# 机器学习-期中考试

## 单选、多选、判断、简答


## 深度学习的一般步骤?

- 数据处理：从本地或网络读取数据，并完成预处理操作，保证模型可读取
- 模型设计：网络结构设计
- 训练配置：设定模型采用的寻解算法，即优化器，并指定计算资源（配置超参数）
- 训练过程：循环调用训练过程，每轮都包括前向计算、损失函数（优化目标）和反向传播三个步骤
- 模型保存：将训练好的模型保存，模型预测时调用
- 模型部署应用

## 深度学习中的超参数：

- **learning rate**
- epochs(迭代次数，也可称为 num of iterations)
- num of hidden layers(隐层数目)

- num of hidden layer units(隐层的单元数/神经元数)
- **activation function**(激活函数)
- **batch-size**(用mini-batch SGD的时候每个批量的大小)
- **optimizer**(选择什么优化器，如SGD、RMSProp、Adam)
- 用诸如RMSProp、Adam优化器的时候涉及到的β1，β2等等

**部分超参数如何影响模型性能?**

| 超参数 | 如何影响模型容量 | 原因 | 注意事项 |
| --- | --- | --- | --- |
| 学习率 | 调至最优，提升有效容量 | 过高或者过低的学习率，都会由于优化失败而导致降低模型有效容限 | 学习率最优点，在训练的不同时间点都可能变化，所以需要一套有效的学习率衰减策略 |
| 损失函数部分超参数 | 调至最优，提升有效容量 | 损失函数超参数大部分情况都会可能影响优化，不合适的超参数会使即便是对目标优化非常合适的损失函数同样难以优化模型，降低模型有效容限。 | 对于部分损失函数超参数其变化会对结果十分敏感，而有些则并不会太影响。在调整时，建议参考论文的推荐值，并在该推荐值数量级上进行最大最小值调试该参数对结果的影响。 |
| 批样本数量 | 过大过小，容易降低有效容量 | 大部分情况下，选择适合自身硬件容量的批样本数量，并不会对模型容限造成。 | 在一些特殊的目标函数的设计中，如何选择样本是很可能影响到模型的有效容限的，例如度量学习（metric learning）中的N-pair loss。这类损失因为需要样本的多样性，可能会依赖于批样本数量。 |
| 丢弃法 | 比率降低会提升模型的容量 | 较少的丢弃参数意味着模型参数量的提升，参数间适应性提升，模型容量提升，但不一定能提升模型有效容限 | |
| 权重衰减系数 | 调至最优，提升有效容量 | 权重衰减可以有效的起到限制参数变化的幅度，起到一定的正则作用 | |
| 优化器动量 | 调至最优，可能提升有效容量 | 动量参数通常用来加快训练，同时更容易跳出极值点，避免陷入局部最优解。 | |
| 模型深度 | 同条件下，深度增加，模型容量提升 | 同条件，下增加深度意味着模型具有更多的参数，更强的拟合能力。 | 同条件下，深度越深意味着参数越多，需要的时间和硬件资源也越高。 |
| 卷积核尺寸 | 尺寸增加，模型容量提升 | 增加卷积核尺寸意味着参数量的增加，同条件下，模型参数也相应的增加。 | |



# 机器学习的三要素

机器学习的三要素：模型、策略、算法

# 机器学习中数据集的分类?

- 训练数据：用来训练模型，让模型能够拟合训练数据， 这个样本通常会大一些
- 验证数据：用来验证模型的预测准确率，通常不对验证 数据训练；不应与训练数据混在一起（ImageNet google）
- 测试数据：一般只用一次，要求测试数据完全没有出现 在训练数据中过，而且也绝对不能对测试数据进行训练。

# 机器学习中的Error

- 训练误差:训练过程中的误差
- 验证误差:在验证集上进行交叉验证选择参数（调参），最终模型在验证集上的误差
- 测试误差:训练完毕、调参完毕的模型，在新的测试集上的误差
- 泛化误差:假如所有数据来自一个整体，模型在这个整体上的误差。通常说来，测试误差的平均值或者说期望就是泛化误差

