第七周上机课

> 介绍自动微分技术、计算图原理
> 手动编写一个自动微分代码 *with Matlab*

自动微分

为什么需要计算梯度？

在神经网络的训练中，我们总是使用梯度下降的方法来优化神经网络的参数，从而减少神经网络的经验误差，来达到训练目的（拟合/分类）。从数学上，这是一个函数的优化问题：

$$\min_{\{w_i,b_i\}_i} L(Data;w,b)=\frac{1}{|Data|}\sum_{\{x_k,y_k\}\in Data}\|f(x_k;w,b)-y_k\|_2^2$$

损失函数 L 可以看作是数据的函数，亦可以看作神经网络参数 w,b 的函数。要最小化等式的右边，可以利用“梯度方向是函数增加最快的方向”这一特点，将网络参数向负梯度方向优化，减小经验误差。

$$w^{k+1}=w^k-\eta\cdot\nabla_w L(x;w)$$

因此，更新参数的关键在于计算出损失函数相对于参数的梯度！

损失函数 L 的表达式里面有神经网络 $f(x;w)$，要想高效准确的计算 $\nabla_x L$，就必须要解决如何高效的计算 $\nabla_w f$.

$$L=\frac{1}{|Data|}\sum_{\{x_k,y_k\}\in Data}\|f(x_k;w,b)-y_k\|_2^2$$

$$\nabla_w L=\frac{2}{|Data|}\sum_{\{x_k,y_k\}\in Data}(f(x_k;w,b)-y_k)\cdot\nabla_x f$$

神经网络训练的问题转移到如何高效计算 $\nabla_w f$.

