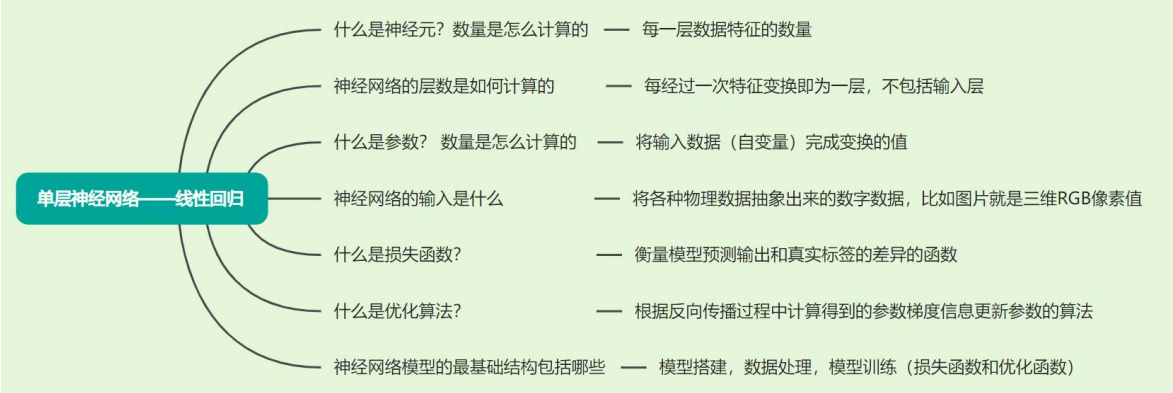
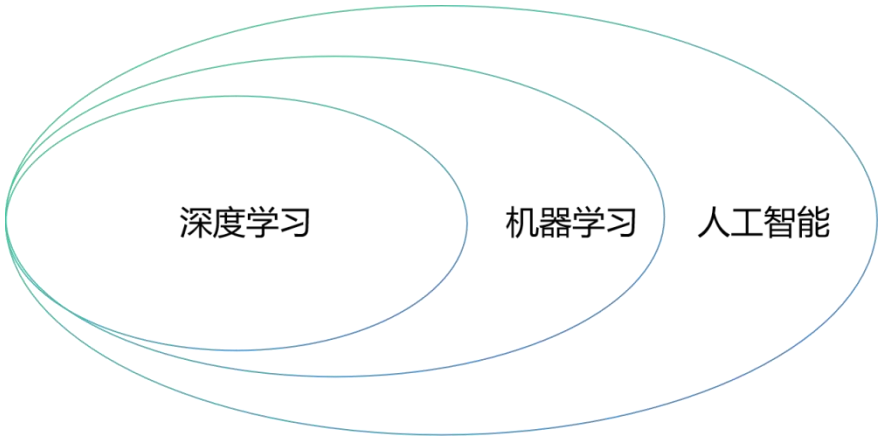
问题



深度学习属于机器学习的一个子集，可以分为无监督学习、强化学习和有监督学习。在无监督学习中，由于缺乏足够的先验知识，有些数据难以进行人工标注类别或是标注成本过高，所以无监督算法通常是基于无标签数据 x 隐式的或是显式的学习数据中的概率分布。在有监督学习中，输入的数据必须是带有真实标签的样本集合，样本 x 和标签 y 必须是一一对应关系。监督算法通过学习 x 和 y 的对应关系，从中发现有效的特征分布，从而具备基于输入 x 预测输出近似 y 的结果。在强化学习中，样本数据更加复杂，比如在许多状态决策或是智能控制的问题上，所研究的数据通常都是结构化无规则的。强化学习当前的研究也十分火热，应用领域包括工业控制、机器人控制和网络路由。



人工神经网络（英语：Artificial Neural Network，ANN），简称**神经网络**（Neural Network，NN）或**类神经网络**，本质就是函数的一种估计和近似，特征就相当于函数里的自变量，参数就是函数里的系数，组合变换就是函数，通过设计神经网络，对于海量繁杂的数据，从中挖掘提取出重要的特征（未知），然后将这些特征通过参数进行各种组合变换，获取最后想要的输出的过程。