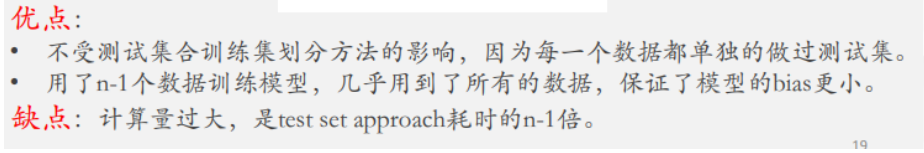
> 训练误差 < 验证误差 < 测试误差 ~= 泛化误差

# 模型训练过程中的一些概念（超参数）

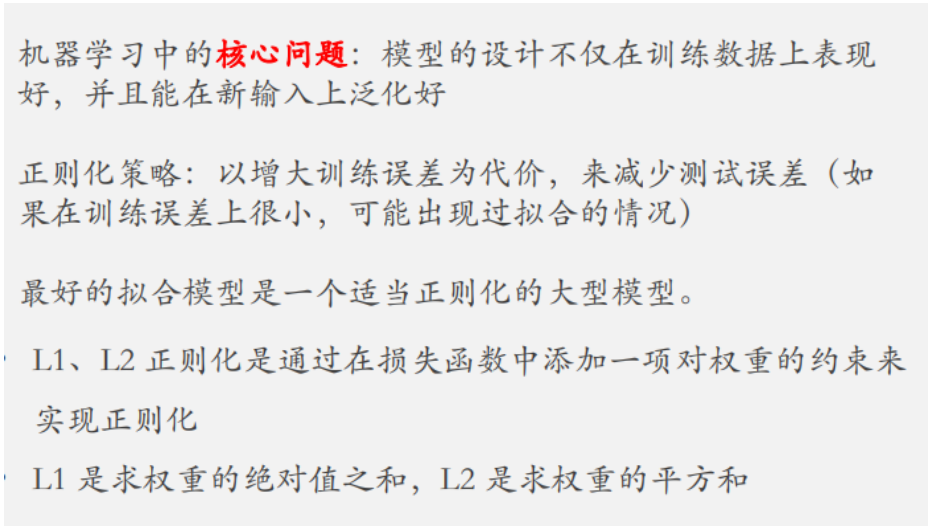
- Epoch：一个epoch等于使用训练集中的全部样本训练一次的 过程。当一个完整的数据集通过了神经网络一次并且返回了 一次，即进行了一次正向传播和反向传播，这个过程称为一 个epoch。
- Batch：整个训练集分成多个小块，也就是就是分成多个Batch
- Iteration：训练一个batch的过程为一个iteration

# 简述模型选择的方法

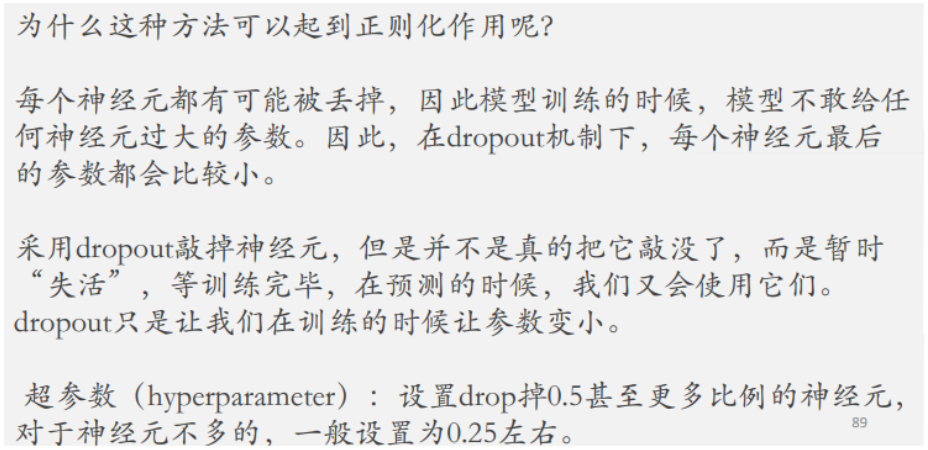
- 交叉验证：留出法、留一法交叉验证、K折交叉验证
    - 很多时候数据不够充足，这种时候可以取消验证集，采用交 叉验证方法，通过反复划分训练集和测试集来避免用同一批数 据训练和评估一个模型，相当于将验证集和测试集合二为一了。
    - 留一法：每次只使用一个作为测试集，剩下的全部作为训练集。 这种方法得出的结果与训练整个测试集的期望值最为接近，但是成本过于庞大。




- 正则化：L1、L2正则化




- dropout法：dropout是神经网络中常用的正则化方法，就是 在训练过程中，随机地“丢弃”一些神经元，强行 简化模型




- 早停法


# 欠拟合与过拟合问题

- 欠拟合的算法会表现出Bias非常高，Variance非常低
- 过拟合的算法会表现出Bias非常低，Variance非常高

- **Bias（偏差）**是指算法的预测与真实结果的偏离程度。度量了学习算法的期望输出与真实结果的偏离程度，刻画了算法的拟合能力，Bias 偏高表示预测函数与真实结果差异很大。

- **Variance（方差）**是指训练集的变动导致性能变化的程度，Variance 偏高表示模型很不稳定。**相当于用多考几次试，看每次得分是不是差不多。**

# 第一章节&第二章节

有监督学习：从给定的有标注的训练数据集中学习出一个函数（模型参数），当心的数据到来时可以根据这个函数预测结果。常见任务包括分类与回归。

无监督学习：没有标注的训练数据集，需要根据样本间的统计规律对样本集进行分析。有主成分分析、因果关系和概率图模型问题、生成对抗性网络等几类。

半监督学习：结合（少量的）标注训练数据和（大量的）未标注数据来进行数据的分类学习。其两个基本假设为聚类假设和流形假设。

强化学习：外部环境对输出只给出评价信息而非正确答案，学习机通过强化受奖励的动作来改善自身性能。

常用的机器（深度）学习框架：Tensorflow、Caffe、Theano、MXNet、Pytorch

## 小结

- **深度学习存储和操作数据的主要接口是张量（n维数组）。它提供了各种功能，包括基本数学运算、广播、索引、切片、内存节省和转换其他Python对象**
- **pandas软件包是Python中常用的数据分析工具中，pandas可以与张量兼容**
- **用pandas处理缺失的数据时，我们可根据情况选择用插值法和删除法**