如何计算梯度？

通过前面的学习，我们已经知道，神经网络（全连接神经网络）函数 f 本质上是一系列激活函数和线性变换的复合。它的梯度可以由以下方法计算：

1. 手动求导。

   没错是手动计算。事实上，第五次作业的例程中的梯度就是手动写出来的。

   ```
   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);
   ```

   手动计算梯度明显是较为繁琐的，特别是网络层数较多，激活函数较为复杂时候更是如此。

2. 数值微分

   利用微分的差分近似来计算梯度 / 微分。这是在微分方程数值解领域非常常用的梯度计算方法。

   $$f'(x_0) \approx \frac{f(x_0+h)-f(x_0-h)}{2h},\ 0<h\ll 1$$

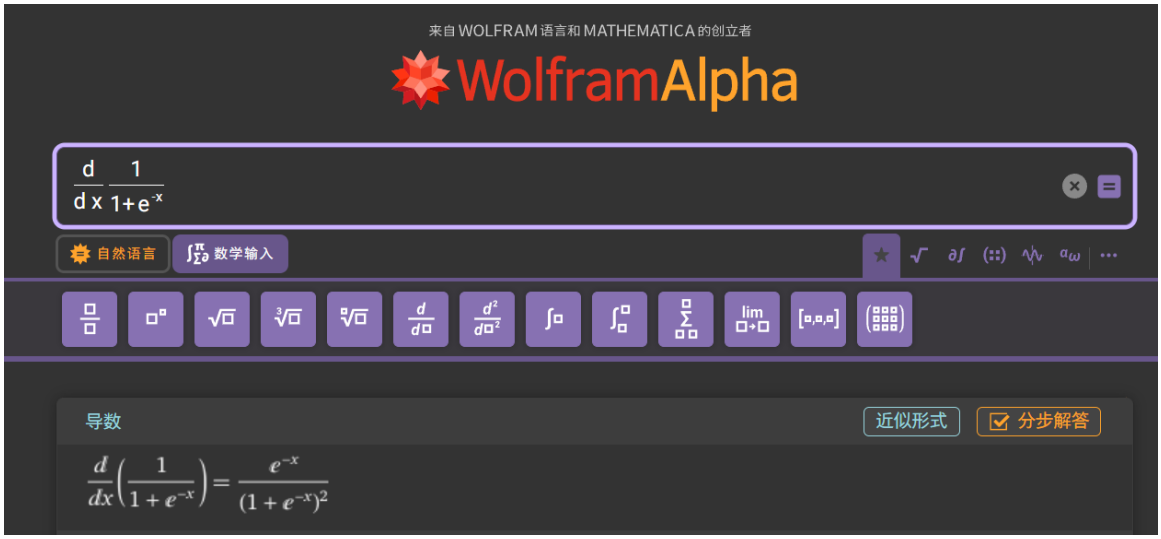
   因此，只需要两次前向计算，就可以得到梯度。

   这种方法非常简单，缺点在于不够高效（每次梯度计算需要两次前向计算）；精度不高（差分近似本身就有误差），网络很难收敛到局部极小值点。

   不过，目前仍然有相当多的现代神经网络文献使用这种方法辅助计算梯度，原因在于网络结构非常复杂时，反向传播方法求解梯度的代价很高，远不如两次前向传播快。同时误差问题可以使用数学理论来精细的选择h的大小来控制。

3. 符号微分

   使用符号推理机器，将导函数显式的计算出来，然后进行存储并计算。本质上是将手动求导的工作交给了Mathematica等现代符号推理软件完成。这是一类非常可靠的方法，优点在于精确，梯度的计算是显式的，几乎没有任何误差，同时无需额外的数据存储成本（相对于自动微分技术）。缺点在于这个显式的导函数非常非常难看，以至于编码工作也难以完成。



