神经网络与深度学习笔记（1）

概论

人工神经网络

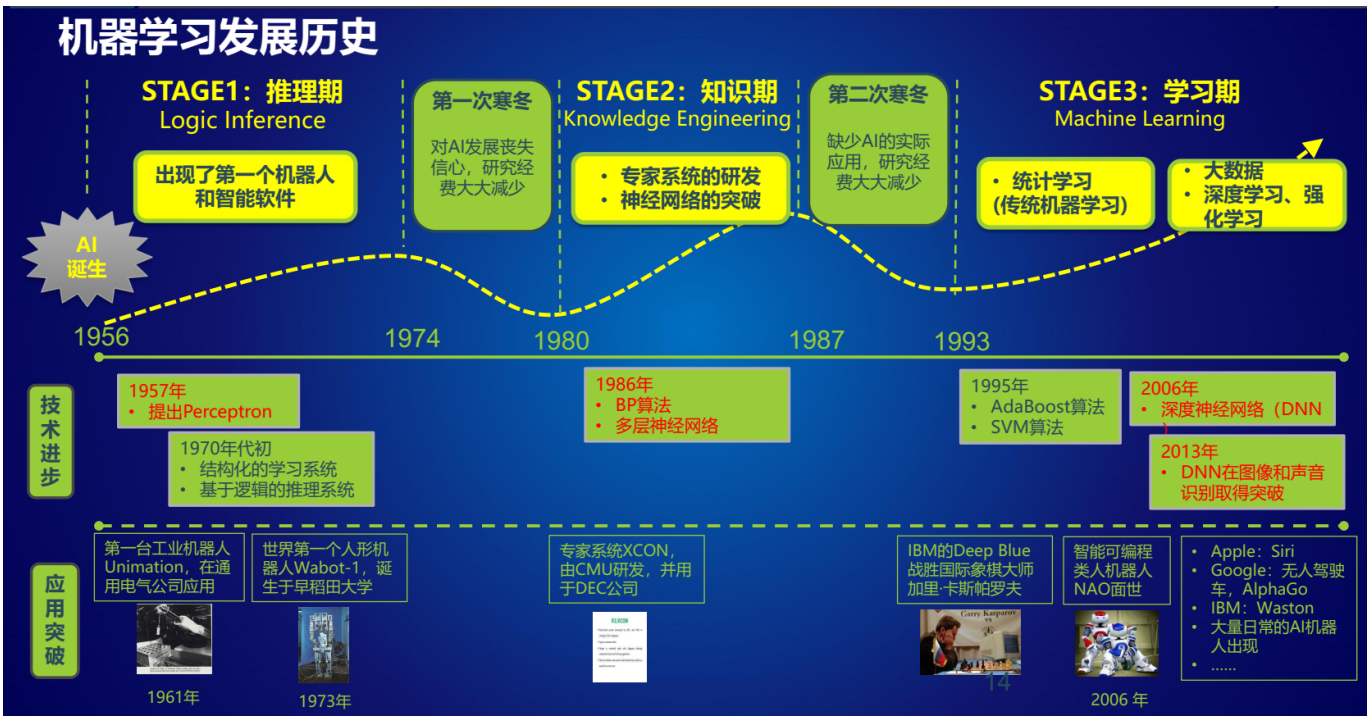
定义：

是从微观结构与功能上模拟人脑神经系统而建立的一类 模型，是模拟人的智能的一条途径。

特点：

网络的信息处理，由神经元间的相互作用实现，具有并行处理的特点； 知识与信息的存储，表现为神经元间分布式的物理联系； 网络的学习和识别，决定于神经元联接权系数的动态演化过程；具有联想记忆特性

机器学习发展：



人工智能应用级别：弱人工智能，强人工智能，超人工智能

人工智能基石：数据，算法，计算

人工智能关键技术：计算机视觉，自然语言处理，机器人，语音识别，机器学习，图像识别

线性分类

1.线性回归：

问题描述：针对样本$(x^i, y^i), i = 1, 2, 3...n$.其中$x$为$m$维数据，$y$为1维数据。
构造代价函数$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y^i - h_\theta(x^i))^2$,其中$h_\theta$（$x^i$）$=$
$\theta^T x$,θ为m维列向量。
目标：寻找超平面参数θ,使$J(\theta)$最小.