## 小结

- 标量、向量、矩阵和张量是线性代数中的基本数学对象
- 向量泛化自标量，矩阵泛化自向量
- 标量、向量、矩阵和张量分别具有零、一、二和任意数量的轴
- 一个张量可以通过sum和mean沿指定的轴降低维度
- 两个矩阵的按元素乘法被称为他们的Hadamard积,它与矩阵乘法不同
- 在深度学习中，我们经常使用范数，如L1范数、L2范数和Frobenius范数
- 我们可以对标量、向量、矩阵和张量执行各种操作

103

## 小结

- 微分和积分是微积分的两个分支，前者可以应用于深度学习中的优化问题
- 导数为函数相对于其变量的瞬时变化率，是函数曲线的切线的斜率
- 链式法则可以用来微分复合函数
- 我们可以从概率分布中采样
- 我们可以使用联合分布、条件分布、Bayes定理、边缘化和独立性假设来分析多个随机变量
- 期望和方差为概率分布的关键特征的概括提供了实用的度量形式

# 第三章

# 线性回归

回归（regression）是能为一个或多个自变量与因变量之间关系建模的一类方法。在自然科学和社会科学领 域，回归经常用来表示输入和输出之间的关系。

损失函数（loss function） 能够量化目标的实际值与预测值之间的差距。通常我们会选择非负数作为损失，且数值越小表示损失越小， 完美预测时的损失为0。回归问题中最常用的损失函数是平方误差函数。

当样本i的预测值为yˆ (i)，其相应的真 实标签为y (i)时，平方误差可以定义为以下公式：

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2.$$

解析解 线性回归的解可 以用一个公式简单地表达出来，这类解叫作解析解（analytical solution）。

# 梯度下降

梯度就是定义一个函数在某一个点的全部偏导数构成的向量

凸函数判断依据：
- f((x1+x2)/2)<=(f(x1)+f(x2))/2
- 简单来说如果任意两点连接而成的线段与函数没有交点，则可以说此函数为凸函数

梯度下降常分为BGD（批量梯度下降）和SGD（随机梯度下降）

其中，又有MSGD（小批量随机梯度下降）

可能会考简答题：分析BGD、SGD、MSGD各自的优缺点

- BGD 批量梯度下降
  - 优点：（1）一次迭代是对所有样本进行计算，此时利用矩阵进行操作，实现了并行。
（2）由全数据集确定的方向能够更好地代表样本总体，从而更准确地朝向极值所在的方向。当目标函数为凸函数时，BGD一定能够得到全局最优。
  - 缺点：（1）当样本数目 m 很大时，每迭代一步都需要对所有样本计算，训练过程会很慢。
- SGD 随机梯度下降
  - 优点：（1）由于不是在全部训练数据上的损失函数，而是在每轮迭代中，随机优化某一条训练数据上的损失函数，这样每一轮参数的更新速度大大加快。
  - 缺点：（1）准确度下降。由于即使在目标函数为强凸函数的情况下，SGD仍旧无法做到线性收敛。
（2）可能会收敛到局部最优，由于单个样本并不能代表全体样本的趋势.
（3）不易于并行实现。
- MSGD 小批量随机梯度下降

- 优点：（1）通过矩阵运算，每次在一个batch上优化神经网络参数并不会比单个数据慢太多。
（2）每次使用一个batch可以大大减小收敛所需要的迭代次数，同时可以使收敛到的结果更加接近梯度下降的效果。
（3）可实现并行化。
- 缺点：（1）batch_size的不当选择可能会带来一些问题。

# 如果在线性回归的实现过程中，将权重初始化为零，会发生什么？算法仍然有效吗？

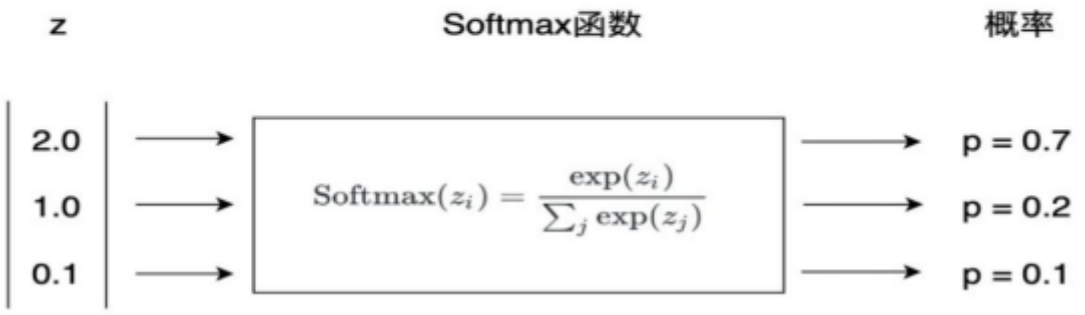
当将权重全部初始化为零时，算法仍然有效，但是会导致模型无法学习到数据中的任何模式，因为所有权重将始终保持为零。这意味着模型将始终输出固定的预测，损失函数将始终保持不变。因此，算法仍然有效，但是无法实现有意义的学习。

如果将损失函数定义为L1损失，即绝对误差，那么当权重全部初始化为零时，梯度将始终指向同一方向，即如果预测值小于真实值，则将权重向正方向调整；如果预测值大于真实值，则将权重向负方向调整。这样，权重将根据数据中的模式进行调整，但是这可能需要更长时间才能收敛到最优解。