在目标函数表达式较为容易的时候，这仍应该成为首选方法。

4. 自动微分

   自动微分克服了上述的诸多困难，当然，同时也付出了“一点代价”。

   自动微分便于编码（优于符号微分），梯度求解是几乎精确的（优于数值微分），获得所有梯度只需要一遍计算（优于数值微分），适合大型神经网络梯度计算（优于手动求导）。

   代价是，网络本身（计算图）需要额外的存储空间，来存储当前的梯度值。

在讲解什么是自动微分，以及它的工作原理之前，有必要先介绍一下什么是计算图，以及在计算机中，如何实现神经网络的存储和计算。

计算图

什么是计算图？

图由节点和边构成，边可以是有向或者无向的。

计算图（Computation Graph）是一种用于表示数学计算或程序执行的图结构，是有向无环图。

在计算图中：

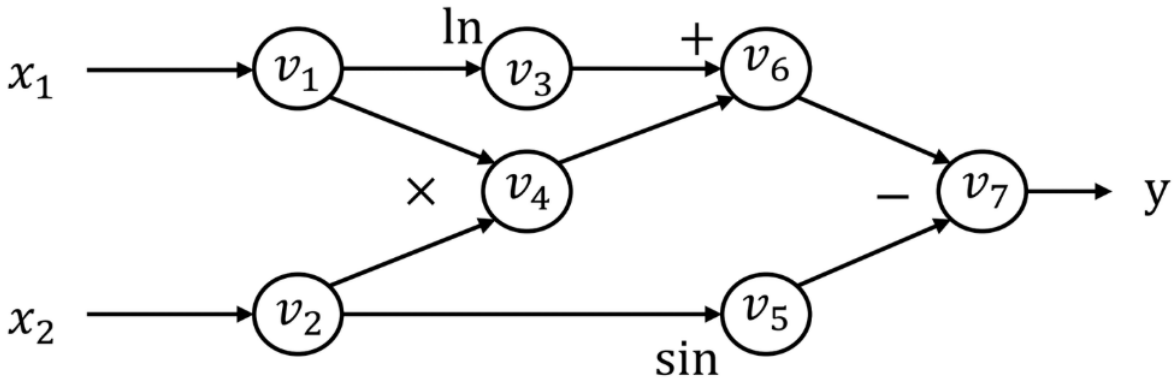
- 节点表示一种运算 或 变量，叶子节点仅表示变量
- 边代表数据的流向

例如：

在计算机中表示如下的计算过程：

$$f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin x_2$$

就有如下的计算图：



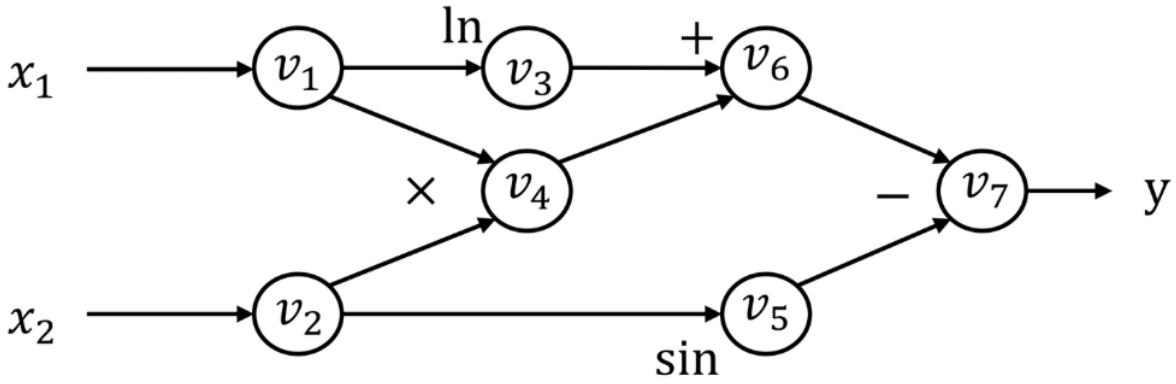
正向计算求值的过程我们称为正向传播，即给输入，计算输出的值。以 $x_1 = 2, x_2 = 5$为例。

$$\begin{aligned}
v_1 &= x_1 = 2 \\
v_2 &= x_2 = 5 \\
v_3 &= \ln v_1 = \ln 2 = 0.693 \\
v_4 &= v_1 \times v_2 = 10 \\
v_5 &= \sin v_2 = \sin 5 = -0.959 \\
v_6 &= v_3 + v_4 = 10.693 \\
v_7 &= v_6 - v_5 = 10.693 + 0.959 = 11.652 \\
y &= v_7 = 11.652
\end{aligned}$$

计算图如何求导？

正向计算导数

从叶子节点x_1, x_2开始，逐个计算当前节点的值对原始输入 x_1, x_2的梯度。例如，计算上面的计算图中所有节点对输入 x_1 的梯度。记 $\dot{v_i} = \frac{\partial v_i}{\partial x_1}$. 以 $x_1 = 2, x_2 = 5$为例。



$$
\begin{aligned}
\dot{v}_1 &= 1 \\
\dot{v}_2 &= 0 \\
\dot{v}_3 &= \dot{v}_1 / v_1 = 0.5 \\
\dot{v}_4 &= \dot{v}_1 v_2 + \dot{v}_2 v_1 = 1 \times 5 + 0 \times 2 = 5 \\
\dot{v}_5 &= \dot{v}_2 \cos v_2 = 0 \times \cos 5 = 0 \\
\dot{v}_6 &= \dot{v}_3 + \dot{v}_4 = 0.5 + 5 = 5.5 \\
\dot{v}_7 &= \dot{v}_6 - \dot{v}_5 = 5.5 - 0 = 5.5
\end{aligned}
$$

注意，上述的正向计算中的导数公式仍然使用了链式法则，具体规则由节点的类型决定。

正向导数计算非常自然，可以一边计算节点值v_i一边计算导数，但是不是目前的主流方法，这是因为神经网络通常是一个高维输入低维输出的函数，上述的正向梯度计算过程需要对每一个输入都进行一遍，才能够获得所有的梯度，计算代价很高。

