，使用dlarray创建数据：

```
x_train = linspace(-pi, pi, 100)';
y_train = sin(x_train);

% 转换为dlarray
x_dl = dlarray(x_train', 'CB');  % 1×100, CB格式
y_dl = dlarray(y_train', 'CB');
```

定义损失计算函数

```
% 损失计算函数
function [loss, gradients] = computeLoss(net, x, y)
    % 前向传播
    y_pred = forward(net, x);

    % 计算损失
    loss = mse(y_pred, y);

    % 计算梯度 - 使用 net.Learnables 作为第二个参数
    gradients = dlgradient(loss, net.Learnables);
end
```

损失计算函数几乎总是需要自己实现。接收网络net、自变量x、响应y作为输入。输出是损失和梯度。

其中，forward函数、mse函数和dlgradient函数都是matlab已经封装好的函数。

作为训练过程中的前向计算，应该使用forward函数，此时dropout层将会起作用（随机将输入置为0）。
如果是作为测试或者验证网络的输出，应该使用predict函数，此时dropout层将失效。

dlgradient是matlab自带的梯度计算函数，第一个输入是损失，第二个输入是被求导的变量。注意！由于这里是训练过程，需要计算对所有参数的梯度，因此第二个变量为net.Learnables。

进行训练过程

```
% 训练参数
numEpochs = 1000;
learningRate = 0.01;

% 训练过程记录
lossHistory = zeros(numEpochs, 1);

figure(1);
vel = [];
for epoch = 1:numEpochs
    % 使用 dlfeval 包装损失和梯度计算
    [loss, gradients] = dlfeval(@computeLoss, net, x_dl, y_dl);
    lossHistory(epoch) = extractdata(loss);

    % 更新网络参数
    [net, vel] = sgdmupdate(net, gradients, vel, learningRate, 0.99);

    % 每50轮显示一次结果
    if mod(epoch, 50) == 0
        fprintf('Epoch %d, Loss = %.4e\n', epoch, lossHistory(epoch));
    end
end
```

