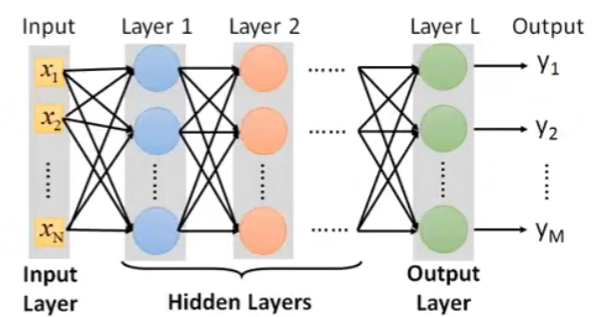
机器学习是通过找到一个复杂函数过程完成逆熵过程。

机器学习的三个任务regression、classification、structured learning。

# 深度学习

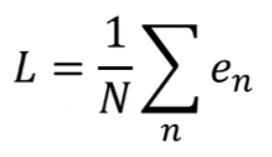
将以前找pixel feature的过程转变为设计network structure。



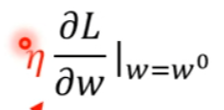
深度学习一般是多层叠加，中间神经元需要自己设计。上图是Fully connect feedforward network结构。

## 简单线性模型

建立线性函数F的训练，第一步模糊函数f=b+wx，b是偏差w是权重x是特征；第二步L（b，w）带入f，与label得出（平方差或绝对差）

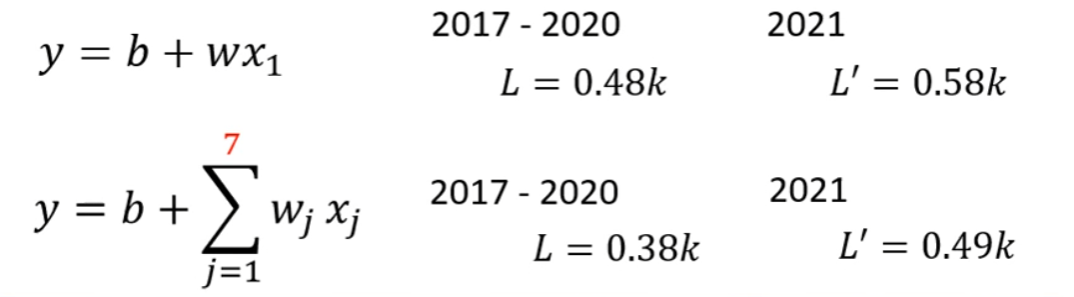