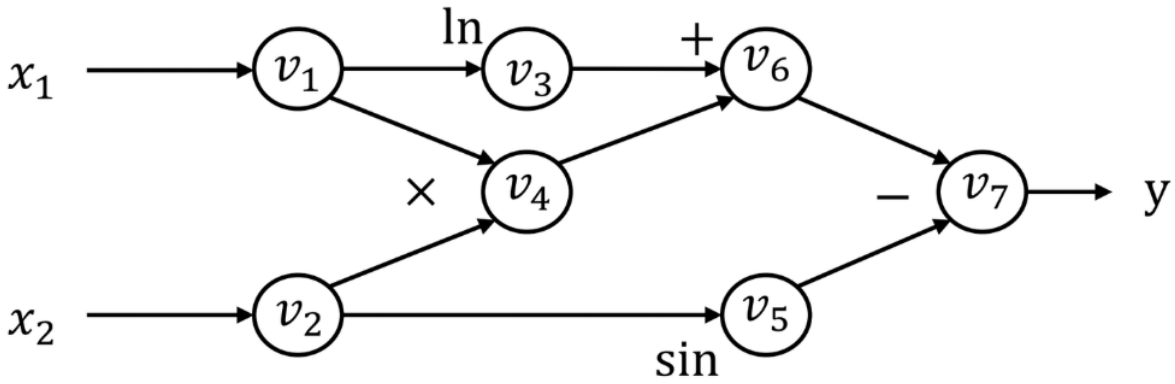
反向计算梯度（反向传播）

反向传播中，我们首先从输出节点开始，反向逐个计算最终输出对当前节点的梯度。

区别：

- 正向传播：从前往后，逐个计算当前节点对原始输入 x_1, x_2的梯度。$\frac{\partial v_i}{\partial x_1}$
- 反向传播：从后往前，逐个计算最终输出结果对当前节点的梯度。$\frac{\partial y}{\partial v_i}$

记 $\overline{v_i} = \frac{\partial y}{\partial v_i}$. 以 $x_1 = 2, x_2 = 5$为例。



$$
\begin{aligned}
\overline{v_7} &= \frac{\partial y}{\partial v_7} = 1 \\
\overline{v_6} &= \overline{v_7}\frac{\partial v_7}{\partial v_6} = \overline{v_7} \times 1 = 1 \\
\overline{v_5} &= \overline{v_7}\frac{\partial v_7}{\partial v_5} = \overline{v_7} \times (-1) = -1 \\
\overline{v_4} &= \overline{v_6}\frac{\partial v_6}{\partial v_4} = \overline{v_6} \times 1 = 1 \\
\overline{v_3} &= \overline{v_6}\frac{\partial v_6}{\partial v_3} = \overline{v_6} \times 1 = 1 \\
\overline{v_2} &= \overline{v_5}\frac{\partial v_5}{\partial v_2} + \overline{v_4}\frac{\partial v_4}{\partial v_2} = \overline{v_5} \times \cos v_2 + \overline{v_4} \times v_1 = -0.284 + 2 = 1.716 \\
\overline{v_1} &= \overline{v_4}\frac{\partial v_4}{\partial v_1} + \overline{v_3}\frac{\partial v_3}{\partial v_1} = \overline{v_4} \times v_2 + \overline{v_3}\frac{1}{v_1} = 5 + \frac{1}{2} = 5.5
\end{aligned}
$$

最后一行，$\overline{v_1}$ 作为两个节点的输入，因此输出函数对其的梯度应该由 v_3, v_4 共同提供，可以理解为：

$$
\begin{aligned}
y &= f(v_3, v_4) = f(v_3(v_1), v_4(v_1)) \\
\frac{\partial y}{\partial v_1} &= \frac{\partial y}{\partial v_3} \cdot \frac{\partial v_3}{\partial v_1} + \frac{\partial y}{\partial v_4} \cdot \frac{\partial v_4}{\partial v_1}
\end{aligned}
$$

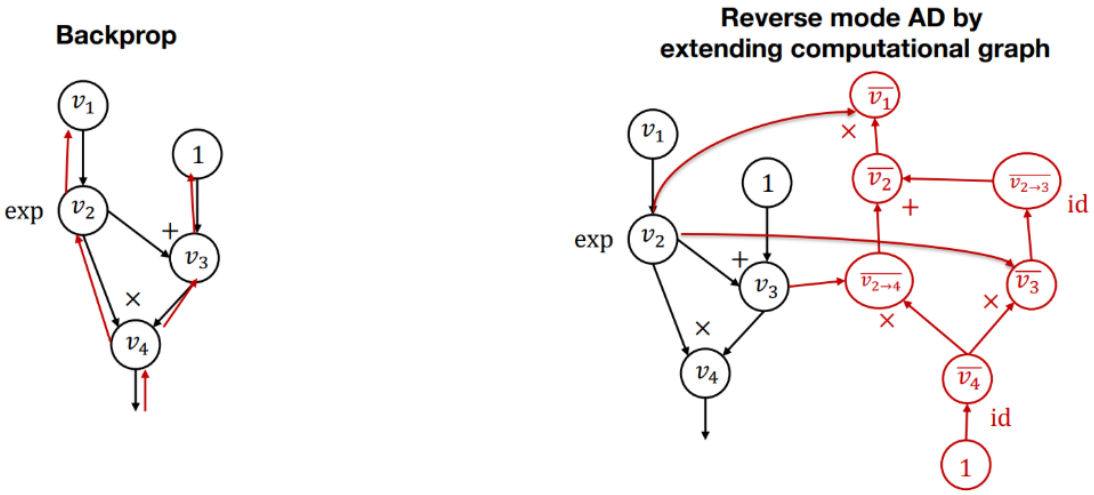
上述方法要求我们必须首先进行一次前向传播，得到全部的节点值 v_i. 这是和正向传播不一样的地方之一。