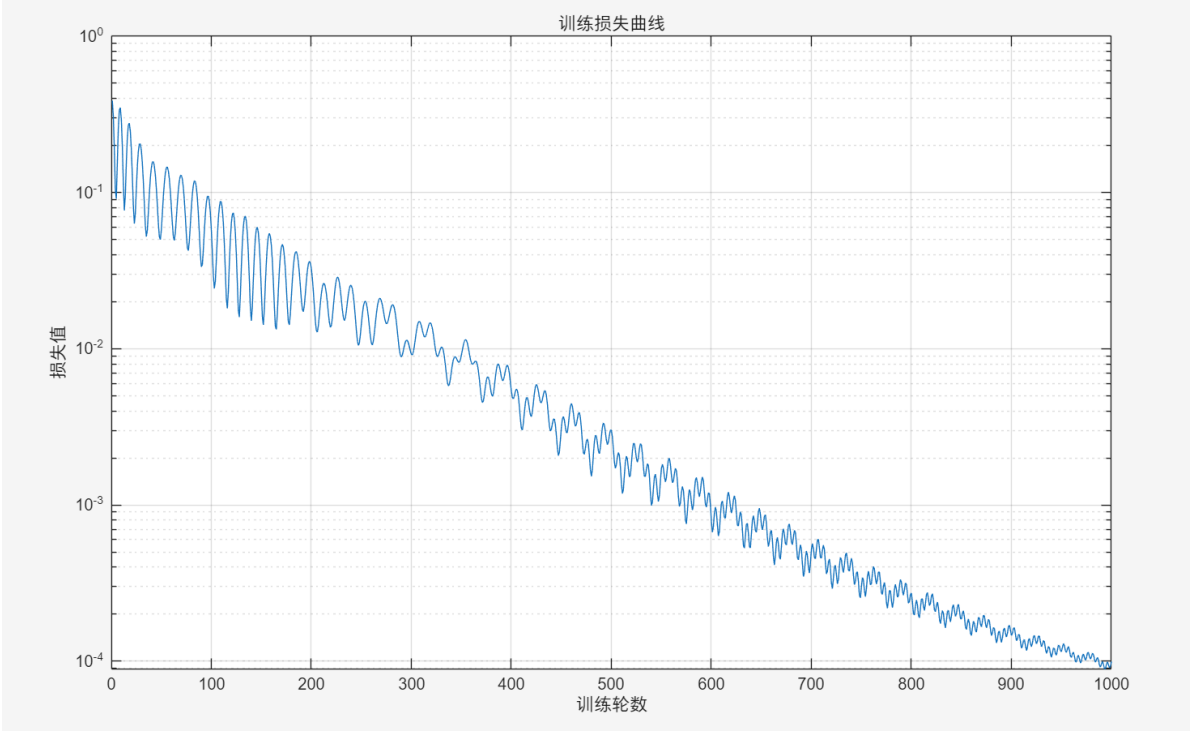
dlfeval函数用于客制化的训练过程，第一个输入是之前定义过的损失计算函数。

网络的参数更新过程使用了SGD算法，其中vel参数存储了历史梯度信息的加权和，为参数更新提供“惯性”，用于更新动量，最后一个参数是动量因子。

```
vel = momentum * vel - learningRate * gradient
parameters = parameters + vel
```

最后绘制训练过程：

```
% 绘制损失曲线
figure(2);
semilogy(lossHistory);
xlabel('训练轮数');
ylabel('损失值');
title('训练损失曲线');
grid on;
```



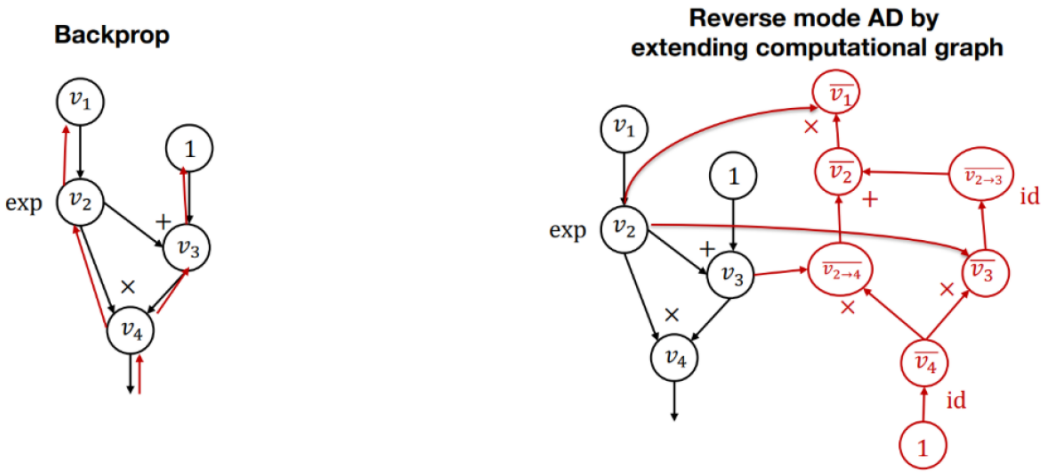
自动微分

利用 dlgradient 可以方便的计算神经网络对于 输入 / 参数 的各阶梯度。计算原理为第七次课中的自动微分过程。

```
% 定义测试函数
test_x = dlarray(1.0);  % 在x=1处计算导数

% 定义函数 f(x) = x² + sin(x)
f = @(x) x.^2 + sin(x);

% 使用dlfeval和dlgradient计算导数
[func_value, derivative] = dlfeval(@computeFunctionAndDerivative, f, test_x);

fprintf('在 x = 1.0 处：\n');
fprintf('f(x) = x² + sin(x) = %.4f\n', extractdata(func_value));
fprintf('f''(x) = 2x + cos(x) = %.4f (自动微分)\n', extractdata(derivative));
fprintf('解析解: 2*1 + cos(1) = %.4f\n', 2*1 + cos(1));

% 计算二阶导数
[~, second_deriv] = dlfeval(@computeSecondDerivative, f, test_x);
fprintf('f''''(x) = %.4f (自动微分)\n', extractdata(second_deriv));
fprintf('解析解: 2 - sin(1) = %.4f\n', 2 - sin(1));
```

```
% 函数值和一阶导数计算
function [f_val, df_dx] = computeFunctionAndDerivative(fun, x)
    f_val = fun(x);
    df_dx = dlgradient(f_val, x);
end

% 二阶导数计算
function [f_val, d2f_dx2] = computeSecondDerivative(fun, x)
    f_val = fun(x);
    df_dx = dlgradient(f_val, x, 'EnableHigherDerivatives', true);
    d2f_dx2 = dlgradient(df_dx, x);
end
```