只是这个自变量、参数以及函数本身是未知的，这也是人工智能被称为黑箱子的原因。我们能做的就是根据最终的结果反馈不断地优化前面的这些参数，这也就是网络的训练过程，使得这些网络能够输出正确的结果。

假如小明想去买香蕉吃，在店里询问香蕉单价为 2 元每斤，为了促进环保，使用塑料袋的话需要额外加 0.1 元费用，小明买了 3 斤，那么总价 y 便可以根据公式：

总价=单价*数量+塑料袋费用

计算得出，

y=2*3+0.1=7 元；

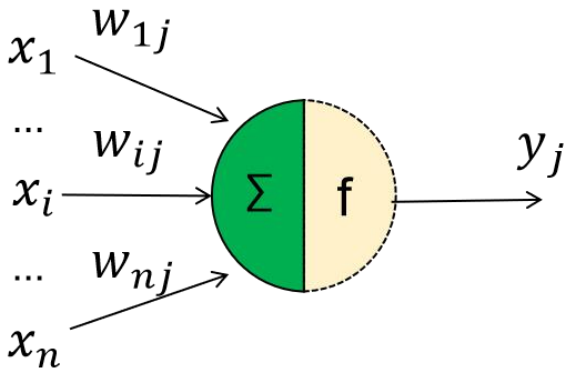
无论买多少斤（设为 x），都可以套用这个公式得出总价：

y=2x+0.1；

也就是说，在我们知道运算公式的情况下，给定输入，便可以得到输出。

神经元模型

对于一个公式，我们可以使用确切的数学表达式来表示输入和输出之间的关系，但是在机器学习里，没有一个确切的表达式，因此研究者们就模仿生物神经元的结构，创立了人工神经元模型。具体来说，神经元模型接收各种输入（特征），每个输入都有一个权重参数（w，这也是神经网络的参数），对所有输入加权求和之后，得到一个输出（特征），这就是最基础的神经元模型。不过很多情况下，加权求和之后还会再经过一个激活函数，使得网络具有非线性。1943 年，McCulloch 和 Pitts 将生物神经系统归纳为下图所示的“M-P 神经元模型”（有改动）：



其中： x_1，x_i,…x_n为各个输入的分量；w_{1j}，w_{ij},…w_{nj}为各个输入分量对应的权重参数；b 为偏置，可以省略；f 为激活函数，用以将前面线性求和得到的结果进行非线性运算；y_j为此神经元的输出；

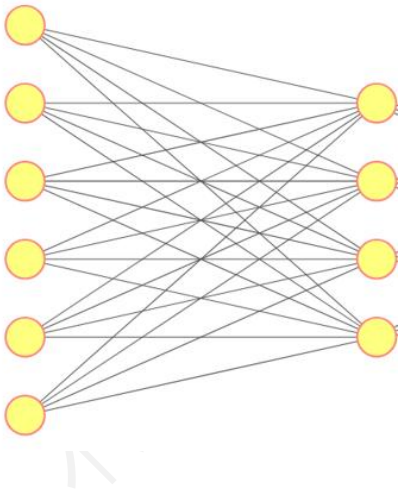
使用数学公式表示就是：

$$y = f(W^T X + b)$$

把许多这样的神经元按照一定的层次结构连接起来，就得到了神经网络。

单层神经网络——线性回归模型

单层神经网络是最基本的神经元网络形式，由有限个神经元构成。如图所示，计算神经网络层数的时候只会统计中间的隐藏层和最后的输出层，第一列的网络是最初的输入层，不会包含在内，因此单层神经网络只包含输出层。输出层中每一个神经元都会产生一个标量结果，所以需要得到多少维度的特征或输出，该层就需要多少个神经元。



前面一直在讲，神经网络是对函数的一种拟合或近似估计，接下来我们拿最简单的线性回归模型（最常见的波士顿房价预测）做一下示例，看看神经网络是如何从数据中拟合出函数的。

模型创建

模型创建前先思考第一个问题：输入是什么？或者说影响结果的自变量是什么？在总价=单价*数量这个公式里，影响总价的自变量是水果的单价，但在房价预测时，没有明确的计算公式，我们不知道具体是哪个自变量在影响房价，因此只能凭借着先验知识初步筛选自变量有哪些，我们将这些自变量称为特征。

