

Linear Regression

Gradient Descent

- 给定Model: $y = w \cdot x + b$
 - Model可以更加复杂，即使用更高阶的参数来描述
 - 这使得Model在training data上表现得更优秀，但在testing data中容易出现Overfitting
- Loss: $L(w, b) = \sum_{i=1}^n (y - (w \cdot x + b))^2$
- 两个参数实现不断更新
 - $w_1 = w_0 - \alpha \frac{\partial L}{\partial w} \Big|_{w=w_0}$
 - $b_1 = b_0 - \alpha \frac{\partial L}{\partial b} \Big|_{b=b_0}$
 - α 表示Learning rate，控制梯度下降的步长

Regularization

- 重新定义Loss function
 - $L(w, b) = \sum_{i=1}^n (y - (w \cdot x + b))^2 + \lambda \sum (w_i)^2$
 - 根据 λ 的值来确定 w 占Loss function的权重
 - λ 越大，正则项的影响越大，使得权重 w 更趋向于 0，减少过拟合，使曲线更平滑
 - λ 越小，正则项影响越小，模型会更关注数据拟合，可能会导致过拟合

Classification

1. 分类的基本概念

- 分类的核心任务是预测数据点所属类别。
- 使用概率模型来计算每个类别的后验概率 $P(y=k | x)$ ，选择概率最高的类别作为最终分类结果。
- 主要依赖贝叶斯公式进行计算：

$$P(y = k | x) = \frac{P(x|y=k)P(y=k)}{P(x)}$$

其中：

- $P(y = k | x)$ ：后验概率，表示给定 x ，属于类别 k 的概率。
- $P(x | y = k)$ ：似然，表示类别 k 生成特征 x 的可能性。
- $P(y = k)$ ：先验概率，表示类别 k 本身的出现概率。
- $P(x)$ ：归一化项，保证概率总和为 1，但分类时可以忽略。

2. 为什么用贝叶斯定理?

- 直接计算 $P(y = k | x)$ 很难, 但 $P(x | y = k)$ 和 $P(y = k)$ 比较容易建模。
- 贝叶斯定理提供了一种从已知概率推导后验概率的方法, 从而进行分类。
- 通过最大化后验概率, 找到最可能的类别。

3. 具体实现步骤

- 计算先验概率 $P(y = k)$: 在训练数据集中统计每个类别出现的频率。
- 计算似然 $P(x | y = k)$:
 - 输入为特征向量 x , 将输入给到训练好的模型, 由模型计算出似然值
- 计算后验概率 $P(y = k | x)$, 然后选取最大值的类别作为分类结果。

4. 训练模型的过程

- 最大似然估计 (MLE): 估计每个类别的均值 μ 和协方差矩阵 Σ 。(前提是假设样本服从高斯分布)
 - 给定 $L(\mu, \Sigma) = f_{\mu, \Sigma}(x^1) f_{\mu, \Sigma}(x^2) f_{\mu, \Sigma}(x^3) f_{\mu, \Sigma}(x^4) \dots f_{\mu, \Sigma}(x^{79})$
 - 找出使得 $L(\mu, \Sigma)$ 最大的 μ, Σ
 - 优化目标: 找到 $\mu_1, \mu_2, \Sigma^1, \Sigma^2$ 使得训练数据的似然函数最大:
- 最终分类: 对于新的数据点 x , 计算 $P(y = 1 | x)$ 和 $P(y = 2 | x)$, 选择概率大的类别。

5. 最终结果

- $P(C_1 | x) = \frac{P(x | C_1)P(C_1)}{P(x | C_1)P(C_1) + P(x | C_2)P(C_2)} = \frac{1}{1 + \frac{P(x | C_2)P(C_2)}{P(x | C_1)P(C_1)}} = \frac{1}{1 + e^{-z}} = \sigma(z)$
 - $\sigma(z)$ 是 sigmoid 函数, 取值范围在 $[0, 1]$, 适用于二分类问题
- 可以把 z 看成 $z = wx + b$
 - 根据 $z = \ln \frac{P(x | C_1)P(C_1)}{P(x | C_2)P(C_2)}$, 经过数学推导得:
 - $w^T = (\mu^1 - \mu^2) \Sigma^{-1}$
 - $b = -\frac{1}{2}(\mu^1)^T (\Sigma^1)^{-1} \mu^1 + \frac{1}{2}(\mu^2)^T (\Sigma^2)^{-1} \mu^2 + \ln \frac{N_1}{N_2}$
 - 其中 $\mu^1, \mu^2, \Sigma^1, \Sigma^2, N_1, N_2$ 分别是训练集一和训练集二的均值, 协方差矩阵和样本数量, 其中 $\Sigma^1 = \Sigma^2 = \Sigma$