同样的，导数公式和节点的类型有关。

上面的反向传播公式要求我们存储每个节点的伴随值 $\bar{v}_i = \frac{\partial y}{\partial v_i}$，在计算机中一般有两种实现方法：

- 伴随值直接存储进节点中，此时节点除了存储了节点值 v_i 外，还存储了其伴随值 $\bar{v}_i$
- 将伴随值作为新的计算节点，构建一个更大的图——拓展计算图

  例如对于计算 $y = e^x(e^x + 1)$.



以 $\bar{v}_3$ 为例，由于 $v_4 = v_2 \cdot v_3$ 是乘法运算节点

$$\bar{v}_3 = \frac{\partial y}{\partial v_3} = \frac{\partial y}{\partial v_4} \cdot \frac{\partial v_4}{\partial v_3} = \bar{v}_4 \cdot v_2$$

所以 $\bar{v}_3$ 可以使用乘法节点，连接 $\bar{v}_4$ 和 v_2 来计算。

第一代的框架（caffe, cuda-convnet）等都用的是反向传播，而现代的框架（PyTorch, TensorFlow）等都是用反向模式进行自动求导。不过这两种方法都需要额外的存储空间来存储梯度值。

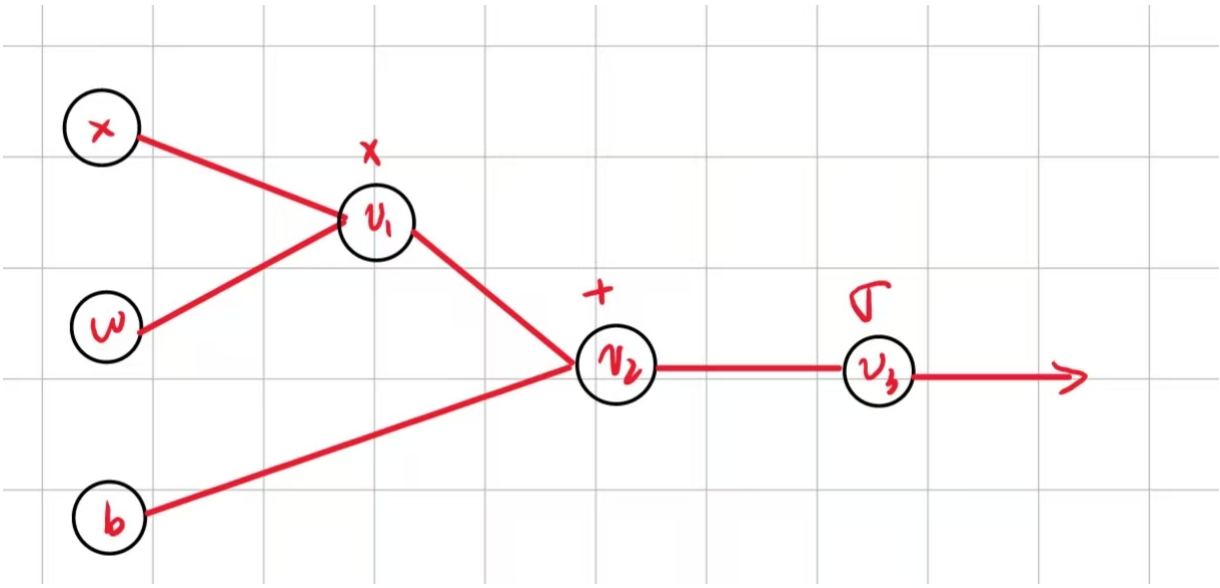
反向传播进行训练

上面的例子中，反向传播是对输入变量求的梯度，而神经网络参数更新需要计算输出对参数 w, b 的梯度。

$$w^{k+1} = w^k - \eta \cdot \nabla_w L(x; w)$$

这很简单，只需要将参数 w, b 也视为神经网络的输入即可。

$$y = \sigma(w \cdot x + b)$$



注意到，由于我们通常只需要更新参数 w, b，反向传播过程往往不需要计算 $\frac{\partial y}{\partial x}$，所以Pytorch等现代神经网络工具包中的输入张量 x 的 `requires_grad` 属性默认值是 `False`.

上机代码

由于拓展计算图的代码相对复杂，我们只实现第一代框架，即将梯度直接存储进节点。

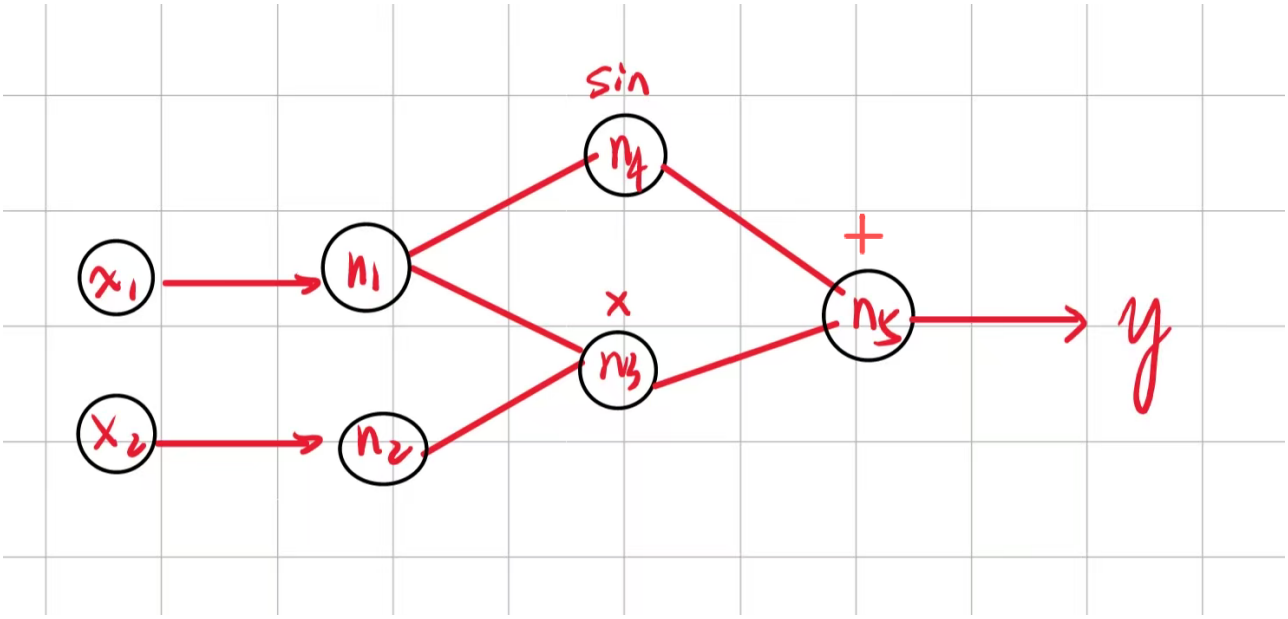
思考一下，代码如何实现：

- 需要构建计算图节点：
    - 节点要存储值、对应的伴随值（输出对其的梯度）
    - 要知道节点类型，以便于链式法则
- 需要实现前向传播
    - 从值到值，梯度不用计算
    - 需要知道节点类型
- 需要实现反向传播
    - 从梯度到梯度
    - 需要知道节点类型