求解方法：

1：直接求解，即$\theta = (X^T X)^{-1} X^T y$

2：优化方法$->$梯度下降法

2.线性分类（以二分类为例）

问题描述：针对样本$(x^i, y^i), i = 1, 2, 3...n$.其中$x$为$m$维数据，$y$为0或1。
构造代价函数$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y^i - h_\theta(x^i))^2$,其中$h_\theta$（$x^i$）$= \frac{1}{1+e^{-z}}, z = \theta_1 x_1 +$
$\theta_2 x_2 + ... + \theta_0$。
目标：寻找参数θ,使$J(\theta)$最小.

$求解方法->梯度下降法\$

构造迭代序列$\theta_1, \theta_2, \theta_3, ...$求解：
其中θ_1任取，$\theta_{i+1} = \theta_i + \Delta\theta_i, i = 1, 2, 3, 4...$
$\Delta\theta_i = -\alpha\frac{dJ(\theta_i)}{d\theta_i}$,
$\frac{dJ(\theta_i)}{d\theta_i} = \sum_{i=1}^{n}(y^i - h_\theta(x^i))(-\frac{\partial h_\theta(x^i)}{\partial\theta})$
$\frac{\partial h_\theta(x^i)}{\partial\theta} = h_\theta(1 - h_\theta)x^i$

3.对数回归

思路：转化维条件概率角度求解

代价函数：$J(\theta) = -\sum(y^i log(h_\theta(x^i)) + (1 - y^i)log(1 - h_\theta(x^i)))$

求解方法：梯度下降法

$\Delta_\theta J(\theta) = \sum_i x^i(h_\theta(x^i) - y^i)$

4.线性多分类

代价函数：$J(\theta)=-[\sum_{i=1}^{n}\sum_{k=1}^{k}\log\frac{e^{\theta^T x^i}}{\sum_{j=1}^{k}e^{\theta^T x^i}}]$

求解方法：梯度下降法

$\Delta_{\theta^k}=-\sum_{i=1}^{m}[x^i(1\{y^i=k\}-p(y^i=k|x^i;\theta)]$

感知机

1：神经元模型

1:基本模型

spiking模型：

对于每个时刻的脉冲按加权求和达到阈值输出脉冲信号实例：脉冲神经网络SNN

integrate-and-fire模型：

仿照生物电信号放电过程，神经元在电流刺激下达到阈值电位就发出动作信号，

mp模型：从神经元的模型中可以得出神经元的输出$y=f(\sum_1^n\omega_i x_i-\theta)$,
其中θ为神经元的激活阈值，
函数$f(\cdot)$被是激活函数。函数$f(\cdot)$可用阶跃方程表示，大于阈值激活；否则则抑制
因此通常用$sigmoid$函数来表示函数$f(\cdot)$。

2：作用函数

1：非对称$sigmoid$函数：$f(x)=\frac{1}{1+e^{-\beta x}}$
2：对称$sigmoid$函数：$f(x)=\frac{1}{1+e^{-\beta x}}$
3：对称性阶跃函数$f(x)=\begin{cases}+1, & x\geqslant 0\\ -1, & x<0\end{cases}$

2：单层感知机模型

用途：解决线性分类问题

缺点：难以解决xor问题

模型：$y=f(x)=sign(\omega^T x)$

其中$sign(x)=\begin{cases}+1, & x\geqslant 0\\ -1, & x<0\end{cases}$

算法：

1.赋初值ω_0,数据序号$i=1$，迭代次数$k=0$。
2.选择数据点(x^i,y^i)
3.判断该数据点是否为当前模型的误分类点，即判断若$y^i(\omega^T x^i)\geq 0$,则更新权值：$\omega_{i+1}=\omega_i+\eta y^i x^i$,
4.转到2，直到训练集中没有误分类点

损失函数：$L(\omega)=-\frac{1}{||\omega||}\sum y^i(\omega^T x)$

3：多层感知机模型

多层感知器网络，有如下定理：

> 定理1 若隐层节点（单元）可任意设置，用三层阈值节点的网络，可以实现任意的二值逻辑函数。 定理2 若隐层节点（单元）可任意设置，用三层S型非线性特性节点的网络，可以一致逼近紧集上的连续函数或按 范数逼近紧集上的平方可积函数。

例：

$$y_1^1=f[w_{11}^1u_1+w_{12}^1u_2-\theta_1^1]$$
$$y_2^1=f[w_{21}^1u_1+w_{22}^1u_2-\theta_2^1]$$
$$y=f[w_1^2y_1^1+w_2^2y_2^1-\theta]$$
$$f[\bullet]=\begin{cases}1, & \bullet\geq 0\\ 0, & \bullet<0\end{cases}$$