Logistic Regression

Loss function

假设数据服从 $f_{w,b}(X) = P_{w,b}(C_1 | x)$, 二分类问题

- 给定训练集

特征向量	x^1	x^2	x^3	...
类别	C_1	C_1	C_2	...
类别	$\hat{y} = 1$	$\hat{y} = 1$	$\hat{y} = 0$...

- 可得似然函数: $f(w, b) = f_{w,b}(x^1)f_{w,b}(x^2)(1 - f_{w,b}(x^3)) \dots f_{w,b}(x^N)$

- 这里的 $f_{w,b}(x)$ 表示的是在给定输入 x 的情况下, 数据属于类别 C_1 的概率, 即:

$$f_{w,b}(x) = P_{w,b}(C_1|x)$$

由于只有两个类别 (假设是 C_1 和 C_2) , 那么属于 C_2 的概率就是:

$$P_{w,b}(C_2|x) = 1 - P_{w,b}(C_1|x) = 1 - f_{w,b}(x)$$

- 先找出最佳 w, b 使得似然函数最大, 而

$$w^*, b^* = \operatorname{argmax} L(w, b) = \operatorname{argmin} (-\ln L(w, b))$$

- $-\ln L(w, b) = \ln f_{w,b}(x^1) + \ln f_{w,b}(x^2) + \ln(1 - f_{w,b}(x^3)) = \sum_n -[\hat{y}^n \ln f_{w,b}(x^n) + (1 - \hat{y}^n) \ln(1 - f_u$

- 可得 **Cross entropy**: $C(f(x^n), \hat{y}^n) = \sum_n -[\hat{y}^n \ln f_{w,b}(x^n) + (1 - \hat{y}^n) \ln(1 - f_{w,b}(x^n))]$ (交叉熵损失函数)

使用Gradient Descent求解最佳参数

对 **Cross entropy**: $C(f(x^n), \hat{y}^n) = \sum_n -[\hat{y}^n \ln f_{w,b}(x^n) + (1 - \hat{y}^n) \ln(1 - f_{w,b}(x^n))]$ 使用 **Gradient Descent**:

- $\frac{\partial \ln(1 - f_{w,b}(x))}{\partial w_i} = \frac{\partial \ln(1 - f_{w,b}(x))}{\partial z} \frac{\partial z}{\partial w_i} = -\sigma(z) x_i$
 - $\frac{\partial \ln(1 - \sigma(z))}{\partial z} = -\frac{1}{1 - \sigma(z)} \sigma(z) (1 - \sigma(z))$
 - $\frac{\partial z}{\partial w_i} = x_i$
- 同理: $\frac{\partial \ln f_{w,b}(x)}{\partial w_i} = \frac{\partial \ln f_{w,b}(x)}{\partial z} \frac{\partial z}{\partial w_i} = (1 - \sigma(z)) x_i$
- 最后: $\frac{\partial (-\ln(w, b))}{\partial w_i} = \sum_n -(\hat{y}^n - f_{w,b}(x^n)) x_i^n$
 - 根据此结果进行参数迭代更新即可

不用Square Error的原因

- 令 **Loss function** 为: $L(f) = \frac{1}{2} \sum_n (f_{w,b}(x^n) - \hat{y}^n)^2$
- $\frac{\partial (f_{w,b}(x) - \hat{y}^n)^2}{\partial w_i} = 2(f_{w,b}(x) - \hat{y}) f_{w,b}(x) (1 - f_{w,b}(x)) x_i$
 - 当 $\hat{y}^n = 0$ 时, 如果预测值 $f_{w,b}(x) = 1$, 此时梯度为0, 会导致参数无法更新, 使得函数无法收敛找到最佳参数
 - 当 $\hat{y}^n = 1$ 时, 如果预测值 $f_{w,b}(x) = 0$, 此时梯度为0, 会导致参数无法更新, 使得函数无法收敛找到最佳参数