房子的价格取决于很多特征因素，如房子的面积，房龄，地段，市场行情，城市发展状况等等（这些就是能够输出房子价格的特征），这些特征就是我们从数据集中挖掘提取的。一般情况下，神经网络的输入都是多维的。数据集里面有13 个特征，假设价格只取决于面积（平方米），房龄（年）和到市中心的距离（km），并认为三者和价格线性相关（这里进行简化仅说明网络结构）。

设房子的面积为x_1，房龄为x_2 ，到市中心距离为x_3,售出价格为 y。

且假设线性关系:$y = w_1x_1+w_2x_2 + w_3x_3 + b$

那么第二个问题来了？这个函数模型里的参数是多少？还是前面那个例子，在总价=单价*数量这个公式里，我们知道影响自变量的参数就是数量，但是在房价预测里，每个特征对结果的影响权重是未知的，这些参数w_1，w_2和w_3就是神经网络的参数，需要在网络训练中学习得到（怎么训练学习得到下面介绍）。然后便可以利用这些参数根据新房子的特征推测房价。

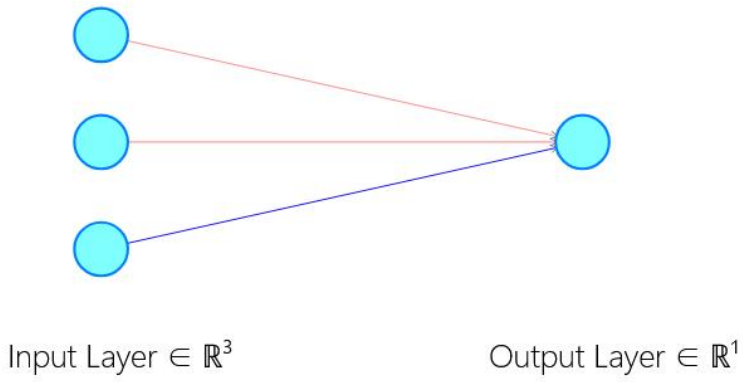


图 单层神经网络

如图所示，我们创建了线性回归的神经网络模型，这个模型包括输入层和输出层，输入层的输入个数为 3。输入个数也叫特征数或特征向量维度。输出层的输出个数为 1，由于输入层并不涉及计算，按照惯例不算到神经网络的层数里，图所示的神经网络的层数为 1。所以，线性回归是一个单层神经网络。

另外，输出层中的神经元和输入层中每个输入都连接。因此，这里的输出层又叫全连接层（fully-connected layer）或稠密层（dense layer）。后面在讲到卷积神经网络的时候会再进一步介绍。

模型训练

模型训练的本质就是利用已知的数据获得想要的参数。具体过程就是，利用数据集中已有的输入和当前的参数，计算得到一个预测结果，然后和数据集中的真实结果（标签）作比较，利用误差去优化模型参数，不断训练，直到达到设置的训练次数或者预测结果和真实结果的误差在一定范围。

数据收集

数据是人工智能的基础和核心，本节暂且简单说明一下，卷积神经网络章节再详细介绍。

通过收集大量的真实数据，每组数据包括房子的真实售价和它们对应的面积，房龄和到市中心距离，我们希望在这个数据上面利用上面创建的网络模型进行训练，学习得到模型参数来使模型的预测价格与真实价格的误差最小。

在机器学习术语里，该数据集被称为训练数据集（training data set）或训练

集（training set），数据里房子的真实售价叫作标签（label），用来预测标签的三个输入叫作特征（feature），特征用来表征样本的特点。一组数据被称为一个样本（sample），包含输入和标签。