L越小，数据越好；第三步求解最佳化，求使L最小的w和b，常用梯度下降，即控制变量，随机选取。

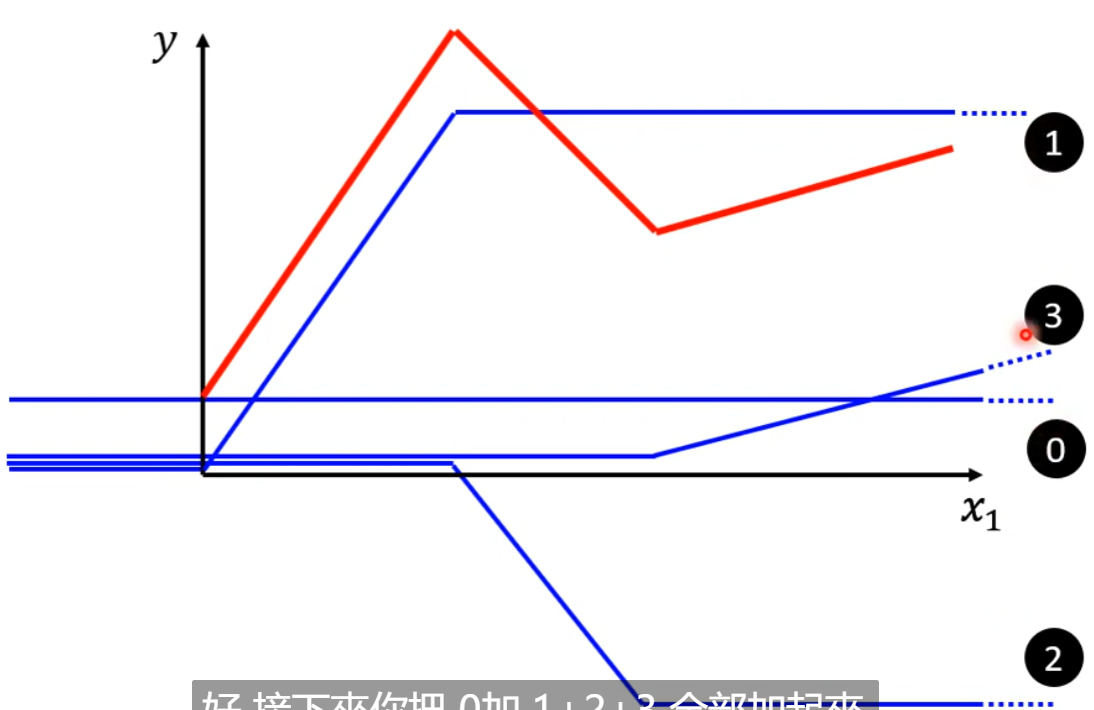


缩小w，最后得到b和w的最优解。

之后就是F的优化，在使用模型过程中，对函数过程了解之后，更改F函数

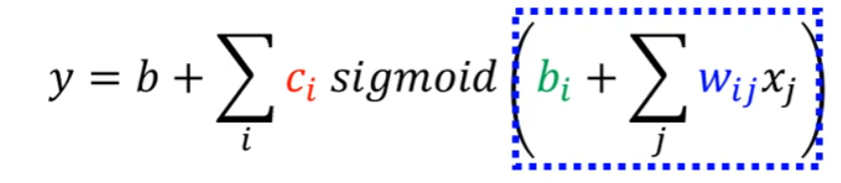


## 复杂线性模型



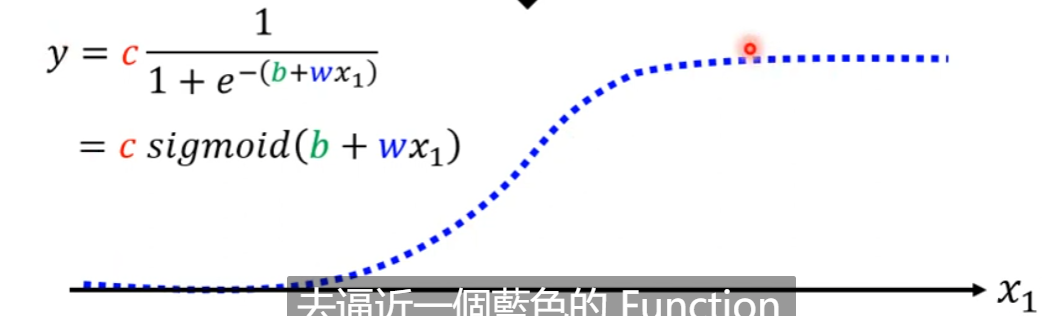
通过多条蓝色曲线的叠加来拟合红色曲线

公式表达为

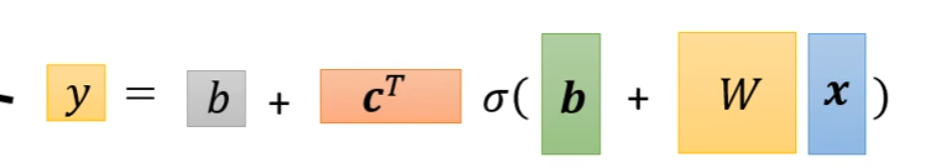


其中i为曲线序号，j为每次最佳运算的序号。

其中sigmoid函数具体为

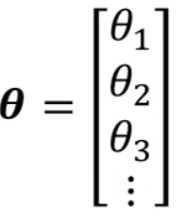


通过S函数来代替蓝色折线，最终将所求红色线性方程具体化。