Multi-class Classification

对于每个类别, 有自己的weight和bias

- $C_1: w^1, b_1 \quad z_1 = w^1 \cdot x + b_1$
- $C_2: w^2, b_2 \quad z_2 = w^2 \cdot x + b_2$
- $C_3: w^3, b_3 \quad z_3 = w^3 \cdot x + b_3$

1. 使用 `softmax` 函数进行分类

- $f(z) = \frac{e^z}{\sum_{i=1}^n e^i}$
- $f(z)$ 取值范围为[0,1]，且各个样本概率之和为1

2. 使用 `Cross entropy` 进行误差估计，并进行参数更新

- $-\sum_{i=1}^n \hat{y}_i \ln y_i$

Limitation of Logistic Regression

- 逻辑回归最后的**决策边界是一条直线**，意味着不能对某些复杂数据进行准确分类
- 决策边界为 $\sigma(z) = 0.5$ ，即 $w \cdot x + b = 0$

解决办法：使用**Feature transformation**，对数据进行特征变换再进行分类

Discriminative and Generative

Discriminative（判别式模型）

核心思想：直接学习类别之间的决策边界，使得分类误差最小

- 不关心数据具体分布，只关心**给定特征值x**，它属于哪个类别
- 直接对**后验概率** $P(y|x)$ 建模
- 表现通常更好，因为不依赖**数据的分布**，当**数据量大且分布未知**时用判别式模型

Generative（生成式模型）

核心思想：先学习数据概率分布，再推导出分类决策边界

- 先对**先验概率** $P(x|y)$ 建模，再根据**贝叶斯公式**计算后验概率
- 在前面例子中可以很直观看到，**生成式模型是先对数据统计分布（均值和协方差）进行估计**，再计算w和b
- **依赖数据分布**，若真实数据不满足数据分布假设表现可能不好，当**数据量小且分布已知**用生成式模型

Deep Learning

Three steps for Deep Learning

1. Define a set of function

- 需要定义神经网络的架构，用来表示问题的模型
- 需要决定
 - **模型架构**：选择神经网络的类型（如前馈神经网络、卷积神经网络、递归神经网络等）
 - **激活函数**：为每一层选择激活函数（如 ReLU、Sigmoid、Tanh 等）
 - **损失函数**：选择一个损失函数，用来衡量模型预测结果与实际标签之间的误差（如均方误差、交叉熵损失等）
 - **优化函数**：选择优化算法（如随机梯度下降、Adam等），用于在训练过程中最小化损失函数

激活函数

- 为什么不在所有层都使用 Sigmoid，而更偏向使用 ReLU？ReLU 有哪些优缺点？

Sigmoid 的问题：

- 输出范围是 $(0, 1)$ ，这会导致：
 - 梯度太小，尤其当输入很大或很小的时候，梯度几乎为 0（**梯度消失问题**）
- 它不是零中心的，输出总是正的，会导致网络收敛慢

ReLU 的优势：

- 输出是 $[0, \infty)$ ，只要是正数就保留，负数直接变 0
- 计算简单，不会饱和（不会出现 sigmoid 那种梯度快变没的情况）
- 训练更快、更深的网络更稳定

但 ReLU 也有问题：

- **ReLU 死亡现象**：一旦某个神经元输出为 0，在训练中可能永远都不会激活（比如输入是负数时）

2. Goodness of function

- 在评估多个模型之后，选择表现最好的模型。这个阶段包括：
 - **超参数调优**：调整超参数（如学习率、批量大小、网络层数等），找到能使模型表现最佳的配置
 - **交叉验证**：使用交叉验证技术（如k折交叉验证）在不同数据子集上测试模型的性能，减少过拟合的风险
 - **模型比较**：如果有多个模型或架构，使用评估指标对它们进行比较，选择能在新数据上泛化表现最好的模型
 - **最终测试**：一旦选择了最佳模型，使用一个独立的测试集进行最终评估，确认其在真实场景中的表现