假设我们采集的样本数为 n，索引为 i 的样本的特征为$\mathbf{x^i}$，标签为$\hat{\mathbf{y}}^{\mathbf{i}}$。对于索引为 i 的房屋，线性回归模型的房屋价格预测表达式为

$$\hat{y}^i = x_1^i w_1 + x_2^i w_2 + x_3^i w_3 + b$$

也就是说输入数据的形状是 n*3*1,输出形状是 1，中间无隐藏层，权重参数维度为 1*3。

参数初始化

前面模型创建章节建立好模型后，却并不知道参数是多少，那如何得到预测售价呢？答案就是先手动设置参数的初始值，方法有很多，比如全设置为 0，或者随机初始化。

神经网络参数初始化是深度学习中的关键步骤之一。参数初始化的选择和设置对于网络的训练和性能具有重要影响。在后面我们将进一步讨论。这里我们先选择全设置为 0。

损失函数

有了预测模型，知道了影响房价的特征输入和参数，便可以像单价*数量=总价那样计算得到房价了。但问题在于影响房价的特征是凭经验选的，参数更是随机初始化的，并不是数学意义上严格的计算公式。

此时思考一下小孩子在学习辨认鹿和马的过程是怎样的？映入眼帘一只鹿，观察到它的鹿角，皮毛，颜色，四肢，大人告诉这是一只鹿，小孩记住了，再次看到的时候，如果小孩子说是马，大人多次进行纠正，最后针对这些特征便可以辨认出这是鹿，只是我们并不知道大脑进行了怎样的完善。

同样的，对于神经网络的训练，首先根据特征输入和初始的参数，计算出预测结果，然后与真实结果进行比较，得到它们之间的差值。

研究人员提出来损失函数这个概念，损失函数又可称为代价函数或目标函数，是用来衡量算法模型预测结果和真实标签之间吻合程度（误差）的函数。

通常会选择非负数作为预测值和真实值之间的误差，误差越小，则模型越好。本例中选用简单的平方函数，即：

$$L^i(w_1, w_2, b) = \frac{1}{2}(\hat{y}^i - y^i)^2$$

训练模型的目标便是将此损失函数的值尽可能的小。后面再介绍其它的损失函数。通常，我们用训练数据集中所有样本误差的平均来衡量模型预测的质量，在模型训练中，我们希望找出一组模型参数，来使训练样本平均损失最小。

优化算法

有了模型和初始参数，我们便得到了预测目标值；有了预测目标值和真实标签值，代入到损失函数便可以得到误差；那么如何根据这个误差去优化模型，就像在辨认鹿的过程中给予的纠正一样呢？神经网络训练使用的就是优化算法。

损失函数即预测值和真实值的误差是关于参数的函数，优化算法就是用来寻找模型参数，使得训练样本的平均损失最小。

当模型和损失函数形式较为简单时，上面的误差最小化问题的解可以直接用公式表达出来。这类解叫作解析解（analytical solution）。本节使用的线性回归和平方误差刚好属于这个范畴。

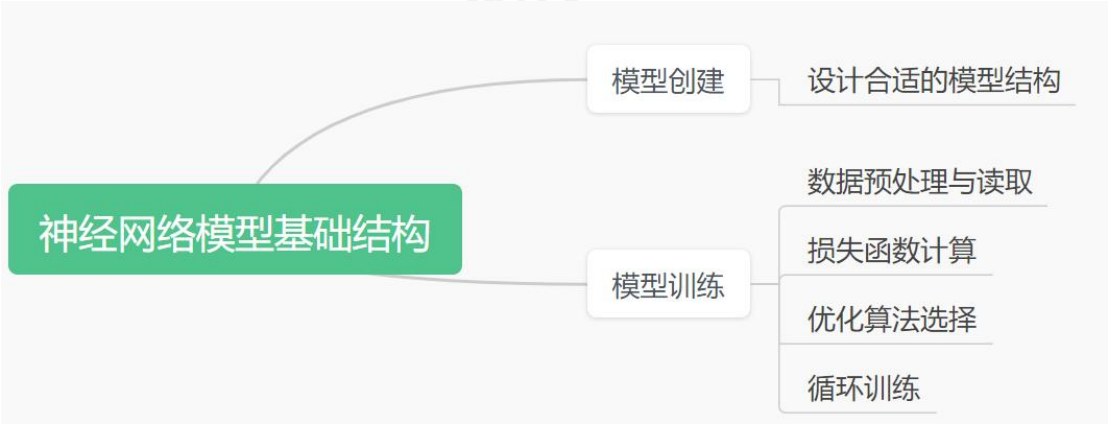
然而，大多数深度学习模型并没有解析解，只能通过优化算法有限次迭代模型参数来尽可能降低损失函数的值。这类解叫作数值解（numerical solution）。