# softmax回归

之前提到，回归问题的主要任务是预测及分类。

$$\mathrm{Softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$




Softmax函数或称归一化指数函数，是逻辑函数的一种推广，多个神经元输出映射到(0,1)区间内

Softmax函数一般作为神经网络的最后一层，接受来自上一层网络的输入值，然后将其转化为概率

**softmax 回归**(softmax regression)其实是 logistic 回归的一般形式，logistic 回归用于二分类，而 softmax 回归用于**多分类**

为了估计所有可能类别的条件概率，我们需要一个有多个输出的模型，每个类别对应一个输出。为了解决线 性模型的分类问题，我们需要和输出一样多的仿射函数（affine function）。每个输出对应于它自己的仿射函 数。

与线性回归一样，softmax回归也是一个单层神经网络。 由于计算每个输出o1、o2和o3取决于所有输入x1、x2、x3和x4，所以softmax回归的输出层也是全连接层。

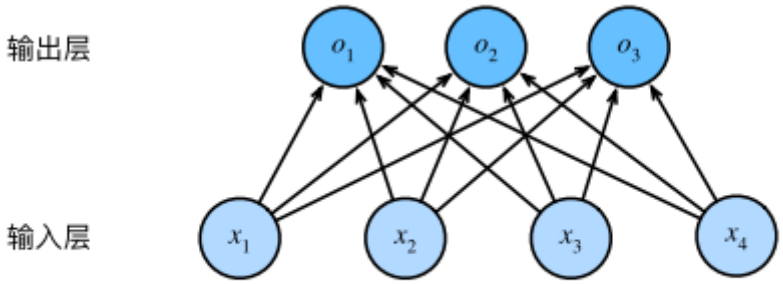


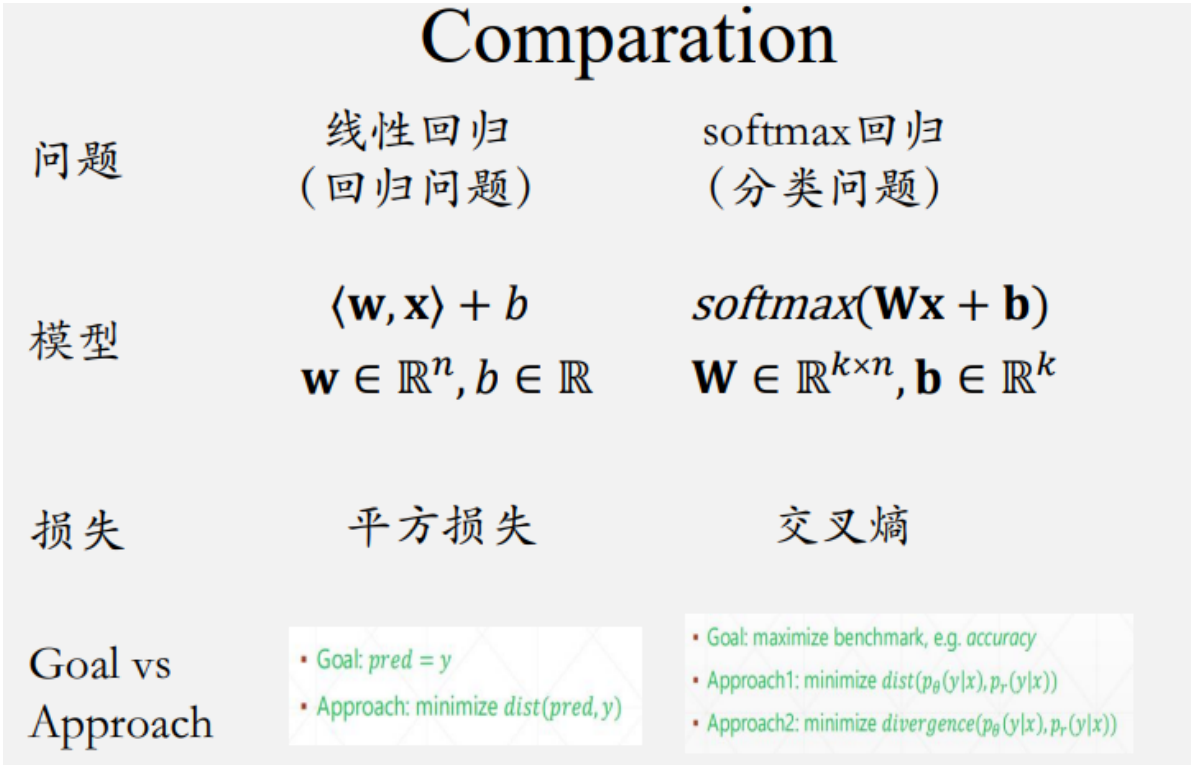

图3.4.1: softmax回归是一种单层神经网络

# softmax回归选择的损失函数

MSE或者交叉熵损失

交叉熵时梯度比较陡峭，利于做优化

MSE 导致的梯度有很大的平坦区域，优化过程很可能卡住。




# softmax回归是一个线性模型吗？为什么？

softmax运算不会改变未规范化 的预测o之间的大小次序，只会确定分配给每个类别的概率。因此，在预测过程中，我们仍然可以用下式来选 择最有可能的类别。

$$\operatorname*{argmax}_j \hat{y}_j = \operatorname*{argmax}_j o_j. \tag{3.4.4}$$