最终可简化为向量矩阵形式。

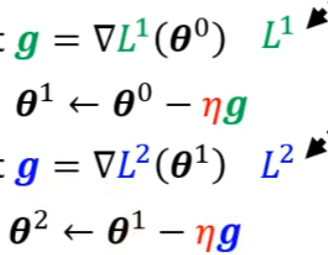
而在Loss函数中，用向量代替w和b。





G是gradient的简写。

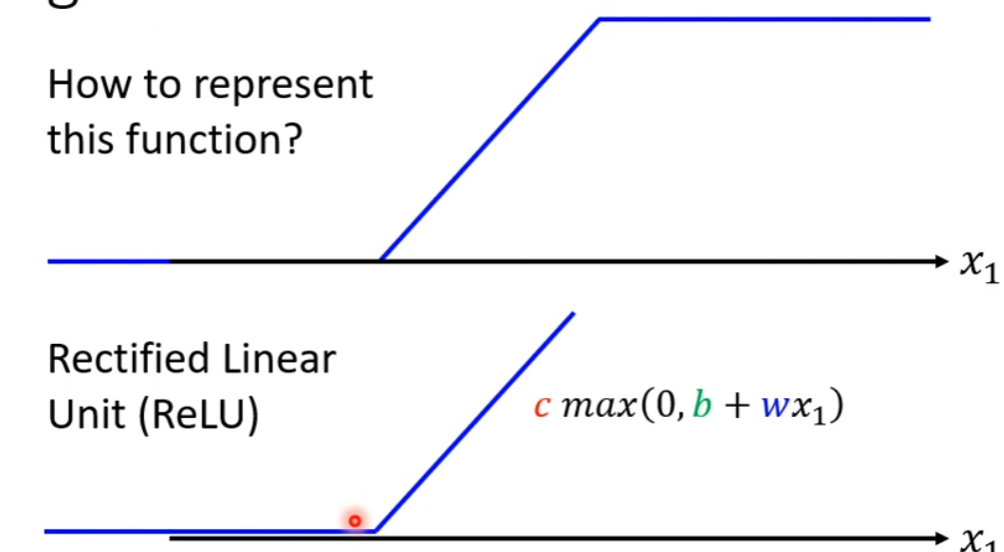
常常把大量的数据区分为很多个batch，每组batch更新一次参数。



Epoch是指对所有batch遍历完一次，可以通过总样本数batch容量，来算出一次epoch更新了多少次updates。

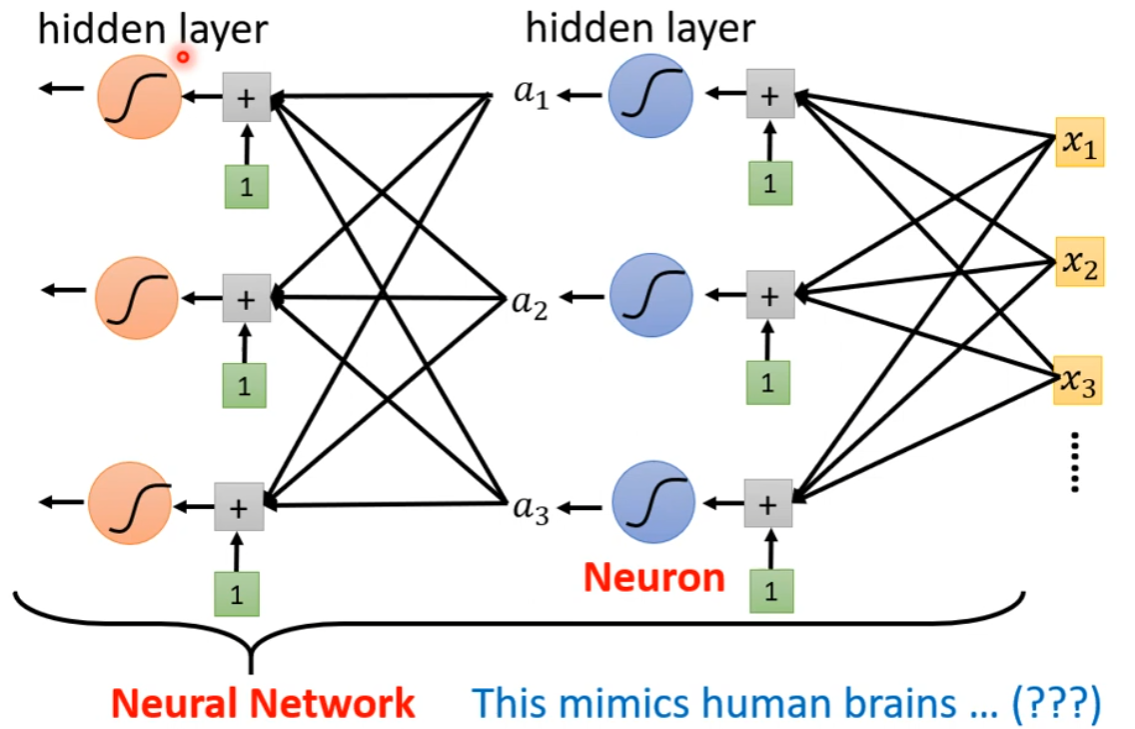
## ReLu函数

可以用多个Relu函数来代替sigmoid函数，如下图中，1个sigmoid函数可以用2个relu函数代替.



Relu函数的效果在视频实例中优于Sigmoid表现。

## Hidden layer/neuron



过去每一层函数表示被叫做神经元，层级嵌套组成神经网路。现在被叫做hidden layer，组合成深度学习。

Overfitting形容在训练资料上误差变小但是在实际预测中误差没有变小的情况。

# Pytorch

可以用GPU参与高维矩阵运算，简易计算梯度。

## Numpy

Numpy包含了很多数组操作。

import numpy as np

创建数组

a = np.array([1,2,3,4])

b = np.array([(1.5,2,3), (4,5,6)]) //数组要用【】括起来

a = np.arange(15).reshape(3, 5) //创建一个3x5的0-14的数组

array, zeros, zeros\_like, ones, ones\_like, empty, empty\_like, arange, linspace, numpy.random.Generator.rand, numpy.random.Generator.randn, fromfunction, fromfile。