问题

计算 $f(x_1, x_2) = x_1 x_2 + \sin(x_1)$ 函数在 $(2, 3)$ 处的梯度。

画出计算图



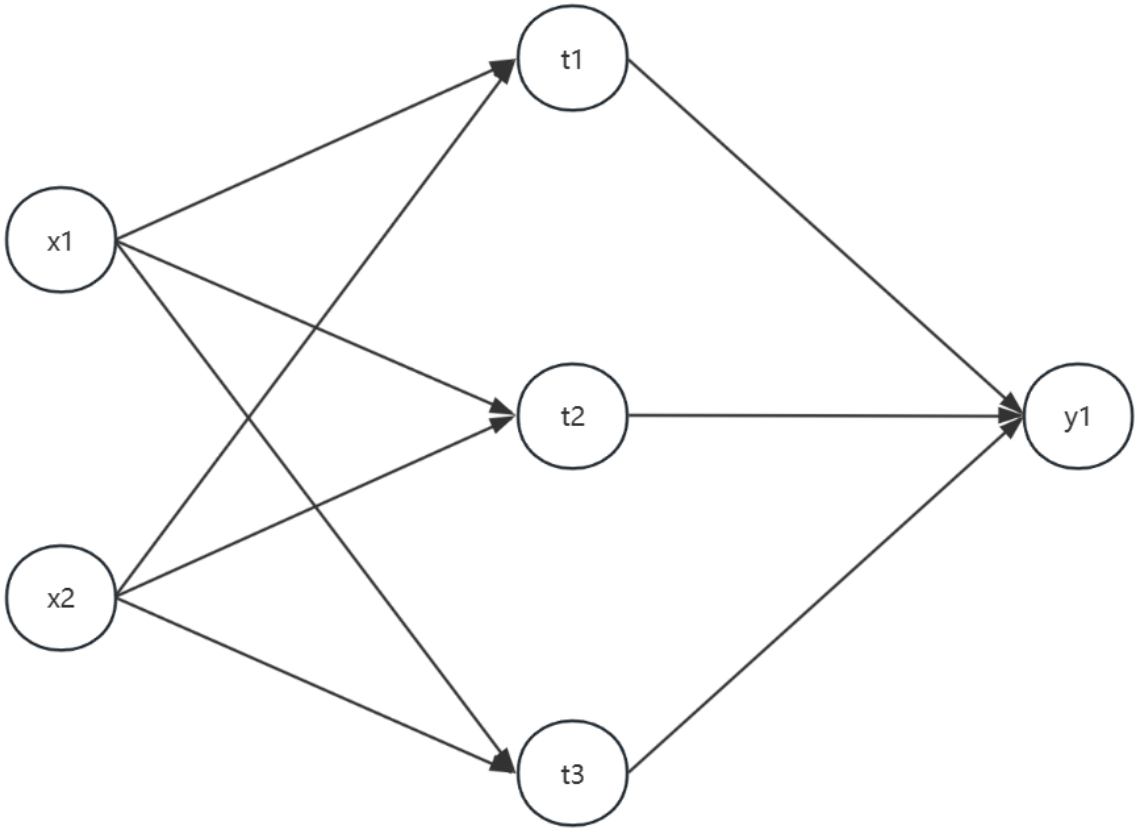

```
% 自动微分课堂演示：反向模式
% 计算 f(x1, x2) = x1*x2 + sin(x1) 在 (2,3) 处的梯度

clear; clc;

% 定义节点（结构体）
x1.value = 2; x1.grad = 0;
x2.value = 3; x2.grad = 0;

% 前向传播
n3.value = x1.value * x2.value; n3.grad = 0;
n4.value = sin(x1.value); n4.grad = 0;
n5.value = n3.value + n4.value; n5.grad = 0;

% 反向传播
n5.grad = 1;    % 输出节点梯度为1

% 反向传播到n3和n4（加法节点）
% 梯度累加而非直接赋值，因为可能出现一个节点作为多个节点的输入情况
n3.grad = n3.grad + 1 * n5.grad;    % 加法节点对两个输入的偏导都是1
n4.grad = n4.grad + 1 * n5.grad;

% 反向传播到x1和x2从n3（乘法节点）
x1.grad = x1.grad + n3.grad * x2.value;    % n3对x1的偏导是x2
x2.grad = x2.grad + n3.grad * x1.value;
```

```
% 反向传播到x1从n4（正弦节点）
x1.grad = x1.grad + n4.grad * cos(x1.value);   % n4对x1的偏导是
cos(x1)

% 显示结果
fprintf('函数值 f(2,3) = %f\n', n5.value);
fprintf('梯度: [df/dx1, df/dx2] = [%f, %f]\n', x1.grad, x2.grad);

% 与符号微分对比，验证结果
syms x1_sym x2_sym
f = x1_sym * x2_sym + sin(x1_sym);
grad = gradient(f, [x1_sym, x2_sym]);
grad_val = double(subs(grad, [x1_sym, x2_sym], [2,3]));
fprintf('符号微分梯度: [df/dx1, df/dx2] = [%f, %f]\n', grad_val(1),
grad_val(2));
```