3. Pick the best function

- 在评估多个模型之后，选择表现最好的模型。这个阶段包括：
 - **超参数调优**：调整超参数（如学习率、批量大小、网络层数等），找到能使模型表现最佳的配置
 - **交叉验证**：使用交叉验证技术（如k折交叉验证）在不同数据子集上测试模型的性能，减少过拟合的风险
 - **模型比较**：如果有多个模型或架构，使用评估指标对它们进行比较，选择能在新数据上泛化表现最好的模型
 - **最终测试**：一旦选择了最佳模型，使用一个独立的测试集进行最终评估，确认其在真实场景中的表现

Gradient Descent

- Learning Rate需要设置**适宜**
 - 设置太小**训练时间长**
 - 设置太大反而**无法收敛**

梯度下降的本质可以由Taylor Series（泰勒展开式）解释

Adagrad优化器

一个自适应调整学习率的算法

Stochastic Gradient Descent

给定损失函数 $L = \sum_n (\hat{y}^n - (b + \sum (w_i x_i^n)))^2$

- **Gradient Descent:** $\hat{y} = L(\hat{y})$, 选取先前测过的所有样本
- **Stochastic Gradient Descent:** 随机取样本, 只算某一个example的损失函数, 再进行参数更新
 - 因为每次只取一个样本, 计算时间变短, 使得随机梯度下降更新参数的速度快于梯度下降

Feature Scaling

将不同尺度的特征值转换到相似范围, 常见特征缩放方法有: 归一化, 标准化等

- 能够防止某些特征因数值过大主导模型训练
- 加速模型训练, 有利于模型收敛

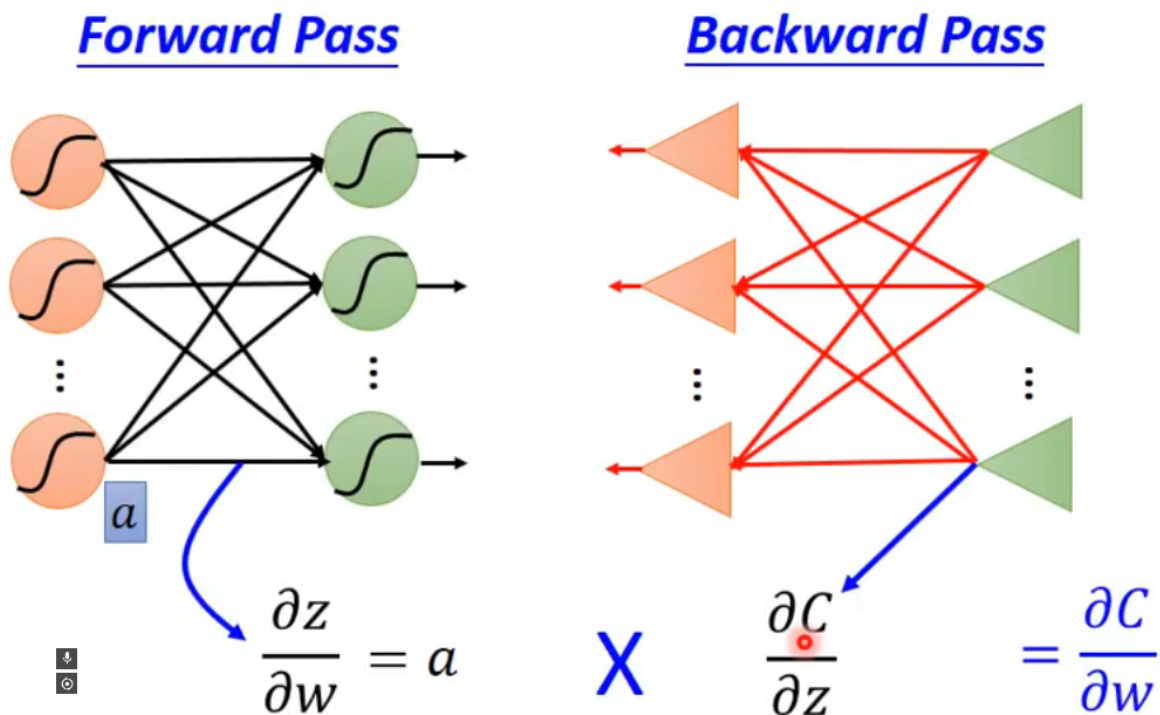
方法:

- $x_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i}$
 - 求每个维度第i个值得均值 m_i 与方差 σ_i
 - 最后得到的数据的均值为0, 方差为1

Backpropagation

- 由Forward pass和Backward pass组成
 - 前向传播会计算网络中每一层输出值和一些特定中间值, 并计算损失
 - 特定中间值包括一些可以直接计算的偏导数等
 - 反向传播会根据前向传播计算的中间值计算张量的梯度

Backpropagation – Summary



Forward pass（前向传播）

Backword pass（反向传播）

BERT

做的是文字填空的工作

Unsupervised Learning

Self-supervised Learning

Word Embedding（词嵌入）

可以说是一种降维技术

降维过程：

1. 从高维到低维的转换：

- 传统的词表示如独热编码(**one-hot encoding**)将每个词表示为词表大小维度的向量(可能是数万甚至数十万维)
- 词嵌入将这些高维稀疏向量转换为低维密集向量(通常仅50-300维)

2. 保留语义信息：

- 尽管维度大幅降低，但词嵌入空间保留了词语之间的语义关系
- 相似词在嵌入空间中距离较近，实现了语义压缩

Predication-based

Prediction-based（预测式）模型 是一种通过预测一个词或上下文中其它词来学习词向量的方法

以根据目标词的前两个词预测目标词为例子

训练过程

1. 输入：目标词的前两个词的单热编码（one-hot encoding）

- 比如预测”我喜欢吃”中的”吃”，输入就是”我”和”喜欢”的单热编码

2. 转换为词向量：单热编码与词嵌入矩阵相乘

- “我”的单热编码 \times 词嵌入矩阵 \rightarrow “我”的词向量
- “喜欢”的单热编码 \times 词嵌入矩阵 \rightarrow “喜欢”的词向量
 - **处理词向量**：对这两个词向量进行处理
 - 可以求平均、拼接或通过其他方式组合
 - 然后通过神经网络层进行进一步处理

4. 预测目标词：输出所有词的概率分布

- 模型预测词表中每个词作为目标词的概率

- 理想情况下，正确目标词的概率应该最高

5. **训练优化**：通过比较预测概率和真实标签（目标词的单热编码）来更新参数

- 使用交叉熵损失函数
- 反向传播更新词嵌入矩阵和其他网络参数

在基于预测的词嵌入训练中，模型会更新多个地方的参数：

1. 词嵌入矩阵（**最重要的**）：
 - 这是模型的核心参数，也是我们最终想要得到的词向量
 - 在训练过程中不断更新，使相似上下文的词获得相似的向量表示
2. 其他网络参数：
 - 如果使用了隐藏层，那么隐藏层的权重和偏置也会更新
 - 输出层的权重矩阵（用于将隐藏层表示转换为预测概率）

Seq2Seq

Sequence to Sequence Model，专门设计用于处理序列数据到序列数据的转换任务

主要组件

Seq2Seq模型的**核心架构**包括两个主要组件：

1. 编码器（Encoder）：接收输入序列，并将其编码为固定长度的上下文向量或表示
2. 解码器（Decoder）：根据编码器产生的表示生成输出序列

AT（Auto-regressive（自回归））VS NAT（Non-Autoregressive Transformer（非自回归））

自回归模型（AR/AT）：

- 顺序生成输出，每次预测一个元素（如一个词或字符）
- 每个预测依赖于先前已生成的所有元素
- 生成质量通常更高，但速度较慢
- 例如：GPT系列、**原始Transformer的解码器部分**

非自回归模型（NAT）：

- 并行生成输出，**一次性预测**所有元素
- 预测之间相互独立，不依赖于模型自己生成的其他输出
- 生成速度快（理论上可以获得显著的推理速度提升）
- 但通常质量较低，可能产生重复或缺失内容