基本运算

a-b //每个元素相减

a\*\*2 //a的平方

a\*b //向量乘

[a@b/a.dot(b)](mailto:a@b/a.dot(b)) //矩阵乘

a.sum(axis=0) //axis0是垂直1是水平，求和

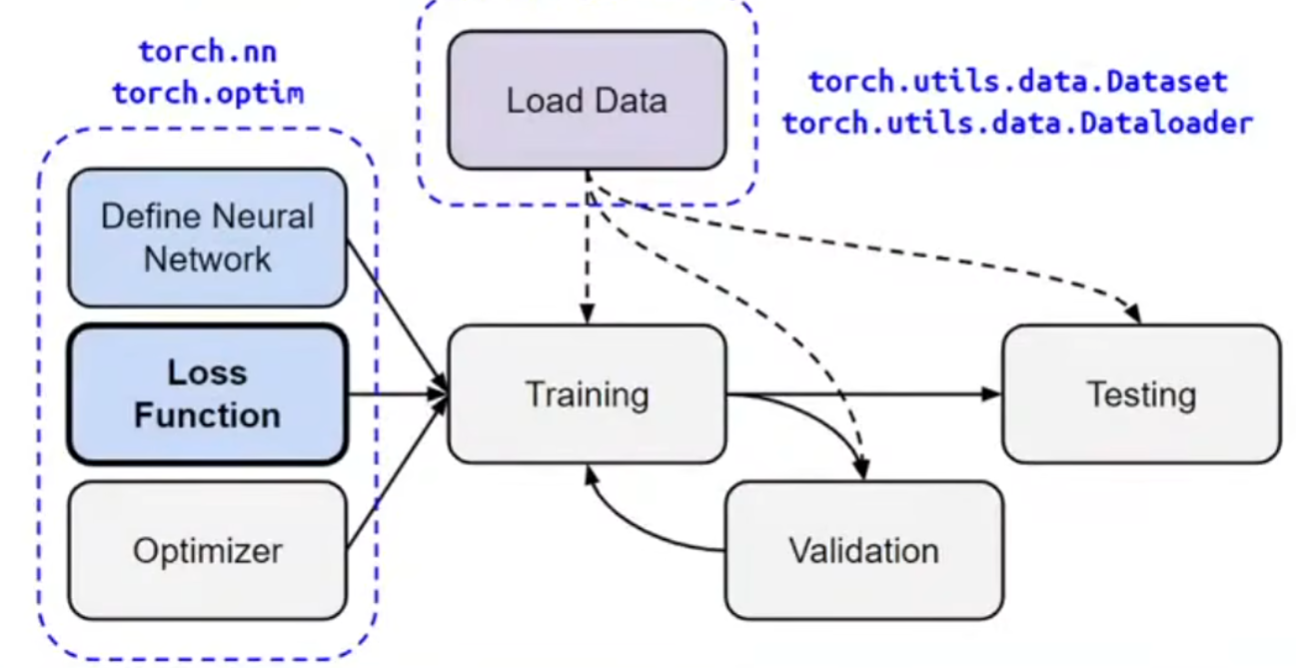
a,min(axis=0) //垂直方向最小值

np.add(a,b) //两个数组加

np.exp(a) //e的次方

np.sqrt(a) //开根

## Pytorch



用 dataset载入大量数据，dataloader按batch分类，选择training1或者testing0模式，初始化数据。Torch.nn建立模型,模型初始化,forward函数定义,Loss计算;torch.optim完成最优化更新。

###### tensor

Tensor是pytorch中类似于numpy中ndarray的结构。可以用from\_numpy(a)函数将numpy数组转换为tensor结构，也可以用tensor(a)强制转换。

Zeros(),ones(),rand()函数也可用于生成tensor结构。

Tensor的属性包括size、dtype、device(a.to(‘cuda’))等。(torch.cuda.is\_available()检测是否可用GPU加速)

Tensor的运算与numpy类似，包括：

索引

切片

Join

加法 +.add()

乘法 \*.@

Resize reshape().squeeze(dim).unsqueeze(dim).

Torch.cat([x,y,z]，dim=1)从1维度合并xyz矩阵。2x1，2x3，2x4合并成2x8

###### Autograd

自动求导包，应用于深度学习中的求偏导过程

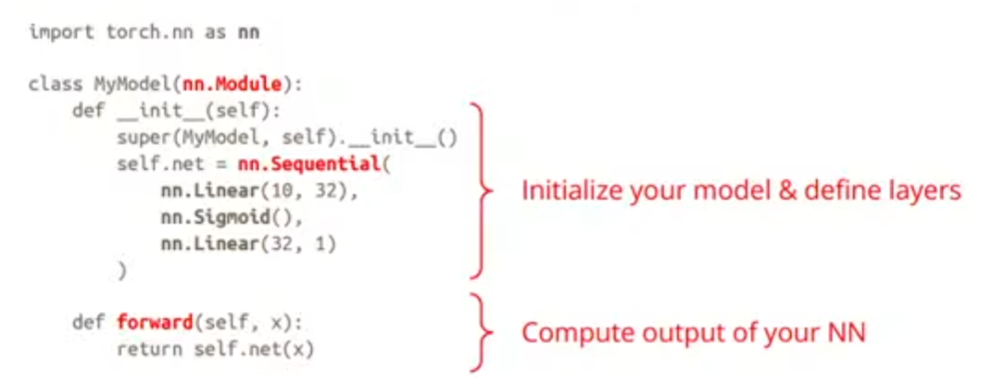
A=torch.tensor([…],requires\_grad=true) A可以计算梯度，当置false时可以排除某些不必计算节点