循环训练

设置循环次数，每一个遍历循环过程，都是根据输入的特征计算预测结果，然后求得损失函数值，再利用优化算法更新模型参数。

在达到设定的训练次数或者误差值在一定范围之后训练停止，这时得到一个拟合好的函数模型。面对新的未知的数据，只需要将特征输入到神经网络模型中，便可以像普通函数公式那样计算得到较为准确的输出结果了。

小结



代码

```
import torch
import numpy
import random
import matplotlib.pyplot as plt
#import models.linear-regression
#随机生成数据集
```

```
def make_data(w,b,num_examples):
    #生成均值为 0 标准差为 1 的正态分布数据，数据的维度是
num_examples*len(w),
    x=torch.normal(0,1,(num_examples,len(w)))
    #生成真实的 y
    y=torch.matmul(x,w)+b
    #加入噪声，模拟实际数据集中的标签值
    y+=torch.normal(0,0.001,y.shape)
    return x,y.reshape((-1,1))

#读取数据集，在 pytorch 里面有预先写好的读取数据的类，这里我们用函数简单
写一下过程
#读取数据的时候，最关注的就是数据的 batch_size 大小，以及输入特征维度和
标签值
def data_loader(batch_size,fetures,labels):
    #这里是获取一共有多少组数据
    num_examples=len(features)
    indices=list(range(num_examples))
    # 这些样本是随机读取的，没有特定的顺序
    random.shuffle(indices)
    #然后根据 batch_size 进行分组,
    for i in range(0,num_examples,batch_size):
        #储存每一 batch 所对应的数据样本的索引值
        batch_indices=torch.tensor(
            indices[i:min(i+batch_size,num_examples)]
        )
      #得到每一 batch 索引值对应的数据样本中的输入和标签
        yield features[batch_indices],labels[batch_indices]

# for X, y in data_loader(batch_size, features, labels):
#     print(X, '\n', y)
#     break

#创建模型
def linreg(x,w,b):
    return torch.matmul(x,w)+b


#初始化模型参数

w=torch.normal(0,0.001,size=(3,1),requires_grad=True)
b=torch.zeros(1,requires_grad=True)

#定义损失函数
```

```
def squared_loss(y_hat,y):
    #y_hat 表示预测值，y 表示真实值，计算二者的均方损失函数
    return (y_hat-y.reshape(y_hat.shape))**2/2

#定义优化算法,优化算法针对的是模型参数 params，优化速度需要设置超参数学习率，还要知道批大小 batch_size
def sgd(params,lr,batch_size):
    with torch.no_grad():
        #根据参数的导数和学习率来更新参数（后面反向传播再详细介绍）
        for param in params:
            param-=lr*param.grad/batch_size
            param.grad.zero_()

#开始训练

#读取数据
true_w = torch.tensor([2, 3.4,1.9])
true_b = 4.2
features, labels =make_data(true_w, true_b, 1000)
#设置超参数
lr = 0.03
num_epochs = 3
batch_size = 10

net = linreg
loss = squared_loss

for epoch in range(num_epochs):
    for X, y in data_loader(batch_size, features, labels):
        l = loss(net(X, w, b), y)  # X 和 y 的小批量损失
        # 因为 l 形状是(batch_size,1)，而不是一个标量。l 中的所有元素被加到一起，
        # 并以此计算关于[w,b]的梯度
        l.sum().backward()
        sgd([w, b], lr, batch_size)  # 使用参数的梯度更新参数
    with torch.no_grad():
        train_l = loss(net(features, w, b), labels)
        print(f'epoch {epoch + 1}, loss {float(train_l.mean()):f}')
```

代码总结

1 学习线性层的网络搭建，即 nn.Linear(input_features, output_features)，
尤为注意两点，一是 input_features, output_features 是线性层的网络参数，并非输

入输出数据；
二是网络参数只针对输入输出数据的最后一个维度的形状，前面保持一致，即对于输入数据[batch_size,input_features]，对应输出是[batch_size,output_features]

参考资料

https://zh.d2l.ai/index.html