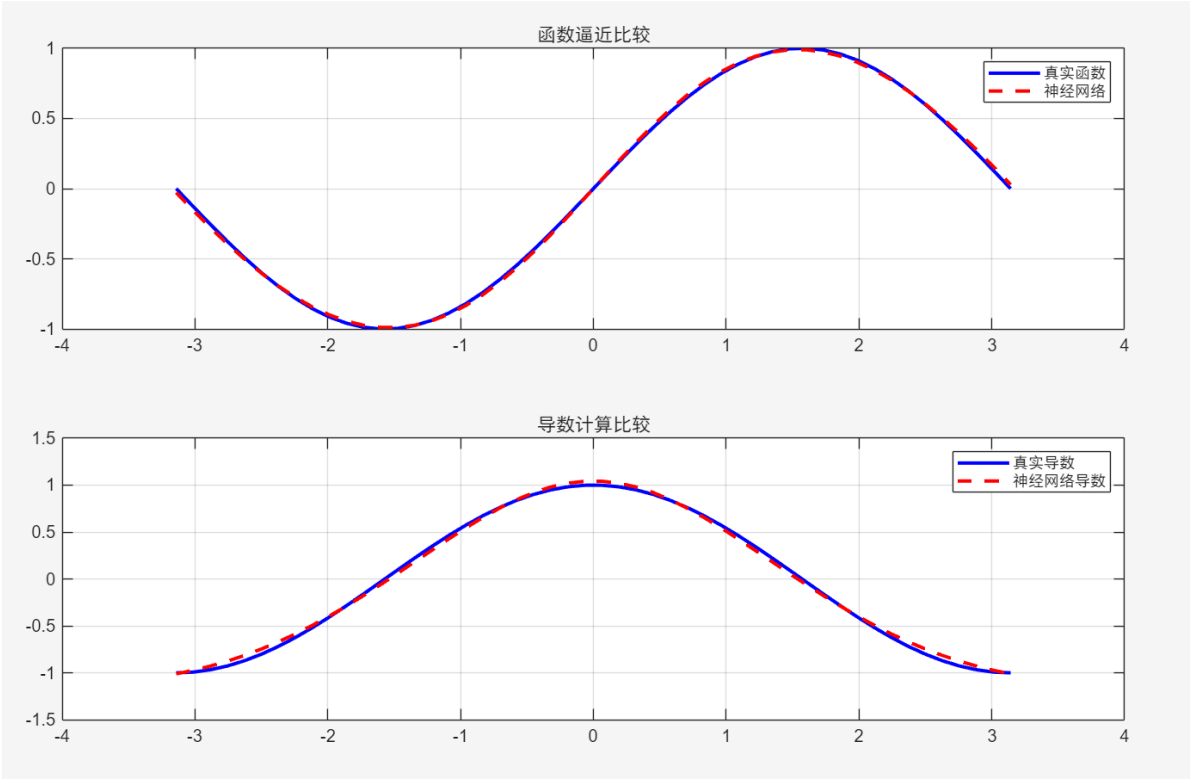
需要注意的是，计算二阶导数时，需要将 `'EnableHigherDerivatives'` 设置为 `true`。这是因为，maltab属于第七次课中讲到的使用增广计算图的架构来计算梯度。因此，额外的计算图将会在梯度计算结束后释放掉，即如下右图中红色节点所占用的内存。



如果需要计算高阶梯度，需要再增加计算图节点，因此使用 EnableHigherDerivatives 参数将计算图保存下来。

可以对比一下自动微分得到的梯度结果和精确解的区别：

```
% 使用训练好的网络计算导数
x_test = dlarray(0.5);
[net_output, net_derivative] = dlfeval(@computeNetDerivative, net, x_test);

fprintf('在 x = 0.5 处：\n');
fprintf('神经网络输出：%.4f\n', extractdata(net_output));
fprintf('神经网络导数：%.4f\n', extractdata(net_derivative));
fprintf('cos(0.5) = %.4f (真实导数)\n', cos(0.5));

% 绘制导数比较
figure(3);
x_range = linspace(-pi, pi, 50);
plotDerivativeComparison(net, x_range);
```



作业

1. 使用matlab在 $x \in [-2, 2]$ 上逼近如下函数：

$$y = x^3 - 2x + e^{-x}$$

讨论不同网络网络结构（隐藏层数量、神经元数量、激活函数（tanh vs sigmoid））对拟合结果的影响。

2. 第1问训练结束后，使用自动微分技术，在 $x \in [-3, 3]$ 上计算网络的一阶导数、二阶导数。

与解析解进行比较，计算相对误差并**绘制对比图**