尽管softmax是一个非线性函数，但softmax回归的输出仍然由输入特征的仿射变换决定。因此，softmax回 归是一个线性模型（linear model）。

小结

- softmax运算获取一个向量并将其映射为概率。
- softmax回归适用于分类问题，它使用了softmax运算中输出类别的概率分布。
- 交叉熵是一个衡量两个概率分布之间差异的很好的度量，它测量给定模型编码数据所需的比特数。

- 借助softmax回归，我们可以训练多分类的模型。
- 训练softmax回归循环模型与训练线性回归模型非常相似：先读取数据，再定义模型和损失函数，然后使用优化算法训练模型。大多数常见的深度学习模型都有类似的训练过程。

# 熵

在信息论中，熵用来衡量一个随机事件的不确定性

信息熵 熵表示一个事件所包含的信息量

- 熵越高，则随机变量的信息越多
- 熵越低，则随机变量的信息越少

KL散度是衡量两个概率分布的相似性的一个度量指标

# 逻辑回归

logistic回归用于分类，假设得到的类别为0或者1，使用sigmoid函 数处理输入，这个函数类似于阶跃函数但是又是连续型函数。

$\text{sigmod}(x)$ 其实衡量的是输入数据 $x$ 归属于类别 1 的概率，当 $x < 0$ 的时候，$\text{sigmod}(x) < 0.5$，可以认为 $x$ 归属于类别 0 的概率较大，当 $x > 0$ 的时候，$\text{sigmod}(x) > 0.5$，可以认为 $x$ 归属于类别 1 的概率较大。如果我们将线性加权得到的 $g(x)$ 作为 sigmoid 函数的输入，得到

$$f(x) = \frac{1}{1+e^{-g(x)}} = \sigma(g(x)) = \sigma(w^T x)$$

# 第四章：感知机

激活函数为sign(x)

感知机是人工神经网络的第一个实际应用，标志着神经网络进入 了新的发展阶段。

> Q：人工神经网络的第一个实际应用是什么?
>
> A：感知机

神经网络的感知机其实就是一个线 性方程再套上一个激活（励）函数， 是组成神经网络的基本单元。

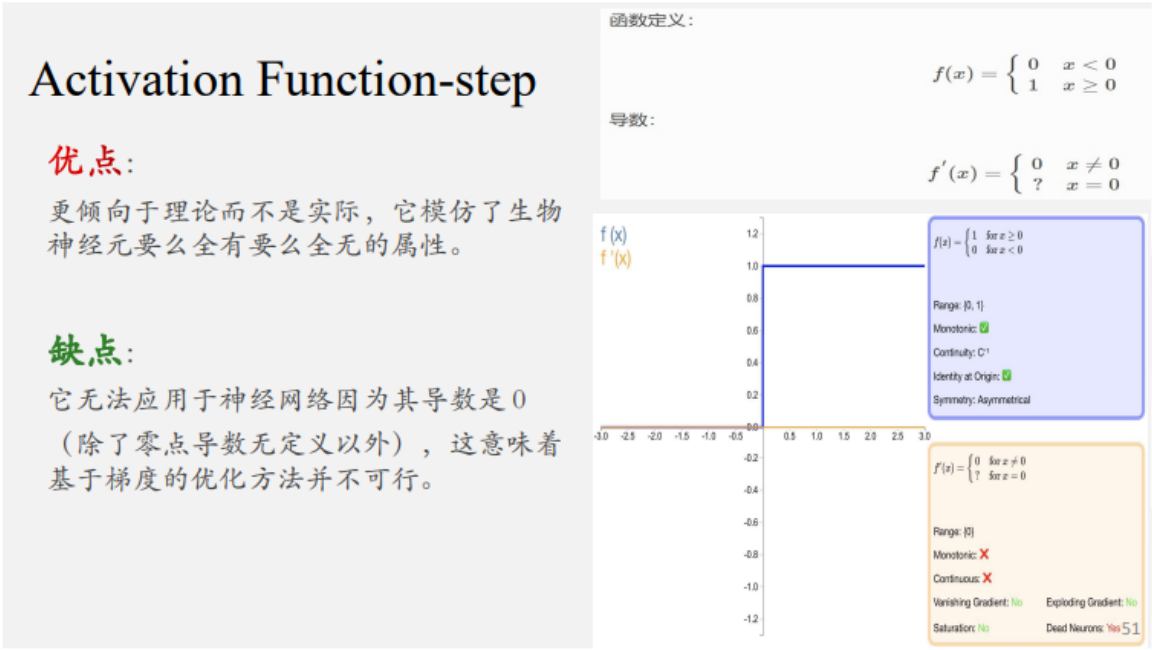
# 讲讲激活函数的作用及特点?