由于本课程没有讲解链表等方便构建计算图的工具，因此，手动画一个计算图来方便记住某个节点的输入输出以及类型是非常好的习惯。

上机作业

如下是一个单隐藏层，隐藏层神经元数量为3，输入维数为2，输出维数为1，激活函数全为`ReLU`函数的全连接神经网络的示意图。



$$ReLU(x) = \max(x, 0)$$

说明：

- 规定 `ReLU` 节点的导数在0处的值为0. 代码中应该注意这一点（使用 $\geq or >$）.
- 上面的图是网络示意图，而非计算图，因为显然缺失了很多个 w, b. 实际上缺少了9个w和4个b.
- 图中已经给出关键节点的命名，你的代码和计算图中应该保持一致。
- 最后一层没有激活函数

作业：

1. 写出上面网络的公式：

   $$y_1 = \sum_{j=1}^{3} (\cdots)$$

   请适当使用下标，例如w_{ij}，使得节点命名相对易读。

2. 画出上述网络的计算图

   计算图中应该包含上一问中命名的 w, b 节点需要指明类型（加法节点、乘法节点、激活函数节点）

你需要在计算图中给出节点的命名，例如n_1，并在后续代码中使用它们。

3. 将你的网络权重w初始化为1，偏置b初始化为0，并计算前向传播在$(2,-2)$ 处的值。

   答案是一个固定的数字，但是你需要使用代码计算它。

4. 保持上述权重和偏置的初始化不变，计算输出结果 y_1 关于输入x_1,x_2 在$(2,-2)$的梯度

$$[\frac{\partial y_1}{\partial x_1}(2,-2),\frac{\partial y_1}{\partial x_2}(2,-2)]$$

   答案是固定的一个向量，但是你需要使用代码计算它。

5. 以如下函数为目标函数：

$$f(x_1,x_2)=2x_1+x_1x_2,-1\leq x_1,x_2\leq 1$$

   利用第五/第六次作业的代码，将网络结构、前向传播过程、梯度反向传播更新替换为计算图的形式，实现对目标函数的拟合任务。数据采样点数、学习率、训练轮数任选。

   当你训练好网络后，使用如下代码生成测试点，并将你的网络在测试点的预测结果保存为工作区变量。

```
% 定义x和y坐标向量
x = -1:0.1:1;     % x从-1到1，步长0.1
y = -1:0.1:1;     % y从-1到1，步长0.1

% 生成网格坐标
[X, Y] = meshgrid(x, y);
```

   你保存的文件应该为 21×21 的double类型矩阵，以`学号.mat`命名。

   如果使用Python，请同样导出为`.mat`格式。

提交：

1. pdf / docx报告：

   逐问的解答，计算图可以手绘

   第5问需要包含必要的说明（采样数量、学习率、训练轮数等）

2. 可运行的代码

   Matlab和Python均可，必须包含生成测试点结果的代码。

3. 以`学号.mat`命名的数据，将用于评测机评分。

4. 提交截至：周二晚23:59

参考：

https://zhuanlan.zhihu.com/p/571591103

https://pytorch.ac.cn/blog/overview-of-pytorch-autograd-engine/