当定义了forward()函数之后，backward()函数会自动定义（通过chain rule计算）。

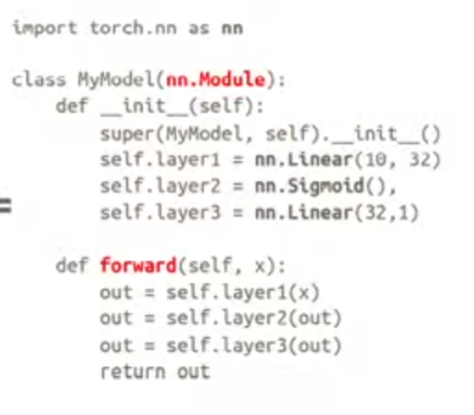
a.grad()可计算梯度

###### torch.nn

深度学习的模型搭建可以通过torch.nn包完成。Linear、sigmoid、relu函数全部可在nn包中调用。



定义一个封装三层神经网络，forward函数返回三层网络计算之后的x



两者等价

Loss 函数 nn.MSELoss() 用于regression

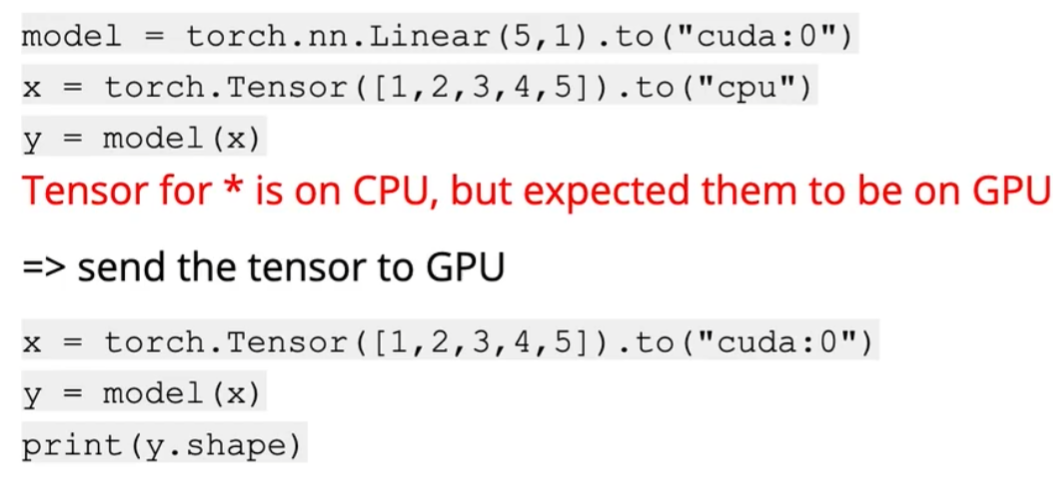
nn.CrossEntropyLoss() 用于classification

###### torch.optim

optim.SGD(net.parameters(),lr=xx) //模型参数集和学习效率设定

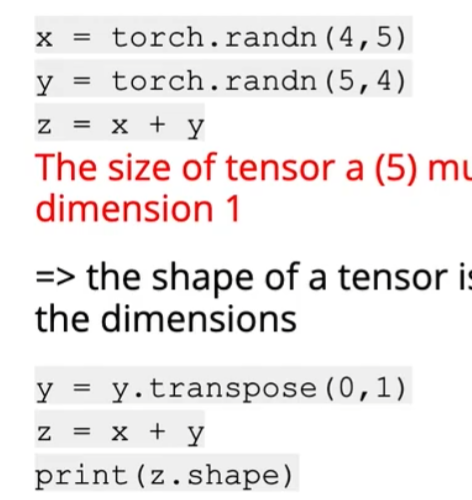
# 常见错误

###### 设备不同



模型与数据应该放在同一个设备上

###### 维度大小不同



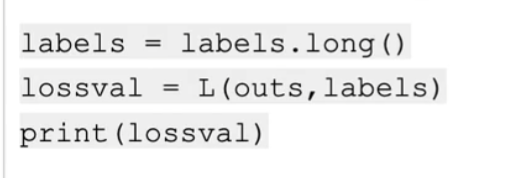
同大小的数组才可做运算。

###### 内存溢出

显存不够存储数据

采用迭代方式减小显存要求

###### 函数要求类型不匹配



# 总结

第一步，数据集建立

From torch.utils.data import Dataset,DataLoader

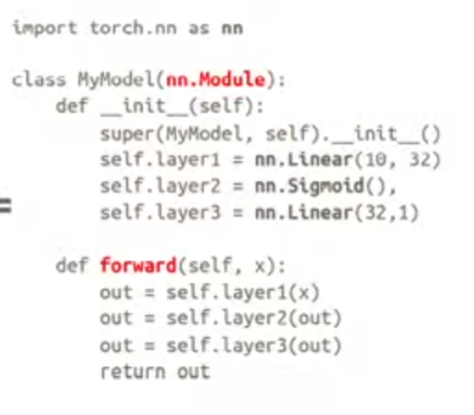


第二步，定义深度学习结构

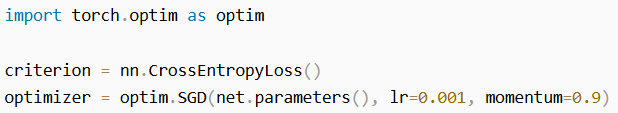
例

import torch.nn as nn

import torch.nn.functional as F



第三步，定义Loss函数和最优化



第四步，训练模型



第五步，可以从loss和正确率方面测试模型