- 激活函数是人工神经网络的一个极其重要的特征
- 激活函数决定一个神经元是否应该被激活，激活代表神经元接收的信息与给定的信息有关
- 激活函数对输入信息进行非线性变换，然后将变换后的输出信息作为输入信息传给下一层神经元。

激活函数的作用

如果不用激活函数，每一层输出都是上层输入的线性函数，无论神经网 络有多少层，最终的输出都是输入的线性组合。 激活函数给神经元引 入了非线性因素，使得神经网络可以任意逼近任何非线性函数。

# 各种激活函数的优缺点：

# Activation Function-Sigmoid-S型

函数定义：

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

导数：

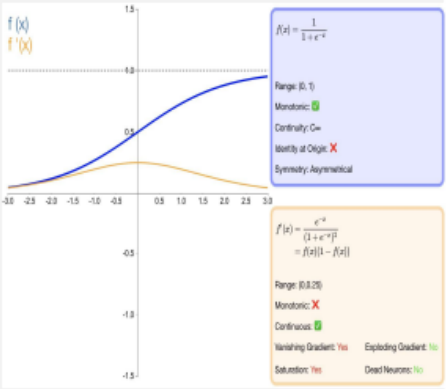
$$f'(x) = f(x)(1-f(x))$$

**优点**：

1. 将很大范围内的输入特征值压缩到0~1之间，使得在深层网络中可以保持数据幅度不会出现较大的变化；
2. 在物理意义上最为接近生物神经元；
3. 根据其输出范围，该函数适用于将预测概率作为输出的模型；

**缺点**：

1. 在深度神经网络中梯度反向传递时导致**梯度爆炸和梯度消失**。
2. **Sigmoid 的 output 不是0均值（即zero-centered）**
3. **其解析式中含有幂运算，计算机求解时相对来讲比较耗时。**






# Activation Function-Tanh

函数定义：

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

导数：

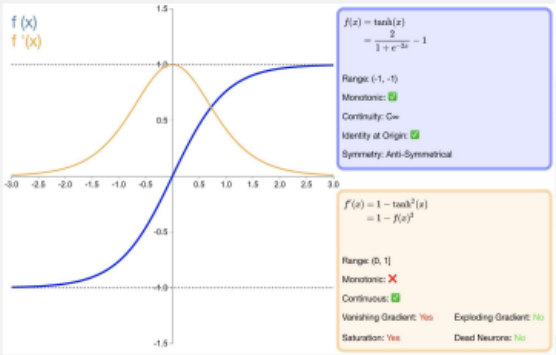
$$f'(x) = 1 - f(x)^2$$

**优点**：

1. 比 sigmoid函数收敛速度更快；
2. 相比 sigmoid函数，tanh是以 0为中心的；

**缺点**：

1. 与 sigmoid函数相同，由于饱和性容易产生的梯度消失；
2. 与 sigmoid函数相同，由于具有幂运算，计算复杂度较高，运算速度较慢。




它解决了Sigmoid函数的不是zero-centered输出问题，然而，梯度消失（gradient vanishing）的问题和幂运算的问题仍然存在。



$$\mathrm{ReLU}(x) = max(x, 0)$$

# Activation Function-Relu

**优点**：

1. 收敛速度远快于sigmoid和tanh；
2. 相较于 sigmoid和 tanh中涉及了幂运算，导致计算复杂度高， ReLU可以更加简单的实现；
3. 当输入 x>=0时，ReLU 的导数为常数，这样可有效缓解梯度消失问题；
4. 当 x<0时，ReLU 的梯度总是 0，提供了神经网络的稀疏表达能力；

**缺点**：

1. ReLU 的输出不是以 0为中心的；
2. Dead ReLU Problem（神经元坏死），某些神经元可能永远不会被激活，导致相应参数永远不会被更新；
3. 不能避免梯度爆炸问题；

ReLU目前仍是最常用的activation function，在搭建人工神经网络的时候推荐优先尝试！

# Activation Function-Leaky Relu

函数定义：

$$f(x)=\begin{cases}\alpha x & x<0\\ x & x\geq 0\end{cases}$$

导数：

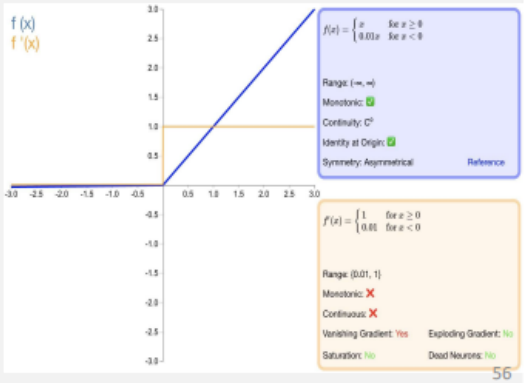
$$f(x)'=\begin{cases}\alpha & x<0\\ 1 & x\geq 0\end{cases}$$

**优点**：

1. Leaky ReLU有ReLU的所有优点，外加不会有Dead ReLU问题；
2. leaky扩大了Relu函数的范围，其中α的值一般设置为一个较小值，如0.01；

**缺点**：

1. 理论上来说，该函数具有比Relu函数更好的效果，但是大量的实践证明，其效果不稳定，故实际中该函数的应用并不多。
2. 由于在不同区间应用的不同的函数所带来的不一致结果，将导致无法为正负输入值提供一致的关系预测。