4：反馈算法

思路：

1：随机初始化多层感知机网络参数

2：针对随机化的网络参数，求出网络输出值

3：针对描述网络输出值与实际值的关系的损失函数计算关于参数的梯度

4：使用求得的梯度对参数更新

5：使用更新的参数求出网络输出值

6：重复3-5步，直至网络达到要求精度

损失函数可增加正则项

优点：

1. 学习完全自主；
2. n可逼近任意非线性函数

缺点：

1. 算法非全局收敛
2. 收敛速度慢
3. 学习速率α选择
4. 神经网络如何设计(几层？节点数？)

例子：对sin(x)进行训练

```
# -*- coding: utf-8 -*-
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
import numpy as np
X=torch.linspace(0, 6*np.pi,10000)
X=torch.unsqueeze(X, dim=1)
Y=orch.sin(X)
class multiplayer(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear1=nn.Linear(1, 100)
        self.linear2=nn.Linear(100, 1)

    def forward(self,X):
        X=self.linear1(X)
        X=torch.sigmoid(X)
        X=self.linear2(X)
        return X
net=multiplayer()
opt=torch.optim.Adam(net.parameters(),lr=0.005)
#opt = torch.optim.SGD(net.parameters(),lr=0.5)
#设置学习率为0.5，用随机梯度下降发优化神经网络的参数
lossfunc = torch.nn.MSELoss()
#设置损失函数为均方损失函数，用来计算每次的误差
losses=[]
for t in range(10000):
#进行100次的优化
    prediction = net(X)
    #得到预测值
    loss=lossfunc(prediction,Y)
    opt.zero_grad()
    #梯度清零
    loss.backward()
    #反向传播
    opt.step()
    #梯度优化
```

性能优化

性能优化算法比较

> 动量法：在梯度下降法中将下降梯度增加惯性项
>
> 自适应梯度法：在梯度下降法中，保持梯度方向不变，按迭代次数依次降低梯度大小以减小震荡
>
> RMSProp算法：在自适应梯度法加入超参数控制梯度大小衰减速率
>
> adam算法：在RMSProp算法中保留历史梯度的指数衰减平均值

1：动量法

算法 8.2 使用动量的随机梯度下降（SGD）

Require: 学习率 ϵ，动量参数 α

Require: 初始参数 $\boldsymbol{\theta}$，初始速度 $\boldsymbol{v}$

while 没有达到停止准则 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度估计：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

计算速度更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

end while

2：自适应梯度法

算法 8.4 AdaGrad 算法

Require: 全局学习率 ϵ

Require: 初始参数 $\boldsymbol{\theta}$

Require: 小常数 δ，为了数值稳定大约设为 10^{-7}

初始化梯度累积变量 $\boldsymbol{r} = 0$

while 没有达到停止准则 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

累积平方梯度：$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

计算更新：$\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$ （逐元素地应用除和求平方根）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

算法 8.5 RMSProp 算法

Require: 全局学习率 ϵ，衰减速率 ρ

Require: 初始参数 $\boldsymbol{\theta}$

Require: 小常数 δ，通常设为 10^{-6}（用于被小数除时的数值稳定）

初始化累积变量 $\boldsymbol{r} = 0$

while 没有达到停止准则 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

累积平方梯度：$\boldsymbol{r} \leftarrow \rho\boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$

计算参数更新：$\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$ （$\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ 逐元素应用）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

算法 8.7 Adam 算法

Require: 步长 ϵ（建议默认为：0.001）

Require: 矩估计的指数衰减速率，ρ_1 和 ρ_2 在区间 $[0, 1)$ 内。（建议默认为：分别为 0.9 和 0.999）

Require: 用于数值稳定的小常数 δ（建议默认为：10^{-8}）

Require: 初始参数 $\boldsymbol{\theta}$

初始化一阶和二阶矩变量 $\boldsymbol{s} = 0, \boldsymbol{r} = 0$

初始化时间步 $t = 0$

while 没有达到停止准则 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

计算梯度：$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

$t \leftarrow t + 1$

更新有偏一阶矩估计：$\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$

更新有偏二阶矩估计：$\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

修正一阶矩的偏差：$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

修正二阶矩的偏差：$\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

计算更新：$\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$（逐元素应用操作）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while