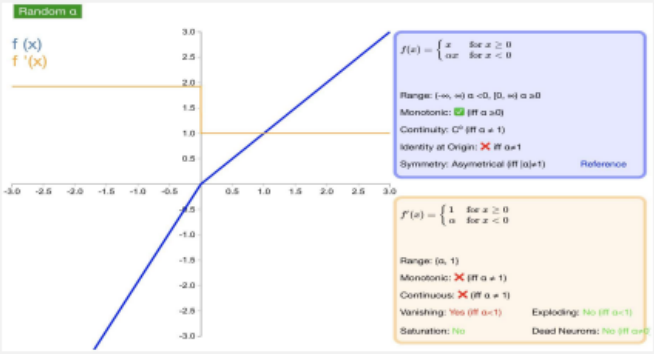

# Activation Function-RRelu

函数定义：

$$f(\alpha,x)=\begin{cases}\alpha & x<0\\ x & x\geq 0\end{cases}$$

导数：

$$f(\alpha,x)'=\begin{cases}\alpha & x<0\\ 1 & x\geq 0\end{cases}$$




**优点**：为负值输入添加了一个线性项，这个线性项的斜率在每一个节点上都是随机分配的（通常服从均匀分布）。



# Activation Function- ELU

Exponential Linear Units

函数定义：

$$f(\alpha,x)=\begin{cases}\alpha\left(e^{x}-1\right) & x<0\\ x & x\geq 0\end{cases}$$

导数：

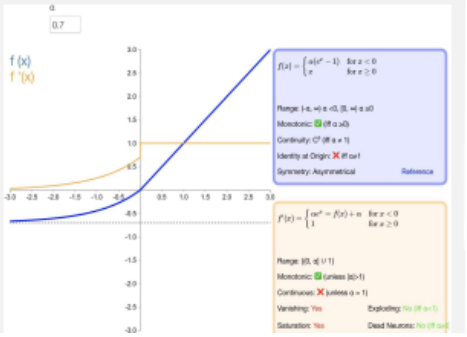
$$f(\alpha,x)'=\begin{cases}f(\alpha,x)+\alpha & x<0\\ 1 & x\geq 0\end{cases}$$

**优点**：

1. 导数收敛为零，从而提高学习效率；
2. 能得到负值输出，这能帮助网络向正确的方向推动权重和偏置变化；
3. 防止死神经元出现。

**缺点**：

1. 计算量大，其表现并不一定比 ReLU 好；
2. 无法避免梯度爆炸问题；



## 选择激活函数的考虑要点

- 尽量选择输出具有zero-centered特点的激活函数以加快模型的 收敛速度
- 浅层网络在分类器时，sigmoid函数及其组合通常效果更好
- 由于梯度消失问题，有时要避免使用 sigmoid和 tanh函数
- relu函数是一个通用的激活函数，目前在大多数情况下使用
- relu函数只能在隐藏层中使用
- 如果使用 ReLU，那么一定要小心设置 learning rate，而且要注 意不要让网络出现很多 “dead” 神经元，可以试试 其他函数。

## 感知机不能学习什么问题？为什么？为了解决这一缺点提出了什么？

感知机无法学习 XOR 问题（神经元只能生成线性分离器）

由于无法模拟诸如异或以及其他复杂函数的功能，使得单层感 知机的应用较为单一。

为了解决线性不可分问题，提出了多层感知机（基本都是两层， 因为BP算法提出之前只能训练一层网络）。

## 多层感知机

相邻层所包含的神经元之间通常使用“全连接”方式进 行连接。所谓“全连接”是指两个相邻层之间的神经元 相互成对连接，但同一层内神经元之间没有连接。

多层感知机可以模拟复杂非线性函数功能，让神经网络 具有更强拟合能力，所模拟函数的复杂性取决于网络隐 藏层数目和各层中神经元数目。

多层感知机在单 层神经网络的基 础上引入了一到 多个隐藏层（hidden layer）。 隐藏层位于输入 层和输出层之间。

多层感知机就是含有至少一个隐藏层的 由全连接层组成的神经网络，且每个隐 藏层的输出通过激活函数进行变换。

## 线性分类模型小结

| 线性模型 | 激活函数 | 损失函数 | 优化方法 |
| --- | --- | --- | --- |
| 线性回归 | - | $(y-\boldsymbol{w}^{\top}\boldsymbol{x})^2$ | 最小二乘、梯度下降 |
| Logistic 回归 | $\sigma(\boldsymbol{w}^{\top}\boldsymbol{x})$ | $\boldsymbol{y}\log\sigma(\boldsymbol{w}^{\top}\boldsymbol{x})$ | 梯度下降 |
| Softmax 回归 | $\text{softmax}(\boldsymbol{W}^{\top}\boldsymbol{x})$ | $\boldsymbol{y}\log\text{softmax}(\boldsymbol{W}^{\top}\boldsymbol{x})$ | 梯度下降 |
| 感知器 | $\text{sgn}(\boldsymbol{w}^{\top}\boldsymbol{x})$ | $\max(0,-y\boldsymbol{w}^{\top}\boldsymbol{x})$ | 随机梯度下降 |
| 支持向量机 | $\text{sgn}(\boldsymbol{w}^{\top}\boldsymbol{x})$ | $\max(0,1-y\boldsymbol{w}^{\top}\boldsymbol{x})$ | 二次规划、SMO 等 |

# 反向传播算法

训练一个神经网络可以分为两个步骤：

- 前向传播，计算损失函数。
- 反向传播，计算梯度，更新参数

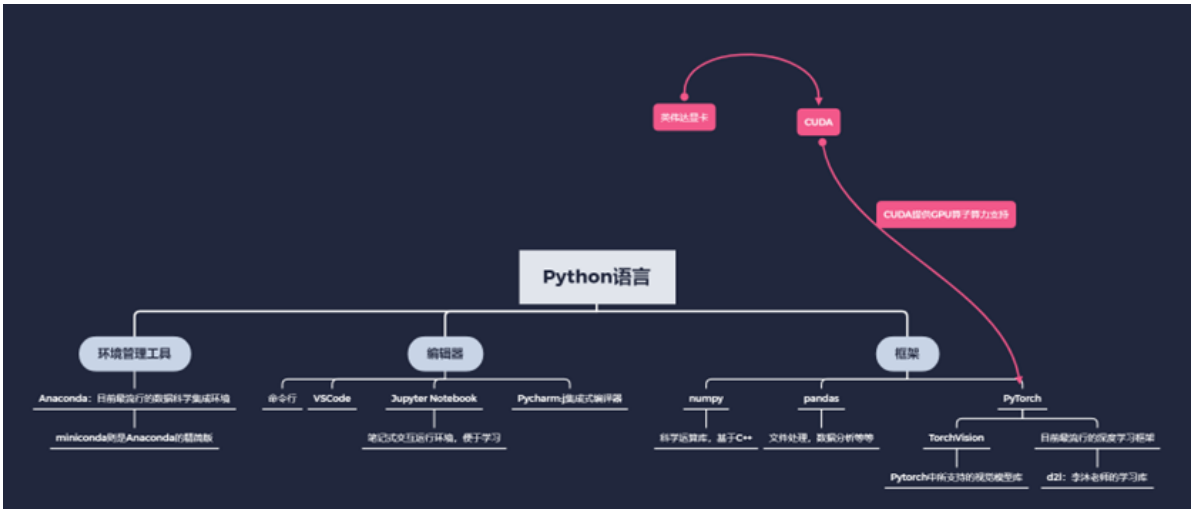
- 梯度下降法（Gradient Descendent）是机器学习的核心算法之一，自动微分则是梯度下降法的核心；
- 梯度下降法用于求损失函数的最优值，梯度下降是通过计算参数与损失函数的梯度并在梯度的方向不断迭代求得极值；
- 机器学习、深度学习中很多求导往往是很复杂的，手动使用链式法则求导很容易出错，借助于计算机也只能是硬编码

# 实验相关

## miniconda、cuda、pycharm、pytorch之间的关系是什么

- miniconda：miniconda是迷你化精简版的anaconda发行版，省去了一些冗余无关的环境包，提供了python虚拟环境的管理、安装、使用等功能

- CUDA：CUDA是NVIDIA英伟达发明的一种并行计算平台和变成模型，通过利用GPU大幅提升了算力和计算性能，只有英伟达显卡支持使用CUDA，而且不同的显卡所能兼容的CUDA版本号不同，在西在安装前需要先确认当前显卡是否支持CUDA
- pycharm：类似于C++可以使用命令行、VS Code、Visual Studio2019等工具进行开发，python语言同样可以使用轻量型编辑器、集成编辑编译环境等等，而Pycharm就是JetBrains发行的一款集成式Python环境，能够很方便的对Python代码进行编写、调试和使用
- d2l：是dive into Deep　Learning的语义缩写，d2l是李沐老师对于《动手学深度学习》中提供的代码使用的公共库，包含了三大类的可使用库：MXnet、PyTorch、TensorFlow
- PyTorch:是FaceBook人工智能研究院（FAIR）于2017年1月基于Torch推出的开源的Python机器学习库，提供了很多高级功能，是目前最流行的深度学习框架
- torchvision：torchvision是pytorch的一个图形库，其服务于Pytorch框架，主要用来构建计算机视觉模型。




# 训练增加迭代周期的数量。为什么测试精度会在一段时间后降低，属于什么现象?

训练增加迭代周期的数量会导致模型在训练集上的误差逐渐减小，但在测试集上的误差可能会先减小后增加，这被称为过拟合现象。

过拟合现象是由于模型在训练集上过于复杂，导致其在测试集上的泛化能力下降。为了解决这个问题，可以采用以下方法：

- 增加数据量：通过增加训练集和测试集的数据量来减少过拟合现象。
- 正则化：通过增加正则化项来限制模型的复杂度，从而减少过拟合现象。
- 早停法：在训练过程中，当测试集上的误差开始上升时，停止训练，从而避免过拟合现象。
- Dropout：在神经网络中加入Dropout层，随机将一部分神经元的输出置为0，从而减少过拟合现象。
- 数据增强：通过对训练集的数据进行旋转、平移、缩放等变换，增加数据的多样性，从而减少过拟合现象。

## 简略解释Relu、leakyRelu和pRelu的关系及优缺点。

Relu（Rectified Linear Unit）是一种神经网络激活函数，主要用于解决梯度消失问题。它的公式很简单：f(x)=max(0,x)，若x大于0，则f(x)=x，否则f(x)=0。

Leaky Relu是对于Relu的改进，主要解决relu函数中的缺点：对于负数部分输出为0，这类数据对计算结果没有影响，但是对于网络的权重更新造成麻烦。因此，Leaky Relu使用一个小的斜率来定义负数部分输出，即：f(x)= max(ax,x)。

PReLU是在Leaky ReLU的基础上进行了改进，它是带参数的ReLU，可以通过反向传播来进行参数优化调整，即： f(x)=max(0,x)+a*min(0,x)。PReLU 的优化参数可以通过神经网络的优化来学习。